WISDEM

Release 2.0

NREL WISDEM Team

May 19, 2021
## CONTENTS

1 License 3
2 Disclaimer 5
3 Important Links 7
4 Feedback 9

5 Documentation Outline 11
   5.1 WISDEM Installation 11
   5.2 How WISDEM works 12
   5.3 First Steps in WISDEM 14
   5.4 WISDEM capabilities 22
   5.5 WISDEM Inputs 24
   5.6 WISDEM Outputs 55
   5.7 Examples 107
   5.8 Module documentation 209
   5.9 Publications 291
   5.10 Known issues within WISDEM 293
   5.11 How to contribute code to WISDEM 294
   5.12 How to write docs for WISDEM code 295

6 Indices and Tables 299

Bibliography 301

Python Module Index 305

Index 307
The Wind-plant Integrated System Design and Engineering Model (WISDEM) includes integrated assemblies for the assessment of system behavior of wind turbines and plants. These assemblies can be used as is, but a richer use-case involves treating the assemblies as temples, modifying the source code and OpenMDAO problems to answer specific research questions. For example, any variable in these assemblies can be a design variable, an objective, or part of a constraint in a multidisciplinary optimization. WISDEM should therefore be viewed as a toolbox of analysis tools and the basic structure for connecting tools across subsystems and fidelity levels, which can be extended in a multitude of directions according to the user’s needs.
WISDEM is licensed under Apache Version 2.0.
This software is provided as-is and without warranty. There are no guarantees it is bug free or provides the correct answers, even if it is used for the intended purpose. By using this software and as a condition of the Apache license, you agree to not hold any WISDEM developer liable for damages.
CHAPTER
THREE

IMPORTANT LINKS

• Source Code Repository
• OpenMDAO
For software issues please use the Github Issues Tracker. For functionality and theory related questions and comments please use the NWTC forum for Systems Engineering Software Questions.
5.1 WISDEM Installation

Installation with Anaconda is the recommended approach because of the ability to create self-contained environments suitable for testing and analysis. WISDEM requires Anaconda 64-bit.

5.1.1 Configure Anaconda Environment

The installation instructions below use the environment name, “wisdem-env,” but any name is acceptable.

Setup and activate the Anaconda environment from a prompt (Anaconda3 Power Shell on Windows or Terminal.app on Mac):

```bash
conda config --add channels conda-forge
conda create -y --name wisdem-env python=3.7
conda activate wisdem-env
```

Note that any future occasion on which you wish to use WISDEM, you will only have to start with `conda activate wisdem-env`.

5.1.2 Install WISDEM

In order to directly use the examples in the repository and peek at the code when necessary, we recommend all users install WISDEM in developer mode. This is done by first installing WISDEM as a conda package to easily satisfy all dependencies, but then removing the WISDEM conda package and reinstalling from the Github source code. Note the differences between Windows and Mac/Linux build systems.

```bash
conda install -y wisdem git
conda remove --force wisdem
pip install simpy marmot-agents nlopt
```

For Mac/Linux systems:

```bash
conda install compilers
```

For Windows systems:

```bash
conda install m2w64-toolchain libpython
```

Finally, for all systems:
Install `pyOptSparse (Optional)`

`pyOptSparse` is a package that provides additional optimization solvers with OpenMDAO support:

```bash
git clone https://github.com/evan-gaertner/pyoptsparse.git
cd pyoptsparse
python setup.py install
cd ..
```

### 5.1.3 Run Unit Tests

Each package has its own set of unit tests. These can be run in batch with the `test_all.py` script located in the top level `test`-directory:

```bash
cd test
python test_all.py
```

### 5.2 How WISDEM works

#### 5.2.1 Introduction

Full wind plants are comprised of multiple subsystems with varying degrees of technical complexity and many interfaces between many stakeholders. A systems engineering approach can transcend these subsystem boundaries and rigid interfaces to identify lower cost and higher performing designs that could not otherwise be achieved by focusing on individual components. The same approach also enables full system cost-benefit tradeoff and sensitivity studies when evaluating new component or logistical innovations. The Wind-plant Integrated System Design and Engineering Model (WISDEM) is an open-source software package that aims to meet these challenges and empower researcher to meet the following objects:

- Apply multidisciplinary analysis and optimization (MDAO) to engineering and cost models in an open framework to enable full wind turbine and plant system analysis
- Integrate technology or logistic innovations into the turbine and plant design through full system cost-benefit tradeoffs and sensitivity analyses
- Promote collaborative research and analysis among national laboratories, industry, and academia

WISDEM is written in Python using OpenMDAO to manage data flow between analysis blocks and to specify the workflow when performing an analysis or optimization. WISDEM consists of a collection of physics and cost models for different components, at different fidelity levels, that can be combined together to answer system level research questions. All WISDEM models are steady-state and computational efficiency has always represented an important goal during the development of WISDEM to support wide explorations of the solution space.

We’ve also provided an XDSM diagram showing how the WISDEM analysis blocks are connected. This diagram is based on the workflow for a turbine optimization where we vary blade design parameters.
5.2. How WISDEM works
5.3 First Steps in WISDEM

WISDEM just needs three input files for many optimization and analysis operations. These files specify the geometry of the turbine, modeling options, and analysis options. The files and details about them are outlined in the following table:

<table>
<thead>
<tr>
<th>Description</th>
<th>Suggested Default to Modify</th>
<th>Where to learn more</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geometry of the turbine</td>
<td>nrel5mw.yaml</td>
<td>WindIO docs</td>
</tr>
<tr>
<td>Modeling options</td>
<td>modeling_options.yaml</td>
<td>Modeling Options Inputs</td>
</tr>
<tr>
<td>Analysis options</td>
<td>analysis_options.yaml</td>
<td>Analysis Options Inputs</td>
</tr>
</tbody>
</table>

There are two options to run WISDEM with these files. The first option is to use a text editor to modify files and to run WISDEM from the command line. The second option is to edit the files with a GUI and run WISDEM with the click of a button. This document will describe both of these options in turn.

5.3.1 The first step for either option is to make copies of example files

Before you start editing your WISDEM input files, please make copies of the original files in a separate folder. This ensures that if you edit copies of the original files you can always revert back to a version of the files that is known to execute successfully. Each one of the linked files in the table is a good starting point for a turbine simulation.

5.3.2 Option 1: Text editor and command line

First, edit the files in the text editor. You can use the ontology guide as a reference when you create the geometry file. Edit the geometry, modeling options, and analysis options as you need them:
Second, after you are done editing, run WISDEM from the command line with a command of the following form:

```
conda activate wisdem-env
wisdem [geometry file].yaml [modeling file].yaml [analysis file].yaml
```

Substitute `[geometry file].yaml` with the filename for your geometry, `[modeling file].yaml` with your modeling filename and `[analysis file].yaml` with your analysis filename. If you were to run WISDEM with the example files provided above, the command would be `wisdem nrel5mw.yaml modeling_options.yaml analysis_options.yaml`

WISDEM will produce output messages as it runs. At the end, if everything executes correctly, you will see output similar to the following:

- **Objectives**
  - **Turbine AEP:** 24.7350498151 GWh
  - **Blade Mass:** 16403.6823269407 kg
  - **LCOE:** 48.8313243406 USD/MWh
  - **Tip Defl.:** 4.1150829328 m

A command line session to execute WISDEM in this way will look similar to the following figure. This will vary depending on your installation, but the basic elements should be there. Note the final lines that WISDEM outputs, as shown above:

```
(wisdem-env) → wisdem_data wisdem nrel5mw.yaml modeling_options.yaml analysis_options.yaml
WARNING: NO shear modulus, G, was provided for material "resin". The code assumes 2G*(1 + nu) = E, which is only valid for isotropic materials.
Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant to do this, you must specify 'dtype=object' when creating the ndarray.
Blade is stalling at span location 27.59 %
Blade is stalling at span location 31.03 %
Blade is stalling at span location 34.48 %
Blade is stalling at span location 41.38 %
```

Outputs from the run are stored in the `outputs` folder that is created within the folder in which you executed the WISDEM command. These are described in detail in the `outputs` section of this document.

### 5.3.3 Option 2: WISDEM GUI

Launching the GUI is simpler than launching the command line. Activate your environment and execute the WISDEM GUI with the following commands:

```
conda activate wisdem-env
wisdem
```

The WISDEM GUI is laid out from left to right to edit the geometry, modeling, and analysis files. The upper part of the interface shows a status bar and at the right side there is a `Run WISDEM` button.
When you start the WISDEM GUI, a window similar to the following will appear, depending on whether you are running macOS or Windows:

First, load a geometry file. The `nrel5mw.yaml` file is loaded in the following figure. Load this file by clicking the Select geometry YAML button and selecting your copy of `nrel5mw.yaml`.

Similarly, open your copies of the `modeling_options.yaml` as in the following figure:

Finally, open the `analysis_options.yaml` as seen in this figure:

In the GUI, click on the Run WISDEM button. The following dialog box will appear:

When you see this dialog box, the GUI has written the YAML files. WISDEM may take a while to run, so you are asked to confirm that you want to execute the run of WISDEM. Click OK to continue. Once you click OK, the GUI will stop responding while WISDEM executes. Watch the command line window for messages as WISDEM executes. When WISDEM has finished, you will see the following message:

5.3.4 Working with Outputs Manually

In the outputs folder there are several files. Each of them hold all the output variables from a run but are in different formats for various environments:

```
$ ls -1 outputs
refturb_output.mat
refturb_output.npz
refturb_output.pkl
refturb_output.xlsx
refturb_output.yaml
refturb_output-modeling.yaml
refturb_output-analysis.yaml
```
5.3. First Steps in WISDEM
5.3. First Steps in WISDEM
Run WISDEM: Configuration files complete!

Click cancel to back out and continue editing. Click OK to run WISDEM.

WISDEM executed successfully
As an example, the sample_plot.py script plots Axial Induction versus Blade Nondimensional Span by extracting the values from the Python pickle file. The script content is:

```python
import matplotlib.pyplot as plt
from wisdem.glue_code.runWISDEM import load_wisdem

refturb, _, _ = load_wisdem("outputs/refturb_output.pkl")
x = refturb["wt.wt_init.blade.outer_shape_bem.compute_blade_outer_shape_bem.s_default"]
y = refturb["wt.rp.power_curve.compute_power_curve.ax_induct_regII"]
fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(10, 5))
ax.plot(x, y)
ax.set_xlabel("Blade Nondimensional Span [-]")
ax.set_ylabel("Axial Induction [-]"")
plt.show()
```

This script generates the following plot:
5.3.5 Working with Outputs Using compare_designs

WISDEM also comes with a built-in command to help post-process results automatically. Just as there is a wisdem console command installed, there is a compare_designs command that plots and prints WISDEM results given a turbine yaml file. If the turbine yaml file is part of an output series, the values are loaded, otherwise it runs WISDEM using a default set of options. This command can be used within any directory where you have turbine yaml files you want to investigate.

For example, you can use the command to view the results from the previous example simulation. Invoke the command by typing compare_designs outputs/refturb_output.yaml in your terminal. Since this example was just run, it loads in the values, then prints key results to screen and saves plots for variable distributions along the blade span.

You can also compare designs of multiple turbine yaml files using this command, which is especially useful for comparing initial and optimal designs. To do this, simply list as many yaml files as you want to compare after invoking the compare_designs command. Additionally, you can supply your own modeling and analysis options if you want to customize the type of simulation performed when comparing results.

For any further modifications, you can customize the output of the compare_designs command by locally editing the script in the wisdem/postprocessing/compare_designs.py file.

5.4 WISDEM capabilities

Previous sections in the documentation have focused on How WISDEM works at a high-level without delving into WISDEM’s modeling assumptions and coding implementation. This page covers some of those details, explains when to use WISDEM versus other software packages, and where WISDEM’s capabilities start and stop.

5.4.1 What WISDEM can do

WISDEM as a conceptual-level design tool

Because WISDEM models the entire turbine, from wind to LCOE, it is helpful in capturing system-level effects of changes in designs. This makes WISDEM useful at the conceptual design level to help down-select from many potential designs to the most viable designs.

Computational expense

Depending on the model complexity, the disciplines included, and the discretization levels, design problems take on the order of seconds to minutes to solve using WISDEM. For example, tower optimization with fixed loads may take approximately 10 seconds, whereas full system optimization controlling the blade twist takes 5-20 minutes.

Past design problems solved using WISDEM

Since its creation, WISDEM has been used in a variety of problems for many different turbine designs. An overview of these problems is provided below with links to the studies where the results were presented. This is not an exhaustive list and WISDEM’s capabilities have changed over the years, but this table is meant as a quick glance at some of the problems that WISDEM can tackle.
Table 5.1: Previous problems solved using WISDEM

<table>
<thead>
<tr>
<th>Disciplines considered</th>
<th>Problem size</th>
<th>Design variables</th>
<th>Publication link</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aerostructural blades, tower, costs</td>
<td>15 design variables</td>
<td>Chord, twist, spar cap thickness, TSR, rotor diameter, machine rating</td>
<td>10.2514/6.2013-201</td>
</tr>
<tr>
<td>Aerostructural blades, tower, nacelle</td>
<td>35 design variables</td>
<td>Chord, twist, spar cap thickness, precurve, TSR, tower height, tower diameter, wall thickness</td>
<td>10.1002/we.1972</td>
</tr>
<tr>
<td>Tower</td>
<td>6 design variables</td>
<td>Tower diameters and wall thickness</td>
<td>Report</td>
</tr>
<tr>
<td>Generator/drivetrain</td>
<td>7 design variables</td>
<td>Generator design parameters</td>
<td>10.2514/6.2018-1000</td>
</tr>
<tr>
<td>Aerostructural blades, rail transport</td>
<td>~20 design variables</td>
<td>Blade twist, chord, spar cap thickness</td>
<td>Paper</td>
</tr>
</tbody>
</table>

5.4.2 What WISDEM cannot do

WISDEM cannot model time-varying effects

Because WISDEM is developed to tackle preliminary and conceptual design problems, physical phenomena cannot always be captured accurately. Specifically, WISDEM uses steady-state models, which means that time-varying aspects of turbine design cannot be examined using WISDEM. This means that any phenomena related to cyclic, transient, stochastic, or resonance-induced loads should not be studied using WISDEM. That being said, portions of WISDEM can be connected to other software, such as OpenFAST, to take advantage of WISDEM’s computationally efficient systems-approach while also including higher-fidelity models.
WISDEM is not a push-button solution to turbine design

It would be nice to have a tool where you press a button and receive an optimal turbine design. However, design optimization requires expert knowledge to perform correctly and interpret the results. WISDEM exists to minimize time needed to evaluate possible turbine designs by providing a framework and set of models. This enables turbine designers to more quickly assess performance trade-offs and make decisions based on the results from WISDEM.

5.4.3 When to use tools other than WISDEM

WISDEM is useful for a subset of wind turbine design problems. If you need time-varying effects, controllers in the loop, or more complex floating offshore capabilities, WISDEM is not the correct tool. WEIS and OpenFAST are two other software packages that meet different needs.

When to use WEIS

As part of the ARPA-E Atlantis program, NREL is developing WEIS, the Wind Energy with Integrated Servo-control toolset. WEIS enables studies of floating offshore wind turbines using multifidelity design processes. This tool is especially useful for doing optimization of the full floating turbine system and includes integrations for WISDEM, OpenFAST, and other existing NREL codes.

When to use OpenFAST

OpenFAST is a well-established higher-fidelity tool for turbine simulation. Whereas WISDEM and WEIS are design for design optimization, OpenFAST focuses on turbine analysis. OpenFAST gives more physically accurate results but has a much larger computational cost than WISDEM. OpenFAST has been used across multiple decades by hundreds of researchers for a huge number of research projects.

5.5 WISDEM Inputs

5.5.1 Geometry Inputs

Significant effort has been invested to develop an ontology for wind turbines systems analysis by the IEA Wind Task 37 - Systems Engineering in its WindIO project. WISDEM uses this ontology for the physical description of the turbine, and its components, and the required level of fidelity for systems analysis.

Full documentation of the WISDEM geometry input file can be found at the WindIO documentation

5.5.2 Modeling Options Inputs

The following inputs describe the options available in the modeling_options file. This example is from the 02_reference_turbines case in the examples directory.

```yaml
# Generic modeling options file to run standard WISDEM case
General:
  verbosity: False # When set to True, the code prints to screen many infos
WISDEM:
  RotorSE:
    flag: True
    spar_cap_ss: Spar_Cap_SS # Name in the yaml of the spar cap laminate on the
    suction side
```

(continues on next page)
spar_cap_ps: Spar_Cap_PS # Name in the yaml of the spar cap laminate on the suction side

DriveSE:
    # Options of TowerSE module
    flag: True

TowerSE:
    flag: True

BOS:
    flag: True

**General**

*verbosity* [Boolean] Prints additional outputs to screen (and to a file log in the future)

*Default* = False

**RotorSE**

*flag* [Boolean] Whether or not to run RotorSE and ServoSE

*Default* = False

*n_aoa* [Integer] Number of angles of attack in a common grid to define polars

*Default* = 200

*n_xy* [Integer] Number of coordinate point used to define airfoils

*Default* = 200

*n_span* [Integer] Number of spanwise stations in a common grid used to define blade properties

*Default* = 30

*n_pc* [Integer] Number of wind speeds to compute the power curve

*Default* = 20

*n_pc_spline* [Integer] Number of wind speeds to spline the power curve

*Default* = 200

*n_pitch_perf_surfaces* [Integer] Number of pitch angles to determine the Cp-Ct-Cq-surfaces

*Default* = 20

*min_pitch_perf_surfaces* [Float] Min pitch angle of the Cp-Ct-Cq-surfaces

*Default* = -5.0

*max_pitch_perf_surfaces* [Float] Max pitch angle of the Cp-Ct-Cq-surfaces

*Default* = 30.0

*n_tsr_perf_surfaces* [Integer] Number of tsr values to determine the Cp-Ct-Cq-surfaces

*Default* = 20

*min_tsr_perf_surfaces* [Float] Min TSR of the Cp-Ct-Cq-surfaces

*Default* = 2.0
**max_tsr_perf_surfaces** [Float] Max TSR of the Cp-Ct-Cq-surfaces

Default = 12.0

**n_U_perf_surfaces** [Integer] Number of wind speeds to determine the Cp-Ct-Cq-surfaces

Default = 1

**regulation_reg_III** [Boolean] Flag to derive the regulation trajectory in region III in terms of pitch and TSR

Default = False

**spar_cap_ss** [String] Composite layer modeling the spar cap on the suction side in the geometry yaml. This entry is used to compute ultimate strains and it is linked to the design variable spar_cap_ss.

Default = none

**spar_cap_ps** [String] Composite layer modeling the spar cap on the pressure side in the geometry yaml. This entry is used to compute ultimate strains and it is linked to the design variable spar_cap_ps.

Default = none

**te_ss** [String] Composite layer modeling the trailing edge reinforcement on the suction side in the geometry yaml. This entry is used to compute ultimate strains and it is linked to the design variable te_ss.

Default = none

**te_ps** [String] Composite layer modeling the trailing edge reinforcement on the pressure side in the geometry yaml. This entry is used to compute ultimate strains and it is linked to the design variable te_ps.

Default = none

**gamma_freq** [Float] Partial safety factor for modal frequencies

Default = 1.1

Minimum = 1.0 Maximum = 5.0

**gust_std** [Float] Number of standard deviations for strength of gust

Default = 3.0

Minimum = 0.0 Maximum = 5.0

**root_fastener_s_f** [Float] Safety factor for the max stress of blade root fasteners

Default = 2.5

Minimum = 0.1 Maximum = 1.e+2

**DriveSE**

**flag** [Boolean] Whether or not to run RotorSE and ServoSE

Default = False

**model_generator** [Boolean] Whether or not to do detailed generator modeling using tools formerly in GeneratorSE

Default = False

**gamma_f** [Float] Partial safety factor on loads

Default = 1.35

Minimum = 1.0 Maximum = 5.0
gamma_m [Float] Partial safety factor for materials
   Default = 1.3
   Minimum = 1.0 Maximum = 5.0

gamma_n [Float] Partial safety factor for consequence of failure
   Default = 1.0
   Minimum = 1.0 Maximum = 5.0

hub

hub_gamma [Float] Partial safety factor for hub sizing
   Default = 2.0
   Minimum = 1.0 Maximum = 5.0

spinner_gamma [Float] Partial safety factor for spinner sizing
   Default = 1.5
   Minimum = 1.0 Maximum = 5.0

TowerSE

flag [Boolean] Whether or not to run RotorSE and ServoSE
   Default = False

nlC [Integer] Number of load cases
   Default = 1

wind [String from, ['PowerWind', 'LogisticWind']] Wind scaling relationship with height
   Default = PowerWind

gamma_f [Float] Partial safety factor on loads
   Default = 1.35
   Minimum = 1.0 Maximum = 5.0

gamma_m [Float] Partial safety factor for materials
   Default = 1.3
   Minimum = 1.0 Maximum = 5.0

gamma_n [Float] Partial safety factor for consequence of failure
   Default = 1.0
   Minimum = 1.0 Maximum = 5.0

gamma_b [Float] Partial safety factor for buckling
   Default = 1.1
   Minimum = 1.0 Maximum = 5.0
**gamma_freq** [Float] Partial safety factor for modal frequencies

- Default = 1.1
- Minimum = 1.0 Maximum = 5.0

**gamma_fatigue** [Float] Partial safety factor for fatigue failure

- Default = 1.0
- Minimum = 1.0 Maximum = 5.0

**buckling_length** [Float, m] Buckling length factor in Eurocode safety check

- Default = 1.0
- Minimum = 1.0 Maximum = 100.0

**frame3dd**

Set of Frame3DD options used for tower analysis

- **shear** [Boolean] Inclusion of shear area for symmetric sections
  - Default = True

- **geom** [Boolean] Inclusion of shear stiffening through axial loading
  - Default = True

- **nM** [Integer] Number of tower eigenvalue modes to calculate
  - Default = 6
  - Minimum = 0 Maximum = 20

- **tol** [Float] Convergence tolerance for modal eigenvalue solution
  - Default = 1e-09
  - Minimum = 1e-12 Maximum = 0.1

**BOS**

- **flag** [Boolean] Whether or not to run balance of station cost models (LandBOSSE or ORBIT)
  - Default = False

**FloatingSE**

- **flag** [Boolean] Whether or not to run the floating design modules (FloatingSE)
  - Default = False
Loading

This is only used if not running the full WISDEM turbine Group and you need to input the mass properties, forces, and moments for a tower-only or nacelle-only analysis

**mass** [Float, kilogram] Mass at external boundary of the system. For the tower, this would be the RNA mass.

*Default* = 0.0

**center_of_mass** [Array of Floats, meter] Distance from system boundary to center of mass of the applied load. For the tower, this would be the RNA center of mass in tower-top coordinates.

*Default* = [0.0, 0.0, 0.0]

**moment_of_inertia** [Array of Floats, kg*m^2] Moment of inertia of external mass in coordinate system at the system boundary. For the tower, this would be the RNA MoI in tower-top coordinates.

*Default* = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]

loads

**force** [Array of Floats, Newton] Force vector applied at system boundary

*Default* = [0.0, 0.0, 0.0]

**moment** [Array of Floats, N*m] Force vector applied at system boundary

*Default* = [0.0, 0.0, 0.0]

**velocity** [Float, meter] Applied wind reference velocity, if necessary

*Default* = 0.0

5.5.3 Analysis Options Inputs

The following inputs describe the options available in the analysis_options file. This example is from the 03_blade case in the examples directory.

```
general:
    folder_output: outputs_aerostruct
    fname_output: blade_out

design_variables:
    rotor_diameter:
        flag: True
        minimum: 190
        maximum: 240
    blade:
        aero_shape:
            twist:
                flag: True # Flag to optimize the twist
                inverse: False # Flag to determine twist from the user-defined...
                desired margin to stall (defined in constraints)
                n_opt: 4 # Number of control points along blade span
                max_decrease: 0.08722222222222221 # Maximum decrease for the twist in...
                [rad] at the n_opt locations
                max_increase: 0.08722222222222221 # Maximum increase for the twist in...
                [rad] at the n_opt locations
```
index_start: 2  # Lock the first two DVs from blade root
index_end: 4  # All DVs close to blade tip are active

chord:
flag: True  # Flag to optimize the chord
n_opt: 4  # Number of control points along blade span
max_decrease: 0.3  # Minimum multiplicative gain on existing chord,
at the n_opt locations
max_increase: 3.  # Maximum multiplicative gain on existing chord,
at the n_opt locations

index_start: 2  # Lock the first two DVs from blade root
index_end: 4  # The last DV at blade tip is locked

structure:
spar_cap_ss:
flag: True  # Flag to optimize the spar cap thickness on the,
suction side
n_opt: 4  # Number of control points along blade span
max_decrease: 0.7  # Maximum nondimensional decrease at the n_opt,
locations
max_increase: 1.3  # Maximum nondimensional increase at the n_opt,
locations
index_start: 1  # Lock the first DV from blade root
index_end: 3  # The last DV at blade tip is locked

spar_cap_ps:
flag: True  # Flag to optimize the spar cap thickness on the,
pressure side
equal_to_suction: True  # Flag to impose the spar cap thickness on
pressure and suction sides equal
n_opt: 4  # Number of control points along blade span
max_decrease: 0.7  # Maximum nondimensional decrease at the n_opt,
locations
max_increase: 1.3  # Maximum nondimensional increase at the n_opt,
locations
index_start: 1  # Lock the first DV from blade root
index_end: 3  # The last DV at blade tip is locked

merit_figure: LCOE

constraints:
  blade:
    strains_spar_cap_ss:
      flag: True  # Flag to impose constraints on maximum strains (absolute, value) in the spar cap on the blade suction side
      max: 3500.e-6  # Value of maximum strains [-]
      index_start: 1  # Do not enforce constraint at the first station from
blades root of the n_opt from spar_cap_ss
      index_end: 3  # Do not enforce constraint at the last station at blade,
tip of the n_opt from spar_cap_ss
    strains_spar_cap_ps:
      flag: True  # Flag to impose constraints on maximum strains (absolute, value) in the spar cap on the blade pressure side
      max: 3500.e-6  # Value of maximum strains [-]
      index_start: 1  # Do not enforce constraint at the first station from
blade root of the n_opt from spar_cap_ps
      index_end: 3  # Do not enforce constraint at the last station at blade,
index_end: 3  # Do not enforce constraint at the last station at blade tip of the n_opt from spar_cap_ps
tip_deflection:
  flag: True
  margin: 1.4175
stall:
  flag: True  # Constraint on minimum stall margin
  margin: 0.087  # Value of minimum stall margin in [rad]
driver:
  optimization:
    flag: True  # Flag to enable optimization
    tol: 1.e-5  # Optimality tolerance
    # max_major_iter: 10  # Maximum number of major design iterations (SNOPT)
    # max_minor_iter: 100  # Maximum number of minor design iterations (SNOPT)
    max_iter: 2  # Maximum number of iterations (SLSQP)
    solver: SLSQP  # Optimization solver. Other options are 'SLSQP' - 'CONMIN'
    step_size: 1.e-3  # Step size for finite differencing
    form: forward  # Finite differencing mode, either forward or central
recorder:
  flag: False  # Flag to activate OpenMDAO recorder
  file_name: log_opt.sql  # Name of OpenMDAO recorder

general

folder_output [String] Name of folder to dump output files
  Default = output

fname_output [String] File prefix for output files
  Default = output

design_variables

Sets the design variables in a design optimization and analysis

rotor_diameter

Adjust the rotor diameter by changing the blade length (all blade properties constant with respect to non-dimensional span coordinates)

flag [Boolean] Activates as a design variable or constraint
  Default = False

minimum: Float, m
  Default = 0.0
  Minimum = 0.0 Maximum = 1000.0

maximum: Float, m
Default = 0.0

Minimum = 0.0 Maximum = 1000.0

blade

Design variables associated with the wind turbine blades

aero_shape

Design variables associated with the blade aerodynamic shape

twist

Blade twist as a design variable by adding or subtracting radians from the initial value at spline control points along the span.

flag [Boolean] Activates as a design variable or constraint

Default = False

inverse [Boolean] When set to True, the twist is defined inverting the blade-element momentum equations to achieve a desired margin to stall, which is defined among the constraints. flag and inverse cannot be simultaneously be set to True

Default = False

n_opt [Integer] Number of equally-spaced control points of the spline parametrizing the twist distribution along blade span.

Default = 8

Minimum = 4

lower_bound [Array of Floats, rad] Lowest number of radians that can be added (typically negative to explore smaller twist angles)

Default = [-0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1]

upper_bound [Array of Floats, rad] Largest number of radians that can be added (typically positive to explore greater twist angles)

Default = [0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1]

index_start [Integer] Integer setting the first DV of the n_opt along span that is optimized. It is recommended to set index_start to 1 to lock the first DV and prevent the optimizer to try to optimize the twist of the blade root cylinder.

Default = 0

index_end [Integer] Integer setting the last DV of the n_opt along span that is optimized.

Default = 8
chord

Blade chord as a design variable by scaling (multiplying) the initial value at spline control points along the span.

**flag** [Boolean] Activates as a design variable or constraint

*Default* = False

**n_opt** [Integer] Number of equally-spaced control points of the spline parametrizing the chord distribution along blade span.

*Default* = 8

*Minimum* = 4

**max_decrease** [Float] Maximum nondimensional decrease of the blade chord at each optimization location

*Default* = 0.5

**max_increase** [Float] Maximum nondimensional increase of the blade chord at each optimization location

*Default* = 1.5

**index_start** [Integer] Integer setting the first DV of the n_opt along span that is optimized. Setting index_start to 1 or 2 locks the blade root diameter.

*Default* = 0

**index_end** [Integer] Integer setting the last DV of the n_opt along span that is optimized. It is recommended to lock the last point close to blade tip, setting index_end to n_opt minus 1. The last point controls the chord length at blade tip and due to the imperfect tip loss models of CCBlade, it is usually a good idea to taper the chord manually and do not let a numerical optimizer control it.

*Default* = 8

af_positions

Adjust airfoil positions along the blade span.

**flag** [Boolean] Activates as a design variable or constraint

*Default* = False

**af_start** [Integer] Index of airfoil where the optimization can start shifting airfoil position. The airfoil at blade tip is always locked. It is advised to keep the airfoils close to blade root locked.

*Default* = 4

*Minimum* = 1

structure

Design variables associated with the internal blade structure

5.5. WISDEM Inputs
spar_cap_ss

Blade suction-side spar cap thickness as a design variable by scaling (multiplying) the initial value at spline control points along the span.

**flag** [Boolean] Activates as a design variable or constraint

*Default = False*

**n_opt** [Integer] Number of equally-spaced control points of the spline parametrizing the thickness of the spar cap on the suction side.

*Default = 8*
*Minimum = 4*

**max_decrease** [Float] Maximum nondimensional decrease of the spar cap thickness on the suction-side at each optimization location

*Default = 0.5*

**max_increase** [Float] Maximum nondimensional increase of the spar cap thickness on the suction-side at each optimization location

*Default = 1.5*

**index_start** [Integer] Integer setting the first DV of the n_opt along span that is optimized. It is recommended to set index_start to 1 to lock the first DV and impose a pre-defined taper to small thicknesses and mimic a blade manufacturability constraint.

*Default = 0*

**index_end** [Integer] Integer setting the last DV of the n_opt along span that is optimized. It is recommended to lock the last point close to blade tip, setting index_end to n_opt minus 1. This imposes a predefined taper to small thicknesses and mimic a blade manufacturability constraint.

*Default = 8*

spar_cap_ps

Blade pressure-side spar cap thickness as a design variable by scaling (multiplying) the initial value at spline control points along the span.

**flag** [Boolean] Activates as a design variable or constraint

*Default = False*

**n_opt** [Integer] Number of equally-spaced control points of the spline parametrizing the thickness of the spar cap on the pressure side.

*Default = 8*
*Minimum = 4*

**max_decrease** [Float] Maximum nondimensional decrease of the spar cap thickness on the pressure-side at each optimization location

*Default = 0.5*

**max_increase** [Float] Maximum nondimensional increase of the spar cap thickness on the pressure-side at each optimization location

*Default = 1.5*
**index_start** [Integer] Integer setting the first DV of the n_opt along span that is optimized. It is recommended to set index_start to 1 to lock the first DV and impose a pre-defined taper to small thicknesses and mimic a blade manufacturability constraint.

*Default* = 0

**index_end** [Integer] Integer setting the last DV of the n_opt along span that is optimized. It is recommended to lock the last point close to blade tip, setting index_end to n_opt minus 1. This imposes a predefined taper to small thicknesses and mimic a blade manufacturability constraint.

*Default* = 8

**te_ss**

Blade suction-side trailing edge reinforcement thickness as a design variable by scaling (multiplying) the initial value at spline control points along the span.

**flag** [Boolean] Activates as a design variable or constraint

*Default* = False

**n_opt** [Integer] Number of equally-spaced control points of the spline parametrizing the thickness of the trailing edge reinforcement on the suction side. By default, the first point close to blade root and the last point close to blade tip are locked. This is done to impose a pre-defined taper to small thicknesses and mimic a blade manufacturability constraint.

*Default* = 8

*Minimum* = 4

**min_gain** [Float] Lower bound on scalar multiplier that will be applied to value at control points

*Default* = 0.5

**max_gain** [Float] Upper bound on scalar multiplier that will be applied to value at control points

*Default* = 1.5

**te_ps**

Blade pressure-side trailing edge reinforcement thickness as a design variable by scaling (multiplying) the initial value at spline control points along the span.

**flag** [Boolean] Activates as a design variable or constraint

*Default* = False

**n_opt** [Integer] Number of equally-spaced control points of the spline parametrizing the thickness of the trailing edge reinforcement on the pressure side. By default, the first point close to blade root and the last point close to blade tip are locked. This is done to impose a pre-defined taper to small thicknesses and mimic a blade manufacturability constraint.

*Default* = 8

*Minimum* = 4

**min_gain** [Float] Lower bound on scalar multiplier that will be applied to value at control points

*Default* = 0.5
**max_gain** [Float] Upper bound on scalar multiplier that will be applied to value at control points

*Default* = 1.5

**control**

Design variables associated with the control of the wind turbine

**tsr**

Adjust the tip-speed ratio (ratio between blade tip velocity and steady hub-height wind speed)

**flag** [Boolean] Activates as a design variable or constraint

*Default* = False

**minimum** [Float] Minimum allowable value

*Default* = 0.0

*Minimum* = 0.0 *Maximum* = 30.0

**maximum** [Float] Maximum allowable value

*Default* = 0.0

*Minimum* = 0.0 *Maximum* = 30.0

**hub**

Design variables associated with the hub

**cone**

Adjust the blade attachment coning angle (positive values are always away from the tower whether upwind or downwind)

**flag** [Boolean] Activates as a design variable or constraint

*Default* = False

**lower_bound** [Float, rad] Design variable bound

*Default* = 0.0

*Minimum* = 0.0 *Maximum* = 0.5235987756

**upper_bound** [Float, rad] Design variable bound

*Default* = 0.0

*Minimum* = 0.0 *Maximum* = 0.5235987756
**hub_diameter**

Adjust the rotor hub diameter

*flag* [Boolean] Activates as a design variable or constraint  
*Default* = False

*lower_bound* [Float, m] Lowest value allowable for hub diameter  
*Default* = 0.0  
*Minimum* = 0.0 *Maximum* = 30.0

*upper_bound* [Float, m] Highest value allowable for hub diameter  
*Default* = 30.0  
*Minimum* = 0.0 *Maximum* = 30.0

**drivetrain**

Design variables associated with the drivetrain

**uptilt**

Adjust the drive shaft tilt angle (positive values tilt away from the tower whether upwind or downwind)

*flag* [Boolean] Activates as a design variable or constraint  
*Default* = False

*lower_bound* [Float, rad] Design variable bound  
*Default* = 0.0  
*Minimum* = 0.0 *Maximum* = 0.5235987756

*upper_bound* [Float, rad] Design variable bound  
*Default* = 0.0  
*Minimum* = 0.0 *Maximum* = 0.5235987756

**overhang**

Adjust the x-distance, parallel to the ground or still water line, from the tower top center to the rotor apex.

*flag* [Boolean] Activates as a design variable or constraint  
*Default* = False

*lower_bound* [Float, m] Lowest value allowable for design variable  
*Default* = 0.1  
*Minimum* = 0.1 *Maximum* = 30.0

*upper_bound* [Float, m] Highest value allowable for design variable  
*Default* = 0.1  
*Minimum* = 0.1 *Maximum* = 30.0
distance_tt_hub

Adjust the z-dimension height from the tower top to the rotor apex

flag [Boolean] Activates as a design variable or constraint

    Default = False

lower_bound [Float, m] Lowest value allowable for design variable

    Default = 0.1
    Minimum = 0.1 Maximum = 30.0

upper_bound [Float, m] Highest value allowable for design variable

    Default = 0.1
    Minimum = 0.1 Maximum = 30.0

distance_hub_mb

Adjust the distance along the drive shaft from the hub flange to the first main bearing

flag [Boolean] Activates as a design variable or constraint

    Default = False

lower_bound [Float, m] Lowest value allowable for design variable

    Default = 0.1
    Minimum = 0.1 Maximum = 30.0

upper_bound [Float, m] Highest value allowable for design variable

    Default = 0.1
    Minimum = 0.1 Maximum = 30.0

distance_mb_mb

Adjust the distance along the drive shaft from the first to the second main bearing

flag [Boolean] Activates as a design variable or constraint

    Default = False

lower_bound [Float, m] Lowest value allowable for design variable

    Default = 0.1
    Minimum = 0.1 Maximum = 30.0

upper_bound [Float, m] Highest value allowable for design variable

    Default = 0.1
    Minimum = 0.1 Maximum = 30.0
**generator_length**

Adjust the distance along the drive shaft between the generator rotor drive shaft attachment to the stator bedplate attachment

**flag** [Boolean] Activates as a design variable or constraint

*Default* = False

**lower_bound** [Float, m] Lowest value allowable for design variable

*Default* = 0.1

*Minimum* = 0.1 *Maximum* = 30.0

**upper_bound** [Float, m] Highest value allowable for design variable

*Default* = 0.1

*Minimum* = 0.1 *Maximum* = 30.0

---

**gear_ratio**

For geared configurations only, adjust the gear ratio of the gearbox that multiplies the shaft speed and divides the torque

**flag** [Boolean] Activates as a design variable or constraint

*Default* = False

**lower_bound** : Float

*Default* = 1.0

*Minimum* = 1.0 *Maximum* = 500.0

**upper_bound** : Float

*Default* = 150.0

*Minimum* = 1.0 *Maximum* = 1000.0

---

**lss_diameter**

Adjust the diameter at the beginning and end of the low speed shaft (assumes a linear taper)

**flag** [Boolean] Activates as a design variable or constraint

*Default* = False

**lower_bound** [Float, m] Lowest value allowable for design variable

*Default* = 0.1

*Minimum* = 0.1 *Maximum* = 30.0

**upper_bound** [Float, m] Highest value allowable for design variable

*Default* = 0.1

*Minimum* = 0.1 *Maximum* = 30.0
hss_diameter

Adjust the diameter at the beginning and end of the high speed shaft (assumes a linear taper)

flag [Boolean] Activates as a design variable or constraint

Default = False

lower_bound [Float, m] Lowest value allowable for design variable

Default = 0.1
Minimum = 0.1 Maximum = 30.0

upper_bound [Float, m] Highest value allowable for design variable

Default = 0.1
Minimum = 0.1 Maximum = 30.0

nose_diameter

For direct-drive configurations only, adjust the diameter at the beginning and end of the nose/turret (assumes a linear taper)

flag [Boolean] Activates as a design variable or constraint

Default = False

lower_bound [Float, m] Lowest value allowable for design variable

Default = 0.1
Minimum = 0.1 Maximum = 30.0

upper_bound [Float, m] Highest value allowable for design variable

Default = 0.1
Minimum = 0.1 Maximum = 30.0

lss_wall_thickness

Adjust the thickness at the beginning and end of the low speed shaft (assumes a linear taper)

flag [Boolean] Activates as a design variable or constraint

Default = False

lower_bound : Float, m

Default = 0.001
Minimum = 0.001 Maximum = 3.0

upper_bound : Float, m

Default = 1.0
Minimum = 0.01 Maximum = 5.0
**hss_wall_thickness**

Adjust the thickness at the beginning and end of the high speed shaft (assumes a linear taper)

**flag** [Boolean] Activates as a design variable or constraint

- Default = False
- **lower_bound** : Float, m
  - Default = 0.001
  - Minimum = 0.001 Maximum = 3.0
- **upper_bound** : Float, m
  - Default = 1.0
  - Minimum = 0.01 Maximum = 5.0

**nose_wall_thickness**

For direct-drive configurations only, adjust the thickness at the beginning and end of the nose/turret (assumes a linear taper)

**flag** [Boolean] Activates as a design variable or constraint

- Default = False
- **lower_bound** : Float, m
  - Default = 0.001
  - Minimum = 0.001 Maximum = 3.0
- **upper_bound** : Float, m
  - Default = 1.0
  - Minimum = 0.01 Maximum = 5.0

**bedplate_wall_thickness**

For direct-drive configurations only, adjust the wall thickness along the elliptical bedplate

**flag** [Boolean] Activates as a design variable or constraint

- Default = False
- **lower_bound** : Float, m
  - Default = 0.001
  - Minimum = 0.001 Maximum = 3.0
- **upper_bound** : Float, m
  - Default = 1.0
  - Minimum = 0.01 Maximum = 5.0
**bedplate_web_thickness**

For geared configurations only, adjust the I-beam web thickness of the bedplate

flag [Boolean] Activates as a design variable or constraint

Default = False

lower_bound : Float, m

Default = 0.001

Minimum = 0.001 Maximum = 3.0

upper_bound : Float, m

Default = 1.0

Minimum = 0.01 Maximum = 5.0

**bedplate_flange_thickness**

For geared configurations only, adjust the I-beam flange thickness of the bedplate

flag [Boolean] Activates as a design variable or constraint

Default = False

lower_bound : Float, m

Default = 0.001

Minimum = 0.001 Maximum = 3.0

upper_bound : Float, m

Default = 1.0

Minimum = 0.01 Maximum = 5.0

**bedplate_flange_width**

For geared configurations only, adjust the I-beam flange width of the bedplate

flag [Boolean] Activates as a design variable or constraint

Default = False

lower_bound : Float, m

Default = 0.001

Minimum = 0.001 Maximum = 3.0

upper_bound : Float, m

Default = 1.0

Minimum = 0.01 Maximum = 5.0
tower

Design variables associated with the tower or monopile

**outer_diameter**

Adjust the outer diameter of the cylindrical column at nodes along the height. Linear tapering is assumed between the nodes, creating conical frustums in each section

**flag** [Boolean] Activates as a design variable or constraint

*Default* = False

**lower_bound** [Float, m] Design variable bound

*Default* = 5.0

*Minimum* = 0.1 *Maximum* = 100.0

**upper_bound** [Float, m] Design variable bound

*Default* = 5.0

*Minimum* = 0.1 *Maximum* = 100.0

**layer_thickness**

Adjust the layer thickness of each section in the column

**flag** [Boolean] Activates as a design variable or constraint

*Default* = False

**lower_bound** [Float, m] Design variable bound

*Default* = 0.01

*Minimum* = 1e-05 *Maximum* = 1.0

**upper_bound** [Float, m] Design variable bound

*Default* = 0.01

*Minimum* = 1e-05 *Maximum* = 1.0

**section_height**

Adjust the height of each conical section

**flag** [Boolean] Activates as a design variable or constraint

*Default* = False

**lower_bound** [Float, m] Design variable bound

*Default* = 5.0

*Minimum* = 0.1 *Maximum* = 100.0
**upper_bound** [Float, m] Design variable bound

*Default = 5.0*

*Minimum = 0.1 Maximum = 100.0*

**monopile**

Design variables associated with the tower or monopile

**outer_diameter**

Adjust the outer diameter of the cylindrical column at nodes along the height. Linear tapering is assumed between the nodes, creating conical frustums in each section

**flag** [Boolean] Activates as a design variable or constraint

*Default = False*

**lower_bound** [Float, m] Design variable bound

*Default = 5.0*

*Minimum = 0.1 Maximum = 100.0*

**upper_bound** [Float, m] Design variable bound

*Default = 5.0*

*Minimum = 0.1 Maximum = 100.0*

**layer_thickness**

Adjust the layer thickness of each section in the column

**flag** [Boolean] Activates as a design variable or constraint

*Default = False*

**lower_bound** [Float, m] Design variable bound

*Default = 0.01*

*Minimum = 1e-05 Maximum = 1.0*

**upper_bound** [Float, m] Design variable bound

*Default = 0.01*

*Minimum = 1e-05 Maximum = 1.0*
**section_height**

Adjust the height of each conical section

**flag** [Boolean] Activates as a design variable or constraint

*Default* = False

**lower_bound** [Float, m] Design variable bound

*Default* = 5.0

*Minimum* = 0.1 *Maximum* = 100.0

**upper_bound** [Float, m] Design variable bound

*Default* = 5.0

*Minimum* = 0.1 *Maximum* = 100.0

**constraints**

Activate the constraints that are applied to a design optimization

**blade**

Constraints associated with the blade design

**strains_spar_cap_ss**

Enforce a maximum allowable strain in the suction-side spar caps

**flag** [Boolean] Activates as a design variable or constraint

*Default* = False

**max** [Float] Maximum allowable strain value

*Default* = 0.004

*Minimum* = 1e-08 *Maximum* = 0.1

**strains_spar_cap_ps**

Enforce a maximum allowable strain in the pressure-side spar caps

**flag** [Boolean] Activates as a design variable or constraint

*Default* = False

**max** [Float] Maximum allowable strain value

*Default* = 0.004

*Minimum* = 1e-08 *Maximum* = 0.1
**tip_deflection**

Enforce a maximum allowable blade tip deflection towards the tower expressed as a safety factor on the parked margin. Meaning a parked distance to the tower of 30m and a constraint value here of 1.5 would mean that $30/1.5=20$m of deflection is the maximum allowable

*flag* [Boolean] Activates as a design variable or constraint

*Default* = False

*margin*: Float

*Default* = 1.4175

*Minimum* = 1.0 *Maximum* = 10.0

**rail_transport**

Enforce sufficient blade flexibility such that they can be transported on rail cars without exceeding maximum blade strains or derailment. User can activate either 8-axle flatcars or 4-axle

*8_axle* [Boolean] Activates as a design variable or constraint

*Default* = False

*4_axle* [Boolean] Activates as a design variable or constraint

*Default* = False

**stall**

Ensuring blade angles of attacks do not approach the stall point. Margin is expressed in radians from stall.

*flag* [Boolean] Activates as a design variable or constraint

*Default* = False

*margin*: Float, radians

*Default* = 0.05233

*Minimum* = 0.0 *Maximum* = 0.5

**chord**

Enforcing max chord length limit at all points along blade span.

*flag* [Boolean] Activates as a design variable or constraint

*Default* = False

*max*: Float, meter

*Default* = 4.3

*Minimum* = 0.1 *Maximum* = 20.0
**root_circle_diameter**

Enforcing the minimum blade root circle diameter.

*flag* [Boolean] Activates as a design variable or constraint

*Default* = False

**frequency**

Frequency separation constraint between blade fundamental frequency and blade passing (3P) frequency at rated conditions using gamma_freq margin. Can be activated for blade flap and/or edge modes.

*flap_3P* [Boolean] Activates as a design variable or constraint

*Default* = False

*edge_3P* [Boolean] Activates as a design variable or constraint

*Default* = False

**moment_coefficient**

(EXPERIMENTAL) Targeted blade moment coefficient (useful for managing root flap loads or inverse design approaches that is not recommendend for general use)

*flag* [Boolean] Activates as a design variable or constraint

*Default* = False

*min* : Float

*Default* = 0.15

*Minimum* = 0.01 *Maximum* = 5.0

*max* : Float

*Default* = 0.15

*Minimum* = 0.01 *Maximum* = 5.0

**match_cl_cd**

(EXPERIMENTAL) Targeted airfoil cl/cd ratio (useful for inverse design approaches that is not recommendend for general use)

*flag_cl* [Boolean] Activates as a design variable or constraint

*Default* = False

*flag_cd* [Boolean] Activates as a design variable or constraint

*Default* = False

*filename* [String] file path to constraint data

*Default* =
match_L_D

(EXPERIMENTAL) Targeted blade moment coefficient (useful for managing root flap loads or inverse design approaches that is not recommend for general use)

flag_L [Boolean] Activates as a design variable or constraint
   Default = False

flag_D [Boolean] Activates as a design variable or constraint
   Default = False

filename [String] file path to constraint data
   Default =

tower

Constraints associated with the tower design

height_constraint

Double-sided constraint to ensure total tower height meets target hub height when adjusting section heights

flag [Boolean] Activates as a design variable or constraint
   Default = False

lower_bound : Float, m
   Default = 0.01
   Minimum = 1e-06 Maximum = 10.0

upper_bound : Float, m
   Default = 0.01
   Minimum = 1e-06 Maximum = 10.0

stress

Enforce a maximum allowable von Mises stress relative to the material yield stress with safety factor of gamma_f * gamma_m * gamma_n

flag [Boolean] Activates as a design variable or constraint
   Default = False
**global_buckling**

Enforce a global buckling limit using Eurocode checks with safety factor of $\gamma_f \times \gamma_b$

**flag** [Boolean] Activates as a design variable or constraint

*Default = False*

**shell_buckling**

Enforce a shell buckling limit using Eurocode checks with safety factor of $\gamma_f \times \gamma_b$

**flag** [Boolean] Activates as a design variable or constraint

*Default = False*

**slope**

Ensure that the diameter moving up the tower at any node is always equal or less than the diameter of the node preceding it

**flag** [Boolean] Activates as a design variable or constraint

*Default = False*

**d_to_t**

Double-sided constraint to ensure target diameter to thickness ratio for manufacturing and structural objectives

**flag** [Boolean] Activates as a design variable or constraint

*Default = False*

**lower_bound** : Float

*Default = 50.0

*Minimum = 1.0 Maximum = 2000.0*

**upper_bound** : Float

*Default = 50.0

*Minimum = 1.0 Maximum = 2000.0*

**taper**

Enforcing a max allowable conical frustum taper ratio per section

**flag** [Boolean] Activates as a design variable or constraint

*Default = False*

**lower_bound** : Float

*Default = 0.5

*Minimum = 0.001 Maximum = 1.0*
**frequency**

Frequency separation constraint between all tower modal frequencies and blade period (1P) and passing (3P) frequencies at rated conditions using gamma_freq margin.

**flag** [Boolean] Activates as a design variable or constraint

*Default = False*

**frequency_1**

Targeted range for tower first frequency constraint. Since first and second frequencies are generally the same for the tower, this usually governs the second frequency as well (both fore-aft and side-side first frequency)

**flag** [Boolean] Activates as a design variable or constraint

*Default = False*

**lower_bound** : Float, Hz

*Default = 0.1

*Minimum = 0.01 Maximum = 1.0*

**upper_bound** : Float, Hz

*Default = 0.1

*Minimum = 0.01 Maximum = 1.0*

**monopile**

Constraints associated with the monopile design

**pile_depth**

Ensures that the submerged suction pile depth meets a minimum value

**flag** [Boolean] Activates as a design variable or constraint

*Default = False*

**lower_bound** : Float, m

*Default = 0.0

*Minimum = 0.0 Maximum = 200.0*
**stress**

Enforce a maximum allowable von Mises stress relative to the material yield stress with safety factor of \( \gamma_f \cdot \gamma_m \cdot \gamma_n \)

**flag** [Boolean] Activates as a design variable or constraint

*Default* = False

**global_buckling**

Enforce a global buckling limit using Eurocode checks with safety factor of \( \gamma_f \cdot \gamma_b \)

**flag** [Boolean] Activates as a design variable or constraint

*Default* = False

**shell_buckling**

Enforce a shell buckling limit using Eurocode checks with safety factor of \( \gamma_f \cdot \gamma_b \)

**flag** [Boolean] Activates as a design variable or constraint

*Default* = False

**slope**

Ensure that the diameter moving up the tower at any node is always equal or less than the diameter of the node preceding it

**flag** [Boolean] Activates as a design variable or constraint

*Default* = False

**d_to_t**

Double-sided constraint to ensure target diameter to thickness ratio for manufacturing and structural objectives

**flag** [Boolean] Activates as a design variable or constraint

*Default* = False

**lower_bound** : Float

*Default* = 50.0

*Minimum* = 1.0 * Maximum = 2000.0

**upper_bound** : Float

*Default* = 50.0

*Minimum* = 1.0 * Maximum = 2000.0
taper

Enforcing a max allowable conical frustum taper ratio per section

flag [Boolean] Activates as a design variable or constraint

\[\text{Default} = \text{False}\]

lower_bound : Float

\[\text{Default} = 0.5\]
\[\text{Minimum} = 0.001\] \[\text{Maximum} = 1.0\]

frequency

Frequency separation constraint between all tower modal frequencies and blade period (1P) and passing (3P) frequencies at rated conditions using gamma_freq margin.

flag [Boolean] Activates as a design variable or constraint

\[\text{Default} = \text{False}\]

frequency_1

Targeted range for tower first frequency constraint. Since first and second frequencies are generally the same for the tower, this usually governs the second frequency as well (both fore-aft and side-side first frequency)

flag [Boolean] Activates as a design variable or constraint

\[\text{Default} = \text{False}\]

lower_bound : Float, Hz

\[\text{Default} = 0.1\]
\[\text{Minimum} = 0.01\] \[\text{Maximum} = 1.0\]

upper_bound : Float, Hz

\[\text{Default} = 0.1\]
\[\text{Minimum} = 0.01\] \[\text{Maximum} = 1.0\]

hub

hub_diameter

Ensure that the diameter of the hub is sufficient to accommodate the number of blades and blade root diameter

flag [Boolean] Activates as a design variable or constraint

\[\text{Default} = \text{False}\]
**drivetrain**

**lss**

Enforce a maximum allowable von Mises stress relative to the material yield stress with safety factor of $\gamma_f \cdot \gamma_m \cdot \gamma_n$

**flag** [Boolean] Activates as a design variable or constraint

*Default = False*

**hss**

Enforce a maximum allowable von Mises stress relative to the material yield stress with safety factor of $\gamma_f \cdot \gamma_m \cdot \gamma_n$

**flag** [Boolean] Activates as a design variable or constraint

*Default = False*

**bedplate**

Enforce a maximum allowable von Mises stress relative to the material yield stress with safety factor of $\gamma_f \cdot \gamma_m \cdot \gamma_n$

**flag** [Boolean] Activates as a design variable or constraint

*Default = False*

**mb1**

Ensure that the angular deflection at this mean bearing does not exceed the maximum allowable deflection for the bearing type

**flag** [Boolean] Activates as a design variable or constraint

*Default = False*

**mb2**

Ensure that the angular deflection at this mean bearing does not exceed the maximum allowable deflection for the bearing type

**flag** [Boolean] Activates as a design variable or constraint

*Default = False*
length

Ensure that the bedplate length is sufficient to meet desired overhang value

flag [Boolean] Activates as a design variable or constraint

   Default = False

height

Ensure that the bedplate height is sufficient to meet desired nacelle height value

flag [Boolean] Activates as a design variable or constraint

   Default = False

access

For direct-drive configurations only, ensure that the inner diameter of the nose/turret is big enough to allow human access

flag [Boolean] Activates as a design variable or constraint

   Default = False

   lower_bound [Float, meter] Minimum size to ensure human maintenance access

   Default = 2.0

   Minimum = 0.1 Maximum = 5.0

ecc

For direct-drive configurations only, ensure that the elliptical bedplate length is greater than its height

flag [Boolean] Activates as a design variable or constraint

   Default = False

merit_figure [String from, ['LCOE', 'AEP', 'Cp', 'blade_mass', 'tower_mass', 'tower_cost', 'monopile_mass', 'monopile_cost', 'structural_mass', 'structural_cost', 'blade_tip_deflection', 'My_std', 'flp1_std']] Objective function/merit figure for optimization. Choices are LCOE- levelized cost of energy, AEP- turbine annual energy production, Cp- rotor power coefficient, blade_mass, tower_mass, tower_cost, monopile_mass, monopile_cost, structural_mass, tower+monopile mass, structural_cost, tower+monopile cost, blade_tip_deflection, blade tip deflection distance towards tower, My_std- blade flap moment standard deviation, flp1_std- trailing flap standard deviation

   Default = LCOE
**driver**

Specification of the optimization driver (optimization algorithm) parameters

**tol** [Float] Convergence tolerance (relative)

- Default: $1e^{-6}$
- Minimum: $1e^{-12}$, Maximum: 1.0

**max_iter** [Integer] Max number of optimization iterations

- Default: 100
- Minimum: 0, Maximum: 100000

**max_function_calls** [Integer] Max number of calls to objective function evaluation

- Default: 100000
- Minimum: 0, Maximum: 100000000

**solver** [String from, ['SLSQP', 'CONMIN', 'COBYLA', 'SNOPT']] Optimization driver. Can be one of [SLSQP, CONMIN, COBYLA, SNOPT]

- Default: SLSQP

**step_size** [Float] Maximum step size

- Default: 0.001
- Minimum: $1e^{-10}$, Maximum: 100.0

**form** [String from, ['central', 'forward', 'complex']] Finite difference calculation mode

- Default: central

**recorder**

Optimization iteration recording via OpenMDAO

**flag** [Boolean] Activates as a design variable or constraint

- Default: False

**file_name** [String] OpenMDAO recorder output SQL database file

- Default: log_opt.sql

### 5.6 WISDEM Outputs

This table may be downloaded as a csv-file [here](#)

<table>
<thead>
<tr>
<th>Variable</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>airfoils.Re</td>
<td></td>
<td>1D array of the Reynolds numbers used to define the polars of the airfoils. All airfoils defined in openmdao share this grid.</td>
</tr>
<tr>
<td>airfoils.ac</td>
<td></td>
<td>1D array of the aerodynamic centers of each airfoil.</td>
</tr>
<tr>
<td>airfoils.aoa</td>
<td>rad</td>
<td>1D array of the angles of attack used to define the polars of the airfoils. All airfoils defined in openmdao share this grid.</td>
</tr>
</tbody>
</table>

continues on next page
### Table 5.2 – continued from previous page

<table>
<thead>
<tr>
<th>Variable</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>airfoils.cd</td>
<td></td>
<td>4D array with the drag coefficients of the airfoils. Dimension 0 is along the different airfoils defined in the yaml, dimension 1 is along the angles of attack, dimension 2 is along the Reynolds number, dimension 3 is along the number of tabs, which may describe multiple sets at the same station, for example in presence of a flap.</td>
</tr>
<tr>
<td>airfoils.cl</td>
<td></td>
<td>4D array with the lift coefficients of the airfoils. Dimension 0 is along the different airfoils defined in the yaml, dimension 1 is along the angles of attack, dimension 2 is along the Reynolds number, dimension 3 is along the number of tabs, which may describe multiple sets at the same station, for example in presence of a flap.</td>
</tr>
<tr>
<td>airfoils.cm</td>
<td></td>
<td>4D array with the moment coefficients of the airfoils. Dimension 0 is along the different airfoils defined in the yaml, dimension 1 is along the angles of attack, dimension 2 is along the Reynolds number, dimension 3 is along the number of tabs, which may describe multiple sets at the same station, for example in presence of a flap.</td>
</tr>
<tr>
<td>airfoils.coord_xy</td>
<td></td>
<td>3D array of the x and y airfoil coordinates of the n_af airfoils.</td>
</tr>
<tr>
<td>airfoils.name</td>
<td>Unavailable</td>
<td>1D array of names of airfoils.</td>
</tr>
<tr>
<td>airfoils.r_thick</td>
<td></td>
<td>1D array of the relative thicknesses of each airfoil.</td>
</tr>
<tr>
<td>assembly.blade_length</td>
<td>m</td>
<td>Scalar of the 3D blade length computed along its axis, scaled based on the user defined rotor diameter.</td>
</tr>
<tr>
<td>assembly.blade_ref_axis</td>
<td>m</td>
<td>2D array of the coordinates (x,y,z) of the blade reference axis scaled based on rotor diameter, defined along blade span. The coordinate system is the one of BeamDyn: it is placed at blade root with x pointing the suction side of the blade, y pointing the trailing edge and z along the blade span. A standard configuration will have negative x values (prebend), if swept positive y values, and positive z values.</td>
</tr>
<tr>
<td>assembly.blade_ref_axis_user</td>
<td>m</td>
<td>2D array of the coordinates (x,y,z) of the blade reference axis, defined along blade span. The coordinate system is the one of BeamDyn: it is placed at blade root with x pointing the suction side of the blade, y pointing the trailing edge and z along the blade span. A standard configuration will have negative x values (prebend), if swept positive y values, and positive z values.</td>
</tr>
<tr>
<td>assembly.distance_tt_hub</td>
<td>m</td>
<td>Vertical distance from tower top to hub center.</td>
</tr>
<tr>
<td>assembly.hub_height</td>
<td>m</td>
<td>Height of the hub in the global reference system, i.e. distance rotor center to ground._hub height of wind turbine above ground / sea level</td>
</tr>
<tr>
<td>assembly.hub_height_user</td>
<td>m</td>
<td>Height of the hub specified by the user.</td>
</tr>
<tr>
<td>assembly.hub_radius</td>
<td>m</td>
<td>Radius of the hub. It defines the distance of the blade root from the rotor center along the coned line.</td>
</tr>
<tr>
<td>assembly.r_blade</td>
<td>m</td>
<td>1D array of the dimensional spanwise grid defined along the rotor (hub radius to blade tip projected on the plane)</td>
</tr>
<tr>
<td>assembly.rotor_diameter</td>
<td>m</td>
<td>Diameter of the rotor used in WISDEM. It is defined as two times the blade length plus the hub diameter.rotor diameter</td>
</tr>
<tr>
<td>assembly.rotor_diameter_user</td>
<td>m</td>
<td>Diameter of the rotor specified by the user. It is defined as two times the blade length plus the hub diameter.</td>
</tr>
<tr>
<td>assembly.rotor_radius</td>
<td>m</td>
<td>Scalar of the rotor radius, defined ignoring prebend and sweep curvatures, and cone and uptilt angles.</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Variable</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>assembly.tower_ref_axis</td>
<td>m</td>
<td>2D array of the coordinates (x,y,z) of the tower reference axis. The coordinate system is the global coordinate system of OpenFAST: it is placed at tower base with x pointing downwind, y pointing on the side and z pointing vertically upwards. A standard tower configuration will have zero x and y values and positive z values.</td>
</tr>
<tr>
<td>assembly.tower_ref_axis_user</td>
<td>m</td>
<td>2D array of the coordinates (x,y,z) of the tower reference axis. The coordinate system is the global coordinate system of OpenFAST: it is placed at tower base with x pointing downwind, y pointing on the side and z pointing vertically upwards. A standard tower configuration will have zero x and y values and positive z values.</td>
</tr>
<tr>
<td>blade.internal_structure_2d_fem.chord</td>
<td>m</td>
<td>1D array of the chord values defined along blade span. Chord length at each section.</td>
</tr>
<tr>
<td>blade.internal_structure_2d_fem.coord_xy_dim</td>
<td>m</td>
<td>3D array of the dimensional x and y airfoil coordinates of the airfoils interpolated along span for n_span stations. The origin is placed at the pitch axis.</td>
</tr>
<tr>
<td>blade.internal_structure_2d_fem.definition_layer</td>
<td></td>
<td>1D array of flags identifying how layers are specified in the yaml. 1) all around (skin, paint, ...) 2) offset+rotation twist+width (spar caps) 3) offset+user defined rotation+width 4) midpoint TE+width (TE reinf) 5) midpoint LE+width (LE reinf) 6) layer position fixed to other layer (core fillers) 7) start and width 8) end and width 9) start and end 10) web layer</td>
</tr>
<tr>
<td>blade.internal_structure_2d_fem.definition_web</td>
<td></td>
<td>1D array of flags identifying how webs are specified in the yaml. 1) offset+rotation=twist 2) offset+rotation</td>
</tr>
<tr>
<td>blade.internal_structure_2d_fem.index_layer_end</td>
<td></td>
<td>Index used to fix a layer to another</td>
</tr>
<tr>
<td>blade.internal_structure_2d_fem.index_layer_start</td>
<td></td>
<td>Index used to fix a layer to another</td>
</tr>
<tr>
<td>blade.internal_structure_2d_fem.layer_end_nd</td>
<td></td>
<td>2D array of the non-dimensional end point defined along the outer profile of a layer. The TE suction side is 0, the TE pressure side is 1. The first dimension represents each layer, the second dimension represents each entry along blade span.</td>
</tr>
<tr>
<td>blade.internal_structure_2d_fem.layer_midpoint_nd</td>
<td></td>
<td>2D array of the non-dimensional midpoint defined along the outer profile of a layer. The first dimension represents each layer, the second dimension represents each entry along blade span.</td>
</tr>
<tr>
<td>blade.internal_structure_2d_fem.layer_offset_y_pa</td>
<td>m</td>
<td>2D array of the offset along the y axis to set the position of a layer. Positive values move the layer towards the trailing edge, negative values towards the leading edge. The first dimension represents each layer, the second dimension represents each entry along blade span.</td>
</tr>
<tr>
<td>blade.internal_structure_2d_fem.layer_offset_y_pa_yaml</td>
<td>m</td>
<td>2D array of the offset along the y axis to set the position of a layer. Positive values move the layer towards the trailing edge, negative values towards the leading edge. The first dimension represents each layer, the second dimension represents each entry along blade span.</td>
</tr>
</tbody>
</table>
### Table 5.2 – continued from previous page

<table>
<thead>
<tr>
<th>Variable</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>blade.internal_structure_2d_fem.layer_rotation</td>
<td>rad</td>
<td>2D array of the rotation angle of a layer with respect to the chord line. The first dimension represents each layer, the second dimension represents each entry along blade span. If the rotation is equal to negative twist $-\alpha$ a constant, then the layer is built straight.</td>
</tr>
<tr>
<td>blade.internal_structure_2d_fem.layer_rotation_yaml</td>
<td>rad</td>
<td>2D array of the rotation angle of a layer with respect to the chord line. The first dimension represents each layer, the second dimension represents each entry along blade span. If the rotation is equal to negative twist $-\alpha$ a constant, then the layer is built straight.</td>
</tr>
<tr>
<td>blade.internal_structure_2d_fem.layer_side</td>
<td>unavailable</td>
<td>1D array setting whether the layer is on the suction or pressure side. This entry is only used if definition_layer is equal to 1 or 2.</td>
</tr>
<tr>
<td>blade.internal_structure_2d_fem.layer_start_nd</td>
<td>m</td>
<td>2D array of the non-dimensional start point defined along the outer profile of a layer. The TE suction side is 0, the TE pressure side is 1. The first dimension represents each layer, the second dimension represents each entry along blade span.</td>
</tr>
<tr>
<td>blade.internal_structure_2d_fem.layer_start_nd_yaml</td>
<td>m</td>
<td>2D array of the non-dimensional start point defined along the outer profile of a layer. The TE suction side is 0, the TE pressure side is 1. The first dimension represents each layer, the second dimension represents each entry along blade span.</td>
</tr>
<tr>
<td>blade.internal_structure_2d_fem.layer_thickness</td>
<td>m</td>
<td>2D array of the thickness of the layers of the blade structure. The first dimension represents each layer, the second dimension represents each piecewise-constant entry of the column sections.</td>
</tr>
<tr>
<td>blade.internal_structure_2d_fem.layer_web</td>
<td>m</td>
<td>1D array of the web id the layer is associated to. If the layer is on the outer profile, this entry can simply stay equal to zero.</td>
</tr>
<tr>
<td>blade.internal_structure_2d_fem.layer_width</td>
<td>m</td>
<td>2D array of the width along the outer profile of a layer. The first dimension represents each layer, the second dimension represents each entry along blade span.</td>
</tr>
<tr>
<td>blade.internal_structure_2d_fem.layer_width_yaml</td>
<td>m</td>
<td>2D array of the width along the outer profile of a layer. The first dimension represents each layer, the second dimension represents each entry along blade span.</td>
</tr>
<tr>
<td>blade.internal_structure_2d_fem.pitch_axis</td>
<td></td>
<td>1D array of the chordwise position of the pitch axis (0-LE, 1-TE), defined along blade span.</td>
</tr>
<tr>
<td>blade.internal_structure_2d_fem.s</td>
<td>m</td>
<td>1D array of the non-dimensional spanwise grid defined along blade axis (0-blade root, 1-blade tip)</td>
</tr>
<tr>
<td>blade.internal_structure_2d_fem.twist</td>
<td>rad</td>
<td>1D array of the twist values defined along blade span. The twist is defined positive for negative rotations around the z axis (the same as in BeamDyn).</td>
</tr>
<tr>
<td>blade.internal_structure_2d_fem.web_end_nd</td>
<td>m</td>
<td>2D array of the non-dimensional end point defined along the outer profile of a web. The TE suction side is 0, the TE pressure side is 1. The first dimension represents each web, the second dimension represents each entry along blade span.</td>
</tr>
<tr>
<td>blade.internal_structure_2d_fem.web_end_nd_yaml</td>
<td>m</td>
<td>2D array of the non-dimensional end point defined along the outer profile of a web. The TE suction side is 0, the TE pressure side is 1. The first dimension represents each web, the second dimension represents each entry along blade span.</td>
</tr>
</tbody>
</table>

*continues on next page*
<table>
<thead>
<tr>
<th>Variable</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>blade.internal_structure_2d_fem.web_offset_y_pa</td>
<td>m</td>
<td>2D array of the offset along the y axis to set the position of the shear webs. Positive values move the web towards the trailing edge, negative values towards the leading edge. The first dimension represents each shear web, the second dimension represents each entry along blade span.</td>
</tr>
<tr>
<td>blade.internal_structure_2d_fem.web_offset_y_pa_yaml</td>
<td>m</td>
<td>2D array of the offset along the y axis to set the position of the shear webs. Positive values move the web towards the trailing edge, negative values towards the leading edge. The first dimension represents each shear web, the second dimension represents each entry along blade span.</td>
</tr>
<tr>
<td>blade.internal_structure_2d_fem.web_rotation</td>
<td>rad</td>
<td>2D array of the rotation angle of the shear webs in respect to the chord line. The first dimension represents each shear web, the second dimension represents each entry along blade span. If the rotation is equal to negative twist + a constant, then the web is built straight.</td>
</tr>
<tr>
<td>blade.internal_structure_2d_fem.web_rotation_yaml</td>
<td>rad</td>
<td>2D array of the rotation angle of the shear webs in respect to the chord line. The first dimension represents each shear web, the second dimension represents each entry along blade span. If the rotation is equal to negative twist + a constant, then the web is built straight.</td>
</tr>
<tr>
<td>blade.internal_structure_2d_fem.web_start_nd</td>
<td>2D array of the non-dimensional start point defined along the outer profile of a web. The TE suction side is 0, the TE pressure side is 1. The first dimension represents each web, the second dimension represents each entry along blade span.</td>
<td></td>
</tr>
<tr>
<td>blade.internal_structure_2d_fem.web_start_nd_yaml</td>
<td>2D array of the non-dimensional start point defined along the outer profile of a web. The TE suction side is 0, the TE pressure side is 1. The first dimension represents each web, the second dimension represents each entry along blade span.</td>
<td></td>
</tr>
<tr>
<td>blade.interp_airfoils.ac</td>
<td>1D array of the aerodynamic centers of each airfoil.</td>
<td></td>
</tr>
<tr>
<td>blade.interp_airfoils.ac_interp</td>
<td>1D array of the aerodynamic center of the blade defined along span.</td>
<td></td>
</tr>
<tr>
<td>blade.interp_airfoils.af_position</td>
<td>1D array of the non dimensional positions of the airfoils af_used defined along blade span.</td>
<td></td>
</tr>
<tr>
<td>blade.interp_airfoils.aoa</td>
<td>rad</td>
<td>1D array of the angles of attack used to define the polars of the airfoils. All airfoils defined in openmdao share this grid.</td>
</tr>
<tr>
<td>blade.interp_airfoils.cd</td>
<td>4D array with the drag coefficients of the airfoils. Dimension 0 is along the different airfoils defined in the yaml, dimension 1 is along the angles of attack, dimension 2 is along the Reynolds number, dimension 3 is along the number of tabs, which may describe multiple sets at the same station, for example in presence of a flap.</td>
<td></td>
</tr>
<tr>
<td>blade.interp_airfoils.cd_interp</td>
<td>4D array with the drag coefficients of the airfoils. Dimension 0 is along the blade span for n_span stations, dimension 1 is along the angles of attack, dimension 2 is along the Reynolds number, dimension 3 is along the number of tabs, which may describe multiple sets at the same station, for example in presence of a flap.</td>
<td></td>
</tr>
<tr>
<td>blade.interp_airfoils.chord</td>
<td>m</td>
<td>1D array of the chord values defined along blade span. Chord length at each section.</td>
</tr>
<tr>
<td>blade.interp_airfoils.cl</td>
<td>4D array with the lift coefficients of the airfoils. Dimension 0 is along the different airfoils defined in the yaml, dimension 1 is along the angles of attack, dimension 2 is along the Reynolds number, dimension 3 is along the number of tabs, which may describe multiple sets at the same station, for example in presence of a flap.</td>
<td></td>
</tr>
<tr>
<td>Variable</td>
<td>Units</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>-------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>blade.interp_airfoils.cl_interp</td>
<td></td>
<td>4D array with the lift coefficients of the airfoils. Dimension 0 is along the blade span for n_span stations, dimension 1 is along the angles of attack, dimension 2 is along the Reynolds number, dimension 3 is along the number of tabs, which may describe multiple sets at the same station, for example in presence of a flap.</td>
</tr>
<tr>
<td>blade.interp_airfoils.cm</td>
<td></td>
<td>4D array with the moment coefficients of the airfoils. Dimension 0 is along the different airfoils defined in the yaml, dimension 1 is along the angles of attack, dimension 2 is along the Reynolds number, dimension 3 is along the number of tabs, which may describe multiple sets at the same station, for example in presence of a flap.</td>
</tr>
<tr>
<td>blade.interp_airfoils.cm_interp</td>
<td></td>
<td>4D array with the moment coefficients of the airfoils. Dimension 0 is along the blade span for n_span stations, dimension 1 is along the angles of attack, dimension 2 is along the Reynolds number, dimension 3 is along the number of tabs, which may describe multiple sets at the same station, for example in presence of a flap.</td>
</tr>
<tr>
<td>blade.interp_airfoils.coord_xy</td>
<td></td>
<td>3D array of the x and y airfoil coordinates of the n_af airfoils.</td>
</tr>
<tr>
<td>blade.interp_airfoils.coord_xy_dim</td>
<td></td>
<td>3D array of the dimensional x and y airfoil coordinates of the airfoils interpolated along span for n_span stations. The origin is placed at the pitch axis.</td>
</tr>
<tr>
<td>blade.interp_airfoils.coord_xy_interp</td>
<td></td>
<td>3D array of the non-dimensional x and y airfoil coordinates of the airfoils interpolated along span for n_span stations. The leading edge is place at x=0 and y=0.</td>
</tr>
<tr>
<td>blade.interp_airfoils.name</td>
<td>Un-av-</td>
<td>1D array of names of airfoils.</td>
</tr>
<tr>
<td>available</td>
<td></td>
<td></td>
</tr>
<tr>
<td>blade.interp_airfoils.pitch_axis</td>
<td></td>
<td>1D array of the chordwise position of the pitch axis (0-LE, 1-TE), defined along blade span.</td>
</tr>
<tr>
<td>blade.interp_airfoils.r_thick</td>
<td></td>
<td>1D array of the relative thicknesses of each airfoil.</td>
</tr>
<tr>
<td>blade.interp_airfoils.r_thick_interp</td>
<td></td>
<td>1D array of the relative thicknesses of the blade defined along span.</td>
</tr>
<tr>
<td>blade.interp_airfoils.s</td>
<td></td>
<td>1D array of the non-dimensional spanwise grid defined along blade axis (0-blade root, 1-blade tip)1D array of the non-dimensional grid defined along the column axis (0-column base, 1-column top)</td>
</tr>
<tr>
<td>blade.opt_var.af_position</td>
<td></td>
<td>1D array of the non dimensional positions of the airfoils af_used defined along blade span.</td>
</tr>
<tr>
<td>blade.opt_var.chord_opt_gain</td>
<td></td>
<td></td>
</tr>
<tr>
<td>blade.opt_var.s_opt_chord</td>
<td></td>
<td></td>
</tr>
<tr>
<td>blade.opt_var.s_opt_twist</td>
<td></td>
<td></td>
</tr>
<tr>
<td>blade.opt_var.spar_cap_ps_opt_gain</td>
<td></td>
<td></td>
</tr>
<tr>
<td>blade.opt_var.spar_cap_ss_opt_gain</td>
<td></td>
<td></td>
</tr>
<tr>
<td>blade.opt_var.tweak_opt_gain</td>
<td></td>
<td></td>
</tr>
<tr>
<td>blade.outer_shape_bem.af_position</td>
<td></td>
<td>1D array of the chord values defined along blade span.Chord length at each section.</td>
</tr>
<tr>
<td>blade.outer_shape_bem.chord</td>
<td>m</td>
<td>1D array of the chord values defined along blade span.</td>
</tr>
<tr>
<td>blade.outer_shape_bem.chord_yaml</td>
<td></td>
<td>1D array of the chord values defined along blade span.</td>
</tr>
<tr>
<td>blade.outer_shape_bem.pitch_axis</td>
<td></td>
<td>1D array of the chordwise position of the pitch axis (0-LE, 1-TE), defined along blade span.</td>
</tr>
<tr>
<td>blade.outer_shape_bem.pitch_axis_yaml</td>
<td></td>
<td>1D array of the chordwise position of the pitch axis (0-LE, 1-TE), defined along blade span.</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Variable</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>blade.outer_shape_bem.ref_axis</td>
<td>m</td>
<td>2D array of the coordinates ((x,y,z)) of the blade reference axis, defined along blade span. The coordinate system is the one of BeamDyn: it is placed at blade root with (x) pointing the suction side of the blade, (y) pointing the trailing edge and (z) along the blade span. A standard configuration will have negative (x) values (prebend), if swept positive (y) values, and positive (z) values.</td>
</tr>
<tr>
<td>blade.outer_shape_bem.ref_axis.yaml</td>
<td>m</td>
<td>2D array of the coordinates ((x,y,z)) of the blade reference axis, defined along blade span. The coordinate system is the one of BeamDyn: it is placed at blade root with (x) pointing the suction side of the blade, (y) pointing the trailing edge and (z) along the blade span. A standard configuration will have negative (x) values (prebend), if swept positive (y) values, and positive (z) values.</td>
</tr>
<tr>
<td>blade.outer_shape_bem.s</td>
<td></td>
<td>1D array of the non-dimensional spanwise grid defined along blade axis (0-blade root, 1-blade tip)</td>
</tr>
<tr>
<td>blade.outer_shape_bem.s.default</td>
<td></td>
<td>1D array of the non-dimensional spanwise grid defined along blade axis (0-blade root, 1-blade tip)</td>
</tr>
<tr>
<td>blade.outer_shape_bem.span_end</td>
<td></td>
<td>1D array of the positions along blade span where something (a DAC device?) starts and we want a grid point. Only values between 0 and 1 are meaningful.</td>
</tr>
<tr>
<td>blade.outer_shape_bem.span_ext</td>
<td></td>
<td>1D array of the extensions along blade span where something (a DAC device?) lives and we want a grid point. Only values between 0 and 1 are meaningful.</td>
</tr>
<tr>
<td>blade.outer_shape_bem.twist</td>
<td>rad</td>
<td>1D array of the twist values defined along blade span. The twist is defined positive for negative rotations around the (z) axis (the same as in BeamDyn).</td>
</tr>
<tr>
<td>blade.outer_shape_bem.twist.yaml</td>
<td>rad</td>
<td>1D array of the twist values defined along blade span. The twist is defined positive for negative rotations around the (z) axis (the same as in BeamDyn).</td>
</tr>
<tr>
<td>blade.pa.chord_opt_gain</td>
<td></td>
<td>1D array of the non-dimensional gains to optimize the blade spanwise distribution of the chord</td>
</tr>
<tr>
<td>blade.pa.chord_original</td>
<td>m</td>
<td>1D array of the chord values defined along blade span. The chord is the one defined in the yaml.</td>
</tr>
<tr>
<td>blade.pa.chord_param</td>
<td>m</td>
<td>1D array of the chord values defined along blade span. The chord is the result of the parameterization.</td>
</tr>
<tr>
<td>blade.pa.max_chord_constr</td>
<td></td>
<td>1D array of the ratio between chord values and maximum chord along blade span.</td>
</tr>
<tr>
<td>blade.pa.s</td>
<td></td>
<td>1D array of the non-dimensional spanwise grid defined along blade axis (0-blade root, 1-blade tip)</td>
</tr>
<tr>
<td>blade.pa.s_opt_chord</td>
<td></td>
<td>1D array of the non-dimensional spanwise grid defined along blade axis to optimize the blade chord</td>
</tr>
<tr>
<td>blade.pa.s_opt_twist</td>
<td></td>
<td>1D array of the non-dimensional spanwise grid defined along blade axis to optimize the blade twist angle</td>
</tr>
<tr>
<td>blade.pa.twist_opt_gain</td>
<td></td>
<td>1D array of the non-dimensional gains to optimize the blade spanwise distribution of the twist angle</td>
</tr>
<tr>
<td>blade.pa.twist_original</td>
<td>rad</td>
<td>1D array of the twist values defined along blade span. The twist is the one defined in the yaml.</td>
</tr>
<tr>
<td>blade.pa.twist_param</td>
<td>rad</td>
<td>1D array of the twist values defined along blade span. The twist is the result of the parameterization.</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Variable</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>blade.ps.layer_thickness_original</td>
<td>m</td>
<td>2D array of the thickness of the layers of the blade structure. The first dimension represents each layer, the second dimension represents each entry along blade span.</td>
</tr>
<tr>
<td>blade.ps.layer_thickness_param</td>
<td>m</td>
<td>2D array of the thickness of the layers of the blade structure after the parametrization. The first dimension represents each layer, the second dimension represents each entry along blade span.</td>
</tr>
<tr>
<td>blade.ps.s</td>
<td></td>
<td>1D array of the non-dimensional spanwise grid defined along blade axis (0-blade root, 1-blade tip)</td>
</tr>
<tr>
<td>blade.ps.s_opt_spar_cap_ps</td>
<td></td>
<td>1D array of the non-dimensional spanwise grid defined along blade axis to optimize the blade spar cap pressure side</td>
</tr>
<tr>
<td>blade.ps.s_opt_spar_cap_ss</td>
<td></td>
<td>1D array of the non-dimensional spanwise grid defined along blade axis to optimize the blade spar cap suction side</td>
</tr>
<tr>
<td>blade.ps.spar_cap_ps_opt_gain</td>
<td></td>
<td>1D array of the non-dimensional gains to optimize the blade spanwise distribution of the spar caps pressure side</td>
</tr>
<tr>
<td>blade.ps.spar_cap_ss_opt_gain</td>
<td></td>
<td>1D array of the non-dimensional gains to optimize the blade spanwise distribution of the spar caps suction side</td>
</tr>
<tr>
<td>bos.boem_review_cost</td>
<td>USD</td>
<td></td>
</tr>
<tr>
<td>boscommissioning_pct</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bos.construction_operations_plan_cost</td>
<td>USD</td>
<td></td>
</tr>
<tr>
<td>bos.decommissioning_pct</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bos.distance_to_interconnection</td>
<td>km</td>
<td></td>
</tr>
<tr>
<td>bos.distance_to_landfall</td>
<td>km</td>
<td></td>
</tr>
<tr>
<td>bos.distance_to_substation</td>
<td>km</td>
<td></td>
</tr>
<tr>
<td>bos.interconnect_voltage</td>
<td>kV</td>
<td></td>
</tr>
<tr>
<td>bos.plant_row_spacing</td>
<td></td>
<td>Distance between turbine rows in rotor diameters</td>
</tr>
<tr>
<td>bos.plant_turbine_spacing</td>
<td></td>
<td>Distance between turbines in rotor diameters</td>
</tr>
<tr>
<td>bos.port_cost_per_month</td>
<td>USD</td>
<td></td>
</tr>
<tr>
<td>bos.site_assessment_cost</td>
<td>USD</td>
<td></td>
</tr>
<tr>
<td>bos.site_assessment_plan_cost</td>
<td>USD</td>
<td></td>
</tr>
<tr>
<td>bos.site_auction_price</td>
<td>USD</td>
<td></td>
</tr>
<tr>
<td>bos.site_distance</td>
<td>km</td>
<td></td>
</tr>
<tr>
<td>ccblade.CM</td>
<td></td>
<td>Blade flapwise moment coefficient</td>
</tr>
<tr>
<td>ccblade.CP</td>
<td></td>
<td>Rotor power coefficient</td>
</tr>
<tr>
<td>ccblade.D_n_opt</td>
<td>N/m</td>
<td>Distributed drag force</td>
</tr>
<tr>
<td>ccblade.DragF</td>
<td>N/m</td>
<td>Distributed drag force</td>
</tr>
<tr>
<td>ccblade.L_n_opt</td>
<td>N/m</td>
<td>Distributed lift force</td>
</tr>
<tr>
<td>ccblade.LiftF</td>
<td>N/m</td>
<td>Distributed lift force</td>
</tr>
<tr>
<td>ccblade.Px_af</td>
<td>N/m</td>
<td>Distributed loads in airfoil x-direction</td>
</tr>
<tr>
<td>ccblade.Px_b</td>
<td>N/m</td>
<td>Distributed loads in blade-aligned x-direction</td>
</tr>
<tr>
<td>ccblade.Py_af</td>
<td>N/m</td>
<td>Distributed loads in airfoil y-direction</td>
</tr>
<tr>
<td>ccblade.Py_b</td>
<td>N/m</td>
<td>Distributed loads in blade-aligned y-direction</td>
</tr>
<tr>
<td>ccblade.Pz_af</td>
<td>N/m</td>
<td>Distributed loads in airfoil z-direction</td>
</tr>
<tr>
<td>ccblade.Pz_b</td>
<td>N/m</td>
<td>Distributed loads in blade-aligned z-direction</td>
</tr>
<tr>
<td>ccblade.Rhub</td>
<td>m</td>
<td>Hub radius</td>
</tr>
<tr>
<td>ccblade.Rtip</td>
<td>m</td>
<td>Tip radius</td>
</tr>
<tr>
<td>ccblade.Uhub</td>
<td>m/s</td>
<td>Undisturbed wind speed</td>
</tr>
<tr>
<td>ccblade.a</td>
<td></td>
<td>Axial induction along blade span</td>
</tr>
<tr>
<td>ccblade.airfoils_Re</td>
<td></td>
<td>Reynolds numbers of polars</td>
</tr>
</tbody>
</table>

continues on next page
Table 5.2 – continued from previous page

<table>
<thead>
<tr>
<th>Variable</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ccbblade.airfoils_aoa</td>
<td>deg</td>
<td>angle of attack grid for polarsAngle of attack grid for polars.</td>
</tr>
<tr>
<td>ccbblade.airfoils_cd</td>
<td></td>
<td>drag coefficients, spanwiseDrag coefficients, spanwise.</td>
</tr>
<tr>
<td>ccbblade.airfoils_cl</td>
<td></td>
<td>lift coefficients, spanwiseLift coefficients, spanwise.</td>
</tr>
<tr>
<td>ccbblade.airfoils_cm</td>
<td></td>
<td>moment coefficients, spanwiseMoment coefficients, spanwise.</td>
</tr>
<tr>
<td>ccbblade.alpha</td>
<td>deg</td>
<td>Angles of attack along blade span</td>
</tr>
<tr>
<td>ccbblade.ap</td>
<td></td>
<td>Tangential induction along blade span</td>
</tr>
<tr>
<td>ccbblade.cd</td>
<td></td>
<td>Drag coefficients along blade span</td>
</tr>
<tr>
<td>ccbblade.cd_n_opt</td>
<td></td>
<td>Drag coefficients along blade span</td>
</tr>
<tr>
<td>ccbblade.chord</td>
<td>m</td>
<td>chord length at each sectionChord length at each section.</td>
</tr>
<tr>
<td>ccbblade.cl</td>
<td></td>
<td>Lift coefficients along blade span</td>
</tr>
<tr>
<td>ccbblade.cl_n_opt</td>
<td></td>
<td>Lift coefficients along blade span</td>
</tr>
<tr>
<td>ccbblade.hub_height</td>
<td>m</td>
<td>hub heightHub height of wind turbine above ground / sea level</td>
</tr>
<tr>
<td>ccbblade.hubloss</td>
<td>Unavail-</td>
<td>include Prandtl hub loss modelInclude Prandtl hub loss model.</td>
</tr>
<tr>
<td>ccbblade.mu</td>
<td>kg/(m*s)</td>
<td>dynamic viscosity of airDynamic viscosity of air</td>
</tr>
<tr>
<td>ccbblade.nBlades</td>
<td>Unavail-</td>
<td>number of bladesNumber of blades</td>
</tr>
<tr>
<td>ccbblade.nSector</td>
<td>Unavail-</td>
<td>number of sectors to divide rotor face into in computing thrust and power-Number of sectors to divide rotor face into in computing thrust and power.</td>
</tr>
<tr>
<td>ccbblade.pitch</td>
<td>deg</td>
<td>Pitch angle</td>
</tr>
<tr>
<td>ccbblade.precone</td>
<td>deg</td>
<td>precone angleRotor precone angle</td>
</tr>
<tr>
<td>ccbblade.precurve</td>
<td>m</td>
<td>precurve at each sectionPrecurve at each section.</td>
</tr>
<tr>
<td>ccbblade.precurveTip</td>
<td>m</td>
<td>precurve at tipPrecurve at tip.</td>
</tr>
<tr>
<td>ccbblade.presweep</td>
<td>m</td>
<td>presweep at each section</td>
</tr>
<tr>
<td>ccbblade.presweepTip</td>
<td>m</td>
<td>presweep at tip</td>
</tr>
<tr>
<td>ccbblade.r</td>
<td>m</td>
<td>radial locations where blade is defined (should be increasing and not go all the way to hub or tip)Radial locations where blade is defined. Should be increasing and not go all the way to hub or tip.</td>
</tr>
<tr>
<td>ccbblade.rho</td>
<td>kg/m**3</td>
<td>density of airDensity of the materials along the column sections.</td>
</tr>
<tr>
<td>ccbblade.rthick</td>
<td></td>
<td>1D array of the relative thicknesses of the blade defined along span.</td>
</tr>
<tr>
<td>ccbblade.s_opt_chord</td>
<td></td>
<td>1D array of the non-dimensional spanwise grid defined along blade axis to optimize the blade chord</td>
</tr>
<tr>
<td>ccbblade.s_opt_twist</td>
<td></td>
<td>1D array of the non-dimensional spanwise grid defined along blade axis to optimize the blade twist</td>
</tr>
<tr>
<td>ccbblade.shearExp</td>
<td></td>
<td>shear exponentsShear exponent</td>
</tr>
<tr>
<td>ccbblade.theta</td>
<td>rad</td>
<td>Twist angle at each section (positive decreases angle of attack)Twist angle at each section (positive decreases angle of attack).</td>
</tr>
<tr>
<td>ccbblade.tilt</td>
<td>deg</td>
<td>shaft tiltNacelle up tilt angle</td>
</tr>
<tr>
<td>ccbblade.tiploss</td>
<td>Unavail-</td>
<td>include Prandtl tip loss modelInclude Prandtl tip loss model.</td>
</tr>
<tr>
<td>ccbblade.tsr</td>
<td></td>
<td>Tip speed ratio</td>
</tr>
<tr>
<td>ccbblade.twist</td>
<td>rad</td>
<td>twist angle at each section (positive decreases angle of attack)</td>
</tr>
<tr>
<td>ccbblade.usecd</td>
<td>Unavail-</td>
<td>use drag coefficient in computing induction factorsUse drag coefficient in computing induction factors.</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Variable</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ccbblade.wakerotation</td>
<td>Unavailable</td>
<td>Include effect of wake rotation (i.e., tangential induction factor is nonzero).</td>
</tr>
<tr>
<td>ccbblade.yaw</td>
<td>deg</td>
<td>Yaw error angle</td>
</tr>
<tr>
<td>configuration.gearbox_type</td>
<td>Unavailable</td>
<td>Gearbox configuration (geared, direct-drive, etc.).</td>
</tr>
<tr>
<td>configuration.hub_height_user</td>
<td>m</td>
<td>Height of the hub center over the ground (land-based) or the mean sea level (offshore) specified by the user.</td>
</tr>
<tr>
<td>configuration.n_blades</td>
<td>Unavailable</td>
<td>Number of blades of the rotor.</td>
</tr>
<tr>
<td>configuration.rated_power</td>
<td>W</td>
<td>Electrical rated power of the generator.</td>
</tr>
<tr>
<td>configuration.rotor_diameter_user</td>
<td>m</td>
<td>Diameter of the rotor specified by the user. It is defined as two times the blade length plus the hub diameter.</td>
</tr>
<tr>
<td>configuration.rotor_orientation</td>
<td>Unavailable</td>
<td>Rotor orientation, either upwind or downwind.</td>
</tr>
<tr>
<td>configuration.turb_class</td>
<td>Unavailable</td>
<td>IEC wind turbine category. A - high turbulence intensity (land-based), B - mid turbulence, C - low turbulence (offshore).</td>
</tr>
<tr>
<td>configuration.upwind</td>
<td>Unavailable</td>
<td>Convenient boolean for upwind (True) or downwind (False). Flag whether the design is upwind or downwind.</td>
</tr>
<tr>
<td>configuration.ws_class</td>
<td>Unavailable</td>
<td>IEC wind turbine class. I - offshore, II coastal, III - land-based, IV - low wind speed site.</td>
</tr>
<tr>
<td>control.V_in</td>
<td>m/s</td>
<td>Cut in wind speed. This is the wind speed where region II begins.</td>
</tr>
<tr>
<td>control.V_out</td>
<td>m/s</td>
<td>Cut out wind speed. This is the wind speed where region III ends.</td>
</tr>
<tr>
<td>control.maxOmega</td>
<td>rad/s</td>
<td>Maximum allowed rotor speed.</td>
</tr>
<tr>
<td>control.max_Ts</td>
<td>m/s</td>
<td>Maximum allowed blade tip speed.</td>
</tr>
<tr>
<td>control.max_pitch_rate</td>
<td>rad/s</td>
<td>Maximum allowed blade pitch rate.</td>
</tr>
<tr>
<td>control.max_torque_rate</td>
<td>N*m/s</td>
<td>Maximum allowed generator torque rate.</td>
</tr>
<tr>
<td>control.minOmega</td>
<td>rad/s</td>
<td>Minimum allowed rotor speed.</td>
</tr>
<tr>
<td>control.rating TSR</td>
<td></td>
<td>Constant tip speed ratio in region II.</td>
</tr>
<tr>
<td>control.rating pitch</td>
<td>rad</td>
<td>Constant pitch angle in region II.</td>
</tr>
<tr>
<td>costs.bearing_mass_cost_coeff</td>
<td>USD/kg</td>
<td>Main bearing mass-cost coeff.</td>
</tr>
<tr>
<td>costs.bedplate_mass_cost_coeff</td>
<td>USD/kg</td>
<td>Bedplate mass-cost coeff.</td>
</tr>
<tr>
<td>costs.bos_per_kW</td>
<td>USD/kW</td>
<td>Balance of station/plant capital cost.</td>
</tr>
<tr>
<td>costs.controls_machine_rating_cost_coeff</td>
<td>USD/kW</td>
<td>Controls cost coefficient per kW.</td>
</tr>
<tr>
<td>costs.cover_mass_coeff</td>
<td>USD/kg</td>
<td>Nacelle cover mass_cost coeff.</td>
</tr>
<tr>
<td>costs.elec_connec_machine_rating_cost_coeff</td>
<td>USD/kW</td>
<td>Electrical connections cost coefficient per kW.</td>
</tr>
<tr>
<td>costs.elec_connec_mass_coeff</td>
<td>USD/kg</td>
<td>Electrical connections mass cost coeff.</td>
</tr>
<tr>
<td>costs.generator_mass_cost_coeff</td>
<td>USD/kg</td>
<td>Generator mass cost coeff.</td>
</tr>
<tr>
<td>costs.hss_mass_cost_coeff</td>
<td>USD/kg</td>
<td>High speed shaft mass-cost coeff.</td>
</tr>
<tr>
<td>costs.hub_mass_cost_coeff</td>
<td>USD/kg</td>
<td>Hub mass-cost coeff.</td>
</tr>
</tbody>
</table>
Table 5.2 – continued from previous page

<table>
<thead>
<tr>
<th>Variable</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>costs.hvac_mass_cost_coeff</td>
<td>USD/kg</td>
<td>hydraulic and cooling system mass cost coeff</td>
</tr>
<tr>
<td>costs.labor_rate</td>
<td>USD/h</td>
<td></td>
</tr>
<tr>
<td>costs.lss_mass_cost_coeff</td>
<td>USD/kg</td>
<td>slow speed shaft mass-cost coeff</td>
</tr>
<tr>
<td>costs.offset_tcc_per_kW</td>
<td>USD/kW</td>
<td>Offset to turbine capital cost Offset to turbine capital cost</td>
</tr>
<tr>
<td>costs.opex_per_kW</td>
<td>USD/kW/yr</td>
<td>Average annual operational expenditures of the turbine Average annual operational expenditures of the turbine</td>
</tr>
<tr>
<td>costs.painting_rate</td>
<td>USD/m²²</td>
<td></td>
</tr>
<tr>
<td>costs.pitch_system_mass_cost_coeff</td>
<td>USD/kg</td>
<td>pitch system mass-cost coeff</td>
</tr>
<tr>
<td>costs.platforms_mass_cost_coeff</td>
<td>USD/kg</td>
<td>nacelle platforms mass cost coeff</td>
</tr>
<tr>
<td>costs.spinner_mass_cost_coeff</td>
<td>USD/kg</td>
<td>spinner/nose cone mass-cost coeff</td>
</tr>
<tr>
<td>costs.tower_mass_cost_coeff</td>
<td>USD/kg</td>
<td>tower mass-cost coeff</td>
</tr>
<tr>
<td>costs.transformer_mass_cost_coeff</td>
<td>USD/kg</td>
<td>transformer mass cost coeff</td>
</tr>
<tr>
<td>costs.turbine_number</td>
<td>Unavailable</td>
<td>Number of turbines at plant Number of turbines at plant</td>
</tr>
<tr>
<td>costs.wake_loss_factor</td>
<td>The losses in AEP due to waked conditions</td>
<td>The losses in AEP due to waked conditions</td>
</tr>
<tr>
<td>costs.yaw_mass_cost_coeff</td>
<td>USD/kg</td>
<td>yaw system mass cost coeff</td>
</tr>
<tr>
<td>drivese.D_bearing1</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>drivese.D_bearing2</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>drivese.D_bedplate</td>
<td>m</td>
<td>Bedplate diameters</td>
</tr>
<tr>
<td>drivese.D_gearbox</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>drivese.D_top</td>
<td>m</td>
<td>Tower top outer diameter</td>
</tr>
<tr>
<td>drivese.E_mat</td>
<td>Pa</td>
<td>2D array of the Youngs moduli of the materials. Each row represents a material, the three columns represent E11, E22 and E33.</td>
</tr>
<tr>
<td>drivese.F_generator</td>
<td>N</td>
<td>Force vector applied to the hub (WITH WEIGHT????)</td>
</tr>
<tr>
<td>drivese.F_hub</td>
<td>N</td>
<td>Force vector applied to bearing 1 in hub c.s.</td>
</tr>
<tr>
<td>drivese.F_mb1</td>
<td>N</td>
<td>Force vector applied to bearing 2 in hub c.s.</td>
</tr>
<tr>
<td>drivese.F_mb2</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>drivese.F_torq</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>drivese.G_mat</td>
<td>Pa</td>
<td>2D array of the shear moduli of the materials. Each row represents a material, the three columns represent G12, G13 and G23.</td>
</tr>
<tr>
<td>drivese.H_bedplate</td>
<td>m</td>
<td>height of bedplate</td>
</tr>
<tr>
<td>drivese.L_12</td>
<td>m</td>
<td>Length from bearing #1 to bearing #2</td>
</tr>
<tr>
<td>drivese.L_bedplate</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>drivese.L_drive</td>
<td>m</td>
<td>Length of drivetrain from bedplate to hub flang</td>
</tr>
<tr>
<td>drivese.L_gearbox</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>drivese.L_generator</td>
<td>m</td>
<td>Generator stack width</td>
</tr>
<tr>
<td>drivese.L_h1</td>
<td>m</td>
<td>Length from hub / start of lss to bearing #1</td>
</tr>
<tr>
<td>drivese.L_hss</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>drivese.L_lss</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>drivese.L_nose</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>drivese.M_generator</td>
<td>N*m</td>
<td>Moment vector applied to the hub</td>
</tr>
<tr>
<td>drivese.M_hub</td>
<td>N*m</td>
<td>Moment vector applied to bearing 1 in hub c.s.</td>
</tr>
<tr>
<td>drivese.M_mb1</td>
<td>N*m</td>
<td>Moment vector applied to bearing 2 in hub c.s.</td>
</tr>
<tr>
<td>drivese.M_mb2</td>
<td>N*m</td>
<td></td>
</tr>
<tr>
<td>drivese.M_torq</td>
<td>N*m</td>
<td></td>
</tr>
<tr>
<td>drivese.R_generator</td>
<td>m</td>
<td>Generator outer diameter</td>
</tr>
<tr>
<td>drivese.Xt_mat</td>
<td>Pa</td>
<td></td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Variable</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>drivese.above_yaw_I</td>
<td>kg*m²</td>
<td></td>
</tr>
<tr>
<td>drivese.above_yaw_cm</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>drivese.above_yaw_mass</td>
<td>kg</td>
<td></td>
</tr>
<tr>
<td>drivese.access_diameter</td>
<td>m</td>
<td>Minimum diameter required for maintenance access</td>
</tr>
<tr>
<td>drivese.base_F</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>drivese.base_M</td>
<td>N*m</td>
<td></td>
</tr>
<tr>
<td>drivese.bear1.D_bearing</td>
<td>m</td>
<td>bearing diameter/facewidth</td>
</tr>
<tr>
<td>drivese.bear1.D_shaft</td>
<td>m</td>
<td>Shaft diameter</td>
</tr>
<tr>
<td>drivese.bear1.bearing_type</td>
<td>Unavailable</td>
<td>bearing mass type</td>
</tr>
<tr>
<td>drivese.bear1.mb_I</td>
<td>kg*m²</td>
<td></td>
</tr>
<tr>
<td>drivese.bear1.mb_mass</td>
<td>kg</td>
<td></td>
</tr>
<tr>
<td>drivese.bear1.mb_max_defl_rad</td>
<td></td>
<td></td>
</tr>
<tr>
<td>drivese.bear2.D_bearing</td>
<td>m</td>
<td>bearing diameter/facewidth</td>
</tr>
<tr>
<td>drivese.bear2.D_shaft</td>
<td>m</td>
<td>Shaft diameter</td>
</tr>
<tr>
<td>drivese.bear2.bearing_type</td>
<td>Unavailable</td>
<td>bearing mass type</td>
</tr>
<tr>
<td>drivese.bear2.mb_I</td>
<td>kg*m²</td>
<td></td>
</tr>
<tr>
<td>drivese.bear2.mb_mass</td>
<td>kg</td>
<td></td>
</tr>
<tr>
<td>drivese.bear2.mb_max_defl_rad</td>
<td></td>
<td></td>
</tr>
<tr>
<td>drivese.bedplate_E</td>
<td>Pa</td>
<td>modulus of elasticity</td>
</tr>
<tr>
<td>drivese.bedplate_G</td>
<td>Pa</td>
<td>shear modulus</td>
</tr>
<tr>
<td>drivese.bedplate_I</td>
<td>kg*m²</td>
<td>component I</td>
</tr>
<tr>
<td>drivese.bedplate_Xy</td>
<td>Pa</td>
<td>yield stress</td>
</tr>
<tr>
<td>drivese.bedplate_axial_stress</td>
<td>Pa</td>
<td></td>
</tr>
<tr>
<td>drivese.bedplate_bending_stress</td>
<td>Pa</td>
<td></td>
</tr>
<tr>
<td>drivese.bedplate_cm</td>
<td>m</td>
<td>component CM</td>
</tr>
<tr>
<td>drivese.bedplate_deflection</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>drivese.bedplate_flange_thickness</td>
<td>m</td>
<td>Bedplate is two parallel I beams, this is the flange thickness</td>
</tr>
<tr>
<td>drivese.bedplate_flange_width</td>
<td>m</td>
<td>Bedplate is two parallel I beams, this is the flange width</td>
</tr>
<tr>
<td>drivese.bedplate_mass</td>
<td>kg</td>
<td>component mass</td>
</tr>
<tr>
<td>drivese.bedplate_mat_cost</td>
<td>USD/kg</td>
<td></td>
</tr>
<tr>
<td>drivese.bedplate_material</td>
<td>Unavailable</td>
<td></td>
</tr>
<tr>
<td>drivese.bedplate_nose_axial_stress</td>
<td>Pa</td>
<td></td>
</tr>
<tr>
<td>drivese.bedplate_nose_bending_stress</td>
<td>Pa</td>
<td></td>
</tr>
<tr>
<td>drivese.bedplate_nose_shear_stress</td>
<td>Pa</td>
<td></td>
</tr>
<tr>
<td>drivese.bedplate_rho</td>
<td>kg/m³</td>
<td>material density</td>
</tr>
<tr>
<td>drivese.bedplate_rotation</td>
<td>rad</td>
<td></td>
</tr>
<tr>
<td>drivese.bedplate_shear_stress</td>
<td>Pa</td>
<td></td>
</tr>
<tr>
<td>drivese.bedplate_wall_thickness</td>
<td>m</td>
<td>Bedplate wall thickness</td>
</tr>
<tr>
<td>drivese.bedplate_web_height</td>
<td>m</td>
<td>Bedplate is two parallel I beams, this is the web height</td>
</tr>
<tr>
<td>drivese.bedplate_web_thickness</td>
<td>m</td>
<td>Bedplate is two parallel I beams, this is the web thickness</td>
</tr>
<tr>
<td>drivese.blade_mass</td>
<td>kg</td>
<td>Total mass of one blade</td>
</tr>
<tr>
<td>drivese.blade_root_diameter</td>
<td>m</td>
<td>Outer diameter of blade root</td>
</tr>
<tr>
<td>drivese.blades_I</td>
<td>kg*m²</td>
<td>Mass moments of inertia of all blades about hub center</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Variable</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>drivese.blades_mass</td>
<td>kg</td>
<td>Mass of all bladea</td>
</tr>
<tr>
<td>drivese.brake_I</td>
<td>kg*m^2</td>
<td>Component I</td>
</tr>
<tr>
<td>drivese.brake_cm</td>
<td>m</td>
<td>Component CM</td>
</tr>
<tr>
<td>drivese.brake_mass</td>
<td>kg</td>
<td>Component mass</td>
</tr>
<tr>
<td>drivese.brake_mass_user</td>
<td>kg</td>
<td>User override of brake mass</td>
</tr>
<tr>
<td>drivese.carrier_I</td>
<td>kg*m^2</td>
<td>Component inertia</td>
</tr>
<tr>
<td>drivese.carrier_mass</td>
<td>kg</td>
<td>Component mass</td>
</tr>
<tr>
<td>drivese.clearance_hub_spinner</td>
<td>m</td>
<td>Clearance between spinner and hub</td>
</tr>
<tr>
<td>drivese.constr_access</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>drivese.constr_bedplate_vonmises</td>
<td></td>
<td></td>
</tr>
<tr>
<td>drivese.constr_cm</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>drivese.constr_ecc</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>drivese.constr_height</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>drivese.constr_hub_diameter</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>drivese.constr_length</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>drivese.constr_lss_vonmises</td>
<td></td>
<td></td>
</tr>
<tr>
<td>drivese.constr_mb1_defl</td>
<td></td>
<td></td>
</tr>
<tr>
<td>drivese.constr_mb2_defl</td>
<td></td>
<td></td>
</tr>
<tr>
<td>drivese.converter_I</td>
<td>kg*m^2</td>
<td>Moments of Inertia for the component [Ixx, Iyy, Izz] around its center of mass</td>
</tr>
<tr>
<td>drivese.converter_cm</td>
<td>m</td>
<td>Center of mass of the component in [x,y,z] for an arbitrary coordinate system</td>
</tr>
<tr>
<td>drivese.converter_mass</td>
<td>kg</td>
<td>Overall component mass</td>
</tr>
<tr>
<td>drivese.converter_mass_user</td>
<td>kg</td>
<td>Override regular regression-based calculation of converter mass with this value</td>
</tr>
<tr>
<td>drivese.cover_I</td>
<td>m</td>
<td>Component mass moments of inertia</td>
</tr>
<tr>
<td>drivese.cover_cm</td>
<td>m</td>
<td>Component center of mass</td>
</tr>
<tr>
<td>drivese.cover_mass</td>
<td>kg</td>
<td>Component mass</td>
</tr>
<tr>
<td>drivese.drive_height</td>
<td>m</td>
<td>Hub height above tower top</td>
</tr>
<tr>
<td>drivese.flange_ID2flange_OD</td>
<td></td>
<td>Ratio of flange inner diameter to flange outer diameter</td>
</tr>
<tr>
<td>drivese.flange_OD2hub_D</td>
<td></td>
<td>Ratio of flange outer diameter to hub diameter</td>
</tr>
<tr>
<td>drivese.flange_t2shell_t</td>
<td></td>
<td>Ratio of flange thickness to shell thickness</td>
</tr>
<tr>
<td>drivese.gear_configuration</td>
<td>Unavail-able</td>
<td>3-letter string of Es or Ps to denote epicyclic or parallel gear configuration</td>
</tr>
<tr>
<td>drivese.gear_ratio</td>
<td></td>
<td>Overall gearbox ratio</td>
</tr>
<tr>
<td>drivese.gearbox_I</td>
<td>kg*m^2</td>
<td>Gearbox moment of inertia (measured about its cm)</td>
</tr>
<tr>
<td>drivese.gearbox_cm</td>
<td>m</td>
<td>Component CM</td>
</tr>
<tr>
<td>drivese.gearbox_mass</td>
<td>kg</td>
<td>Gearbox rotor mass</td>
</tr>
<tr>
<td>drivese.generator.A_1</td>
<td>mm**2</td>
<td></td>
</tr>
<tr>
<td>drivese.generator.A_Curcalc</td>
<td>mm**2</td>
<td></td>
</tr>
<tr>
<td>drivese.generator.A_Cuscalc</td>
<td>mm**2</td>
<td></td>
</tr>
<tr>
<td>drivese.generator.B_g</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>drivese.generator.B_g1</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>drivese.generator.B_pm1</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>drivese.generator.B_r</td>
<td>T</td>
<td>Remnant flux density</td>
</tr>
<tr>
<td>drivese.generator.B_rymax</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>drivese.generator.B_smax</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>drivese.generator.B_symax</td>
<td>T</td>
<td>Peak Stator Yoke flux density B_ymax</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Variable</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>drivese.generator.B_tmax</td>
<td>T</td>
<td>Peak Teeth flux density</td>
</tr>
<tr>
<td>drivese.generator.B_trmax</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>drivese.generator.B_tsmax</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>drivese.generator.C_Cu</td>
<td>USD/kg</td>
<td>Specific cost of copper</td>
</tr>
<tr>
<td>drivese.generator.C_Fe</td>
<td>USD/kg</td>
<td>Specific cost of magnetic steel/iron</td>
</tr>
<tr>
<td>drivese.generator.C_Fes</td>
<td>USD/kg</td>
<td>Specific cost of structural steel</td>
</tr>
<tr>
<td>drivese.generator.C_PM</td>
<td>USD/kg</td>
<td>Specific cost of Magnet</td>
</tr>
<tr>
<td>drivese.generator.Copper</td>
<td>kg</td>
<td>Copper mass</td>
</tr>
<tr>
<td>drivese.generator.Current_ratio</td>
<td></td>
<td></td>
</tr>
<tr>
<td>drivese.generator.D_nose</td>
<td>m</td>
<td>Nose outer diameter</td>
</tr>
<tr>
<td>drivese.generator.D_ratio</td>
<td></td>
<td></td>
</tr>
<tr>
<td>drivese.generator.D_ratio_L</td>
<td></td>
<td></td>
</tr>
<tr>
<td>drivese.generator.D_ratio_LL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>drivese.generator.D_ratio_U</td>
<td></td>
<td></td>
</tr>
<tr>
<td>drivese.generator.D_ratio_UL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>drivese.generator.D_shaft</td>
<td>m</td>
<td>Shaft diameter</td>
</tr>
<tr>
<td>drivese.generator.E</td>
<td>Pa</td>
<td>Isotropic Youngs modulus of the materials along the column sections.</td>
</tr>
<tr>
<td>drivese.generator.E_p</td>
<td>V</td>
<td>Stator phase voltage</td>
</tr>
<tr>
<td>drivese.generator.G</td>
<td>Pa</td>
<td>Isotropic shear modulus of the materials along the column sections.</td>
</tr>
<tr>
<td>drivese.generator.I_0</td>
<td>A</td>
<td>No-load excitation current</td>
</tr>
<tr>
<td>drivese.generator.I_s</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>drivese.generator.Iron</td>
<td>kg</td>
<td>Iron mass</td>
</tr>
<tr>
<td>drivese.generator.J_actual</td>
<td>A/m**2</td>
<td></td>
</tr>
<tr>
<td>drivese.generator.J_r</td>
<td></td>
<td></td>
</tr>
<tr>
<td>drivese.generator.J_s</td>
<td>A*m**2</td>
<td></td>
</tr>
<tr>
<td>drivese.generator.K_rad</td>
<td></td>
<td></td>
</tr>
<tr>
<td>drivese.generator.K_rad_L</td>
<td></td>
<td></td>
</tr>
<tr>
<td>drivese.generator.K_rad_LL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>drivese.generator.K_rad_U</td>
<td></td>
<td></td>
</tr>
<tr>
<td>drivese.generator.K_rad_UL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>drivese.generator.L_r</td>
<td></td>
<td></td>
</tr>
<tr>
<td>drivese.generator.L_s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>drivese.generator.L_sm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>drivese.generator.Losses</td>
<td>W</td>
<td></td>
</tr>
<tr>
<td>drivese.generator.Mass_tooth_stator</td>
<td></td>
<td></td>
</tr>
<tr>
<td>drivese.generator.Mass_yoke_rotor</td>
<td></td>
<td></td>
</tr>
<tr>
<td>drivese.generator.Mass_yoke_stator</td>
<td></td>
<td></td>
</tr>
<tr>
<td>drivese.generator.N_c</td>
<td></td>
<td>Number of turns per coil</td>
</tr>
<tr>
<td>drivese.generator.N_r</td>
<td></td>
<td></td>
</tr>
<tr>
<td>drivese.generator.N_s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>drivese.generator.P_Fe0e</td>
<td>W/kg</td>
<td>Specific eddy losses @ 1.5T, 50Hz</td>
</tr>
<tr>
<td>drivese.generator.P_Fe0h</td>
<td>W/kg</td>
<td>Specific hysteresis losses W / kg @ 1.5 T @50 Hz</td>
</tr>
<tr>
<td>drivese.generator.Q_r</td>
<td>W</td>
<td>Shaft mechanical power</td>
</tr>
<tr>
<td>drivese.generator.R_R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>drivese.generator.R_OUT</td>
<td>m</td>
<td>Outer radius</td>
</tr>
<tr>
<td>drivese.generator.R_s</td>
<td>ohm</td>
<td></td>
</tr>
<tr>
<td>drivese.generator.S</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

continues on next page
Table 5.2 – continued from previous page

<table>
<thead>
<tr>
<th>Variable</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>drivese.generator.S_N</td>
<td></td>
<td>Slip</td>
</tr>
<tr>
<td>drivese.generator.S_Nmax</td>
<td></td>
<td>Max rated Slip</td>
</tr>
<tr>
<td>drivese.generator.Slot_aspect_ratio</td>
<td></td>
<td></td>
</tr>
<tr>
<td>drivese.generator.Slot_aspect_ratio1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>drivese.generator.Slot_aspect_ratio2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>drivese.generator.Structural_mass</td>
<td>kg</td>
<td>Structural mass</td>
</tr>
<tr>
<td>drivese.generator.Structural_mass_rotor</td>
<td></td>
<td></td>
</tr>
<tr>
<td>drivese.generator.Structural_mass_stator</td>
<td></td>
<td></td>
</tr>
<tr>
<td>drivese.generator.TC1</td>
<td>m**3</td>
<td></td>
</tr>
<tr>
<td>drivese.generator.TC2r</td>
<td>m**3</td>
<td></td>
</tr>
<tr>
<td>drivese.generator.TC2s</td>
<td>m**3</td>
<td></td>
</tr>
<tr>
<td>drivese.generator.TCr</td>
<td>m**3</td>
<td></td>
</tr>
<tr>
<td>drivese.generator.TCs</td>
<td>m**3</td>
<td></td>
</tr>
<tr>
<td>drivese.generator.T_e</td>
<td>N*m</td>
<td></td>
</tr>
<tr>
<td>drivese.generator.alpha_p</td>
<td></td>
<td>Slot pole combination</td>
</tr>
<tr>
<td>drivese.generator.b_allow_r</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>drivese.generator.b_allow_s</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>drivese.generator.b_arm</td>
<td>m</td>
<td>arm width</td>
</tr>
<tr>
<td>drivese.generator.b_m</td>
<td></td>
<td></td>
</tr>
<tr>
<td>drivese.generator.b_r</td>
<td></td>
<td></td>
</tr>
<tr>
<td>drivese.generator.b_r_tau_r</td>
<td></td>
<td>Rotor Slot width / Slot pitch ratio</td>
</tr>
<tr>
<td>drivese.generator.b_ro</td>
<td>m</td>
<td>Rotor slot opening width</td>
</tr>
<tr>
<td>drivese.generator.b_s</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>drivese.generator.b_s_tau_s</td>
<td></td>
<td>Stator Slot width/Slot pitch ratio</td>
</tr>
<tr>
<td>drivese.generator.b_so</td>
<td>m</td>
<td>Stator slot opening width</td>
</tr>
<tr>
<td>drivese.generator.b_st</td>
<td>m</td>
<td>arm width b_st</td>
</tr>
<tr>
<td>drivese.generator.b_t</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>drivese.generator.b_tr</td>
<td></td>
<td></td>
</tr>
<tr>
<td>drivese.generator.b_trmin</td>
<td></td>
<td></td>
</tr>
<tr>
<td>drivese.generator.c</td>
<td></td>
<td>Slot pole combination</td>
</tr>
<tr>
<td>drivese.generator.cofi</td>
<td></td>
<td>power factor</td>
</tr>
<tr>
<td>drivese.generator.con_Bsmax</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>drivese.generator.con_TC2r</td>
<td>m**3</td>
<td></td>
</tr>
<tr>
<td>drivese.generator.con_TC2s</td>
<td>m**3</td>
<td></td>
</tr>
<tr>
<td>drivese.generator.con_br</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>drivese.generator.con_bst</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>drivese.generator.con_uar</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>drivese.generator.con_uas</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>drivese.generator.con_yar</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>drivese.generator.con_yas</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>drivese.generator.con_zar</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>drivese.generator.con_zas</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>drivese.generator.d_r</td>
<td>m</td>
<td>arm depth d_r</td>
</tr>
<tr>
<td>drivese.generator.d_s</td>
<td>m</td>
<td>arm depth d_s</td>
</tr>
<tr>
<td>drivese.generator.f</td>
<td></td>
<td></td>
</tr>
<tr>
<td>drivese.generator.freq</td>
<td>Hz</td>
<td>grid frequency</td>
</tr>
<tr>
<td>drivese.generator.h_0</td>
<td>m</td>
<td>Slot height</td>
</tr>
<tr>
<td>drivese.generator.h_i</td>
<td>m</td>
<td>coil insulation thickness</td>
</tr>
</tbody>
</table>

continues on next page
Table 5.2 – continued from previous page

<table>
<thead>
<tr>
<th>Variable</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>drivese.generator.h_m</td>
<td>m</td>
<td>magnet height</td>
</tr>
<tr>
<td>drivese.generator.h_s</td>
<td>m</td>
<td>Yoke height h_s</td>
</tr>
<tr>
<td>drivese.generator.h_sr</td>
<td>m</td>
<td>Structural Mass</td>
</tr>
<tr>
<td>drivese.generator.h_ss</td>
<td>m</td>
<td>Stator yoke height</td>
</tr>
<tr>
<td>drivese.generator.h_sy0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>drivese.generator.h_t</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>drivese.generator.h_w</td>
<td>m</td>
<td>Slot wedge height</td>
</tr>
<tr>
<td>drivese.generator.h_yr</td>
<td>m</td>
<td>Yoke height</td>
</tr>
<tr>
<td>drivese.generator.h_yts</td>
<td></td>
<td></td>
</tr>
<tr>
<td>drivese.generator.k_fes</td>
<td></td>
<td>Stator iron fill factor per Grauers</td>
</tr>
<tr>
<td>drivese.generator.k_fillr</td>
<td></td>
<td>Rotor slot fill factor</td>
</tr>
<tr>
<td>drivese.generator.k_fillss</td>
<td></td>
<td>Stator Slot fill factor</td>
</tr>
<tr>
<td>drivese.generator.k_s</td>
<td>m</td>
<td>magnetic saturation factor for iron</td>
</tr>
<tr>
<td>drivese.generator.len_ag</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>drivese.generator.len_s</td>
<td>m</td>
<td>Stator core length</td>
</tr>
<tr>
<td>drivese.generator.m</td>
<td></td>
<td>added mass</td>
</tr>
<tr>
<td>drivese.generator.mass_PM</td>
<td>kg</td>
<td>Magnet mass</td>
</tr>
<tr>
<td>drivese.generator.mu_0</td>
<td>m*kg/s**2</td>
<td>permeability of free space</td>
</tr>
<tr>
<td>drivese.generator.mu_r</td>
<td>m*kg/s**2</td>
<td>relative permeability (neodymium)</td>
</tr>
<tr>
<td>drivese.generator.n_r</td>
<td></td>
<td>number of arms n</td>
</tr>
<tr>
<td>drivese.generator.n_s</td>
<td></td>
<td>number of stator arms n_s</td>
</tr>
<tr>
<td>drivese.generator.p</td>
<td>rad</td>
<td>tilt angle (during transportation)</td>
</tr>
<tr>
<td>drivese.generator.q</td>
<td>N/m**2</td>
<td></td>
</tr>
<tr>
<td>drivese.generator.q1</td>
<td></td>
<td>Stator slots per pole per phase</td>
</tr>
<tr>
<td>drivese.generator.q2</td>
<td></td>
<td>Rotor slots per pole per phase</td>
</tr>
<tr>
<td>drivese.generator.rad_ag</td>
<td>m</td>
<td>airgap radius</td>
</tr>
<tr>
<td>drivese.generator.ratio_mw2pp</td>
<td></td>
<td>ratio of magnet width to pole pitch(bm / self.tau_p)</td>
</tr>
<tr>
<td>drivese.generator.rho_Cu</td>
<td>ohm/m</td>
<td>Copper resistivity</td>
</tr>
<tr>
<td>drivese.generator.rho_Copper</td>
<td>kg*m**3</td>
<td>Copper density</td>
</tr>
<tr>
<td>drivese.generator.rho_Fe</td>
<td></td>
<td>Magnetic Steel density</td>
</tr>
<tr>
<td>drivese.generator.rho_Fes</td>
<td>kg*m**3</td>
<td>Structural Steel density</td>
</tr>
<tr>
<td>drivese.generator.rho_PM</td>
<td>kg*m**3</td>
<td>Magnet density</td>
</tr>
<tr>
<td>drivese.generator.shaft_rpm</td>
<td>rpm</td>
<td>rated speed of input shaft (lss for direct, hss for geared)</td>
</tr>
<tr>
<td>drivese.generator.sigma</td>
<td>Pa</td>
<td>assumed max shear stress</td>
</tr>
<tr>
<td>drivese.generator.t_r</td>
<td>m</td>
<td>Rotor disc thickness</td>
</tr>
<tr>
<td>drivese.generator.t_s</td>
<td>m</td>
<td>Stator disc thickness</td>
</tr>
<tr>
<td>drivese.generator.t_wr</td>
<td>m</td>
<td>arm depth thickness</td>
</tr>
<tr>
<td>drivese.generator.t_ws</td>
<td>m</td>
<td>arm depth thickness</td>
</tr>
<tr>
<td>drivese.generator.tau_p</td>
<td>m</td>
<td>Pole pitch self.tau_p</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Variable</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>drivese.generator.tau_s</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>drivese.generator.theta_bd</td>
<td>rad</td>
<td>Slope at the bedplate</td>
</tr>
<tr>
<td>drivese.generator.theta_sh</td>
<td>rad</td>
<td>slope of shaft</td>
</tr>
<tr>
<td>drivese.generator.twist_r</td>
<td>deg</td>
<td></td>
</tr>
<tr>
<td>drivese.generator.twist_s</td>
<td>deg</td>
<td></td>
</tr>
<tr>
<td>drivese.generator.u_allow_pcent</td>
<td></td>
<td>Radial deflection as a percentage of air gap diameter</td>
</tr>
<tr>
<td>drivese.generator.u_allow_r</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>drivese.generator.u_allow_s</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>drivese.generator.u_ar</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>drivese.generator.u_as</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>drivese.generator.v</td>
<td></td>
<td>poisson ratio</td>
</tr>
<tr>
<td>drivese.generator.y_allow_pcent</td>
<td></td>
<td>Radial deflection as a percentage of air gap diameter</td>
</tr>
<tr>
<td>drivese.generator.y_allow_r</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>drivese.generator.y_allow_s</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>drivese.generator.y_ar</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>drivese.generator.y_as</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>drivese.generator.y_bd</td>
<td>m</td>
<td>Deflection of the bedplate</td>
</tr>
<tr>
<td>drivese.generator.y_sh</td>
<td>m</td>
<td>Shaft deflection</td>
</tr>
<tr>
<td>drivese.generator.y_tau_p</td>
<td></td>
<td>Stator coil span to pole pitch</td>
</tr>
<tr>
<td>drivese.generator.y_tau_pr</td>
<td></td>
<td>Rotor coil span to pole pitch</td>
</tr>
<tr>
<td>drivese.generator.z_allow_deg</td>
<td>deg</td>
<td>Allowable torsional twist</td>
</tr>
<tr>
<td>drivese.generator.z_allow_r</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>drivese.generator.z_allow_s</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>drivese.generator.z_ar</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>drivese.generator.z_as</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>drivese.generator_I</td>
<td>kg*m**2</td>
<td>Component I</td>
</tr>
<tr>
<td>drivese.generator_cm</td>
<td>m</td>
<td>Component CM</td>
</tr>
<tr>
<td>drivese.generator_cost</td>
<td>USD</td>
<td>Generator cost</td>
</tr>
<tr>
<td>drivese.generator_efficiency</td>
<td></td>
<td></td>
</tr>
<tr>
<td>drivese.generator_mass</td>
<td>kg</td>
<td>Actual mass</td>
</tr>
<tr>
<td>drivese.generatorRotor_I</td>
<td>kg*m**2</td>
<td>Generator rotor moment of inertia (measured about its cm)</td>
</tr>
<tr>
<td>drivese.generatorRotor_mass</td>
<td>kg</td>
<td>Generator rotor mass</td>
</tr>
<tr>
<td>drivese.generatorStator_I</td>
<td>kg*m**2</td>
<td>Generator stator moment of inertia (measured about cm)</td>
</tr>
<tr>
<td>drivese.generatorStator_mass</td>
<td>kg</td>
<td>Generator stator mass</td>
</tr>
<tr>
<td>drivese.hss_E</td>
<td>Pa</td>
<td>Modulus of elasticity</td>
</tr>
<tr>
<td>drivese.hss_G</td>
<td>Pa</td>
<td>Shear modulus</td>
</tr>
<tr>
<td>drivese.hss_I</td>
<td>kg*m**2</td>
<td>Component I</td>
</tr>
<tr>
<td>drivese.hss_Xy</td>
<td>Pa</td>
<td>Yield stress</td>
</tr>
<tr>
<td>drivese.hss_axial_stress</td>
<td>Pa</td>
<td></td>
</tr>
<tr>
<td>drivese.hss_bending_stress</td>
<td>Pa</td>
<td></td>
</tr>
<tr>
<td>drivese.hss_cm</td>
<td>m</td>
<td>Component CM</td>
</tr>
<tr>
<td>drivese.hss_cost</td>
<td>USD/kg</td>
<td>Hss cost</td>
</tr>
<tr>
<td>drivese.hss_diameter</td>
<td>m</td>
<td>Lss discretized diameter values at coordinates</td>
</tr>
<tr>
<td>drivese.hss_mass</td>
<td>kg</td>
<td>Component mass</td>
</tr>
<tr>
<td>drivese.hss_material</td>
<td>Unavailable</td>
<td></td>
</tr>
<tr>
<td>drivese.hss_rho</td>
<td>kg/m**3</td>
<td>Material density</td>
</tr>
<tr>
<td>drivese.hss_rpm</td>
<td>rpm</td>
<td></td>
</tr>
</tbody>
</table>
Table 5.2 – continued from previous page

<table>
<thead>
<tr>
<th>Variable</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>drivese.hss_shear_stress</td>
<td>Pa</td>
<td>Lss discretized thickness values at coordinates</td>
</tr>
<tr>
<td>drivese.hss_wall_thickness</td>
<td>m</td>
<td>Lss discretized thickness values at coordinates</td>
</tr>
<tr>
<td>drivese.hub_E</td>
<td>Pa</td>
<td>Total mass moment of inertia of the hub about its cm</td>
</tr>
<tr>
<td>drivese.hub_G</td>
<td>Pa</td>
<td>Total mass moment of inertia of the hub about its cm</td>
</tr>
<tr>
<td>drivese.hub_I</td>
<td>kg*m**2</td>
<td>Total mass moment of inertia of the hub about its cm</td>
</tr>
<tr>
<td>drivese.hub_cm</td>
<td>m</td>
<td>Distance between hub/shaft flange and hub center of mass</td>
</tr>
<tr>
<td>drivese.hub_cost</td>
<td>USD</td>
<td>Cost of the hub shell, including flanges</td>
</tr>
<tr>
<td>drivese.hub_diameter</td>
<td>m</td>
<td>Outer diameter of the hub</td>
</tr>
<tr>
<td>drivese.hub_in2out_circ</td>
<td></td>
<td></td>
</tr>
<tr>
<td>drivese.hub_mass</td>
<td>kg</td>
<td>Total mass of the hub shell, including the flanges</td>
</tr>
<tr>
<td>drivese.hub_mat_cost</td>
<td>USD/kg</td>
<td></td>
</tr>
<tr>
<td>drivese.hub_material</td>
<td>Unavail</td>
<td></td>
</tr>
<tr>
<td>drivese.hub_rho</td>
<td>kg/m**3</td>
<td></td>
</tr>
<tr>
<td>drivese.hub_shell.Xy</td>
<td>Pa</td>
<td>Yield strength metal</td>
</tr>
<tr>
<td>drivese.hub_shell.metal_cost</td>
<td>USD/kg</td>
<td>Unit cost metal</td>
</tr>
<tr>
<td>drivese.hub_shell.n_blades</td>
<td>Unavail</td>
<td>Number of rotor blades</td>
</tr>
<tr>
<td>drivese.hub_shell.rho</td>
<td>kg/m**3</td>
<td>Density of the materials along the column sections.</td>
</tr>
<tr>
<td>drivese.hub_stress_concentration</td>
<td></td>
<td>Stress concentration factor. Stress concentration occurs at all fillets,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>notches, lifting lugs, hatches and are accounted for by assigning a stress</td>
</tr>
<tr>
<td></td>
<td></td>
<td>concentration factor</td>
</tr>
<tr>
<td>drivese.hub_system_I</td>
<td>kg*m**2</td>
<td>Hub system moment of inertia</td>
</tr>
<tr>
<td>drivese.hub_system_cm</td>
<td>m</td>
<td>Hub system center of mass distance from hub flange</td>
</tr>
<tr>
<td>drivese.hub_system_cost</td>
<td>USD</td>
<td>Cost for hub system</td>
</tr>
<tr>
<td>drivese.lss_E</td>
<td>Pa</td>
<td>Modulus of elasticity</td>
</tr>
<tr>
<td>drivese.lss_G</td>
<td>Pa</td>
<td>Shear modulus</td>
</tr>
<tr>
<td>drivese.lss_I</td>
<td>kg*m**2</td>
<td>LSS moment of inertia around cm in axial (hub-aligned) c.s.</td>
</tr>
<tr>
<td>drivese.lss_Xy</td>
<td>Pa</td>
<td>Yield stress</td>
</tr>
<tr>
<td>drivese.lss_axial_stress</td>
<td>Pa</td>
<td></td>
</tr>
<tr>
<td>drivese.lss_cm</td>
<td>m</td>
<td>LSS center of mass along shaft axis from bedplate</td>
</tr>
<tr>
<td>drivese.lss_cost</td>
<td>USD/kg</td>
<td>LSS cost</td>
</tr>
<tr>
<td>drivese.lss_diameter</td>
<td>m</td>
<td>LSS outer diameter from hub to bearing 2</td>
</tr>
<tr>
<td>drivese.lss_mass</td>
<td>kg</td>
<td>LSS mass</td>
</tr>
<tr>
<td>drivese.lss_material</td>
<td>Unavail</td>
<td></td>
</tr>
<tr>
<td>drivese.lss_rho</td>
<td>kg/m**3</td>
<td>LSS material density</td>
</tr>
<tr>
<td>drivese.lss_rpm</td>
<td>rpm</td>
<td></td>
</tr>
<tr>
<td>drivese.lss_shear_stress</td>
<td>Pa</td>
<td></td>
</tr>
<tr>
<td>drivese.lss_wall_thickness</td>
<td>m</td>
<td>LSS wall thickness</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Variable</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>drivese.machine_rating</td>
<td>kW</td>
<td>Machine rating</td>
</tr>
<tr>
<td>drivese.material_names</td>
<td>Un-available</td>
<td>1D array of names of materials.</td>
</tr>
<tr>
<td>drivese.max_torque</td>
<td>N*m</td>
<td>Max torque that the hub needs to resist (Mx in a hub aligned reference system)</td>
</tr>
<tr>
<td>drivese.mb1_I</td>
<td>kg*m</td>
<td>Component I</td>
</tr>
<tr>
<td>drivese.mb1_cm</td>
<td>m</td>
<td>Component CM</td>
</tr>
<tr>
<td>drivese.mb1_deflection</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>drivese.mb1_mass</td>
<td>kg</td>
<td>Component mass</td>
</tr>
<tr>
<td>drivese.mb1_max_defl_ang</td>
<td>rad</td>
<td>Maximum allowable deflection angle</td>
</tr>
<tr>
<td>drivese.mb1_rotation</td>
<td>rad</td>
<td></td>
</tr>
<tr>
<td>drivese.mb2_I</td>
<td>kg*m</td>
<td>Component I</td>
</tr>
<tr>
<td>drivese.mb2_cm</td>
<td>m</td>
<td>Component CM</td>
</tr>
<tr>
<td>drivese.mb2_deflection</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>drivese.mb2_mass</td>
<td>kg</td>
<td>Component mass</td>
</tr>
<tr>
<td>drivese.mb2_max_defl_ang</td>
<td>rad</td>
<td>Maximum allowable deflection angle</td>
</tr>
<tr>
<td>drivese.mb2_rotation</td>
<td>rad</td>
<td></td>
</tr>
<tr>
<td>drivese.mean_bearing_mass</td>
<td>kg</td>
<td></td>
</tr>
<tr>
<td>drivese.minimum_rpm</td>
<td>rpm</td>
<td>Minimum shaft rotations-per-minute (rpm), usually set by controller</td>
</tr>
<tr>
<td>drivese.n_blades</td>
<td>Un-available</td>
<td>Number of rotor blades</td>
</tr>
<tr>
<td>drivese.n_front_brackets</td>
<td>Un-available</td>
<td>Number of front spinner brackets</td>
</tr>
<tr>
<td>drivese.n_rear_brackets</td>
<td>Un-available</td>
<td>Number of rear spinner brackets</td>
</tr>
<tr>
<td>drivese.nacelle_I</td>
<td>kg*m</td>
<td>Mass moments of inertia of nacelle about its CoM</td>
</tr>
<tr>
<td>drivese.nacelle_cm</td>
<td>m</td>
<td>Nacelle center of mass relative to tower top in yaw-aligned c.s.</td>
</tr>
<tr>
<td>drivese.nacelle_mass</td>
<td>kg</td>
<td>Mass of nacelle system</td>
</tr>
<tr>
<td>drivese.nose_I</td>
<td>kg*m</td>
<td>Nose moment of inertia around cm in axial (hub-aligned) c.s.</td>
</tr>
<tr>
<td>drivese.nose_cm</td>
<td>m</td>
<td>Nose center of mass along nose axis from bedplate</td>
</tr>
<tr>
<td>drivese.nose_diameter</td>
<td>m</td>
<td>Nose outer diameter from bearing 1 to bedplate</td>
</tr>
<tr>
<td>drivese.nose_mass</td>
<td>kg</td>
<td>Nose mass</td>
</tr>
<tr>
<td>drivese.nose_wall_thickness</td>
<td>m</td>
<td>Nose wall thickness</td>
</tr>
<tr>
<td>drivese.other_mass</td>
<td>kg</td>
<td>Mass of other nacelle components that rest on mainplate</td>
</tr>
<tr>
<td>drivese.overhang</td>
<td>m</td>
<td>Horizontal distance between hub and tower-top axis</td>
</tr>
<tr>
<td>drivese.pitch_I</td>
<td>kg*m</td>
<td>Total mass moment of inertia of the pitch system about central point</td>
</tr>
<tr>
<td>drivese.pitch_cost</td>
<td>USD</td>
<td>Cost of the pitch system</td>
</tr>
<tr>
<td>drivese.pitch_mass</td>
<td>kg</td>
<td>Total mass of the pitch system</td>
</tr>
<tr>
<td>drivese.pitch_system.BRFM</td>
<td>N*m</td>
<td>Flapwise bending moment at blade root</td>
</tr>
<tr>
<td>drivese.pitch_system.Xy</td>
<td>Pa</td>
<td>Yield strength metal</td>
</tr>
<tr>
<td>drivese.pitch_system.rho</td>
<td>kg/m</td>
<td>Density of the materials along the column sections.</td>
</tr>
<tr>
<td>drivese.pitch_system_scaling_factor</td>
<td></td>
<td>Scaling factor to tune the total mass (0.34 is recommended for modern designs)</td>
</tr>
<tr>
<td>drivese.planet_numbers</td>
<td>Un-available</td>
<td></td>
</tr>
<tr>
<td>Variable</td>
<td>Units</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>drivese.platform_I</td>
<td>m</td>
<td>component mass moments of inertia</td>
</tr>
<tr>
<td>drivese.platform_cm</td>
<td>m</td>
<td>component center of mass</td>
</tr>
<tr>
<td>drivese.platform_mass</td>
<td>kg</td>
<td>component mass</td>
</tr>
<tr>
<td>drivese.rated_rpm</td>
<td>rpm</td>
<td>Rated shaft rotations-per-minute (rpm)</td>
</tr>
<tr>
<td>drivese.rated_torque</td>
<td>N*m</td>
<td>rotor torque at rated power</td>
</tr>
<tr>
<td>drivese.rho_castiron</td>
<td>kg/m**3</td>
<td></td>
</tr>
<tr>
<td>drivese.rho_fiberglass</td>
<td>kg/m**3</td>
<td>material density of fiberglass</td>
</tr>
<tr>
<td>drivese.rho_mat</td>
<td>kg/m**3</td>
<td>3D array of the density of the materials. For composites, this is the density of the laminate.</td>
</tr>
<tr>
<td>drivese.rna_I_TT</td>
<td>kg*m**2</td>
<td></td>
</tr>
<tr>
<td>drivese.rna_cm</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>drivese.rna_mass</td>
<td>kg</td>
<td>Mass of RNA</td>
</tr>
<tr>
<td>drivese.rotor_diameter</td>
<td>m</td>
<td>rotor diameter</td>
</tr>
<tr>
<td>drivese.rotor_mass</td>
<td>kg</td>
<td>Total rotor mass</td>
</tr>
<tr>
<td>drivese.s_drive</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>drivese.s_gearbox</td>
<td>m</td>
<td>Gearbox s-coordinate measured from bedplate</td>
</tr>
<tr>
<td>drivese.s_generator</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>drivese.s_hss</td>
<td>m</td>
<td>Discretized s-coordinates along drivetrain, measured from bedplate (direct) or tower center (geared)</td>
</tr>
<tr>
<td>drivese.s_lss</td>
<td>m</td>
<td>Discretized s-coordinates along drivetrain, measured from bedplate (direct) or tower center (geared)</td>
</tr>
<tr>
<td>drivese.s_mb1</td>
<td>m</td>
<td>Bearing 1 s-coordinate along drivetrain, measured from bedplate</td>
</tr>
<tr>
<td>drivese.s_mb2</td>
<td>m</td>
<td>Bearing 2 s-coordinate along drivetrain, measured from bedplate</td>
</tr>
<tr>
<td>drivese.s_nose</td>
<td>m</td>
<td>Discretized s-coordinates along drivetrain, measured from bedplate</td>
</tr>
<tr>
<td>drivese.s_rotor</td>
<td>m</td>
<td>Generator rotor attachment to lss s-coordinate measured from bedplate (direct) or tower center (geared)</td>
</tr>
<tr>
<td>drivese.s_stator</td>
<td>m</td>
<td>Generator stator attachment to lss s-coordinate measured from bedplate</td>
</tr>
<tr>
<td>drivese.sigma_y_mat</td>
<td>Pa</td>
<td>2D array of the yield strength of the materials. Each row represents a material, the three columns represent Xt12, Xt13 and Xt23.</td>
</tr>
<tr>
<td>drivese.spin_hole_incr</td>
<td></td>
<td>Ratio between access hole diameter in the spinner and blade root diameter. Typical value 1.2</td>
</tr>
<tr>
<td>drivese.spinner.Xy</td>
<td>Pa</td>
<td>Yield strength metal</td>
</tr>
<tr>
<td>drivese.spinner.composite_Xt</td>
<td>Pa</td>
<td>Tensile strength of the composite material of the shell. A glass CFM (continuous fiber mat) is often used.</td>
</tr>
<tr>
<td>drivese.spinner.composite_cost</td>
<td>USD/kg</td>
<td>Unit cost composite of the shell</td>
</tr>
<tr>
<td>drivese.spinner.composite_rho</td>
<td>kg/m**3</td>
<td>Density of composite of the shell</td>
</tr>
<tr>
<td>drivese.spinner.metal_cost</td>
<td>USD/kg</td>
<td>Unit cost metal</td>
</tr>
<tr>
<td>drivese.spinner.metal_rho</td>
<td>kg/m**3</td>
<td>Density metal</td>
</tr>
<tr>
<td>drivese.spinner.spinner_diameter</td>
<td></td>
<td></td>
</tr>
<tr>
<td>drivese.spinner_I</td>
<td>kg*m**2</td>
<td>Total mass moment of inertia of the spinner about its cm</td>
</tr>
<tr>
<td>drivese.spinner_Xt</td>
<td>Pa</td>
<td></td>
</tr>
<tr>
<td>drivese.spinner_cm</td>
<td>m</td>
<td>Radius / Distance between center of mass of the spinner and outer surface</td>
</tr>
<tr>
<td>drivese.spinner_cost</td>
<td>kg</td>
<td>Cost of the spinner</td>
</tr>
<tr>
<td>drivese.spinner_gust_ws</td>
<td>m/s</td>
<td>Extreme gust wind speed</td>
</tr>
<tr>
<td>drivese.spinner_mass</td>
<td>kg</td>
<td>Total mass of the spinner</td>
</tr>
<tr>
<td>drivese.spinner_mat_cost</td>
<td>USD/kg</td>
<td></td>
</tr>
<tr>
<td>drivese.spinner_material</td>
<td>Unavailable</td>
<td></td>
</tr>
<tr>
<td>drivese.spinner_rho</td>
<td>kg/m**3</td>
<td></td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Variable</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>drivese.stage_ratios</td>
<td></td>
<td></td>
</tr>
<tr>
<td>drivese.stator_deflection</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>drivese.stator_rotation</td>
<td>rad</td>
<td></td>
</tr>
<tr>
<td>drivese.stop_time</td>
<td>s</td>
<td>Time required for the turbine rotor to come to a complete stop</td>
</tr>
<tr>
<td>drivese.t_bedplate</td>
<td>m</td>
<td>Bedplate wall thickness (mirrors input)</td>
</tr>
<tr>
<td>drivese.tilt</td>
<td>deg</td>
<td>Nacelle uptilt angle</td>
</tr>
<tr>
<td>drivese.torq_deflection</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>drivese.torq_rotation</td>
<td>rad</td>
<td></td>
</tr>
<tr>
<td>drivese.total_bedplate_mass</td>
<td>kg</td>
<td></td>
</tr>
<tr>
<td>drivese.transformer_I</td>
<td>kg*m</td>
<td>Moments of Inertia for the component [Ixx, Iyy, Izz] around its center of</td>
</tr>
<tr>
<td></td>
<td></td>
<td>mass</td>
</tr>
<tr>
<td>drivese.transformer_cm</td>
<td>m</td>
<td>Center of mass of the component in [x,y,z] for an arbitrary coordinate</td>
</tr>
<tr>
<td></td>
<td></td>
<td>system</td>
</tr>
<tr>
<td>drivese.transformer_mass</td>
<td>kg</td>
<td>Overall component mass</td>
</tr>
<tr>
<td>drivese.transformer_mass_user</td>
<td>kg</td>
<td>Override regular regression-based calculation of transformer mass with</td>
</tr>
<tr>
<td></td>
<td></td>
<td>this value</td>
</tr>
<tr>
<td>drivese.unit_cost_mat</td>
<td>USD/</td>
<td>1D array of the unit costs of the materials.</td>
</tr>
<tr>
<td></td>
<td>gD</td>
<td></td>
</tr>
<tr>
<td>drivese.uptower</td>
<td></td>
<td>Power electronics are placed in the nacelle at the tower top</td>
</tr>
<tr>
<td>drivese.upwind</td>
<td>Unavail</td>
<td>Flag whether the design is upwind or downwind</td>
</tr>
<tr>
<td>drivese.x_bedplate</td>
<td>m</td>
<td>Bedplate centerline x-coordinates</td>
</tr>
<tr>
<td>drivese.x_bedplate_inner</td>
<td>m</td>
<td>Bedplate lower curve x-coordinates</td>
</tr>
<tr>
<td>drivese.x_bedplate_outer</td>
<td>m</td>
<td>Bedplate outer curve x-coordinates</td>
</tr>
<tr>
<td>drivese.yaw.rho</td>
<td>kg/m**2</td>
<td>Density of the materials along the column sections.</td>
</tr>
<tr>
<td>drivese.yaw_I</td>
<td>kg*m</td>
<td>Moments of Inertia for the component [Ixx, Iyy, Izz] around its center of</td>
</tr>
<tr>
<td></td>
<td></td>
<td>mass</td>
</tr>
<tr>
<td>drivese.yaw_cm</td>
<td>m</td>
<td>Center of mass of the component in [x,y,z] for an arbitrary coordinate</td>
</tr>
<tr>
<td></td>
<td></td>
<td>system</td>
</tr>
<tr>
<td>drivese.yaw_mass</td>
<td>kg</td>
<td>Overall component mass</td>
</tr>
<tr>
<td>drivese.z_bedplate</td>
<td>m</td>
<td>Bedplate centerline z-coordinates</td>
</tr>
<tr>
<td>drivese.z_bedplate_inner</td>
<td>m</td>
<td>Bedplate lower curve z-coordinates</td>
</tr>
<tr>
<td>drivese.z_bedplate_outer</td>
<td>m</td>
<td>Bedplate outer curve z-coordinates</td>
</tr>
<tr>
<td>env.G_soil</td>
<td>N/m**2</td>
<td>Shear stress of soil</td>
</tr>
<tr>
<td>env.Hsig_wave</td>
<td>m</td>
<td>Significant wave height</td>
</tr>
<tr>
<td>env.Tsig_wave</td>
<td>s</td>
<td>Significant wave period</td>
</tr>
<tr>
<td>env.mu_air</td>
<td>kg/(m*s)</td>
<td>Dynamic viscosity of air</td>
</tr>
<tr>
<td>env.mu_water</td>
<td>kg/(m*s)</td>
<td>Dynamic viscosity of ocean water</td>
</tr>
<tr>
<td>env.nu_soil</td>
<td></td>
<td>Poisson ratio of soil</td>
</tr>
<tr>
<td>env.rho_air</td>
<td>kg/m**2</td>
<td>Density of air</td>
</tr>
<tr>
<td>env.rho_water</td>
<td>kg/m**2</td>
<td>Density of ocean water</td>
</tr>
<tr>
<td>env.speed_sound_air</td>
<td>m/s</td>
<td>Speed of sound in air</td>
</tr>
<tr>
<td>env.water_depth</td>
<td>m</td>
<td>Water depth for analysis. Values &gt; 0 mean offshore water depth</td>
</tr>
<tr>
<td>env.weibull_k</td>
<td></td>
<td>Shape parameter of the Weibull probability density function of the wind.</td>
</tr>
<tr>
<td>financese.bos_per_kW</td>
<td>USD/</td>
<td>Balance of system costs of the turbine</td>
</tr>
<tr>
<td></td>
<td>kW</td>
<td></td>
</tr>
<tr>
<td>financese.fixed_charge_rate</td>
<td>USD/</td>
<td>Fixed charge rate for coe calculation</td>
</tr>
<tr>
<td></td>
<td>W/h</td>
<td></td>
</tr>
<tr>
<td>financese.lcoe</td>
<td>USD/</td>
<td></td>
</tr>
<tr>
<td></td>
<td>W/h</td>
<td></td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Variable</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>financese.machine_rating</td>
<td>kW</td>
<td>Machine rating</td>
</tr>
<tr>
<td>financese.offset_tcc_per_kW</td>
<td>USD/kW</td>
<td>Offset to turbine capital cost</td>
</tr>
<tr>
<td>financese.opex_per_kW</td>
<td>USD/kW/yr</td>
<td>Average annual operational expenditures of the turbine</td>
</tr>
<tr>
<td>financese.plant_aep</td>
<td>USD/kW/yr</td>
<td>Annual Energy Production of the wind plant</td>
</tr>
<tr>
<td>financese.tcc_per_kW</td>
<td>USD/kW</td>
<td>Turbine capital cost</td>
</tr>
<tr>
<td>financese.turbine_aep</td>
<td>kW*h</td>
<td>Annual Energy Production of the wind turbine</td>
</tr>
<tr>
<td>financese.turbine_number</td>
<td>Un-available</td>
<td>Number of turbines at plant</td>
</tr>
<tr>
<td>financese.wake_loss_factor</td>
<td></td>
<td>The losses in AEP due to waked conditions</td>
</tr>
<tr>
<td>generator.B_r</td>
<td>T</td>
<td>Remnant flux density</td>
</tr>
<tr>
<td>generator.B_symax</td>
<td>T</td>
<td>Peak Stator Yoke flux density B_ymax</td>
</tr>
<tr>
<td>generator.B_tmax</td>
<td>T</td>
<td>Peak Teeth flux density</td>
</tr>
<tr>
<td>generator.C_Cu</td>
<td>USD/kg</td>
<td>Specific cost of copper</td>
</tr>
<tr>
<td>generator.C_Fe</td>
<td>USD/kg</td>
<td>Specific cost of magnetic steel/iron</td>
</tr>
<tr>
<td>generator.C_Fes</td>
<td>USD/kg</td>
<td>Specific cost of structural steel</td>
</tr>
<tr>
<td>generator.C_PM</td>
<td>USD/kg</td>
<td>Specific cost of Magnet</td>
</tr>
<tr>
<td>generator.E_p</td>
<td>V</td>
<td>Stator phase voltage</td>
</tr>
<tr>
<td>generator.I_0</td>
<td>A</td>
<td>no-load excitation current</td>
</tr>
<tr>
<td>generator.N_c</td>
<td></td>
<td>Number of turns per coil</td>
</tr>
<tr>
<td>generator.P_Fe0e</td>
<td>W/kg</td>
<td>specific eddy losses @ 1.5T, 50Hz</td>
</tr>
<tr>
<td>generator.P_Fe0h</td>
<td>W/kg</td>
<td>specific hysteresis losses W / kg @ 1.5 T @50 Hz</td>
</tr>
<tr>
<td>generator.S_N</td>
<td></td>
<td>Slip</td>
</tr>
<tr>
<td>generator.S_Nmax</td>
<td></td>
<td>Max rated Slip</td>
</tr>
<tr>
<td>generator.alpha_p</td>
<td></td>
<td></td>
</tr>
<tr>
<td>generator.b</td>
<td></td>
<td>Slot pole combination</td>
</tr>
<tr>
<td>generator.b_r_tau_r</td>
<td></td>
<td>Rotor Slot width / Slot pitch ratio</td>
</tr>
<tr>
<td>generator.b_ro</td>
<td>m</td>
<td>Rotor slot opening width</td>
</tr>
<tr>
<td>generator.b_s_tau_s</td>
<td></td>
<td>Stator Slot width/Slot pitch ratio</td>
</tr>
<tr>
<td>generator.b_so</td>
<td>m</td>
<td>Stator slot opening width</td>
</tr>
<tr>
<td>generator.b_st</td>
<td>m</td>
<td>arm width b_st</td>
</tr>
<tr>
<td>generator.c</td>
<td></td>
<td>Slot pole combination</td>
</tr>
<tr>
<td>generator.cofh</td>
<td></td>
<td>power factor</td>
</tr>
<tr>
<td>generator.d_r</td>
<td>m</td>
<td>arm depth d_r</td>
</tr>
<tr>
<td>generator.d_s</td>
<td>m</td>
<td>arm depth d_s</td>
</tr>
<tr>
<td>generator.freq</td>
<td>Hz</td>
<td>grid frequency</td>
</tr>
<tr>
<td>generator.h_i</td>
<td>m</td>
<td>coil insulation thickness</td>
</tr>
<tr>
<td>generator.h_m</td>
<td>m</td>
<td>magnet height</td>
</tr>
<tr>
<td>generator.h_s</td>
<td>m</td>
<td>Yoke height h_s</td>
</tr>
<tr>
<td>generator.h_sr</td>
<td>m</td>
<td>Structural Mass</td>
</tr>
<tr>
<td>generator.h_ss</td>
<td>m</td>
<td>Stator yoke height</td>
</tr>
<tr>
<td>generator.h_sy0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>generator.h_w</td>
<td>m</td>
<td>Slot wedge height</td>
</tr>
<tr>
<td>generator.h_yr</td>
<td>m</td>
<td>rotor yoke height</td>
</tr>
<tr>
<td>generator.h_yr</td>
<td>m</td>
<td>rotor yoke height</td>
</tr>
<tr>
<td>generator.k_fes</td>
<td></td>
<td>Stator iron fill factor per Grauers</td>
</tr>
<tr>
<td>generator.k_fillr</td>
<td></td>
<td>Rotor slot fill factor</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Variable</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>generator.k_fills</td>
<td></td>
<td>Stator Slot fill factor</td>
</tr>
<tr>
<td>generator.k_s</td>
<td></td>
<td>magnetic saturation factor for iron</td>
</tr>
<tr>
<td>generator.len_s</td>
<td>m</td>
<td>Stator core length</td>
</tr>
<tr>
<td>generator.m</td>
<td>Unavailable</td>
<td>added mass</td>
</tr>
<tr>
<td>generator.mu_0</td>
<td>m*kg/s**2</td>
<td>permeability of free space</td>
</tr>
<tr>
<td>generator.mu_r</td>
<td>m*kg/s**2</td>
<td>relative permeability (neodymium)</td>
</tr>
<tr>
<td>generator.n_r</td>
<td></td>
<td>number of arms n</td>
</tr>
<tr>
<td>generator.n_s</td>
<td></td>
<td>number of stator arms n_s</td>
</tr>
<tr>
<td>generator.p</td>
<td></td>
<td>pressure oscillation</td>
</tr>
<tr>
<td>generator.phi</td>
<td>rad</td>
<td>tilt angle (during transportation)</td>
</tr>
<tr>
<td>generator.q1</td>
<td>Unavailable</td>
<td>Stator slots per pole per phase</td>
</tr>
<tr>
<td>generator.q2</td>
<td>Unavailable</td>
<td>Rotor slots per pole per phase</td>
</tr>
<tr>
<td>generator.rad_ag</td>
<td>m</td>
<td>airgap radius</td>
</tr>
<tr>
<td>generator.ratio_mw2pp</td>
<td></td>
<td>ratio of magnet width to pole pitch(bm / self.tau_p)</td>
</tr>
<tr>
<td>generator.resist_Cu</td>
<td>ohm/m</td>
<td>Copper resistivity</td>
</tr>
<tr>
<td>generator.rho_Copper</td>
<td>kg*m/3</td>
<td>Copper density</td>
</tr>
<tr>
<td>generator.rho_Fe</td>
<td>kg*m/3</td>
<td>Magnetic Steel density</td>
</tr>
<tr>
<td>generator.rho_Fes</td>
<td>kg*m/3</td>
<td>Structural Steel density</td>
</tr>
<tr>
<td>generator.rho_PM</td>
<td>kg*m/3</td>
<td>Magnet density</td>
</tr>
<tr>
<td>generator.sigma</td>
<td>Pa</td>
<td>assumed max shear stress</td>
</tr>
<tr>
<td>generator.t_r</td>
<td>m</td>
<td>Rotor disc thickness</td>
</tr>
<tr>
<td>generator.t_s</td>
<td>m</td>
<td>Stator disc thickness</td>
</tr>
<tr>
<td>generator.t_ws</td>
<td>m</td>
<td>arm depth thickness</td>
</tr>
<tr>
<td>generator.t_wr</td>
<td>m</td>
<td>arm depth thickness</td>
</tr>
<tr>
<td>generator.y_allow_pcent</td>
<td></td>
<td>Radial deflection as a percentage of air gap diameter</td>
</tr>
<tr>
<td>generator.y_allow_pct</td>
<td></td>
<td>Radial deflection as a percentage of air gap diameter</td>
</tr>
<tr>
<td>generator.y_tau_p</td>
<td></td>
<td>Stator coil span to pole pitch</td>
</tr>
<tr>
<td>generator.y_tau_pr</td>
<td></td>
<td>Rotor coil span to pole pitch</td>
</tr>
<tr>
<td>generator.z_allow_deg</td>
<td>deg</td>
<td>Allowable torsional twist</td>
</tr>
<tr>
<td>hub.clearance_hub_spinner</td>
<td>m</td>
<td>Clearance between spinner and hub</td>
</tr>
<tr>
<td>hub.cone</td>
<td>rad</td>
<td>Cone angle of the rotor. It defines the angle between the rotor plane and the blade pitch axis. A standard machine has positive values.</td>
</tr>
<tr>
<td>hub.diameter</td>
<td>m</td>
<td>cylinder diameter at corresponding locations</td>
</tr>
<tr>
<td>hub.flange_ID2flange_OD</td>
<td></td>
<td>Ratio of flange inner diameter to flange outer diameter</td>
</tr>
<tr>
<td>hub.flange_OD2hub_D</td>
<td></td>
<td>Ratio of flange outer diameter to hub diameter</td>
</tr>
<tr>
<td>hub.flange_t2shell_t</td>
<td></td>
<td>Ratio of flange thickness to shell thickness</td>
</tr>
<tr>
<td>hub.hub_in2out_circ</td>
<td></td>
<td></td>
</tr>
<tr>
<td>hub.hub_material</td>
<td>Unavailable</td>
<td></td>
</tr>
</tbody>
</table>
Table 5.2 – continued from previous page

<table>
<thead>
<tr>
<th>Variable</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hub.hub_stress_concentration</td>
<td></td>
<td>Stress concentration factor. Stress concentration occurs at all fillets, notches, lifting lugs, hatches and are accounted for by assigning a stress concentration factor</td>
</tr>
<tr>
<td>hub.n_front_brackets</td>
<td>Unavailable</td>
<td>Number of front spinner brackets</td>
</tr>
<tr>
<td>hub.n_rear_brackets</td>
<td>Unavailable</td>
<td>Number of rear spinner brackets</td>
</tr>
<tr>
<td>hub.pitch_system_scaling_factor</td>
<td></td>
<td>Scaling factor to tune the total mass (0.54 is recommended for modern designs)</td>
</tr>
<tr>
<td>hub.radius</td>
<td>m</td>
<td>Radius of the hub. It defines the distance of the blade root from the rotor center along the coned line.</td>
</tr>
<tr>
<td>hub.spin_hole_incr</td>
<td></td>
<td>Ratio between access hole diameter in the spinner and blade root diameter. Typical value 1.2</td>
</tr>
<tr>
<td>hub.spinner_gust_ws</td>
<td>m/s</td>
<td>Extreme gust wind speed</td>
</tr>
<tr>
<td>hub.spinner_material</td>
<td>Unavailable</td>
<td></td>
</tr>
<tr>
<td>landbosse.Mass tonne</td>
<td>t</td>
<td></td>
</tr>
<tr>
<td>landbosse.allow_same_flag</td>
<td>Unavailable</td>
<td>Allow same crane for base and topping (True or False)</td>
</tr>
<tr>
<td>landbosse.bearing_pressure_n_m2</td>
<td></td>
<td>Bearing Pressure (n/m2)</td>
</tr>
<tr>
<td>landbosse.blade_drag_coefficient</td>
<td></td>
<td></td>
</tr>
<tr>
<td>landbosse.blade_drag_multiplier</td>
<td></td>
<td></td>
</tr>
<tr>
<td>landbosse.blade_install_cycle_time</td>
<td>h</td>
<td></td>
</tr>
<tr>
<td>landbosse.blade_lever_arm</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>landbosse.blade_mass</td>
<td>kg</td>
<td>The mass of one rotor blade. Total mass of one blade</td>
</tr>
<tr>
<td>landbosse.blade_offload_cycle_time</td>
<td>h</td>
<td></td>
</tr>
<tr>
<td>landbosse.blade_offload_hook_height</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>landbosse.bos_capex</td>
<td>USD</td>
<td>Total BOS CAPEX not including commissioning or decommissioning.</td>
</tr>
<tr>
<td>landbosse.bos_capex_kW</td>
<td>USD/kW</td>
<td>Total BOS CAPEX per kW not including commissioning or decommissioning.</td>
</tr>
<tr>
<td>landbosse.breakpoint_between_base_and_topping_percent</td>
<td></td>
<td>Breakpoint between base and topping (percent)</td>
</tr>
<tr>
<td>landbosse.cable_specs</td>
<td>Unavailable</td>
<td>cable specs for collection system</td>
</tr>
<tr>
<td>landbosse.commissioning_pct</td>
<td>Unavailable</td>
<td>Dataframe of components for tower, blade, nacelle</td>
</tr>
<tr>
<td>landbosse.components</td>
<td>Unavailable</td>
<td></td>
</tr>
</tbody>
</table>

continues on next page
Table 5.2 – continued from previous page

<table>
<thead>
<tr>
<th>Variable</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>landbosse.construct_duration</td>
<td></td>
<td>Total project construction time (months)</td>
</tr>
<tr>
<td>landbosse.crane_breakdown_fraction</td>
<td></td>
<td>0 means the crane is never broken down. 1 means it is broken down every turbine.</td>
</tr>
<tr>
<td>landbosse.crane_specs</td>
<td>Unavailable</td>
<td>Dataframe of specifications of cranes</td>
</tr>
<tr>
<td>landbosse.crane_width</td>
<td>m</td>
<td>Crane width (m)</td>
</tr>
<tr>
<td>landbosse.crew</td>
<td>Unavailable</td>
<td>Dataframe of crew configurations</td>
</tr>
<tr>
<td>landbosse.crew_price</td>
<td>Unavailable</td>
<td>Dataframe of costs per hour for each type of worker.</td>
</tr>
<tr>
<td>landbosse.critical_height_non_erection_wind_delays_m</td>
<td>m</td>
<td>Non-Erection Wind Delay Critical Height (m)</td>
</tr>
<tr>
<td>landbosse.critical_speed_non_erection_wind_delays_m_per_s</td>
<td>m/s</td>
<td>Non-Erection Wind Delay Critical Speed (m/s)</td>
</tr>
<tr>
<td>landbosse.decommissioning_pct</td>
<td></td>
<td></td>
</tr>
<tr>
<td>landbosse.depth</td>
<td>m</td>
<td>Foundation depth mdepth of foundation in the soil</td>
</tr>
<tr>
<td>landbosse.development_labor_cost_usd</td>
<td>USD</td>
<td>The cost of labor in the development phase</td>
</tr>
<tr>
<td>landbosse.distance_to_interconnect</td>
<td>mi</td>
<td>Distance to interconnect (miles)</td>
</tr>
<tr>
<td>landbosse.equip</td>
<td>Unavailable</td>
<td>Collections of equipment to perform erection operations.</td>
</tr>
<tr>
<td>landbosse.equip_price</td>
<td>Unavailable</td>
<td>Prices for various type of equipment.</td>
</tr>
<tr>
<td>landbosse.erection_component_name_topvbase</td>
<td>Unavailable</td>
<td>List of components and whether they are a topping or base operation</td>
</tr>
<tr>
<td>landbosse.erection_components</td>
<td>Unavailable</td>
<td>List of components with their values modified from the defaults.</td>
</tr>
<tr>
<td>landbosse.erection_crane_choice</td>
<td>Unavailable</td>
<td>The crane choices for erection.</td>
</tr>
<tr>
<td>landbosse.foundation_height</td>
<td>m</td>
<td>starting height of tower</td>
</tr>
<tr>
<td>landbosse.fraction_new_roads</td>
<td></td>
<td>Percent of roads that will be constructed (0.0 - 1.0)</td>
</tr>
<tr>
<td>landbosse.fuel_cost_usd_per_gal</td>
<td></td>
<td>Fuel cost USD/gal</td>
</tr>
<tr>
<td>landbosse.gust_velocity_m_per_s</td>
<td>m/s</td>
<td>50-year Gust Velocity (m/s)</td>
</tr>
<tr>
<td>landbosse.hour_day</td>
<td>Unavailable</td>
<td>Dictionary of normal and long hours for construction in a day in the form of {'long': 24, 'normal': 10}</td>
</tr>
</tbody>
</table>
Table 5.2 – continued from previous page

<table>
<thead>
<tr>
<th>Variable</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>landbosse.hub_height_meters</td>
<td>m</td>
<td>Hub height m</td>
</tr>
<tr>
<td>landbosse.hub_mass</td>
<td>kg</td>
<td>Mass of the rotor hubTotal mass of the hub shell, including the flanges</td>
</tr>
<tr>
<td>landbosse.installation_capex</td>
<td>USD</td>
<td>Total foundation and erection installation cost.</td>
</tr>
<tr>
<td>landbosse.installation_capex_kW</td>
<td>USD</td>
<td>Total foundation and erection installation cost per kW.</td>
</tr>
<tr>
<td>landbosse.installation_time_months</td>
<td></td>
<td>Total balance of system installation time (months).</td>
</tr>
<tr>
<td>landbosse.interconnect_voltage_kV</td>
<td>kV</td>
<td>Interconnect Voltage (kV)</td>
</tr>
<tr>
<td>landbosse.labor_cost_multiplier</td>
<td></td>
<td>Labor cost multiplier</td>
</tr>
<tr>
<td>landbosse.landbosse_costs_by_module_type_operation</td>
<td>Unavailable</td>
<td>The costs by module, type and operation</td>
</tr>
<tr>
<td>landbosse.landbosse_details_by_module</td>
<td>Unavailable</td>
<td>The details from the run of LandBOSSE. This includes some costs, but mostly other things</td>
</tr>
<tr>
<td>landbosse.line_frequency_hz</td>
<td>Hz</td>
<td>Line Frequency (Hz)</td>
</tr>
<tr>
<td>landbosse.markup_contingency</td>
<td></td>
<td>Markup contingency</td>
</tr>
<tr>
<td>landbosse.markup_overhead</td>
<td></td>
<td>Markup overhead</td>
</tr>
<tr>
<td>landbosse.markup_profit_margin</td>
<td></td>
<td>Markup profit margin</td>
</tr>
<tr>
<td>landbosse.markup_sales_and_use_tax</td>
<td></td>
<td>Markup sales and use tax</td>
</tr>
<tr>
<td>landbosse.markup_warranty_management</td>
<td></td>
<td>Markup warranty management</td>
</tr>
<tr>
<td>landbosse.material_price</td>
<td>Unavailable</td>
<td>Prices of materials for foundations and roads</td>
</tr>
<tr>
<td>landbosse.nacelle_mass</td>
<td>kg</td>
<td>Mass of nacelle system</td>
</tr>
<tr>
<td>landbosse.new_switchyard</td>
<td>Unavailable</td>
<td>New Switchyard (True or False)</td>
</tr>
<tr>
<td>landbosse.num_access_roads</td>
<td>Unavailable</td>
<td>Number of access roads</td>
</tr>
<tr>
<td>landbosse.num_hwy_permits</td>
<td>Unavailable</td>
<td>Number of highway permits</td>
</tr>
<tr>
<td>landbosse.num_turbines</td>
<td>Unavailable</td>
<td>Number of turbines in projectNumber of turbines that need scouring protection.</td>
</tr>
<tr>
<td>landbosse.number_of_blades</td>
<td>Unavailable</td>
<td>Number of blades on the rotor</td>
</tr>
<tr>
<td>landbosse.overtime_multiplier</td>
<td></td>
<td>Overtime multiplier</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Variable</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>landbosse.project_data</td>
<td>Unavailable</td>
<td>Dictionary of all dataframes of data</td>
</tr>
<tr>
<td>landbosse.rate_of_deliveries</td>
<td>Unavailable</td>
<td>Rate of deliveries (turbines per week)</td>
</tr>
<tr>
<td>landbosse.rated_thrust_N</td>
<td>N</td>
<td>Rated Thrust (N)</td>
</tr>
<tr>
<td>landbosse.road_distributed_wind</td>
<td>Unavailable</td>
<td></td>
</tr>
<tr>
<td>landbosse.road_length_adder_m</td>
<td>m</td>
<td>Road length adder (m)</td>
</tr>
<tr>
<td>landbosse.road_quality</td>
<td></td>
<td>Road Quality (0-1)</td>
</tr>
<tr>
<td>landbosse.road_thickness</td>
<td></td>
<td>Road thickness (in)</td>
</tr>
<tr>
<td>landbosse.road_width_ft</td>
<td>ft</td>
<td>Road width (ft)</td>
</tr>
<tr>
<td>landbosse.rotor_diameter_m</td>
<td>m</td>
<td>Rotor diameter m</td>
</tr>
<tr>
<td>landbosse.row_spacing_rotor_diameters</td>
<td></td>
<td>Row spacing (times rotor diameter)</td>
</tr>
<tr>
<td>landbosse.rsmeans</td>
<td>Unavailable</td>
<td>RSMeans price data</td>
</tr>
<tr>
<td>landbosse.site_facility_building_area_df</td>
<td>Unavailable</td>
<td>site_facility_building_area DataFrame</td>
</tr>
<tr>
<td>landbosse.time_construct</td>
<td>Unavailable</td>
<td>One of the keys in the hour_day dictionary to specify how many hours per day construction happens.</td>
</tr>
<tr>
<td>landbosse.total_capex</td>
<td>USD</td>
<td>Total BOS CAPEX including commissioning and decommissioning.</td>
</tr>
<tr>
<td>landbosse.total_capex_kW</td>
<td>USD/kW</td>
<td>Total BOS CAPEX per kW including commissioning and decommissioning.</td>
</tr>
<tr>
<td>landbosse.tower_mass</td>
<td>kg</td>
<td>Mass of tower</td>
</tr>
<tr>
<td>landbosse.tower_section_length_m</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>landbosse.trench_len_to_substation_km</td>
<td>km</td>
<td>Combined Homerun Trench Length to Substation (km)</td>
</tr>
<tr>
<td>landbosse.turbine_rating_MW</td>
<td>MW</td>
<td>Turbine rating MW</td>
</tr>
<tr>
<td>landbosse.turbine_spacing_rotor_diameters</td>
<td></td>
<td>Turbine spacing (times rotor diameter)</td>
</tr>
<tr>
<td>landbosse.user_defined_distance_to_grid_connection</td>
<td>Unavailable</td>
<td>Flag for user-defined home run trench length (True or False)</td>
</tr>
<tr>
<td>landbosse.user_defined_home_run_trench</td>
<td>Unavailable</td>
<td>Flag for user-defined home run trench length (0 = no; 1 = yes)</td>
</tr>
<tr>
<td>landbosse.weather_window</td>
<td>Unavailable</td>
<td>Dataframe of wind toolkit data</td>
</tr>
<tr>
<td>landbosse.wind_shear_exponent</td>
<td></td>
<td>Wind shear exponent</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Variable</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>materials.E</td>
<td>Pa</td>
<td>2D array of the Youngs moduli of the materials. Each row represents a material, the three columns represent E11, E22 and E33. Isotropic Youngs modulus of the materials along the column sections.</td>
</tr>
<tr>
<td>materials.G</td>
<td>Pa</td>
<td>2D array of the shear moduli of the materials. Each row represents a material, the three columns represent G12, G13 and G23. Isotropic shear modulus of the materials along the column sections.</td>
</tr>
<tr>
<td>materials.Xc</td>
<td>Pa</td>
<td>2D array of the Ultimate Compressive Strength (UCS) of the materials. Each row represents a material, the three columns represent Xc12, Xc13 and Xc23.</td>
</tr>
<tr>
<td>materials.Xt</td>
<td>Pa</td>
<td>2D array of the Ultimate Tensile Strength (UTS) of the materials. Each row represents a material, the three columns represent Xt12, Xt13 and Xt23.</td>
</tr>
<tr>
<td>materials.component_id</td>
<td>Unavail</td>
<td>1D array of flags to set whether a material is used in a blade: 0 - coating, 1 - sandwich filler, 2 - shell skin, 3 - shear webs, 4 - spar caps, 5 - TE reinf. isotropic.</td>
</tr>
<tr>
<td>materials.fvf</td>
<td></td>
<td>1D array of the non-dimensional fiber volume fraction of the composite materials. Non-composite materials are kept at 0.</td>
</tr>
<tr>
<td>materials.fvf_from_yaml</td>
<td></td>
<td>1D array of the non-dimensional fiber volume fraction of the composite materials. Non-composite materials are kept at 0.</td>
</tr>
<tr>
<td>materials.fwf</td>
<td></td>
<td>1D array of the non-dimensional fiber weight-fraction of the composite materials. Non-composite materials are kept at 0.</td>
</tr>
<tr>
<td>materials.fwf_from_yaml</td>
<td></td>
<td>1D array of the non-dimensional fiber weight-fraction of the composite materials. Non-composite materials are kept at 0.</td>
</tr>
<tr>
<td>materials.name</td>
<td>Unavail</td>
<td>1D array of names of materials.</td>
</tr>
<tr>
<td>materials.nu</td>
<td></td>
<td>2D array of the Poisson ratio of the materials. Each row represents a material, the three columns represent nu12, nu13 and nu23. poisson's ratio of column material</td>
</tr>
<tr>
<td>materials.orth</td>
<td>Unavail</td>
<td>1D array of flags to set whether a material is isotropic (0) or orthotropic (1). Each entry represents a material.</td>
</tr>
<tr>
<td>materials.ply_t</td>
<td>m</td>
<td>1D array of the ply thicknesses of the materials. Non-composite materials are kept at 0.</td>
</tr>
<tr>
<td>materials.ply_t_from_yaml</td>
<td>m</td>
<td>1D array of the ply thicknesses of the materials. Non-composite materials are kept at 0.</td>
</tr>
<tr>
<td>materials.rho</td>
<td>kg/m**3</td>
<td>3D array of the density of the materials. For composites, this is the density of the laminate. Density of the materials along the column sections.</td>
</tr>
<tr>
<td>materials.rho_area_dry</td>
<td>kg/m**2</td>
<td>2D array of the dry aerial density of the composite fabrics. Non-composite materials are kept at 0.</td>
</tr>
<tr>
<td>materials.rho_fiber</td>
<td>kg/m**3</td>
<td>3D array of the density of the fibers of the materials.</td>
</tr>
<tr>
<td>materials.roll_mass</td>
<td>kg</td>
<td>1D array of the roll mass of the composite fabrics. Non-composite materials are kept at 0.</td>
</tr>
<tr>
<td>materials.sigma_y</td>
<td>Pa</td>
<td>Yield stress of the material (in the principle direction for composites). Isotropic yield strength of the materials along the column sections.</td>
</tr>
<tr>
<td>materials.unit_cost</td>
<td>USD/kg</td>
<td>1D array of the unit costs of the materials. Unit costs of the materials along the column sections.</td>
</tr>
<tr>
<td>materials.waste</td>
<td></td>
<td>1D array of the non-dimensional waste fraction of the materials.</td>
</tr>
<tr>
<td>monopile.diameter</td>
<td>m</td>
<td>1D array of the outer diameter values defined along the tower axis. Cylinder diameter at corresponding locations</td>
</tr>
<tr>
<td>monopile.foundation_height</td>
<td>m</td>
<td>Foundation height in respect to the ground level. Starting height of tower.</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Variable</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>monopile.gravity_foundation_mass</td>
<td>kg</td>
<td>extra mass of gravity foundation point mass of transition piece</td>
</tr>
<tr>
<td>monopile.height</td>
<td>m</td>
<td>Scalar of the tower height computed along the z axis. Scalar of the column height computed along the z axis.</td>
</tr>
<tr>
<td>monopile.layer_mat</td>
<td>Unavail.</td>
<td>1D array of the names of the materials of each layer modeled in the tower structure.</td>
</tr>
<tr>
<td>monopile.layer_name</td>
<td>Unavail.</td>
<td>1D array of the names of the layers modeled in the tower structure.</td>
</tr>
<tr>
<td>monopile.layer_thickness</td>
<td>m</td>
<td>2D array of the thickness of the layers of the tower structure. The first dimension represents each layer, the second dimension represents each piecewise-constant entry of the tower sections. 2D array of the thickness of the layers of the column structure. The first dimension represents each layer, the second dimension represents each piecewise-constant entry of the column sections.</td>
</tr>
<tr>
<td>monopile.length</td>
<td>m</td>
<td>Scalar of the tower length computed along its curved axis. A standard straight tower will be as high as long. Length of a single cable connecting the OSS to the interconnection in km.</td>
</tr>
<tr>
<td>monopile.outfitting_factor</td>
<td></td>
<td>Multiplier that accounts for secondary structure mass inside of tower Mass fraction added for outfitting</td>
</tr>
<tr>
<td>monopile.ref_axis</td>
<td>m</td>
<td>2D array of the coordinates (x,y,z) of the tower reference axis. The coordinate system is the global coordinate system of OpenFAST: it is placed at tower base with x pointing downwind, y pointing on the side and z pointing vertically upwards. A standard tower configuration will have zero x and y values and positive z values.</td>
</tr>
<tr>
<td>monopile.s</td>
<td></td>
<td>1D array of the non-dimensional grid defined along the tower axis (0-tower base, 1-tower top) 1D array of the non-dimensional grid defined along the column axis (0-column base, 1-column top)</td>
</tr>
<tr>
<td>monopile.transition_piece_cost</td>
<td>USD</td>
<td>cost of transition piece Cost of transition piece</td>
</tr>
<tr>
<td>monopile.transition_piece_mass</td>
<td>kg</td>
<td>point mass of transition piece point mass of transition piece</td>
</tr>
<tr>
<td>nacelle.L_generator</td>
<td>m</td>
<td>Generator length along shaft Generator stack width</td>
</tr>
<tr>
<td>nacelle.bedplate_flange_thickness</td>
<td>m</td>
<td>Bedplate I-beam flange thickness Bedplate is two parallel I beams, this is the flange thickness</td>
</tr>
<tr>
<td>nacelle.bedplate_flange_width</td>
<td>m</td>
<td>Bedplate I-beam flange width Bedplate is two parallel I beams, this is the flange width</td>
</tr>
<tr>
<td>nacelle.bedplate_material</td>
<td>Unavail.</td>
<td>Material name identifier for the bedplate</td>
</tr>
<tr>
<td>nacelle.bedplate_wall_thickness</td>
<td>m</td>
<td>Thickness of hollow elliptical bedplate Bedplate wall thickness</td>
</tr>
<tr>
<td>nacelle.bedplate_web_thickness</td>
<td>m</td>
<td>Bedplate I-beam web thickness Bedplate is two parallel I beams, this is the web thickness</td>
</tr>
<tr>
<td>nacelle.brake_mass_user</td>
<td>kg</td>
<td>Override regular regression-based calculation of brake mass with this value User override of brake mass</td>
</tr>
<tr>
<td>nacelle.converter_mass_user</td>
<td>kg</td>
<td>Override regular regression-based calculation of converter mass with this value Override regular regression-based calculation of converter mass with this value</td>
</tr>
<tr>
<td>nacelle.distance_hub2mb</td>
<td>m</td>
<td>Distance from hub flange to first main bearing along shaft</td>
</tr>
<tr>
<td>nacelle.distance_mb2mb</td>
<td>m</td>
<td>Distance from first to second main bearing along shaft</td>
</tr>
<tr>
<td>nacelle.distance_tt_hub</td>
<td>m</td>
<td>Vertical distance from tower top plane to hub flange</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Variable</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nacelle.gear_configuration</td>
<td>Unavailable</td>
<td>3-letter string of Es or Ps to denote epicyclic or parallel gear configuration</td>
</tr>
<tr>
<td>nacelle.gear_ratio</td>
<td></td>
<td>Total gear ratio of drivetrain (use 1.0 for direct) overall gearbox ratio</td>
</tr>
<tr>
<td>nacelle.gearbox_efficiency</td>
<td></td>
<td>Efficiency of the gearbox. Set to 1.0 for direct-drive</td>
</tr>
<tr>
<td>nacelle.hss_diameter</td>
<td>m</td>
<td>Diameter of high speed shaft Lss discretized diameter values at coordinates</td>
</tr>
<tr>
<td>nacelle.hss_length</td>
<td>m</td>
<td>Length of high speed shaft</td>
</tr>
<tr>
<td>nacelle.hss_material</td>
<td>Unavailable</td>
<td>Material name identifier for the high speed shaft</td>
</tr>
<tr>
<td>nacelle.hss_wall_thickness</td>
<td>m</td>
<td>Wall thickness of high speed shaft Lss discretized thickness values at coordinates</td>
</tr>
<tr>
<td>nacelle.hvac_mass_coeff</td>
<td>kg/kW</td>
<td>Regression-based scaling coefficient on machine rating to get HVAC system mass</td>
</tr>
<tr>
<td>nacelle.lss_diameter</td>
<td>m</td>
<td>Diameter of low speed shaft LSS outer diameter from hub to bearing 2</td>
</tr>
<tr>
<td>nacelle.lss_material</td>
<td>Unavailable</td>
<td>Material name identifier for the low speed shaft</td>
</tr>
<tr>
<td>nacelle.lss_wall_thickness</td>
<td>m</td>
<td>Thickness of low speed shaft LSS wall thickness</td>
</tr>
<tr>
<td>nacelle.mb1Type</td>
<td>Unavailable</td>
<td>Type of main bearing: CARB / CRB / SRB / TRB</td>
</tr>
<tr>
<td>nacelle.mb2Type</td>
<td>Unavailable</td>
<td>Type of main bearing: CARB / CRB / SRB / TRB</td>
</tr>
<tr>
<td>nacelle.nose_diameter</td>
<td>m</td>
<td>Diameter of nose (also called turret or spindle) Nose outer diameter from bearing 1 to bedplate</td>
</tr>
<tr>
<td>nacelle.nose_wall_thickness</td>
<td>m</td>
<td>Thickness of nose (also called turret or spindle) Nose wall thickness</td>
</tr>
<tr>
<td>nacelle.overhang</td>
<td>m</td>
<td>Horizontal distance from tower top edge to hub flange Horizontal distance between hub and tower-top axis</td>
</tr>
<tr>
<td>nacelle.planet_numbers</td>
<td>Unavailable</td>
<td>Number of planets for epicyclic stages (use 0 for parallel)</td>
</tr>
<tr>
<td>nacelle.transformer_mass_user</td>
<td>kg</td>
<td>Override regular regression-based calculation of transformer mass with this value Override regular regression-based calculation of transformer mass with this value</td>
</tr>
<tr>
<td>nacelle.uptilt</td>
<td>rad</td>
<td>Nacelle uptilt angle. A standard machine has positive values.</td>
</tr>
<tr>
<td>nacelle.uptower</td>
<td>Unavailable</td>
<td>If power electronics are located uptower (True) or at tower base (False) Power electronics are placed in the nacelle at the tower top</td>
</tr>
<tr>
<td>orbit.anchor_mass</td>
<td>kg</td>
<td>Total mass of an anchor</td>
</tr>
<tr>
<td>orbit.anchor_type</td>
<td>Unavailable</td>
<td>Number of mooring lines per platform SUCTIONPILE or DRAGEMBEDMENT</td>
</tr>
<tr>
<td>orbit.blade_deck_space</td>
<td>m**2</td>
<td>Deck space required to transport a blade. Defaults to 0 in order to not be a constraint on installation.</td>
</tr>
<tr>
<td>orbit.blade_mass</td>
<td>t</td>
<td>mass of an individual blade. Total mass of one blade</td>
</tr>
<tr>
<td>orbit.boem_review_cost</td>
<td>USD</td>
<td>Cost for additional review by U.S. Dept of Interior Bureau of Ocean Energy Management (BOEM)</td>
</tr>
<tr>
<td>Variable</td>
<td>Units</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>orbit.bos_capex</td>
<td>USD</td>
<td>Total BOS CAPEX not including commissioning or decommissioning.</td>
</tr>
<tr>
<td>orbit.commissioning_pct</td>
<td></td>
<td>Commissioning percent.</td>
</tr>
<tr>
<td>orbit.construction_operations_plan_cost</td>
<td>USD</td>
<td>Cost to do construction planning.</td>
</tr>
<tr>
<td>orbit.decommissioning_pct</td>
<td>USD</td>
<td>Decommissioning percent.</td>
</tr>
<tr>
<td>orbit.design_install_plan_cost</td>
<td></td>
<td>Cost to do installation planning.</td>
</tr>
<tr>
<td>orbit.feeder</td>
<td>Unavail</td>
<td>Vessel configuration to use for (optional) feeder barges.</td>
</tr>
<tr>
<td>orbit.hub_height</td>
<td>m</td>
<td>Turbine hub height. Hub height of wind turbine above ground / sea level.</td>
</tr>
<tr>
<td>orbit.installation_capex</td>
<td>USD</td>
<td>Total balance of system installation cost.</td>
</tr>
<tr>
<td>orbit.installation_time</td>
<td>h</td>
<td>Total balance of system installation time.</td>
</tr>
<tr>
<td>orbit.interconnection_distance</td>
<td>km</td>
<td>Distance from landfall to interconnection.</td>
</tr>
<tr>
<td>orbit.monopile_deck_space</td>
<td>m**2</td>
<td>Deck space required to transport a monopile. Defaults to 0 in order to not</td>
</tr>
<tr>
<td></td>
<td></td>
<td>be a constraint on installation.</td>
</tr>
<tr>
<td>orbit.monopile_diameter</td>
<td>m</td>
<td>Diameter of monopile.</td>
</tr>
<tr>
<td>orbit.monopile_length</td>
<td>m</td>
<td>Length of monopile.</td>
</tr>
<tr>
<td>orbit.monopile_mass</td>
<td>t</td>
<td>mass of an individual monopile. Monopile mass</td>
</tr>
<tr>
<td>orbit.mooring_line_diameter</td>
<td>m</td>
<td>Cross-sectional diameter of a mooring line</td>
</tr>
<tr>
<td>orbit.mooring_line_length</td>
<td>m</td>
<td>Unstretched mooring line length.</td>
</tr>
<tr>
<td>orbit.mooring_line_mass</td>
<td>kg</td>
<td>Total mass of a mooring line</td>
</tr>
<tr>
<td>orbit.nacelle_deck_space</td>
<td>m**2</td>
<td>Deck space required to transport the rotor nacelle assembly (RNA). Defaults</td>
</tr>
<tr>
<td></td>
<td></td>
<td>to 0 in order to not be a constraint on installation.</td>
</tr>
<tr>
<td>orbit.nacelle_mass</td>
<td>t</td>
<td>mass of the rotor nacelle assembly (RNA). Mass of nacelle system</td>
</tr>
<tr>
<td>orbit.num_assembly_lines</td>
<td>Unavail</td>
<td>Number of assembly lines used when assembly occurs at the port.</td>
</tr>
<tr>
<td>orbit.num_feeders</td>
<td>Unavail</td>
<td>Number of feeder barges to use for installation of foundations and turbines.</td>
</tr>
<tr>
<td>orbit.num_mooring_lines</td>
<td>Unavail</td>
<td>Number of mooring lines per platform.</td>
</tr>
<tr>
<td>orbit.num_port_cranes</td>
<td>Unavail</td>
<td>Number of cranes used at the port to load feeders / WTVIs when assembly</td>
</tr>
<tr>
<td></td>
<td></td>
<td>occurs on-site or assembly cranes when assembling at port.</td>
</tr>
<tr>
<td>orbit.num_station_keeping</td>
<td>Unavail</td>
<td>Number of station keeping vessels that attach to floating platforms under</td>
</tr>
<tr>
<td></td>
<td></td>
<td>tow-out.</td>
</tr>
<tr>
<td>orbit.num_towing</td>
<td>Unavail</td>
<td>Number of towing vessels to use for floating platforms that are assembled</td>
</tr>
<tr>
<td></td>
<td></td>
<td>at port (with or without the turbine).</td>
</tr>
<tr>
<td>orbit.number_of_blades</td>
<td>Unavail</td>
<td>Number of blades per turbine.</td>
</tr>
<tr>
<td>orbit.number_of_turbines</td>
<td>Unavail</td>
<td>Number of turbines.</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Variable</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>orbit.oss_install_vessel</td>
<td>Unavailable</td>
<td>Vessel configuration to use for installation of offshore substations.</td>
</tr>
<tr>
<td>orbit.plant_row_spacing</td>
<td>km</td>
<td>Row spacing in rotor diameters. Not used in ring layouts.</td>
</tr>
<tr>
<td>orbit.plant_substation_distance</td>
<td>km</td>
<td>Distance from first turbine in string to substation.</td>
</tr>
<tr>
<td>orbit.plant_turbine_spacing</td>
<td>m</td>
<td>Turbine spacing in rotor diameters.</td>
</tr>
<tr>
<td>orbit.port_cost_per_month</td>
<td>USD/mo</td>
<td>Monthly port costs.</td>
</tr>
<tr>
<td>orbit.site_assessment_cost</td>
<td>USD</td>
<td>Cost to execute site assessment.</td>
</tr>
<tr>
<td>orbit.site_assessment_plan_cost</td>
<td>USD</td>
<td>Cost to do engineering plan for site assessment.</td>
</tr>
<tr>
<td>orbit.site_auction_price</td>
<td>USD</td>
<td>Cost to secure site lease.</td>
</tr>
<tr>
<td>orbit.site_depth</td>
<td>m</td>
<td>Site depth. Average depth at the site in km.</td>
</tr>
<tr>
<td>orbit.site_distance</td>
<td>km</td>
<td>Distance from site to installation port.</td>
</tr>
<tr>
<td>orbit.site_distance_to_landfall</td>
<td>km</td>
<td>Distance from site to landfall for export cable.</td>
</tr>
<tr>
<td>orbit.site_mean_windspeed</td>
<td>m/s</td>
<td>Mean windspeed of the site.</td>
</tr>
<tr>
<td>orbit.takt_time</td>
<td>h</td>
<td>Substructure assembly cycle time when doing assembly at the port.</td>
</tr>
<tr>
<td>orbit.total_capex</td>
<td>USD</td>
<td>Total BOS CAPEX including commissioning and decommissioning.</td>
</tr>
<tr>
<td>orbit.total_capex_kW</td>
<td>USD/kW</td>
<td>Total BOS CAPEX including commissioning and decommissioning.</td>
</tr>
<tr>
<td>orbit.tower_deck_space</td>
<td>m²</td>
<td>Deck space required to transport the tower. Defaults to 0 in order to not be a constraint on installation.</td>
</tr>
<tr>
<td>orbit.tower_length</td>
<td>m</td>
<td>Total length of the tower.</td>
</tr>
<tr>
<td>orbit.tower_mass</td>
<td>t</td>
<td>Mass of the total tower. Mass of tower.</td>
</tr>
<tr>
<td>orbit.transition_piece_deck_space</td>
<td>m²</td>
<td>Deck space required to transport a transition piece. Defaults to 0 in order to not be a constraint on installation.</td>
</tr>
<tr>
<td>orbit.transition_piece_mass</td>
<td>t</td>
<td>Mass of an individual transition piece. Mass of transition piece.</td>
</tr>
<tr>
<td>orbit.turbine_capex</td>
<td>USD/kW</td>
<td>Turbine CAPEX.</td>
</tr>
<tr>
<td>orbit.turbine_rated_windspeed</td>
<td>m/s</td>
<td>Rated windspeed of the turbine.</td>
</tr>
<tr>
<td>orbit.turbine_rating</td>
<td>MW</td>
<td>Rated capacity of a turbine. Capacity of an individual turbine in MW.</td>
</tr>
<tr>
<td>orbit.turbineRotor_diameter</td>
<td>m</td>
<td>Turbine rotor diameter.</td>
</tr>
<tr>
<td>orbit.wtiv</td>
<td>Unavailable</td>
<td>Vessel configuration to use for installation of foundations and turbines.</td>
</tr>
<tr>
<td>outputs_2_screen.Flp_omega</td>
<td>rad/s</td>
<td></td>
</tr>
<tr>
<td>outputs_2_screen.Flp_zeta</td>
<td>rad</td>
<td></td>
</tr>
<tr>
<td>outputs_2_screen.My_std</td>
<td>N*m</td>
<td>Total mass of one blade.</td>
</tr>
<tr>
<td>outputs_2_screen.PC_omega</td>
<td>rad/s</td>
<td></td>
</tr>
<tr>
<td>outputs_2_screen.PC_zeta</td>
<td>rad</td>
<td></td>
</tr>
<tr>
<td>outputs_2_screen.VS_omega</td>
<td>rad/s</td>
<td></td>
</tr>
<tr>
<td>outputs_2_screen.VS_zeta</td>
<td>rad</td>
<td></td>
</tr>
<tr>
<td>outputs_2_screen.aep</td>
<td>GW * h</td>
<td></td>
</tr>
<tr>
<td>outputs_2_screen.blade_mass</td>
<td>kg</td>
<td></td>
</tr>
<tr>
<td>outputs_2_screen.flp1_std</td>
<td>deg</td>
<td></td>
</tr>
<tr>
<td>outputs_2_screen.lcoe</td>
<td>USD/MW</td>
<td></td>
</tr>
<tr>
<td>outputs_2_screen.tip_deflection</td>
<td>m</td>
<td>Blade tip deflection in yaw x-direction.</td>
</tr>
</tbody>
</table>

*continues on next page*
<table>
<thead>
<tr>
<th>Variable</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>re.A</td>
<td>m²</td>
<td>airfoil cross section material areamagnitude of wave acceleration</td>
</tr>
<tr>
<td>re.EA</td>
<td>N had</td>
<td>axial stiffness</td>
</tr>
<tr>
<td>re.Elxx</td>
<td>N*m²</td>
<td>2gewise stiffness (bending about x-axis of airfoil aligned coordinate system)</td>
</tr>
<tr>
<td>re.Elxy</td>
<td>N*m²</td>
<td>2fipped flap-edge stiffness</td>
</tr>
<tr>
<td>re.EIyy</td>
<td>N*m²</td>
<td>Flatwise stiffness (bending about y-axis of airfoil aligned coordinate system)</td>
</tr>
<tr>
<td>re.GJ</td>
<td>N*m²</td>
<td>Borsional stiffness (about axial z-axis of airfoil aligned coordinate system)</td>
</tr>
<tr>
<td>re.Tw_iner</td>
<td>m</td>
<td>Orientation of the section principal inertia axes with respect the blade reference plane</td>
</tr>
<tr>
<td>re.chord</td>
<td>m</td>
<td>Chord length at each section</td>
</tr>
<tr>
<td>re.precomp.E</td>
<td>Pa</td>
<td>2D array of the Youngs moduli of the materials. Each row represents a material, the three columns represent E11, E22 and E33. Isotropic Youngs modulus of the materials along the column sections.</td>
</tr>
<tr>
<td>re.precomp.G</td>
<td>Pa</td>
<td>2D array of the shear moduli of the materials. Each row represents a material, the three columns represent G12, G13 and G23. Isotropic shear modulus of the materials along the column sections.</td>
</tr>
<tr>
<td>re.precomp.I_all_blades</td>
<td>kg*m³</td>
<td>Mass moments of inertia of all blades in yaw c.s. order: Ixx, Iyy, Izz, Ixy, Ixz, Iyz</td>
</tr>
<tr>
<td>re.precomp.blade_mass</td>
<td>kg</td>
<td>Total mass of one blade</td>
</tr>
<tr>
<td>re.precomp.blade_moment_of_inertia</td>
<td>kg/m</td>
<td>Mass moment of inertia of blade about hub</td>
</tr>
<tr>
<td>re.precomp.component_id</td>
<td>Unavailable</td>
<td>1D array of flags to set whether a material is used in a blade: 0 - coating, 1 - sandwich filler, 2 - shell skin, 3 - shear webs, 4 - spar caps, 5 - TE reinf.isotropic.</td>
</tr>
<tr>
<td>re.precomp.coord_xy_interp</td>
<td>Unavailable</td>
<td>3D array of the non-dimensional x and y airfoil coordinates of the airfoils interpolated along span for n_span stations.</td>
</tr>
<tr>
<td>re.precomp.definition_layer</td>
<td>Unavailable</td>
<td>1D array of flags identifying how layers are specified in the yaml. 1) all around (skin, paint, ) 2) offset+rotation twist+width (spar caps) 3) offset+user defined rotation+width 4) midpoint TE+width (TE reinf) 5) midpoint LE+width (LE reinf) 6) layer position fixed to other layer (core fillers) 7) start and width 8) end and width 9) start and end nd 10) web layer</td>
</tr>
<tr>
<td>re.precomp.edge_iner</td>
<td>kg/m</td>
<td>Section lag inertia about the X_G axis per unit length</td>
</tr>
<tr>
<td>re.precomp.fiber_orientation</td>
<td>deg</td>
<td>2D array of the orientation of the layers of the blade structure. The first dimension represents each layer, the second dimension represents each entry along blade span.</td>
</tr>
<tr>
<td>re.precomp.flap_iner</td>
<td>kg/m</td>
<td>Section flap inertia about the Y_G axis per unit length.</td>
</tr>
<tr>
<td>re.precomp.fvf</td>
<td>1D array of the non-dimensional fiber volume fraction of the composite materials. Non-composite materials are kept at 0.</td>
<td></td>
</tr>
<tr>
<td>re.precomp.fwf</td>
<td>1D array of the non-dimensional fiber weight- fraction of the composite materials. Non-composite materials are kept at 0.</td>
<td></td>
</tr>
<tr>
<td>re.precomp.layer_end_nd</td>
<td>2D array of the non-dimensional end point defined along the outer profile of a layer. The TE suction side is 0, the TE pressure side is 1. The first dimension represents each layer, the second dimension represents each entry along blade span.</td>
<td></td>
</tr>
</tbody>
</table>
| re.precomp.layer_start_nd | 2D array of the non-dimensional start point defined along the outer profile of a layer. The TE suction side is 0, the TE pressure side is 1. The first dimension represents each layer, the second dimension represents each entry along blade span. | continues on next page
<table>
<thead>
<tr>
<th>Variable</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>re.precomp.layer_thickness</td>
<td>m</td>
<td>2D array of the thickness of the layers of the blade structure. The first dimension represents each layer, the second dimension represents each entry along blade span. 2D array of the thickness of the layers of the column structure. The first dimension represents each layer, the second dimension represents each piecewise-constant entry of the column sections.</td>
</tr>
<tr>
<td>re.precomp.layer_web</td>
<td></td>
<td>1D array of the web id the layer is associated to. If the layer is on the outer profile, this entry can simply stay equal to 0.</td>
</tr>
<tr>
<td>re.precomp.mass_all_blades</td>
<td>kg</td>
<td>mass of all blades</td>
</tr>
<tr>
<td>re.precomp.mat_name</td>
<td>Unavailable</td>
<td>1D array of names of materials.</td>
</tr>
<tr>
<td>re.precomp.n_blades</td>
<td>Unavailable</td>
<td>Number of blades of the rotor. Number of rotor blades</td>
</tr>
<tr>
<td>re.precomp.nu</td>
<td></td>
<td>2D array of the Poisson ratio of the materials. Each row represents a material, the three columns represent nu12, nu13 and nu23. poisons ratio of column material</td>
</tr>
<tr>
<td>re.precomp.orth</td>
<td>Unavailable</td>
<td>1D array of flags to set whether a material is isotropic (0) or orthotropic (1). Each entry represents a material.</td>
</tr>
<tr>
<td>re.precomp.pitch_axis</td>
<td></td>
<td>1D array of the chordwise position of the pitch axis (0-LE, 1-TE), defined along blade span.</td>
</tr>
<tr>
<td>re.precomp.ply_t</td>
<td>m</td>
<td>1D array of the ply thicknesses of the materials. Non-composite materials are kept at 0.</td>
</tr>
<tr>
<td>re.precomp.rho</td>
<td>kg/m**3</td>
<td>3D array of the density of the materials. For composites, this is the density of the laminate. Density of the materials along the column sections.</td>
</tr>
<tr>
<td>re.precomp.rho_area_dry</td>
<td>kg/m**2</td>
<td>2D array of the dry aerial density of the composite fabrics. Non-composite materials are kept at 0.</td>
</tr>
<tr>
<td>re.precomp.rho_fiber</td>
<td>kg/m**3</td>
<td>3D array of the density of the fibers of the materials.</td>
</tr>
<tr>
<td>re.precomp.roll_mass</td>
<td>kg</td>
<td>1D array of the roll mass of the composite fabrics. Non-composite materials are kept at 0.</td>
</tr>
<tr>
<td>re.precomp.total_blade_cost</td>
<td>USD</td>
<td>total blade cost</td>
</tr>
<tr>
<td>re.precomp.total_blade_mass</td>
<td>USD</td>
<td>total blade cost</td>
</tr>
<tr>
<td>re.precomp.unit_cost</td>
<td>USD/kg</td>
<td>1D array of the unit costs of the materials. Unit costs of the materials along the column sections.</td>
</tr>
<tr>
<td>re.precomp.uptilt</td>
<td>deg</td>
<td>Nacelle uptilt angle. A standard machine has positive values.</td>
</tr>
<tr>
<td>re.precomp.waste</td>
<td></td>
<td>1D array of the non-dimensional waste fraction of the materials.</td>
</tr>
<tr>
<td>re.precomp.web_end_nd</td>
<td></td>
<td>2D array of the non-dimensional end point defined along the outer profile of a web. The TE suction side is 0, the TE pressure side is 1. The first dimension represents each web, the second dimension represents each entry along blade span.</td>
</tr>
<tr>
<td>re.precomp.web_start_nd</td>
<td></td>
<td>2D array of the non-dimensional start point defined along the outer profile of a web. The TE suction side is 0, the TE pressure side is 1. The first dimension represents each web, the second dimension represents each entry along blade span.</td>
</tr>
<tr>
<td>re.precomp.x_cg</td>
<td>m</td>
<td>X-coordinate of the center-of-mass offset with respect to the XR-YR axes</td>
</tr>
<tr>
<td>re.precomp.x_tc</td>
<td>m</td>
<td>X-coordinate of the tension-center offset with respect to the XR-YR axes</td>
</tr>
<tr>
<td>re.precomp.xl_strain_spar</td>
<td></td>
<td>x-position of midpoint of spar cap on lower surface for strain calculation</td>
</tr>
<tr>
<td>re.precomp.xl_strain_te</td>
<td></td>
<td>x-position of midpoint of trailing-edge panel on lower surface for strain calculation</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Variable</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>re.precomp.xu_strain_spar</td>
<td></td>
<td>x-position of midpoint of spar cap on upper surface for strain calculation</td>
</tr>
<tr>
<td>re.precomp.xu_strain_te</td>
<td></td>
<td>x-position of midpoint of trailing-edge panel on upper surface for strain</td>
</tr>
<tr>
<td></td>
<td></td>
<td>calculation</td>
</tr>
<tr>
<td>re.precomp.y_cg</td>
<td>m</td>
<td>Chordwise offset of the section center of mass with respect to the XR-YR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>axes</td>
</tr>
<tr>
<td>re.precomp.y_tc</td>
<td>m</td>
<td>Chordwise offset of the section tension-center with respect to the XR-YR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>axes</td>
</tr>
<tr>
<td>re.precomp.yl_strain_spar</td>
<td></td>
<td>y-position of midpoint of spar cap on lower surface for strain calculation</td>
</tr>
<tr>
<td>re.precomp.yl_strain_te</td>
<td></td>
<td>y-position of midpoint of trailing-edge panel on lower surface for strain</td>
</tr>
<tr>
<td></td>
<td></td>
<td>calculation</td>
</tr>
<tr>
<td>re.precomp.yu_strain_spar</td>
<td></td>
<td>y-position of midpoint of spar cap on upper surface for strain calculation</td>
</tr>
<tr>
<td>re.precomp.yu_strain_te</td>
<td></td>
<td>y-position of midpoint of trailing-edge panel on upper surface for strain</td>
</tr>
<tr>
<td></td>
<td></td>
<td>calculation</td>
</tr>
<tr>
<td>re.precomp.z</td>
<td>m</td>
<td>locations of properties along beam location along cylinder. start at bottom</td>
</tr>
<tr>
<td></td>
<td></td>
<td>and go to top</td>
</tr>
<tr>
<td>re.precurve</td>
<td>m</td>
<td>precurve at each section</td>
</tr>
<tr>
<td>re.presweep</td>
<td>m</td>
<td>presweep at each section</td>
</tr>
<tr>
<td>re.r</td>
<td>m</td>
<td>radial locations where blade is defined (should be increasing and not go all</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the way to hub or tip)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Radial locations where blade is defined. Should be increasing and not go all</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the way to hub or tip.</td>
</tr>
<tr>
<td>re.rail.blade_ref_axis</td>
<td>m</td>
<td>2D array of the coordinates (x,y,z) of the blade reference axis, defined</td>
</tr>
<tr>
<td></td>
<td></td>
<td>along blade span. The coordinate system is the one of BeamDyn: it is placed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>at blade root with x pointing the suction side of the blade, y pointing the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>trailing edge and z along the blade span. A standard configuration will</td>
</tr>
<tr>
<td></td>
<td></td>
<td>have negative x values (prebend), if swept positive y values, and positive</td>
</tr>
<tr>
<td></td>
<td></td>
<td>z values</td>
</tr>
<tr>
<td>re.rail.constr_LV_4axle_horiz</td>
<td></td>
<td>Constraint for max L/V for a 4-axle flatcar on horiz curves, violated when</td>
</tr>
<tr>
<td></td>
<td></td>
<td>bigger than 1</td>
</tr>
<tr>
<td>re.rail.constr_LV_4axle_vert</td>
<td></td>
<td>Constraint for max L/V for a 4-axle flatcar on vert curves, violated when</td>
</tr>
<tr>
<td></td>
<td></td>
<td>bigger than 1</td>
</tr>
<tr>
<td>re.rail.constr_LV_8axle_horiz</td>
<td></td>
<td>Constraint for max L/V for an 8-axle flatcar on horiz curves, violated when</td>
</tr>
<tr>
<td></td>
<td></td>
<td>bigger than 1</td>
</tr>
<tr>
<td>re.rail.constr_LV_8axle_vert</td>
<td></td>
<td>Constraint for max L/V for an 8-axle flatcar on vert curves, violated when</td>
</tr>
<tr>
<td></td>
<td></td>
<td>bigger than 1</td>
</tr>
<tr>
<td>re.rail.constr_strainLE</td>
<td></td>
<td>Strain along leading edge side of blade on a vertical curve</td>
</tr>
<tr>
<td>re.rail.constr_strainPS</td>
<td></td>
<td>Strain along pressure side of blade on a horizontal curve</td>
</tr>
<tr>
<td>re.rail.constr_strainSS</td>
<td></td>
<td>Strain along suction side of blade on a horizontal curve</td>
</tr>
<tr>
<td>re.rail.constr_strainTE</td>
<td></td>
<td>Strain along trailing edge side of blade on a vertical curve</td>
</tr>
<tr>
<td>re.rail.coord_xy_dim</td>
<td>m</td>
<td>3D array of the dimensional x and y airfoil coordinates of the airfoils</td>
</tr>
<tr>
<td></td>
<td></td>
<td>interpolated along span for n_span stations. The origin is placed at the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>pitch axis</td>
</tr>
<tr>
<td>re.rail.coord_xy_interp</td>
<td></td>
<td>3D array of the non-dimensional x and y airfoil coordinates of the airfoils</td>
</tr>
<tr>
<td></td>
<td></td>
<td>interpolated along span for n_span stations. The leading edge is place at</td>
</tr>
<tr>
<td></td>
<td></td>
<td>x=0 and y=0</td>
</tr>
<tr>
<td>re.rail.deck_height</td>
<td>m</td>
<td>Height of the deck of the flatcar from the rails (4 feet)</td>
</tr>
<tr>
<td>re.rail.flatcar_tc_length</td>
<td>m</td>
<td>Flatcar truck center to truck center length</td>
</tr>
<tr>
<td>re.rail.horizontal_angle_deg</td>
<td>deg</td>
<td>Angle of horizontal turn (defined for an chord of 100 feet)</td>
</tr>
<tr>
<td>re.rail.lateral_clearance</td>
<td>m</td>
<td>Clearance profile horizontal (22 feet)</td>
</tr>
<tr>
<td>re.rail.max_LV</td>
<td></td>
<td>Max allowable ratio between lateral and vertical forces</td>
</tr>
<tr>
<td>re.rail.max_flatcar_weight_4ax</td>
<td>kg</td>
<td>Max mass of an 4-axle flatcar (286000 lbm)</td>
</tr>
<tr>
<td>Variable</td>
<td>Units</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>re.rail.max_top_car_weight_8axle</td>
<td>kg</td>
<td>Max mass of an 8-axle flatcar (480000 lbm)</td>
</tr>
<tr>
<td>re.rail.max_root_rot_deg</td>
<td>deg</td>
<td>Max degree of angle at blade root</td>
</tr>
<tr>
<td>re.rail.max_strains</td>
<td></td>
<td>Max allowable strains during transport</td>
</tr>
<tr>
<td>re.rail.min_vertical_radius</td>
<td>m</td>
<td>Minimum radius of a vertical curvature (hill or sag) (2000 feet)</td>
</tr>
<tr>
<td>re.rail.pitch_axis</td>
<td>m</td>
<td>1D array of the chordwise position of the pitch axis (0-LE, 1-TE), defined along blade span.</td>
</tr>
<tr>
<td>re.rail.vertical_clearance</td>
<td>m</td>
<td>Clearance profile vertical (23 feet)</td>
</tr>
<tr>
<td>re.rhoA</td>
<td>kg/m</td>
<td>mass per unit length</td>
</tr>
<tr>
<td>re.rhoJ</td>
<td>kg*m</td>
<td>polar mass moment of inertia per unit length</td>
</tr>
<tr>
<td>re.sc_ps_mats</td>
<td></td>
<td>spar cap, pressure side, boolean of materials in each composite layer spanwise, passed as floats for differentiability, used for Fatigue Analysis</td>
</tr>
<tr>
<td>re.sc_ss_mats</td>
<td></td>
<td>spar cap, suction side, boolean of materials in each composite layer spanwise, passed as floats for differentiability, used for Fatigue Analysis</td>
</tr>
<tr>
<td>re.te_ps_mats</td>
<td></td>
<td>trailing edge reinforcement, pressure side, boolean of materials in each composite layer spanwise, passed as floats for differentiability, used for Fatigue Analysis</td>
</tr>
<tr>
<td>re.te_ss_mats</td>
<td></td>
<td>trailing edge reinforcement, suction side, boolean of materials in each composite layer spanwise, passed as floats for differentiability, used for Fatigue Analysis</td>
</tr>
<tr>
<td>re.theta</td>
<td>deg</td>
<td>Twist angle at each section (positive decreases angle of attack) Twist angle at each section (positive decreases angle of attack).</td>
</tr>
<tr>
<td>re.x_ec</td>
<td>m</td>
<td>x-distance to elastic center from point about which above structural properties are computed (airfoil aligned coordinate system)</td>
</tr>
<tr>
<td>re.x_sc</td>
<td>m</td>
<td>X-coordinate of the shear-center offset with respect to the XR-YR axes</td>
</tr>
<tr>
<td>re.y_ec</td>
<td>m</td>
<td>y-distance to elastic center from point about which above structural properties are computed</td>
</tr>
<tr>
<td>re.y_sc</td>
<td>m</td>
<td>Chordwise offset of the section shear-center with respect to the reference frame, XR-YR</td>
</tr>
<tr>
<td>rp.AEP</td>
<td>kW*h</td>
<td>annual energy production</td>
</tr>
<tr>
<td>rp.Rhub</td>
<td>m</td>
<td>hub radius</td>
</tr>
<tr>
<td>rp.Rtip</td>
<td>m</td>
<td>tip radius</td>
</tr>
<tr>
<td>rp.aep.CDF_V</td>
<td>m/s</td>
<td>cumulative distribution function evaluated at each wind speed</td>
</tr>
<tr>
<td>rp.aep.P</td>
<td>W</td>
<td>power curve (power)</td>
</tr>
<tr>
<td>rp.aep.lossFactor</td>
<td></td>
<td>multiplicative factor for availability and other losses (soiling, array, etc.)</td>
</tr>
<tr>
<td>rp.airfoils_Re</td>
<td></td>
<td>Reynolds numbers of polars</td>
</tr>
<tr>
<td>rp.airfoils_aoa</td>
<td>deg</td>
<td>angle of attack grid for polars</td>
</tr>
<tr>
<td>rp.airfoils_cd</td>
<td></td>
<td>drag coefficients, spanwise</td>
</tr>
<tr>
<td>rp.airfoils_cm</td>
<td></td>
<td>lift coefficients, spanwise</td>
</tr>
<tr>
<td>rp.cdf.F</td>
<td>m/s</td>
<td>magnitude of wind speed at each z location</td>
</tr>
<tr>
<td>rp.cdf.k</td>
<td></td>
<td>shape or form factor</td>
</tr>
<tr>
<td>rp.cdf.x</td>
<td>m/s</td>
<td>corresponding reference height</td>
</tr>
<tr>
<td>rp.cdf.xbar</td>
<td>m</td>
<td>mean value of distribution</td>
</tr>
<tr>
<td>rp.chord</td>
<td>m</td>
<td>chord length at each section</td>
</tr>
<tr>
<td>rp.control_maxTS</td>
<td>m/s</td>
<td>maximum allowed blade tip speed</td>
</tr>
<tr>
<td>rp.control_pitch</td>
<td>deg</td>
<td>pitch angle in region 2 (and region 3 for fixed pitch machines)</td>
</tr>
<tr>
<td>rp.drivetrainType</td>
<td>Unavailable</td>
<td></td>
</tr>
<tr>
<td>rp.gust.V_gust</td>
<td>m/s</td>
<td>gust wind speed</td>
</tr>
</tbody>
</table>
Table 5.2 – continued from previous page

<table>
<thead>
<tr>
<th>Variable</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rp.gust.V_hub</td>
<td>m/s</td>
<td>hub height wind speed</td>
</tr>
<tr>
<td>rp.gust.V_mean</td>
<td>m/s</td>
<td>IEC average wind speed for turbine class</td>
</tr>
<tr>
<td>rp.gust.std</td>
<td></td>
<td>number of standard deviations for strength of gust</td>
</tr>
<tr>
<td>rp.gust.turbulence_class</td>
<td>Unavailable</td>
<td>IEC turbulence class</td>
</tr>
<tr>
<td>rp_hub_height</td>
<td>m</td>
<td>hub height of wind turbine above ground / sea level</td>
</tr>
<tr>
<td>rp.mu</td>
<td>kg/(m*s)</td>
<td>dynamic viscosity of air</td>
</tr>
<tr>
<td>rp.nBlades</td>
<td>Unavailable</td>
<td>number of blades</td>
</tr>
<tr>
<td>rp.omega_max</td>
<td>rpm</td>
<td>maximum allowed rotor rotation speed</td>
</tr>
<tr>
<td>rp.omega_min</td>
<td>rpm</td>
<td>minimum allowed rotor rotation speed</td>
</tr>
<tr>
<td>rp.powercurve.Cm_aero</td>
<td></td>
<td>rotor aerodynamic moment coefficient</td>
</tr>
<tr>
<td>rp.powercurve.Cp</td>
<td></td>
<td>rotor electrical power coefficient</td>
</tr>
<tr>
<td>rp.powercurve.Cp_aero</td>
<td></td>
<td>rotor aerodynamic power coefficient</td>
</tr>
<tr>
<td>rp.powercurve.Cp_regII</td>
<td></td>
<td>power coefficient at cut-in wind speed</td>
</tr>
<tr>
<td>rp.powercurve.Cq_aero</td>
<td></td>
<td>rotor aerodynamic torque coefficient</td>
</tr>
<tr>
<td>rp.powercurve.Ct_aero</td>
<td></td>
<td>rotor aerodynamic thrust coefficient</td>
</tr>
<tr>
<td>rp.powercurve.M</td>
<td>N*m</td>
<td>blade root moment</td>
</tr>
<tr>
<td>rp.powercurve.Omega</td>
<td>rpm</td>
<td>rotor rotational speed</td>
</tr>
<tr>
<td>rp.powercurve.Omega_spline</td>
<td>rpm</td>
<td>omega</td>
</tr>
<tr>
<td>rp.powercurve.P</td>
<td>W</td>
<td>rotor electrical power</td>
</tr>
<tr>
<td>rp.powercurve.P_aero</td>
<td>W</td>
<td>rotor mechanical power</td>
</tr>
<tr>
<td>rp.powercurve.P_spline</td>
<td>W</td>
<td>rotor electrical power</td>
</tr>
<tr>
<td>rp.powercurve.Q</td>
<td>N*m</td>
<td>rotor aerodynamic torque</td>
</tr>
<tr>
<td>rp.powercurve.T</td>
<td>N</td>
<td>rotor aerodynamic thrust</td>
</tr>
<tr>
<td>rp.powercurve.V</td>
<td>m/s</td>
<td>wind vector</td>
</tr>
<tr>
<td>rp.powercurve.V_R25</td>
<td>m/s</td>
<td>region 2.5 transition wind speed</td>
</tr>
<tr>
<td>rp.powercurve.V_spline</td>
<td>m/s</td>
<td>wind vector</td>
</tr>
<tr>
<td>rp.powercurve.aoa_regII</td>
<td>deg</td>
<td>angle of attack distribution along blade span at cut-in wind speed</td>
</tr>
<tr>
<td>rp.powercurve.ax_induct_regII</td>
<td></td>
<td>rotor axial induction at cut-in wind speed along blade span</td>
</tr>
<tr>
<td>rp.powercurve.cd_regII</td>
<td></td>
<td>drag coefficient distribution along blade span at cut-in wind speed</td>
</tr>
<tr>
<td>rp.powercurve.ci_regII</td>
<td></td>
<td>lift coefficient distribution along blade span at cut-in wind speed</td>
</tr>
<tr>
<td>rp.powercurve.gearbox_efficiency</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rp.powercurve.generator_efficiency</td>
<td></td>
<td>Generator efficiency at various rpm values to support table lookup</td>
</tr>
<tr>
<td>rp.powercurve.hubloss</td>
<td>Unavailable</td>
<td>include Prandtl hub loss model</td>
</tr>
<tr>
<td>rp.powercurve.lss_rpm</td>
<td>rpm</td>
<td>Low speed shaft RPM values at which the generator efficiency values are given</td>
</tr>
<tr>
<td>rp.powercurve.nSector</td>
<td>Unavailable</td>
<td>number of sectors to divide rotor face into in computing thrust and power-Number of sectors to divide rotor face into in computing thrust and power.</td>
</tr>
<tr>
<td>rp.powercurve.pitch</td>
<td>deg</td>
<td>rotor pitch schedule</td>
</tr>
<tr>
<td>rp.powercurve.rated_Omega</td>
<td>rpm</td>
<td>rotor rotation speed at rated</td>
</tr>
<tr>
<td>rp.powercurve.rated_Q</td>
<td>N*m</td>
<td>rotor aerodynamic torque at rated</td>
</tr>
<tr>
<td>rp.powercurve.rated_T</td>
<td>N</td>
<td>rotor aerodynamic thrust at rated</td>
</tr>
<tr>
<td>rp.powercurve.rated_V</td>
<td>m/s</td>
<td>rated wind speed</td>
</tr>
<tr>
<td>rp.powercurve.rated_efficiency</td>
<td></td>
<td>Efficiency at rated conditions</td>
</tr>
</tbody>
</table>

continues on next page
### Table 5.2 – continued from previous page

<table>
<thead>
<tr>
<th>Variable</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rp.powercurve.rated_mech</td>
<td>W</td>
<td>Mechanical shaft power at rated</td>
</tr>
<tr>
<td>rp.powercurve.rated_pitch</td>
<td>deg</td>
<td>pitch setting at rated</td>
</tr>
<tr>
<td>rs.3d_curv</td>
<td>deg</td>
<td>total cone angle from precone and curvature</td>
</tr>
<tr>
<td>rs.aero_gust.V_load</td>
<td>m/s</td>
<td>Hub height wind speed.</td>
</tr>
<tr>
<td>rs.aero_gust.azimuth_load</td>
<td>deg</td>
<td>Blade azimuthal location.</td>
</tr>
<tr>
<td>rs.aero_gust.loads_Px</td>
<td>N/m</td>
<td></td>
</tr>
<tr>
<td>rs.aero_gust.loads_Py</td>
<td>N/m</td>
<td></td>
</tr>
<tr>
<td>rs.EIxx</td>
<td>N*m**2</td>
<td>2dwise stiffness (bending about x-axis of airfoil aligned coordinate system)</td>
</tr>
<tr>
<td>rs.EIyy</td>
<td>N*m**2</td>
<td>2apwise stiffness (bending about y-axis of airfoil aligned coordinate system)</td>
</tr>
<tr>
<td>rs.GJ</td>
<td>N*m**2</td>
<td>torsional stiffness (about axial z-direction of airfoil aligned coordinate system)</td>
</tr>
<tr>
<td>rs.theta</td>
<td>deg</td>
<td>twist angle at each section (positive decreases angle of attack)</td>
</tr>
<tr>
<td>rs.omega_load</td>
<td>rpm</td>
<td>Rotor rotation speed.</td>
</tr>
<tr>
<td>rs.Rhub</td>
<td>m</td>
<td>Hub radius.</td>
</tr>
<tr>
<td>rs.Rtip</td>
<td>m</td>
<td>Blade tip location in z_b</td>
</tr>
<tr>
<td>rs.aero_gust.V_load</td>
<td>m/s</td>
<td>Hub height wind speed.</td>
</tr>
<tr>
<td>rs.aero_gust.azimuth_load</td>
<td>deg</td>
<td>Blade azimuthal location.</td>
</tr>
<tr>
<td>rs.aero_gust.loads_Px</td>
<td>N/m</td>
<td></td>
</tr>
<tr>
<td>rs.aero_gust.loads_Py</td>
<td>N/m</td>
<td></td>
</tr>
<tr>
<td>rp.powercurve.shearExp</td>
<td>shear exponent</td>
<td>shear exponent</td>
</tr>
<tr>
<td>rp.powercurve.tang_induct_regII</td>
<td>rotor tangential induction at cut-in wind speed along blade span</td>
<td></td>
</tr>
<tr>
<td>rp.powercurve.tiploss</td>
<td>unavailable</td>
<td>include Prandtl tip loss model</td>
</tr>
<tr>
<td>rp.powercurve.usecd</td>
<td>unavailable</td>
<td>use drag coefficient in computing induction factors</td>
</tr>
<tr>
<td>rp.powercurve.wakerotation</td>
<td>unavailable</td>
<td>include effect of wake rotation (i.e., tangential induction factor is nonzero)</td>
</tr>
<tr>
<td>rp.precone</td>
<td>deg</td>
<td>precone angle</td>
</tr>
<tr>
<td>rp.precurve</td>
<td>m</td>
<td>precurve at each section</td>
</tr>
<tr>
<td>rp.precurveTip</td>
<td>m</td>
<td>precurve at tip</td>
</tr>
<tr>
<td>rp.presweep</td>
<td>m</td>
<td>presweep at each section</td>
</tr>
<tr>
<td>rp.presweepTip</td>
<td>m</td>
<td>presweep at tip</td>
</tr>
<tr>
<td>rp.r</td>
<td>m</td>
<td>radial locations where blade is defined (should be increasing and not go all the way to hub or tip)</td>
</tr>
<tr>
<td>rp.rated_power</td>
<td>W</td>
<td>electrical rated power</td>
</tr>
<tr>
<td>rp.rho</td>
<td>kg/m**3</td>
<td>density of air</td>
</tr>
<tr>
<td>rp.tilt</td>
<td>deg</td>
<td>shaft tilt</td>
</tr>
<tr>
<td>rp.tsr_operational</td>
<td>tip-speed ratio in Region 2 (should be optimized externally)</td>
<td></td>
</tr>
<tr>
<td>rp.v_max</td>
<td>m/s</td>
<td>cut-out wind speed</td>
</tr>
<tr>
<td>rp.v_min</td>
<td>m/s</td>
<td>cut-in wind speed</td>
</tr>
<tr>
<td>rp.yaw</td>
<td>deg</td>
<td>yaw error</td>
</tr>
<tr>
<td>rs.A</td>
<td>m**2</td>
<td>airfoil cross section material area</td>
</tr>
<tr>
<td>rs.EA</td>
<td>N</td>
<td>axial stiffness</td>
</tr>
<tr>
<td>rs.EIxx</td>
<td>N*m**2</td>
<td>2dwise stiffness (bending about x-axis of airfoil aligned coordinate system)</td>
</tr>
<tr>
<td>rs.EIxy</td>
<td>N*m**2</td>
<td>coupled flap-edge stiffness</td>
</tr>
<tr>
<td>rs.EIyy</td>
<td>N*m**2</td>
<td>2apwise stiffness (bending about y-axis of airfoil aligned coordinate system)</td>
</tr>
<tr>
<td>rs.GJ</td>
<td>N*m**2</td>
<td>torsional stiffness (about axial z-direction of airfoil aligned coordinate system)</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Variable</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rs.aero_gust.loads_Pz</td>
<td>N/m</td>
<td></td>
</tr>
<tr>
<td>rs.aero_gust.loads_r</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>rs.aero_gust.nSector</td>
<td></td>
<td>Number of sectors to divide rotor face into in computing thrust and power.</td>
</tr>
<tr>
<td>rs.aero_gust.precurve</td>
<td>m</td>
<td>Precurve at each section.</td>
</tr>
<tr>
<td>rs.aero_gust.precurveTip</td>
<td>m</td>
<td>Precurve at tip.</td>
</tr>
<tr>
<td>rs.aero_gust.shearExp</td>
<td></td>
<td>shear exponent</td>
</tr>
<tr>
<td>rs.aero_gust.tiploss</td>
<td></td>
<td>Include Prandtl tip loss model.</td>
</tr>
<tr>
<td>rs.aero_gust.usecd</td>
<td></td>
<td>Use drag coefficient in computing induction factors.</td>
</tr>
<tr>
<td>rs.aero_gust.wakerotation</td>
<td></td>
<td>Include effect of wake rotation (i.e., tangential induction factor is nonzero).</td>
</tr>
<tr>
<td>rs.aero_gust.yaw</td>
<td>deg</td>
<td>yaw angle</td>
</tr>
<tr>
<td>rs.aero_hub_loads.Fxyz_blade_aero</td>
<td></td>
<td>Forces at blade root from aerodynamic loading in the blade c.s.</td>
</tr>
<tr>
<td>rs.aero_hub_loads.Fxyz_hub_aero</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rs.aero_hub_loads.Mxyz_blade_aero</td>
<td></td>
<td>Moments at blade root from aerodynamic loading in the blade c.s.</td>
</tr>
<tr>
<td>rs.aero_hub_loads.Mxyz_hub_aero</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rs.aero_hub_loads.V_load</td>
<td>m/s</td>
<td>Hub height wind speed.</td>
</tr>
<tr>
<td>rs.aero_hub_loads.hubloss</td>
<td></td>
<td>Include Prandtl hub loss model.</td>
</tr>
<tr>
<td>rs.aero_hub_loads.precurve</td>
<td>m</td>
<td>Precurve at each section.</td>
</tr>
<tr>
<td>rs.aero_hub_loads.precurveTip</td>
<td>m</td>
<td>Precurve at tip.</td>
</tr>
<tr>
<td>rs.aero_hub_loads.shearExp</td>
<td></td>
<td>shear exponent</td>
</tr>
<tr>
<td>rs.aero_hub_loads.tiploss</td>
<td></td>
<td>Include Prandtl tip loss model.</td>
</tr>
<tr>
<td>rs.aero_hub_loads.usecd</td>
<td></td>
<td>Use drag coefficient in computing induction factors.</td>
</tr>
<tr>
<td>rs.aero_hub_loads.wakerotation</td>
<td></td>
<td>Include effect of wake rotation (i.e., tangential induction factor is nonzero).</td>
</tr>
<tr>
<td>rs.aero_hub_loads.yaw</td>
<td>deg</td>
<td>yaw angle</td>
</tr>
<tr>
<td>rs.airfoils_Re</td>
<td></td>
<td>Reynolds numbers of polars.</td>
</tr>
<tr>
<td>rs.airfoils_aoa</td>
<td>deg</td>
<td>Angle of attack grid for polars.</td>
</tr>
<tr>
<td>rs.airfoils_cd</td>
<td></td>
<td>Drag coefficients, spanwise.</td>
</tr>
<tr>
<td>rs.airfoils_cl</td>
<td></td>
<td>Lift coefficients, spanwise.</td>
</tr>
<tr>
<td>rs.airfoils_cm</td>
<td></td>
<td>Moment coefficients, spanwise.</td>
</tr>
<tr>
<td>rs.chord</td>
<td>m</td>
<td>Chord length at each section.</td>
</tr>
<tr>
<td>rs.constr.blade_number</td>
<td></td>
<td>number of rotor blades</td>
</tr>
<tr>
<td>rs.constr.constr_edge_f_margin</td>
<td></td>
<td>constraint on edge blade frequency such that ratio of 3P/f is above or below gamma with constraint &lt;= 0</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Variable</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rs.constr.flap_f_margin</td>
<td></td>
<td>constraint on flap blade frequency such that ratio of 3P/f is above or below gamma with constraint &lt;= 0</td>
</tr>
<tr>
<td>rs.constr.max_strainL_spar</td>
<td></td>
<td>constraint for maximum strain in spar cap pressure side</td>
</tr>
<tr>
<td>rs.constr.max_strainU_spar</td>
<td></td>
<td>constraint for maximum strain in spar cap suction side</td>
</tr>
<tr>
<td>rs.constr.edge_mode_freqs</td>
<td>Hz</td>
<td>Frequencies associated with mode shapes in the edge direction</td>
</tr>
<tr>
<td>rs.constr.flap_mode_freqs</td>
<td>Hz</td>
<td>Frequencies associated with mode shapes in the flap direction</td>
</tr>
<tr>
<td>rs.constr.max_strainL_spar</td>
<td></td>
<td>maximum strain in spar cap pressure side</td>
</tr>
<tr>
<td>rs.constr.max_strainU_spar</td>
<td></td>
<td>maximum strain in spar cap suction side</td>
</tr>
<tr>
<td>rs.constr.min_strainL_spar</td>
<td></td>
<td>minimum strain in spar cap pressure side</td>
</tr>
<tr>
<td>rs.constr.min_strainU_spar</td>
<td></td>
<td>minimum strain in spar cap suction side</td>
</tr>
<tr>
<td>rs.constr.rated_Omega</td>
<td>rpm</td>
<td>rotor rotation speed at rated</td>
</tr>
<tr>
<td>rs.constr.s</td>
<td></td>
<td>1D array of the non-dimensional spanwise grid defined along blade axis (0-blade root, 1-blade tip)</td>
</tr>
<tr>
<td>rs.constr.s_opt_spar_cap_ps</td>
<td></td>
<td>1D array of the non-dimensional spanwise grid defined along blade axis to optimize the blade spar cap suction side</td>
</tr>
<tr>
<td>rs.constr.s_opt_spar_cap_ss</td>
<td></td>
<td>1D array of the non-dimensional spanwise grid defined along blade axis to optimize the blade spar cap suction side</td>
</tr>
<tr>
<td>rs.constr.strainL_spar</td>
<td></td>
<td>strain in spar cap on lower surface at location xl,yl_strain with loads P_strain</td>
</tr>
<tr>
<td>rs.constr.strainU_spar</td>
<td></td>
<td>strain in spar cap on upper surface at location xu,yu_strain with loads P_strain</td>
</tr>
<tr>
<td>rs.curvature.s</td>
<td>m</td>
<td>cumulative path length along blade</td>
</tr>
<tr>
<td>rs.frame.Px_af</td>
<td></td>
<td>distributed load (force per unit length) in airfoil x-direction</td>
</tr>
<tr>
<td>rs.frame.Py_af</td>
<td></td>
<td>distributed load (force per unit length) in airfoil y-direction</td>
</tr>
<tr>
<td>rs.frame.Pz_af</td>
<td></td>
<td>distributed load (force per unit length) in airfoil z-direction</td>
</tr>
<tr>
<td>rs.frame.all_mode_shapes</td>
<td></td>
<td>6-degree polynomial coefficients of mode shapes in the edge direction (x^2..x^6, no linear or constant term)</td>
</tr>
<tr>
<td>rs.frame.dx</td>
<td>m</td>
<td>deflection of blade section in airfoil x-direction</td>
</tr>
<tr>
<td>rs.frame.dy</td>
<td>m</td>
<td>deflection of blade section in airfoil y-direction</td>
</tr>
<tr>
<td>rs.frame.dz</td>
<td>m</td>
<td>deflection of blade section in airfoil z-direction</td>
</tr>
<tr>
<td>rs.frame.edge_mode_freqs</td>
<td>Hz</td>
<td>Frequencies associated with mode shapes in the edge direction</td>
</tr>
<tr>
<td>rs.frame.edge_mode_shapes</td>
<td></td>
<td>6-degree polynomial coefficients of mode shapes in the edge direction (x^2..x^6, no linear or constant term)</td>
</tr>
<tr>
<td>rs.frame.flap_mode_freqs</td>
<td>Hz</td>
<td>Frequencies associated with mode shapes in the flap direction</td>
</tr>
<tr>
<td>rs.frame.flap_mode_shapes</td>
<td></td>
<td>6-degree polynomial coefficients of mode shapes in the flap direction (x^2..x^6, no linear or constant term)</td>
</tr>
<tr>
<td>rs.frame.freq_distance</td>
<td></td>
<td>ratio of 2nd and 1st natural frequencies, should be ratio of edgewise to flapwise</td>
</tr>
<tr>
<td>rs.frame.freqs</td>
<td>Hz</td>
<td>ratio of 2nd and 1st natural frequencies, should be ratio of edgewise to flapwise</td>
</tr>
<tr>
<td>rs.frame.root_F</td>
<td>N</td>
<td>Blade root forces in blade c.s.</td>
</tr>
<tr>
<td>rs.frame.root_M</td>
<td>N*m</td>
<td>Blade root moment in blade c.s.</td>
</tr>
<tr>
<td>rs.frame.strainL_spar</td>
<td></td>
<td>strain in spar cap on lower surface at location xl,yl_strain with loads P_strain</td>
</tr>
<tr>
<td>rs.frame.strainL_te</td>
<td></td>
<td>strain in trailing-edge panels on lower surface at location xl,yl_te with loads P_te</td>
</tr>
<tr>
<td>rs.frame.strainU_spar</td>
<td></td>
<td>strain in spar cap on upper surface at location xu,yu_strain with loads P_strain</td>
</tr>
</tbody>
</table>

continues on next page
Table 5.2 – continued from previous page

<table>
<thead>
<tr>
<th>Variable</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rs.frame.strainU_te</td>
<td></td>
<td>strain in trailing-edge panels on upper surface at location xu,yu_te with loads P_te</td>
</tr>
<tr>
<td>rs.hub_height</td>
<td>m</td>
<td>hub height of wind turbine above ground / sea level</td>
</tr>
<tr>
<td>rs.mu</td>
<td>kg/(m*s)</td>
<td>Dynamic viscosity of air</td>
</tr>
<tr>
<td>rs.nBlades</td>
<td>Unavailable</td>
<td>Number of blades</td>
</tr>
<tr>
<td>rs.pitch_load</td>
<td>deg</td>
<td>Blade pitch setting.</td>
</tr>
<tr>
<td>rs.precone</td>
<td>deg</td>
<td>Rotor precone angle</td>
</tr>
<tr>
<td>rs.precurve</td>
<td>m</td>
<td>location in blade x-coordinate Precurve at each section.</td>
</tr>
<tr>
<td>rs.presweep</td>
<td>m</td>
<td>location in blade y-coordinate</td>
</tr>
<tr>
<td>rs.r</td>
<td>m</td>
<td>Radial locations where blade is defined. Should be increasing and not go all the way to hub or tip.</td>
</tr>
<tr>
<td>rs.rho</td>
<td>kg/m**3</td>
<td>Density of the materials along the column sections.</td>
</tr>
<tr>
<td>rs.rhoA</td>
<td>kg/m</td>
<td>mass per unit length</td>
</tr>
<tr>
<td>rs.rhoJ</td>
<td>kg*m</td>
<td>polar mass moment of inertia per unit length</td>
</tr>
<tr>
<td>rs.theta</td>
<td>deg</td>
<td>Rotor precone angle</td>
</tr>
<tr>
<td>rs.tilt</td>
<td>deg</td>
<td>Nacelle uptilt angle</td>
</tr>
<tr>
<td>rs.tip_pos.3d_curv_tip</td>
<td>deg</td>
<td>total coning angle including precone and curvature</td>
</tr>
<tr>
<td>rs.tip_pos.dx_tip</td>
<td>m</td>
<td>deflection at tip in blade x-direction</td>
</tr>
<tr>
<td>rs.tip_pos.dy_tip</td>
<td>m</td>
<td>deflection at tip in blade y-direction</td>
</tr>
<tr>
<td>rs.tip_pos.dz_tip</td>
<td>m</td>
<td>deflection at tip in blade z-direction</td>
</tr>
<tr>
<td>rs.tip_pos.tip_deflection</td>
<td>m</td>
<td>deflection at tip in yaw x-direction Blade tip deflection in yaw x-direction</td>
</tr>
<tr>
<td>rs.tot_loads_gust.Px_af</td>
<td></td>
<td>total distributed loads in airfoil x-direction</td>
</tr>
<tr>
<td>rs.tot_loads_gust.Py_af</td>
<td></td>
<td>total distributed loads in airfoil y-direction</td>
</tr>
<tr>
<td>rs.tot_loads_gust.Pz_af</td>
<td></td>
<td>total distributed loads in airfoil z-direction</td>
</tr>
<tr>
<td>rs.tot_loads_gust.aeroloads_Omega</td>
<td>rpm</td>
<td>rotor rotation speed</td>
</tr>
<tr>
<td>rs.tot_loads_gust.aeroloads_Px</td>
<td>N/m</td>
<td>distributed loads in blade-aligned x-direction</td>
</tr>
<tr>
<td>rs.tot_loads_gust.aeroloads_Py</td>
<td>N/m</td>
<td>distributed loads in blade-aligned y-direction</td>
</tr>
<tr>
<td>rs.tot_loads_gust.aeroloads_Pz</td>
<td>N/m</td>
<td>distributed loads in blade-aligned z-direction</td>
</tr>
<tr>
<td>rs.tot_loads_gust.aeroloads_azimuth</td>
<td>azimuth</td>
<td>azimuthal angle</td>
</tr>
<tr>
<td>rs.tot_loads_gust.aeroloads_pitch</td>
<td>deg</td>
<td>pitch angle</td>
</tr>
<tr>
<td>rs.xl_strain_spar</td>
<td></td>
<td>x-position of midpoint of spar cap on lower surface for strain calculation</td>
</tr>
<tr>
<td>rs.xl_strain_te</td>
<td></td>
<td>x-position of midpoint of trailing-edge panel on lower surface for strain calculation</td>
</tr>
<tr>
<td>rs.xu_strain_spar</td>
<td></td>
<td>x-position of midpoint of spar cap on upper surface for strain calculation</td>
</tr>
<tr>
<td>rs.xu_strain_te</td>
<td></td>
<td>x-position of midpoint of trailing-edge panel on upper surface for strain calculation</td>
</tr>
<tr>
<td>rs.yl_strain_spar</td>
<td></td>
<td>location of blade in azimuth x-coordinate system (prebend)</td>
</tr>
<tr>
<td>rs.yl_strain_te</td>
<td></td>
<td>x-distance to elastic center from point about which above structural properties are computed (airfoil aligned coordinate system)</td>
</tr>
<tr>
<td>rs.yu_strain_spar</td>
<td></td>
<td>y-position of midpoint of spar cap on lower surface for strain calculation</td>
</tr>
<tr>
<td>rs.yu_strain_te</td>
<td></td>
<td>y-position of midpoint of trailing-edge panel on lower surface for strain calculation</td>
</tr>
<tr>
<td>rs.y_axis</td>
<td></td>
<td>location of blade in azimuth y-coordinate system (sweep)</td>
</tr>
<tr>
<td>rs.y_ec</td>
<td>m</td>
<td>y-distance to elastic center from point about which above structural properties are computed</td>
</tr>
<tr>
<td>continues on next page</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Variable</td>
<td>Units</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>rs.yu_strain_te</td>
<td></td>
<td>y-position of midpoint of trailing-edge panel on upper surface for strain</td>
</tr>
<tr>
<td>rs.z_az</td>
<td>m</td>
<td>location of blade in azimuth z-coordinate system</td>
</tr>
<tr>
<td>stall_check.airfoils_aoa</td>
<td>deg</td>
<td>angle of attack grid for polars Angle of attack grid for polars.</td>
</tr>
<tr>
<td>stall_check.airfoils_cd</td>
<td></td>
<td>drag coefficients, spanwise Drag coefficients, spanwise.</td>
</tr>
<tr>
<td>stall_check.airfoils_cm</td>
<td></td>
<td>lift coefficients, spanwise Lift coefficients, spanwise.</td>
</tr>
<tr>
<td>stall_check.airfoils_cm</td>
<td>deg</td>
<td>moment coefficients, spanwise Moment coefficients, spanwise.</td>
</tr>
<tr>
<td>stall_check.aao_along_span</td>
<td>deg</td>
<td>Angle of attack along blade span</td>
</tr>
<tr>
<td>stall_check.min_s</td>
<td></td>
<td>Minimum nondimensional coordinate along blade span where to define the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>constraint (blade root typically stalls)</td>
</tr>
<tr>
<td>stall_check.no_stall_constraint</td>
<td></td>
<td>Constraint, ratio between angle of attack plus a margin and stall angle</td>
</tr>
<tr>
<td>stall_check.s</td>
<td></td>
<td>1D array of the non-dimensional spanwise grid defined along blade axis (0-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>blade root, 1-blade tip) 1D array of the non-dimensional grid defined along</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the column axis (0-column base, 1-column top)</td>
</tr>
<tr>
<td>stall_check.stall_angle_along_span</td>
<td>deg</td>
<td>Stall angle along blade span</td>
</tr>
<tr>
<td>stall_check.stall_margin</td>
<td>deg</td>
<td>Minimum margin from the stall angle</td>
</tr>
<tr>
<td>tcc.bearing_mass_cost_coeff</td>
<td>USD/kg</td>
<td>main bearing mass-cost coeff</td>
</tr>
<tr>
<td>tcc.bedplate_cost</td>
<td>USD</td>
<td>bedplate cost</td>
</tr>
<tr>
<td>tcc.bedplate_mass</td>
<td>kg</td>
<td>component mass</td>
</tr>
<tr>
<td>tcc.bedplate_mass_cost_coeff</td>
<td>USD/kg</td>
<td>bedplate mass-cost coeff</td>
</tr>
<tr>
<td>tcc.blade_cost</td>
<td>USD</td>
<td>Individual blade cost</td>
</tr>
<tr>
<td>tcc.blade_cost_external</td>
<td>USD</td>
<td>Blade cost computed by RotorSE</td>
</tr>
<tr>
<td>tcc.blade_mass</td>
<td>kg</td>
<td>Total mass of one blade</td>
</tr>
<tr>
<td>tcc.blade_mass_cost_coeff</td>
<td>USD/kg</td>
<td>blade mass-cost coeff</td>
</tr>
<tr>
<td>tcc.blade_number</td>
<td>Unavail-</td>
<td>number of rotor blades</td>
</tr>
<tr>
<td></td>
<td>able</td>
<td></td>
</tr>
<tr>
<td>tcc.brake_cost</td>
<td>USD</td>
<td>brake cost</td>
</tr>
<tr>
<td>tcc.brake_mass</td>
<td>kg</td>
<td>component mass</td>
</tr>
<tr>
<td>tcc.brake_mass_cost_coeff</td>
<td>USD/kg</td>
<td>brake mass-cost coeff</td>
</tr>
<tr>
<td>tcc.controls_cost</td>
<td>USD</td>
<td>controls cost</td>
</tr>
<tr>
<td>tcc.controls_cost</td>
<td>USD</td>
<td>controls cost per kW</td>
</tr>
<tr>
<td>tcc.controls_machine_rating_cost_coeff</td>
<td>USD/kW</td>
<td>electrical connections cost coefficient per kW</td>
</tr>
<tr>
<td>tcc.converter_cost</td>
<td>USD</td>
<td>converter cost</td>
</tr>
<tr>
<td>tcc.converter_mass</td>
<td>kg</td>
<td>overall component mass</td>
</tr>
<tr>
<td>tcc.converter_mass_cost_coeff</td>
<td>USD/kg</td>
<td>variable speed electronics mass cost coeff</td>
</tr>
<tr>
<td>tcc.cover_cost</td>
<td>USD</td>
<td>cover cost</td>
</tr>
<tr>
<td>tcc.cover_mass</td>
<td>kg</td>
<td>component mass</td>
</tr>
<tr>
<td>tcc.cover_mass_cost_coeff</td>
<td>USD/kg</td>
<td>nacelle cover mass coeff</td>
</tr>
<tr>
<td>tcc.crane</td>
<td>Unavail-</td>
<td>flag for presence of onboard crane</td>
</tr>
<tr>
<td></td>
<td>able</td>
<td></td>
</tr>
<tr>
<td>tcc.crane_cost</td>
<td>USD</td>
<td>crane cost if present</td>
</tr>
<tr>
<td>tcc.elec_connc_machine_rating_cost_coeff</td>
<td>USD/kW</td>
<td>electrical connections cost coefficient per kW</td>
</tr>
<tr>
<td>tcc.elec_cost</td>
<td>USD</td>
<td>elec cost</td>
</tr>
<tr>
<td>tcc.gearbox_cost</td>
<td>USD</td>
<td>gearbox cost</td>
</tr>
<tr>
<td>tcc.gearbox_mass</td>
<td>kg</td>
<td>Gearbox rotor mass</td>
</tr>
<tr>
<td>tcc.gearbox_mass_cost_coeff</td>
<td>USD/kg</td>
<td>gearbox mass-cost coeff</td>
</tr>
<tr>
<td>tcc.generator_cost</td>
<td>USD</td>
<td>generator cost</td>
</tr>
<tr>
<td>tcc.generator_cost_external</td>
<td>USD</td>
<td>Generator cost computed by GeneratorSE</td>
</tr>
<tr>
<td>tcc.generator_mass</td>
<td>kg</td>
<td>Actual mass</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Variable</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tcc.generator_mass_cost_coeff</td>
<td>USD/kg</td>
<td>generator mass cost coeff</td>
</tr>
<tr>
<td>tcc.hss_cost</td>
<td>USD</td>
<td>hss cost</td>
</tr>
<tr>
<td>tcc.hss_mass</td>
<td>kg</td>
<td>component mass</td>
</tr>
<tr>
<td>tcc.hss_mass_cost_coeff</td>
<td>USD/kg</td>
<td>high speed shaft mass-cost coeff</td>
</tr>
<tr>
<td>tcc.hub_assemblyCostMultiplier</td>
<td></td>
<td>Rotor assembly cost multiplier</td>
</tr>
<tr>
<td>tcc.hub_cost</td>
<td>USD</td>
<td>Cost of the hub shell, including flanges</td>
</tr>
<tr>
<td>tcc.hub_mass</td>
<td>kg</td>
<td>Total mass of the hub shell, including the flanges</td>
</tr>
<tr>
<td>tcc.hub_mass_cost_coeff</td>
<td>USD/kg</td>
<td>hub mass-cost coeff</td>
</tr>
<tr>
<td>tcc.hub_overheadCostMultiplier</td>
<td></td>
<td>Rotor overhead cost multiplier</td>
</tr>
<tr>
<td>tcc.hub_profileMultiplier</td>
<td></td>
<td>Rotor profit multiplier</td>
</tr>
<tr>
<td>tcc.hub_system_cost</td>
<td>USD</td>
<td>Cost for hub system</td>
</tr>
<tr>
<td>tcc.hub_system_mass_tcc</td>
<td>kg</td>
<td>Mass for hub system</td>
</tr>
<tr>
<td>tcc.hub_transportMultiplier</td>
<td></td>
<td>Rotor transport multiplier</td>
</tr>
<tr>
<td>tcc.hvac_cost</td>
<td>USD</td>
<td>hvac cost</td>
</tr>
<tr>
<td>tcc.hvac_mass</td>
<td>kg</td>
<td>component mass</td>
</tr>
<tr>
<td>tcc.hvac_mass_cost_coeff</td>
<td>USD/kg</td>
<td>hydraulic and cooling system mass cost coeff</td>
</tr>
<tr>
<td>tcc.lss_cost</td>
<td>USD</td>
<td>lss cost</td>
</tr>
<tr>
<td>tcc.lss_mass</td>
<td>kg</td>
<td>LSS mass</td>
</tr>
<tr>
<td>tcc.lss_mass_costCoeff</td>
<td>USD/kg</td>
<td>slow speed shaft mass-cost coeff</td>
</tr>
<tr>
<td>tcc.machine_rating</td>
<td>kW</td>
<td>Machine rating</td>
</tr>
<tr>
<td>tcc.main_bearing_cost</td>
<td>USD</td>
<td>main_bearing cost</td>
</tr>
<tr>
<td>tcc.main_bearing_mass</td>
<td>kg</td>
<td>Main bearing mass</td>
</tr>
<tr>
<td>tcc.main_bearing_number</td>
<td>Unavailable</td>
<td>number of bearings</td>
</tr>
<tr>
<td>tcc.nacelle_assemblyCostMultiplier</td>
<td>USD</td>
<td>nacelle assembly cost multiplier</td>
</tr>
<tr>
<td>tcc.nacelle_cost</td>
<td>USD</td>
<td>Nacelle cost</td>
</tr>
<tr>
<td>tcc.nacelle_mass_tcc</td>
<td>kg</td>
<td>Nacelle mass</td>
</tr>
<tr>
<td>tcc.nacelle_overheadCostMultiplier</td>
<td>USD</td>
<td>nacelle overhead cost multiplier</td>
</tr>
<tr>
<td>tcc.nacelle_profitMultiplier</td>
<td></td>
<td>nacelle profit multiplier</td>
</tr>
<tr>
<td>tcc.nacelle_transportMultiplier</td>
<td></td>
<td>nacelle transport multiplier</td>
</tr>
<tr>
<td>tcc.pitch_system_cost</td>
<td>USD</td>
<td>pitch_system cost</td>
</tr>
<tr>
<td>tcc.pitch_system_mass</td>
<td>kg</td>
<td>component mass</td>
</tr>
<tr>
<td>tcc.pitch_system_mass_cost_coeff</td>
<td>USD/kg</td>
<td>pitch system mass-cost coeff</td>
</tr>
<tr>
<td>tcc.platforms_cost</td>
<td>USD</td>
<td>plattforms cost</td>
</tr>
<tr>
<td>tcc.platforms_mass</td>
<td>kg</td>
<td>component mass</td>
</tr>
<tr>
<td>tcc.platforms_mass_cost_coeff</td>
<td>USD/kg</td>
<td>nacelle platforms mass cost coeff</td>
</tr>
<tr>
<td>tcc.rotor_cost</td>
<td>USD</td>
<td>Rotor cost</td>
</tr>
<tr>
<td>tcc.rotor_mass_tcc</td>
<td>kg</td>
<td>Rotor mass</td>
</tr>
<tr>
<td>tcc.spinner_cost</td>
<td>USD</td>
<td>Cost of the spinner</td>
</tr>
<tr>
<td>tcc.spinner_mass</td>
<td>kg</td>
<td>Total mass of the spinner</td>
</tr>
<tr>
<td>tcc.spinner_mass_cost_coeff</td>
<td>USD/kg</td>
<td>spinner/nose cone mass-cost coeff</td>
</tr>
<tr>
<td>tcc.tower_assemblyCostMultiplier</td>
<td>USD</td>
<td>tower assembly cost multiplier</td>
</tr>
<tr>
<td>tcc.tower_cost</td>
<td>USD</td>
<td>Tower cost</td>
</tr>
<tr>
<td>tcc.tower_cost_external</td>
<td>USD</td>
<td>Tower cost computed by TowerSE</td>
</tr>
<tr>
<td>tcc.tower_mass</td>
<td>kg</td>
<td>Mass of tower</td>
</tr>
<tr>
<td>tcc.tower_mass_cost_coeff</td>
<td>USD/kg</td>
<td>tower mass-cost coeff</td>
</tr>
<tr>
<td>tcc.tower_overheadCostMultiplier</td>
<td>USD</td>
<td>tower overhead cost multiplier</td>
</tr>
<tr>
<td>tcc.tower_parts_cost</td>
<td>USD</td>
<td>component cost</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Variable</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tcc.tower_profitMultiplier</td>
<td></td>
<td>tower profit cost multiplier</td>
</tr>
<tr>
<td>tcc.tower_transportMultiplier</td>
<td></td>
<td>tower transport cost multiplier</td>
</tr>
<tr>
<td>tcc.transformer_cost</td>
<td>USD</td>
<td>transformer cost</td>
</tr>
<tr>
<td>tcc.transformer_mass</td>
<td>kg</td>
<td>overall component mass</td>
</tr>
<tr>
<td>tcc.transformer_mass_cost_coeff</td>
<td>USD/kg</td>
<td>transformer mass cost coeff</td>
</tr>
<tr>
<td>tcc.turbine_assemblyCostMultiplier</td>
<td></td>
<td>Turbine multiplier for assembly cost in manufacturing</td>
</tr>
<tr>
<td>tcc.turbine_cost</td>
<td>USD</td>
<td>Overall turbine costs</td>
</tr>
<tr>
<td>tcc.turbine_cost_kW</td>
<td>USD/kW</td>
<td>Overall wind turbine capital costs including transportation costs per kW</td>
</tr>
<tr>
<td>tcc.turbine_mass_tcc</td>
<td>kg</td>
<td>Turbine mass</td>
</tr>
<tr>
<td>tcc.turbine_overheadCostMultiplier</td>
<td></td>
<td>Turbine multiplier for overhead</td>
</tr>
<tr>
<td>tcc.turbine_profitMultiplier</td>
<td></td>
<td>Turbine multiplier for profit markup</td>
</tr>
<tr>
<td>tcc.turbine_transportMultiplier</td>
<td></td>
<td>Turbine multiplier for transport costs</td>
</tr>
<tr>
<td>tcc.yaw_mass</td>
<td>kg</td>
<td>overall component mass</td>
</tr>
<tr>
<td>tcc.yaw_mass_cost_coeff</td>
<td>USD/kg</td>
<td>yaw system mass cost coeff</td>
</tr>
<tr>
<td>tcc.yaw_system_cost</td>
<td>USD</td>
<td>yaw_system cost</td>
</tr>
<tr>
<td>tcons.Rtip</td>
<td>m</td>
<td>Blade tip location in z_b</td>
</tr>
<tr>
<td>tcons.blade_number</td>
<td></td>
<td>number of rotor blades</td>
</tr>
<tr>
<td>tcons.blade_tip_tower_clearance</td>
<td></td>
<td>constraint on tower frequency such that ratio of 1P/f is above or below</td>
</tr>
<tr>
<td>tcons.constr_tower_f_1Pmargin</td>
<td></td>
<td>gamma with constraint &lt;= 0</td>
</tr>
<tr>
<td>tcons.constr_tower_f_NPmargin</td>
<td></td>
<td>constraint on tower frequency such that ratio of 3P/f is above or below</td>
</tr>
<tr>
<td>tcons.d_full</td>
<td>m</td>
<td>cylinder diameter at corresponding locations</td>
</tr>
<tr>
<td>tcons.max_allowable_td_ratio</td>
<td></td>
<td>Safety factor of the tip deflection to stay within the tower clearance</td>
</tr>
<tr>
<td>tcons.overhang</td>
<td>m</td>
<td>Horizontal distance between hub and tower-top axis</td>
</tr>
<tr>
<td>tcons.precone</td>
<td>deg</td>
<td>Rotor precone angle</td>
</tr>
<tr>
<td>tcons.rated_Omega</td>
<td>rpm</td>
<td>rotor rotation speed at rated</td>
</tr>
<tr>
<td>tcons.ref_axis_blade</td>
<td>m</td>
<td>2D array of the coordinates (x,y,z) of the blade reference axis, defined</td>
</tr>
<tr>
<td></td>
<td></td>
<td>along blade span. The coordinate system is the one of BeamDyn: it is placed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>at blade root with x pointing the suction side of the blade, y pointing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the trailing edge and z along the blade span. A standard configuration will</td>
</tr>
<tr>
<td></td>
<td></td>
<td>have negative x values (prebend), if swept positive y values, and positive</td>
</tr>
<tr>
<td></td>
<td></td>
<td>z values.</td>
</tr>
<tr>
<td>tcons.ref_axis_tower</td>
<td>m</td>
<td>2D array of the coordinates (x,y,z) of the tower reference axis. The</td>
</tr>
<tr>
<td></td>
<td></td>
<td>coordinate system is the global coordinate system of OpenFAST: it is placed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>at tower base with x pointing downwind, y pointing on the side and z</td>
</tr>
<tr>
<td></td>
<td></td>
<td>pointing vertically upwards. A standard tower configuration will have zero</td>
</tr>
<tr>
<td></td>
<td></td>
<td>x and y values and positive z values.</td>
</tr>
<tr>
<td>tcons.rotor_orientation</td>
<td>Unavail</td>
<td>Rotor orientation, either upwind or downwind.</td>
</tr>
<tr>
<td>tcons.tilt</td>
<td>deg</td>
<td>Nacelle uptilt angle</td>
</tr>
<tr>
<td>tcons.tip_deflection</td>
<td>m</td>
<td>Blade tip deflection in yaw x-direction</td>
</tr>
<tr>
<td>tcons.tip_deflection_ratio</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tcons.tower_freq</td>
<td>Hz</td>
<td>First natural frequencies of tower (and substructure)</td>
</tr>
<tr>
<td>tower.cd</td>
<td>1D array of the drag coefficients defined along the tower height.</td>
<td></td>
</tr>
<tr>
<td>tower.diameter</td>
<td>m</td>
<td>1D array of the outer diameter values defined along the tower axis.</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Variable</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tower.layer_mat</td>
<td></td>
<td>1D array of the names of the materials of each layer modeled in the tower</td>
</tr>
<tr>
<td></td>
<td></td>
<td>structure.</td>
</tr>
<tr>
<td>tower.layer_name</td>
<td></td>
<td>1D array of the names of the layers modeled in the tower structure.</td>
</tr>
<tr>
<td>tower.layer_thickness</td>
<td>m</td>
<td>2D array of the thickness of the layers of the tower structure. The first</td>
</tr>
<tr>
<td></td>
<td></td>
<td>dimension represents each layer, the second dimension represents each piecewise-constant entry of the tower sections.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2D array of the thickness of the layers of the column structure. The first</td>
</tr>
<tr>
<td></td>
<td></td>
<td>dimension represents each layer, the second dimension represents each piecewise-constant entry of the column sections.</td>
</tr>
<tr>
<td>tower.outfitting_factor</td>
<td></td>
<td>Multiplier that accounts for secondary structure mass inside of towerMass</td>
</tr>
<tr>
<td></td>
<td></td>
<td>fraction added for outfitting</td>
</tr>
<tr>
<td>tower.ref_axis</td>
<td>m</td>
<td>2D array of the coordinates (x,y,z) of the tower reference axis. The coor-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>dinate system is the global coordinate system of OpenFAST: it is placed at</td>
</tr>
<tr>
<td></td>
<td></td>
<td>tower base with x pointing downwind, y pointing on the side and z pointing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>vertically upwards. A standard tower configuration will have zero x and y</td>
</tr>
<tr>
<td></td>
<td></td>
<td>values and positive z values.</td>
</tr>
<tr>
<td>tower_grid.foundation_height</td>
<td>m</td>
<td>Foundation height in respect to the ground level. starting height of tower</td>
</tr>
<tr>
<td>tower_grid.height</td>
<td>m</td>
<td>Scalar of the tower height computed along the z axis. Scalar of the column</td>
</tr>
<tr>
<td></td>
<td></td>
<td>height computed along the z axis.</td>
</tr>
<tr>
<td>tower_grid.length</td>
<td>m</td>
<td>Scalar of the tower length computed along its curved axis. A standard</td>
</tr>
<tr>
<td></td>
<td></td>
<td>straight tower will be as high as long. Length of a single cable connecting</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the OSS to the interconnection in km.</td>
</tr>
<tr>
<td>tower_grid.ref_axis</td>
<td>m</td>
<td>2D array of the coordinates (x,y,z) of the tower reference axis. The coor-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>dinate system is the global coordinate system of OpenFAST: it is placed at</td>
</tr>
<tr>
<td></td>
<td></td>
<td>tower base with x pointing downwind, y pointing on the side and z pointing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>vertically upwards. A standard tower configuration will have zero x and y</td>
</tr>
<tr>
<td></td>
<td></td>
<td>values and positive z values.</td>
</tr>
<tr>
<td>tower_grid.s</td>
<td></td>
<td>1D array of the non-dimensional grid defined along the tower axis (0-tower</td>
</tr>
<tr>
<td></td>
<td></td>
<td>base, 1-tower top) 1D array of the non-dimensional grid defined along the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>column axis (0-column base, 1-column top)</td>
</tr>
<tr>
<td>towerse.Asx</td>
<td>m**2</td>
<td>x shear area</td>
</tr>
<tr>
<td>towerse.Asy</td>
<td>m**2</td>
<td>y shear area</td>
</tr>
<tr>
<td>towerse.Az</td>
<td>m**2</td>
<td>cross-sectional area</td>
</tr>
<tr>
<td>towerse.E</td>
<td>Pa</td>
<td>Isotropic Youngs modulus of the materials along the column sections.</td>
</tr>
<tr>
<td>towerse.E_full</td>
<td>N/m**2</td>
<td>Modulus of elasticity modulus of elasticity</td>
</tr>
<tr>
<td>towerse.E_mat</td>
<td></td>
<td>2D array of the Youngs moduli of the materials. Each row represents a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>material, the three columns represent E11, E22 and E33.</td>
</tr>
<tr>
<td>towerse.G</td>
<td>Pa</td>
<td>Isotropic shear modulus of the materials along the column sections.</td>
</tr>
<tr>
<td>towerse.G_full</td>
<td>Pa</td>
<td>Isotropic shear modulus of the materials along the column sections.</td>
</tr>
<tr>
<td>towerse.G_mat</td>
<td>Pa</td>
<td>2D array of the shear moduli of the materials. Each row represents a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>material, the three columns represent G12, G13 and G23.</td>
</tr>
<tr>
<td>towerse.G_soil</td>
<td>Pa</td>
<td></td>
</tr>
<tr>
<td>towerse.Hsig_wave</td>
<td>m</td>
<td>significant wave height</td>
</tr>
<tr>
<td>towerse.Ixx</td>
<td>m**4</td>
<td>area moment of inertia about x-axis</td>
</tr>
<tr>
<td>towerse.Iyy</td>
<td>m**4</td>
<td>area moment of inertia about y-axis</td>
</tr>
<tr>
<td>towerse.Jz</td>
<td>m**4</td>
<td>polar moment of inertia</td>
</tr>
<tr>
<td>towerse.Tsig_wave</td>
<td>s</td>
<td>period of maximum wave height</td>
</tr>
</tbody>
</table>

continues on next page
Table 5.2 – continued from previous page

<table>
<thead>
<tr>
<th>Variable</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>towerse.axial_stff</td>
<td>N</td>
<td>sectional axial stiffness</td>
</tr>
<tr>
<td>towerse.beta_wave</td>
<td>deg</td>
<td>corresponding wave angles relative to inertial coordinate system</td>
</tr>
<tr>
<td>towerse.beta_wind</td>
<td>deg</td>
<td>corresponding wind angles relative to inertial coordinate system</td>
</tr>
<tr>
<td>towerse.cg_ofst</td>
<td>m</td>
<td>offset from the sectional center of mass</td>
</tr>
<tr>
<td>towerse.cm</td>
<td></td>
<td>mass coefficient</td>
</tr>
<tr>
<td>towerse.cm.I_base</td>
<td>kg*m**2</td>
<td></td>
</tr>
<tr>
<td>towerse.cm.center_of_mass</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>towerse.cm.cost</td>
<td>USD</td>
<td></td>
</tr>
<tr>
<td>towerse.cm.mass</td>
<td>kg</td>
<td>added mass</td>
</tr>
<tr>
<td>towerse.cm.material_cost_rate</td>
<td>USD/kg</td>
<td>Raw material cost rate: steel $1.1/kg, aluminum $3.5/kg</td>
</tr>
<tr>
<td>towerse.cm.outfitting_factor</td>
<td></td>
<td>Mass fraction added for outfitting</td>
</tr>
<tr>
<td>towerse.cm.rho</td>
<td>kg/m**3</td>
<td>Density of the materials along the column sections.</td>
</tr>
<tr>
<td>towerse.cm.section_center_of_mass</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>towerse.constr_d_to_t</td>
<td></td>
<td></td>
</tr>
<tr>
<td>towerse.constr_taper</td>
<td></td>
<td></td>
</tr>
<tr>
<td>towerse.d_full</td>
<td>m</td>
<td>cylinder diameter at corresponding locations</td>
</tr>
<tr>
<td>towerse.distLoads.Px</td>
<td>N/m</td>
<td>force per unit length in x-direction</td>
</tr>
<tr>
<td>towerse.distLoads.Py</td>
<td>N/m</td>
<td>force per unit length in y-direction</td>
</tr>
<tr>
<td>towerse.distLoads.Pz</td>
<td>N/m</td>
<td>force per unit length in z-direction</td>
</tr>
<tr>
<td>towerse.distLoads.qdyn</td>
<td>N/m**2</td>
<td>Dynamic pressure</td>
</tr>
<tr>
<td>towerse.distLoads.wind Loads Px</td>
<td>N/m</td>
<td>distributed loads, force per unit length in x-direction</td>
</tr>
<tr>
<td>towerse.distLoads.wind Loads Py</td>
<td>N/m</td>
<td>distributed loads, force per unit length in y-direction</td>
</tr>
<tr>
<td>towerse.distLoads.wind Loads Pz</td>
<td>N/m</td>
<td>distributed loads, force per unit length in z-direction</td>
</tr>
<tr>
<td>towerse.distLoads.wind Loads beta</td>
<td>deg</td>
<td>wind/wave angle relative to inertia c.s.</td>
</tr>
<tr>
<td>towerse.distLoads.wind Loads d</td>
<td>m</td>
<td>corresponding diameters</td>
</tr>
<tr>
<td>towerse.distLoads.wind Loads z</td>
<td>m</td>
<td>corresponding heights</td>
</tr>
<tr>
<td>towerse.distLoads.wind Loads qdyn</td>
<td>N/m**2</td>
<td>Dynamic pressure</td>
</tr>
<tr>
<td>towerse.distLoads.wind Loads Px</td>
<td>N/m</td>
<td>distributed loads, force per unit length in x-direction</td>
</tr>
<tr>
<td>towerse.distLoads.wind Loads Py</td>
<td>N/m</td>
<td>distributed loads, force per unit length in y-direction</td>
</tr>
<tr>
<td>towerse.distLoads.wind Loads Pz</td>
<td>N/m</td>
<td>distributed loads, force per unit length in z-direction</td>
</tr>
<tr>
<td>towerse.distLoads.wind Loads beta</td>
<td>deg</td>
<td>wind/wave angle relative to inertia c.s.</td>
</tr>
<tr>
<td>towerse.distLoads.wind Loads d</td>
<td>m</td>
<td>corresponding diameters</td>
</tr>
<tr>
<td>towerse.distLoads.wind Loads qdyn</td>
<td>N/m**2</td>
<td>Dynamic pressure</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Variable</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>towerse.distLoads.windLoads_z</code></td>
<td>m</td>
<td>corresponding heights</td>
</tr>
<tr>
<td><code>towerse.distLoads.z</code></td>
<td>m</td>
<td>location along cylinder, start at bottom and go to top</td>
</tr>
<tr>
<td><code>towerse.foreaft_iner</code></td>
<td>kg*m</td>
<td>sectional fore-aft inertia per unit length about the Y_G inertia axis</td>
</tr>
<tr>
<td><code>towerse.foreaft_stiff</code></td>
<td>N*m**2</td>
<td>sectional fore-aft bending stiffness per unit length about the Y_E elastic axis</td>
</tr>
<tr>
<td><code>towerse.geometry.foundation_height</code></td>
<td>m</td>
<td>starting height of tower</td>
</tr>
<tr>
<td><code>towerse.gravity_foundation_I</code></td>
<td>kg*m**2</td>
<td></td>
</tr>
<tr>
<td><code>towerse.gravity_foundation_mass</code></td>
<td>kg</td>
<td>point mass of transition piece</td>
</tr>
<tr>
<td><code>towerse.height_constraint</code></td>
<td>m</td>
<td></td>
</tr>
<tr>
<td><code>towerse.hub_height</code></td>
<td>m</td>
<td>hub height of wind turbine above ground / sea level</td>
</tr>
<tr>
<td><code>towerse.labor_cost_rate</code></td>
<td>USD/min</td>
<td>Labor cost</td>
</tr>
<tr>
<td><code>towerse.life</code></td>
<td></td>
<td>fatigue life of tower</td>
</tr>
<tr>
<td><code>towerse.mass_den</code></td>
<td>kg/m</td>
<td>sectional mass per unit length</td>
</tr>
<tr>
<td><code>towerse.material_names</code></td>
<td>Unavailable</td>
<td>1D array of names of materials.</td>
</tr>
<tr>
<td><code>towerse.monopile_cost</code></td>
<td>USD</td>
<td></td>
</tr>
<tr>
<td><code>towerse.monopile_foundation_height</code></td>
<td>m</td>
<td></td>
</tr>
<tr>
<td><code>towerse.monopile_height</code></td>
<td>m</td>
<td>Scalar of the tower height computed along the z axis.</td>
</tr>
<tr>
<td><code>towerse.monopile_layer_materials</code></td>
<td>Unavailable</td>
<td>1D array of the names of the materials of each layer modeled in the tower structure.</td>
</tr>
<tr>
<td><code>towerse.monopile_layer_thickness</code></td>
<td>m</td>
<td>2D array of the thickness of the layers of the tower structure. The first dimension represents each layer, the second dimension represents each piecewise-constant entry of the tower sections.</td>
</tr>
<tr>
<td><code>towerse.monopile_length</code></td>
<td>m</td>
<td></td>
</tr>
<tr>
<td><code>towerse.monopile_mass</code></td>
<td>kg</td>
<td>Monopile mass</td>
</tr>
<tr>
<td><code>towerse.monopile_outer_diameter_in</code></td>
<td>m</td>
<td>cylinder diameter at corresponding locations</td>
</tr>
<tr>
<td><code>towerse.monopile_outfitting_factor</code></td>
<td></td>
<td>Multiplier that accounts for secondary structure mass inside of cylinder</td>
</tr>
<tr>
<td><code>towerse.monopile_s</code></td>
<td>1D array of the non-dimensional grid defined along the tower axis (0-tower base, 1-tower top)</td>
<td></td>
</tr>
<tr>
<td><code>towerse.mu_air</code></td>
<td>kg/(m*s)</td>
<td>dynamic viscosity of air</td>
</tr>
<tr>
<td><code>towerse.mu_water</code></td>
<td>kg/(m*s)</td>
<td>dynamic viscosity of water</td>
</tr>
<tr>
<td><code>towerse.nu_soil</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>towerse.outfitting_factor</code></td>
<td></td>
<td>Mass fraction added for outfitting</td>
</tr>
<tr>
<td><code>towerse.outfitting_full</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>towerse.painting_cost_rate</code></td>
<td>USD/m/m</td>
<td>Painting / surface finishing cost rate</td>
</tr>
<tr>
<td><code>towerse.post.Fz</code></td>
<td>N</td>
<td>point force in z-direction</td>
</tr>
<tr>
<td><code>towerse.post.Mxx</code></td>
<td>N*m</td>
<td>point moment about x-axis</td>
</tr>
<tr>
<td><code>towerse.post.Myy</code></td>
<td>N*m</td>
<td>point moment about y-axis</td>
</tr>
<tr>
<td><code>towerse.post.axial_stress</code></td>
<td>N/m**2</td>
<td>Axial stress in tower elements</td>
</tr>
<tr>
<td><code>towerse.post.fore_aft_freqs</code></td>
<td></td>
<td>Frequencies associated with mode shapes in the tower fore-aft direction</td>
</tr>
<tr>
<td><code>towerse.post.fore_aft_modes</code></td>
<td>6-degree polynomial coefficients of mode shapes in the tower fore-aft direction (x^2..x^6, no linear or constant term)</td>
<td></td>
</tr>
<tr>
<td>Variable</td>
<td>Units</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>towerse.post.freqs</td>
<td>Hz</td>
<td>Natural frequencies of the structure</td>
</tr>
<tr>
<td>towerse.post.global_buckling</td>
<td></td>
<td>Global buckling constraint. Should be &lt; 1 for feasibility. Includes safety</td>
</tr>
<tr>
<td>towerse.post.hoop_stress</td>
<td>N/m**2</td>
<td>Hoop stress in tower elements</td>
</tr>
<tr>
<td>towerse.post.shear_stress</td>
<td>N/m**2</td>
<td>Shear stress in tower elements</td>
</tr>
<tr>
<td>towerse.post.shell_buckling</td>
<td></td>
<td>Shell buckling constraint. Should be &lt; 1 for feasibility. Includes safety</td>
</tr>
<tr>
<td>towerse.post.side_side_freqs</td>
<td></td>
<td>Frequencies associated with mode shapes in the tower side-side direction</td>
</tr>
<tr>
<td>towerse.post.side_side_modes</td>
<td></td>
<td>6-degree polynomial coefficients of mode shapes in the tower side-side direction</td>
</tr>
<tr>
<td>towerse.post.post_freqs</td>
<td></td>
<td>Structural frequencies outputted from FEM calculation</td>
</tr>
<tr>
<td>towerse.post.post_deflection</td>
<td>m</td>
<td>Deflection of tower top in yaw-aligned +x direction</td>
</tr>
<tr>
<td>towerse.post.post_deflection_x</td>
<td>m</td>
<td>Deflection of tower top in yaw-aligned +x direction</td>
</tr>
<tr>
<td>towerse.post.turbine_F</td>
<td>N</td>
<td>Total force on tower+rna</td>
</tr>
<tr>
<td>towerse.post.turbine_M</td>
<td>N*m</td>
<td>Total x-moment on tower+rna measured at base</td>
</tr>
<tr>
<td>towerse.post.x_mode_freqs</td>
<td></td>
<td>Frequencies associated with mode shapes in the x-direction</td>
</tr>
<tr>
<td>towerse.post.x_mode_shapes</td>
<td></td>
<td>6-degree polynomial coefficients of mode shapes in the x-direction</td>
</tr>
<tr>
<td>towerse.post.y_mode_freqs</td>
<td></td>
<td>Frequencies associated with mode shapes in the y-direction</td>
</tr>
<tr>
<td>towerse.post.y_mode_shapes</td>
<td></td>
<td>6-degree polynomial coefficients of mode shapes in the y-direction</td>
</tr>
<tr>
<td>towerse.pre.Fx</td>
<td>N</td>
<td>point force in x-direction</td>
</tr>
<tr>
<td>towerse.pre.Fy</td>
<td>N</td>
<td>point force in y-direction</td>
</tr>
<tr>
<td>towerse.pre.Fz</td>
<td>N</td>
<td>point force in z-direction</td>
</tr>
<tr>
<td>towerse.pre.Mxx</td>
<td>N*m</td>
<td>point moment about x-axis</td>
</tr>
<tr>
<td>towerse.pre.Myy</td>
<td>N*m</td>
<td>point moment about y-axis</td>
</tr>
<tr>
<td>towerse.pre.Mzz</td>
<td>N*m</td>
<td>point moment about z-axis</td>
</tr>
<tr>
<td>towerse.pre.k_monopile</td>
<td>N/m</td>
<td>Stiffness BCs for ocean soil. Only used if monoflag input is True</td>
</tr>
<tr>
<td>towerse.pre.kidx</td>
<td></td>
<td>indices of z where external stiffness reactions should be applied.</td>
</tr>
<tr>
<td>towerse.pre.ktx</td>
<td>N/m</td>
<td>spring stiffness in theta_x-rotation</td>
</tr>
<tr>
<td>towerse.pre.kty</td>
<td>N/m</td>
<td>spring stiffness in theta_y-rotation</td>
</tr>
<tr>
<td>towerse.pre.ktz</td>
<td>N/m</td>
<td>spring stiffness in theta_z-rotation</td>
</tr>
<tr>
<td>towerse.pre.kx</td>
<td>N/m</td>
<td>spring stiffness in x-direction</td>
</tr>
<tr>
<td>towerse.pre.ky</td>
<td>N/m</td>
<td>spring stiffness in y-direction</td>
</tr>
<tr>
<td>towerse.pre.kz</td>
<td>N/m</td>
<td>spring stiffness in z-direction</td>
</tr>
<tr>
<td>towerse.pre.m</td>
<td>kg</td>
<td>added mass</td>
</tr>
<tr>
<td>towerse.pre.mlxx</td>
<td>kg*m**2</td>
<td>mass moment of inertia about some point p</td>
</tr>
<tr>
<td>towerse.pre.mlxy</td>
<td>kg*m**2</td>
<td>mass moment of inertia about some point p</td>
</tr>
<tr>
<td>towerse.pre.mlxz</td>
<td>kg*m**2</td>
<td>mass moment of inertia about some point p</td>
</tr>
<tr>
<td>towerse.pre.mlyy</td>
<td>kg*m**2</td>
<td>mass moment of inertia about some point p</td>
</tr>
<tr>
<td>towerse.pre.mlzy</td>
<td>kg*m**2</td>
<td>mass moment of inertia about some point p</td>
</tr>
<tr>
<td>towerse.pre.mlzz</td>
<td>kg*m**2</td>
<td>mass moment of inertia about some point p</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Variable</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>towerse.pre.midx</td>
<td></td>
<td>indices where added mass should be applied.</td>
</tr>
<tr>
<td>towerse.pre.mrholox</td>
<td>m</td>
<td>x-location of p relative to node</td>
</tr>
<tr>
<td>towerse.pre.mrholoy</td>
<td>m</td>
<td>y-location of p relative to node</td>
</tr>
<tr>
<td>towerse.pre.mrholoz</td>
<td>m</td>
<td>z-location of p relative to node</td>
</tr>
<tr>
<td>towerse.pre.plidx</td>
<td></td>
<td>indices where point loads should be applied.</td>
</tr>
<tr>
<td>towerse.pre.rna_F</td>
<td>N</td>
<td>rna force</td>
</tr>
<tr>
<td>towerse.pre.rna_M</td>
<td>N*m</td>
<td>rna moment</td>
</tr>
<tr>
<td>towerse.props.d</td>
<td>m</td>
<td>Sectional tower diameters</td>
</tr>
<tr>
<td>towerse.props.t</td>
<td>m</td>
<td>Sectional tower wall thicknesses</td>
</tr>
<tr>
<td>towerse.rho</td>
<td>kg/m**2</td>
<td>Density of the materials along the column sections.</td>
</tr>
<tr>
<td>towerse.rho_air</td>
<td>kg/m**3</td>
<td>Air density</td>
</tr>
<tr>
<td>towerse.rho_full</td>
<td>kg/m**3</td>
<td>Density of the materials along the tower sections.material density</td>
</tr>
<tr>
<td>towerse.rho_mat</td>
<td>kg/m**3</td>
<td>3D array of the density of the materials. For composites, this is the density of the laminate.</td>
</tr>
<tr>
<td>towerse.rho_water</td>
<td>kg/m**3</td>
<td>density of water</td>
</tr>
<tr>
<td>towerse.rna_I</td>
<td>kg*m**2</td>
<td>Moments about turbine main</td>
</tr>
<tr>
<td>towerse.rna_cg</td>
<td>m</td>
<td>Location of RNA center of mass relative to tower top</td>
</tr>
<tr>
<td>towerse.rna_mass</td>
<td>kg</td>
<td>Mass of RNA</td>
</tr>
<tr>
<td>towerse.sc_offst</td>
<td>m</td>
<td>offset from the sectional shear center</td>
</tr>
<tr>
<td>towerse.sec_loc</td>
<td></td>
<td>normalized sectional location</td>
</tr>
<tr>
<td>towerse.shearExp</td>
<td></td>
<td>shear exponent</td>
</tr>
<tr>
<td>towerse.sideside_iner</td>
<td>kg*m**3</td>
<td>sectional side-side inertia per unit length about the Y_G inertia axis</td>
</tr>
<tr>
<td>towerse.sideside_stff</td>
<td>N*m**3</td>
<td>Sectional side-side bending stiffness per unit length about the Y_E elastic axis</td>
</tr>
<tr>
<td>towerse.sigma_y</td>
<td>Pa</td>
<td>Isotropic yield strength of the materials along the column sections.</td>
</tr>
<tr>
<td>towerse.sigma_y_full</td>
<td>N/m**2</td>
<td>2yield stressyield stress</td>
</tr>
<tr>
<td>towerse.sigma_y_mat</td>
<td>Pa</td>
<td>2D array of the yield strength of the materials. Each row represents a material, the three columns represent Xt12, Xt13 and Xt23.</td>
</tr>
<tr>
<td>towerse.slope</td>
<td></td>
<td></td>
</tr>
<tr>
<td>towerse.soil.d0</td>
<td>m</td>
<td>diameter of base of tower</td>
</tr>
<tr>
<td>towerse.soil.k</td>
<td>N/m</td>
<td></td>
</tr>
<tr>
<td>towerse.soil.k_usr</td>
<td>N/m</td>
<td>User overrides of stiffness values. Use positive values and for rigid use np.inf. Order is x, theta_x, y, theta_y, z, theta_z</td>
</tr>
<tr>
<td>towerse.str_tw</td>
<td>deg</td>
<td>structural twist of section</td>
</tr>
<tr>
<td>towerse.structural_cost</td>
<td>USD</td>
<td>Mass of whole turbine except for mooring lines</td>
</tr>
<tr>
<td>towerse.structural_mass</td>
<td>kg</td>
<td></td>
</tr>
<tr>
<td>towerse.suctionpile_depth</td>
<td>m</td>
<td>shell thickness at corresponding locations</td>
</tr>
<tr>
<td>towerse.tc_offst</td>
<td>m</td>
<td>offset from the sectional tension center</td>
</tr>
<tr>
<td>towerse.tm.cylinder_I_base</td>
<td>kg*m**3</td>
<td>Mass moment of inertia of cylinder about base [xx yy zz xy xz yz]</td>
</tr>
<tr>
<td>towerse.tm.cylinder_center_of_mass</td>
<td>m</td>
<td>z position of center of mass of cylinder</td>
</tr>
<tr>
<td>towerse.tm.cylinder_cost</td>
<td>USD</td>
<td>Total cylinder cost</td>
</tr>
<tr>
<td>towerse.tm.cylinder_mass</td>
<td>kg</td>
<td>Total cylinder mass</td>
</tr>
<tr>
<td>towerse.tm.cylinder_section_center_of_mass</td>
<td>m</td>
<td>z position of center of mass of each can in the cylinder</td>
</tr>
<tr>
<td>towerse.tor_stff</td>
<td>N*m**3</td>
<td>Sectional torsional stiffness</td>
</tr>
<tr>
<td>towerse.tower.E</td>
<td>N/m**2</td>
<td>3 isotropic Youngs modulus of the materials along the column sections.</td>
</tr>
<tr>
<td>towerse.tower.Fx</td>
<td>N</td>
<td>point force in x-direction</td>
</tr>
</tbody>
</table>
Table 5.2 – continued from previous page

<table>
<thead>
<tr>
<th>Variable</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>towerse.tower.Fy</td>
<td>N</td>
<td>point force in y-direction</td>
</tr>
<tr>
<td>towerse.tower.Fz</td>
<td>N</td>
<td>point force in z-direction</td>
</tr>
<tr>
<td>towerse.tower.Fz_out</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>towerse.tower.G</td>
<td>N/m**2</td>
<td>isotropic shear modulus of the materials along the column sections.</td>
</tr>
<tr>
<td>towerse.tower.Mxx</td>
<td>N*m</td>
<td>point moment about x-axis</td>
</tr>
<tr>
<td>towerse.tower.Mxx_out</td>
<td>N*m</td>
<td></td>
</tr>
<tr>
<td>towerse.tower.Myy</td>
<td>N*m</td>
<td>point moment about y-axis</td>
</tr>
<tr>
<td>towerse.tower.Myy_out</td>
<td>N*m</td>
<td></td>
</tr>
<tr>
<td>towerse.tower.Mzz</td>
<td>N*m</td>
<td>point moment about z-axis</td>
</tr>
<tr>
<td>towerse.tower.Mzz_out</td>
<td>N*m</td>
<td></td>
</tr>
<tr>
<td>towerse.tower.Px</td>
<td>N/m</td>
<td>force per unit length in x-direction</td>
</tr>
<tr>
<td>towerse.tower.Py</td>
<td>N/m</td>
<td>force per unit length in y-direction</td>
</tr>
<tr>
<td>towerse.tower.Pz</td>
<td>N/m</td>
<td>force per unit length in z-direction</td>
</tr>
<tr>
<td>towerse.tower.Vx_out</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>towerse.tower.Vy_out</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>towerse.tower.axial_stress</td>
<td>N/m**2</td>
<td>axial stress in tower elements</td>
</tr>
<tr>
<td>towerse.tower.base_F</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>towerse.tower.base_M</td>
<td>N*m</td>
<td></td>
</tr>
<tr>
<td>towerse.tower.d</td>
<td>m</td>
<td>Sectional tower diameters</td>
</tr>
<tr>
<td>towerse.tower.f1</td>
<td>Hz</td>
<td></td>
</tr>
<tr>
<td>towerse.tower.f2</td>
<td>Hz</td>
<td></td>
</tr>
<tr>
<td>towerse.tower.freqs</td>
<td>Hz</td>
<td>Natural frequencies of the structure</td>
</tr>
<tr>
<td>towerse.tower.hoop_stress</td>
<td>N/m**2</td>
<td>hoop stress in tower elements</td>
</tr>
<tr>
<td>towerse.tower.hoop_stress_euro</td>
<td>N/m**2</td>
<td>2</td>
</tr>
<tr>
<td>towerse.tower.kidx</td>
<td></td>
<td>indices of z where external stiffness reactions should be applied.</td>
</tr>
<tr>
<td>towerse.tower.ktx</td>
<td>N/m</td>
<td>spring stiffness in theta_x-rotation</td>
</tr>
<tr>
<td>towerse.tower.kty</td>
<td>N/m</td>
<td>spring stiffness in theta_y-rotation</td>
</tr>
<tr>
<td>towerse.tower.ktz</td>
<td>N/m</td>
<td>spring stiffness in theta_z-rotation</td>
</tr>
<tr>
<td>towerse.tower.kx</td>
<td>N/m</td>
<td>spring stiffness in x-direction</td>
</tr>
<tr>
<td>towerse.tower.ky</td>
<td>N/m</td>
<td>spring stiffness in y-direction</td>
</tr>
<tr>
<td>towerse.tower.kz</td>
<td>N/m</td>
<td>spring stiffness in z-direction</td>
</tr>
<tr>
<td>towerse.tower.m</td>
<td>kg</td>
<td>added mass</td>
</tr>
<tr>
<td>towerse.tower.mlxx</td>
<td>kg*m**4</td>
<td>mass moment of inertia about some point p</td>
</tr>
<tr>
<td>towerse.tower.mlxy</td>
<td>kg*m**4</td>
<td>mass moment of inertia about some point p</td>
</tr>
<tr>
<td>towerse.tower.mlxz</td>
<td>kg*m**4</td>
<td>mass moment of inertia about some point p</td>
</tr>
<tr>
<td>towerse.tower.mlyy</td>
<td>kg*m**4</td>
<td>mass moment of inertia about some point p</td>
</tr>
<tr>
<td>towerse.tower.mlzy</td>
<td>kg*m**4</td>
<td>mass moment of inertia about some point p</td>
</tr>
<tr>
<td>towerse.tower.mlzz</td>
<td>kg*m**4</td>
<td>mass moment of inertia about some point p</td>
</tr>
<tr>
<td>towerse.tower.mass</td>
<td>kg</td>
<td>added mass</td>
</tr>
<tr>
<td>towerse.tower.midx</td>
<td></td>
<td>indices where added mass should be applied.</td>
</tr>
<tr>
<td>towerse.tower.mrhox</td>
<td>m</td>
<td>x-location of p relative to node</td>
</tr>
<tr>
<td>towerse.tower.mrhoy</td>
<td>m</td>
<td>y-location of p relative to node</td>
</tr>
<tr>
<td>towerse.tower.mrhoz</td>
<td>m</td>
<td>z-location of p relative to node</td>
</tr>
<tr>
<td>towerse.tower.plidx</td>
<td></td>
<td>indices where point loads should be applied.</td>
</tr>
<tr>
<td>towerse.tower.qdyn</td>
<td>N/m**1</td>
<td>Dynamic pressure</td>
</tr>
<tr>
<td>towerse.tower.rho</td>
<td>kg/m**3</td>
<td>density of the materials along the column sections.</td>
</tr>
<tr>
<td>towerse.tower.shear_stress</td>
<td>N/m**2</td>
<td>shear stress in tower elements</td>
</tr>
<tr>
<td>towerse.tower.t</td>
<td>m</td>
<td>Sectional tower wall thicknesses</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Variable</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>towerse.tower.top_deflection</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>towerse.tower.x_mode_freqs</td>
<td></td>
<td>Frequencies associated with mode shapes in the x-direction</td>
</tr>
<tr>
<td>towerse.tower.x_mode_shapes</td>
<td></td>
<td>6-degree polynomial coefficients of mode shapes in the x-direction</td>
</tr>
<tr>
<td>towerse.tower.y_mode_freqs</td>
<td></td>
<td>Frequencies associated with mode shapes in the y-direction</td>
</tr>
<tr>
<td>towerse.tower.y_mode_shapes</td>
<td></td>
<td>6-degree polynomial coefficients of mode shapes in the x-direction</td>
</tr>
<tr>
<td>towerse.tower.z</td>
<td>m</td>
<td>location along cylinder. start at bottom and go to top</td>
</tr>
<tr>
<td>towerse.tower_I_base</td>
<td>kg*m</td>
<td>Moments about tower main</td>
</tr>
<tr>
<td>towerse.tower_center_of_mass</td>
<td>m</td>
<td>z-position of center of tower mass</td>
</tr>
<tr>
<td>towerse.tower_cost</td>
<td>USD</td>
<td>Tower cost</td>
</tr>
<tr>
<td>towerse.tower.Foundation_height</td>
<td>m</td>
<td>Scalar of the tower height computed along the z axis.</td>
</tr>
<tr>
<td>towerse.tower_height</td>
<td></td>
<td></td>
</tr>
<tr>
<td>towerse.layer_materials</td>
<td>Unavailable</td>
<td>1D array of the names of the materials of each layer modeled in the tower structure.</td>
</tr>
<tr>
<td>towerse.layer_thickness</td>
<td>m</td>
<td>2D array of the thickness of the layers of the tower structure. The first dimension represents each layer, the second dimension represents each piecewise-constant entry of the tower sections.</td>
</tr>
<tr>
<td>towerse.tower_mass</td>
<td>kg</td>
<td>Mass of tower</td>
</tr>
<tr>
<td>towerse.tower_outer_diameter</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>towerse.tower_outer_diameter_in</td>
<td>m</td>
<td>cylinder diameter at corresponding locations</td>
</tr>
<tr>
<td>towerse.tower_outfitting_factor</td>
<td></td>
<td>Multiplier that accounts for secondary structure mass inside of cylinder</td>
</tr>
<tr>
<td>towerse.tower_s</td>
<td></td>
<td>1D array of the non-dimensional grid defined along the tower axis (0-tower base, 1-tower top)</td>
</tr>
<tr>
<td>towerse.tower_section_center_of_mass</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>towerse.tower_section_height</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>towerse.tower_wall_thickness</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>towerse.transition_piece_I</td>
<td>kg*m</td>
<td>**2</td>
</tr>
<tr>
<td>towerse.transition_piece_cost</td>
<td>USD</td>
<td>Cost of transition piece</td>
</tr>
<tr>
<td>towerse.transition_piece_height</td>
<td>m</td>
<td>height of transition piece above water line</td>
</tr>
<tr>
<td>towerse.transition_piece_mass</td>
<td>kg</td>
<td>point mass of transition piece</td>
</tr>
<tr>
<td>towerse.turbine_I_base</td>
<td>kg*m</td>
<td>**2</td>
</tr>
<tr>
<td>towerse.turbine_center_of_mass</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>towerse.turbine_mass</td>
<td>kg</td>
<td></td>
</tr>
<tr>
<td>towerse.tw_iner</td>
<td>deg</td>
<td>inertial twist of section</td>
</tr>
<tr>
<td>towerse.unit_cost</td>
<td>USD/kg</td>
<td>Unit costs of the materials along the column sections.</td>
</tr>
<tr>
<td>towerse.unit_cost_full</td>
<td>USD/kg</td>
<td>Raw material cost: steel $1.1/kg, aluminum $3.5/kg</td>
</tr>
<tr>
<td>towerse.unit_cost_mat</td>
<td>USD/D</td>
<td>1D array of the unit costs of the materials.</td>
</tr>
<tr>
<td>towerse.water_depth</td>
<td>m</td>
<td>water depth</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Variable</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>towerse.wave.A</td>
<td>m/s²</td>
<td>magnitude of wave acceleration</td>
</tr>
<tr>
<td>towerse.wave.U</td>
<td>m/s</td>
<td>magnitude of wind speed</td>
</tr>
<tr>
<td>towerse.wave.Uc</td>
<td>m/s</td>
<td>mean current speed</td>
</tr>
<tr>
<td>towerse.wave.V</td>
<td>m/s</td>
<td></td>
</tr>
<tr>
<td>towerse.wave.W</td>
<td>m/s</td>
<td></td>
</tr>
<tr>
<td>towerse.wave.p</td>
<td>N/m²</td>
<td>pressure oscillation</td>
</tr>
<tr>
<td>towerse.wave.phase_speed</td>
<td>m</td>
<td>location along cylinder. start at bottom and go to top</td>
</tr>
<tr>
<td>towerse.wave.z</td>
<td>m</td>
<td>location along cylinder. start at bottom and go to top</td>
</tr>
<tr>
<td>towerse.waveLoads.A</td>
<td>m/s²</td>
<td>magnitude of wave acceleration</td>
</tr>
<tr>
<td>towerse.waveLoads.U</td>
<td>m/s</td>
<td>magnitude of wind speed</td>
</tr>
<tr>
<td>towerse.waveLoads.d</td>
<td>m</td>
<td>Sectional tower diameters</td>
</tr>
<tr>
<td>towerse.waveLoads.p</td>
<td>N/m²</td>
<td>pressure oscillation</td>
</tr>
<tr>
<td>towerse.waveLoads.waveLoads_Px</td>
<td>N/m</td>
<td>distributed loads, force per unit length in x-direction</td>
</tr>
<tr>
<td>towerse.waveLoads.waveLoads_Py</td>
<td>N/m</td>
<td>distributed loads, force per unit length in y-direction</td>
</tr>
<tr>
<td>towerse.waveLoads.waveLoads_Pz</td>
<td>N/m</td>
<td>distributed loads, force per unit length in z-direction</td>
</tr>
<tr>
<td>towerse.waveLoads.waveLoads_beta</td>
<td>deg</td>
<td>wind/wave angle relative to inertia c.s.</td>
</tr>
<tr>
<td>towerse.waveLoads.waveLoads_d</td>
<td>m</td>
<td>corresponding diameters</td>
</tr>
<tr>
<td>towerse.waveLoads.waveLoads Pt</td>
<td>N/m²</td>
<td></td>
</tr>
<tr>
<td>towerse.waveLoads.waveLoads_qdyn</td>
<td>N/m²</td>
<td>Dynamic pressure</td>
</tr>
<tr>
<td>towerse.waveLoads.z</td>
<td>m</td>
<td>corresponding heights</td>
</tr>
<tr>
<td>towerse.wind.U</td>
<td>m/s</td>
<td>magnitude of wind speed</td>
</tr>
<tr>
<td>towerse.wind.Uref</td>
<td>m/s</td>
<td>reference wind speed (usually at hub height)</td>
</tr>
<tr>
<td>towerse.wind.z</td>
<td>m</td>
<td>location along cylinder. start at bottom and go to top</td>
</tr>
<tr>
<td>towerse.wind Loads.U</td>
<td>m/s</td>
<td>magnitude of wind speed</td>
</tr>
<tr>
<td>towerse.windLoads.d</td>
<td>m</td>
<td>Sectional tower diameters</td>
</tr>
<tr>
<td>towerse.windLoads.Px</td>
<td>N/m</td>
<td>distributed loads, force per unit length in x-direction</td>
</tr>
<tr>
<td>towerse.windLoads.Py</td>
<td>N/m</td>
<td>distributed loads, force per unit length in y-direction</td>
</tr>
<tr>
<td>towerse.windLoads.Pz</td>
<td>N/m</td>
<td>distributed loads, force per unit length in z-direction</td>
</tr>
<tr>
<td>towerse.windLoads.beta</td>
<td>deg</td>
<td>wind/wave angle relative to inertia c.s.</td>
</tr>
<tr>
<td>towerse.windLoads.d</td>
<td>m</td>
<td>corresponding diameters</td>
</tr>
<tr>
<td>towerse.windLoads.qdyn</td>
<td>N/m²</td>
<td>Dynamic pressure</td>
</tr>
<tr>
<td>towerse.windLoads.z</td>
<td>m</td>
<td>corresponding heights</td>
</tr>
</tbody>
</table>

continues on next page
Table 5.2 – continued from previous page

<table>
<thead>
<tr>
<th>Variable</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>towerse.wind_reference_height</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>towerse.wind_z0</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>towerse.yaw</td>
<td>deg</td>
<td>yaw angle</td>
</tr>
<tr>
<td>towerse.z_full</td>
<td>m</td>
<td>z-coordinates of section nodes</td>
</tr>
<tr>
<td>towerse.z_param</td>
<td>m</td>
<td>parameterized locations along tower, linear lofting between</td>
</tr>
<tr>
<td>towerse.z_start</td>
<td>m</td>
<td></td>
</tr>
<tr>
<td>wt_class.V_extreme1</td>
<td>m/s</td>
<td></td>
</tr>
<tr>
<td>wt_class.V_extreme50</td>
<td>m/s</td>
<td></td>
</tr>
<tr>
<td>wt_class.V_mean</td>
<td>m/s</td>
<td></td>
</tr>
<tr>
<td>wt_class.V_mean_overwrite</td>
<td></td>
<td>overwrite value for mean velocity for using user defined CDFs</td>
</tr>
<tr>
<td>wt_class.turbine_class</td>
<td>Unavailable</td>
<td>IEC turbine class</td>
</tr>
</tbody>
</table>

5.7 Examples

All of these case studies below can be found in the examples directory. For the ones that exercise just a single WISDEM module, it is easiest (for now) to call them directly from a Python script, with code examples provided through the module-specific documentation. For the case studies that exercise multiple modules together, they are described in-depth in the links below.

5.7.1 1. NREL Cost and Scaling Model Example

List of Examples

- 1. NREL Cost and Scaling Model Example
  - Turbine Component Masses Using the NREL_CSM (2015)
  - Turbine Component Masses and Costs Using the NREL_CSM (2015)
  - Turbine Component Costs Using the NREL_CSM (2015)
  - Parametric Studies Using the NREL_CSM (2015)

Turbine Component Masses Using the NREL_CSM (2015)

As an example of estimating turbine component masses (only) using the 2015 update of the NREL Cost and Scaling Model (CSM), let us simulate the NREL 5MW Reference Model [JBMS09].

The first step is to import OpenMDAO and the model itself:

```python
import openmdao.api as om
from wisdem.nrelcsm.nrel_csm_mass_2015 import nrel_csm_mass_2015
```
Next, we initialize an OpenMDAO instance and assign the model to be the \texttt{nrel\_csm\_mass\_2015} module. The \texttt{setup()} command completes the high-level configuration readies the model for variable input:

```python
# OpenMDAO Problem instance
prob = om.Problem()
prob.model = nrel_csm_mass_2015()
prob.setup()
```

The turbine scaling relies on key turbine configuration parameters. These filter down to the individual component models through the rotor, nacelle, and tower as described on the \textit{Theory} page. The variables are set like a Python dictionary:

```python
# Initialize variables for NREL CSM
prob["machine_rating"] = 5000.0
prob["rotor_diameter"] = 126.0
prob["turbine_class"] = 2
prob["hub_height"] = 90.0
prob["blade_number"] = 3
prob["blade_has_carbon"] = False
prob["max_tip_speed"] = 80.0
prob["max_efficiency"] = 0.90
prob["main_bearing_number"] = 2
prob["crane"] = True
```

We can now run the model to compute the component masses:

```python
# Evaluate the model
prob.run_model()
```

We can then print out an exhaustive listing of the inputs and outputs to each submodule:

```python
# Print all intermediate inputs and outputs to the screen
prob.model.list_inputs(units=True)
prob.model.list_outputs(units=True)
```

The final lines highlight the mass breakdown summaries:

```plaintext
>>> turbine
>>> hub_system_mass [47855.49446548] kg
>>> rotor_mass [95109.93575675] kg
>>> nacelle_mass [165460.38774975] kg
>>> turbine_mass [442906.80408368] kg
```

See the full source for this example on Github.
Turbine Component Masses and Costs Using the NREL_CSM (2015)

It is often desired to estimate the component costs and cost of energy of a hypothetical turbine, not just the component masses in the previous example. To do so, all that is required is import the full 2015 Cost and Scaling model with:

```python
import openmdao.api as om
from wisdem.nrelcsm.nrel_csm_mass_2015 import nrel_csm_2015
```

The OpenMDAO problem instance must also be assigned this model (`nrel_csm_2015`):

```python
# OpenMDAO Problem instance
prob = om.Problem()
prob.model = nrel_csm_2015()
prob.setup()
```

The model inputs remain the same:

```python
# Initialize variables for NREL CSM
prob["machine_rating"] = 5000.0
prob["rotor_diameter"] = 126.0
prob["turbine_class"] = 2
prob["hub_height"] = 90.0
prob["blade_number"] = 3
prob["blade_has_carbon"] = False
prob["max_tip_speed"] = 80.0
prob["max_efficiency"] = 0.90
prob["main_bearing_number"] = 2
prob["crane"] = True
```

We can now run the model to compute the component masses and costs:

```python
# Evaluate the model
prob.run_model()
```

Then we can again print out an exhaustive listing of the inputs and outputs:

```python
# Print all intermediate inputs and outputs to the screen
prob.model.list_inputs(units=True)
prob.model.list_outputs(units=True)
```

The final screen output is:

```python
>>> turbine_c
>>> turbine_mass_tcc [434686.14646457] kg
>>> turbine_cost [3543676.12253719] USD
>>> turbine_cost_kW [708.73522451] USD/kW
```

See the full source for this example on Github.
Turbine Component Costs Using the NREL_CSM (2015)

As an example of estimating turbine component costs (only), if the component masses are already known, using the 2015 update of the NREL Cost and Scaling Model (CSM), let us simulate the NREL 5MW Reference Model [JBMS09].

The first step is to import OpenMDAO and the model itself:

```python
import openmdao.api as om
from wisdem.nrelcsm.nrel_csm_cost_2015 import Turbine_CostsSE_2015
```

Next, we initialize an OpenMDAO instance and assign the model to be the `Turbine_CostsSE_2015` module. This module has a configuration option to print to the screen a nicely formatted summary of the outputs, which is accessed by setting `verbosity=True`. The `setup()` command completes the high-level configuration readies the model for variable input:

```python
# OpenMDAO Problem instance
prob = om.Problem()
prob.model = Turbine_CostsSE_2015(verbosity=True)
prob.setup()
```

The turbine scaling relies on key turbine configuration parameters. These filter down to the individual component models through the rotor, nacelle, and tower as described on the Theory page. The variables are set like a Python dictionary:

```python
# Initialize variables for NREL CSM
prob["machine_rating"] = 5000.0
prob["blade_number"] = 3
prob["crane"] = True
prob["main_bearing_number"] = 2
```

Next we set the individual component masses. These values might come from publicly available data, the other WISDEM modules, or through parametric study. In this example, we grab the masses computed in the previous example:

```python
# Component masses
prob["blade_mass"] = 15751.48043042
prob["hub_mass"] = 37548.40498997
prob["pitch_system_mass"] = 9334.08947551
prob["spinner_mass"] = 973.0
prob["lss_mass"] = 20568.96284886
prob["main_bearing_mass"] = 2245.41649102
prob["gearbox_mass"] = 43468.32086769
prob["hss_mass"] = 994.7
prob["generator_mass"] = 14900.0
prob["bedplate_mass"] = 41765.26095285
prob["yaw_mass"] = 12329.96247921
prob["hvac_mass"] = 400.0
prob["cover_mass"] = 6836.69
prob["tower_mass"] = 182336.48057717
prob["transformer_mass"] = 11485.0
prob["platforms_mass"] = 8220.65761911
```

We can now run the model to compute the component costs:
# Evaluate the model
prob.run_model()

A formatted tabular output is printed to the screen:

```
>>> Computation of costs of the main turbine components from TurbineCostSE
>>> Blade cost 229.972 k USD mass 15751.480 kg
>>> Pitch system cost 206.283 k USD mass 9334.089 kg
>>> Hub cost 146.439 k USD mass 37548.405 kg
>>> Spinner cost 10.800 k USD mass 973.000 kg
>>>---------------------------------------------------------------------
>>> Rotor cost 1053.437 k USD mass 95109.936 kg
>>>---------------------------------------------------------------------
>>> LSS cost 244.771 k USD mass 20568.963 kg
>>> Main bearing cost 10.104 k USD mass 2245.416 kg
>>> Gearbox cost 560.741 k USD mass 43468.321 kg
>>> HSS cost 6.764 k USD mass 994.700 kg
>>> Generator cost 184.760 k USD mass 14900.000 kg
>>> Bedplate cost 121.119 k USD mass 41765.261 kg
>>> Yaw system cost 102.339 k USD mass 12329.962 kg
>>> HVAC cost 49.600 k USD mass 400.000 kg
>>> Nacelle cover cost 38.969 k USD mass 6836.690 kg
>>> Electr connection cost 209.250 k USD
>>> Controls cost 105.750 k USD
>>> Other main frame cost 101.273 k USD
>>> Transformer cost 215.918 k USD mass 11485.000 kg
>>> Converter cost 0.000 k USD mass 0.000 kg
>>>---------------------------------------------------------------------
>>> Nacelle cost 1961.463 k USD mass 157239.730 kg
>>>---------------------------------------------------------------------
>>> Tower cost 528.776 k USD mass 182336.481 kg
>>>---------------------------------------------------------------------
>>>---------------------------------------------------------------------
>>> Turbine cost 3543.676 k USD mass 434686.146 kg
>>> Turbine cost per kW 708.735 k USD/kW
>>> # Print all intermediate inputs and outputs to the screen
prob.model.list_inputs(units=True)
prob.model.list_outputs(units=True)
```

We can also print out an exhaustive listing of the inputs and outputs to each submodule:

```
# Print all intermediate inputs and outputs to the screen
prob.model.list_inputs(units=True)
prob.model.list_outputs(units=True)
```

See the full source for this example on Github.
Parametric Studies Using the NREL_CSM (2015)

The simplicity and rapid execution of the NREL CSM makes it well suited for parametric studies. This example runs approximately 6000 points in a Design of Experiment (DoE) parametric analysis varying machine rating, rotor diameter (and thereby hub_height), the blade mass scaling exponent, the average wind speed, and wind shear.

As above, the first step is to import OpenMDAO and the model itself, but we will also need other Python and WISDEM packages. In this case, the NumPy library and the annual energy production (AEP) estimator from the older (~2010) CSM code:

```python
import numpy as np
import openmdao.api as om
from wisdem.nrelcsm.nrel_csm_mass_2015 import nrel_csm_2015
from wisdem.nrelcsm.nrel_csm_orig import aep_csm
```

Next, we initialize an OpenMDAO instance and assign the model to be the nrel_csm_2015 module. We also initialize an instance of the AEP model:

```python
# OpenMDAO Problem instance
prob = om.Problem()
prob.model = nrel_csm_2015()
prob.setup()

# Initialize AEP calculator from CSM model
aep_instance = aep_csm()
```

The CSM model initialization is abbreviated here because some of the variables will be modified within the DoE loop. The remaining ones are:

```python
# Initialize variables for NREL CSM
prob["turbine_class"] = -1  # Sets blade mass based on user input, not auto-determined
prob["blade_number"] = 3
prob["blade_has_carbon"] = False
prob["max_tip_speed"] = max_tip_speed = 90.0
prob["max_efficiency"] = max_efficiency = 0.9
prob["main_bearing_number"] = 2
prob["crane"] = True
```

Note that the turbine_class variable has been set to -1 to allow us to override the blade_mass_exp value as described in the Source Documentation documentation. Also, two variables are jointly assigned to local Python variables for use in the AEP estimation. The AEP model requires a number of other inputs to define the turbine power curve. To keep things simple, we focus on a single turbine, and ignore many of the other losses and options:

```python
# Initialize variables for AEP calculation
opt_tsr = 9.0  # Optimal tip speed ratio
max_Cp = 0.47  # Max (aerodynamic) power coefficient
max.Ct = 0.8  # Max thrust coefficient
max_eff = 0.95  # Drivetrain efficiency
cut_in = 4.0  # m/s
cut_out = 25.0  # m/s
altitude = 0.0  # m (assume sea level)
rho_air = 1.225  # kg/m^3
array_losses = 0.0  # Focusing on single turbine
```

(continues on next page)
availability = 1.0 # Assume 100% uptime
soiling_losses = 0.0 # Ignore this for now
turbine_number = 1 # Focus on single turbine
weibull_k = 2.0 # Weibull shape parameter

Next we define our parametric axes using NumPy's arange function that provides evenly spaced intervals:

```python
# Set Design of Experiment (DoE) parametric sweep bounds
machine_rating = 1e3 * np.arange(2.0, 10.1, 1.0) # kW
rotor_diameter = np.arange(60, 201.0, 20.0) # m
blade_mass_exp = np.arange(2.1, 2.41, 0.1) # Relationship between blade length and mass
shear_exp = np.arange(0.1, 0.31, 0.1)
wind_speed = np.arange(5.0, 11.1, 1.0) # m/s
```

To run our n-dimensional DOE, we do a “tensor” or “outer” multiplication of the arrays using NumPy’s meshgrid, but then flatten them into 1-D vectors for easy enumeration of all of the scenarios:

```python
# Enumerate DoE through tensor multiplication and then flatten to get vector of all of the runs
[Rating, Diameter, Bladeexp, Shear, WindV] = np.meshgrid(
    machine_rating, rotor_diameter, blade_mass_exp, shear_exp, wind_speed)

# Shift to flattened arrays to run through each scenario easily
Rating = Rating.flatten()
Diameter = Diameter.flatten()
Bladeexp = Bladeexp.flatten()
Shear = Shear.flatten()
WindV = WindV.flatten()

# Initialize output containers
tcc = np.zeros(Rating.shape)
aep = np.zeros(Rating.shape)
```

We are now ready to loop through all of the points, and evaluate the CSM model and AEP model:

```python
# Calculation loop
npts = Rating.size
print("Running, ", npts, " points in the parametric study")
for k in range(npts):
    # Populate remaining NREL CSM inputs for this iteration
    prob["machine_rating"] = Rating[k]
    prob["rotor_diameter"] = Diameter[k]
    prob["blade_user_exp"] = Bladeexp[k]
    prob["hub_height"]= hub_height = 0.5 * Diameter[k] + 30.0

    # Compute Turbine capital cost using the NREL CSM (2015) and store the result
    prob.run_model()
    tcc[k] = float(prob["turbine_cost_kW"])

    # Compute AEP using original CSM function and store result
    aep_instance.compute(
```

5.7. Examples
To store for later postprocessing, we save everything into a large csv-file. Flattening the arrays makes this fairly straightforward using NumPy’s concatenation shortcuts:

```python
alldata = np.c_[Rating, Diameter, Bladeexp, Shear, WindV, tcc, aep]
header = "Rating [kW], Rotor Diam [m], Blade Mass Exp, Shear Exp, Wind Vel [m/s], TCC [USD/kW], AEP [kWh/yr]"
np.savetxt("parametric_scaling.csv", alldata, delimiter="", header=header)
```

See the full source for this example on Github.
### 5.7.2 2. Running the NREL 5-MW and IEA Wind 15-MW Reference Wind Turbines

The next example involves running two reference turbine examples, the NREL 5-MW land-based and IEA Wind 15-MW offshore fixed-bottom turbines. There are multiple ways to run WISDEM, each is perfectly valid and users should adopt whatever approach they are most comfortable with.

#### Calling WISDEM

**Option 1. Use the GUI**

Installing WISDEM creates a `wisdem` command that should be available at the command prompt (a terminal prompt on Linux / Mac machines or the Anaconda PowerShell prompt for Windows). Just typing `wisdem` at the command line opens the GUI. From there, you can use the dialogue menus to load in the yaml input files and run WISDEM.

**Option 2. Pass YAML-files directly to the `wisdem` command**

Installing WISDEM creates a `wisdem` command that should be available at the command prompt (a terminal prompt on Linux / Mac machines or the Anaconda PowerShell prompt for Windows). You can pass that command the three input yaml-files in order to run WISDEM.

```
$ wisdem nrel5mw.yaml modeling_options.yaml analysis_options.yaml
```

Alternatively, you can create a summary WISDEM file that points to each file,

```
$ wisdem nrel5mw_driver.yaml
```

Where the contents of `nrel5mw_driver.yaml` are,

```
geometry_file: nrel5mw.yaml
modeling_file: modeling_options.yaml
analysis_file: analysis_options.yaml
```

Note that to run the IEA Wind 15-MW reference wind turbine, simply substitute the file, `IEA-15-240-RWT.yaml`, in as the geometry file. The `modeling_options.yaml` and `analysis_options.yaml` file can remain the same.

**Option 3. Call WISDEM from a Python Script**

For those users who are comfortable with Python scripting, the WISDEM yaml input files can also be passed as path names to the main WISDEM function in this way.

```
$ python nrel5mw_driver.py
```

Where the contents of `nrel5mw_driver.py` are,

```python
import os
from wisdem import run_wisdem

## File management
mydir = os.path.dirname(os.path.realpath(__file__))  # get path to this file
fname_wt_input = mydir + os.sep + "nrel5mw.yaml"
```

(continues on next page)
fname_modeling_options = mydir + os.sep + "modeling_options.yaml"
fname_analysis_options = mydir + os.sep + "analysis_options.yaml"

wt_opt, analysis_options, opt_options = run_wisdem(fname_wt_input, fname_modeling_options, fname_analysis_options)

Screen Output

Successfully running WISDEM should show the following screen output for the NREL 5-MW reference wind turbine:

==
wt
==
NL: NLBGS Converged in 2 iterations

#===============================================
Objectives
Turbine AEP: 24.0796812417 GWh
Blade Mass: 16403.6823269407 kg
LCOE: 49.4740771484 USD/MWh
Tip Defl.: 4.1950872846 m
#===============================================

And this output for the IEA Wind 15-MW reference wind turbine:

==
wt
==

WARNING: the blade cost model is used beyond its applicability range. No team can limit the main mold cycle time to 24 hours. 100 workers are assumed at the low-pressure mold, but this is incorrect.
WARNING: the blade cost model is used beyond its applicability range. No team can limit the main mold cycle time to 24 hours. 100 workers are assumed at the high-pressure mold, but this is incorrect.
WARNING: the blade cost model is used beyond its applicability range. No team can limit the assembly cycle time to 24 hours. 100 workers are assumed at the assembly line, but this is incorrect.

| ==
| wt driveseve
| ==
| NL: NLBGS Converged in 2 iterations
| WARNING: the blade cost model is used beyond its applicability range. No team can limit the main mold cycle time to 24 hours. 100 workers are assumed at the low-pressure mold, but this is incorrect.
| WARNING: the blade cost model is used beyond its applicability range. No team can limit the main mold cycle time to 24 hours. 100 workers are assumed at the high-pressure mold, but this is incorrect.
| WARNING: the blade cost model is used beyond its applicability range. No team can limit the assembly cycle time to 24 hours. 100 workers are assumed at the assembly line, but this is incorrect.
|
Some helpful summary information is printed to the screen. More detailed output can be found in the outputs directory. This creates output files that can be read-in by Matlab, Numpy, Python pickle-package, and Excel. These files have the complete list of all WISDEM variables (with extended naming based on their OpenMDAO Group hierarchy) and the associated values. An output yaml-file is also written, in case any input values were altered in the course of the analysis.

```bash
$ ls -1 outputs
refturb_output.mat
refturb_output.npz
refturb_output.pkl
refturb_output.xlsx
refturb_output.yaml
refturb_output-modeling.yaml
refturb_output-analysis.yaml
```

<table>
<thead>
<tr>
<th>Extension</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.mat</td>
<td>MatLab output format</td>
</tr>
<tr>
<td>.npz</td>
<td>Archive of NumPy arrays</td>
</tr>
<tr>
<td>.pkl</td>
<td>Python Pickle format</td>
</tr>
<tr>
<td>.xlsx</td>
<td>Microsoft Excel format</td>
</tr>
<tr>
<td>.yaml</td>
<td>YAML format</td>
</tr>
</tbody>
</table>

As an example, the `sample_plot.py` script plots Axial Induction versus Blade Nondimensional Span by extracting the values from the Python pickle file. The script content is:

```python
import matplotlib.pyplot as plt
from wisdem.glue_code.runWISDEM import load_wisdem

refturb, _, _ = load_wisdem("outputs/refturb_output.pkl")
xs = refturb["wt.wt_init.blade.outer_shape_bem.compute_blade_outer_shape_bem.s_default"]
ys = refturb["wt.rp.powercurve.compute_power_curve.ax_induct_regII"]
fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(10, 5))
am.plot(xs, ys)
am.set_xlabel("Blade Nondimensional Span [-]")
am.set_ylabel("Axial Induction [-]"")
plt.show()
```
This script generates the following plot:

![Plot](image)

### 5.7.3 3. Blade Optimization Example

This example walks through a blade optimization problem with increasing complexity. All of the iterations use the same geometry input file, `BAR0.yaml`, which describes a baseline design from the NREL-Sandia Big Adaptive Rotor (BAR) project. This blade uses glass fiber-reinforced polymer in the spar cap design. The same `modeling_options.yaml` file is also common to all iterations and shows that all modules are called, the airfoil polars are discretized at 200 angles of attack, etc. The example file runs four cases one after the other for testing purposes. To run the cases one by one, make sure to comment out all cases at lines 15-18 except the case that should run.

#### Baseline Design

Whenever conducting a design optimization, it is helpful to first run the starting point design and evaluate the baseline performance. The file, `analysis_options_no_opt.yaml`, does not have any optimization variables activated and is meant for this purpose. Outputs are generated in the `outputs` directory.

```bash
$ wisdem BAR0.yaml modeling_options.yaml analysis_options_no_opt.yaml
```

#### Simple Aerodynamic Optimization

The file, `analysis_options_aero.yaml`, is used first to run a blade twist optimization. This is activated by turning on the appropriate design variable flags in the file.

We are also setting the number of spline control points at 8 and the maximum decrease and increase that the optimizer can apply to the twist (in radians) at each evenly spaced control point along the blade span. We also need to set the objective function to be AEP with,

```
merit_figure: AEP
```
To better guide the optimization, we activate a stall margin constraint using the same flag type of setting, with a value of $5^\circ \text{deg} \approx 0.087 \text{rad}$.

The maximum iteration limit currently used in the file is 2, to keep the examples short. However, if you want to see more progress in the optimization, change the following lines from:

```
optimization:
    flag: True         # Flag to enable optimization
    tol: 1.e-3         # Optimality tolerance
    max_major_iter: 10 # Maximum number of major design iterations (SNOPT)
    max_minor_iter: 100# Maximum number of minor design iterations (SNOPT)
    max_iter: 2        # Maximum number of iterations (SLSQP)
    solver: SLSQP      # Optimization solver. Other options are 'SLSQP' - 'CONMIN'
    step_size: 1.e-3   # Step size for finite differencing
```

.. code:: python

To:

```
optimization:
    flag: True         # Flag to enable optimization
    tol: 1.e-3         # Optimality tolerance
    max_iter: 10       # Maximum number of minor design iterations
    solver: SLSQP      # Optimization solver. Other options are 'SLSQP' - 'CONMIN'
    step_size: 1.e-3   # Step size for finite differencing
```

Now to run the optimization we do,

```
$ wisdem BAR0.yaml modeling_options.yaml analysis_options_aero.yaml
```

or we comment out lines 16, 17, and 18 in blade_driver.py and we do:

```
$ python blade_driver.py
```

or to run in parallel using multiple CPU cores (Mac or Linux only):

```
$ mpirun -np 4 python blade_driver.py
```

where the `blade_driver.py` script is:

```
import os

from wisdem import run_wisdem
from wisdem.commonse.mpi_tools import MPI
from wisdem.postprocessing.compare_designs import run

mydir = os.path.dirname(os.path.realpath(__file__)) # get path to this file
fname_wt_input = mydir + os.sep + "BAR0.yaml"
fname_modeling_options = mydir + os.sep + "modeling_options.yaml"
fname_analysis_options_no_opt = mydir + os.sep + "analysis_options_no_opt.yaml"
fname_analysis_options_aero = mydir + os.sep + "analysis_options_aero.yaml"
fname_analysis_options_struct = mydir + os.sep + "analysis_options_struct.yaml"
fname_analysis_options_aerostruct = mydir + os.sep + "analysis_options_aerostruct.yaml"

wt_opt, modeling_options, analysis_options = run_wisdem(fname_wt_input, fname_modeling_options, fname_analysis_options_no_opt)
wt_opt, modeling_options, analysis_options = run_wisdem(fname_wt_input, fname_modeling_options, fname_analysis_options_aero)
wt_opt, modeling_options, analysis_options = run_wisdem(fname_wt_input, fname_modeling_options, fname_analysis_options_struct)
```

(continues on next page)
wt_opt, modeling_options, analysis_options = run_wisdem(fname_wt_input, fname_modeling_options, fname_analysis_options_aerostruct)

if MPI:
    rank = MPI.COMM_WORLD.Get_rank()
else:
    rank = 0

if rank == 0:
    print("RUN COMPLETED. RESULTS ARE AVAILABLE HERE: "+os.path.join(mydir, analysis_options["general"]["folder_output"]))

run([wt_opt], ["optimized"], modeling_options, analysis_options)

The CPU run time is approximately 5 minutes. As the script runs, you will see some output to your terminal, such as performance metrics and some analysis warnings. The optimizer might report that it has failed, but we have artificially limited the number of steps it can take during optimization, so that is expected. Once the optimization terminates, type in the terminal:

$ compare_designs outputs/BAR0.yaml outputs_aero/blade_out.yaml

This script compares the initial and optimized designs. Some screen output is generated, as well as plots (contained in the outputs folder), such as in Fig. 5.1 and Fig. 5.2. The twist optimization had to cope with a wider margin to stall than the baseline was originally designed to. The results show higher twist angles towards the blade tip, but the AEP is only mildly reduced by 0.18%.

**Simple Structural Optimization**

Next, we shift from an aerodynamic optimization of the blade to a structural optimization. In this case, we make the following changes,

- The design variables as the thickness of the blade structural layers Spar_cap_ss and Spar_cap_ps
- The thickness is parameterized in 8 locations along span and can vary between 70 and 130% of the initial value (using the max_decrease and max_increase options)
- The merit figure is blade mass instead of AEP
- A max allowable strain of $3500\mu e$ and the blade tip deflection constrain the problem, but the latter ratio is relaxed from a safety factor of 1.4175 to 1.134

To run this optimization problem, we can use the same geometry and modeling input files, and the optimization problem is captured in analysis_options_struct.yaml. The design variables are,

with the objective function set to:

```yaml
merit_figure: blade_mass
```

and the constraints are,

To run the optimization, just be sure to pass in this new analysis options,

```bash
$ wisdem BAR0.yaml modeling_options.yaml analysis_options_struct.yaml
```
5.7. Examples

Fig. 5.1: Baseline versus optimized induction profiles

Fig. 5.2: Baseline versus optimized twist profiles
or, to use the Python driver, be sure to comment out lines 15, 16, and 18 and only leave this uncommented:

```python
wt_opt, modeling_options, analysis_options = run_wisdem(fname_wt_input, fname_modeling_options, fname_analysis_options_struct)
```

and then do,

```bash
$ python blade_driver.py
```

(parallel calculation is also available if desired).

Once the optimization terminates, the same `compare_designs` script can be used again to plot the differences:

```bash
$ compare_designs outputs/BAR0.yaml outputs_aero/blade_out.yaml outputs_struct/blade_out.yaml
```

The relaxed tip deflection constraint compared to when the baseline was created allows the spar cap thickness to come down and the overall blade mass drops from 60.3 metric tons to 54.5 metric tons. This is shown in Fig. 5.3 and Fig. 5.4.

![Fig. 5.3: Baseline versus optimized spar cap thickness profiles](image)

Fig. 5.3: Baseline versus optimized spar cap thickness profiles
Finally, we will combine the previous two scenarios and use the levelized cost of energy, LCOE, as a way to balance the power production and minimum mass/cost objectives. This problem formulation is represented in the file, analysis_options_aerostruct.yaml. The design variables are,

```
aero_shape:
  twist:
    flag: True  # Flag to optimize the twist
    inverse: False  # Flag to determine twist from the user-defined
  desired margin to stall (defined in constraints)
  n_opt: 4  # Number of control points along blade span
  max_decrease: 0.08722222222222221  # Maximum decrease for the twist in [rad] at the n_opt locations
  max_increase: 0.08722222222222221  # Maximum increase for the twist in [rad] at the n_opt locations
  index_start: 2  # Lock the first two DVs from blade root
  index_end: 4  # All DVs close to blade tip are active

chord:
  flag: True  # Flag to optimize the chord
  n_opt: 4  # Number of control points along blade span
  max_decrease: 0.3  # Minimum multiplicative gain on existing chord
  max_increase: 3.  # Maximum multiplicative gain on existing chord
  index_start: 2  # Lock the first two DVs from blade root
```

(continues on next page)
with blade chord also activated as a design variable. The objective function set to:

\[ \text{merit\_figure: LCOE} \]

and the constraints are,

One more change for this final example is tighter optimization convergence tolerance \((1e^{-5})\), because LCOE tends to move only a very small amount from one design to the next.

\[
\begin{align*}
\text{optimization:} \\
\text{flag: True} & \quad \text{Flag to enable optimization} \\
\text{tol: 1.e-5} & \quad \text{Optimality tolerance} \\
\text{# max\_major\_iter: 10} & \quad \text{Maximum number of major design iterations (SNOPT)} \\
\text{# max\_minor\_iter: 100} & \quad \text{Maximum number of minor design iterations (SNOPT)} \\
\text{max\_iter: 2} & \quad \text{Maximum number of iterations (SLSQP)} \\
\text{solver: SLSQP} & \quad \text{Optimization solver. Other options are 'SLSQP' - 'CONMIN'} \\
\text{step\_size: 1.e-3} & \quad \text{Step size for finite differencing}
\end{align*}
\]

To run the optimization, just be sure to pass in this new analysis options

\[$ \text{wisdem BAR0.yaml modeling\_options.yaml analysis\_options\_aerostruct.yaml} \$

or, to use the Python driver, be sure run line 18 as above to be

\[ \text{wt\_opt, modeling\_options, analysis\_options = run\_wisdem(fname\_wt\_input, fname\_modeling\_options, fname\_analysis\_options\_aerostruct)} \]

and then do,

\[$ \text{python blade\_driver.py} \$

We can then use the \text{compare\_designs} command in the same way as above to plot the optimization results, two of which are shown in, Fig. 5.5 and Fig. 5.6. With more moving parts, it can be harder to interpret the results. In the end, LCOE is reduced marginally compared to the structural optimization-only case.

### 5.7.4 4. OpenMDAO Examples

WISDEM can be run through the yaml-input files if the intention is to do a full turbine and LCOE roll-up. Sometimes though, a user might just want to evaluate or optimize a single component within WISDEM. This can also be done through the yaml-input files, and some of the examples for the tower, monopile, and drivetrain show how that is accomplished. However, for other modules it may be simpler to interface with WISDEM directly with a python script. Since WISDEM is built on top of the OpenMDAO library, this tutorial is a cursory introduction into OpenMDAO syntax and problem structure.

OpenMDAO serves to connect the various components of turbine models into a cohesive whole that can be optimized in systems engineering problems. WISDEM uses OpenMDAO to build up modular \text{components} and \text{groups} of components to represent a wind turbine. Fortunately, OpenMDAO already provides some excellent training examples on their \text{website}. 

124 Chapter 5. Documentation Outline
Fig. 5.5: Baseline versus optimized chord profiles

Fig. 5.6: Baseline versus optimized twist profiles
List of Examples

- 4. OpenMDAO Examples
  - Tutorial 1: Betz Limit
  - Tutorial 2. The Sellar Problem

Tutorial 1: Betz Limit

This tutorial is based on the OpenMDAO example, Optimizing an Actuator Disk Model to Find Betz Limit for Wind Turbines, which we have extracted and added some additional commentary. The aim of this tutorial is to summarize the key points you’ll use to create basic WISDEM models. For those interested in WISDEM development, getting comfortable with all of the core OpenMDAO training examples is strongly encouraged.

A classic problem of wind energy engineering is the Betz limit. This is the theoretical upper limit of power that can be extracted from wind by an idealized rotor. While a full explanation is beyond the scope of this tutorial, it is explained in many places online and in textbooks. One such explanation is on Wikipedia https://en.wikipedia.org/wiki/Betz%27s_law.

Problem formulation

According to the Betz limit, the maximum power a turbine can extract from wind is:

\[
C_p = \frac{16}{27} \approx 0.593
\]

Where \( C_p \) is the coefficient of power.

We will compute this limit using OpenMDAO by optimizing the power coefficient as a function of the induction factor (ratio of rotor plane velocity to freestream velocity) and a model of an idealized rotor using an actuator disk.

Here is our actuator disc:

Where \( V_u \) freestream air velocity, upstream of rotor, \( V_r \) is air velocity at rotor exit plane and \( V_d \) is slipstream air velocity downstream of rotor, all measured in \( \frac{m}{s} \).

There are few other variables we’ll have:

- \( \alpha \): Induced Velocity Factor
- \( \text{Area} \): Rotor disc area in \( m^2 \)
- \( \text{thrust} \): Thrust produced by the rotor in N
- \( C_t \): Thrust coefficient
- \( \text{power} \): Power produced by rotor in W
- \( \rho \): Air density in \( kg/m^3 \)

Before we start in on the source code, let’s look at a few key snippets of the code
Fig. 5.7: Actuator disc
OpenMDAO implementation

First we need to import OpenMDAO

```python
import openmdao.api as om
```

Now we can make an `ActuatorDisc` class that models the actuator disc theory for the optimization. This is derived from an OpenMDAO class

```python
class ActuatorDisc(om.ExplicitComponent):
    # Inputs and Outputs
def setup(self):
        # Inputs into the model
        self.add_input("a", 0.5, desc="Induced velocity factor")
        self.add_input("Area", 10.0, units="m**2", desc="Rotor disc area")
        self.add_input("rho", 1.225, units="kg/m**3", desc="Air density")
        self.add_input("Vu", 10.0, units="m/s", desc="Freestream air velocity, upstream of rotor")

        # Outputs
        self.add_output("Vr", 0.0, units="m/s", desc="Air velocity at rotor exit plane")
        self.add_output("Vd", 0.0, units="m/s", desc="Slipstream air velocity, downstream of rotor")
        self.add_output("Ct", 0.0, desc="Thrust coefficient")
        self.add_output("Cp", 0.0, desc="Power coefficient")
        self.add_output("power", 0.0, units="W", desc="Power produced by the rotor")
        self.add_output("thrust", 0.0, units="m/s")

        # Declare which outputs are dependent on which inputs
        self.declare_partials("Vr", ["a", "Vu"])  
        self.declare_partials("Vd", ["a"])  
        self.declare_partials("Ct", ["a"])  
        self.declare_partials("thrust", ["a", "Area", "rho", "Vu"])  
        self.declare_partials("Cp", ["a"])  
        self.declare_partials("power", ["a", "Area", "rho", "Vu"])  
    # --------

    # Core theory
    def compute(self, inputs, outputs):
        a = inputs["a"]
        Vu = inputs["Vu"]
        rho = inputs["rho"]
        Area = inputs["Area"]
        qA = 0.5 * rho * Area * Vu ** 2
        outputs["Vr"] = Vr = Vu * (1 - 2 * a)
        outputs["Vd"] = Vd = Vu * (1 - 2 * a)
        outputs["Ct"] = Ct = 4 * a * (1 - a)
        outputs["Cp"] = Cp = Ct * qA
        outputs["power"] = power = Cp * qA * Vu
    # --------
```

(continues on next page)
def compute_partials(self, inputs, J):
    a = inputs["a"]
    Vu = inputs["Vu"]
    Area = inputs["Area"]
    rho = inputs["rho"]

    a_times_area = a * Area
    one_minus_a = 1.0 - a
    a_area_rho_vu = a_times_area * rho * Vu

    J["Vr", "a"] = -Vu
    J["Vr", "Vu"] = one_minus_a
    J["Vd", "a"] = -2.0 * Vu
    J["Ct", "a"] = 4.0 - 8.0 * a
    J["thrust", "a"] = 0.5 * rho * Vu ** 2 * Area * J["Ct", "a"]
    J["thrust", "Area"] = 2.0 * Vu ** 2 * a * rho * one_minus_a
    J["thrust", "Vu"] = 4.0 * a_area_rho_vu * one_minus_a
    J["Ct", "Area"] = 4.0 * a * (2.0 * a - 2.0) + 4.0 * one_minus_a ** 2
    J["Ct", "rho"] = 4.0 * a_area_rho_vu * one_minus_a ** 2
    J["Ct", "Vu"] = 2.0 * Area * Vu ** 3 * a * rho * (2.0 * a - 2.0) + 2.0 * Area * Vu ** 3 * a * rho * one_minus_a ** 2
    J["thrust", "rho"] = 2.0 * a_times_area * Vu ** 3 * (one_minus_a) ** 2
    J["thrust", "Vu"] = 6.0 * Area * Vu ** 2 * a * rho * one_minus_a ** 2

The class declaration, class ActuatorDisc(om.ExplicitComponent): shows that our class, ActuatorDisc inherits from the ExplicitComponent class in OpenMDAO. In WISDEM, 99% of all coded components are of the ExplicitComponent class, so this is the most fundamental building block to get accustomed to. Other types of components are described in the OpenMDAO docs here.

The ExplicitComponent class provides a template for the user to: - Declare their input and output variables in the setup method - Calculate the outputs from the inputs in the compute method. In an optimization loop, this is called at every iteration. - Calculate analytical gradients of outputs with respect to inputs in the compute_partials method.

The variable declarations take the form of self.add_input or self.add_output where a variable name and default/initial value is assigned. The value declaration also tells the OpenMDAO internals about the size and shape for any vector or multi-dimensional variables. Other optional keywords that can help with code documentation and model consistency are units= and desc=.

Working with analytical derivatives

We need to tell OpenMDAO which derivatives will need to be computed. That happens in the following lines:

```python
self.declare_partials("Vr", ["a", "Vu"])
sel.declare_partials("Vd", "a")
sel.declare_partials("Ct", "a")
sel.declare_partials("thrust", ["a", "Area", "rho", "Vu"])
sel.declare_partials("Cp", "a")
sel.declare_partials("power", ["a", "Area", "rho", "Vu"])
```
Note that lines like `self.declare_partials('Vr', ['a', 'Vu'])` references both the derivatives $\frac{\partial}{\partial a}$ and $\frac{\partial}{\partial Vu}$.

The Jacobian in which we provide solutions to the derivatives is

```python
def compute_partials(self, inputs, J):
    a = inputs['a']
    Vu = inputs['Vu']
    Area = inputs['Area']
    rho = inputs['rho']

    a_times_area = a * Area
    one_minus_a = 1.0 - a
    a_area_rho_vu = a_times_area * rho * Vu

    J['Vr', 'a'] = -Vu
    J['Vr', 'Vu'] = one_minus_a
    J['Vd', 'a'] = -2.0 * Vu
    J['Ct', 'a'] = 4.0 - 8.0 * a
    J['thrust', 'a'] = -2.0 * Area * rho * (2.0 * a - 2.0) + 4.0 * one_minus_a ** 2
    J['power', 'a'] = (2.0 * Area * Vu ** 3 * a * rho * one_minus_a ** 2
                      + 2.0 * Area * Vu ** 3 * rho * one_minus_a ** 2
                      + 6.0 * Area * Vu ** 3 * (one_minus_a) ** 2
                      + 6.0 * Area * Vu ** 3 * rho * (one_minus_a) ** 2
                      + 6.0 * Area * Vu ** 3 * (one_minus_a) ** 2
                      + 6.0 * Area * Vu ** 3 * rho * (one_minus_a) ** 2)
    J['power', 'Area'] = 2.0 * Area * Vu ** 3 * rho * one_minus_a ** 2
    J['power', 'rho'] = 2.0 * a * rho * Area * Vu ** 3 * (one_minus_a) ** 2
    J['power', 'Vu'] = 6.0 * Area * Vu ** 3 * a * rho * (one_minus_a) ** 2
```

In OpenMDAO, multiple components can be connected together inside of a Group. There will be some other new elements to review, so let's take a look:

**Betz Group:**

```python
class Betz(om.Group):
    ""
    Group containing the actuator disc equations for deriving the Betz limit.
    ""

def setup(self):
    indeps = self.add_subsystem("indeps", om.IndepVarComp(), promotes=['*'])
    indeps.add_output("a", 0.5)
    indeps.add_output("Area", 10.0, units="m**2")
    indeps.add_output("rho", 1.225, units="kg/m**3")
    indeps.add_output("Vu", 10.0, units="m/s")

    self.add_subsystem("a_disk", ActuatorDisc(), promotes=['a', 'Area', 'rho', 'Vu'])
```

The Betz class derives from the OpenMDAO Group class, which is typically the top-level class that is used in an analysis. The OpenMDAO Group class allows you to cluster models in hierarchies. We can put multiple components in groups. We can also put other groups in groups.
Components are added to groups with the `self.add_subsystem` command, which has two primary arguments. The first is the string name to call the subsystem that is added and the second is the component or sub-group class instance. A common optional argument is `promotes=`, which elevates the input/output variable string names to the top-level namespace. The Betz group shows examples where the `promotes=` can be passed a list of variable string names or the `'*'` wildcard to mean all input/output variables.

The first subsystem that is added is an IndepVarComp, which are the independent variables of the problem. Subsystem inputs that are not tied to other subsystem outputs should be connected to an independent variables. For optimization problems, design variables must be part of an IndepVarComp. In the Betz problem, we have a, Area, rho, and Vu. Note that they are promoted to the top level namespace, otherwise we would have to access them by 'indeps.x' and 'indeps.z'.

The next subsystem that is added is an instance of the component we created above:

```python
self.add_subsystem('a_disk', ActuatorDisc(), promotes=['a', 'Area', 'rho', 'Vu'])
```

The `promotes=` can also serve to connect variables. In OpenMDAO, two variables with the same string name in the same namespace are automatically connected. By promoting the same variable string names as in the IndepVarComp, they are automatically connected. For variables that are not connected in this way, explicit connect statements are required, which is demonstrated in the next tutorial.

Let’s optimize our system!

Even though we have all the pieces in a Group, we still need to put them into a Problem to be executed. The Problem instance is where we can assign design variables, objective functions, and constraints. It is also how the user interacts with the Group to set initial conditions and interrogate output values.

First, we instantiate the Problem and assign an instance of Betz to be the root model:

```python
prob = om.Problem() prob.model = Betz()
```

Next we assign an optimization driver to the problem instance. If we only wanted to evaluate the model once and not optimize, then a driver is not needed:

```python
prob.driver = om.ScipyOptimizeDriver() prob.driver.options['optimizer'] = 'SLSQP'
```

With the optimization driver in place, we can assign design variables, objective(s), and constraints. Any IndepVarComp can be a design variable and any model output can be an objective or constraint.

We want to maximize the objective, but OpenMDAO will want to minimize it as it is consistent with the standard optimization problem statement. So we minimize the negative to find the maximum. Note that Cp is not promoted from `a_disk`. Therefore we must reference it with `a_disk.Cp`

```python
prob.model.add_design_var("a", lower=0.0, upper=1.0) prob.model.add_design_var("Area", lower=0.0, upper=1.0) prob.model.add_objective("a_disk.Cp", scaler=-1.0)
```

Now we can run the optimization:

```python
prob.setup() prob.run_driver()
```

Optimization terminated successfully. (Exit mode 0) Current function value: -0.5925925906659251 Iterations: 5 Function evaluations: 6

(continues on next page)
Finally, the result:

Above, we see a summary of the steps in our optimization. Next, we print out the values we care about and list all of the inputs and outputs that are problem used.

```python
print("Coefficient of power Cp = ", prob["a_disk.Cp"])  
print("Induction factor a =", prob["a"])  
print("Rotor disc Area =", prob["Area"], "m^2")  
prob.model.list_inputs(units=True)  
prob.model.list_outputs(units=True)
```

Coefficient of power Cp = [0.59259259]
Induction factor a = [0.33335528]
Rotor disc Area = [1.] m^2
4 Input(s) in 'model'

<table>
<thead>
<tr>
<th>varname</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>top</td>
<td></td>
</tr>
<tr>
<td>a_disk</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>[0.33335528]</td>
</tr>
<tr>
<td>Area</td>
<td>[1.]</td>
</tr>
<tr>
<td>rho</td>
<td>[1.225]</td>
</tr>
<tr>
<td>Vu</td>
<td>[10.]</td>
</tr>
</tbody>
</table>

10 Explicit Output(s) in 'model'

<table>
<thead>
<tr>
<th>varname</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>top</td>
<td></td>
</tr>
<tr>
<td>indeps</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>[0.33335528]</td>
</tr>
<tr>
<td>Area</td>
<td>[1.]</td>
</tr>
<tr>
<td>rho</td>
<td>[1.225]</td>
</tr>
<tr>
<td>Vu</td>
<td>[10.]</td>
</tr>
<tr>
<td>a_disk</td>
<td></td>
</tr>
<tr>
<td>Vr</td>
<td>[6.6664472]</td>
</tr>
<tr>
<td>Vd</td>
<td>[3.33289439]</td>
</tr>
<tr>
<td>Ct</td>
<td>[0.88891815]</td>
</tr>
<tr>
<td>Cp</td>
<td>[0.59259259]</td>
</tr>
<tr>
<td>power</td>
<td>[362.96296178]</td>
</tr>
<tr>
<td>thrust</td>
<td>[54.44623668]</td>
</tr>
</tbody>
</table>
And there we have it. This matched the Betz limit of:

\[ C_p = \frac{16}{27} \approx 0.593 \]

**Tutorial 2. The Sellar Problem**

This tutorial is based on the OpenMDAO example, *Sellar - A Two-Discipline Problem with a Nonlinear Solver*, which we have extracted and added some additional commentary. The aim of this tutorial is to summarize the key points needed to create or understand basic WISDEM models. For those interested in WISDEM development, getting comfortable with all of the core OpenMDAO training examples is strongly encouraged.

**Problem formulation**

The Sellar problem are a couple of components (what Wikipedia calls models) that are simple equations. There is an objective to optimize and a couple of constraints to follow.

![Sellar XDSM Diagram](image)

**Fig. 5.8: Sellar XDSM**

This is an XDSM diagram that is used to describe the problem and optimization setups. For more reference on this notation and general reference for multidisciplinary design analysis and optimization (MDAO), see:

- **Problem formulation section of multidisciplinary design optimization on Wikipedia**: Read the definitions for *design variables, constraints, objectives and models*.
- **Lambe and Martins: Extensions to the Design Structure Matrix for the Description of Multidisciplinary Design, Analysis, and Optimization Processes**: Read section 2 “Terminology and Notation” for further explanation of *design variables, discipline analysis, response variables, target variables and coupling variables*. Read section 4 about XDSM diagrams that describe MDO processes.
OpenMDAO implementation

First we need to import OpenMDAO

```python
import openmdao.api as om
import numpy as np
```

Let's build Discipline 1 first. On the XDSM diagram, notice the parallelograms connected to Discipline 1 by thick grey lines. These are variables pertaining to the Discipline 1 component.

- **z**: An input. Since the components \(z_1, z_2\) can form a vector, we call the variable \(z\) in the code and initialize it to \((0, 0)\) with `np.zeros(2)`. Note that components of \(z\) are found in 3 of the white \(z\) parallelograms connected to multiple components and the objective, so this is a global design variable.

- **x**: An input. A local design variable for Discipline 1. Since it isn’t a vector, we just initialize it as a float.

- **y_2**: An input. This is a coupling variable coming from an output on Discipline 2

- **y_1**: An output. This is a coupling variable going to an input on Discipline 2

Let's take a look at the Discipline 1 component and break it down piece by piece. ### Discipline 1

```python
class SellarDis1(om.ExplicitComponent):
    """
    Component containing Discipline 1 -- no derivatives version.
    """

def setup(self):
    # Global Design Variable
    self.add_input("z", val=np.zeros(2))

    # Local Design Variable
    self.add_input("x", val=0.0)

    # Coupling parameter
    self.add_input("y2", val=1.0)

    # Coupling output
    self.add_output("y1", val=1.0)

    # Finite difference all partials.
    self.declare_partials("*", "*", method="fd")

def compute(self, inputs, outputs):
    """
    Evaluates the equation
    \[ y_1 = z_1^2 + z_2 + x_1 - 0.2 y_2 \]
    """
    z1 = inputs["z"][0]
    z2 = inputs["z"][1]
    x1 = inputs["x"]
    y2 = inputs["y2"]

    outputs["y1"] = z1 ** 2 + z2 + x1 - 0.2 * y2
```
The class declaration, class `SellarDis1(om.ExplicitComponent)`: shows that our class, `SellarDis1` inherits from the `ExplicitComponent` class in OpenMDAO. In WISDEM, 99% of all coded components are of the `ExplicitComponent` class, so this is the most fundamental building block to get accustomed to. Keen observers will notice that the `Sellar Problem` has implicitly defined variables that will need to be addressed, but that is addressed below. Other types of components are described in the OpenMDAO docs here.

The `ExplicitComponent` class provides a template for the user to: - Declare their input and output variables in the `setup` method - Calculate the outputs from the inputs in the `compute` method. In an optimization loop, this is called at every iteration. - Calculate analytical gradients of outputs with respect to inputs in the `compute_partials` method. This is absent from the `Sellar Problem`.

The variable declarations take the form of `self.add_input` or `self.add_output` where a variable name and default/initial value is assigned. The value declaration also tells the OpenMDAO internals about the size and shape for any vector or multi-dimensional variables. Other optional keywords that can help with code documentation and model consistency are `units=` and `desc=`.

Finally `self.declare_partials('*', '*', method='fd')` tell OpenMDAO to use finite difference to compute the partial derivative of the outputs with respect to the inputs. OpenMDAO provides many finite difference capabilities including: - Forward and backward differencing - Central differencing for second-order accurate derivatives - Differentiating in the complex domain which can offer improved accuracy for the models that support it.

Now lets take a look at `Discipline 2`.

- **z**: An input comprised of $z_1, z_2$.
- **$y_2$**: An output. This is a coupling variable going to an input on `Discipline 1`
- **$y_1$**: An input. This is a coupling variable coming from an output on `Discipline 1`

**Discipline 2**

class SellarDis2(om.ExplicitComponent):
    """
    Component containing Discipline 2 -- no derivatives version.
    """

def setup(self):
    # Global Design Variable
    self.add_input("z", val=np.zeros(2))

    # Coupling parameter
    self.add_input("y1", val=1.0)

    # Coupling output
    self.add_output("y2", val=1.0)

    # Finite difference all partials.
    self.declare_partials("*", ",*", method="fd")

def compute(self, inputs, outputs):
    """
    Evaluates the equation
    y2 = y1**(.5) + z1 + z2
    """
\[ z_1 = \text{inputs["z"]}[0] \]
\[ z_2 = \text{inputs["z"]}[1] \]
\[ y_1 = \text{inputs["y1"]} \]
\[ \text{outputs["y2"]} = y_1 ^ {0.5} + z_1 + z_2 \]

In OpenMDAO, multiple components can be connected together inside of a Group. There will be some other new elements to review, so let’s take a look:

**Sellar Group:**

```python
class SellarMDA(om.Group):
    
    Group containing the Sellar MDA.
    
    def setup(self):
        indeps = self.add_subsystem("indeps", om.IndepVarComp(), promotes=["*"])
        indeps.add_output("x", 1.0)
        indeps.add_output("z", np.array([5.0, 2.0]))

        self.add_subsystem("d1", SellarDis1(), promotes=["y1", "y2"],
                        connected="x: d1.x")
        self.add_subsystem("d2", SellarDis2(), promotes=["y1", "y2"],
                        connected="z: d1.z, d2.z")

        self.nonlinear_solver = om.NonlinearBlockGS()

        self.add_subsystem("obj_cmp", om.ExecComp("obj = x**2 + z[1] + y1 + \exp(-y2)", z=np.array([0.0, 0.0]), x=0.0),
                        connected="x, z, y1, y2, obj")

        self.add_subsystem("con_cmp1", om.ExecComp("con1 = 3.16 - y1"), promotes=["con1", "y1"],
                        connected="y1")
        self.add_subsystem("con_cmp2", om.ExecComp("con2 = y2 - 24.0"), promotes=["con2", "y2"],
                        connected="y2")
```

The SellarMDA class derives from the OpenMDAO Group class, which is typically the top-level class that is used in an analysis. The OpenMDAO Group class allows you to cluster models in hierarchies. We can put multiple components in groups. We can also put other groups in groups.

Components are added to groups with the `self.add_subsystem` command, which has two primary arguments. The first is the string name to call the subsystem that is added and the second is the component or sub-group class instance. A common optional argument is `promotes=[...`, which elevates the input/output variable string names to the top-level namespace. The SellarMDA group shows examples where the `promotes=[...` can be passed a list of variable string names or the '* wildcard to mean all input/output variables.

The first subsystem that is added is an IndepVarComp, which are the independent variables of the problem. Subsystem
inputs that are not tied to other subsystem outputs should be connected to an independent variables. For optimization problems, design variables must be part of an IndepVarComp. In the Sellar problem, we have \( x \) and \( z \). Note that they are promoted to the top level namespace, otherwise we would have to access them by ‘indeps.x’ and ‘indeps.z’.

The next subsystems that are added are instances of the components we created above:

```python
self.add_subsystem('d1', SellarDis1(), promotes=['y1', 'y2'])
self.add_subsystem('d2', SellarDis2(), promotes=['y1', 'y2'])
```

The `promotes=` can also serve to connect variables. In OpenMDAO, two variables with the same string name in the same namespace are automatically connected. By promoting \( y_1 \) and \( y_2 \) in both \( d_1 \) and \( d_2 \), they are automatically connected. For variables that are not connected in this way, explicit connect statements are required such as:

```python
self.connect('x', ['d1.x', 'd2.x'])
self.connect('z', ['d1.z', 'd2.z'])
```

These statements connect the IndepVarComp versions of \( x \) and \( z \) to the \( d_1 \) and \( d_2 \) versions. Note that if \( x \) and \( z \) could easily have been promoted in \( d_1 \) and \( d_2 \) too, which would have made these connect statements unnecessary, but including them is instructive.

The next statement, `self.nonlinear_solver = om.NonlinearBlockGS()`, handles the required internal iteration between \( y_1 \) and \( y_2 \) is our two components. OpenMDAO is able to identify a cycle between input/output variables and requires the user to specify a solver to handle the nested iteration loop. WISDEM does its best to avoid cycles.

Finally, we have a series of three subsystems that use instances of the OpenMDAO ExecComp component. This is a useful way to defining an ExplicitComponent inline, without having to create a whole new class. OpenMDAO is able to parse the string expression and populate the setup and compute methods automatically. This technique is used to create our objective function and two constraint functions directly:

```python
self.add_subsystem('obj_cmp', om.ExecComp('obj = x**2 + z[1] + y1 + exp(-y2)',
                                       z=np.array([0.0, 0.0]), x=0.0),
                        promotes=['x', 'z', 'y1', 'y2', 'obj'])
self.add_subsystem('con_cmp1', om.ExecComp('con1 = 3.16 - y1'),
                        promotes=['con1', 'y1'])
self.add_subsystem('con_cmp2', om.ExecComp('con2 = y2 - 24.0'),
                        promotes=['con2', 'y2'])
```

**Let’s optimize our system!**

Even though we have all the pieces in a Group, we still need to put them into a Problem to be executed. The Problem instance is where we can assign design variables, objective functions, and constraints. It is also how the user interacts with the Group to set initial conditions and interrogate output values.

First, we instantiate the Problem and assign an instance of SellarMDA to be the root model:

```python
prob = om Problem()
prob.model = SellarMDA()
```

Next we assign an optimization driver to the problem instance. If we only wanted to evaluate the model once and not optimize, then a driver is not needed:

```python
prob.driver = om.ScipyOptimizeDriver()
prob.driver.options["optimizer"] = "SLSQP"
prob.driver.options["tol"] = 1e-8
```

With the optimization driver in place, we can assign design variables, objective(s), and constraints. Any IndepVarComp can be a design variable and any model output can be an objective or constraint.
prob.model.add_design_var("x", lower=0, upper=10)
prob.model.add_design_var("z", lower=0, upper=10)
prob.model.add_objective("obj")
prob.model.add_constraint("con1", upper=0)
prob.model.add_constraint("con2", upper=0)

# Ask OpenMDAO to finite-difference across the whole model to compute the total→ gradients for the optimizer
# The other approach would be to finite-difference for the partials and build up the→ total derivative
prob.model.approx_totals()

Now we are ready for to ask OpenMDAO to setup the model, to use finite differences for gradient approximations, and to run the driver:

prob.setup()
prob.run_driver()

print("minimum found at")
print(float(prob["x"]))
print(prob["z"])
print("minimum objective")
print(float(prob["obj"]))

NL: NLBGS Converged in 7 iterations
NL: NLBGS Converged in 0 iterations
NL: NLBGS Converged in 3 iterations
NL: NLBGS Converged in 4 iterations
NL: NLBGS Converged in 8 iterations
NL: NLBGS Converged in 3 iterations
NL: NLBGS Converged in 4 iterations
NL: NLBGS Converged in 4 iterations
NL: NLBGS Converged in 9 iterations
NL: NLBGS Converged in 4 iterations
NL: NLBGS Converged in 5 iterations
NL: NLBGS Converged in 4 iterations
NL: NLBGS Converged in 9 iterations
NL: NLBGS Converged in 4 iterations
NL: NLBGS Converged in 4 iterations
NL: NLBGS Converged in 4 iterations
NL: NLBGS Converged in 8 iterations
NL: NLBGS Converged in 4 iterations
NL: NLBGS Converged in 5 iterations
NL: NLBGS Converged in 4 iterations
NL: NLBGS Converged in 4 iterations
NL: NLBGS Converged in 5 iterations
NL: NLBGS Converged in 4 iterations
NL: NLBGS Converged in 4 iterations
NL: NLBGS Converged in 5 iterations
NL: NLBGS Converged in 4 iterations

Optimization terminated successfully. (Exit mode 0)
Current function value: 3.183393951735934
Iterations: 6
Function evaluations: 6

(continues on next page)
Gradient evaluations: 6
Optimization Complete
-----------------------------------
minimum found at
0.0
[1.97763888e+00 2.83540724e-15]
minimum objective
3.183393951735934

### 5.7.5 5. Tower and Monopile Example

In this example we show how to perform simulation and optimization of a land-based tower and an offshore tower-monopile combination. Both examples can be executed either by limiting the input yaml to only the necessary components or by using a python script that calls WISDEM directly.

**Land-based Tower Design**

The following example, demonstrates how to set up and run analysis or optimization for a land-based tower. Some of the geometric parameters are seen in Figure 5.9. The tower is not restricted to 3 sections, any number of sections can be defined.

![Example of tower geometric parameterization.](image)

Fig. 5.9: Example of tower geometric parameterization.
Invoking with YAML files

To run just the tower analysis from the YAML input files, we just need to include the necessary elements. First dealing with the geometry_option.yaml file, this always includes the assembly section. Of the components, this means just the tower section. Also, the materials, environment, and costs section,

```yaml
name: 5MW Tower only

assembly:
  turbine_class: I
  turbulence_class: B
drivetrain: Geared
  rotor_orientation: Upwind
  number_of_blades: 3
  hub_height: 90.
  rotor_diameter: 126.
  rated_power: 5.e+6

components:
  tower:
    outer_shape_bem:
      reference_axis: &ref_axis_tower
      x:
        grid: [0.0, 1.0]
        values: [0.0, 0.0]
      y:
        grid: [0.0, 1.0]
        values: [0.0, 0.0]
      z:
        grid: &grid_tower [0., 0.5, 1.]
        values: [0., 43.8, 87.6]
    outer_diameter:
      grid: *grid_tower
      values: [6., 4.935, 3.87]
    drag_coefficient:
      grid: [0.0, 1.0]
      values: [1.0, 1.0]
    internal_structure_2d_fem:
      outfitting_factor: 1.07
      reference_axis: *ref_axis_tower
      layers:
        - name: tower_wall
          material: steel
          thickness:
            grid: *grid_tower
            values: [0.027, 0.0222, 0.019]

materials:
  - name: steel
    description: Steel of the tower and monopile, ASTM A572 Grade 50
    source: http://www.matweb.com/search/DataSheet.aspx?
      MatGUID=9ced5dc901c54bd1aef19403d0385d7f
    orth: 0
```

(continues on next page)
rho: 7800
alpha: 0.0
E: 200.e+009
nu: 0.265
G: 79.3e+009
GIc: 0  #Place holder, currently not used
GIIc: 0  #Place holder, currently not used
alp0: 0  #Place holder, currently not used
Xt: 1.12e+9
Xc: 2.16e+9
S: 0
Xy: 345.e+6
unit_cost: 0.7

environment:
  air_density: 1.225
  air_dyn_viscosity: 1.81e-5
  weib_shape_parameter: 2.
  air_speed_sound: 340.
  shear_exp: 0.2
  water_density: 1025.0
  water_dyn_viscosity: 1.3351e-3
  #water_depth: 0.0
  significant_wave_height: 0.0
  significant_wave_period: 0.0
  soil_shear_modulus: 140.e+6
  soil_poisson: 0.4

costs:
  labor_rate: 58.8
  painting_rate: 30.0

The modeling_options.yaml file is also limited to just the sections we need. Note that even though the monopile options are included here, since there was no specification of a monopile in the geometry inputs, this will be ignored. One new section is added here, a loading section that specifies the load scenarios that are applied to the tower. Since there is no rotor simulation to generate the loads, they must be specified by the user directly. Note that two load cases are specified. This creates a set of constraints for both of them.

# Generic modeling options file to run standard WISDEM case
General:
  verbosity: False  # When set to True, the code prints to screen many infos
WISDEM:
  RotorSE:
    flag: False
  DriveSE:
    flag: False
  TowerSE:
    flag: True  # Options of TowerSE module
    nLC: 2  # Number of design load cases
    wind: PowerWind  # Wind used
    gamma_f: 1.35  # Safety factor for fatigue loads
    gamma_m: 1.3  # Safety factor for material properties
    gamma_n: 1.0  # Safety factor for ...

(continues on next page)
gamma_b: 1.1 # Safety factor for ...
gamma_fatigue: 1.755 # Safety factor for fatigue loads
buckling_method: dnvgl # Buckling code type [eurocode or dnvgl]
buckling_length: 30 # Buckling parameter
soil_springs: True
gravity_foundation: False
frame3dd:
  shear: True
gem: True
tol: 1e-9
BOS:
  flag: False

Loading:
  mass: 285598.8
center_of_mass: [-1.13197635, 0.0, 0.50875268]
moment_of_inertia: [1.14930678e08, 2.20354030e07, 1.87597425e07, 0.0, 5.
  ...03710467e05, 0.0]
loads:
  - force: [1284744.19620519, 0.0, -2914124.84400512]
moment: [3963732.76208099, -2275104.79420872, -346781.68192839]
  velocity: 11.73732
  - force: [930198.60063279, 0.0, -2883106.12368949]
moment: [-1683669.22411597, -2522475.34625363, 147301.97023764]
  velocity: 70.0

The analysis_options.yaml poses the optimization problem,

```yaml
general:
  folder_output: outputs
  fname_output: tower_output
design_variables:

  tower:
    outer_diameter:
      flag: True
    lower_bound: 3.87
    upper_bound: 8.0
    layer_thickness:
      flag: True
    lower_bound: 4e-3
    upper_bound: 2e-1

merit_figure: tower_mass

constraints:
  tower:
    height_constraint:
      flag: False
    lower_bound: 1.0e-2
    upper_bound: 1.0e-2
```

(continues on next page)
flag: True

global_buckling:
flag: True

shell_buckling:
flag: True

d_to_t:
flag: True
lower_bound: 120.0
upper_bound: 500.0

taper:
flag: True
lower_bound: 0.2

slope:
flag: True

frequency_1:
flag: True
lower_bound: 0.13
upper_bound: 0.40

driver:

optimization:
flag: True  # Flag to enable optimization
tol: 1.e-6  # Optimality tolerance
max_major_iter: 10  # Maximum number of major design iterations (SNOPT)
max_minor_iter: 100  # Maximum number of minor design iterations (SNOPT)
max_iter: 100  # Maximum number of iterations (SLSQP)
solver: SLSQP  # Optimization solver. Other options are ‘SLSQP’ - ‘CONMIN’
step_size: 1.e-6  # Step size for finite differencing
form: forward  # Finite differencing mode, either forward or central

design_of_experiments:
flag: False  # Flag to enable design of experiments
run_parallel: True  # Flag to run using parallel processing
generator: Uniform  # Type of input generator. (Uniform)
um_samples: 5  # number of samples for (Uniform only)

recorder:
flag: False  # Flag to activate OpenMDAO recorder
file_name: log_opt.sql  # Name of OpenMDAO recorder

The yaml files can be called directly from the command line, or via a python script that passes them to the top-level WISDEM function,

```
$ wisdem nrel5mw_tower.yaml modeling_options.yaml analysis_options.yaml
```

or

```
$ python tower_driver.py
```

Where the contents of `tower_driver.py` are,

```
import os

from wisdem import run_wisdem
```

5.7. Examples
Calling Python Directly

The tower optimization can also be written using direct calls to WISDEM’s TowerSE module. First, we import the modules we want to use and setup the tower configuration. We set flags for if we want to perform analysis or optimization, as well as if we want plots to be shown at the end. Next, we set the tower height, diameter, and wall thickness.

```python
# Optimization by flag
# Two load cases
import numpy as np
import openmdao.api as om
from wisdem.towerse.tower import TowerSE
from wisdem.commonse.fileIO import save_data

# Set analysis and optimization options and define geometry
plot_flag = False
opt_flag = True

n_control_points = 3
n_materials = 1
n_load_cases = 2

h_param = np.diff(np.linspace(0.0, 87.6, n_control_points))
d_param = np.linspace(6.0, 3.87, n_control_points)
t_param = 1.3 * np.linspace(0.025, 0.021, n_control_points)
max_diam = 8.0
```

We then set many analysis options for the tower, including materials, safety factors, and FEM settings. The module uses the frame finite element code Frame3DD to perform the FEM analysis.

```python
modeling_options = {}
modeling_options["flags"] = {}
modeling_options["materials"] = {}
modeling_options["WISDEM"] = {}
modeling_options["WISDEM"]["TowerSE"] = {}
modeling_options["WISDEM"]["TowerSE"]["buckling_method"] = "eurocode"
modeling_options["WISDEM"]["TowerSE"]["buckling_length"] = 30.0
modeling_options["flags"]["monopile"] = False

# Monopile foundation only
modeling_options["WISDEM"]["TowerSE"]["soil_springs"] = False
modeling_options["WISDEM"]["TowerSE"]["gravity.Foundation"] = False
```

(continues on next page)
# safety factors

```python
modeling_options['WISDEM']['TowerSE']['gamma_f'] = 1.35
modeling_options['WISDEM']['TowerSE']['gamma_m'] = 1.3
modeling_options['WISDEM']['TowerSE']['gamma_n'] = 1.0
modeling_options['WISDEM']['TowerSE']['gamma_b'] = 1.1
modeling_options['WISDEM']['TowerSE']['gamma_fatigue'] = 1.35 * 1.3 * 1.0
```

# Frame3DD options

```python
modeling_options['WISDEM']['TowerSE']['frame3dd'] = {}
modeling_options['WISDEM']['TowerSE']['frame3dd']['shear'] = True
modeling_options['WISDEM']['TowerSE']['frame3dd']['geom'] = True
modeling_options['WISDEM']['TowerSE']['frame3dd']['tol'] = 1e-9
```

```python
modeling_options['WISDEM']['TowerSE']['n_height_tower'] = n_control_points
modeling_options['WISDEM']['TowerSE']['n_layers_tower'] = 1
modeling_options['WISDEM']['TowerSE']['n_height_monopile'] = 0
modeling_options['WISDEM']['TowerSE']['n_layers_monopile'] = 0
modeling_options['WISDEM']['TowerSE']['wind'] = "PowerWind"
modeling_options['WISDEM']['TowerSE']['nLC'] = n_load_cases
modeling_options['materials']['n_mat'] = n_materials
```

Next, we instantiate the OpenMDAO problem and add a tower model to this problem.

```python
prob = om.Problem()
prob.model = TowerSE(modeling_options=modeling_options)
```

Next, the script proceeds to set-up the design optimization problem if `opt_flag` is set to True. In this case, the optimization driver is first be selected and configured. We then set the objective, in this case tower mass, and scale it so it is of order 1 for better convergence behavior. The tower diameters and thicknesses are added as design variables. Finally, constraints are added. Some constraints are based on the tower geometry and others are based on the stress and buckling loads experienced in the loading cases.

```python
if opt_flag:
    # Choose the optimizer to use
    prob.driver = om.ScipyOptimizeDriver()
    prob.driver.options['optimizer'] = "SLSQP"

    # Add objective
    prob.model.add_objective("tower_mass", scaler=1e-6)

    # Add design variables, in this case the tower diameter and wall thicknesses
    prob.model.add_design_var("tower_outer_diameter_in", lower=3.87, upper=max_diam)
    prob.model.add_design_var("tower_layer_thickness", lower=4e-3, upper=2e-1)

    # Add constraints on the tower design
    prob.model.add_constraint("post1.constr_stress", upper=1.0)
    prob.model.add_constraint("post1.constr_global_buckling", upper=1.0)
    prob.model.add_constraint("post1.constr_shell_buckling", upper=1.0)
    prob.model.add_constraint("post2.constr_stress", upper=1.0)
    prob.model.add_constraint("post2.constr_global_buckling", upper=1.0)
    prob.model.add_constraint("post2.constr_shell_buckling", upper=1.0)
```
We then call setup() on the OpenMDAO problem, which finalizes the components and groups for the tower analysis or optimization. Once setup() has been called, we can access the problem values or modify them for a given analysis.

```python
prob.setup()
```

Now that we've set up the tower problem, we set values for tower, soil, and RNA assembly properties. For the soil, shear and modulus properties for the soil can be defined, but in this example we assume that all directions are rigid (3 translation and 3 rotation). The center of mass locations are defined relative to the tower top in the yaw-aligned coordinate system. Blade and hub moments of inertia should be defined about the hub center, nacelle moments of inertia are defined about the center of mass of the nacelle.

```python
prob["hub_height"] = prob["tower_height"] = h_param.sum()
prob["tower_foundation_height"] = 0.0
prob["tower_s"] = np.cumsum(np.r_[0.0, h_param]) / h_param.sum()
prob["tower_outer_diameter_in"] = d_param
prob["tower_layer_thickness"] = t_param.reshape((1, -1))
prob["tower_outfitting_factor"] = 1.07
prob["yaw"] = 0.0

# material properties
prob["E_mat"] = 210e9 * np.ones((n_materials, 3))
prob["G_mat"] = 80.8e9 * np.ones((n_materials, 3))
prob["rho_mat"] = [8500.0]
prob["sigma_y_mat"] = [450e6]

# extra mass from RNA
prob["rna_mass"] = np.array([285598.8])
mIxx = 1.14930678e08
mIyy = 2.20354030e07
mIzz = 1.87597425e07
mIxy = 0.0
mIxz = 5.03710467e05
mIyz = 0.0
prob["rna_I"] = np.array([mIxx, mIyy, mIzz, mIxy, mIxz, mIyz])
prob["rna_cg"] = np.array([-1.13197635, 0.0, 0.50875268])
```

For the power-law wind profile, the only parameter needed is the shear exponent. In addition, some geometric parameters for the wind profile's extend must be defined, the base (or no-slip location) at $z_0$, and the height at which a reference velocity will be defined.

```python
prob["unit_cost_mat"] = [2.0] # USD/kg
prob["labor_cost_rate"] = 100.0 / 60.0 # USD/min
prob["painting_cost_rate"] = 30.0 # USD/m^2

# wind & wave values
prob["wind_reference_height"] = 90.0
prob["z0"] = 0.0
```
prob["cd_usr"] = -1.0
prob["rho_air"] = 1.225
prob["mu_air"] = 1.7934e-5
prob["beta_wind"] = 0.0
if modeling_options["WISDEM"]["TowerSE"]["wind"] == "PowerWind":
    prob["shearExp"] = 0.2

As mentioned earlier, we are allowing for two separate loading cases. The wind speed, and rotor force/moments for those two cases are now defined. The wind speed location corresponds to the reference height defined previously as wind_zref. In this simple case, we include only thrust and torque, but in general all 3 components of force and moments can be defined in the hub-aligned coordinate system. The assembly automatically handles translating the forces and moments defined at the rotor to the tower top.

prob["wind1.Uref"] = 11.73732
Fx1 = 1284744.19620519
Fy1 = 0.0
Fz1 = -2914124.84400512
Mxx1 = 3963732.76208099
Myy1 = -2275104.79420872
Mzz1 = -346781.68192839
prob["pre1.rna_F"] = np.array([Fx1, Fy1, Fz1])
prob["pre1.rna_M"] = np.array([Mxx1, Myy1, Mzz1])

prob["wind2.Uref"] = 70.0
Fx2 = 930198.60063279
Fy2 = 0.0
Fz2 = -2883106.12368949
Mxx2 = -1683669.22411597
Myy2 = -2522475.34625363
Mzz2 = 147301.97023764
prob["pre2.rna_F"] = np.array([Fx2, Fy2, Fz2])
prob["pre2.rna_M"] = np.array([Mxx2, Myy2, Mzz2])

We can now run the model and display some of the outputs.

prob.model.approx_totals()
if opt_flag:
    prob.run_driver()
else:
    prob.run_model()
os.makedirs("outputs", exist_ok=True)
save_data(os.path.join("outputs", "tower_example"), prob)
Results

Whether invoking from the yaml files or running via python directly, the optimization result is the same. It should look something like this:

```markdown
Optimization terminated successfully  (Exit mode 0)
  Current function value: 0.24021234
  Iterations: 11
  Function evaluations: 12
  Gradient evaluations: 11
Optimization Complete
```

The python scripts, whether passing the yaml files or calling TowerSE directly, also print information to the screen and make a quick plot of the constraints along the tower,

```python
z = 0.5 * (prob["z_full"][:-1] + prob["z_full"][1:])
print("zs = ", prob["z_full"])  
print("ds = ", prob["d_full"])  
print("ts = ", prob["t_full"])  
print("mass (kg) = ", prob["tower_mass"])  
print("cg (m) = ", prob["tower_center_of_mass"]) 
print("d:t constraint = ", prob["constr_d_to_t"])  
print("taper ratio constraint = ", prob["constr_taper"])  
print("\nwind: ", prob["wind1.Uref"])  
print("freq (Hz) = ", prob["tower1.structural_frequencies"]) 
print("Fore-aft mode shapes = ", prob["tower1.fore_aft_modes"]) 
print("Side-side mode shapes = ", prob["tower1.side_side_modes"]) 
print("top_deflection1 (m) = ", prob["tower1.top_deflection"]) 
print("Tower base forces1 (N) = ", prob["tower1.base_F"]) 
print("Tower base moments1 (Nm) = ", prob["tower1.base_M"]) 
print("stress1 = ", prob["post1.constr_stress"])  
print("GL buckling = ", prob["post1.constr_global_buckling"])  
print("Shell buckling = ", prob["post1.constr_shell_buckling"]) 

if plot_flag: 
    import matplotlib.pyplot as plt

    # Old line plot 
    stress1 = np.copy(prob["post1.constr_stress"])  
    shellBuckle1 = np.copy(prob["post1.constr_shell_buckling"])  
    globalBuckle1 = np.copy(prob["post1.constr_global_buckling"])
```
stress2 = prob["post2.constr_stress"]
shellBuckle2 = prob["post2.constr_shell_buckling"]
globalBuckle2 = prob["post2.constr_global_buckling"]

plt.figure(figsize=(5.0, 3.5))
plt.subplot2grid((3, 3), (0, 0), colspan=2, rowspan=3)
plt.plot(stress1, z, label="stress 1")
plt.plot(stress2, z, label="stress 2")
plt.plot(shellBuckle1, z, label="shell buckling 1")
plt.plot(shellBuckle2, z, label="shell buckling 2")
plt.plot(globalBuckle1, z, label="global buckling 1")
plt.plot(globalBuckle2, z, label="global buckling 2")
plt.legend(bbox_to_anchor=(1.05, 1.0), loc=2)
plt.xlabel("utilization")
plt.ylabel("height along tower (m)")
plt.tight_layout()
plt.show()

This generates the terminal screen output of,

```
zs = [ 0. 15. 30. 45. 60. 75. 90. ]
ds = [7.96388868 6.87580097 5.78771327 4.69962557 4.42308371 4.14654186 3.87 ]
ts = [0.02013107 0.02013107 0.02013107 0.01793568 0.01793568 0.01793568]
mass (kg) = [240212.34456004]
cg (m) = [37.93106541]
d:t constraint = [314.5266543 238.89879914]
taper ratio constraint = [0.59011693 0.82346986]
wind: [11.73732]
```

```
freq (Hz) = [0.2853175 0.30501523 1.10109606 1.39479974 2.07484605 4.33899489]
```

```
Fore-aft mode shapes = [[ 1.1546986 -2.42796755 5.84302789 -4.96078724 1. -39102831]
 [ -22.1365112 -0.8234945 -26.50387043 97.18894013 -46.72506397]
```

```
Side-side mode shapes = [[ 1.11021561 -35.45954107 80.57590888 -55.60717732 9.78953989]
 [ 7.52307725 -8.97016199 27.3799066 -38.2942352 13.3620168 ]
 [ 1.70121561 -35.45954107 80.57590888 -55.60717732 9.78953989]]
```

```
top_deflection1 (m) = [0.97689316]
```
Side-side mode shapes = 

\[
\begin{bmatrix}
1.11003145 & -2.37051178 & 5.66866315 & -4.71969548 & 1.31151264 \\
7.5229338 & -8.96852486 & 27.37550871 & -38.28969538 & 13.35977773 \\
1.70100363 & -35.45896917 & 80.57556268 & -55.60741575 & 9.78981861
\end{bmatrix}
\]

top_deflection2 (m) = [0.88335072]

Tower base forces2 (N) = 

\[
\begin{bmatrix}
1.67397530e+06 & -4.65661287e-10 & -7.88527891e+06
\end{bmatrix}
\]

Tower base moments2 (Nm) = 

\[
\begin{bmatrix}
-1.87422266e+06 & 1.19912567e+08 & 1.47301970e+05
\end{bmatrix}
\]

stress2 = [0.64216922 0.66921096 0.69062355 0.64457187 0.35261396 0.27325483]

GL buckling = [0.6482481 0.68801235 0.74578503 0.77196602 0.55533402 0.52692342]

Shell buckling = [0.98272504 0.90444378 0.81799653 0.74124285 0.2767209 0.17886837]

The stress, buckling, and damage loads are shown in Figure 5.10. Each is a utilization and so should be <1 for a feasible result. Because the result shown here was for an optimization case, we see some of the utilization values are right at the 1.0 upper limit.

**Offshore Monopile Design**

Monopile design in WISDEM is modeled as an extension of the tower. In this example, the tower from above is applied offshore with the key differences being:

- Tower base is now at 10m above the water level at a transition piece coupling with the monopile
- Monopile extends from 10m above the water level through to the sea floor, at a water depth of 30m, with the pile extending an additional 25m into the surface.
- Monopile also has three sections, with one section for the submerged pile, the middle section for the water column, and the top section above the water up until the transition piece.
- Maximum allowable diameter for the monopile is 8m

**Invoking with YAML files**

To run just the monopile, there is an additional monopile section that must be added to the components section of the yaml,

```yaml
monopile:
  transition_piece_mass: 100e3
  transition_piece_cost: 100e3
  outer_shape_bem:
    reference_axis: &ref_axis_mono
    x:
      grid: [0.0, 1.0]
      values: [0.0, 0.0]
    y:
      grid: [0.0, 1.0]
      values: [0.0, 0.0]
    z:
      grid: &grid_mono [0., 0.3846, 0.8462, 1.0]
      values: [-55.0, -30.0, 0.0, 10.0]
  outer_diameter:
    grid: *grid_mono
    values: [8., 8.0, 8.0, 8.0]
```

(continues on next page)
Fig. 5.10: Utilization along tower for ultimate stress, shell buckling, global buckling, and fatigue damage.
The environment section must also be updated with the offshore properties,

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>shear_exp</td>
<td>0.1</td>
</tr>
<tr>
<td>water_density</td>
<td>1025.0</td>
</tr>
<tr>
<td>water_dyn_viscosity</td>
<td>1.3351e-3</td>
</tr>
<tr>
<td>water_depth</td>
<td>30.0</td>
</tr>
<tr>
<td>significant_wave_height</td>
<td>4.52</td>
</tr>
<tr>
<td>significant_wave_period</td>
<td>9.45</td>
</tr>
<tr>
<td>soil_shear_modulus</td>
<td>140.e+6</td>
</tr>
<tr>
<td>soil_poisson</td>
<td>0.4</td>
</tr>
</tbody>
</table>

The modeling_options.yaml file already contains a section for the monopile, with entries identical to the tower section. The input loading scenarios are also the same. The analysis_options.yaml file, however, is different and activates the design variables and constraints associated with the monopile. Note also that the objective function now says structural_mass, to capture the combined mass of both the tower and monopile.

```
drag_coefficient:
    grid: [0.0, 1.0]
    values: [1.0, 1.0]
internal_structure_2d_fem:
    outfitting_factor: 1.07
    reference_axis: *ref_axis_mono
layers:
    - name: tower_wall
      material: steel
      thickness:
        grid: *grid_mono
        values: [0.055, 0.055, 0.055, 0.055]

shear_exp: 0.1
water_density: 1025.0
water_dyn_viscosity: 1.3351e-3
water_depth: 30.0
significant_wave_height: 4.52
significant_wave_period: 9.45
soil_shear_modulus: 140.e+6
soil_poisson: 0.4
```

```
general:
    folder_output: outputs
    fname_output: refturb_output
design_variables:

tower:
    outer_diameter:
        flag: True
        lower_bound: 3.87
        upper_bound: 8.0
    layer_thickness:
        flag: True
        lower_bound: 4e-3
        upper_bound: 2e-1

monopile:
    outer_diameter:
        flag: True
        lower_bound: 3.87
        upper_bound: 8.0
    layer_thickness:
        flag: True
        lower_bound: 4e-3
        upper_bound: 2e-1
```

(continues on next page)
merit_figure: structural_mass

constraints:
  tower:
    height_constraint:
      flag: False
      lower_bound: 1.e-2
      upper_bound: 1.e-2
    stress:
      flag: True
    global_buckling:
      flag: True
    shell_buckling:
      flag: True
    d_to_t:
      flag: True
      lower_bound: 120.0
      upper_bound: 2000.0
  taper:
    flag: True
    lower_bound: 0.2
    slope:
    flag: True
    frequency_1:
      flag: True
      lower_bound: 0.13
      upper_bound: 0.40
  monopile:
    stress:
      flag: True
    global_buckling:
      flag: True
    shell_buckling:
      flag: True
    d_to_t:
      flag: True
      lower_bound: 120.0
      upper_bound: 2000.0
    taper:
      flag: True
      lower_bound: 0.2
    slope:
      flag: True
    frequency_1:
      flag: True
      lower_bound: 0.13
      upper_bound: 0.40
    pile_depth:
      flag: True
      lower_bound: 0.0
driver:
  optimization:
    flag: True  # Flag to enable optimization
    tol: 1.e-3  # Optimality tolerance
    max_major_iter: 10  # Maximum number of major design iterations (SNOPT)
    max_minor_iter: 100  # Maximum number of minor design iterations (SNOPT)
    max_iter: 100  # Maximum number of iterations (SLSQP)
    solver: SLSQP  # Optimization solver. Other options are 'SLSQP' - 'CONMIN'
    step_size: 1.e-3  # Step size for finite differencing
    form: forward  # Finite differencing mode, either forward or central
  design_of_experiments:
    flag: False  # Flag to enable design of experiments
    run_parallel: True  # Flag to run using parallel processing
    generator: Uniform  # Type of input generator. (Uniform)
    num_samples: 5  # number of samples for (Uniform only)

recorder:
  flag: False  # Flag to activate OpenMDAO recorder
  file_name: log_opt.sql  # Name of OpenMDAO recorder

The yaml files can be called directly from the command line, or via a python script that passes them to the top-level WISDEM function,

```
$ wisdem nrel5mw_monopile.yaml modeling_options.yaml analysis_options_monopile.yaml
```

or

```
$ python monopile_driver.py
```

Where the contents of monopile_driver.py are,

```python
import os

from wisdem import run_wisdem

## File management
mydir = os.path.dirname(os.path.realpath(__file__))  # get path to this file
fname_wt_input = mydir + os.sep + "nrel5mw_monopile.yaml"
fname_modeling_options = mydir + os.sep + "modeling_options.yaml"
fname_analysis_options = mydir + os.sep + "analysis_options_monopile.yaml"

wt_opt, analysis_options, opt_options = run_wisdem(fname_wt_input, fname_modeling_options, fname_analysis_options)
```
Calling Python Directly

The monopile optimization script using direct calls to WISDEM’s TowerSE module also resembles the tower code, with some key additions to expand the design accordingly. First, the script setup now includes the monopile initial condition and the transition piece location between the monopile and tower. In this script, \( z = 0 \) corresponds to the mean sea level (MSL).

```python
# Optimization by flag
# Two load cases
import os

import numpy as np
import openmdao.api as om
from wisdem.towerse.tower import TowerSE
from wisdem.commonse.fileIO import save_data

# Set analysis and optimization options and define geometry
plot_flag = False
opt_flag = True

n_control_points = 4
n_materials = 1
n_load_cases = 2

# Tower initial condition
hubH = 87.6
htrans = 10.0
h_paramT = np.diff(np.linspace(htrans, hubH, n_control_points))
d_paramT = np.linspace(7.0, 3.87, n_control_points)
t_paramT = 1.3 * np.linspace(0.025, 0.021, n_control_points)

# Monopile initial condition
pile_depth = 25.0
water_depth = 30.0
h_paramM = np.r_[pile_depth, water_depth, htrans]
d_paramM = np.linspace(8.0, 7.0, n_control_points)
t_paramM = 0.025 * np.ones(n_control_points)
max_diam = 8.0

The modeling options only needs to add in the number of sections and control points for the monopile,

```python
modeling_options = {}
modeling_options["flags"] = {}
modeling_options["materials"] = {}
modeling_options["WISDEM"] = {}
modeling_options["WISDEM"]["TowerSE"] = {}
modeling_options["WISDEM"]["TowerSE"]["buckling_length"] = 30.0
modeling_options["WISDEM"]["TowerSE"]["buckling_method"] = "dnvgl"
modeling_options["flags"]["monopile"] = True

# Monopile foundation
modeling_options["WISDEM"]["TowerSE"]["soil_springs"] = True
```

(continues on next page)
modeling_options["WISDEM"]['TowerSE'] ['gravity_foundation'] = False

# safety factors
modeling_options["WISDEM"]['TowerSE'] ['gamma_f'] = 1.35
modeling_options["WISDEM"]['TowerSE'] ['gamma_m'] = 1.3
modeling_options["WISDEM"]['TowerSE'] ['gamma_n'] = 1.0
modeling_options["WISDEM"]['TowerSE'] ['gamma_b'] = 1.1
modeling_options["WISDEM"]['TowerSE'] ['gamma_fatigue'] = 1.35 * 1.3 * 1.0

# Frame3DD options
modeling_options["WISDEM"]['TowerSE'] ['frame3dd'] = {}
modeling_options["WISDEM"]['TowerSE'] ['frame3dd'] ['shear'] = True
modeling_options["WISDEM"]['TowerSE'] ['frame3dd'] ['geom'] = True
modeling_options["WISDEM"]['TowerSE'] ['frame3dd'] ['tol'] = 1e-9

modeling_options["WISDEM"]['TowerSE'] ['n_height_tower'] = n_control_points
modeling_options["WISDEM"]['TowerSE'] ['n_layers_tower'] = 1
modeling_options["WISDEM"]['TowerSE'] ['n_height_monopile'] = n_control_points
modeling_options["WISDEM"]['TowerSE'] ['n_layers_monopile'] = 1
modeling_options["WISDEM"]['TowerSE'] ['wind'] = "PowerWind"
modeling_options["WISDEM"]['TowerSE'] ['nLC'] = n_load_cases
modeling_options["materials"]['n_mat'] = n_materials

Next, we instantiate the OpenMDAO problem as before,

prob = om.Problem()
prob.model = TowerSE(modeling_options=modeling_options)

The optimization problem switches the objective function to the total structural mass of the tower plus the monopile, and the monopile diameter and thickness schedule are appended to the list of design variables.

if opt_flag:
    # Choose the optimizer to use
    prob.driver = om.ScipyOptimizeDriver()
    prob.driver.options["optimizer"] = "SLSQP"

    # Add objective
    # prob.model.add_objective('tower_mass', scaler=1e-6) # Only tower
    # prob.model.add_objective('monopile_mass', scaler=1e-6) # Only monopile
    prob.model.add_objective("structural_mass", scaler=1e-6) # Both

    # Add design variables, in this case the tower diameter and wall thicknesses
    prob.model.add_design_var("monopile_outer_diameter_in", lower=3.87, upper=max_diam)
    prob.model.add_design_var("monopile_layer_thickness", lower=4e-3, upper=2e-1)

    prob.model.add_design_var("tower_outer_diameter_in", lower=3.87, upper=max_diam)
    prob.model.add_design_var("tower_layer_thickness", lower=4e-3, upper=2e-1)

    # Add constraints on the tower design
    prob.model.add_constraint("post1.constr_stress", upper=1.0)
    prob.model.add_constraint("post1.constr_global_buckling", upper=1.0)
    prob.model.add_constraint("post1.constr_shell_buckling", upper=1.0)
prob.model.add_constraint("post2.constr_stress", upper=1.0)
prob.model.add_constraint("post2.constr_global_buckling", upper=1.0)
prob.model.add_constraint("post2.constr_shell_buckling", upper=1.0)
prob.model.add_constraint("constr_d_to_t", lower=120.0, upper=500.0)
prob.model.add_constraint("constr_taper", lower=0.2)
prob.model.add_constraint("slope", upper=1.0)
prob.model.add_constraint("suctionpile_depth", lower=0.0)
prob.model.add_constraint("tower1.f1", lower=0.13, upper=0.40)
prob.model.add_constraint("tower2.f1", lower=0.13, upper=0.40)

We then call setup() as before,

prob.setup()

Next, the additional inputs for the monopile include its discretization of the monopile and starting depth.

prob["hub_height"] = hubH
prob["water_depth"] = water_depth

prob["tower.Foundation_height"] = htrans
prob["tower_height"] = h_paramT.sum()
prob["tower.s"] = np.cumsum(np.r_[0.0, h_paramT]) / h_paramT.sum()
prob["tower.outer_diameter_in"] = d_paramT
prob["tower.layer_thickness"] = t_paramT.reshape((1, -1))
prob["tower.outfitting_factor"] = 1.07

prob["transition.piece_mass"] = 100e3
prob["monopile.foundation_height"] = -55.0
prob["monopile.height"] = h_paramM.sum()
prob["monopile.s"] = np.cumsum(np.r_[0.0, h_paramM]) / h_paramM.sum()
prob["monopile.outer_diameter_in"] = d_paramM
prob["monopile.layer_thickness"] = t_paramM.reshape((1, -1))
prob["monopile.outfitting_factor"] = 1.07

prob["yaw"] = 0.0

# offshore specific
prob["G.soil"] = 140e6
prob["nu.soil"] = 0.4

# material properties
prob["E.mat"] = 210e9 * np.ones((n_materials, 3))
prob["G.mat"] = 80.8e9 * np.ones((n_materials, 3))
prob["rho.mat"] = [8500.0]
prob["sigma_y.mat"] = [450e6]

# extra mass from RNA
prob["rna.mass"] = np.array([285598.8])
mIxx = 1.14930678e08
mIyy = 2.20354030e07
mIzz = 1.87597425e07

(continues on next page)
The offshore environment parameters, such as significant wave heights and water density, must also be set.

```python
prob["unit_cost_mat"] = [2.0]  # USD/kg
prob["labor_cost_rate"] = 100.0 / 60.0  # USD/min
prob["painting_cost_rate"] = 30.0  # USD/m^2

# wind & wave values
prob["wind_reference_height"] = 90.0
prob["z0"] = 0.0
prob["cd_usr"] = -1.0
prob["rho_air"] = 1.225
prob["rho_water"] = 1025.0
prob["mu_air"] = 1.7934e-5
prob["mu_water"] = 1.3351e-3
prob["beta_wind"] = 0.0
prob["Hsig_wave"] = 4.52
prob["Tsig_wave"] = 9.52
if modeling_options["WISDEM"]["TowerSE"]["wind"] == "PowerWind":
    prob["shearExp"] = 0.1
```

The load cases are exactly the same as in the tower-only design case,

```python
prob["wind1.Uref"] = 11.73732
Fx1 = 1284744.19620519
Fy1 = 0.0
Fz1 = -2914124.84400512
Mxx1 = 3963732.76208099
Myy1 = -2275104.79420872
Mzz1 = -346781.68192839
prob["pre1.rna_F"] = np.array([Fx1, Fy1, Fz1])
prob["pre1.rna_M"] = np.array([Mxx1, Myy1, Mzz1])

prob["wind2.Uref"] = 70.0
Fx2 = 930198.60063279
Fy2 = 0.0
Fz2 = -2883106.12368949
Mxx2 = -1683669.22411597
Myy2 = -2522475.34625363
Mzz2 = 147301.97023764
prob["pre2.rna_F"] = np.array([Fx2, Fy2, Fz2])
prob["pre2.rna_M"] = np.array([Mxx2, Myy2, Mzz2])
```

We can now run the model and display some of the outputs, such as in Figure 5.11.

```python
prob.model.approx_totals()
if opt_flag:
```

(continues on next page)
5.7.6 6. Drivetrain Model Example

This example optimizes the sizing of key drivetrain components, assuming their overall length and height have already been determined by a target blade tip deflection and hub height, respectively. Only Python-scripts are supported for now, as a yaml-based input using only hub and nacelle parameters is not yet available.

Direct-Drive Design

This example is for design sizing (diameter and thickness) of the shaft, nose/turret, and bedplate wall thickness of the direct-drive layout shown in Fig. 5.12 and in Fig. 5.13.

Specifically, the design variables are,

- $L_{h1}$
- $L_{12}$
- $D_{hub}$
- $D_{i_{ss}}$
- $D_{nose}$
- $l_{i_{ss}}$
- $t_{nose}$
- $t_{bed}$

The design script starts with importing the required libraries,

```python
import numpy as np
import openmdao.api as om
from wisdem.commonse.fileIO import save_data
from wisdem.drivetrainse.drivetrain import DrivetrainSE

opt_flag = True
```

The modeling options dictionary sets the partial safety factors, the vector sizes, and specifies that a detailed generator design will not be run.

```python
opt = {}
opt["WISDEM"] = {}
opt["WISDEM"]["DriveSE"] = {}
opt["WISDEM"]["DriveSE"]["direct"] = True
opt["WISDEM"]["DriveSE"]["hub"] = {}
opt["WISDEM"]["DriveSE"]["hub"]["hub_gamma"] = 2.0
opt["WISDEM"]["DriveSE"]["hub"]["spinner_gamma"] = 1.5
opt["WISDEM"]["DriveSE"]["gamma_f"] = 1.35
```
Fig. 5.11: Utilization along tower and monopile for ultimate stress, shell buckling, global buckling, and fatigue damage.
Next, we instantiate the OpenMDAO problem and assign the drivetrain as the model of the problem,

```python
prob = om.Problem()
prob.model = DrivetrainSE(modeling_options=opt, n_dlcs=1)
```

Next, the script proceeds to set-up the design optimization problem if opt_flag is set to True. In this case, the optimization driver is first be selected and configured. We then set the objective, in this case nacelle mass, and scale it so it is of order 1 for better convergence behavior. The drivetrain diameters and thicknesses are added as design variables, as well as the lengths that determine the shaft, nose, and bedplate lengths to reach the intended overhang distance. Finally, a number of constraints are added, which fall into the categories of,

- **von Mises stress utilizations**: These constraints must be less than 1 and capture that the von Mises stress in the load-bearing components, multiplied by a safety factor must be less than the shear stress of the material.

- **Main bearing deflection utilizations**: Each bearing type has an associated maximum deflection, measured as an angle. These constraints ensure that the shaft and nose/turret deflections at the bearing attachment points do not exceed those limits.

---

5.7. Examples
Fig. 5.13: Detailed direct-drive configuration with key user inputs and derived values.
• **Minimum allowable hub diameter**: For a given blade root radius and number of blades, this is the minimum hub radius that can safely accommodate those dimensions.

• **Satisfy target overhang and hub height**: Ensure that the shaft, turret, and bedplate lengths are sufficient to meet the desired overhang and hub height

• **Allow maintenance access**: Specify the minimum height required to allow human access into the nose/turret for maintenance activities.

```python
if opt_flag:
    # Choose the optimizer to use
    prob.driver = om.ScipyOptimizeDriver()
    prob.driver.options['optimizer'] = "SLSQP"
    prob.driver.options['tol'] = 1e-5

    # Add objective
    prob.model.add_objective("nacelle_mass", scaler=1e-6)

    # Add design variables, in this case the tower diameter and wall thicknesses
    prob.model.add_design_var("L_12", lower=0.1, upper=5.0)
    prob.model.add_design_var("L_h1", lower=0.1, upper=5.0)
    prob.model.add_design_var("hub_diameter", lower=3.0, upper=15.0)
    prob.model.add_design_var("lss_diameter", lower=0.5, upper=6.0)
    prob.model.add_design_var("lss_wall_thickness", lower=4e-3, upper=5e-1, ref=1e-2)
    prob.model.add_design_var("nose_diameter", lower=0.5, upper=6.0)
    prob.model.add_design_var("nose_wall_thickness", lower=4e-3, upper=5e-1, ref=1e-2)
    prob.model.add_design_var("bedplate_wall_thickness", lower=4e-3, upper=5e-1, ref=1e-2)

    # Add constraints on the tower design
    prob.model.add_constraint("constr_lss_vonmises", upper=1.0)
    prob.model.add_constraint("constr_bedplate_vonmises", upper=1.0)
    prob.model.add_constraint("constr_mb1_defl", upper=1.0)
    prob.model.add_constraint("constr_mb2_defl", upper=1.0)
    prob.model.add_constraint("constr_hub_diameter", lower=0.0)
    prob.model.add_constraint("constr_length", lower=0.0)
    prob.model.add_constraint("constr_length", lower=0.0)
    prob.model.add_constraint("constr_access", lower=0.0)
    prob.model.add_constraint("constr_ecc", lower=0.0)
```

With the optimization problem defined, the OpenMDAO problem is activated

```python
prob.setup()
```

Now we can specify the high level inputs that describe the turbine

```python
opt = {}
opt["WISDEM"] = {}
opt["WISDEM"]["DriveSE"] = {}
opt["WISDEM"]["DriveSE"]["direct"] = True
opt["WISDEM"]["DriveSE"]["hub"] = {}
opt["WISDEM"]["DriveSE"]["hub"]["hub_gamma"] = 2.0
opt["WISDEM"]["DriveSE"]["hub"]["spinner_gamma"] = 1.5
opt["WISDEM"]["DriveSE"]["gamma_f"] = 1.35
opt["WISDEM"]["DriveSE"]["gamma_m"] = 1.3
```

(continues on next page)
A number of blade properties and other parameters are needed for the hub and spinner designs,

```
prob["blade_mass"] = 65252.0
prob["blades_mass"] = 3 * prob["blade_mass"]
prob["pitch_system.BRFM"] = 26648449.0
prob["pitch_system_scaling_factor"] = 0.75
prob["blade_root_diameter"] = 5.2
prob["flange_t2shell_t"] = 6.0
prob["flange_OD2hub_D"] = 0.6
prob["flange_ID2flange_OD"] = 0.8
prob["hub_in2out_circ"] = 1.2
prob["hub_stress_concentration"] = 3.0
prob["n_front_brackets"] = 5
prob["n_rear_brackets"] = 5
prob["clearance_hub_spinner"] = 0.5
prob["spin_hole_incr"] = 1.2
prob["spinner_gust_ws"] = 70.0
prob["hub_diameter"] = 7.94
prob["blades_I"] = np.r_[4.12747714e08, 1.97149973e08, 1.54854398e08, np.zeros(3)]
```

Next is the layout of the drivetrain and the initial conditions of some of the design variables,

```
prob["bear1.bearing_type"] = "CARB"
prob["bear2.bearing_type"] = "SRB"
prob["L_12"] = 1.2
prob["L_h1"] = 1.0
prob["L_generator"] = 2.15  # 2.75
prob["overhang"] = 12.0313
prob["drive_height"] = 5.614
prob["tilt"] = 6.0
prob["access_diameter"] = 2.0

myones = np.ones(2)
prob["lss_diameter"] = 3.0 * myones
prob["nose_diameter"] = 2.2 * myones
prob["lss_wall_thickness"] = 0.1 * myones
prob["nose_wall_thickness"] = 0.1 * myones
prob["bedplate_wall_thickness"] = 0.05 * np.ones(4)
prob["bear1.D_shaft"] = 2.2
prob["bear2.D_shaft"] = 2.2
```

Finally, the material properties that are used in the design are set. Here we assume that the shaft and nose/turret are made of a steel with a slightly higher carbon content than the bedplate. The hub is cast iron and the spinner is made of glass fiber.
The simulation can now be run. If doing an optimization, we select finite differencing around the total derivatives instead of the partial derivatives.

```python
if opt_flag:
    prob.model.approx_totals()
    prob.run_driver()
else:
    prob.run_model()
save_data("drivetrain_example", prob)
```

All of the inputs and outputs are displayed on the screen, followed by a curated list of values relating to the optimization problem.

```python
# prob.model.list_inputs(units=True)
# prob.model.list_outputs(units=True)

# Print out the objective, design variables and constraints
print("nacelle_mass:", prob["nacelle_mass"])
print("")
print("L_h1:", prob["L_h1"])  
print("L_12:", prob["L_12"])  
print("L_lss:", prob["L_lss"])  
print("L_nose:", prob["L_nose"])  
print("L_generator:", prob["L_generator"])  
print("L_bedplate:", prob["L_bedplate"])  
print("H_bedplate:", prob["H_bedplate"])  
print("hub_diameter:", prob["hub_diameter"])  
print("lss_diameter:", prob["lss_diameter"])  
print("lss_wall_thickness:", prob["lss_wall_thickness"])  
print("nose_diameter:", prob["nose_diameter"])  
print("nose_wall_thickness:", prob["nose_wall_thickness"])  
print("bedplate_wall_thickness:", prob["bedplate_wall_thickness"])  
print(""")
print("constr_lss_vonmises:", prob["constr_lss_vonmises"]).flatten())
print("constr_bedplate_vonmises:", prob["constr_bedplate_vonmises"]).flatten())
print("constr_mb1_defl:", prob["constr_mb1_defl"])
print("constr_mb2_defl:", prob["constr_mb2_defl"])  
print("constr_hub_diameter:", prob["constr_hub_diameter"])```

(continues on next page)
print("constr_length:", prob["constr_length"])  
print("constr_height:", prob["constr_height"])  
print("constr_access:", prob["constr_access"])  
print("constr_ecc:", prob["constr_ecc"])  

The screen output should look something like the following,  

Optimization terminated successfully  
    (Exit mode 0)  
    Current function value: [0.22768958]  
    Iterations: 31  
    Function evaluations: 47  
    Gradient evaluations: 31  
Optimization Complete  
-----------------------------------  
...

nacelle_mass: [227689.58069311]
L_h1: [0.1]
L_12: [1.04818685]
L_lss: [1.14818685]
L_nose: [3.15181315]
L_generator: [2.15]
L_bedplate: [4.92501211]
H_bedplate: [4.86709905]
hub_diameter: [7.78723633]
lss_diameter: [2.67134366 2.67189989]
lss_wall_thickness: [0.08664636 0.08664614]
nose_diameter: [2.08469731 2.08525375]
nose_wall_thickness: [0.08469731 0.08525375]
bedplate_wall_thickness: [0.004 0.004 0.004 0.03417892]
constr_lss_vonmises: [0.07861671 0.0784489 0.09307185 0.19116645]
constr_bedplate_vonmises: [0.93281517 0.85272928 0.77435574 0.69658358 0.62163064 0.55110369 0.41195844 0.31955169 0.3206749 0.34412109 0.17707953 0.20135497 0.07195264 0.06577744]
constr_mb1_defl: [0.18394075]
constr_mb2_defl: [0.02072981]
constr_hub_diameter: [0.58190497]
constr_length: [1.67501211]
constr_height: [4.86709905]
constr_access: [[0.00000000e+00 1.35447209e-13]
[1.56319402e-13 4.44089210e-16]]
constr_ecc: [0.05791306]
Geared Design

This example is for design sizing (diameter and thickness) of the shaft, nose/turret, and bedplate wall thickness of the direct-drive layout shown in Fig. 5.14 and in Fig. 5.15.

Specifically, the design variables are,
- \( L_{h1} \)
- \( L_{12} \)
- \( L_{hss} \)
- \( D_{hss} \)
- \( D_{hub} \)
The design script is quite similar to the direct-drive version, so we will walk through it more succinctly, noting distinctions with the example above. The design script starts with importing the required libraries,

```python
import numpy as np
import openmdao.api as om
from wisdem.commonse.fileIO import save_data
from wisdem.drivetrainse.drivetrain import DrivetrainSE
```

The only difference in the modeling options is to set the direct drive flag to False,

```python
opt_flag = False  # True
```

Next, we instantiate the OpenMDAO problem and assign the drivetrain as the model of the problem,

```python
prob = om.Problem()
prob.model = DrivetrainSE(modeling_options=opt, n_dlcs=1)
```

For the optimization problem setup, the only differences are swapping in the high speed shaft parameters instead of the nose/turret design variables and constraints. The bedplate I-beam geometry parameters have also been added as design variables. There also are not any constraints for maintenance accessibility because of the easier access already afforded in this layout.

```python
if opt_flag:
    # Choose the optimizer to use
    prob.driver = om.ScipyOptimizeDriver()
    prob.driver.options["optimizer"] = "SLSQP"
```

(continues on next page)
prob.driver.options["tol"] = 1e-5

# Add objective
prob.model.add_objective("nacelle_mass", scaler=1e-6)

# Add design variables, in this case the tower diameter and wall thicknesses
prob.model.add_design_var("L_12", lower=0.1, upper=5.0)
prob.model.add_design_var("L_h1", lower=0.1, upper=5.0)
prob.model.add_design_var("L_hss", lower=0.1, upper=5.0)
prob.model.add_design_var("hub_diameter", lower=2.0, upper=5.0)
prob.model.add_design_var("lss_diameter", lower=0.5, upper=6.0)
prob.model.add_design_var("lss_wall_thickness", lower=4e-3, upper=5e-1, ref=1e-2)
prob.model.add_design_var("hss_diameter", lower=0.5, upper=6.0)
prob.model.add_design_var("hss_wall_thickness", lower=4e-3, upper=5e-1, ref=1e-2)
prob.model.add_design_var("bedplate_web_thickness", lower=4e-3, upper=5e-1, ref=1e-2)
prob.model.add_design_var("bedplate_flange_thickness", lower=4e-3, upper=5e-1, ref=1e-2)
prob.model.add_design_var("bedplate_flange_width", lower=0.1, upper=2.0)

# Add constraints on the tower design
prob.model.add_constraint("constr_lss_vonmises", upper=1.0)
prob.model.add_constraint("constr_hss_vonmises", upper=1.0)
prob.model.add_constraint("constr_bedplate_vonmises", upper=1.0)
prob.model.add_constraint("constr_mb1_defl", upper=1.0)
prob.model.add_constraint("constr_mb2_defl", upper=1.0)
prob.model.add_constraint("constr_hub_diameter", lower=0.0)
prob.model.add_constraint("constr_length", lower=0.0)
prob.model.add_constraint("constr_height", lower=0.0)
prob.model.add_constraint("constr_length", lower=0.0)
prob.model.add_constraint("constr_height", lower=0.0)

With the optimization problem defined, the OpenMDAO problem is activated

prob.setup()

Now we can specify the high level inputs that describe the turbine. Whereas the direct-drive example was for a 15-MW machine, this one is for a 5-MW machine with much smaller components.

opt = {}
opt["WISDEM"] = {}
opt["WISDEM"]["DriveSE"] = {}
opt["WISDEM"]["DriveSE"]["direct"] = False
opt["WISDEM"]["DriveSE"]["hub"] = {}
opt["WISDEM"]["DriveSE"]["hub"]["hub_gamma"] = 2.0
opt["WISDEM"]["DriveSE"]["hub"]["spinner_gamma"] = 1.5
opt["WISDEM"]["DriveSE"]["gamma_f"] = 1.35
opt["WISDEM"]["DriveSE"]["gamma_m"] = 1.3
opt["WISDEM"]["DriveSE"]["gamma_n"] = 1.0
opt["WISDEM"]["RotorSE"] = {}
opt["WISDEM"]["RotorSE"]["n_pc"] = 20
opt["materials"] = {}
opt["materials"]["n_mat"] = 4
opt["flags"] = {}
opt["flags"]["generator"] = False
The blade properties and other parameters needed for the hub and spinner designs reflect the smaller machine size,

\[
\begin{align*}
\text{prob}['\text{blade\_mass}'] &= 16403.0 \\
\text{prob}['\text{blades\_mass}'] &= 3 \times \text{prob}['\text{blade\_mass}'] \\
\text{prob}['\text{pitch\_system.BRFM}'] &= 14239550.0 \\
\text{prob}['\text{pitch\_system\_scaling\_factor}'] &= 0.54 \\
\text{prob}['\text{blade\_root\_diameter}'] &= 3.542 \\
\text{prob}['\text{flange\_t2shell\_t}'] &= 4.0 \\
\text{prob}['\text{flange\_OD2hub\_D}'] &= 0.5 \\
\text{prob}['\text{flange\_ID2flange\_OD}'] &= 0.8 \\
\text{prob}['\text{hub\_in2out\_circ}'] &= 1.2 \\
\text{prob}['\text{hub\_stress\_concentration}'] &= 2.5 \\
\text{prob}['\text{n\_front\_brackets}'] &= 3 \\
\text{prob}['\text{n\_rear\_brackets}'] &= 3 \\
\text{prob}['\text{clearance\_hub\_spinner}'] &= 1.0 \\
\text{prob}['\text{spin\_hole\_incr}'] &= 1.2 \\
\text{prob}['\text{spinner\_gust\_ws}'] &= 70.0 \\
\text{prob}['\text{hub\_diameter}'] &= 3.0 \\
\text{prob}['\text{blades\_I}'] &= \text{np.r}_[36494351.0, 17549243.0, 14423664.0, \text{np.zeros}(3)]
\end{align*}
\]

Next is the layout of the drivetrain and the initial conditions of some of the design variables, with additional inputs needed for the gearbox design,

\[
\begin{align*}
\text{prob}['\text{bear1.bearing\_type}'] &= \text{''CARB''} \\
\text{prob}['\text{bear2.bearing\_type}'] &= \text{''SRB''} \\
\text{prob}['\text{L\_12}'] &= 0.368 \\
\text{prob}['\text{L\_h1}'] &= 1.912 \\
\text{prob}['\text{L\_hss}'] &= 1.5 \\
\text{prob}['\text{L\_generator}'] &= 2.0 \\
\text{prob}['\text{L\_gearbox}'] &= 1.5 \\
\text{prob}['\text{overhang}'] &= 5.0 \\
\text{prob}['\text{drive\_height}'] &= 2.3 \\
\text{prob}['\text{tilt}'] &= 5.0 \\
\text{prob}['\text{planet\_numbers}'] &= \text{np.array}([[3, 3, 0]]) \\
\text{prob}['\text{gear\_configuration}'] &= \text{''eep''} \\
\text{prob}['\text{gear\_ratio}'] &= 96.0 \\
\text{myones} &= \text{np.ones}(2) \\
\text{prob}['\text{lss\_diameter}'] &= 1.0 \times \text{myones} \\
\text{prob}['\text{hss\_diameter}'] &= 0.288 \times \text{myones} \\
\text{prob}['\text{lss\_wall\_thickness}'] &= 0.288 \times \text{myones} \\
\text{prob}['\text{hss\_wall\_thickness}'] &= 0.1 \times \text{myones} \\
\text{prob}['\text{bedplate\_web\_thickness}'] &= 0.1 \\
\text{prob}['\text{bedplate\_flange\_thickness}'] &= 0.1 \\
\text{prob}['\text{bedplate\_flange\_width}'] &= 1.0 \\
\text{prob}['\text{bear1.D\_shaft}'] &= 2.2 \\
\text{prob}['\text{bear2.D\_shaft}'] &= 2.2
\end{align*}
\]

Finally, the material properties and assumptions are the same as above,

\[
\begin{align*}
\text{prob}['\text{E\_mat}'] &= \text{np.c}_[200e9 \times \text{np.ones}(3), 205e9 \times \text{np.ones}(3), 118e9 \times \text{np.ones}(3), [4.46e10, 1.7e10, 1.67e10]].\text{T} \\
\text{prob}['\text{G\_mat}'] &= \text{np.c}_[79.3e9 \times \text{np.ones}(3), 80e9 \times \text{np.ones}(3), 47.6e9 \times \text{np.ones}(3), [3.27e9, 3.48e9, 3.5e9]].\text{T}
\end{align*}
\]
prob["Xt_mat"] = np.c_[450e6 * np.ones(3), 814e6 * np.ones(3), 310e6 * np.ones(3), [6.092e8, 3.81e7, 1.529e7]].T
prob["rho_mat"] = np.r_[7800.0, 7850.0, 7200.0, 1940.0]
prob["sigma_y_mat"] = np.r_[345e6, 485e6, 265e6, 18.9e6]
prob["unit_cost_mat"] = np.r_[0.7, 0.9, 0.5, 1.9]
prob["lss_material"] = prob["hss_material"] = "steel_drive"
prob["bedplate_material"] = "steel"
prob["hub_material"] = "cast_iron"
prob["spinner_material"] = "glass_uni"
prob["material_names"] = ["steel", "steel_drive", "cast_iron", "glass_uni"]

The simulation can now be run. If doing an optimization, the geared simulation takes longer than the direct-drive version due to the internal design iterations for the gearbox,

```python
if opt_flag:
    prob.model.approx_totals()
    prob.run_driver()
else:
    prob.run_model()
save_data("drivetrain_example", prob)
```

All of the inputs and outputs are displayed on the screen, followed by a curated list of values relating to the optimization problem.

```python
prob.model.list_inputs(units=True)
prob.model.list_outputs(units=True)

# Print out the objective, design variables and constraints
print("nacelle_mass:", prob["nacelle_mass"])  
print("")
print("L_h1:", prob["L_h1"])  
print("L_12:", prob["L_12"])  
print("L_lss:", prob["L_lss"])  
print("L_hss:", prob["L_hss"])  
print("L_generator:", prob["L_generator"])  
print("L_gearbox:", prob["L_gearbox"])  
print("L_bedplate:", prob["L_bedplate"])  
print("H_bedplate:", prob["H_bedplate"])  
print("hub_diameter:", prob["hub_diameter"])  
print("lss_diameter:", prob["lss_diameter"])  
print("lss_wall_thickness:", prob["lss_wall_thickness"])  
print("hss_diameter:", prob["hss_diameter"])  
print("hss_wall_thickness:", prob["hss_wall_thickness"])  
print("bedplate_web_thickness:", prob["bedplate_web_thickness"])  
print("bedplate_flange_thickness:", prob["bedplate_flange_thickness"])  
print("bedplate_flange_width:", prob["bedplate_flange_width"])  
print("")
print("constr_lss_vonmises:", prob["constr_lss_vonmises"]).flatten())
print("constr_hss_vonmises:", prob["constr_hss_vonmises"]).flatten())
print("constr_bedplate_vonmises:", prob["constr_bedplate_vonmises"]).flatten())
print("constr_mb1_defl:", prob["constr_mb1_defl"])  
print("constr_mb2_defl:", prob["constr_mb2_defl"])  
```

(continues on next page)
print("constr_hub_diameter:", prob["constr_hub_diameter"])
print("constr_length:", prob["constr_length"])
print("constr_height:", prob["constr_height"])

The screen output should look something like the following,

Optimization terminated successfully  (Exit mode 0)
   Current function value: [0.14948563]
   Iterations: 12
   Function evaluations: 12
   Gradient evaluations: 12
Optimization Complete

...nacelle_mass: [149485.62543167]
L_h1: [0.22298585]
L_12: [0.1]
L_lss: [0.42298585]
L_hss: [0.52650472]
L_generator: [2.]
L_gearbox: [1.512]
L_bedplate: [4.44451325]
H_bedplate: [1.69326612]
hub_diameter: [5.]
lss_diameter: [0.69921744 0.70132537]
lss_wall_thickness: [0.2878711 0.28786974]
hss_diameter: [0.5 0.5]
hss_wall_thickness: [0.0998473 0.0998473]
bedplate_web_thickness: [0.09595255]
bedplate_flange_thickness: [0.09892329]
bedplate_flange_width: [0.1]
constr_lss_vonmises: [0.98187486 0.9993906 0.99831682 0.99952247]
constr_hss_vonmises: [0.03251358 0.03215481]
constr_bedplate_vonmises: [1.51989918e-03 9.77245633e-03 6.01676078e-01 5.99836120e-01
  6.33223628e-01 8.63086490e-02 8.72758878e-02 4.39265181e-02
  4.30670107e-02 1.50407068e-04 2.08972837e-07 1.51990174e-03
  2.02074935e-02 6.91033399e-01 6.92826935e-01 7.50983681e-01
  1.07687995e-01 1.12023851e-01 4.39357428e-02 4.30706497e-02
  1.51237490e-04 1.59815686e-07]
constr_mbl1_defl: [0.07958005]
constr_mbl2_defl: [0.00904025]
constr_hub_diameter: [0.09206083]
constr_length: [-2.18136176e-10]
constr_height: [1.69326612]
5.7.7 Generator Model Examples

Understanding the generator examples is most easily done with the companion report for GeneratorSE. The design variables in the examples relate to both the electromagnetic and structural design. The five generator example files relate to the five different generator architectures available:

- **PMSG-Outer**: Permanent magnet synchronous generator (outer generator - inner stator)
- **PMSG-Disc**: Permanent magnet synchronous generator (inner generator - outer stator) with solid disc stator support
- **PMSG-Arms**: Permanent magnet synchronous generator (inner generator - outer stator) with arm/spoke stator support
- **EESG**: Electrically excited synchronous generator
- **DFIG**: Doubly fed induction generator
- **SCIG**: Squirrel-cage induction generator

Each of the technologies have slightly different sets of required inputs so while the examples follow the same pattern, the specific design variables and constraints will vary from one to the other. For brevity, only the `pmsg_outer.py` script is presented here.

First, we import the modules we want to use,

```python
import numpy as np
import openmdao.api as om
import wisdem.commonse.fileIO as fio
from wisdem.drivetrainse.generator import Generator
```

Next, we initialize the problem and set some script parameters,

```python
# Whether or not to run optimization
opt_flag = False

# Number of points in efficiency table
n_pc = 20

# Initialize problem instance
prob = om.Problem()
prob.model = Generator(design="pmsg_outer", n_pc=n_pc)
```

If running an optimization, we set the design variables, constraints, and objectives. These are most easily understood with the report linked above. Note that the optimization driver in the script is SLSQP from the OpenMDAO Scipy Driver, as that is available in the regular Conda-based installation of WISDEM. However, we have had some better experience using the CONMIN driver available in the pyOptSparse library.

```python
if opt_flag:
    # Add optimizer and set-up problem (using user defined input on objective function)
    prob.driver = om.ScipyOptimizeDriver()
    prob.driver.options["optimizer"] = "SLSQP"
    prob.driver.options["tol"] = 1e-6

    # prob.driver = om.pyOptSparseDriver()
    # prob.driver.options["optimizer"] = 'CONMIN' # 'SNOPT'
```

(continues on next page)
# Specificiency target efficiency (%)

\[ \text{Eta\_Target} = 0.955 \]

# Design variables

```python
prob.model.add_design_var("rad_ag", lower=3.0, upper=6.0)
prob.model.add_design_var("len_s", lower=1.5, upper=2.5)
prob.model.add_design_var("h_s", lower=0.1, upper=1.00)
prob.model.add_design_var("p", lower=50.0, upper=100.0, ref=1e2)
prob.model.add_design_var("h_m", lower=0.01, upper=0.2, ref=1e-2)
prob.model.add_design_var("h_yr", lower=0.035, upper=0.22, ref=1e-2)
prob.model.add_design_var("h_ys", lower=0.035, upper=0.22, ref=1e-2)
prob.model.add_design_var("B_tmax", lower=1, upper=2.0)
prob.model.add_design_var("t_r", lower=0.05, upper=0.3, ref=1e-2)
prob.model.add_design_var("t_s", lower=0.05, upper=0.3, ref=1e-2)
prob.model.add_design_var("h_ss", lower=0.04, upper=0.2, ref=1e-2)
prob.model.add_design_var("h_sr", lower=0.04, upper=0.2, ref=1e-2)
```

# Constraints

```python
prob.model.add_constraint("B_symax", lower=0.0, upper=2.0)
prob.model.add_constraint("B_rymax", lower=0.0, upper=2.0)
prob.model.add_constraint("b_t", lower=0.01, ref=1e-2)
prob.model.add_constraint("B_g", lower=1.19, upper=1.2)
prob.model.add_constraint("A_Cuscalc", lower=5.0, upper=600, ref=1e1)
prob.model.add_constraint("K_rad", lower=0.15, upper=0.3)
prob.model.add_constraint("Slot_aspect_ratio", lower=4.0, upper=10.0)
prob.model.add_constraint("generator_efficiency", lower=Eta\_Target, indices=[-1])
prob.model.add_constraint("A_1", upper=110000.0, ref=1e6, indices=[-1])
prob.model.add_constraint("T_e", lower=21.03e6, upper=21.1e6, ref=1e6)
prob.model.add_constraint("J_actual", lower=3, upper=6, indices=[-1])
```

# Drivetrain

```python
prob.model.add_constraint("con_uar", lower=1e-2, ref=1e-2)
prob.model.add_constraint("con_yar", lower=1e-2, ref=1e-2)
prob.model.add_constraint("con_uas", lower=1e-2, ref=1e-2)
prob.model.add_constraint("con_yas", lower=1e-2, ref=1e-2)
```

Objective function = "generator\_cost"

```python
prob.model.add_objective(Objective_function, scaler=1e-5)
```

Now, the long list of electromechanical and structural inputs are set prior to execution,

```python
prob.setup()
```

# Specify Target machine parameters

```python
prob["machine_rating"] = 15e6
prob["shaft\_rpm"] = np.linspace(5.0, 7.56, n\_pc)
prob["rated\_torque"] = 21030561.0
prob["P\_mech"] = 15354206.45251639
```

# Drivetrain

```python
prob["D\_shaft"] = 3.0
prob["D\_nose"] = 2.2
```
# Generator inputs
prob["B_g"] = 1.38963289
prob["B_r"] = 1.279
prob["B_max"] = 1.63478983
prob["B_symax"] = 1.63455514
prob["B_tmax"] = 1.88378905
prob["E_p"] = 3300.0 / np.sqrt(3)
prob["L_s"] = 0.01138752
prob["N_c"] = 2
prob["P_Fe0e"] = 1.0
prob["P_Fe0h"] = 4.0
prob["R_s"] = 0.02457052
prob["S"] = 240
prob["S_N"] = -0.002
prob["alpha_p"] = 1.0995574287564276 # 0.5*np.pi*0.7
prob["b"] = 2.0
prob["b_m"] = 0.1288363023
prob["b_max"] = 0.45
prob["b_ro"] = 0.004
prob["b_s"] = 0.0512796888
prob["b_max"] = 0.45
prob["b_so"] = 0.004
prob["b_t"] = 0.08159225451
prob["c"] = 5.0
prob["cfini"] = 0.85
prob["freq"] = 60.0
prob["h_0"] = 0.005
prob["h_0"] = 0.004
prob["h_m"] = 0.0995385122
prob["h_s"] = 0.37694491492
prob["h_max"] = 0.04677
prob["h_max"] = 0.04651
prob["h_t"] = 0.3859449149
prob["h_max"] = 0.005
prob["h_0"] = 0.0361759511
prob["h_0"] = 0.0361814528
prob["k_fes"] = 0.8
prob["k_fillr"] = 0.55
prob["k_fills"] = 0.65
prob["k_s"] = 0.2
prob["len_s"] = 2.23961662
prob["m"] = 3
prob["mu_0"] = 1.2566370614359173e-06 # np.pi*4e-7
prob["mu_0"] = 1.06
prob["p"] = 100
prob["phi"] = 1.5707963267948966 # 90 deg
prob["q1"] = 5
prob["q2"] = 4
prob["rad_ag"] = 0.5 * 10.25246718
prob["ratio_mw2pp"] = 0.8
prob["resist_Cu"] = 2.52e-8 # 1.8e-8*1.4

(continues on next page)
prob["sigma"] = 60e3
prob["t_r"] = 0.061
prob["t_s"] = 0.061
prob["tau_p"] = 0.1610453779
prob["tau_s"] = 0.1339360726
prob["u_allow_pcent"] = 8.5
prob["y_allow_pcent"] = 1.0
prob["y_tau_p"] = 0.8 # 12./15.
prob["y_tau_pr"] = 0.8333333 # 10. / 12
prob["z_allow_deg"] = 0.05

# Specific costs
prob["C_Cu"] = 4.786 # Unit cost of Copper $/kg
prob["C_Fe"] = 0.556 # Unit cost of Iron $/kg
prob["C_Fes"] = 0.50139 # specific cost of Structural_mass $/kg
prob["C_PM"] = 95.0

# Material properties
prob["rho_Fe"] = 7700.0 # Steel density Kg/m3
prob["rho_Fes"] = 7850 # structural Steel density Kg/m3
prob["rho_Copper"] = 8900.0 # copper density Kg/m3
prob["rho_PM"] = 7450.0 # typical density Kg/m3 of neodymium magnets (added 2019 09 18)

The script can now be executed. If running an optimization, we activate finite differencing for the total derivatives instead of the partials,

```python
if opt_flag:
    prob.model.approx_totals()
    prob.run_driver()
else:
    prob.run_model()
```

prob.model.list_inputs(units=True)
prob.model.list_outputs(units=True)

## 5.7.8 8. Plant Finance Example

Using the plant finance model only requires a short script. The plant finance model depends on the costs of the turbine, balance of station, expected operational expenditures, energy production and a few financial parameters. This example shows some representative values for the financial parameters, including a fixed charge rate for COE, the construction financing charge rate, the tax rate for OPEX tax deductions, the time for plant construction, the project lifetime and the sea depth (which would be 0.0 for a land-based plant). In a full turbine analysis, some of these input values come from upstream components or from the turbine design.

```bash
#!/usr/bin/env python3

# Import the libraries
import openmdao.api as om
```

(continues on next page)
from wisdem.plant_basics import PlantFinance

# Initialize the OpenMDAO instance
prob = om.Problem()
prob.model = PlantFinance()
prob.setup()

# Set variable inputs with intended units
prob.set_val("machine_rating", 2.32e3, units="kW")
prob.set_val("tcc_per_kW", 1093, units="USD/kW")
prob.set_val("turbine_number", 87)
prob.set_val("opex_per_kW", 43.56, units="USD/kW/yr")
prob.set_val("fixed_charge_rate", 0.079216644)
prob.set_val("bos_per_kW", 517.0, units="USD/kW")
prob.set_val("wake_loss_factor", 0.15)
prob.set_val("turbine_aep", 9915.95e3, units="kW*h")

# Run the model once
prob.run_model()

# Print all inputs and outputs to the screen
prob.model.list_inputs(units=True)
prob.model.list_outputs(units=True)

The screen output is,

10 Input(s) in 'model'

<table>
<thead>
<tr>
<th>varname</th>
<th>value</th>
<th>units</th>
</tr>
</thead>
<tbody>
<tr>
<td>machine_rating</td>
<td>[2320.]</td>
<td>kW</td>
</tr>
<tr>
<td>tcc_per_kW</td>
<td>[1093.]</td>
<td>USD/kW</td>
</tr>
<tr>
<td>offset_tcc_per_kW</td>
<td>[0.]</td>
<td>USD/kW</td>
</tr>
<tr>
<td>bos_per_kW</td>
<td>[517.]</td>
<td>USD/kW</td>
</tr>
<tr>
<td>opex_per_kW</td>
<td>[43.56]</td>
<td>USD/kW/yr</td>
</tr>
<tr>
<td>park_aep</td>
<td>[0.]</td>
<td>kW*h</td>
</tr>
<tr>
<td>turbine_aep</td>
<td>[9915950.]</td>
<td>kW*h</td>
</tr>
<tr>
<td>wake_loss_factor</td>
<td>[0.15]</td>
<td>None</td>
</tr>
<tr>
<td>fixed_charge_rate</td>
<td>[0.079216644]</td>
<td>None</td>
</tr>
<tr>
<td>turbine_number</td>
<td>87.0</td>
<td>Unavailable</td>
</tr>
</tbody>
</table>

2 Explicit Output(s) in 'model'

<table>
<thead>
<tr>
<th>varname</th>
<th>value</th>
<th>units</th>
</tr>
</thead>
<tbody>
<tr>
<td>lcoe</td>
<td>[0.04709575]</td>
<td>USD/kW/h</td>
</tr>
<tr>
<td>plant_aep</td>
<td>[7.33284502e+08]</td>
<td>USD/kW/h</td>
</tr>
</tbody>
</table>
5.7.9 9. Floating Platform Examples

**Warning:** In our use of FloatingSE, we have not been able to obtain conceptual platform geometry results that we trust are valid and worth pursuing further. This may be due to a fundamental error in the formulation and implementation of FloatingSE. It may also be due to the inherent limitations in using steady-state or quasi-static analysis methods to tackle a problem that is driven by its dynamic nature and dynamic loads. NREL currently advises against reliance on FloatingSE. Instead, we are developing a multifidelity floating turbine and platform design capability in the Wind Energy with Integrated Servo-control (WEIS) project.

5.7.10 10. CCBlade Examples

Three examples are shown below. The first is a complete setup for the *NREL 5-MW model*, the second shows how to model blade *precurvature*, and the final shows how to get the provided *analytic gradients*. Each complete example is also included within the WISDEM/examples/10_ccblade directory.

**List of Examples**

- 10. CCBlade Examples
  - NREL 5-MW
  - Precurve
  - Gradients

**NREL 5-MW**

One example of a CCBlade application is the simulation of the NREL 5-MW reference model’s aerodynamic performance. First, define the geometry and atmospheric properties.

```python
import numpy as np
from math import pi
import matplotlib.pyplot as plt

from wisdem.ccblade.ccblade import CCAirfoil, CCBlade

plot_flag = False

# geometry
Rhub = 1.5
Rtip = 63.0
r = np.array([2.8667, 5.6000, 8.3333, 11.7500, 15.8500, ...
```

(continues on next page)
chord = np.array([19.9500, 24.0500, 28.1500, 32.2500, 36.3500, 40.4500, 44.5500, 48.6500, 52.7500, 56.1667, 58.9000, 61.6333])

(continues on next page)
Airfoil aerodynamic data is specified using the CCAirfoil class. Rather than use the default constructor, this example uses the special constructor designed to read AeroDyn files directly CCAirfoil.initFromAerodynFile().

```python
afinit = CCAirfoil.initFromAerodynFile  # just for shorthand

# load all airfoils
airfoil_types = [0] * 8
airfoil_types[0] = afinit("../_airfoil_files/Cylinder1.dat")
airfoil_types[1] = afinit("../_airfoil_files/Cylinder2.dat")
airfoil_types[2] = afinit("../_airfoil_files/DU40_A17.dat")
airfoil_types[3] = afinit("../_airfoil_files/DU35_A17.dat")
airfoil_types[4] = afinit("../_airfoil_files/DU30_A17.dat")
airfoil_types[5] = afinit("../_airfoil_files/DU25_A17.dat")
airfoil_types[6] = afinit("../_airfoil_files/DU21_A17.dat")
airfoil_types[7] = afinit("../_airfoil_files/NACA64_A17.dat")

# place at appropriate radial stations
af_idx = [0, 0, 1, 2, 3, 3, 4, 5, 5, 6, 6, 7, 7, 7, 7, 7, 7, 7]

af = [0] * len(r)
for i in range(len(r)):
    af[i] = airfoil_types[af_idx[i]]
```

Next, construct the CCBlade object.

```python
# create CCBlade object
rotor = CCBlade(r, chord, theta, af, Rhub, Rtip, B, rho, mu, precone, tilt, yaw,
                shearExp, hubHt, nSector)
```
Evaluate the distributed loads at a chosen set of operating conditions.

```python
# set conditions
Uinf = 10.0
tsr = 7.55
pitch = 0.0
Omega = Uinf * tsr / Rtip * 30.0 / pi  # convert to RPM
azimuth = 0.0

# evaluate distributed loads
loads, derivs = rotor.distributedAeroLoads(Uinf, Omega, pitch, azimuth)
Np = loads["Np"]
Tp = loads["Tp"]
```

Plot the flapwise and lead-lag aerodynamic loading

```python
# plot
rstar = (r - Rhub) / (Rtip - Rhub)

# append zero at root and tip
rstar = np.concatenate([[0.0], rstar, [1.0]])
Np = np.concatenate([[0.0], Np, [0.0]])
Tp = np.concatenate([[0.0], Tp, [0.0]])

if plot_flag:
    plt.plot(rstar, Tp / 1e3, label="lead-lag")
    plt.plot(rstar, Np / 1e3, label="flapwise")
    plt.xlabel("blade fraction")
    plt.ylabel("distributed aerodynamic loads (kN)")
    plt.legend(loc="upper left")
    plt.grid()
    plt.show()
```

as shown in Figure 5.16.

To get the power, thrust, and torque at the same conditions (in both absolute and coefficient form), use the evaluate method. This is generally used for generating power curves so it expects array_like input. For this example a list of size one is used.

```python
outputs, derivs = rotor.evaluate([Uinf], [Omega], [pitch])
P = outputs["P"]
T = outputs["T"]
Q = outputs["Q"]

outputs, derivs = rotor.evaluate([Uinf], [Omega], [pitch], coefficients=True)
CP = outputs["CP"]
CT = outputs["CT"]
CQ = outputs["CQ"]
```

(continues on next page)
print("CP =", CP)
print("CT =", CT)
print("CQ =", CQ)

The result is

```python
>>> CP = [ 0.46488096]
>>> CT = [ 0.76926398]
>>> CQ = [ 0.0616323]
```

Note that the outputs are Numpy arrays (of length 1 for this example). To generate a nondimensional power curve ($\lambda$ vs $c_p$):

```python
# velocity has a small amount of Reynolds number dependence
tsr = np.linspace(2, 14, 50)
Omega = 10.0 * np.ones_like(tsr)
Uinf = Omega * np.pi / 30.0 * Rtip / tsr
pitch = np.zeros_like(tsr)
outputs, derivs = rotor.evaluate(Uinf, Omega, pitch, coefficients=True)

CP = outputs["CP"]
CT = outputs["CT"]
```

(continues on next page)
CQ = outputs["CQ"]

if plot_flag:
    plt.figure()
    plt.plot(tsr, CP)
    plt.xlabel("$\lambda$")
    plt.ylabel("$c_p$")
    plt.show()

Figure 5.17 shows the resulting plot.

CCBlade provides a few additional options in its constructor. The other options are shown in the following example with their default values.

```python
# create CCBlade object
rotor = CCBlade(r, chord, theta, af, Rhub, Rtip, B, rho, mu,
                 precone, tilt, yaw, shearExp, hubHt, nSector
                 tiploss=True, hubloss=True, wakerotation=True, usecd=True, iterRe=1)
```

The parameters `tiploss` and `hubloss` toggle Prandtl tip and hub losses respectively. The parameter `wakerotation` toggles wake swirl (i.e., $a' = 0$). The parameter `usecd` can be used to disable the inclusion of drag in the calculation of the induction factors (it is always used in calculations of the distributed loads). However, doing so may cause potential failure in the solution methodology (see [Nin13]). In practice, it should work fine, but special care for that particular case has not yet been examined, and the default implementation allows for the possibility of convergence failure. All four of these parameters are `True` by default. The parameter `iterRe` is for advanced usage. Referring to [Nin13], this parameter controls the number of internal iterations on the Reynolds number. One iteration is almost always sufficient, but for high accuracy in the Reynolds number `iterRe` could be set at 2. Anything larger than that is unnecessary.

5.7. Examples
Precurve

CCBlade can also simulate blades with precurve. This is done by using the `precurve` and `precurveTip` parameters. These correspond precisely to the `r` and `Rtip` parameters. Precurve is defined as the position of the blade reference axis in the `x`-direction of the blade-aligned coordinate system (`r` is the position in the `z`-direction of the same coordinate system). Presweep can be specified in the same manner, by using the `presweep` and `presweepTip` parameters (position in blade-aligned `y`-axis). Generally, it is advisable to set `precone=0` for blades with precurve. There is no loss of generality in defining the blade shape, and including a nonzero precone would change the rotor diameter in a nonlinear way. As an example, a downwind machine with significant curvature could be simulated using:

```python
precone = 0.0
precurve = np.linspace(0, 4.9, len(r)) ** 2
precurveTip = 25.0

# create CCBlade object
rotor = CCBlade(
    r,
    chord,
    theta,
    af,
    Rhub,
    Rtip,
    B,
    rho,
    mu,
    precone,
    tilt,
    yaw,
    shearExp,
    hubHt,
    nSector,
    precurve=precurve,
    precurveTip=precurveTip,
)
```

The shape of the blade is seen in Figure 5.18. Note that the radius of the blade is preserved because we have set the `precone` angle to zero.

Gradients

CCBlade optionally provides analytic gradients of every output with respect to all design variables. This is accomplished using an adjoint method (direct method is identical because there is only one state variable at each blade section). Partial derivatives are provided by Tapenade and hand calculations. Starting with the previous example for the NREL 5-MW reference model we add the keyword value `derivatives=True` in the constructor.

```python
# create CCBlade object
rotor = CCBlade(
    r, chord, theta, af, Rhub, Rtip, B, rho, mu, precone, tilt, yaw, shearExp, hubHt,
    nSector, derivatives=True
)
```

(continues on next page)
Now when we ask for the distributed loads, we also get the gradients. The gradients are returned as a dictionary containing 2D arrays. These can be accessed as follows:

```python
loads, derivs = rotor.distributedAeroLoads(Uinf, Omega, pitch, azimuth)
Np = loads["Np"]
Tp = loads["Tp"]
dNp = derivs["dNp"]
dTp = derivs["dTp"]

n = len(r)

# n x n (diagonal)
dNp_dr = dNp["dr"]
dNp_dchord = dNp["dchord"]
dNp_dtheta = dNp["dtheta"]
dNp_dpresweep = dNp["dpresweep"]

# n x n (tridiagonal)
dNp_dprecurve = dNp["dprecurve"]

# n x 1
dNp_dRhub = dNp["dRhub"]
```

(continues on next page)
Even though many of the matrices are diagonal, the full Jacobian is returned for consistency. We can compare against finite differencing as follows (with a randomly chosen station along the blade):

```python
idx = 8
delta = 1e-6 * r[idx]
r[idx] += delta

rotor_fd = CCBlade(  
    r, chord, theta, af, Rhub, Rtip, B, rho, mu, precone, tilt, yaw, shearExp, hubHt,   
    nSector, derivatives=False  
)

r[idx] -= delta

loads, derivs = rotor_fd.distributedAeroLoads(Uinf, Omega, pitch, azimuth)
Npd = loads["Np"]
Tpd = loads["Tp"]

dNp_dr_fd = (Npd - Np) / delta
dTp_dr_fd = (Tpd - Tp) / delta

print("(analytic) dNp_i/dr_i =", dNp_dr_fd[idx, idx])
print("(fin diff) dNp_i/dr_i =", dNp_dr_fd[idx])
print()
```

The output is:

```console
>>> (analytic) dNp_i/dr_i = 107.680395098  
>>> (fin diff) dNp_i/dr_i = 107.680370762
```

Similarly, when we compute thrust, torque, and power we also get the gradients (for either the non-dimensional or dimensional form). The gradients are also returned as a dictionary containing 2D Jacobians.

```python
loads, derivs = rotor.evaluate([Uinf], [Omega], [pitch])
P = loads["P"]
T = loads["T"]
Q = loads["Q"]

dP = derivs["dP"]
```
Let us compare the derivative of power against finite differencing for one of the scalar quantities (precone):

```python
delta = 1e-6 * precone
precone += delta

rotor_fd = CCBblade(
    r, chord, theta, af, Rhub, Rtip, B, rho, mu, precone, tilt, yaw, shearExp, hubHt,
    nSector, derivatives=False
)

precone -= delta

loads, derivs = rotor_fd.evaluate([Uinf], [Omega], [pitch], coefficients=False)
Pd = loads["P"]
Td = loads["T"]
Qd = loads["Q"]

dT_dprecone_fd = (Td - T) / delta
dQ_dprecone_fd = (Qd - Q) / delta
dP_dprecone_fd = (Pd - P) / delta

print("(analytic) dP/dprecone =", dP_dprecone_fd[0, 0])
print("(fin diff) dP/dprecone =", dP_dprecone_fd[0])
```

5.7. Examples 187
Finally, we compare the derivative of power against finite differencing for one of the vector quantities (\( r \)) at a random index:

```python
dx = 12
delta = 1e-6 * r[dx]
r[dx] += delta

rotor_fd = CCBlade(res, chord, theta, af, Rhub, Rtip, B, rho, mu, precone, tilt, yaw, shearExp, hubHt, ...
  nSector, derivatives=False)

r[dx] -= delta

loads, derivs = rotor_fd.evaluate([Uinf], [Omega], [pitch], coefficients=False)
Pd = loads["P"]
Td = loads["T"]
Qd = loads["Q"]

dT_dr_fd = (Td - T) / delta
dQ_dr_fd = (Qd - Q) / delta
dP_dr_fd = (Pd - P) / delta

print("(analytic) dP/dr_i =", dP_dr[0, dx])
print("(fin diff) dP/dr_i =", dP_dr_fd[0])
```

```
>>> (analytic) dP/dr_i = 848.368037506
>>> (fin diff) dP/dr_i = 848.355994992
```

For more comprehensive comparison to finite differencing, see the unit tests contained in `wisdem/test/test_ccblade/test_gradients.py`.
5.7.11 11. Airfoil Polar Preparation Example

AirfoilPrep.py can be accessed either through the command line or through Python. The command-line interface is the simplest but provides only a limited number of options. The Python interface is useful for more advanced preprocessing and for integration with other codes.

Command-Line Usage

From the terminal, to see the options, invoke help:

```
$ python airfoilprep.py -h
```

When using the command-line options, all files must be AeroDyn formatted files. The command line provides three main methods for working with files directly: 3-D stall corrections, high angle of attack extrapolation, and a blending operation. In all cases, you first specify the name (and path if necessary) of the file you want to work with:

```
$ python airfoilprep.py airfoil.dat
```

The following optional arguments are available

<table>
<thead>
<tr>
<th>flag</th>
<th>arguments</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td></td>
<td>display help</td>
</tr>
<tr>
<td>--stall3D</td>
<td>r/R c/r tsr</td>
<td>3-D rotational corrections near stall</td>
</tr>
<tr>
<td>--extrap</td>
<td>cdmax</td>
<td>high angle of attack extrapolation</td>
</tr>
<tr>
<td>--blend</td>
<td>other weight</td>
<td>blend with other file using specified weight</td>
</tr>
<tr>
<td>--out</td>
<td>outfile</td>
<td>specify a different name for output file</td>
</tr>
<tr>
<td>--plot</td>
<td></td>
<td>plot data (for diagnostic purposes) using matplotlib</td>
</tr>
<tr>
<td>--common</td>
<td></td>
<td>output airfoil data using a common set of angles of attack</td>
</tr>
</tbody>
</table>

Stall Corrections

The first method available from the command line is --stall3D, which reads the file, applies rotational corrections, and then writes the data to a separate file. This argument must specify the parameters used for the correction in the format --stall3D r/R c/r tsr, where r/R is the local radius normalized by the rotor radius, c/r is the local chord normalized by the local radius, and tsr is the local tip-speed ratio. For example, if airfoil.dat contained 2-D data with r/R=0.5, c/r=0.15, tsr=5.0, then we would apply rotational corrections to the airfoil using:

```
$ python airfoilprep.py airfoil.dat --stall3D 0.5 0.15 5.0
```

By default the output file will append _3D to the name. In the above example, the output file would be airfoil_3D.dat. However, this can be overridden with the --out option. To output to a file at /Users/Me/Airfoils/my_new_airfoil.dat

```
$ python airfoilprep.py airfoil.dat --stall3D 0.5 0.15 5.0
  > --out /Users/Me/Airfoils/my_new_airfoil.dat
```

Optionally, you can also plot the results (matplotlib must be installed) with the --plot flag. For example,

```
$ python airfoilprep.py DU21_A17.dat --stall3D 0.2 0.3 5.0 --plot
```
displays Figure 5.19 (only one Reynolds number shown) along with producing the output file.

AirfoilPrep.py can utilize data for which every Reynolds number uses a different set of angles of attack. However, some codes need data on a uniform grid of Reynolds number and angle of attack. To output the data on a common set of angles of attack, use the --common flag.

```
$ python airfoilprep.py airfoil.dat --stall3D 0.5 0.15 5.0 --common
```

**Angle of Attack Extrapolation**

The second method available from the command line is --extrap, which reads the file, applies high angle of attack extrapolations, and then writes the data to a separate file. This argument must specify the maximum drag coefficient to use in the extrapolation across the full +/- 180-degree range --extrap cdmax. For example, if `airfoil_3D.dat` contained 3D stall corrected data and `cdmax=1.3`, then we could extrapolate the airfoil using:

```
$ python airfoilprep.py airfoil_3D.dat --extrap 1.3
```

By default the output file will append _extrap to the name. In the above example, the output file would be `airfoil_3D_extrap.dat`. However, this can also be overridden with the --out flag. The --common flag is also useful here if a common set of angles of attack is needed.

The output can be plotted with the --plot flag. The command

```
$ python airfoilprep.py DU21_A17_3D.dat --extrap 1.3 --plot
```

displays Figure 5.20 (only one Reynolds number shown) along with producing the output file.
Blending

The final capability accessible from the command line is blending of airfoils. This is invoked through `--blend filename weight`, where `filename` is the name (and path if necessary) of a second file to blend with, and `weight` is the weighting used in the blending. The weight ranges on a scale of 0 to 1 where 0 returns the first airfoil and 1 the second airfoil. For example, the following command blends `airfoil1.dat` with `airfoil2.dat` with a weighting of 0.3 (conceptually the new airfoil would equal 0.7*`airfoil1.dat` + 0.3*`airfoil2.dat`).

```
$ python airfoilprep.py airfoil1.dat --blend airfoil2.dat 0.3
```

By default, the output file appends the names of the two files with a `+` sign, then appends the weighting using `_blend` and the value for the weight. In this example, the output file would be `airfoil1+airfoil2_blend0.3.dat`. Just like the previous case, the name of the output file can be overridden by using the `--out` flag. The `--common` flag is also useful here if a common set of angles of attack is needed. This data can also be plotted, but only the blended airfoil data will be shown. Direct comparison to the original data is not always possible, because the blend method allows for the specified airfoils to be defined at different Reynolds numbers. Blending first occurs across Reynolds numbers and then across angle of attack.

Python Usage

The Python interface allows for more flexible usage or integration with other programs. Descriptions of the interfaces for the classes contained in the module are contained in Module Documentation.

This complete example script can be found at WISDEM/examples/11_airfoilprep/example.py.

Airfoils can be created from AeroDyn formatted files,

```python
import os
import numpy as np
from wisdem.airfoilprep import Polar, Airfoil

airfoil = Airfoil.initFromAerodynFile("../_airfoil_files/DU21_A17.dat")
```

or they can be created directly from airfoil data.
\textbf{Re} = 7e6
\textbf{alpha} = [
  -14.50,  
  -12.01,  
  -11.00,  
  -9.98,   
  -8.12,   
  -7.62,   
  -7.11,   
  -6.60,   
  -6.50,   
  -6.00,   
  -5.50,   
  -5.00,   
  -4.50,   
  -4.00,   
  -3.50,   
  -3.00,   
  -2.50,   
  -2.00,   
  -1.50,   
  -1.00,   
  -0.50,   
  0.00,    
  0.50,    
  1.00,    
  1.50,    
  2.00,    
  2.50,    
  3.00,    
  3.50,    
  4.00,    
  4.50,    
  5.00,    
  5.50,    
  6.00,    
  6.50,    
  7.00,    
  7.50,    
  8.00,    
  8.50,    
  9.00,    
  9.50,    
  10.00,   
  10.50,   
  11.00,   
  11.50,   
  12.00,   
  12.50,   
  13.00,   
  13.50,   
  14.00,   
  14.50,   

(continues on next page)
15.00,
15.50,
16.00,
16.50,
17.00,
17.50,
18.00,
18.50,
19.00,
19.50,
20.00,
20.50,
]

c1 = [
   -1.050,
   -0.953,
   -0.900,
   -0.827,
   -0.536,
   -0.467,
   -0.393,
   -0.323,
   -0.311,
   -0.245,
   -0.178,
   -0.113,
   -0.048,
   0.016,
   0.080,
   0.145,
   0.208,
   0.270,
   0.333,
   0.396,
   0.458,
   0.521,
   0.583,
   0.645,
   0.706,
   0.768,
   0.828,
   0.888,
   0.948,
   0.996,
   1.046,
   1.095,
   1.145,
   1.192,
   1.239,
   1.283,
   1.324,
   1.358,
1.385, 1.403, 1.401, 1.358, 1.313, 1.287, 1.274, 1.272, 1.273, 1.273, 1.272, 1.273, 1.273, 1.273, 1.272, 1.273, 1.273, 1.275, 1.281, 1.284, 1.296, 1.306, 1.308, 1.308, 1.308, 1.308, 1.308, 1.307, 1.311, 1.325, 

cd = [ 
0.0567, 0.0271, 0.0303, 0.0287, 0.0124, 0.0109, 0.0092, 0.0083, 0.0089, 0.0082, 0.0074, 0.0069, 0.0065, 0.0063, 0.0061, 0.0058, 0.0057, 0.0057, 0.0057, 0.0057, 0.0057, 0.0057, 0.0057, 0.0058, 0.0058, 0.0058, ]
0.0059, 0.0061, 0.0063, 0.0066, 0.0071, 0.0079, 0.0090, 0.0103, 0.0113, 0.0122, 0.0131, 0.0139, 0.0147, 0.0158, 0.0181, 0.0211, 0.0255, 0.0301, 0.0347, 0.0401, 0.0468, 0.0545, 0.0633, 0.0722, 0.0806, 0.0900, 0.0987, 0.1075, 0.1170, 0.1270, 0.1368, 0.1464, 0.1562, 0.1664, 0.1770, 0.1878, 0.1987, 0.2100, ]

\[
\text{cm} = [ \\
-0.0378, -0.0349, -0.0361, -0.0464, -0.0821, -0.0924, -0.1015, -0.1073, -0.1083, -0.1112, -0.1146, -0.1172, \\
\]

(continues on next page)
<table>
<thead>
<tr>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-0.1194</td>
</tr>
<tr>
<td>-0.1213</td>
</tr>
<tr>
<td>-0.1232</td>
</tr>
<tr>
<td>-0.1252</td>
</tr>
<tr>
<td>-0.1268</td>
</tr>
<tr>
<td>-0.1282</td>
</tr>
<tr>
<td>-0.1297</td>
</tr>
<tr>
<td>-0.1310</td>
</tr>
<tr>
<td>-0.1324</td>
</tr>
<tr>
<td>-0.1337</td>
</tr>
<tr>
<td>-0.1350</td>
</tr>
<tr>
<td>-0.1363</td>
</tr>
<tr>
<td>-0.1374</td>
</tr>
<tr>
<td>-0.1385</td>
</tr>
<tr>
<td>-0.1395</td>
</tr>
<tr>
<td>-0.1403</td>
</tr>
<tr>
<td>-0.1406</td>
</tr>
<tr>
<td>-0.1398</td>
</tr>
<tr>
<td>-0.1390</td>
</tr>
<tr>
<td>-0.1378</td>
</tr>
<tr>
<td>-0.1369</td>
</tr>
<tr>
<td>-0.1353</td>
</tr>
<tr>
<td>-0.1338</td>
</tr>
<tr>
<td>-0.1317</td>
</tr>
<tr>
<td>-0.1291</td>
</tr>
<tr>
<td>-0.1249</td>
</tr>
<tr>
<td>-0.1213</td>
</tr>
<tr>
<td>-0.1177</td>
</tr>
<tr>
<td>-0.1142</td>
</tr>
<tr>
<td>-0.1103</td>
</tr>
<tr>
<td>-0.1066</td>
</tr>
<tr>
<td>-0.1032</td>
</tr>
<tr>
<td>-0.1002</td>
</tr>
<tr>
<td>-0.0971</td>
</tr>
<tr>
<td>-0.0940</td>
</tr>
<tr>
<td>-0.0909</td>
</tr>
<tr>
<td>-0.0883</td>
</tr>
<tr>
<td>-0.0865</td>
</tr>
<tr>
<td>-0.0854</td>
</tr>
<tr>
<td>-0.0849</td>
</tr>
<tr>
<td>-0.0847</td>
</tr>
<tr>
<td>-0.0850</td>
</tr>
<tr>
<td>-0.0858</td>
</tr>
<tr>
<td>-0.0869</td>
</tr>
<tr>
<td>-0.0883</td>
</tr>
<tr>
<td>-0.0901</td>
</tr>
<tr>
<td>-0.0922</td>
</tr>
<tr>
<td>-0.0949</td>
</tr>
<tr>
<td>-0.0980</td>
</tr>
<tr>
<td>-0.1017</td>
</tr>
<tr>
<td>-0.105</td>
</tr>
</tbody>
</table>
pl = Polar(Re, alpha, cl, cd, cm)

# second polar
Re = 9e6
alpha = [
    -14.24,
    -13.24,
    -12.22,
    -11.22,
    -10.19,
    -9.70,
    -9.18,
    -8.18,
    -7.19,
    -6.65,
    -6.13,
    -6.00,
    -5.50,
    -5.00,
    -4.50,
    -4.00,
    -3.50,
    -3.00,
    -2.50,
    -2.00,
    -1.50,
    -1.00,
    -0.50,
    0.00,
    0.50,
    1.00,
    1.50,
    2.00,
    2.50,
    3.00,
    3.50,
    4.00,
    4.50,
    5.00,
    5.50,
    6.00,
    6.50,
    7.00,
    7.50,
    8.00,
    9.00,
    9.50,
    10.00,
    10.50,
    11.00,
    11.50,
12.00,
12.50,
13.00,
13.50,
14.00,
14.50,
15.00,
15.50,
16.00,
16.50,
17.00,
17.50,
18.00,
18.50,
19.00,

] 

c1 = [ 
-1.229,
-1.148,
-1.052,
-0.965,
-0.867,
-0.822,
-0.769,
-0.756,
-0.690,
-0.616,
-0.542,
-0.525,
-0.451,
-0.382,
-0.314,
-0.251,
-0.189,
-0.120,
-0.051,
0.017,
0.085,
0.152,
0.219,
0.288,
0.354,
0.421,
0.487,
0.554,
0.619,
0.685,
0.749,
0.815,
0.879,
0.944,
1.008,
1.072, 1.135, 1.197, 1.256, 1.305, 1.390, 1.424, 1.458, 1.488, 1.512, 1.533, 1.549, 1.558, 1.470, 1.398, 1.354, 1.336, 1.333, 1.326, 1.329, 1.326, 1.321, 1.331, 1.333, 1.340, 1.362,
]
cd = [ 0.1461, 0.1263, 0.1051, 0.0886, 0.0740, 0.0684, 0.0605, 0.0279, 0.0189, 0.0166, 0.0152, 0.0117, 0.0105, 0.0097, 0.0092, 0.0091, 0.0089, 0.0089, 0.0088, 0.0088, 0.0088, 0.0088, 0.0088, 0.0087,
\[ cm = [ \\
0.0087, \\
0.0088, \\
0.0089, \\
0.0090, \\
0.0091, \\
0.0092, \\
0.0093, \\
0.0095, \\
0.0096, \\
0.0097, \\
0.0099, \\
0.0101, \\
0.0103, \\
0.0107, \\
0.0112, \\
0.0125, \\
0.0155, \\
0.0171, \\
0.0192, \\
0.0219, \\
0.0255, \\
0.0307, \\
0.0370, \\
0.0452, \\
0.0630, \\
0.0784, \\
0.0931, \\
0.1081, \\
0.1239, \\
0.1415, \\
0.1592, \\
0.1743, \\
0.1903, \\
0.2044, \\
0.2186, \\
0.2324, \\
0.2455, \\
-0.0378, \\
-0.0349, \\
-0.0361, \\
-0.0464, \\
-0.0821, \\
-0.0924, \\
-0.1015, \\
-0.1073, \\
-0.1083, \\
-0.1112, \\
-0.1146, \\
-0.1172, \\
-0.1194, \\
(continues on next page) \]
\[ p2 = \text{Polar}(\text{Re}, \alpha, c_l, c_d, c_m) \]
# create airfoil object (can contain as many polars as desired)
af = Airfoil([p1, p2])

Blending is easily accomplished just like in the command-line interface. There is no requirement that the two airfoils share a common set of angles of attack.

```python
airfoil1 = Airfoil.initFromAerodynFile("../_airfoil_files/DU21_A17.dat")
airfoil2 = Airfoil.initFromAerodynFile("../_airfoil_files/DU25_A17.dat")

# blend the two airfoils
airfoil_blend = airfoil1.blend(airfoil2, 0.3)
```

Applying 3-D corrections and high alpha extensions directly in Python allows for a few additional options as compared to the command-line version. The following example performs the same 3-D correction as in the command-line version, followed by an alternative 3-D correction that utilizes some of the optional inputs. See correction3D for more details on the optional parameters.

```python
r_over_R = 0.5
cord_over_r = 0.15
tsr = 5.0

# 3D stall correction
af3D_ex1 = af.correction3D(r_over_R, chord_over_r, tsr)

# a second example using the optional inputs
alpha_max_corr = 25  # apply full rotational correction only up to this angle of attack
alpha_linear_min = -3  # angle of attack to start evaluating slope of linear region
alpha_linear_max = 7  # angle of attack to stop evaluating slope of linear region

af3D_ex2 = af.correction3D(
    r_over_R, 
    chord_over_r, 
    tsr, 
    alpha_max_corr=alpha_max_corr, 
    alpha_linear_min=alpha_linear_min, 
    alpha_linear_max=alpha_linear_max,
)
```

The airfoil data can be extended to high angles of attack using the extrapolate method. Just like the previous method, a few optional parameters are available through the Python interface. The following example performs the same extrapolation as in the command-line version, followed by an alternative extrapolation that utilizes some of the optional inputs.

```python
cdmax = 1.3

# compute a 3D corrected and extended airfoil
af_extrap1 = af.extrapolate(cdmax)

# a second example using the optional inputs
AR = 17  # blade aspect ratio. If provided, cdmax is estimated using the aspect ratio.
cdmin = 0.001  # minimum drag coefficient. Viterna's method can occasionally produce
# negative drag coefficients. A minimum is used to prevent unphysical data.
# The passed in value is used to override the default.
```
Some codes need to use the same set of angles of attack data for every Reynolds number defined in the airfoil. The following example performs the same method as in the command-line version followed by an alternate approach where the user can specify the set of angles of attack to use.

```python
af_common1 = af.interpToCommonAlpha()

# default approach uses a union of all defined angles of attack
# alternatively, specify the exact angles to use
alpha = np.arange(-180, 180)
af_common2 = af.interpToCommonAlpha(alpha)
```

For direct access to the underlying data in a grid format (if not already a grid, it is interpolated to a grid first), use the `createDataGrid` method as follows:

```python
alpha, Re, cl, cd, cm = af.createDataGrid()

# cl[i, j] is the lift coefficient for alpha[i] and Re[j]
# write a new AeroDyn file
af.writeToAerodynFile("output.dat")
```

Finally, writing AeroDyn formatted files is straightforward.

```python
af.writeToAerodynFile("output.dat")
```

### 5.7.12 12. pyFrame3DD Example

This document walks through the pyFrame3dd usage that matches the Pyramid Frame (B) example provided by Frame3DD.

**pyFrame3dd Analysis Steps**

- **12. pyFrame3DD Example**
  - Geometry
  - Loading
  - Modal Analysis
  - Added Mass
  - Simulation
  - Output
Geometry

Setting the geometry of the structure involves specifying node locations, element cross-section properties, and boundary conditions. The inputs can be of any units that the user desires, as long as they are self-consistent across all of the entries.

The node locations are specified by:

```python
inode = np.array([1, 2, 3, 4, 5])
x = np.array([0.0, -1200, 1200, 1200, -1200])
y = np.array([0.0, -900, -900, 900, 900])
z = np.array([1000.0, 0.0, 0.0, 0.0, 0.0])
r = np.array([0.0, 0.0, 0.0, 0.0, 0.0])

nodes = pyframe3dd.NodeData(inode, x, y, z, r)
```

The boundary conditions are specified by listing which node degrees of freedom (DOF) have reactions, or are fixed:

```python
rnode = np.array([2, 3, 4, 5])
Rx = np.ones(4)
Ry = np.ones(4)
Rz = np.ones(4)
Rxx = np.ones(4)
Ryy = np.ones(4)
Rzz = np.ones(4)

reactions = pyframe3dd.ReactionData(rnode, Rx, Ry, Rz, Rxx, Ryy, Rzz, rigid=1)
```

The element cross sections include area, “shear area”, moments of inertia, Young’s and shear moduli, and material density:

```python
ielement = np.array([1, 2, 3, 4])
N1 = np.array([2, 1, 1, 5])
N2 = np.array([1, 3, 4, 1])
Ax = 36.0 * np.ones(4)
Asy = 20.0 * np.ones(4)
Asz = 20.0 * np.ones(4)
Jx = 1000.0 * np.ones(4)
Iy = 492.0 * np.ones(4)
Iz = 492.0 * np.ones(4)
E = 200000.0 * np.ones(4)
G = 79300.0 * np.ones(4)
roll = np.zeros(4)
density = 7.85e-9 * np.ones(4)

elements = pyframe3dd.ElementData(ielement, N1, N2, Ax, Asy, Asz, Jx, Iy, Iz, E, G, roll, density)
```

The final geometry element specifies some modeling parameters for Frame3DD:

```python
shear = True  # 1: include shear deformation
geom = True  # 1: include geometric stiffness
dx = 20.0    # x-axis increment for internal forces
other = pyframe3dd.Options(shear, geom, dx)
```
Now we can create a full pyFrame3DD Frame object that stores the geometry:

```python
frame = pyframe3dd.Frame(nodes, reactions, elements, other)
```

### Loading

Frame3DD can assess many different load cases on the same structure simultaneously. It can also handle many different types of loading, only a few of which are featured in this example.

The first load case uses the standard static gravity load and a point force acting in all directions (with no moment). Note that pyFrame3DD initializes all load objects through the gravity static load call. Then the load case must be added to the Frame object:

```python
# gravity in the X, Y, Z, directions (global)
gx = 0.0
gy = 0.0
gz = -9806.33
load = pyframe3dd.StaticLoadCase(gx, gy, gz)

# point load
nF = np.array([1])
Fx = np.array([100.0])
Fy = np.array([-200.0])
Fz = np.array([-100.0])
Mxx = np.array([0.0])
Myy = np.array([0.0])
Mzz = np.array([0.0])
load.changePointLoads(nF, Fx, Fy, Fz, Mxx, Myy, Mzz)

frame.addLoadCase(load)
```

The second load case starts with the same gravity static load, but then adds in uniform and trapezoidally distributed loads along specific elements. Note that element loads are specified in the element coordinate system, not the global coordinate system of the nodes. Finally, a temperature load is also added:

```python
# gravity in the X, Y, Z, directions (global)
gx = 0.0
gy = 0.0
gz = -9806.33
load = pyframe3dd.StaticLoadCase(gx, gy, gz)

# uniform loads
EL = np.array([2, 1])
Ux = np.array([0.0, 0.0])
Uy = np.array([0.1, 0.0])
Uz = np.array([0.0, 0.1])
load.changeUniformLoads(EL, Ux, Uy, Uz)

# trapezoidally distributed loads
EL = np.array([3, 4])
xx1 = np.array([20.0, 0.0])
xx2 = np.array([80.0, 0.0])
wx1 = np.array([0.01, 0.0])
```

(continues on next page)
wx2 = np.array([0.05, 0.0])
xy1 = np.array([0.0, 68.0])
xy2 = np.array([0.0, 330.0])
wy1 = np.array([0.0, 0.05])
wy2 = np.array([0.0, 0.0])
xz1 = np.array([80.0, 80.0])
xz2 = np.array([830.0, 830.0])
wz1 = np.array([-0.05, -0.05])
wz2 = np.array([0.07, 0.07])
load.changeTrapezoidalLoads(EL, xx1, xx2, wx1, wx2, xy1, xy2, wy1, wy2, xz1, xz2, wz1, wz2)

EL = np.array([1])
a = np.array([12e-6])
hy = np.array([10.0])
hz = np.array([10.0])
Typ = np.array([20.0])
Tym = np.array([10.0])
Tzp = np.array([10.0])
Tzm = np.array([-10.0])
load.changeTemperatureLoads(EL, a, hy, hz, Typ, Tym, Tzp, Tzm)
frame.addLoadCase(load)

The final load case features internal loads, although this is not a feature that is used within WISDEM:

gx = 0.0
gy = 0.0
gz = -9806.33
load = pyframe3dd.StaticLoadCase(gx, gy, gz)

# concentrated interior point loads
EL = np.array([1, 2])
Px = np.array([0.0, 0.0])
Py = np.array([100.0, -200.0])
Pz = np.array([-900.0, 200.0])
x = np.array([600.0, 800.0])
load.changeElementLoads(EL, Px, Py, Pz, x)
frame.addLoadCase(load)

Modal Analysis

Frame3DD includes extensive modal analysis options and outputs. These are set in pyFrame3DD via the following call:

nM = 6  # number of desired dynamic modes of vibration
Mmethod = 1  # 1: subspace Jacobi  2: Stodola
lump = 0  # 0: consistent mass ... 1: lumped mass matrix
tol = 1e-9  # mode shape tolerance
shift = 0.0  # shift value ... for unrestrained structures
frame.enableDynamics(nM, Mmethod, lump, tol, shift)

### Added Mass

An extra feature of pyFrame3DD is the ability to include extra mass in both the load and modal calculations. WISDEM takes full advantage of this feature, especially in the support structure analyses. However, this means care must be taken to only add nodal mass once all of the load cases have already been added to the Frame object:

```python
N = np.array([1])
EMs = np.array([0.1])
EMx = np.array([0.0])
EMY = np.array([0.0])
EMz = np.array([0.0])

# frame.changeExtraInertia(N, EMs, EMx, EMY, EMz)
frame.changeExtraNodeMass(N, EMs, EMx, EMY, EMz, [0.0], [0.0], [0.0], [0.0], [0.0], [0.0], False)
```

### Simulation

Running the Frame object with its load cases is a simple call:

```python
displacements, forces, reactions, internalForces, mass, modal = frame.run()
```

### Output

Analysis output is available using the same keywords as the Frame3DD manual, and is all available as Numpy arrays. Outputs of interest and interrogation of the data is user and application specific. In this example, the full data structure is simply printed to the screen.

```python
nC = len(frame.loadCases)  # number of load cases
nN = len(nodes.node)  # number of nodes
nE = len(elements.element)  # number of elements

# mass data
print("total_mass =", mass.total_mass)
print("struct_mass =", mass.struct_mass)
print("node =", mass.node)
print("xmass =", mass.xmass)
print("ymass =", mass.ymass)
print("zinrta =", mass.zinrta)
print("zinrta =", mass.yinrta)
print("zinrta =", mass.zinrta)
print()
print()

# node displacements
for iCase in range(nC):
```

---

5.7. Examples 207
print("case_idx:", iCase)
print("node:", displacements.node[iCase, :])
print("dx:", displacements.dx[iCase, :])
print("dy:", displacements.dy[iCase, :])
print("dz:", displacements.dz[iCase, :])
print("dxrot:", displacements.dxrot[iCase, :])
print("dyrot:", displacements.dyrot[iCase, :])
print("dzrot:", displacements.dzrot[iCase, :])
print()
print("element =", forces.element[iCase, :])
print("node =", forces.node[iCase, :])
print("Nx =", forces.Nx[iCase, :])
print("Vy =", forces.Vy[iCase, :])
print("Vz =", forces.Vz[iCase, :])
print("Txx =", forces.Txx[iCase, :])
print("Myy =", forces.Myy[iCase, :])
print("Mzz =", forces.Mzz[iCase, :])
print()
print("nodesR =", reactions.node[iCase, :])
print("RFx =", reactions.Fx[iCase, :])
print("RFy =", reactions.Fy[iCase, :])
print("RFz =", reactions.Fz[iCase, :])
print("RMxx =", reactions.Mxx[iCase, :])
print("RMyy =", reactions.Myy[iCase, :])
print("RMzz =", reactions.Mzz[iCase, :])
print()
print()

# internal forces
iE = 3  # note just showing for one element
for iCase in range(nC):

    print("case_idx:", iCase)
    print("element_idx:", iE)

    print("x =", internalForces[iE].x[iCase, :])
    print("Nx =", internalForces[iE].Nx[iCase, :])
    print("Vy =", internalForces[iE].Vy[iCase, :])
    print("Vz =", internalForces[iE].Vz[iCase, :])
    print("Tx =", internalForces[iE].Tx[iCase, :])
    print("My =", internalForces[iE].My[iCase, :])
    print("Mz =", internalForces[iE].Mz[iCase, :])
    print("Dx =", internalForces[iE].Dx[iCase, :])
    print("Dy =", internalForces[iE].Dy[iCase, :])
    print("Dz =", internalForces[iE].Dz[iCase, :])
    print("Rx =", internalForces[iE].Rx[iCase, :])
    print()
Lastly, we can also output a Frame3DD file using a built-in method.

```python
frame.write("pyramid_frame_B.3dd")
```

## 5.8 Module documentation

### 5.8.1 AirfoilPrep.py

**Introduction**

AirfoilPrep.py (pronounced Airfoil Preppy) provides functionality to preprocess aerodynamic airfoil data. Essentially, the module is an object oriented version of the AirfoilPrep spreadsheet with additional functionality and is written in the Python language. The intent is to provide the functionality of the AirfoilPrep spreadsheet, but in an easy-to-use format both for stand-alone preprocessing through scripting and for direct implementation within other codes such as blade element momentum methods.

AirfoilPrep.py allows the user to read in two-dimensional (2-D) aerodynamic airfoil data (i.e., from wind tunnel data or numerical simulation), apply three-dimensional (3-D) rotation corrections for wind turbine applications, and extend the data to very large angles of attack. Airfoil data can also be blended together to define intermediate sections between linearly lofted sections. Capabilities unique to the Python version include the ability to read and write to AeroDyn format files directly. The only feature that is contained in the spreadsheet version but is currently missing in AirfoilPrep.py, is handling of pitching moment coefficients.

This document discusses installation, usage, and documentation of the module. Because the theory is simplistic, only a brief overview is provided in the documentation section with corresponding references that contain further detail.
Module Documentation

Two classes are provided in the module: *Polar* and *Airfoil*. Generally, the Polar class is not needed for direct usage except for its constructor. All objects in this module are immutable. In other words, calling `Airfoil.correct3D()` creates a new modified airfoil object rather than editing the existing object.

### Polar Class

A Polar object is meant to represent the variation in lift, drag, and pitching moment coefficient with angle of attack at a fixed Reynolds number. Generally, the methods of this class do not need to be used directly (other than the constructor), but rather are used by the *Airfoil* class.

### Airfoil Class

An Airfoil object encapsulates the aerodynamic forces/moments of an airfoil as a function of angle of attack and Reynolds number. For wind turbine analysis, this class provides capabilities to apply 3-D rotational corrections to 2-D data using the Du-Selig method [DS98] for lift, and the Eggers method [EJCD03] for drag. Airfoil data can also be extrapolated to +/-180 degrees, using Viterna’s method [VJ82]. This class also adds methods to read and write AeroDyn airfoil files directly.

### 5.8.2 CCBlade

**Introduction**

CCBlade predicts aerodynamic loading of wind turbine blades using blade element momentum (BEM) theory. CC stands for continuity and convergence. CCBlade was developed primarily for use in gradient-based optimization applications where $C^1$ continuity and robust convergence are essential.

Typical BEM implementations use iterative solution methods to converge the induction factors (e.g., fixed-point iteration or Newton’s method). Some more complex implementations use numerical optimization to minimize the error in the induction factors. These methods can be fairly robust, but all have at least some regions where the algorithm fails to converge. A new methodology was developed that is provably convergent in every instance (see *Theory*). This robustness is particularly important for gradient-based optimization. To ensure $C^1$ continuity, lift and drag coefficients are computed using a bivariate cubic spline across angle of attack and Reynolds number. Additionally, analytic gradients for distributed loads, thrust, torque, and power are (optionally) provided.

CCBlade is written in Python, but iteration-heavy sections are written in Fortran in order to improve performance. The Fortran code is called from Python as an extension module using f2py. The module AirfoilPrep.py is also included with the source. Although not directly used by CCBlade, the airfoil preprocessing capabilities are often useful for this application.
Module Documentation

The main methodology is contained in CCBlade. Airfoil data is provided by any object that implements Airfoil Interface. The helper class CCAirfoil is provided as a useful default implementation for AirfoilInterface. If CCAirfoil is not used, the user must provide an implementation that produces $C^1$ continuous output (or else accept non-smooth aerodynamic calculations from CCBlade). Some of the underlying implementation for CCBlade is written in Fortran for computational efficiency.

Airfoil Interface

The airfoil objects used in CCBlade need only implement the following evaluate() method. Although using CCAirfoil for the implementation is recommended, any custom class can be used.

CCAirfoil Class

CCAirfoil is a helper class used to evaluate airfoil data with a continuously differentiable bivariate spline across the angle of attack and Reynolds number. The degree of the spline polynomials across the Reynolds number is summarized in the following table (the same applies to the angle of attack although generally, the number of points for the angle of attack is much larger).

<table>
<thead>
<tr>
<th>len(Re)</th>
<th>degree of spline</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>constant</td>
</tr>
<tr>
<td>2</td>
<td>linear</td>
</tr>
<tr>
<td>3</td>
<td>quadratic</td>
</tr>
<tr>
<td>4+</td>
<td>cubic</td>
</tr>
</tbody>
</table>

Class Summary:

CCBlade Class

This class provides aerodynamic analysis of wind turbine rotor blades using BEM theory. It can compute distributed aerodynamic loads and integrated quantities such as power, thrust, and torque. An emphasis is placed on convergence robustness and differentiable output so that it can be used with gradient-based optimization.

Class Summary:

Theory

Note: Only an overview of the theory is included here; details can be found in Ning [Nin13].

The rotor aerodynamic analysis is based on blade element momentum (BEM) theory. Using BEM theory in a gradient-based rotor optimization problem can be challenging because of occasional convergence difficulties of the BEM equations. The standard approach to solving the BEM equations is to arrange the equations as functions of the axial and tangential induction factors and solve the fixed-point problem:

$$(a, a') = f_f(a, a')$$
using either fixed-point iteration, Newton’s method, or a related fixed-point algorithm. An alternative approach is to use nonlinear optimization to minimize the sum of the squares of the residuals of the induction factors (or normal and tangential loads). Although these approaches are generally successful, they suffer from instabilities and failure to converge in some regions of the design space. Thus, they require increased complexity and/or heuristics (but may still not converge).

The new BEM methodology transforms the two-variable, fixed-point problem into an equivalent one-dimensional root-finding problem. This is enormously beneficial as methods exist for one-dimensional root-finding problems that are guaranteed to converge as long as an appropriate bracket can be found. The key insight to this reduction is to use the local inflow angle $\phi$ and the magnitude of the inflow velocity $W$ as the two unknowns in specifying the inflow conditions, rather than the traditional axial and tangential induction factors (see Figure 5.21).

This approach allows the BEM equations to be reduced to a one-dimensional residual function as a function of $\phi$:

$$R(\phi) = \sin \frac{\phi}{1 - a(\phi)} - \frac{\cos \phi}{\lambda_r(1 + a'(\phi))} = 0$$

Figure 5.22 shows the typical behavior of $R(\phi)$ over the range $\phi \in (0, \pi/2]$. Almost all solutions for wind turbines fall within this range (for the provable convergence properties to be true, solutions outside of this range must also be considered). The referenced paper [Nin13] demonstrates through mathematical proof that the methodology will always find a bracket to a zero of $R(\phi)$ without any singularities in the interior. This proof, along with existing proofs for root-finding methods like Brent’s method [Bre71], implies that a solution is guaranteed. Furthermore, not only is the solution guaranteed, but it can be found efficiently and in a continuous manner. This behavior allows the use of gradient-based algorithms to solve rotor optimization problems much more effectively than with traditional BEM solution approaches.

Any corrections to the BEM method can be used with this methodology (e.g., finite number of blades and skewed wake) as long as the axial induction factor can be expressed as a function of $\phi$ (either explicitly or through a numerical solution). CCBlade chooses to include both hub and tip losses using Prandtl’s method [Gla35] and a high-induction factor correction by Buhl [Buh05]. Drag is included in the computation of the induction factors. However, all of these options can be toggled on or off.

Gradients are computed using a direct/adjoint (identical for one state variable) method. Let us define a functional (e.g., distributed load at one section), as:

$$f = N'(x_i, \phi)$$

Using the chain rule the total derivatives are given as

$$\frac{df}{dx_i} = \frac{\partial f}{\partial x_i} - \frac{\partial f}{\partial \phi} \frac{\partial R}{\partial x_i} / \frac{\partial R}{\partial \phi}$$
Fig. 5.22: Residual function of BEM equations using new methodology. Solution point is where $R(\phi) = 0$.

5.8.3 CommonSE

Coordinate System

This module defines coordinate systems for horizontal axis wind turbines and provides convenience methods for transforming vectors between the various coordinate systems. The supplied transformation methods are for rotation only and do not account for any offsets that may be necessary depending on the vector quantity (e.g., transfer of forces between coordinate system does not depend on the location where the force is defined, but position, velocity, moments, etc. do). In other words the vectors are treated as directions only and are independent of the defined position. How the vector should transform based on position is not generalizable and depends on the quantity of interest.

All coordinate systems obey the right-hand rule, $x \times y = z$, and all angles must be input in degrees. The turbine can be either an upwind or downwind configuration, but in either case it is assumed that the blades rotate in the clockwise direction when looking downwind (more specifically the rotor is assumed to rotate about the $+x_k$ axis in Figure 5.25).

The vectors allow for elementary operations (+, -, *, /, +=, -=, *=, /=) between other vectors of the same type, or with scalars (e.g., force_total = force1 + force2).

```python
class wisdem.commonse.csystem.DirectionVector(x, y, z, dx=None, dy=None, dz=None)
```

Handles rotation of direction vectors to appropriate coordinate systems. All angles must be in degrees.
Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>airfoilToBlade(theta)</code></td>
<td>Rotates from airfoil-aligned to blade-aligned</td>
</tr>
<tr>
<td><code>airfoilToProfile()</code></td>
<td>Rotates from airfoil-aligned to profile</td>
</tr>
<tr>
<td><code>azimuthToBlade(Phi)</code></td>
<td>Rotates from azimuth-aligned to blade-aligned</td>
</tr>
<tr>
<td><code>azimuthToHub(Lambda)</code></td>
<td>Rotates from azimuth-aligned to hub-aligned</td>
</tr>
<tr>
<td><code>bladeToAirfoil(theta)</code></td>
<td>Rotates from blade-aligned to airfoil-aligned</td>
</tr>
<tr>
<td><code>bladeToAzimuth(Phi)</code></td>
<td>Rotates from blade-aligned to azimuth-aligned</td>
</tr>
<tr>
<td><code>cross(other)</code></td>
<td>Cross product between two DirectionVectors</td>
</tr>
<tr>
<td><code>cross_deriv(other[, namea, nameb])</code></td>
<td>Defined only for floats for now</td>
</tr>
<tr>
<td><code>fromArray(array)</code></td>
<td>Initialize with NumPy array</td>
</tr>
<tr>
<td><code>hubToAzimuth(Lambda)</code></td>
<td>Rotates from hub-aligned to azimuth-aligned</td>
</tr>
<tr>
<td><code>hubToYaw(Theta[, derivatives])</code></td>
<td>Rotates from hub-aligned to yaw-aligned</td>
</tr>
<tr>
<td><code>inertialToWind(beta)</code></td>
<td>Rotates from inertial to wind-aligned</td>
</tr>
<tr>
<td><code>profileToAirfoil()</code></td>
<td>Rotates from profile to airfoil-aligned</td>
</tr>
<tr>
<td><code>toArray()</code></td>
<td>Convert DirectionVector to NumPy array</td>
</tr>
<tr>
<td><code>windToInertial(beta)</code></td>
<td>Rotates from wind-aligned to inertial</td>
</tr>
<tr>
<td><code>windToYaw(Psi)</code></td>
<td>Rotates from wind-aligned to yaw-aligned</td>
</tr>
<tr>
<td><code>yawToHub(Theta)</code></td>
<td>Rotates from yaw-aligned to hub-aligned</td>
</tr>
<tr>
<td><code>yawToWind(Psi)</code></td>
<td>Rotates from yaw-aligned to wind-aligned</td>
</tr>
</tbody>
</table>

**Inertial and Wind-aligned**

Figure 5.23 defines the transformation between the inertial and wind-aligned coordinate systems. The two coordinate systems share a common origin, and a common z-direction. The wind angle $\beta$ is positive for rotation about the $+z$ axis. The direction of wave loads are defined similarly to the wind loads, but there is no wave-aligned coordinate system.
**Inertial coordinate system**

- **origin**: center of the tower base (ground-level or sea-bed level)
- **x-axis**: any direction as long as used consistently, but convenient to be in primary wind direction
- **y-axis**: follows from the right-hand rule
- **z-axis**: up the tower (opposite to gravity vector)

**Wind-aligned coordinate system**

- **origin**: center of the tower base (ground-level or sea-bed level)
- **x-axis**: in direction of the wind
- **y-axis**: follows from the right-hand rule
- **z-axis**: up the tower (opposite to gravity vector), coincident with inertial z-axis

**TABLE CAPTION**: Inertial-Wind conversion methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inertialToWind(beta)</td>
<td>Rotates from inertial to wind-aligned</td>
</tr>
<tr>
<td>windToInertial(beta)</td>
<td>Rotates from wind-aligned to inertial</td>
</tr>
</tbody>
</table>

**Wind-aligned and Yaw-aligned**

![Diagram of Wind-aligned and Yaw-aligned axes](image)

**Fig. 5.24**: Wind-aligned and yaw-aligned axes. Ψ is the rotor yaw angle.
Figure 5.24 defines the transformation between the wind-aligned and yaw-aligned coordinate systems. The two coordinate systems are offset by the height $h_t$ along the common z-axis. The yaw angle $\Psi$ is positive when rotating about the +z axis, and should be between -180 and +180 degrees.

**Yaw-aligned coordinate system**

- **origin**: Tower top (center of the yaw bearing system)
- **x-axis**: along projection of rotor shaft in horizontal plane (aligned with rotor shaft for zero tilt angle). The positive direction is defined such that the x-axis points downwind at its design operating orientation (i.e., at zero yaw $x_y$ is the same direction as $x_w$). Thus, for a downwind machine the $x_y$ axis would still be downwind at zero yaw, but in terms of nacelle orientation it would point from the back of the nacelle toward the hub.
- **y-axis**: follows from the right-hand rule
- **z-axis**: points up the tower (opposite to gravity vector), coincident with wind-aligned z-axis

**TABLE CAPTION:** Wind-Yaw conversion methods

<table>
<thead>
<tr>
<th>windToYaw($\Psi$)</th>
<th>Rotates from wind-aligned to yaw-aligned</th>
</tr>
</thead>
<tbody>
<tr>
<td>yawToWind($\Psi$)</td>
<td>Rotates from yaw-aligned to wind-aligned</td>
</tr>
</tbody>
</table>

**Yaw-aligned and Hub-aligned**

![Diagram of yaw-aligned and hub-aligned axes](image)

Fig. 5.25: Yaw-aligned and hub-aligned axes. $\Theta$ is the rotor tilt angle.

Figure 5.25 defines the transformation between the yaw-aligned and hub-aligned coordinate systems. The two coordinate systems share a common y axis. The tilt angle $\Theta$ is positive when rotating about the +y axis, which tilts the rotor up for an upwind machine (tilts the rotor down for a downwind machine).
Hub-aligned coordinate system

origin: center of the rotor.

x-axis: along the rotor shaft toward the nominal downwind direction (aligned with x, for zero tilt)

y-axis: coincident with yaw-aligned y-axis

z-axis: right-hand rule (vertical if zero tilt)

TABLE CAPTION: Yaw-Hub conversion methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>yawToHub(Theta)</td>
<td>Rotates from yaw-aligned to hub-aligned</td>
</tr>
<tr>
<td>hubToYaw(Theta[, derivatives])</td>
<td>Rotates from hub-aligned to yaw-aligned</td>
</tr>
</tbody>
</table>

Hub-aligned and Azimuth-aligned

Fig. 5.26: Hub-aligned and azimuth-aligned axes. Λ is the (local) blade azimuth angle.

Figure 5.26 defines the transformation between the hub-aligned and azimuth-aligned coordinate systems. The two coordinate systems share a common x-axis. The azimuth angle Λ is positive when rotating about the +x axis. The blade can employ a variable azimuth angle along the blade axis, to allow for swept blades.

Azimuth-aligned coordinate system

A rotating coordinate system—about the xh axis. The coordinate-system is locally-defined for the case of a variable-swept blade.

origin: blade pitch axis, local to the blade section

x-axis: aligned with the hub-aligned x-axis

y-axis: right-hand rule

z-axis: along projection of blade from root to tip in the yh - zh plane (aligned with blade only for zero precone)

TABLE CAPTION: Hub-Azimuth conversion methods
**Azimuth-aligned and Blade-aligned**

Figure 5.27 defines the transformation between the azimuth-aligned and blade-aligned coordinate systems. The $y_b$ and $y_z$ axes are in the same direction. The two coordinate systems rotate together such that the $x_b - z_b$ plane is always coplanar with the $x_z - z_z$ plane. The precone angle $\Phi$ is positive when rotating about the $-y_z$ axis, and causes the blades to tilt away from the nacelle/tower for a downwind machine (tilts toward tower for upwind machine). The blade can employ a variable precone angle along the blade axis. The blade-aligned coordinate system is considered local to a section of the blade.

**Blade-aligned coordinate system**

A rotating coordinate system that rotates with the azimuth-aligned coordinate system. The coordinate-
system is locally-defined along the blade radius. The direction of blade rotation is in the negative y-axis. A force in the x-axis would be a flapwise shear, and a force in the y-axis would be a lead-lag shear.

**origin:** blade pitch axis, local to the blade section

**x-axis:** follows from the right-hand rule (in nominal downwind direction)

**y-axis:** opposite to rotation direction, positive from section leading edge to trailing edge (for no twist)

**z-axis:** along the blade pitch axis in increasing radius

**TABLE CAPTION:** Azimuth-Blade conversion methods

<table>
<thead>
<tr>
<th>azimuthToBlade((\Phi))</th>
<th>Rotates from azimuth-aligned to blade-aligned</th>
</tr>
</thead>
<tbody>
<tr>
<td>bladeToAzimuth((\Phi))</td>
<td>Rotates from blade-aligned to azimuth-aligned</td>
</tr>
</tbody>
</table>

**Blade-aligned and Airfoil-aligned**

![Blade-aligned and Airfoil-aligned coordinate systems](image)

Fig. 5.28: Blade-aligned and airfoil-aligned coordinate systems. \(\theta\) is the airfoil twist + pitch angle. For convenience the local wind vector and angle of attack is shown.

**Figure 5.28** defines the transformation between the blade-aligned and airfoil-aligned coordinate systems. The \(z_b\) and \(z_a\) axes are in the same direction. The twist angle \(\theta\) is positive when rotating about the \(-z_a\) axis, and causes the angle of attack to decrease.

**Airfoil-aligned coordinate system**

A force in the x-axis would be a flatwise shear, and a force in the y-axis would be an edgewise shear.

**origin:** blade pitch axis, local to the blade section

**x-axis:** follows from the right-hand rule

**y-axis:** along chord line in direction of trailing edge

**z-axis:** along the blade pitch axis in increasing radius, same as \(z_b\) (into the page in above figure)

**TABLE CAPTION:** Blade-Airfoil conversion methods
Airfoil-aligned and Profile

Figure 5.29 defines the transformation between the airfoil-aligned and profile coordinate systems. The profile coordinate system is generally used only to define airfoil profile data.

Profile coordinate system

- **origin**: airfoil noise
- **x-axis**: positive from nose to trailing edge along chord line
- **y-axis**: orthogonal to x-axis, positive from lower to upper surface
- **z-axis**: n/a (profile is a 2-dimensional coordinate system)

**TABLE CAPTION**: Airfoil-Profile conversion methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>airfoilToProfile()</td>
<td>Rotates from airfoil-aligned to profile</td>
</tr>
<tr>
<td>profileToAirfoil()</td>
<td>Rotates from profile to airfoil-aligned</td>
</tr>
</tbody>
</table>
Environment

Environment contains shared wind, wave, and soil models.

Wind

This module defines a wind speed profile at locations $z$, all wind speeds below $z_0$ are 0. The parameters $U_{ref}$ and $z_{ref}$ allow for scaling of a profile shape. Specific implementations of this base component include PowerWind and LogWind. PowerWind assumes a power-law distribution of wind speeds of the form

$$ U(z) = U_{ref} \left( \frac{z - z_0}{z_{ref} - z_0} \right)^\alpha $$

The logarithmic profile is of the form

$$ U = U_{ref} \begin{bmatrix} \log \left( \frac{z - z_0}{\text{roughness}} \right) \\ \log \left( \frac{z_{ref} - z_0}{\text{roughness}} \right) \end{bmatrix} $$

```
class wisdem.commonse.environment.WindBase
class wisdem.commonse.environment.PowerWind
class wisdem.commonse.environment.LogWind
```

Wave

Hydrodynamic speed distributions are estimated using linear wave theory (LinearWaves). According to linear wave theory, the maximum horizontal velocity of a wave is given as

$$ U_{current} = \omega \frac{h}{2} \frac{\cosh(k(z + D))}{\sinh(kD)} \cos(\omega t) $$

and the corresponding maximum acceleration is

$$ A_{current} = \omega U_{current} $$

```
class wisdem.commonse.environment.WaveBase
class wisdem.commonse.environment.LinearWaves
```

Soil

The soil is assumed to not contribute any inertial or applied forces and only affects the stiffness of the foundation. The user may specify directions which are considered rigid. For the other directions, effective spring constants are estimated based on the soil properties (TowerSoil). A simple textbook model is used in this implementation\(^1\). The model allows for computation of an effective spring constant for all six degrees of freedom, each computed as a function of the shear modulus and Poisson’s ratio of the soil.

For example:

\[ k_z = \frac{4Gr}{1 - \nu} \left( 1 + 0.6(1 - \nu) \frac{h}{r} \right) \]

where \( h \) is the depth of the foundation below the soil.

```python
class wisdem.commonse.environment.SoilBase
class wisdem.commonse.environment.TowerSoil
```

**Utilities**

This module contains a collection of utilities for use across the WISDEM model set.

**Differentiability**

Differentiable versions of several functions are provided (along with their derivatives). These are summarized in the below table.

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>linspace_with_deriv</code></td>
<td>Creates linearly spaced arrays, and derivatives for changing end points.</td>
</tr>
<tr>
<td><code>interp_with_deriv</code></td>
<td>Linear interpolation and its derivative.</td>
</tr>
<tr>
<td><code>arc_length_deriv</code></td>
<td>Return the Jacobian for function <code>arc_length()</code>.</td>
</tr>
<tr>
<td><code>trapz_deriv</code></td>
<td>Trapezoidal integration and derivatives with respect to integrand or variable.</td>
</tr>
<tr>
<td><code>smooth_max</code></td>
<td>Array max, uses cubic spline to smoothly transition.</td>
</tr>
<tr>
<td><code>smooth_min</code></td>
<td>Array min, uses cubic spline to smoothly transition.</td>
</tr>
<tr>
<td><code>smooth_abs</code></td>
<td>Smoothed version of absolute value function, with quadratic instead of sharp bottom.</td>
</tr>
<tr>
<td><code>CubicSplineSegment</code></td>
<td>Cubic splines and their derivatives with respect to the variables and the parameters</td>
</tr>
</tbody>
</table>

```python
wisdem.commonse.utilities.linspace_with_deriv

wisdem.commonse.utilities.linspace_with_deriv(start, stop, num)
    Creates linearly spaced arrays, and derivatives for changing end points
```
WISDEM, Release 2.0

**wisdom.commonse.utilitiesinterp_with_deriv**

`wisdom.commonse.utilities.interp_with_deriv(x, xp, yp)`

linear interpolation and its derivative. To be precise, linear interpolation is not differentiable right at the control points, but in general it works well enough

**wisdom.commonse.utilitiesarc_length_deriv**

`wisdom.commonse.utilities.arc_length_deriv(points)`

Return the Jacobian for function arc_length(). See its docstring for more details.

- **Parameters**
  - `points` *(numpy array[n_points, n_dim]*) – Array of coordinate points that we compute the arc distances for.

- **Returns**
  - `d_arc_distances_d_points` – Array, starting at 0, with the cumulative distance from the first point in the points array along the arc.

- **Return type**
  - `numpy array[n_points, n_points * n_dim]`

**wisdom.commonse.utilitiestrapz_deriv**

`wisdom.commonse.utilities.trapz_deriv(y, x)`

trapezoidal integration and derivatives with respect to integrand or variable.

**wisdom.commonse.utilitiessmooth_max**

`wisdom.commonse.utilities.smooth_max(yd, ymax, pct_offset=0.01, dyd=None)`

array max, uses cubic spline to smoothly transition. derivatives with respect to array and max value. width of transition can be controlled, and chain rules for differentiation

**wisdom.commonse.utilitiessmooth_min**

`wisdom.commonse.utilities.smooth_min(yd, ymin, pct_offset=0.01, dyd=None)`

array min, uses cubic spline to smoothly transition. derivatives with respect to array and min value. width of transition can be controlled, and chain rules for differentiation

**wisdom.commonse.utilitiessmooth_abs**

`wisdom.commonse.utilities.smooth_abs(x, dx=0.01)`

smoothed version of absolute value function, with quadratic instead of sharp bottom. Derivative w.r.t. variable of interest. Width of quadratic can be controlled
5.8.4 DrivetrainSE

Drivetrain Model Introduction

The Drivetrain Systems Engineering (DrivetrainSE) module is a set of models for sizing wind turbine drivetrain components as part of the larger WISDEM design and analysis tool. Wind turbine drivetrains physically connect the rotor to the tower and serve as a load-path from one to the other. The drivetrain is also responsible for converting the aerodynamic torque of the rotor into electrical power that can be fed to the grid. Therefore, the drivetrain model interacts with the rotor and tower designs and it is important in looking at the overall design of a wind turbine to consider the coupling that exists between these three primary subsystems. DrivetrainSE provides the capability to take in the aerodynamic loads and rotor properties and to estimate the mass properties and dimensions for all major components; the overall nacelle properties can then be used in subsequent tower design and analysis or as part of a system-level optimization of the wind turbine. In addition, the resulting mass and dimension estimates can then be used to feed into a turbine capital cost model as well as a balance of station cost model that considers cost of assembly and installation of a wind turbine so that a full wind plant system level cost analysis could be performed.

DrivetrainSE uses a slightly different approach to the prior instances of DriveSE and HubSE, although some of the component sizing remains the same. Instead of analytical derivations of the forces and moments on the various elements and closed form expressions for sizing the components, we instead rely on Frame3DD to conduct the analysis and enable the use of optimization with stress constraints to ensure a valid design. This proves to be an easier long-term approach to maintain correct code.

DrivetrainSE features the following capabilities:

- Upwind or downwind rotor configuration
- Direct-drive and geared
- Electromagnetic design of multiple generator technologies, including synchronous and induction generators.
- Up-tower or down-tower electronics
- Sizing of drivetrain components via structural analysis

DrivetrainSE includes sizing for the following components:
- Hub
- Spinner
- Pitch system
- Low speed shaft
- Main bearing(s)
- Gearbox (geared systems only)
- High speed shaft (geared systems only)
- Brake
- Generator
- Generator cooling
- Power electronics
- Bedplate
- Nacelle platform
- Nacelle cover
- Yaw system

Some of these components are sized with very simple, empirical- or regression-based approximations. Others involve more detailed structural analysis, cast as utilization constraints, that are meant to be included in an optimization.

**Layout and Inputs**

The direct-drive and geared drivetrain layouts are quite different, however both use the same set of user inputs. This was intentional to simplify the user-input burden. The common layout parameters are,

- Overhang ($L_{overhang}$)
- Height from hub to tower top ($H_{hett}$)
- Shaft tilt angle ($\gamma$),
- Generator length ($L_{generator}$)
- Distance between the hub flange and the first main bearing ($L_{h1}$)
- Distance between the main bearings ($L_{12}$)

Detailed diagrams of how these parameters set the sizing and layout of the drivetrain is shown in the two subsections below.
Direct-Drive Layout

The overall layout for the direct-drive configuration is shown in Fig. 5.30 and Fig. 5.31. The hub connects to the low-speed shaft, which is a large diameter, hollow cylinder supported by two sets of main bearings attached to the nose, also called the turret. The nose is affixed to the bedplate, which also has a circular cross-section, but follows an elliptical curve down to the tower attachment. The total length, parallel to the ground, from the tower center line to the rotor apex, is considered the overhang. The total height difference between those same two points in the hub to tower top height. The outer-rotor of the generator also attaches to the low-speed shaft, and the corresponding stator attaches to the nose.

The detailed parameters that specify the drivetrain layout are shown in Fig. 5.32.

In addition to the user-defined dimensions, the other values are derived in the following way,

\[
\begin{align*}
L_{grs} &= 0.5L_{h1} \\
L_{gsn} &= L_{generator} - L_{grs} - L_{12} \\
L_{2n} &= 2L_{gsn} \\
L_{lsn} &= L_{12} + L_{h1} \\
L_{nose} &= L_{12} + L_{2n} \\
L_{drive} &= L_{h1} + L_{12} + L_{2n} \\
L_{bedplate} &= L_{overhang} - L_{drive} \cos \gamma \\
H_{bedplate} &= H_{htt} - L_{drive} \sin \gamma
\end{align*}
\]

Here the length from the hub flange to the generator rotor attachment, \(L_{grs}\), is assumed to be at the halfway point between the flange and the first main bearing, \(L_{h1}\). Similarly, the distance between the second main bearing and the nose/turret interface with the bedplate, \(L_{2n}\), is twice the distance as that from the same interface to the generator stator attachment, \(L_{gsn}\). After adding up the total length of the low speed shaft and nose/turret, the total drivetrain length from bedplate to hub can be determined. Then, the bedplate dimensions are determined in order to meet the target overhang and hub-to-tower top height. To ensure that these layout dimensions are adequately satisfied during a design optimization, a constraint is enforced such that \(L_{bed} \geq 0.5D_{top}\).
The user must also specify diameter and thickness values for the low speed shaft ($D_{ls}$ and $t_{ls}$) and the nose/turret ($D_{nose}$ and $t_{nose}$). These can also be assigned as design variables to satisfy the constraints generated in the structural analysis.

The bedplate diagram is shown in Fig. 5.33, and follows an elliptical path from the tower top to the nose/turret attachment point. The length and height of the bedplate (major and minor half axes of the ellipse) are determined from the input user dimensions. The bedplate diameter also follows an elliptical progression from the tower top diameter at the bedplate base to the nose/turrent diameter at the top. The wall thickness schedule is a user defined input, or can be designated as a design variable in order to meet structural constraints.

The attachment of the generator stator to the nose/turret is shown in Fig. 5.34. For the direct-drive configuration, we assume an outer rotor-inner stator, radial flux topology for a permanent magnet synchronous generator. The outer rotor layout facilitates a simple and rugged structure, easy manufacturing, short end windings, and better heat transfer between windings and teeth than an inner rotor configuration.

**Geared Layout**

The overall layout for the geared configuration is shown in Fig. 5.35. The hub connects to the low-speed shaft, supported by two sets of main bearings. The low speed shaft connects to the gearbox which converts the high-torque, low-rpm input into a low-torque, high-rpm output on the high speed shaft. The high speed shaft feeds the generator, which is assumed to be a doubly-fed induction generator (DFIG). The bedplate is a steel platform that sits atop two parallel I-beams to provide the structural support. The bearings and the generator are assumed to be firmly attached to the bedplate. The gearbox attaches to the nacelle platform atop the bedplate with a trunion.

The detailed parameters that specify the drivetrain geared are shown in Fig. 5.36.
Fig. 5.32: Detailed direct-drive configuration with key user inputs and derived values.

Fig. 5.33: Direct-drive configuration layout diagram
Fig. 5.34: Direct-drive configuration layout diagram

Fig. 5.35: Geared configuration layout diagram
In addition to the user-defined dimensions, the other values are derived in the following way,

\[
\begin{align*}
\delta &= 0.1 \\
L_{lss} &= L_{12} + L_{h1} + \delta \\
L_{drive} &= L_{lss} + L_{gearbox} + L_{hss} + L_{generator} \\
L_{bedplate} &= L_{drive} \cos \gamma \\
H_{bedplate} &= H_{htt} - L_{drive} \sin \gamma
\end{align*}
\]

The dimension, \( \delta \), is the space between the second main bearing and the gearbox attachment where the shrink disk lies. This is assumed to be 0.1 meters. The bedplate height is sized to ensure that the desired height from tower top to hub is obtained. To achieve the desired overhang distance, the tower is centered at the exact overhang distance from the hub and a constraint is enforced such that the drivetrain length is sufficient to extend past the tower, \( L_{drive} \cos \gamma - L_{overhang} \geq 0.5D_{top} \).

**Drivetrain Components**

This section describes the theory behind the sizing and estimation of mass properties for all of the hub and nacelle components. Unless otherwise noted, the moment of inertia expressions are about the component center of mass.

**Hub**

The hub is designed as the combination of a main flange and a spherical shell.
**Spinner**

The spinner is the aerodynamic cone that wraps around the hub.

**Pitch System**

The pitch system mass is an empirical estimation based on the blade mass and blade root bending moment. To find the mass in kilograms, 

$$m_{pitch} = 0.22m_{blade}n_{blade} + 12.6|M_{root}|\rho/\sigma$$

where \(|M_{root}|\) is the magnitude of the blade root bending moment (in Newton-meters), \(\rho\) is the density of the pitch bearing material, and \(\sigma\) is the yield stress.

The pitch system moment of inertia assumes that the mass is distributed as a solid ring at the hub diameter, so the moment of inertia about the hub center for axial and transverse values is,

$$I_0 = mR_{hub}^2/2$$

$$I_1 = I_0/2$$

**Main Bearing(s)**

This is a simple, regression-based sizing tool for the main bearings. The same function is called once for configurations with one main bearing or twice for configurations with two. It handles Compact Aligning Roller Bearings (CARB), Cylindrical Roller Bearings (CRB), Spherical Roller Bearings (SRB), and Tapered Roller Bearings (TRB). The face width, mass, and maximum allowable deflection angle of these bearing types are,

<table>
<thead>
<tr>
<th>Type</th>
<th>Face Width</th>
<th>Mass</th>
<th>Max Defl</th>
</tr>
</thead>
<tbody>
<tr>
<td>CARB</td>
<td>0.2663D_{shaft} + 0.0435</td>
<td>1561.4D_z^{infty}_{shaft}</td>
<td>0.5 deg</td>
</tr>
<tr>
<td>CRB</td>
<td>0.1136D_{shaft}</td>
<td>304.19D_{shaft}^{infty}</td>
<td>2/3 deg</td>
</tr>
<tr>
<td>SRB</td>
<td>0.2762D_{shaft}</td>
<td>876.7D_{shaft}^{infty}</td>
<td>4.47 deg</td>
</tr>
<tr>
<td>TRB</td>
<td>0.1499D_{shaft}</td>
<td>543.01D_{shaft}^{infty}</td>
<td>1/20 deg</td>
</tr>
</tbody>
</table>

In addition to the bearing mass, a bearing housing is assumed which scales the total mass value from the table by 3.963.

The bearing moment of inertia is assumed to be that of a generic torus, with axial and transverse values of,

$$I_0 = m(4R_{shaft}^2 + 3R_{bearing}^2)/4$$

$$I_1 = m(4R_{shaft}^2 + 5R_{bearing}^2)/8$$

Where \(R_{shaft}\) is assumed to be half of the face width from the table.

**Low- and High-Speed Shafts**

The low- and high-speed shaft diameter and thickness are user input values, with their length determined by the layout logic given other user inputs for tower top diameter and overhang distance. Since we allow for tapered shafts, the mass calculation is an integral along the shaft axis, 

$$m_{shaft} = \int_0^L \rho\pi [R_{shaft}(x)^2 - (R_{shaft}(x) - t_{shaft}(x))^2] dx$$

where \(\rho\) is the density of steel.

The shaft moment of inertia is assumed to be that of a generic hollow cylinder, with axial and transverse values of,

$$I_0 = m_{shaft}(R_{shaft}^2 + (R_{shaft} - t_{shaft})^2)/2$$

$$I_1 = m_{shaft}[3(R_{shaft}^2 + (R_{shaft} - t_{shaft})^2) + L_{shaft}^2]/12$$
Gearbox

The gearbox design follows the general approach of the previous DriveSE implementation, however with code improvements, the results will likely be different than prior versions. The gearbox is assumed to have 3 stages, with the user specifying a configuration code of either “EEP” or “EPP”, with the “E” representing epicyclic (planetary) gear stages and “P” representing parallel gear stages. For the epicyclic stages, the user also has to specify the number of planets, so the EEP input would require something like \([3, 3, 0]\) and EPP would require \([3, 0, 0]\). The user also specifies the overall target gear ratio, and then DrivetrainSE conducts a mass minimization of the three stage ratios that meet the target and minimize the overall mass.

The mass minimization is done in terms of an empirical estimate, previously presented in a number of papers suing the former DriveSE code. The mass of a single epicyclic or parallel stage is given by,

\[ m_{\text{epicyclic}} = K \tau_i \gamma \left[ \frac{1}{U_i} + \frac{1}{BU_i} + \frac{1}{B(U_i/2 - 1)} + (U_i/2 - 1) + (U_i/2 - 1)^2 + \frac{k_r(U_i - 1)^2}{B} + \frac{k_r(U_i - 1)^2}{B(U_i/2 - 1)} \right] \]

\[ m_{\text{parallel}} = K \tau_i \left[ \frac{1 + 1/U_i + U_i + U_i^2}{\prod_{i=1}^{n} U_j} \right] \]

\[ k_r = 0.4 \]

\[ K = 3.1469E - 3 \]

Where \( \tau_i \) is the input torque to the stage, \( U_i \) is the stage ratio, \( B \) is the number of planets, and \( \gamma \) is a safety factor equal to 1.1 for \( U_i < 5 \), otherwise 1.35.

The mass of the gearbox is the sum of the individual stage masses, plus estimates for the shrink disc and carrier masses, \( m_{\text{shrink disc}} = P_{\text{turbine}}/3 \) and \( m_{\text{carrier}} = 8000 \), where \( P_{\text{turbine}} \) is the turbine rated power in kilowatts.

The gearbox moment of inertia is estimated assuming the gearbox is a solid cylinder, with axial and transverse values of,

\[ I_0 = m_{\text{gearbox}} R_{\text{gearbox}}^2 / 2 \]

\[ I_1 = m_{\text{gearbox}} (3R_{\text{gearbox}}^2 + L_{\text{gearbox}}^2) / 12 \]

\[ R_{\text{gearbox}} = 0.005625 D_{\text{rotor}} \]

\[ L_{\text{gearbox}} = 0.012 D_{\text{rotor}} \]

This approach does not have the fidelity to estimate gearbox efficiency. This is therefore a user input value that is not affected by any of the calculations here.

Brake

The brake attaches to the high speed shaft for geared configurations or directly on the low speed shaft for direct drive configurations. It is regression based, but also allows for a user override of the total mass value. To obtain the brake mass in kilograms from the rotor torque in kilo-Newton meters (updated in 2020 by J. Keller), \( m_{\text{brake}} = 1.22 Q_{\text{rotor}} \).

The brake moment of inertia is taken from the equations of a solid disc with axial and transverse values of,

\[ I_0 = m R_{\text{disc}}^2 / 2 \]

\[ I_1 = I_0 / 2 \]

Where \( R_{\text{disc}} \) is assumed to be 1% of the blade length
Generator

The user has the option to select a simplified sizing of the generator, consistent with the level of fidelity of other components described here. However, a far more detailed and rigorous generator design approach is available through the GeneratorSE set of codes. The description of this methodology is beyond the scope of this document, and is best described in the original GeneratorSE report. Suffice to say here that this approach includes electromagnetic sizing and performance estimation, structural analysis and sizing through optimization constraints, basic thermal design, and more granular mass and cost roll-up. In this way the user can direct the optimizer to trade magnet, copper, and structural mass against one another to achieve the optimal generator design for a specific implementation and set of constraints. The user can choose from a number of different generator technologies:

- **PMSG-Outer**: Permanent magnet synchronous generator (outer generator - inner stator)
- **PMSG-Disc**: Permanent magnet synchronous generator (inner generator - outer stator) with solid disc stator support
- **PMSG-Arms**: Permanent magnet synchronous generator (inner generator - outer stator) with arm/spoke stator support
- **EESG**: Electrically excited synchronous generator
- **DFIG**: Doubly fed induction generator
- **SCIG**: Squirrel-cage induction generator

Each of the technologies have slightly different sets of required inputs that are best captured in the various examples. When doing detailed generator performance and sizing, the default technology for direct-drive configuration is PMSG-Outer and the for geared configurations it is DFIG.

When the user opts for the simplified generator model, the mass is estimated from either the rated torque or rated power. For the mass in kilograms and the rated power in kilowatts and rated torque in kilo-Newton meteers,

\[
m_{\text{generator}} = 37.68Q_{\text{rotor}} \quad \text{(direct-drive)}
\]

\[
m_{\text{generator}} = 7.4412P_{\text{turbine}}^{0.9223} \quad \text{(geared)}
\]

Generator performance is captured in the estimation of the mechanical-to-electrical conversion efficiency. This is reported as a function of rotational speed as a fraction of rated speed, but there is an allowance for user-override:

\[
\bar{\omega} = \omega/\omega_{\text{max}}
\]

\[
\eta(\omega) = 1 - (a/\bar{\omega} + b\bar{\omega} + c)
\]

Where \( \eta \) is the efficiency and the constants are:

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct</td>
<td>0.01007</td>
<td>0.06889</td>
<td>0.0200</td>
</tr>
<tr>
<td>Geared</td>
<td>0.01289</td>
<td>0.0</td>
<td>0.0851</td>
</tr>
</tbody>
</table>

Whether doing detailed or simplified modeling of the generator, the moment of inertia is estimated in the same way. Like the gearbox, for the purposes of estimating the moment of inertia, the generator is assumed to be a solid cylinder, so the axial and transverse values are:

\[
I_0 = m_{\text{generator}}R_{\text{generator}}^2/2
\]

\[
I_1 = m_{\text{generator}}(3R_{\text{generator}}^2 + L_{\text{generator}}^2)/12
\]

\[
R_{\text{generator}} = 0.0075D_{\text{rotor}}
\]

\[
L_{\text{generator}} = 0.027D_{\text{rotor}}
\]

5.8. Module documentation 233
Generator Cooling

The generator cooling, or HVAC system, is a regression based mass estimate from the rated power, with an allowance for a user input override. To obtain the cooling mass in kilograms from the power in kilowatts, 

\[ m_{\text{cool}} = 0.08P_{\text{turbine}}. \]

The cooling system moment of inertia is taken from the equations of a simple ring mass, assuming the cooling mass is located at about 75% of the outer generator radius, with axial and transverse values of,

\[ I_0 = m(0.75R_{\text{generator}})^2 \]
\[ I_1 = I_0/2 \]

Power Electronics

The power electronics (converter and transformer) are empirical, regression based estimates of mass from the rated power of the turbine. There is no electrical load analysis behind these estimates, but a user override of the total mass value can be provided. To obtain the mass in kilograms from the rated power in kiloWatts,

\[ m_{\text{converter}} = 0.77875P_{\text{turbine}} + 302.6 \]
\[ m_{\text{transformer}} = 1.915P_{\text{turbine}} + 1910 \]

Where \( P_{\text{turbine}} \) is the rated power.

The moment of inertia for both converter and transformer assumes that each is a box with side lengths 1.5% of the rotor diameter. For all principal axes, the moment of inertia is \( I = ms^2/6 \). Converter and transformer take on different moment of inertia values due to their different mass values.

Bedplate

Different bedplate models are used depending on if a geared or direct drive configuration is used. The height and length of the bedplate, regardless of configuration, is set by the user input dimensions such as overhang and desired height.

Geared

For geared layouts, the bedplate consists of twin I-beams that run along the bottom of the length of the nacelle. It is assumed that on top of these I-beams sits the platform, upon which the different nacelle sub-components are affixed at the appropriate location and tilt. The mass is the standard summation for I-beam cross sections,

\[ A_I = 2w_ft_f + h_w t_w \]
\[ m_{\text{bedplate}} = 2\rho A_I L_{\text{bedplate}} \]

Where \( w_f \) and \( t_f \) are the flange width and thickness and \( h_w \) and \( t_w \) are the web height and thickness, illustrated in n: numref:fig_i beam_cross. The factor of two on the mass equation is to account for the twin I-beams.

The moment of inertia for the geared bedplate is taken from standard expressions for I-beam of a finite length with a coordinate system of \( x \) along the axial length, \( y \) consistent with a right-hand coordinate system when \( z \) is pointed up (from the base flange to the top flange),

\[ I_{xx} = \rho L_{\text{bedplate}}(2w_f t_f^3 + H_t^3) + m_{\text{bedplate}}y_{off}^2 \]
\[ I_{yy} = \rho L_{\text{bedplate}}(w_f H^3 - (w_f - t_w)h_w^3)/12 + m_{\text{bedplate}}L_{\text{bedplate}}^2/12 \]
\[ I_{zz} = \rho L_{\text{bedplate}}(2t_w w_f^3 - h_w t_w^3)/12 + m_{\text{bedplate}}L_{\text{bedplate}}^2/12 + m_{\text{bedplate}}y_{off}^2 \]
\[ y_{off} = D_{tt}/4 \]

Where \( \rho \) is the density of steel, \( y_{off} \) is the offset of the bedplate from the tower centerline, and \( D_{tt} \) is the diameter of the tower-top.
Direct-Drive

The direct-drive bedplate is a tapered elliptical cone that marries the nose (turret) to the yaw drive at the tower top. The choice of an elliptic cross-sections makes the steps to calculate the mass properties more involved, but using standard geometric equations.

The ellipse is defined in the x-z plane, with the centerline, outer curve, and inner curve defined by,

\[ x_c(\theta) = L_{\text{bedplate}} \cos(\theta) \]
\[ x_{\text{out}}(\theta) = (L_{\text{bedplate}} + D_{tt}/2) \cos(\theta) \]
\[ x_{\text{in}}(\theta) = (L_{\text{bedplate}} - D_{tt}/2) \cos(\theta) \]
\[ z_c(\theta) = H_{\text{bedplate}} \sin(\theta) \]
\[ z_{\text{out}}(\theta) = (H_{\text{bedplate}} + D_{\text{nose}}/2) \sin(\theta) \]
\[ z_{\text{in}}(\theta) = (H_{\text{bedplate}} - D_{\text{nose}}/2) \sin(\theta) \]

Where \( \theta \) is the parametric angle that varies from \([0, \pi/2]\) for standard upwind configurations or \([\pi, \pi/2]\) for downwind, \(L_{\text{bedplate}}\) is the major axis, and \(H_{\text{bedplate}}\) is the minor axis. The effective cross sectional diameter and area is approximated by,

\[ D_{\text{bedplate}}(\theta) = \sqrt{(x_{\text{out}} - x_{\text{in}})^2 + (z_{\text{out}} - z_{\text{in}})^2} \]
\[ A_{\text{bedplate}}(\theta) = \pi(D_{\text{bedplate}}^2 - (D_{\text{bedplate}} - 2t_{\text{bedplate}})^2)/4 \]

To compute the mass, the area must be swept over the arc length of the ellipse. This calculation is made simpler by discretizing the ellipse into a series of arcs and using the average diameter and area in those arcs. The arcs are defined by the central angle relative to the origin, which is related to the parametric angle by \(\tan \phi = (L_{\text{bedplate}}/H_{\text{bedplate}}) \tan \theta\). Arc lengths from the origin are calculated using incomplete elliptic integrals of the second kind, \(s = L_{\text{bedplate}} E(\phi, e)\), so the discrete arc segments are \(s_i = L_{\text{bedplate}} [E(\phi_i, e) - E(\phi_{i-1}, e)]\). The bedplate mass is finally \(\sum_i \rho s_i A_{\text{bedplate},i}\) using the \(\rho\) as the density of steel.

The moment of inertia calculation for the elliptical bedplate could likely be approximated in multiple ways. With the assumption of an effective diameter and arc length, each segment was calculated as a cylindrical shell and then rotated from its angle, \(\phi_i\), to the tower top coordinate system.
Nacelle Platform

The nacelle platform that attaches to the bedplate to provide a floor for the nacelle is currently assumed to have a mass and moment of inertia of 1/8 of the bedplate.

Nacelle Cover

The nacelle cover dimensions are calculated by assuming the biggest element or component in each direction and adding 10% margin. Imagine a box that extends from one end of the bedplate to the hub flange and goes around the generator. The cover is assumed to be made of fiberglass that is 4cm thick. With these assumptions, the cover mass in kilograms can be calculated as,

\[ L_{cover} = 1.1(\text{overhang} + 0.5 \times L_{bedplate}) \]
\[ W_{cover} = 1.1D_{generator} \]
\[ H_{cover} = 1.10.5D_{generator} + \max[0.5D_{generator}, H_{bedplate}] \]
\[ A_{cover} = 2(L_{cover}W_{cover} + L_{cover}H_{cover} + H_{cover}W_{cover}) \]
\[ m_{cover} = \rho t A_{cover} \]
\[ t = 0.04 \]

Where \( D_{generator} \) is the outer diameter of the generator and the terms, \( \rho \) is the density of fiberglass, and \( L, W, H, A \) refer to the length, width, height, and area.

The moment of inertia of the nacelle cover is determined by assuming a hollow, rectangular box. The principal moments of inertia are then,

\[ I_1 = m_{cover}(H_{cover}^2 + W_{cover}^2) - (H_{cover} - t)^2 - (W_{cover} - t)^2)/12 \]
\[ I_2 = m_{cover}(H_{cover}^2 + L_{cover}^2) - (H_{cover} - t)^2 - (L_{cover} - t)^2)/12 \]
\[ I_3 = m_{cover}(L_{cover}^2 + W_{cover}^2) - (L_{cover} - t)^2 - (W_{cover} - t)^2)/12 \]

Yaw System

The yaw system is approximated by assuming that the main mass contributions are from the friction plate and the yaw motors. To obtain the yaw system mass in kilograms,

\[ n_{motors} = 2\text{ceil}(D_{rotor}/30.0) - 2 \]
\[ m_{fp} = 0.0001\rho \pi D_t^2 D_{rotor} \]
\[ m_{yaw} = m_{fp} + n_{motors}m_{motor} \]
\[ m_{motor} = 190.0 \]

Where \( D_{rotor} \) is the rotor diameter in meters, \( D_t \) is the tower-top diameter, and \( \rho \) is the density of steel. The friction plate mass calculation is derived from assuming that the surface width is 10% of the tower top diameter and the thickness is 0.1% of the rotor diameter.

Since the yaw system is at the tower top coordinate system origin, it is assumed to not contribute to the nacelle moment of inertia calculation.
Nacelle and RNA mass summary

To aid in the tower structural analysis, the total mass and moment of inertia of the nacelle is summed about a coordinate system center at the tower top. This is a straightforward summation of the mass, and a mass-weighted average of the component center of mass. For the component moments of inertia, which are given about the component center of mass, the inertia tensor was first rotated through the driveshaft tilt, and then the parallel axis theorem was applied to move from the component center of mass to the tower top coordinate system. These operations can be expressed as,

\[
\begin{align*}
    m_{nac} &= \sum_i m_i \\
    \vec{r}_{nac} &= \frac{1}{m_{nac}} \sum_i m_i \vec{r}_i \\
    I_{nac} &= \sum_i \left[ R(\gamma)I_i R^T(\gamma) + m_i (\vec{r}_i \cdot \vec{r}_i)E_3 - \vec{r}_i \otimes \vec{r}_i \right]
\end{align*}
\]

Where \( m_i \) is the component mass, \( \vec{r}_i \) is the vector from the tower top coordinate system origin to the component center of mass, \( I_i \) is the component moment of inertia tensor, \( R(\gamma) \) is the 3-D rotation matrix about the y-axis for the tilt angle, \( E_3 \) is the 3x3 identity matrix, \( \cdot \) denotes the inner (dot) product, and \( \otimes \) denotes the outer product.

Structural Analysis

The structural analysis in DrivetrainSE is where the largest differences with the previous DriveSE code lies. Instead of analytical derivations of the forces and moments on the various elements and closed form expressions for sizing the components, we instead rely on Frame3DD to conduct the analysis and enable the use of optimization with stress constraints to ensure a valid design. Separate analyses are run for the rotating and non-rotating parts of the drivetrain, with some small and large differences depending on whether a direct-drive or geared configuration is employed.

Rotating Structural Analysis

Low-Speed Shaft

The Frame3DD-based analysis of the rotating components attached to the low-speed shaft is illustrated in Fig. 5.38. It is assumed that the force and moment vectors applied to the hub are transferred to the low speed drive shaft. This shaft is supported by two sets of main bearings. The first main bearing absorbs all of the force components and the second absorbs the \( y \)- and \( z \)-moment components. These bearings, in turn, apply a reaction force on the shaft. The \( x \)-moment component is the shaft torque, which is taken on by either the generator rotor, in direct-drive configurations, or the gearbox, in geared configurations, giving the reaction \( M_g \). Near the gearbox/generator, there are also some weight loads applied. This is the brake system for direct-drive or the shrink disk and carrier in the geared configuration.

Frame3DD determines the reaction forces and moments on the bearings and the stress along the shaft, from which a von Mises stress utilization constraint value is calculated, with a user-proscribed safety factor. The shaft deflection at the gearbox / generator stator connection is also saved as an output. This Frame3DD analysis uses a coordinate system such that the shaft lies along the x-axis (regardless of tilt), with the origin at the node on the far right where \( M_g \) is applied.
High-Speed Shaft

The high-speed shaft structural analysis is only done for geared drivetrain configurations. Compared to the low-speed shaft, it is also a much simpler analysis. The forces and moments are diagrammed in Fig. 5.39. It is assumed that the gearbox applies a pure torque on the shaft. This torque is the value of the torque applied to the low-speed shaft, divided by the gear ratio. The brake is attached to the high speed shaft, so its weight is applied at the shaft midpoint. The torque is absorbed by the generator, which applies a reaction torque in return.

Frame3DD determines the reaction moment and the stress along the shaft, from which a von Mises stress utilization constraint value is calculated, with a user-proscribed safety factor. The shaft deflection at the generator connection is also saved as an output. This Frame3DD analysis uses a coordinate system such that the shaft lies along the x-axis (regardless of tilt), with the origin at the generator connection node on the far right.
Stationary Structural Analysis

The structural analysis of the stationary drivetrain elements was split between direct-drive and geared configurations.

Direct-Drive

For direct-drive configurations, the force diagram modeled in Frame3DD is shown in Fig. 5.40. The forces and moments on the two main bearings, in addition to the bearing and housing weight, are transferred to the nose/turret. Additionally, the forces from the generator stator (usually just the weight) act on the nose/turret as well. The nose/turret is attached to the elliptically curved bedplate that is assumed to be fully clamped, with a corresponding reaction force. The weight force of all of the other nacelle components, such as the power electronics, bedplate, cooling system, fiberglass cover, etc. is also applied at the bedplate base.

As with the rotating analysis, Frame3DD determines the bedplate reaction forces, which are then transferred to the tower in TowerSE analysis. A von Mises stress utilization constraint is computed along the nose/turret and the bedplate as well. Finally, deflections at the bearings and stator attachment are also computed and compared against any structural limits set by the choice of bearing or the GeneratorSE structural design. This Frame3DD analysis uses a coordinate system such that the bedplate base node is at (0, 0, 0).

Geared

For geared configurations, the force diagram modeled in Frame3DD is shown in Fig. 5.41. The forces and moments on the two main bearings, in addition to the bearing and housing weight, the gearbox, generator, cooling, brake, power electronics, and all other miscellaneous nacelle equipment are transferred to the twin I-beams. This is done by creating a “ghost nodes” in Frame3DD that receive these loads, then transfer them through perfectly rigid elements to the I-beams. The I-beams sit atop the yaw drive on the tower top, so are assumed to be fully clamped at those attachment points.

Once the geared stationary analysis is described, the Frame3DD analysis and post-processing is similar to the direct-drive comments. Frame3DD determines the bedplate reaction forces, which are then transferred to the tower in TowerSE analysis. A von Mises stress utilization constraint is computed along the I-beams. Finally, deflections at the bearings and stator attachment are also computed and compared against any structural limits set by the choice of bearing or the GeneratorSE structural design. This Frame3DD analysis uses a coordinate system such that the center of the tower center axis is at (0, 0, 0).
5.8.5 FloatingSE

**Warning:** In our use of FloatingSE, we have not been able to obtain conceptual platform geometry results that we trust are valid and worth pursuing further. This may be due to a fundamental error in the formulation and implementation of FloatingSE. It may also be due to the inherent limitations in using steady-state or quasi-static analysis methods to tackle a problem that is driven by its dynamic nature and dynamic loads. NREL currently advises against reliance on FloatingSE. Instead, we are developing a multifidelity floating turbine and platform design capability in the Wind Energy with Integrated Servo-control (WEIS) project.

**Introduction**

The complexity of the physics, economics, logistics, and operation of a floating offshore wind turbine makes it well suited to systems-focused solutions. *FloatingSE* is the floating substructure cost and sizing module for the WISDEM. This document serves as both a User Manual and Theory Guide for FloatingSE. FloatingSE can be executed as a stand-alone module or coupled to the rest of the turbine through the WISDEM glue code. An overview of the package contents is below and substructure geometry parameterization in Section Geometry. With this foundation, the underlying theory of FloatingSE’s methodology is explained in Section Theory. This helps to understand the analysis execution flow described in Section Execution and the setup of design variables and constraints for optimization problems in Section Optimization.
Package Files

The files that comprise the FloatingSE package are listed in Table 5.13. Each file has a corresponding unit test file in the test_floatingse directory.

Table 5.13: File contents of the FloatingSE module.

<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>floating.py</td>
<td>Top level FloatingSE OpenMDAO Group</td>
</tr>
<tr>
<td>column.py</td>
<td>Components calculating mass, buoyancy, and static stability of vertical frustum columns</td>
</tr>
<tr>
<td>loading.py</td>
<td>Components for Frame3DD analysis of structure, mass summation, and displacement</td>
</tr>
<tr>
<td>map_mooring.py</td>
<td>Mooring analysis using pyMAP module</td>
</tr>
<tr>
<td>substructure.py</td>
<td>Final buoyancy and stability checks of the substructure</td>
</tr>
<tr>
<td>visualize.py</td>
<td>Standalone script that uses MayaVI to view substructure geometry</td>
</tr>
</tbody>
</table>

Geometry

This section describes the variables and methods used to parameterize the substructure geometry in FloatingSE. Typically, substructure designs have fallen into three classical regimes, which are shown in Fig. 5.42, each of which attains static stability through different physical mechanisms. A spar derives its stability from a deep drafted ballast. A semisubmersible derives its stability from distributed waterplane area, achieved with offset columns spread evenly around a main column or central point. A tension leg platform (TLP) uses taut mooring lines for its stability.

![Substructure Diagram](image)

Fig. 5.42: Three classical designs for floating turbine substructures.

Similar to [KHBC17], care was taken to parameterize the substructure in a general manner, so as to be able to use the same set of design variables to describe spars, semisubmersibles, TLPs, and hybrids of those archetypes. The intent is that this modular approach to substructure definition will enable rapid analysis of the majority of designs currently proposed by the floating wind development community, whether classical or novel in nature. Furthermore, generalizing the substructure definition also empowers the optimization algorithm to search a broad tradespace more efficiently by moving fluidly from one region to another.

With that intent in mind, the general configuration of a spar-type substructure is shown in Fig. 5.43, with nomenclature borrowed from the field of naval architecture. A semisubmersible configuration would have a similar diagram, but with multiple offset columns connected with pontoon elements. A TLP might look similar to a spar or semisubmersible, with taut mooring lines instead of the catenary ones shown.
Fig. 5.43: Geometry parameterization with common wind turbine and naval architecture conventions.
Inputs: WindIO

The parameterization of the input variables in the Geometry YAML file into FloatingSE is documented within the larger WindIO effort. When running FloatingSE directly as a standalone with a python script, users are encouraged to review the floating-specific examples for syntax.

Tapered Cylinders (Vertical Frustums)

A number of typical floating substructure designs, such as the spar or semisubmersible, contain vertically oriented columns. In FloatingSE, these columns are assumed to have a circular cross-section making them, formally, vertical frustums. These frustums are assumed to be ring-stiffened to support the buckling loads inherent in a submerged support structure. The number of columns, their geometry, and the ring stiffeners are parameterized in the FloatingSE module according to the diagrams in Fig. 5.43, Fig. 5.44, Fig. 5.45, and Fig. 5.46. The main column is assumed to be centered at \((x = 0, y = 0)\), directly underneath the turbine tower (note that off-centered turbines are not yet supported). Other columns are referred to as offset columns, and are assumed to be evenly spread around the main column. The material of the vertical columns is currently assumed to be ASTM 992 steel. Future developments will include the option to select one of multiple material options for each section in each cylinder.

![Vertical frustum geometry parameterization.](image)

Discretization

To allow for varying geometry parameters along the length of substructure columns, the larger components are divided into sections. The user may specify the number of overall sections, \(n_s\), and the geometry of each section. Some of the geometry parameters are tied to the nodes that bracket each section, such as column diameter and wall thickness, with linear variation between each node. Other parameters are considered constant within each section, such as the spacing between ring stiffeners. The number of sections should resemble the physical number of cans or sections used in the manufacturing of the real article.
Stiffeners

The ring stiffener geometry is depicted in Fig. 5.45, and Fig. 5.46.

Fig. 5.45: Vertical frustum cross-section with stiffeners

Fig. 5.46: Vertical frustum stiffener geometry parameterization.
Material Properties

The material of the vertical columns is commonly assumed to uniformly be ASTM 992 steel. Careful selection of the layering in the yaml-file inputs allows for much more complex material selections for different parts of the platform.

Ballast

Stability of substructure columns with long drafts can be enhanced by placing heavy ballast, such as magnetite iron ore, at their bottom sections. The user can specify the density of the permanent ballast added and the height of the ballast extent within the column. Variable ballast, as opposed to permanent ballast, is water that is added or removed above the permanent ballast to achieve neutral buoyancy as the operating conditions of the turbine change. A discussion of variable water balance in the model is found in Section Hydrostatic Stability.

Buoyancy Tanks (and Heave Plates)

Buoyancy tanks are modeled as a collar around the column and are not subject the same taper or connectivity constraints as the frustum sections. They therefore offer added buoyancy without incurring as much structural mass or cost. Moreover, they can also serve to augment the heave added mass like a plate. In addition to their diameter and height, the user can adjust the location of the buoyancy tank from the column base to the top. Buoyancy tanks can be added to either the main and/or offset columns.

Pontoons and Support Structure

Many substructure designs include the use of pontoons that form a truss to connect the different components, usually columns, together. In this model, all of the pontoons are assumed to have the identical thin-walled tube cross section and made of the same material as the rest of the substructure. The truss configuration and the parameterization of the pontoon elements is based on the members shown in Fig. 5.47 with lettered labels. The members are broken out into the upper and lower rings connecting the offset columns (B and D, respectively), the upper and lower main-to-offset connections (A and C, respectively), the lower-base to upper-offset cross members (E), and the V-shaped cross members between offset columns (F).

Fig. 5.47: Parameterization of truss elements in substructure.
Mooring Lines

The mooring system is described by the number of lines, their geometry, and their interface to the substructure. The mooring diameter is set by the user and determines the breaking load and stiffness of the chain, via correlation, described in Section Theory. The mooring lines attach to the substructure at the fairlead distance below the water plane, as shown in Fig. 5.43. The lines can attach directly to a substructure column or at a some offset from the outer shell. Note that bridle connections are not yet implemented in the model. The mooring lines attach to the sea floor at a variable distance, the anchor radius, from the substructure centerline, also set by the user.

By default, the mooring system is assumed to use a steel chain with drag embedment anchors. Other mooring available for selection are nylon, polyester, steel wire rope (IWRC) and fiber-core wire rope. The only alternative anchor type is currently suction pile anchors, but there are plans to include gravity anchors as well. The standard configuration for TLPs is the use of taut nylon mooring lines with suction-pile anchors.

Mass and Cost Scaling

The mass of all components in the modeled substructure is captured through calculation of each components’ volume and multiplying by its material density. This applies to the frustum shells, the ring stiffeners, the permanent ballast, the pontoons, and the mooring lines. However, the model also acknowledges that the modeled substructure is merely an approximation of an actual substructure and various secondary elements are not captured. These include ladders, walkways, handles, finishing, paint, wiring, etc. To account for these features en masse, multipliers of component masses are offered as parameters for the user as well. Capital cost for all substructure components except the mooring system is assumed to be a linear scaling of the components masses. For the mooring system, cost is dependent on the tension carrying capacity of the line, which itself is an empirical function of the diameter. Cost factors are especially difficult to estimate given the proprietary nature of commercial cost data, so cost rates and estimates should be considered notional.

Theory

Design Load Cases (DLCs)

The user must specify the metocean conditions that drive the wind and wave loads upon the floating substructure. FloatingSE currently only uses the single load case of maximum thrust coincident with maximum wave loading to drive the substructure design. The assumption is that this load case would be the driver for substructure sizing and stability. Ideally, multiple DLCs and metocean conditions would be used for design optimization. The capability to optimize over multiple DLCs will be added to future versions of the model. By not currently including a formal set of IEC DLCs, the conceptual designs derived in this work should be considered preliminary and subject to extensive revision once other load cases, and higher-fidelity analysis, is brought to bear.

Load Path

As with other WISDEM models, the primary simplification in FloatingSE is the treatment of all loads as pseudo-static. This approximation significantly reduces computational time and resources, since an accurate calculation of dynamic loads requires more sophisticated numerical tools and simulations. However, dynamic effects can still dominate drive component sizing for floating platforms, thus the static loading assumption gives a rough approximation of the total loading. Furthermore, fatigue effects and structural lifetime estimates are also excluded for now, but could be incorporated in future developments.

A floating wind turbine undergoes loading from a number of sources. The primary loading source for the tower comes from the aerodynamic loads induced by the rotor. The substructure must resist the combination of both rotor loads and
hydrodynamics loads, with the latter becoming more and more important as water depth and wave heights increase. *FloatingSE*, together with other WISDEM modules, accounts for these two dominant load sources, as well as the self-loading of gravity loads. Other sources of loading, such as installation loads, accidental loads, vortex-induced vibrations, ice, and seismic loads are ignored.

### Wind and Wave Loads

Wind drag loads are applied to the tower body and the upper part of the substructure that extends above the waterline. They are not applied to connecting truss members that may be part of the substructure geometry. These drag loads are computed assuming the tower and columns are smooth circular cross-sections and that the drag coefficient can be selected as a function of the flow Reynolds number [Ros61]. The aerodynamic drag force is a function of height, since the wind profile and cross-sectional geometry varies along that dimension. For the wind profile, the standard power-law scaling is used,

\[
U_a(z) = U_{ref} \left( \frac{z}{z_{ref}} \right)^\alpha,
\]

where \(U_a(z)\) is the wind velocity as a function of height, \(U_{ref}\) is a reference wind speed measured at a reference height, \(z_{ref}\), and \(\alpha\) is the shear exponent used in the power-law approximation of wind profiles. The wind profile then feeds the aerodynamic drag, Reynolds number, and drag coefficient,

\[
dF(z) = \frac{1}{2} \rho_a U_a^2(z) d(z) c_d(Re)dz; \quad Re_d = \frac{\rho_a U_a(z)d(z)}{\mu_a},
\]

where \(Re_d\) is the Reynolds number based on diameter, \(\rho_a\) and \(\mu_a\) are the density and viscosity of air, \(d(z)\) is the diameter of the column as a function of height, \(c_d\) is the 2-D drag coefficient, and \(dF(z)\) is the force per unit length in the \(z\)-direction.

Wave drag loads arise from similar processes, but are computed using Morison’s equation, a semi-empirical expression that predicts the total hydrodynamic loads. It is comprised of two components, one for viscous drag contributions and another for inertial effects (which includes incident, diffracted, and radiated wave effects). For flow past structures with circular cross sections, Morison’s equation for force per unit length \(dF(z)\) takes the form,

\[
dF(z) = \frac{\pi d^2(z)}{4} \rho_w C_m \dot{U}_w(z)dz + \frac{1}{2} \rho_w U_a^2(z) d(z) c_d(Re)dz,
\]

where \(C_m\) is the added mass coefficient (assumed to be \(C_m = 2\)), \(U_w(z)\) is the current speed as a function of height, \(\dot{U}_w(z)\) is the acceleration as a function of height, and the Reynolds number is computed by substituting in the appropriate properties for water,

\[
Re_d = \frac{\rho_w U_w(z)d(z)}{\mu_w}.
\]

To compute Morison’s equation, expressions for local fluid velocity and acceleration are required. Wave particle velocity (not the same as the bulk velocity of the wave) is assumed to follow linear (Airy) wave theory

\[
U_w(z) = a \omega \frac{\cosh[\kappa(z+D)]}{\sinh(\kappa D)} \cosh(\kappa x - \omega t); \quad \omega = \frac{2\pi}{T} = \sqrt{g\kappa \tanh(\kappa D)}.
\]

where \(\omega\) is the circular frequency, \(T\) is the wave period, \(a\) is the wave amplitude (half of the significant wave height), \(D\) is the total water depth, \(g\) is the acceleration of gravity, and \(\kappa\) is the wave number numerically computed from the dispersion relationship given as the last expression in Equation [eqn:Uwave]. Note that the horizontal particle velocity varies in time and space (by the \(\kappa x - \omega t\) term). Thus, the individual particles in the wave are also accelerating at different rates,

\[
U_w(z) = a \omega^2 \frac{\cosh[\kappa(z+D)]}{\sinh(\kappa D)} \sinh(\kappa x - \omega t).
\]

For simplicity, *FloatingSE* only considers the maximum velocity and acceleration at a given height, and makes a conservative assumption that they are concurrent in time and space. This essentially means ignoring the \(\kappa x - \omega t\) term, since the maximum of any hyperbolic sine or cosine term is one.

### 5.8. Module documentation

247
Rotor Nacelle Assembly (RNA) Loads

From a quasi-steady-state point of view, the RNA loads reduce to three forces and three moments along the main coordinate axes [Dan16]. The thrust is the biggest force responsible for the bending moment distribution along the tower and loads on the substructure. There is the additional effect of the gravitational load caused by the offset of the RNA center of mass from the tower centerline. This effect is more pronounced for downwind turbines than upwind turbines, but is included regardless. FloatingSE does not compute the force and moment components directly, but rather accepts them as inputs from other WISDEM modules or from the user directly.

Structural Analysis

The analysis tool, Frame3DD, is an open-source tool for static and dynamic structural analysis of 2-D and 3-D frames and trusses with elastic and geometric stiffness. It computes the static deflections, reactions, internal element forces, natural frequencies, and modal shapes using direct stiffness and mass assembly [GP15]. The WISDEM toolkit developed a python interface, pyFrame3DD, to avoid the use of intermediate input and output text files. The integration of all loads happens within Frame3DD, where the whole floating turbine load path, from the rotor to the keel of the substructure, is modeled with Timoshenko frame elements [TG70].

Discretization

For the finite element structural analysis of the substructure, the discretization of the main columns into a handful of sections is still too coarse to capture the appropriate physics. Long slender components, such as the tower and substructure columns, are broken up into a three-times finer discretization than the physical cans that they are actually made of. The sectional and nodal variables are re-sampled at this finer spacing. These additional discretization points give greater resolution of internal forces and natural frequencies. Substructure pontoons are represented as single frame elements. Frame elements are described by their cross sectional properties (area, moments of inertia, modulus of elasticity, and mass density) and starting and ending nodes. For simple geometries, such as pontoons with tubular cross sections, these properties are straightforward calculations. For the turbine tower, tubular cross section properties are also used, albeit at a finer discretization. For substructure columns, it is assumed that the permanent or variable ballast and bulkheads are not load-bearing, so tubular cross section properties are also used to represent the column shell. However, the material mass density of the frame element is scaled to reflect the true mass of the whole section, including ballast, to ensure that gravity loads are captured correctly.

For the tubular cross sections, the critical properties needed by Frame3DD given user inputs of diameter, $d$, and tube (or wall) thickness, $t$, are,

- Outer radius, $r_o = d/2$
- Inner radius, $r_i = r_o - t$
- Material area, $A = \pi (r_o^2 - r_i^2)$

Bending second moment of area, $I_{xx} = I_{yy} = \pi \left( r_o^4 - r_i^4 \right)$

Torsion second moment of area, $I_{zz} = J_0 = I_{xx} + I_{yy}$

Shear area, $A_s = A/ \left[ 1.124235 + 0.055610 \left( \frac{r_i}{r_o} \right) + 1.097134 \left( \frac{r_i}{r_o} \right)^2 - 0.630057 \left( \frac{r_i}{r_o} \right)^3 \right]$

Bending modulus, $S = I_{xx}/r_o$

Torsion modulus (shear constant), $C = I_{zz}/r_o$

Note that the shear area expression is an empirical relationship as opposed to an analytical expression.
Loads

All of the loads described above are integrated together within Frame3DD. These loads include,

- Rotor-nacelle-assembly loads (thrust, moments, etc)
- Mooring line force
- Wind and wave loading
- Gravity loads (weight distribution)
- Hydrostatic pressure loads, including buoyancy

The forces, moments, and mass properties of the rotor-nacelle assembly (RNA) are inputs to FloatingSE (mass properties are assumed to be relative to the tower top position). It assumed that the RNA is a rigid body with respect to the tower modes and the mass properties, forces, and moments, are applied to the corresponding node in the model. The forces along each mooring line are applied to the connection point nodes on the structure. The wind and wave forces per unit length in Equations [eqn:drag] and [eqn:morison] are applied as trapezoidally varying loads along the column elements. Other loads applied to the structure include the gravity loads, and the buoyancy acting on the submerged elements.

Boundary Conditions

Multiple boundary conditions are applied to the structure. The mooring system stiffness matrix (linearized about the neutral position) is applied at the mooring connection nodes. However, even with the mooring stiffness, the finite element analysis would otherwise still regard the structure as unrestrained and incapable of supporting any static loads. Thus, in order to successfully compute stress and buckling limits in a well-posed problem, an additional rigid boundary condition (in all 6 DOF) is imposed at the bottom node of the main column.

Outputs

Structural analysis outputs include mass properties of the structure, member stresses, and summary forces and moments on the body. Mass properties include the total mass of the floating turbine and the mass of the substructure itself. The calculations also allow for easy computation of the center of mass of the structure (not accounting for variable ballast) and the center of buoyancy (centroid of the submerged volume). The first two natural frequencies of the structure are also computed to compare against the range of standard wave frequencies and rotor passing frequencies (1P and 3P). Next, the reaction forces and moments at the boundary node at the keel are taken as the total loading on the structure. These are used later in the static stability calculations to ensure that the mooring lines provide adequate restoring force and moment. Finally, the axial and shear forces within each frame element are extracted and converted to stresses using
cross-sectional properties. These element member follow the sign convention in Fig. 5.48,

\[\sigma_z = \frac{N_z}{A} - \sqrt{\frac{M_x^2 + M_y^2}{S}}\]
\[\tau_{z\theta} = \frac{T_z}{C} + \sqrt{\frac{V_x^2 + V_y^2}{A_s}}\]

where \(N\) is the axial force (tension or compression), \(T\) is the torsional moment, \(V\) is the shear force, \(M\) is the bending moment, \(\sigma_z\) is the axial stress, and \(\tau_{z\theta}\) is the shear stress across axial and hoop principle directions.

Hoop stress of the tower is estimated from the dynamic pressure of the wind loads using the Eurocode method [EuropeanCfStandardisation93]. Hoop stress of the submerged columns is determined using the dynamic and static pressure heads of the water.

\[\sigma_{\theta,\text{Euro}} = k_w q_{\text{max}} \frac{d - t}{2t}; \quad q_{\text{max}} = \frac{1}{2} \rho_a U_a^2\]
\[\sigma_{\theta,\text{hydro}} = (q_{\text{max}} + p_{\text{hydro}}) \frac{d - t}{2t}; \quad q_{\text{max}} = \frac{1}{2} \rho_w U_w^2\]
\[p_{\text{hydro}} = \rho_w g \left( a \frac{\cosh [\kappa (z + D)]}{\cosh (\kappa D)} - z \right)\]

where \(\sigma_\theta\) is the hoop stress, \(q_{\text{max}}\) is the maximum dynamic pressure on a cross-section, and \(p_{\text{hydro}}\) is the hydrostatic pressure with contributions from wave motion and the static head. In the Eurocode method, \(k_w\) is the dynamic pressure factor for hoop stress calculation using cylinder dimensions and an external pressure buckling factor. Note that the argument, \((z)\), was dropped from many of the terms without losing generality.

**Code Compliance as Utilizations**

Once the stress components of all structural members are computed, they are compared against design code standards for compliance, and serve as design constraints when conducting optimization. Multiple code standards are used across all components. For all columns, the tower, and substructure pontoons, stress components (axial, shear, and hoop) are combined into a von Mises, equivalent, stress,

\[\sigma_{\text{vm}} = \sqrt{\sigma_a^2 + \sigma_\theta^2 - \sigma_a \sigma_\theta + 3 \tau_{a\theta}^2}\]

where \(\sigma_{\text{vm}}\) is the von Mises stress, \(\sigma_a\) is the axial stress, \(\tau_{a\theta}\) is the shear stress across axial and hoop principle directions, and \(\sigma_\theta\) is chosen as the relevant hoop stress. The von Mises stress is compared against the yield stress, \(\sigma_y\), and a safety factor as a utilization criterion.

Main column, offset column, and tower segment stresses and geometry are also evaluated against a shell buckling criterion published by [EuropeanCfStandardisation93] and a global buckling criterion published by [GermanischerLloyd05]. Note that the implementation of the Eurocode buckling is modified slightly so as to produce continuously differentiable output. See [Dam16] for a more detailed exposition.

For submerged columns, additional code standard utilization ratios are taken from the American Petroleum Institute, Bulletin 2U (specifically the procedure outlined in Appendix B) [AmericanPIAPI04]. These standards also apply shell and general buckling criterion with a margin of safety in a manner that accounts for stiffeners and the common buckling modes of submerged structures. Future efforts will also apply Bulletin 2V, the standards for plates, to the legs that support taut mooring lines.
Mooring Lines

The quasi-steady mooring system analysis is handled by the external Mooring Analysis Program (MAP++) library [MJR13], which has convenient Python bindings to access the simulation output, bundled into the WISDEM pyMAP module. MAP++ is designed to model the steady-state forces on a Multi-Segmented, Quasi-Static (MSQS) mooring line. Seabed contact, seabed friction, and multi-element mooring lines with arbitrary connection configurations can be analyzed. MAP++ inputs include sea depth, geometry descriptions of the mooring line connections, and material properties of the lines. For chain and rope-based cables, these material properties are not easily derived and would be typically provided by a manufacturer. We borrow from the approach of the popular Orcina OrcaFlex software [Orcina18] and use the following expressions,

\[
M_{BL} = 2.74 \times 10^7 d^2 (44 - 80d) \quad [N]
\]

\[
\text{mass} = 19.9 \times 10^3 d^2 \quad [\text{kg/m}]
\]

\[
A = 2 \left( \frac{\pi d^2}{4} \right) \quad [m^2]
\]

\[
E_A = 8.54 \times 10^{10} d^2 \quad [N]
\]

\[
\text{cost} = 3.415 \times 10^4 d^2 \quad [\text{USD}]
\]

where \(M_{BL}\) is minimum breaking load, \(d\) is the diameter of a single half-chain link, \(A\) is the chain cross-sectional area, \(E\) is the Young’s modulus, \(E_A\) is the axial stiffness. When conducting optimization, the expression for \(M_{BL}\) is poorly posed due to its limited range of diameter applicability, so a linear fit is used instead,

\[
M_{BL} = 1000 \max (1.0, -5445.3 + 176972.7d)
\]

Hydrostatic Stability

Neutral Buoyancy

Any floating body requires enough water displacement to create sufficient buoyancy force such that the body stays afloat in the most extreme loading and environmental conditions. This level of displacement would otherwise be overkill for more benign loading conditions. Since a floating turbine is designed for a constant hub height, variable amounts of ballast are required to maintain a neutrally buoyant system for all operating conditions. The variable ballast is simply ocean water that is pulled in or pumped out of holding areas within the substructure columns.

In FloatingSE, the variable ballast water mass is calculated as the difference between the total mass of displaced water and the total mass of the floating turbine. This mass is then divided by the water density to obtain the variable ballast volume, which is then compared to the frustum shell cross section profile above the permanent ballast to determine the height of the water ballast within the column. Once this is determined, the final center of mass of the system can be determined.

Surge/Sway Stability

Surge and sway stability is not actively tracked over the coarse of a load case. Instead the total surge force on the structure is calculated at the initial conditions and compared to the restoring force of the mooring system at the maximum allowable surge offset, which is specified by the user.

The surge direction is assumed to be aligned with the wind vector, which is aligned with the \(x\)-axis. Since the rotor yaw is assumed to be \(0^\circ\), the surge forces on the turbine include the rotor thrust and the wind and wave drag on the tower and substructure. The final surge force over the whole structure is taken from the \(x\)-direction reaction force of the reaction node in Frame3DD.

The restoring force is calculated as the smallest possible restoring force after a displacement in any angular direction in the mooring model. Since the alignment of the mooring lines relative to the incoming wind direction is arbitrary,
a maximum offset is simulated at 2° increments around the unit circle. Also recorded in this survey is the maximum mooring line tension in any line, in any direction, for comparison against the minimum breaking load value,

\[ F_{x,\text{restore}} = \min_{i \in a} F_{x,i} \quad T_{\text{moor}} = \max_{l \in L, i \in a} T_{l,i}; \quad L = \{1, 2 \ldots n\text{lines}\}, \; a = \{0^\circ, 2^\circ \ldots 360^\circ\} \]

where \( F_x \) is the surge force and \( T \) is the tension. If restoring force at this maximum offset is greater than the surge force applied, then the system is considered stable in surge. Since the wind and wave profiles are essentially 2-D in the \( x-z \) plane, the sway stability is given the same status as surge stability.

**Pitch Stability**

The approach to pitch stability determination is similar to that of surge stability. The total pitching moment on the floating turbine is calculated and compared to the restoring moment at the maximum allowable angle of heel. If the restoring moment at this max heel angle is greater than the pitching moment applied, the system is said to be statically stable in pitch.

Similar to the surge force calculation, the total pitching moment is determined from the reaction moment at the boundary condition in the Frame3DD analysis. The pitching moment has contributions from the wind and wave loads on the structure, the rotor forces and torques, the buoyancy forces on the submerged substructure, and the off-center weight of components (e.g. the RNA).

The restoring pitching moment has two primary contributions. The first is from the mooring lines. Similar to the surge force calculation, here the floating turbine is deflected in pitch by the maximum allowable heel angle and the mooring forces are recorded. The restoring moment contribution from the mooring system is computed as,

\[ M_{\text{moor}} = \sum_i r_{cm,i} \times F_l \]

where \( r_{cm,i} \) is the vector from the center of mass to the mooring connection, and \( F_l \) is the force applied by the \( l\text{-line} \) mooring line. As above, \( F_l \) is taken as the minimum set over the possible orientations of the mooring lines relative to the direction.

The second contributing restoring moment comes from the motion of the center of buoyancy away from alignment with the center of mass. This is a standard calculation in naval architecture [TD14] and is diagrammed in Figure [fig:metacenter]. In this diagram, the center of mass is denoted, \( G \), the center of buoyancy is \( B \), and the metacenter is \( M \). In neutral conditions (Figure [fig:metacenter][a]), all of these points are vertically aligned.

As the structure lists or heels, the center of buoyancy shifts toward the side of the structure that is more submerged (from \( B \) to \( B' \)) and the buoyancy force no longer passes through the center of mass. Instead, the buoyancy force passes through the metacenter with an effective moment arm of \( GZ \) from the center of mass (Figure [fig:metacenter][b]). The metacenter is defined as the common point through which the buoyancy force acts as it pitches through small displacements, for bodies with sufficient freeboard margin.

The metacenteric height, \( GM \) is most easily calculated as an offset from the center of buoyancy (\( BM \)) by,

\[ h_{\text{meta}} = M - G = GM = BM + BG; \quad BM = \frac{I_w}{V} \]

where \( BG \), the distance between the centers of buoyancy and gravity is easily calculated, \( I_w \) is the second moment of area of the substructure waterplane (with units of ) and \( V \) is the total volume of displacement (with units of ). Note that for semisubmersible type geometries, \( I_w \) is calculated with the parallel axis theorem for all of the columns at the waterplane,

\[ I_w = \sum_i (I_{w,i} + S_i r_i^2) \]

where \( S_i \) is the waterplane cross sectional area of the \( i\text{-col} \) column and \( r_i \) is the distance from the waterplane centroid to the \( i\text{-col} \) column centroid.
Fig. 5.49: Static stability of floating offshore wind turbine in neutral position.
Fig. 5.50: Static stability of floating offshore wind turbine in pitched position.
The restoring moment is then the buoyancy force acting through the restoring arm, $GZ$,

$$M_{\text{meta}} = F_B GZ = F_B G M \sin \phi$$

where $\phi$ is the angle of heel.

For this reason, the metacenter must be located above the center of mass for static stability. This condition is imposed on the design as a constraint. Note that the total volume of displacement, and the subsequent buoyancy force, is not recalculated in the perturbed configuration. It is assumed that the angles of deflection are small and that there is sufficient freeboard and design symmetry such that the total displacement is constant.

The total restoring pitching moment is then the sum of two contributions,

$$M_{y,\text{restore}} = M_{y,\text{moor}} + M_{\text{meta}}$$

**Hydrodynamic Stability**

Floating bodies are typically modeled, for small motions and linearized behavior, as a second-order differential system with mass, damping, and spring stiffness terms,

$$(M + A) \ddot{x} + C \dot{x} + K = F(t)$$

where $x \in \mathbb{R}^6$ is the six-degree of freedom vector (commonly ordered as 1-surge, 2-sway, 3-heave, 4-roll, 5-pitch, 6-yaw), $M$ is the mass matrix, $A$ is the added mass matrix, $C$ is the damping matrix, and $K$ is the stiffness matrix. The right-hand side of the equation captures the time-dependent summation of all forces.

As a low-fidelity, quasi-static sizing and cost module, FloatingSE does not attempt to capture all of the matrix entries or forcing terms of the hydrodynamics. A more sophisticated time- or frequency-domain solver, where these quantities are calculated, may be linked or included into FloatingSE in the future. Nevertheless, it does attempt to compute the diagonal entries of the mass and stiffness matrices in order to derive the rigid body natural frequencies of the system,

$$f_i = \frac{\omega_i}{2\pi} = \frac{1}{2\pi} \sqrt{\frac{K_{ii}}{M_{ii} + A_{ii}}}, \quad \forall i \in [1\ldots6]$$

where $f_i$ are the frequencies of the eigenmodes and $\omega_i$ is the circular frequency. The mass matrix diagonal entries, $M_{ii}$, are simply the mass and moments of inertia of the whole system,

$$M_{11} = M_{22} = M_{33} = m_{\text{sys}}; \quad M_{44} = I_{xx,\text{sys}}; \quad M_{55} = I_{yy,\text{sys}}; \quad M_{66} = I_{zz,\text{sys}};$$

where the coordinate system notation is consistent with that of Figure [fig:diagram].

The added mass matrix diagonal entries are evaluated via standard strip theory for the tapered vertical columns. The added mass for the system is a summation over the columns, using the parallel axis theorem for the rotational degrees of freedom. Pontoon contributions to system added mass are currently ignored. The column quantities are calculated as,

$$A_{11} = A_{22} = \rho V; \quad A_{33} = \left(\frac{1}{2}\right) \frac{8}{3} \rho \max \left[R^3(z)\right]; \quad A_{44} = A_{55} = \pi \rho \int (z - zcb) R^2(z)dz; \quad A_{66} = 0.0;$$

where $\rho$ is the water density, $R(z)$ is the column radius along its axis, and $V$ is the submerged volume. The extra factor of $1/2$ in $A_{33}$ is included to account for the fact that the top of the column extends above the waterline. Also, the integral in $A_{55}$ is only evaluated along the submerged portion of the column.

The stiffness matrix is comprised of contributions from the mooring and hydrostatic stiffness. The mooring linearized stiffness matrix is output directly from MAP++ and needs no additional processing within FloatingSE. The hydrostatic stiffness, for a vertical column, is derived from the same principals described above regarding the metacentric height,

$$K_{ii} = K_{ii}^{\text{moor}} + K_{ii}^{\text{hydro}}; \quad K_{33}^{\text{hydro}} = \rho g S_{\text{sys}}; \quad K_{44}^{\text{hydro}} = K_{55}^{\text{hydro}} = \rho g V h_{\text{meta}}$$

5.8. Module documentation 255
where $S_{sys}$ is the waterplane area of the system.

Once the rigid body natural frequencies (eigenmodes) of the system are calculated, they are compared against the standard wave frequencies range, and expressed as a design constraint (with a partial safety factor).

References

Verification

The International Energy Agency has sponsored a number of international research collaborations to further the state of wind energy technology and tools. One of these, Task 30: Offshore Code Comparison Collaboration (OC3), shared a spar design among many participants to compare performance as modeled by differed tool sets. A description of the OC3 spar is provided in [Jon10]. Since it is already a well-studied geometry, the OC3 spar design was selected as the focus of verification for FloatingSE. As part of the Task 30 effort, an ANSYS model of the OC3 spar, using shell elements combined with stiffeners and bulkheads, was also generated. This was taken as the truth standard for comparison.

Mass Properties

The first step in the verification exercise was to ensure that the mass properties of the spar predicted by FloatingSE matched those calculated by ANSYS. The comparison is shown in Table 5.14, where FloatingSE summary mass estimates are within 1% error of ANSYS. To ensure that these mass property calculations remain consistent over time despite changes in the underlying code, this OC3 mass properties comparison was installed within the FloatingSE test framework.

<table>
<thead>
<tr>
<th>Property</th>
<th>Units</th>
<th>ANSYS</th>
<th>WISDEM</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bulkhead mass</td>
<td>kg</td>
<td>67,682</td>
<td>65,951</td>
<td>-2.6%</td>
</tr>
<tr>
<td>Spar shell mass</td>
<td>kg</td>
<td>1,443,711</td>
<td>1,434,436</td>
<td>-0.6%</td>
</tr>
<tr>
<td>Stiffener mass</td>
<td>kg</td>
<td>77,585</td>
<td>78,252</td>
<td>0.9%</td>
</tr>
<tr>
<td>Total mass (no ballast)</td>
<td>kg</td>
<td>1,588,978</td>
<td>1,578,639</td>
<td>-0.7%</td>
</tr>
<tr>
<td>Center of gravity</td>
<td>m</td>
<td>(0.0,-58.93)</td>
<td>(0.0,-58.84)</td>
<td>-0.1%</td>
</tr>
<tr>
<td>Displaced volume</td>
<td>m$^3$</td>
<td>1,550</td>
<td>1,540</td>
<td>-0.7%</td>
</tr>
<tr>
<td>Ixx moment of inertia</td>
<td>kg $\cdot m^2$</td>
<td>2,178,400,000</td>
<td>2,186,163,760</td>
<td>0.4%</td>
</tr>
<tr>
<td>Iyy moment of inertia</td>
<td>kg $\cdot m^2$</td>
<td>2,178,400,000</td>
<td>2,186,163,760</td>
<td>0.4%</td>
</tr>
<tr>
<td>Izz moment of inertia</td>
<td>kg $\cdot m^2$</td>
<td>32,297,000</td>
<td>31,869,072</td>
<td>-1.3%</td>
</tr>
</tbody>
</table>

Static Loading Stress

Without wind and waves

With the mass property comparison showing little difference between ANSYS and FloatingSE calculations, the verification proceeded to static cases. The spar was simulated in quiescent air and water (no wind, waves, or current), which isolated the weight of the turbine and hydrostatic pressure forces as the only sources on the substructure. The effective von Mises stress, as calculated by FloatingSE, was compared to the ANSYS model. This comparison, as a function of the $z$-coordinate along the spar axis ($z = 0$ at the waterline), is shown in Fig. 5.51. The FloatingSE stress calculation...
matches that of ANSYS nearly exactly over the top half of the spar, but deviates by approximately 5–10% towards the bottom half of the spar. In the bottom half of the spar, FloatingSE actually over-predicts the stress, a more conservative estimate, which is the preferable approach in a low-fidelity cost and sizing model.

![Stress Comparison](image)

**Fig. 5.51:** Effective (von Mises) stress comparison between FloatingSE and WISDEM for a pure static loading case.

**With wind and waves**

At this time, more complicated loading cases, with wind and wave loading included, have not been performed.

**Hydrodynamic Verification**

The rigid body modes predicted by FloatingSE were compared against a FAST model of the OC3 spar. FAST was used as the truth solution in this case because it more accurately handles mooring dynamics than the ANSYS structural model and more accurately captures hydrodynamic phenomenon. The results are shown in Table 5.15. The errors in the surge, sway, roll, and pitch frequencies are 11-12%. FloatingSE actually estimates the heave mode frequency quite accurately, to less than 1% error, but is significantly off in estimating the yaw mode frequency. This was deemed acceptable as there is no focus on the yaw DOF in FloatingSE.

<table>
<thead>
<tr>
<th>Eigenmode</th>
<th>FAST</th>
<th>WISDEM</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Platform surge [Hz]</td>
<td>0.0080479</td>
<td>0.0071256</td>
<td>-11.46%</td>
</tr>
<tr>
<td>Platform sway [Hz]</td>
<td>0.0080475</td>
<td>0.0071256</td>
<td>-11.46%</td>
</tr>
<tr>
<td>Platform heave [Hz]</td>
<td>0.0324294</td>
<td>0.032235</td>
<td>-0.60%</td>
</tr>
<tr>
<td>Platform roll [Hz]</td>
<td>0.0342412</td>
<td>0.0385349</td>
<td>12.54%</td>
</tr>
<tr>
<td>Platform pitch [Hz]</td>
<td>0.0342602</td>
<td>0.0385349</td>
<td>12.48%</td>
</tr>
<tr>
<td>Platform yaw [Hz]</td>
<td>0.1210301</td>
<td>0.0526347</td>
<td>-56.51%</td>
</tr>
</tbody>
</table>
Execution

Executing *FloatingSE* requires additional inputs beyond those of the geometry definition described above in Section *Geometry*. Other user inputs for the metocean and loading environment, and the operational constraints, are required to evaluate the total mass, cost, and code compliance. These variables are also included in the WindIO effort or found in the floating-specific examples for standalone execution.

Simulation Flow

Once the input variables are completely specified, *FloatingSE* executes the analysis of the substructure. Conceptually, the simulation is organized by the flowchart in Fig. 5.52.

![Conceptual diagram of FloatingSE execution.](image)

**Fig. 5.52:** Conceptual diagram of *FloatingSE* execution.

From a more technical perspective, *FloatingSE* is an OpenMDAO Group, so the analysis sequence is broken down by the sub-groups and sub-components in the order that they are listed in Table [tbl:exec]. In an OpenMDAO group, sub-groups and components are given prefixes to aid in referring to specific variables. The prefixes used in *FloatingSE* are also listed in Table 5.16.
Table 5.16: Components and sub-groups within *FloatingSE*.

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1)</td>
<td>tow</td>
<td><em>TowerLeanSE</em> Discretization of tower geometry (but no analysis)</td>
</tr>
<tr>
<td>2)</td>
<td>main</td>
<td><em>Column</em> Discretization and API Bulletin 2U compliance of main vertical column</td>
</tr>
<tr>
<td>3)</td>
<td>off</td>
<td><em>Column</em> Discretization and API Bulletin 2U compliance of offset columns</td>
</tr>
<tr>
<td>4)</td>
<td>sg</td>
<td><em>SubstructureGeometry</em> Geometrical constraints on substructure</td>
</tr>
<tr>
<td>5)</td>
<td>mm</td>
<td><em>MapMooring</em> Mooring system analysis via pyMAP</td>
</tr>
<tr>
<td>6)</td>
<td>load</td>
<td><em>FloatingLoading</em> Structural analysis of complete floating turbine load path via pyFrame3DD</td>
</tr>
<tr>
<td>7)</td>
<td>subs</td>
<td><em>Substructure</em> Static stability and final mass and cost summation for generic substructure</td>
</tr>
</tbody>
</table>

Outputs are accumulated in each sub-group or component, and they either become inputs to other components, become constraints for optimization problems, become design variables for optimization problems, or can simply be ignored. Currently, a single execution of *FloatingSE* takes only a handful of seconds on a modern laptop computer.

**Examples**

As mentioned previously, *floating-specific examples* examples are provided. These files are encoded with default starting configurations (from [Jon10] and [RJM+14], respectively), with some modifications. There is an additional spar example that also has a ready configurations for optimization with design variables, constraints, and solvers options. A visualization of the geometries described by these examples is shown in Fig. 5.53 and Fig. 5.54.
Fig. 5.53: Spar example in FloatingSE taken from OC3 [Jon10] project.

Fig. 5.54: Semi example in FloatingSE taken from OC4 [RJM+14] project.
Optimization

Executing FloatingSE by hand is sufficient to explore some simple one-off or comparison analyses between a few runs. OpenMDAO provides extensive optimization capability, which can give yield richer and more insightful analyses.

Design Variables

In WISDEM, via OpenMDAO, any input parameter can be designated a design variable. The design variables used in this study focused on the geometric specification of the floating substructure and mooring subsystem. Slightly different design variables and bounds were used for spar, semisubmersible, and TLP optimizations. The complete listing of the design variables for each optimization configuration is shown in Table 5.17. Note that the integer design variables were only used in the global optimization with the genetic algorithm, not the local search with the simplex algorithm.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Name</th>
<th>Units</th>
<th>Type</th>
<th>Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main col section height</td>
<td>main_section_height</td>
<td>Float array (n_s)</td>
<td>0.1–50</td>
<td></td>
</tr>
<tr>
<td>Main col outer diameter</td>
<td>main_outer_diameter</td>
<td>Float array (n_s + 1)</td>
<td>2.1–40</td>
<td></td>
</tr>
<tr>
<td>Main col wall thickness</td>
<td>main_wall_thickness</td>
<td>Float array (n_s + 1)</td>
<td>0.001–0.6</td>
<td></td>
</tr>
<tr>
<td>Main col freeboard</td>
<td>main_freeboard</td>
<td>Float scalar</td>
<td>0–50</td>
<td></td>
</tr>
<tr>
<td>Main col stiffener web height</td>
<td>main_stiffener_web_height</td>
<td>Float array (n_s)</td>
<td>0.01–1</td>
<td></td>
</tr>
<tr>
<td>Main col stiffener flange width</td>
<td>main_stiffener_flange_width</td>
<td>Float array (n_s)</td>
<td>0.01–5</td>
<td></td>
</tr>
<tr>
<td>Main col stiffener flange thickness</td>
<td>main_stiffener_flange_thickness</td>
<td>Float array (n_s)</td>
<td>0.001–0.5</td>
<td></td>
</tr>
<tr>
<td>Main col stiffener spacing</td>
<td>main_stiffener_spacing</td>
<td>Float array (n_s)</td>
<td>0.1–100</td>
<td></td>
</tr>
<tr>
<td>Main col permanent ballast height</td>
<td>main_permanent_ballast_height</td>
<td>Float scalar</td>
<td>0.1–50</td>
<td></td>
</tr>
<tr>
<td>Main col buoyancy tank diameter</td>
<td>main_buoyancy_tank_diameter</td>
<td>Float scalar</td>
<td>0–50</td>
<td></td>
</tr>
<tr>
<td>Main col buoyancy tank height</td>
<td>main_buoyancy_tank_height</td>
<td>Float scalar</td>
<td>0–20</td>
<td></td>
</tr>
<tr>
<td>Main col buoyancy tank location (fraction)</td>
<td>main_buoyancy_tank_location</td>
<td>Float scalar</td>
<td>0–1</td>
<td></td>
</tr>
<tr>
<td>Number of offset cols</td>
<td>number_of_offset_columns</td>
<td>Integer scalar</td>
<td>3–5</td>
<td></td>
</tr>
<tr>
<td>Offset col section height</td>
<td>offset_section_height</td>
<td>Float array (n_s)</td>
<td>0.1–50</td>
<td></td>
</tr>
<tr>
<td>Offset col outer diameter</td>
<td>offset_outer_diameter</td>
<td>Float array (n_s + 1)</td>
<td>1.1–40</td>
<td></td>
</tr>
<tr>
<td>Offset col wall thickness</td>
<td>offset_wall_thickness</td>
<td>Float array (n_s + 1)</td>
<td>0.001–0.6</td>
<td></td>
</tr>
<tr>
<td>Offset col freeboard</td>
<td>offset_freeboard</td>
<td>Float scalar</td>
<td>2–15</td>
<td></td>
</tr>
<tr>
<td>Offset col stiffener web height</td>
<td>offset_stiffener_web_height</td>
<td>Float array (n_s)</td>
<td>0.01–1</td>
<td></td>
</tr>
<tr>
<td>Offset col stiffener flange width</td>
<td>offset_stiffener_flange_width</td>
<td>Float array (n_s)</td>
<td>0.01–5</td>
<td></td>
</tr>
<tr>
<td>Offset col stiffener flange thickness</td>
<td>offset_stiffener_flange_thickness</td>
<td>Float array (n_s)</td>
<td>0.001–0.5</td>
<td></td>
</tr>
<tr>
<td>Offset col stiffener spacing</td>
<td>offset_stiffener_spacing</td>
<td>Float array (n_s)</td>
<td>0.1–100</td>
<td></td>
</tr>
<tr>
<td>Offset col permanent ballast height</td>
<td>offset_permanent_ballast_height</td>
<td>Float scalar</td>
<td>0.1–50</td>
<td></td>
</tr>
<tr>
<td>Offset col buoyancy tank diameter</td>
<td>offset_buoyancy_tank_diameter</td>
<td>Float scalar</td>
<td>0–50</td>
<td></td>
</tr>
<tr>
<td>Offset col buoyancy tank height</td>
<td>offset_buoyancy_tank_height</td>
<td>Float scalar</td>
<td>0–20</td>
<td></td>
</tr>
<tr>
<td>Offset col buoyancy tank location (fraction)</td>
<td>offset_buoyancy_tank_location</td>
<td>Float scalar</td>
<td>0–1</td>
<td></td>
</tr>
<tr>
<td>Radius to offset col</td>
<td>radius_to_offset_column</td>
<td>Float scalar</td>
<td>5–100</td>
<td></td>
</tr>
<tr>
<td>Pontoon outer diameter</td>
<td>pontoon_outer_diameter</td>
<td>Float scalar</td>
<td>0.1–10</td>
<td></td>
</tr>
<tr>
<td>Pontoon wall thickness</td>
<td>pontoon_wall_thickness</td>
<td>Float scalar</td>
<td>0.01–1</td>
<td></td>
</tr>
<tr>
<td>Lower main-offset pontoons</td>
<td>lower_attachment_pontoons_int</td>
<td>Integer scalar</td>
<td>0–1</td>
<td></td>
</tr>
<tr>
<td>Upper main-offset pontoons</td>
<td>upper_attachment_pontoons_int</td>
<td>Integer scalar</td>
<td>0–1</td>
<td></td>
</tr>
<tr>
<td>Cross main-offset pontoons</td>
<td>cross_attachment_pontoons_int</td>
<td>Integer scalar</td>
<td>0–1</td>
<td></td>
</tr>
</tbody>
</table>

continues on next page
Table 5.17 – continued from previous page

<table>
<thead>
<tr>
<th>Variable</th>
<th>Name</th>
<th>Units</th>
<th>Type</th>
<th>Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lower offset ring pontoons</td>
<td>lower_ring_pontoons_int</td>
<td>Integer scalar</td>
<td>0–1</td>
<td></td>
</tr>
<tr>
<td>Upper offset ring pontoons</td>
<td>upper_ring_pontoons_int</td>
<td>Integer scalar</td>
<td>0–1</td>
<td></td>
</tr>
<tr>
<td>Outer V-pontoons</td>
<td>outer_cross_pontoons_int</td>
<td>Integer scalar</td>
<td>0–1</td>
<td></td>
</tr>
<tr>
<td>Main col pontoon attach lower (fraction)</td>
<td>main_pontoon_attach_lower</td>
<td>Float scalar</td>
<td>0–0.5</td>
<td></td>
</tr>
<tr>
<td>Main col pontoon attach upper (fraction)</td>
<td>main_pontoon_attach_upper</td>
<td>Float scalar</td>
<td>0.5–1</td>
<td></td>
</tr>
<tr>
<td>Fairlead (fraction)</td>
<td>fairlead_location</td>
<td>Float scalar</td>
<td>0–1</td>
<td></td>
</tr>
<tr>
<td>Fairlead offset from col</td>
<td>fairlead_offset_from_shell</td>
<td>Float scalar</td>
<td>5–30</td>
<td></td>
</tr>
<tr>
<td>Fairlead pontoon diameter</td>
<td>fairlead_support_outer_diameter</td>
<td>Float scalar</td>
<td>0.1–10</td>
<td></td>
</tr>
<tr>
<td>Fairlead pontoon wall thickness</td>
<td>fairlead_support_outer_thickness</td>
<td>Float scalar</td>
<td>0.001–1</td>
<td></td>
</tr>
<tr>
<td>Number of mooring connections</td>
<td>number_of_mooring_connections</td>
<td>Integer scalar</td>
<td>3–5</td>
<td></td>
</tr>
<tr>
<td>Mooring lines per connection</td>
<td>mooring_lines_per_connection</td>
<td>Integer scalar</td>
<td>1–3</td>
<td></td>
</tr>
<tr>
<td>Mooring diameter</td>
<td>mooring_diameter</td>
<td>Float scalar</td>
<td>0.05–2</td>
<td></td>
</tr>
<tr>
<td>Mooring line length</td>
<td>mooring_line_length</td>
<td>Float scalar</td>
<td>0–3000</td>
<td></td>
</tr>
<tr>
<td>Anchor distance</td>
<td>anchor_radius</td>
<td>Float scalar</td>
<td>0–5000</td>
<td></td>
</tr>
</tbody>
</table>

Constraints

Due to the many design variables, permutations of settings, and applied physics, there are many constraints that must be applied for an optimization to close. The constraints capture both physical limitations, such as column buckling, but also inject industry standards, guidelines, and lessons learned from engineering experience into the optimization. As described in the Introduction, this is a critically important element in building a MDAO framework for conceptual design that yields feasible results worth interrogating further with higher-fidelity tools. The constraints used in the substructure design optimization and sensitivity studies are listed in Table 5.18. Where appropriate, some of the constraint values differ from one type of substructure to another. Some additional explanation is provided for a handful of constraints in the subsections below.

Table 5.18: Optimization constraints used in FloatingSE.

<table>
<thead>
<tr>
<th>Lower</th>
<th>Variable</th>
<th>Upper</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tower / Main / Offset Columns</td>
<td>Eurocode global buckling</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Eurocode shell buckling</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Eurocode stress limit</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Manufacturability</td>
<td>0.5</td>
<td>Taper ratio limit</td>
</tr>
<tr>
<td>120.0</td>
<td>Weld-ability</td>
<td>1.0</td>
<td>Diameter:thickness ratio limit</td>
</tr>
<tr>
<td>Main / Offset Columns</td>
<td>Draft ratio</td>
<td>1.0</td>
<td>Ratio of draft to max value</td>
</tr>
<tr>
<td></td>
<td>API 2U general buckling- axial loads</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>API 2U local buckling- axial loads</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>API 2U general buckling- external loads</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>API 2U local buckling- external loads</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Wave height:freeboard ratio</td>
<td>1.0</td>
<td>Maximum wave height relative to freeboard</td>
</tr>
<tr>
<td>1.0</td>
<td>Stiffener flange compactness</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.0</td>
<td>Stiffener web compactness</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Stiffener flange spacing ratio</td>
<td>1.0</td>
<td>Stiffener spacing relative to flange width</td>
</tr>
<tr>
<td></td>
<td>Stiffener radius ratio</td>
<td>0.50</td>
<td>Stiffener height relative to diameter</td>
</tr>
<tr>
<td>Offset Columns</td>
<td>Semi only</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.0</td>
<td>Heel freeboard margin</td>
<td>Height required to stay above waterline at max heel</td>
<td></td>
</tr>
<tr>
<td>0.0</td>
<td>Heel draft margin</td>
<td>Draft required to stay submerged at max heel</td>
<td></td>
</tr>
</tbody>
</table>

continues on next page
Table 5.18 – continued from previous page

<table>
<thead>
<tr>
<th>Lower</th>
<th>Variable</th>
<th>Upper</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Pontoons</td>
<td></td>
<td><em>Semi only</em></td>
</tr>
<tr>
<td></td>
<td>Eurocode stress limit</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Tower</td>
<td>-0.01</td>
<td>Hub height error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.01</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mooring</td>
<td>0.0</td>
<td>Axial stress limit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.0</td>
<td>Loss of tension or catenary hang</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.0</td>
<td>Ratio of overturning moment to restoring moment</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.0</td>
<td>Ratio of surge force to restoring force</td>
</tr>
<tr>
<td></td>
<td>Geometry</td>
<td>1.0</td>
<td>Main-offset spacing</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Minimum spacing between main and offset columns</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.0</td>
<td>Nacelle transition buffer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.0</td>
<td>Tower diameter limit at nacelle junction</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-1.0</td>
<td>Tower transition buffer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.0</td>
<td>Diameter consistency at freeboard point</td>
</tr>
<tr>
<td></td>
<td>Stability</td>
<td>0.10</td>
<td>Metacentric height</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Not applied to TLPs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.0</td>
<td>Wave-Eigenmode boundary (upper)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Natural frequencies below wave frequency range</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.0</td>
<td>Wave-Eigenmode boundary (lower)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Natural frequencies above wave frequency range</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.0</td>
<td>Water ballast height limit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.0</td>
<td>Water ballast mass</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Neutral buoyancy</td>
</tr>
</tbody>
</table>

**Geometry Constraints**

Words Table 5.19

Table 5.19: Constraint variables for the geometry in FloatingSE.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>max_draft</td>
<td>Float scalar</td>
<td>Maximum allowable draft for the substructure</td>
</tr>
</tbody>
</table>

**Manufacturing Constraints**

Manufacturing steel frustum shells requires rolling steel plates into shape and welding along a seam to close the section. To accommodate traditional rolling and welding practices, both the diameter taper over the course of a section and the wall thickness ratio relative to the diameter are capped. Similarly, to facilitate welding the semisubmersible pontoons to the columns, constraints regarding the ratio of diameters between the two are enforced. These limits are determined by user parameters in Table 5.20 and constraints,

Table 5.20: Constraint variables for the manufacturability in FloatingSE.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>min_taper_ratio</td>
<td>Float scalar</td>
<td>For manufacturability of rolling steel</td>
</tr>
<tr>
<td>min_diameter_thickness_ratio</td>
<td>Float scalar</td>
<td>For weld-ability</td>
</tr>
<tr>
<td>connection_ratio_max</td>
<td>Float scalar</td>
<td>For welding pontoons to columns</td>
</tr>
</tbody>
</table>
Stress Limits and Code Compliance

The stress and buckling code compliance constraints are formulated as utilization ratios (ratio of actual to maximum values), with a safety factor, which must be less than one. The safety factor parameters are listed in Table 5.21.

Table 5.21: Variables specifying the factors of safety within FloatingSE.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gamma_f</td>
<td>Float scalar</td>
<td>Safety factor on</td>
</tr>
<tr>
<td>gamma_b</td>
<td>Float scalar</td>
<td>Safety factor on buckling</td>
</tr>
<tr>
<td>gamma_m</td>
<td>Float scalar</td>
<td>Safety factor on materials</td>
</tr>
<tr>
<td>gamma_n</td>
<td>Float scalar</td>
<td>Safety factor on consequence of failure</td>
</tr>
<tr>
<td>gamma_fatigue</td>
<td>Float scalar</td>
<td>Not currently used</td>
</tr>
</tbody>
</table>

Stability

As described above, surge and pitch stability are enforced through similar approaches. The total force and moment acting on the turbine are compared to the restoring forces and moments applied by the mooring system, buoyancy, or other sources at the maximum allowable point of displacement. These constraints are formulated as ratios with the user specifying the maximum allowable limits via the variables in Table 5.22.

Table 5.22: Constraint variables for the mooring system in FloatingSE.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>max_offset</td>
<td>Float scalar</td>
<td>m</td>
<td>Max surge/sway offset</td>
</tr>
<tr>
<td>operational_heel</td>
<td>Float scalar</td>
<td>deg</td>
<td>Max heel (pitching) angle in operating conditions</td>
</tr>
<tr>
<td>max_survival_heel</td>
<td>Float scalar</td>
<td>deg</td>
<td>Max heel (pitching) angle in parked conditions</td>
</tr>
</tbody>
</table>

Objectives

Different analyses will emphasize different metrics, requiring different objective functions. Under the default assumption that the user wishes to minimize cost and adhere to stability constraints, the objective function would be total substructure cost (variable name, total_cost) or mass (variable name, total_mass).

Example

Documentation

The class structure for all the modules is listed below.

**FloatingSE High-Level Group**

class wisdom.floatingse.floating.FloatingSE
Fig. 5.55: Example of optimized spar.

Fig. 5.56: Example of optimized semi.

Fig. 5.57: Example of optimized TLP.
FloatingSE Vertical, Submerged Column of Frustums

class wisdem.floatingse.column.DiscretizationYAML
class wisdem.floatingse.column.ColumnGeometry
class wisdem.floatingse.column.BulkheadMass
class wisdem.floatingse.column.BuoyancyTankProperties
class wisdem.floatingse.column.StiffenerMass
class wisdem.floatingse.column.ColumnProperties
class wisdem.floatingse.column.ColumnBuckling
class wisdem.floatingse.column.Column

FloatingSE Structural Analysis

class wisdem.floatingse.loading.FloatingFrame
class wisdem.floatingse.loading.TrussIntegerToBoolean
class wisdem.floatingse.loading>Loading

FloatingSE Mooring Analysis

class wisdem.floatingse.map_mooring.MapMooring

FloatingSE Stability Analysis

class wisdem.floatingse.substructure.SubstructureGeometry
class wisdem.floatingse.substructure.Substructure

FloatingSE Visualization

class wisdem.floatingse.visualize.Visualize

5.8.6 glue_code.py

openmdao n2 runWISDEM.py

The result will look like this image, which you can dynamically navigate folding and unfolding each block with the commands on the left of the screen.

Introduction

WISDEM is made of several components all wrapped by an OpenMDAO interface. glue_code.py is the code wrapping the modules together.
5.8.7 LandBOSSE

Intro

Capital costs associated with building land-based wind power plants

Capital costs associated with constructing a land-based wind power plant include turbine capital costs, transportation cost, and balance-of-system cost. They are outlined in the table below:

<table>
<thead>
<tr>
<th>Capital Cost</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Turbine capital cost (TCC)</td>
<td>The cost to purchase the turbine components</td>
</tr>
<tr>
<td>Transportation cost</td>
<td>The cost to transport turbine components to the construction site</td>
</tr>
<tr>
<td>Balance of system (BOS)</td>
<td>• Erection cost of towers, nacelles, and rotors with cranes</td>
</tr>
<tr>
<td></td>
<td>• Foundation construction cost</td>
</tr>
<tr>
<td></td>
<td>• Collection system installation cost</td>
</tr>
<tr>
<td></td>
<td>• Site preparation cost</td>
</tr>
<tr>
<td></td>
<td>• Management costs incurred during construction</td>
</tr>
<tr>
<td></td>
<td>• Development costs incurred before construction</td>
</tr>
<tr>
<td></td>
<td>• Substation cost</td>
</tr>
<tr>
<td></td>
<td>• Grid connection cost</td>
</tr>
</tbody>
</table>
What capital costs does LandBOSSE model?

The Land-based Balance-of-System Systems Engineering (LandBOSSE) model is a systems engineering tool that estimates the balance-of-system (BOS) costs associated with installing utility scale land-based wind plants (10, 1.5 MW turbines or larger). The methods used to develop this model and a detailed discussion of its inputs and outputs are in the following report.


Documentation

LandBOSSE and OpenMDAO

In WISDEM, LandBOSSE is presented as an OpenMDAO Group called LandBOSSE. This group wraps an ExplicitComponent called LandBOSSE_API. When using LandBOSSE in assemblies, LandBOSSE should be accessed via the LandBOSSE group.

LandBOSSE Inputs and Outputs

LandBOSSE models the construction of an entire wind plant, which itself is a project with a numerous operations. The number and diversity of inputs and outputs to and from the LandBOSSE component reflects this diversity of construction operations. Text and numeric data, both in single-value and tabular form, are the form of LandBOSSE inputs and outputs. A full listing of every input and output is beyond the scope of this document; however, here is an overview of types of inputs and outputs and what they contain.

Inputs are grouped into three categories:

| Category                                | Examples                                                        |
|-----------------------------------------|                                                                |
| Continuous floating point numeric values| blade_mass, tower_mass, crane_breakdown_fraction                |
| Discrete integer numeric values          | rate_of_deliveries, number_of_blades, num_turbines              |
| Discrete Pandas dataframes               | crane_specs, components, weather_window,                        |

Similarly, outputs are grouped into two categories:

<table>
<thead>
<tr>
<th>Category</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Continuous numeric data</td>
<td>bos_capex_kW, total_capex_kW, installation_time_months</td>
</tr>
<tr>
<td>Discrete Pandas dataframes</td>
<td>landbosse_costs_by_module_type_operation, erection_components</td>
</tr>
</tbody>
</table>

.xlsx spreadsheet

Ultimately, all of these values are inputs into the LandBOSSE group. The dataframe inputs are read from .xlsx spreadsheet with the following worksheets: crane_specs, cable_specs, equip, components, development, crew_price, crew, equip_price, material_price, rsmeans, site_facility_building_area, and weather_window. These sheets are used as lookup tables for capabilities and costs of equipment and crews utilized in the BOS operations.

A file with default data is in the library/landbosse/ge15_public.xlsx file found in the WISDEM repository.
Tutorial

Common use case: A wind plant made of an optimized turbine

A common use case of LandBOSSE within WISDEM is to model the costs of building an entire wind power plant from turbine components generated by RotorSE, DriveSE, and TowerSE. For this use case, the following table lists the needed inputs for calculating BOS costs for a turbine created by WISDEM:

<table>
<thead>
<tr>
<th>Input</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hub_height</td>
<td>m</td>
<td>Hub height of the rotor</td>
</tr>
<tr>
<td>blade_mass</td>
<td>kg</td>
<td>Mass of one blade</td>
</tr>
<tr>
<td>nacelle_mass</td>
<td>kg</td>
<td>Mass of the nacelle</td>
</tr>
<tr>
<td>tower_mass</td>
<td>kg</td>
<td>Mass of the tower</td>
</tr>
<tr>
<td>machine_rating</td>
<td>kW</td>
<td>Rating of the turbine</td>
</tr>
</tbody>
</table>

Troubleshooting note: During some practical construction operations, large nacelles (such as those found in high turbine ratings) are broken into multiple sections to reduce the size of the crane needed for erection. However, DriveSE models a nacelle all as one piece. If you get an error where LandBOSSE reports that a topping crane could not be found, this may mean the calculated nacelle mass exceeds the capacity of the largest crane listed in the crane_specs tab of the spreadsheet.

An example assembly that integrates calculations from RotorSE, DriveSE, TowerSE, and LandBOSSE is in the wisdem/assemblies/land_based.py file.

Theory

Modules

Costs calculated in LandBOSSE are in eight modules. Four modules (erection, foundation, site preparation, and collection) are process based and calculate costs based on models of physical processes that happen to accomplish the scope of work defined for each module. The other four modules (management, grid connection, substation, and development) are based on curve fits from empirical data. The operations modeled by each model are summarized in Table XX below. More details are in the technical report at https://www.nrel.gov/docs/fy19osti/72201.pdf.

Table XX: [TODO: Update citation when paper is published]
### Module Type | Summary of costs included
--- | ---
Foundation process | Operations specific to foundation construction, including excavating the base, installing rebar and a bolt cage, pouring concrete, constructing the pedestal, and backfilling the foundation.
Erection process | Operations specific to erecting the tower and turbine, including removal of components from delivery trucks by offload cranes and erection of the lower tower sections onto the foundation using a base crane and the upper pieces of the tower and the components of the nacelle using a topping crane.
Development curve fit | Evaluation of the wind resource, acquisition of land, completion of environmental permitting, assessment of distribution costs, and marketing of power to be generated.
Management curve fit | Insurance, construction permits, site-specific engineering, construction of facilities for site access and construction staging, site management, and bonding, markup, and contingency.
Collection process | Operations specific to the construction of a collection system, which consists of cabling from the turbines to the substation (does not include power electronics or cabling already included in the turbine capital cost).
Grid Connection curve fit | Operations specific to grid connection (i.e., transmission and interconnection), including a land survey, clearing and grubbing the area, installation of stormwater and pollution mitigation measures, installation of conductors, and restoration of the rights of way.
Site Preparation process | Operations to prepare the wind plant site for other construction operations, including surveying and clearing areas for roads, compacting the soil, and placing rock to allow roads to support the weight of trucks, components, and cranes.
Substation process | Operations specific to substation construction, including a land survey; installation of stormwater and pollution mitigation measures; construction of dead-end structures, foundations, conductors, transformers, relays, controls, and breakers; and restoration of the rights of way.

### 5.8.8 MoorPy

MoorPy is a design-oriented mooring system library for Python based around a quasi-static modeling approach. It is in the early stages of development and does not yet have a documentation available online. It’s capabilities and approach are similar to the pyMAP module that was previously used in WISDEM, but is easier to install and extend with new features. MoorPy is available as a standalone analysis tool from its home GitHub repository.

### 5.8.9 NREL CSM

**Introduction**

The two models contained in this Turbine_CostsSE software allow for the determination of mass and costs for major wind turbine components (rotor, hub, drivetrain, tower, etc.) leading to overall turbine capital costs. The first model, contained in the `nrel_csm_tcc_2015.py` file, estimates turbine component masses from high-level inputs such as machine rating, rotor diameter, and number of turbine blades. The second model, contained in the `turbine_costsse_2015.py` file, estimates component costs given input mass values.

The mass and cost scaling relationships included in these two files were compiled in 2015 as an update to the NREL Cost and Scaling Model (CSM). This model dates back to 2006 [FHL05] and itself drew on prior work in the Wind Partnerships for Advanced Component Technology (WindPACT) project that occurred between roughly 2002 to 2005, and the University of Sunderland model [HJ93]. This 2015 refresh only focused on turbine capital costs, however, so
the prior approximations for balance of station cost, operational and maintenance costs, and annual energy production can still be found in the nrelcsm module that contains the original code.

All mass and cost estimates in this module use simple regressions. However, all component mass and cost values can be overridden from external sources. Full turbine analysis in WISDEM take full advantage of this feature as the component masses estimated through design optimization are considered to be much more accurate than the simple regression. Furthermore, some other WISDEM modules also include detailed, “bottoms-up” cost estimates that are much more accurate than the scaling relationships used here. Nevertheless, the modular design allows the user to override some of the component mass and cost values with better estimates, but leave others in place where less accuracy is acceptable.

Theory

The theory for the models in this software are based directly on the work described in references [FHL05][HJ93][MH06][MHM10]. This section provides an overview of these simple mass and cost models for the major turbine components.

The NREL Cost and Scaling Model [FHL05] provides a simple cost and sizing tool to estimate wind turbine component masses and costs based on a small number of input parameters such as rotor diameter, hub height and rated power. The model was developed over several results following the Wind Partnerships for Advanced Component Technology (WindPACT) work that occurred between roughly 2002 to 2005. The original form of the cost model was based on an earlier model from 1993 out of the University of Sunderland (the Sunderland Model) [HJ93]. The Sunderland Model created a set of wind turbine models to estimate the mass and cost of all major wind turbine components including: blade, hub system [hub, pitch system, and nose cone], nacelle [low speed shaft, main bearings, gearbox, high speed shaft/mechanical brake, generator, variable speed electronics, electrical cabling, mainframe [bedplate, platforms and railings, base hardware, and crane], HVAC system, controls, and nacelle cover], and tower. The Sunderland model was based on a set of semi-empirical models for each component which estimated design loads at the rotor, propagated these loads through the entire system, and used the loads to estimate the size of each component calibrated to data on actual turbines in the field during that time. Cost estimates were then made on a per weight basis using a multiplier again based on field data or industry sources.

To arrive at the NREL Cost and Scaling Model, the WindPACT studies began in many cases with the Sunderland model and updated the results with new coefficients or, in some cases, with entirely new cost equations based on curve fits of key design parameters (rotor diameter, etc) to the results of detailed design studies [MH06]. In addition, the WindPACT work established estimates of costs associated with balance of station and operations and maintenance for a fictitious wind plant in North Dakota which led to an overall cost of energy model for a wind plant. The key cost of energy equation for a wind plant is given in the NREL Cost and Scaling Model [FHL05] as:

\[
\text{COE} = \frac{(FCR \times (BOS + TCC))/AEP + (LLC + LRC + (1 - tr) \times OM)}{AEP}
\]

where \(\text{COE}\) in this equation is a simple estimate of a wind plant cost of energy, \(FCR\) is the fixed charge rate for the project, \(BOS\) are the total balance of station costs for the project, \(TCC\) are the total turbine capital costs for the project, \(AEP\) is the annual energy production for the project, \(LLC\) are the annual land-lease costs, \(LRC\) is the levelized replacement cost for major part replacement, \(tr\) is the tax rate, and \(OM\) are the annual operations and maintenance costs which are tax deductible.

While the NREL Cost and Scaling Model improved the overall cost estimation for larger turbines on the order of 1 MW+, it abstracted away from the engineering analysis foundations of the original Sunderland model. This is depicted in the below figure where it can be seen that the engineering-analysis has been replaced by a series of curve fits which relate a small number of design parameters to mass and cost estimates for major wind turbine components.

The resulting NREL Cost and Scaling Model (as provided in NREL_CSM_TCC) allows for a variety of interesting analyses including scaling of conventional technology from under a MW to 5 MW+, assessing impact of trends in input factors for materials and labor on wind plant cost of energy, etc. However, it does not preserve the underlying engineering relationships of the original Sunderland model and thus loses some fidelity of assessing how design changes may impact system costs.
Fig. 5.59: NREL Cost and Scaling Model Key Input-Output Relationships. TODO: REFRESH GRAPHIC
The goal of the development of the second model, Turbine_CostsSE, then is to provide a set of mass-based component cost calculations. A mass-cost model is developed for each of the major turbine components. These use the data underlying the NREL Cost and Scaling Model to estimate relationships that can then be scaled based on economic multipliers as done in [FHL05]. Details of the models are described next.

**TODO**

- The equation for blade costs includes both materials and manufacturing.

**Blades**

To obtain the blade mass in kilograms and cost in USD from the rotor diameter in meters,

\[
m_{\text{blade}} = k_m (0.5 D_{\text{rotor}})^b \\
c_{\text{blade}} = k_c m_{\text{blade}}
\]

\(k_m = 0.5\)

\(b = (\text{see below})\)

\(k_c = 14.6\)

Where \(D_{\text{rotor}}\) is the rotor diameter and \(b\) is determined by:

- If turbine class I and blade DOES have carbon fiber spar caps, \(b = 2.47\)
- If turbine class I and blade DOES NOT have carbon fiber spar caps, \(b = 2.54\)
- If turbine class II+ and blade DOES have carbon fiber spar caps, \(b = 2.44\)
- If turbine class II+ and blade DOES NOT have carbon fiber spar caps, \(b = 2.50\)
- User override of exponent value

For variable names access to override the default values see the **Source Documentation**.

The mass scaling relationships are based on the following data,
Hub (Shell)

To obtain the hub shell mass in kilograms and cost in USD from the blade mass in kilograms,

\[ m_{\text{hub}} = k_m m_{\text{blade}} + b \]
\[ c_{\text{hub}} = k_c m_{\text{hub}} \]

\[ k_m = 2.3 \]
\[ b = 1320 \]
\[ k_c = 3.9 \]

For variable names access to override the default values see the Source Documentation.

The mass scaling relationships are based on the following data,

![Hub Mass vs Blade Mass graph]

Pitch System

To obtain the pitch bearing and system mass in kilograms and cost in USD from the blade mass in kilograms,

\[ m_{\text{bearing}} = n_{\text{blade}} k_m m_{\text{blade}} + b_1 \]
\[ m_{\text{pitch}} = m_{\text{bearing}} (1 + h) + b_2 \]
\[ c_{\text{pitch}} = k_c m_{\text{pitch}} \]

\[ k_m = 0.1295 \]
\[ b_1 = 491.31 \]
\[ b_2 = 555 \]
\[ h = 0.328 \]
\[ k_c = 22.1 \]

Where \( n_{\text{blade}} \) is the number of blades, \( h \) is fractional mass of the pitch bearing housing.

For variable names access to override the default values see the Source Documentation.
**Spinner (Nose Cone)**

To obtain the spinner (nose cone) mass in kilograms and cost in USD from the rotor diameter in meters,

\[ m_{\text{spin}} = k_m D_{\text{rotor}} + b \]
\[ c_{\text{spin}} = k_c m_{\text{spin}} \]

\[ k_m = 15.5 \]
\[ b = -980 \]
\[ k_c = 11.1 \]

For variable names access to override the default values see the *Source Documentation*.

The mass scaling relationships are based on the following data,

![Graph showing the relationship between nose cone mass and rotor diameter](image)

**Low Speed Shaft**

To obtain the low speed shaft mass in kilograms and cost in USD from the blade mass in kilograms and the machine rating in megawatts,

\[ m_{\text{lss}} = k_m (m_{\text{blade}} P_{\text{turbine}})^{b_1} + b_2 \]
\[ c_{\text{lss}} = k_c m_{\text{lss}} \]

\[ k_m = 13 \]
\[ b_1 = 0.65 \]
\[ b_2 = 775 \]
\[ k_c = 11.9 \]

Where \( P_{\text{turbine}} \) is the machine rating.

For variable names access to override the default values see the *Source Documentation*.

The mass scaling relationships are based on the following data,
Main Bearings

To obtain the main bearings mass in kilograms and cost in USD from the rotor diameter in meters,

\[ m_{\text{bearing}} = n_{\text{bearing}} k_m D_{\text{rotor}}^b \]
\[ c_{\text{bearing}} = k_c m_{\text{bearing}} \]
\[ k_m = 0.0001 \]
\[ b = 3.5 \]
\[ k_c = 4.5 \]

Where \( D_{\text{rotor}} \) is the rotor diameter and \( n_{\text{bearing}} \) is the number of bearings.

For variable names access to override the default values see the Source Documentation.

The mass scaling relationships are based on the following data,
Gearbox

To obtain the main bearings mass in kilograms and cost in USD from the rotor torque in kilo-Newton meters,

\[
m_{\text{gearbox}} = k_m Q_{\text{rotor}}^{b} \\
c_{\text{gearbox}} = k_c m_{\text{gearbox}} \\
k_m = 113 \\
b = 0.71 \\
k_c = 12.9
\]

Where \(Q_{\text{rotor}}\) is the rotor torque and is approximated by,

\[
Q_{\text{rotor}} = \frac{0.5 P_{\text{turbine}} D_{\text{rotor}}}{\eta V_{\text{tip}}}
\]

Where \(P_{\text{turbine}}\) is the machine rating, \(D_{\text{rotor}}\) is the rotor diameter, \(V_{\text{tip}}\) is the max tip speed, and \(\eta\) is the drivetrain efficiency.

For variable names access to override the default values see the Source Documentation.

The mass scaling relationships are based on the following data,

![Gearbox Mass vs Rotor Torque graph](image)

Brake

To obtain the brake mass in kilograms and cost in USD from the rotor torque in kilo-Newton meters (updated in 2020 by J. Keller)),

\[
m_{\text{brake}} = k_m Q_{\text{rotor}} \\
c_{\text{brake}} = k_c m_{\text{brake}} \\
k_m = 1.22 \\
k_c = 3.6254
\]
Where $Q_{rotor}$ is the rotor torque and is approximated above.

For variable names access to override the default values see the Source Documentation.

**High Speed Shaft**

To obtain the high speed shaft mass in kilograms and cost in USD from the machine rating in megawatts,

$$m_{hss} = k_m P_{turbine}$$
$$c_{hss} = k_c m_{hss}$$

$k_m = 198.94$

$k_c = 6.8$

Where $P_{turbine}$ is the machine rating.

For variable names access to override the default values see the Source Documentation.

**Generator**

To obtain the generator mass in kilograms and cost in USD from the machine rating in megawatts,

$$m_{generator} = k_m P_{turbine} + b$$
$$c_{generator} = k_c m_{generator}$$

$k_m = 2300$

$b = 3400$

$k_c = 12.4$

Where $P_{turbine}$ is the machine rating.

For variable names access to override the default values see the Source Documentation.

The mass scaling relationships are based on the following data,


**Yaw System**

To obtain the yaw system mass in kilograms and cost in USD from the rotor diameter in meters,

\[ m_{\text{yaw}} = k_m D_{\text{rotor}}^b \]

\[ c_{\text{yaw}} = k_c m_{\text{yaw}} \]

\[ k_m = 0.00135 \]

\[ b = 3.314 \]

\[ k_c = 8.3 \]

Where \( D_{\text{rotor}} \) is the rotor diameter.

For variable names access to override the default values see the *Source Documentation*.

**Hydraulic Cooling**

To obtain the hydraulic cooling mass in kilograms and cost in USD from the machine rating in megawatts,

\[ m_{\text{hvac}} = k_m P_{\text{turbine}} \]

\[ c_{\text{hvac}} = k_c m_{\text{hvac}} \]

\[ k_m = 80 \]

\[ k_c = 124 \]

Where \( P_{\text{turbine}} \) is the machine rating.

For variable names access to override the default values see the *Source Documentation*.

**Transformer**

To obtain the transformer mass in kilograms and cost in USD from the machine rating in megawatts,

\[ m_{\text{transformer}} = k_m P_{\text{rotor}} + b \]

\[ c_{\text{transformer}} = k_c m_{\text{transformer}} \]

\[ k_m = 1915 \]

\[ b = 1910 \]

\[ k_c = 18.8 \]

For variable names access to override the default values see the *Source Documentation*.

The mass scaling relationships are based on the following data,

**Cabling and Electrical Connections**

To obtain the cabling and electrical connections cost in USD (there is no mass calculated) from the machine rating in megawatts,

\[ c_{\text{connect}} = k_c P_{\text{rotor}} \]

\[ k_c = 41850 \]

Where \( P_{\text{turbine}} \) is the machine rating.

For variable names access to override the default values see the *Source Documentation*. 
Control System

To obtain the control system cost in USD (there is no mass calculated) from the machine rating in megawatts,

\[ c_{\text{control}} = k_c P_{\text{rotor}} \]
\[ k_c = 21150 \]

Where \( P_{\text{turbine}} \) is the machine rating.

For variable names access to override the default values see the Source Documentation.

Other Nacelle Equipment

To obtain the nacelle platform and service crane mass in kilograms and cost in USD from the bedplate mass in kilograms,

\[ m_{\text{platform}} = k_m m_{\text{bedplate}} \]
\[ c_{\text{platform}} = k_c m_{\text{platform}} \]
\[ m_{\text{crane}} = 3000 \]
\[ c_{\text{crane}} = 12000 \]
\[ k_m = 0.125 \]
\[ k_c = 17.1 \]

Note that the service crane is optional with a flag set by the user.

For variable names access to override the default values see the Source Documentation.

Bedplate

To obtain the bedplate mass in kilograms and cost in USD from the rotor diameter in meters,

\[ m_{\text{bedplate}} = D_{\text{rotor}}^b \]
\[ c_{\text{bedplate}} = k_c m_{\text{bedplate}} \]
\[ b = 2.2 \]
\[ k_c = 2.9 \]

Where \( D_{\text{rotor}} \) is the rotor diameter. The mass scaling relationships are based on the following data.

For variable names access to override the default values see the Source Documentation.
Nacelle Cover

To obtain the nacelle cover mass in kilograms and cost in USD from the machine rating in megawatts,

\[ m_{\text{cover}} = k_m P_{\text{turbine}} + b \]
\[ c_{\text{cover}} = k_c m_{\text{cover}} \]
\[ k_m = 1.2817 \]
\[ b = 428.19 \]
\[ k_c = 5.7 \]

Where \( P_{\text{turbine}} \) is the machine rating.

For variable names access to override the default values see the Source Documentation.

Tower

To obtain the tower mass in kilograms and cost in USD from the hub height in meters,

\[ m_{\text{tower}} = k_m L_{\text{hub}}^b \]
\[ c_{\text{tower}} = k_c m_{\text{tower}} \]
\[ k_m = 19.828 \]
\[ b = 2.0282 \]
\[ k_c = 2.9 \]

Where \( L_{\text{hub}} \) is the hub height.

For variable names access to override the default values see the Source Documentation.

The mass scaling relationships are based on the following data,
Sub-System Aggregations

There are further aggregations of the components into sub-systems, at which point additional costs and/or multipliers are included. For the mass accounting, this includes hub system mass, rotor mass, nacelle mass, and total turbine mass,

**Hub System**

It is assumed that the hub system is assembled and transported as a one unit, thus there are additional costs at this level of aggregation,

\[
m_{\text{hubsy}} = m_{\text{hub}} + m_{\text{pitch}} + m_{\text{spinner}}
\]

\[
c_{\text{hubsy}} = (1 + k_t m_{\text{hubsy}} + k_p m_{\text{hubsy}})(1 + k_o m_{\text{hubsy}} + k_a m_{\text{hubsy}})(c_{\text{hub}} + c_{\text{pitch}} + c_{\text{spinner}})
\]

Where conceptually, \( k_t \) is a transportation multiplier, \( k_p \) is a profit multiplier, \( k_o \) is an overhead cost multiplier, and \( k_a \) is an assembly cost multiplier. By default, \( k_t = k_p = k_o = k_a = 0 \).

For variable names access to override the default values see the *Source Documentation*.

**Rotor System**

The rotor mass and cost is aggregated for conceptual convenience, but it is assumed to be transported in separate pieces and assembled on-site, so there are no separate sub-system cost multipliers.

\[
m_{\text{rotor}} = n_{\text{blade}} m_{\text{blade}} + m_{\text{hubsy}}
\]

\[
c_{\text{rotor}} = n_{\text{blade}} c_{\text{blade}} + c_{\text{hubsy}}
\]

For variable names access to override the default values see the *Source Documentation*. 
Nacelle

It is assumed that the nacelle and all of its sub-components are assembled and transported as a one unit, thus there are additional costs at this level of aggregation,

\[
m_{\text{nacelle}} = m_{\text{ls}} + m_{\text{bearing}} + m_{\text{gearbox}} + m_{\text{hss}} + m_{\text{generator}} + m_{\text{bedplate}} + m_{\text{yaw}} + m_{\text{hvac}} + m_{\text{transformer}} + m_{\text{platform}} + m_{\text{cover}}
\]

\[
c_{\text{parts}} = c_{\text{ls}} + c_{\text{bearing}} + c_{\text{gearbox}} + c_{\text{hss}} + c_{\text{generator}} + c_{\text{bedplate}} + c_{\text{yaw}} + c_{\text{hvac}} + c_{\text{transformer}} + c_{\text{connect}} + c_{\text{control}} + c_{\text{platform}} + c_{\text{cover}}
\]

\[
c_{\text{nacelle}} = (1 + k_{t\text{nacelle}} + k_{p\text{nacelle}})(1 + k_{o\text{nacelle}} + k_{a\text{nacelle}})c_{\text{parts}}
\]

Where conceptually, \(k_t\) is a transportation multiplier, \(k_p\) is a profit multiplier, \(k_o\) is an overhead cost multiplier, and \(k_a\) is an assembly cost multiplier. By default, \(k_t = k_p = k_o = k_a = 0\).

For variable names access to override the default values see the \textit{Source Documentation}.

Tower System

The tower is not aggregated with any other component, but for consistency there are allowances for additional costs incurred from transportation and assembly complexity,

\[
c_{\text{towersys}} = (1 + k_{t\text{tower}} + k_{p\text{tower}})(1 + k_{o\text{tower}} + k_{a\text{tower}})c_{\text{tower}}
\]

Where conceptually, \(k_t\) is a transportation multiplier, \(k_p\) is a profit multiplier, \(k_o\) is an overhead cost multiplier, and \(k_a\) is an assembly cost multiplier. By default, \(k_t = k_p = k_o = k_a = 0\).

For variable names access to override the default values see the \textit{Source Documentation}.

Turbine

The final turbine assembly also allows for user specification of other cost multipliers,

\[
m_{\text{turbine}} = m_{\text{rotor}} + m_{\text{nacelle}} + m_{\text{tower}}
\]

\[
c_{\text{turbine}} = (1 + k_{t\text{turbine}} + k_{p\text{turbine}})(1 + k_{o\text{turbine}} + k_{a\text{turbine}})(c_{\text{rotor}} + c_{\text{nacelle}} + c_{\text{towersys}})
\]

For variable names access to override the default values see the \textit{Source Documentation}.

Source Documentation

\texttt{nrel_csm_mass_2015}

\texttt{nrel_csm_cost_2015}

5.8.10 ORBIT

Overview

The Offshore Renewables Balance of system and Installation Tool (ORBIT) is a model developed by the National Renewable Energy Lab (NREL) to study the cost and times associated with Offshore Wind Balance of System (BOS) processes.
ORBIT includes many different modules that can be used to model phases within the BOS process, split into design and installation. It is highly flexible and allows the user to define which phases are needed to model their project or scenario using ProjectManager.

Documentation

ORBIT maintains its own Github repository and documentation (for now). When ORBIT tags a release, its code and tests are copied into the WISDEM project. Included in this code base is an OpenMDAO API that WISDEM interfaces with to provide a complete offshore cost analysis capability.

Usage

ORBIT can be easily used as a standalone module. This can be done through WISDEM or by installing ORBIT from its own repository as a separate project. For examples and documentation on ORBIT usage, see its Tutorial and API guides. If accessing ORBIT through WISDEM, just be sure to modify the python import lines from

```python
>>> import ORBIT
```

to

```python
>>> import wisdem.orbit
```

5.8.11 Plant_FinanceSE

Introduction

The set of models contained in this Plant_FinanceSE software allow for the determination of wind plant cost of energy. Plant_FinanceSE is implemented as an OpenMDAO component. All supporting code is also in OpenMDAO based on the Python programming language.

Documentation

Documentation for PlantFinanceSE

Theory

The theory for the models in this software are based directly on the work described in the reference for the NREL Cost and Scaling Model [FHL05]. The NREL Cost and Scaling Model provides a simple cost and sizing tool to estimate wind plant cost of energy based on a small number of input parameters such as rotor diameter, hub height and rated power. The models here extract the financial calculators from the model as stand-alone modules.
5.8.12 pyFrame3DD

Overview

Frame3DD has its own documentation that describes the theory, applications, and approach in depth. This Python wrapper includes a few modifications to the standard code:

- Elimination of all input/output files in favor of direct variable passing
- Arbitrary stiffness values can be passed in (rather than only rigid or free).
- Frame3DD allows inclusion of concentrated masses but they only affect the modal analysis. In pyFrame3DD they also affect the loads.

pyFrame3DD is used within WISDEM to do almost all of the structural analysis. This includes the rotor blades, drivetrain components such as the shaft and bedplate, tower, offshore support structures (monopile, jacket, floating platform), and others.

License

Frame3DD uses the GNU General Public License (GPL), which carries strong copy-left restrictions. The standalone pyFrame3DD repository is therefore also released under the GNU GPL license. For WISDEM, NREL has obtained a special dispensation from the Frame3DD author to use it within this codebase but still retain the Apache License, Version 2.0.

5.8.13 RotorSE

PreComp

PreComp [Bir06] implements a modified classic laminate theory combined with a shear-flow approach to estimate equivalent sectional inertial and stiffness properties of slender and hollow composite structures. PreComp requires the geometric description of the blade (chord, twist, section profile shapes, web locations), along with the internal structural layup (laminate schedule, orientation of fibers, laminate material properties). It allows for high-flexibility in the specification of the composite layup both spanwise and chordwise.

PreComp offers the attractive advantages of running almost instantaneously and not requiring sophisticated meshing routines. However, PreComp suffers the limitation that it does not estimate the shear stiffness terms. In addition, the other stiffness and inertia terms suffer inaccuracies compared to three dimensional finite element models [RPLG10].

Users interested to know more about PreComp should refer to the PreComp User Guide, which is available here [https://www.nrel.gov/docs/fy06osti/38929.pdf].

Precomp should be handled as a preliminary/conceptual design tool. Users interested in a more accurate tool should consider using the framework SONATA, which is an Python-based open source framework that has the ability to call the commercial solver VABS and the open source solver ANBA4 [FPBM20].
Code

The underlying code PreComp is written in Fortran and is linked to this class with f2py. The source code is available in the file wisdem/rotorse/PreCompPy.f90.

A Python wrapper to the code is implemented in the file wisdem/rotorse/rotor_elasticity.py. The wrapper itself is wrapped within an OpenMDAO explicit component in the same file.

Bibliography

Blade Cost Model

While WISDEM estimates the costs of some turbine components via semi-empirical relations tuned on historical data that get updated every few years, the blade cost model implemented in WISDEM adopts a bottom up approach and estimates the total costs of a blade as the sum of variable and fixed costs. The former are made of materials and direct labor costs, while the latter are the costs from overhead, building, tooling, equipment, maintenance, and capital. The model simulates the whole manufacturing process and it has been tuned to estimate the costs of blades in the range of 30 to 100 meters in length.

The blade cost model is described in detail in the NREL technical report https://www.nrel.gov/docs/fy19osti/73585.pdf.

Users should be made aware that the absolute values of the costs estimated by the cost models implemented in WISDEM certainly suffer a wide band of uncertainty, but the hope is that the models are able to capture the relative trends sufficiently well.

The blade cost model is implemented in the file wisdem/rotorse/rotor_cost.py and it is called in the file wisdem/rotorse/rotor_elasticity.py.

Regulation trajectory and AEP

A conventional variable-speed variable-pitch turbine features four region I, II, III, and IV, plus an intermediate region II1/2:

1. Region I: the turbine does not generate any power since the wind is below the cut-in speed, which is usually set at 3 or 4 m/s.
2. Region II: the turbines operates at its specified tip-speed ratio until either rated power or the maximum rotation speed is reached.
3. Region II1/2: if the maximum rotor speed is reached before rated power, the turbine maintains its rotor speed, therefore reducing the tip speed ratio, and pitches to maximize the power coefficient.
4. Region III: the blades are pitched and the turbine generates its nameplate power at constant rotor speed and generator torque.
5. Region IV: the wind is beyond cut out speed, the turbine is shutdown, and no power is generated.

WISDEM implements this regulation trajectory by running multiple instances of CCBlade. For regions II1/2 and III, sub-optimization routines run CCBlade iteratively to identify the right combinations of tip speed ratio and blade pitch angle to maximize power (region II1/2) or to maintain it constant (region III). These sub-optimization routines run at every wind speed and have a non negligible computational costs (total computational time is in the order of seconds). The code therefore offers the possibility to the user to compute the power curve only in region II or in region II and
WISDEM, Release 2.0

II/2 and simply adopt constant power in region III. The disadvantage of such approach is that the regulation trajectory in terms of tip speed ratio and pitch angle in regions II/2 and/or III is no longer available.

Once the regulation trajectory is completed, the annual energy production (AEP) (in kWh) is calculated as

\[ AEP = 8760 \times \text{loss} \int_{V_{in}}^{V_{out}} P(V) f(V) dV \]

where P is in Watts, loss is the drivetrain efficiency, and f(V) is a probability density function for the site.

Notably, WISDEM does not implement any peak shaving of the aerodynamic thrust.

**Loads, Deflections, and Strains**

**Loads**

WISDEM estimates the ultimate loads by running a steady-state CCBblade simulation at rated pitch and rotor speed values and at a wind speed corresponding to the peak of the three-sigma gust for the extreme turbulence model. This approach to estimate loads is known to be somewhat over-conservative, but it is capable of capturing the relative trends and it is suitable to run iterative optimization loop on standard hardware in just a few minutes, offering to the designer the chance of a wide exploration of the solution space.

The blade loads computed with such setup are used to compute blade deflections, strains, buckling margins, and are finally summed together to estimate overall loads at the rotor hub.

**Deflections**

The loads computed with CCBblade are applied to the beam model of Frame3dd, which returns the overall blade deflections. From those, the tip deflections are extracted.

**Strains**

RotorSE estimates the strains in the mid-point of the spar caps and of the trailing edge reinforcements. The strains are computed combining the sectional properties of the wind turbine blade generated by PreComp with the flapwise and edgewise loads computed in RotorSE. The strains are computed with the formula from [Han08]

\[ \epsilon(x, y) = \frac{M_1}{[EI]_1} y - \frac{M_2}{[EI]_2} x + \frac{N}{[EA]} \]  

(5.1)

**Buckling**

A panel buckling calculation is added to augment the sectional analysis. The constitutive equations for a laminate sequence can be expressed as

\[ \begin{bmatrix} N \\ M \end{bmatrix} = \begin{bmatrix} A & B \\ B & D \end{bmatrix} \begin{bmatrix} \epsilon^0 \\ k \end{bmatrix} \]  

(5.2)

where N and M are the average forces and moments of the laminate per unit length, and \( \epsilon^0 \) and k are the mid-plane strains and curvature (see [Hal92]). The D matrix is a 3 x 3 matrix of the form (while wind turbine blade cross-sections are not always precisely specially orthotropic they are well approximate as such).

\[
\begin{bmatrix}
D_{11} & D_{12} & 0 \\
D_{12} & D_{22} & 0 \\
0 & 0 & D_{66}
\end{bmatrix}
\]
The critical buckling load for long (length greater than twice the width) simply supported panels at a given section is estimated as [Joh94]

\[ N_{cr} = 2 \left( \frac{\pi}{w} \right)^2 \left[ \sqrt{D_{11}D_{22}} + D_{12} + 2D_{66} \right] \]

where \( w \) is the panel width. If we denote the matrix in the constitutive equation (Equation (5.2)) as \( S \) and its inverse as \( S^* \), then \( \epsilon_{zz} \approx S_{11}^{-1}N_z \). This expression ignores laminate shear and bending moment effects (the latter would be zero for a symmetric laminate), a good approximation for slender turbine blades. At the same time, an effective smeared modulus of elasticity can be computed by integrating across the laminate stack

\[ E_{zz} = \frac{1}{\epsilon_{zz}h} \int_{-h/2}^{h/2} \sigma_{zz} dh = \frac{N_z}{\epsilon_{zz}h} \]

where \( N_z \) in this equation is the average force per unit length of the laminate. Combining these equations yields an estimate for the effective axial modulus of elasticity

\[ E_{zz} = \frac{1}{S_{11}h} \]

The critical strain can then be computed as

\[ \epsilon_b = -\frac{N_{cr}}{h E_{zz}} \]

where the negative sign accounts for the fact that the strain is compressive in buckling.

**Bibliography**

**Rail Transport of Blades**

WISDEM simulates the rail transport of blades. Blades can be transported rigidly or via controlled flexing. The module is described in Carron and Bortolotti, 2020 [CB20].

**Bibliography**

**5.8.14 TowerSE**

**Introduction**

TowerSE is a systems engineering tower model for cylindrical shell wind turbine towers. Wind, wave, soil, and structural analyses are designed to be modular. Default implementations are provided, but all modules can be replaced with custom implementations. Default implementations use beam finite element theory, simple wind profiles, cylinder drag theory, Eurocode and Germanischer Lloyd shell and global buckling methods.

TowerSE is implemented as an OpenMDAO group. The beam finite element code is implemented in C++ and the rest of the implementation is in Python. All modules are linked in Python.
Theory

The tower class can be used to model a wind turbine tower or a tower/monopile configuration. No distinction is made between the tower and foundation, and so the term tower will be used throughout to refer to the entire structure. The current implementation assumes that the tower has cylindrical shell sections. The underlying analysis has the capability to handle general sections should such be desired. Dynamics for floating turbines are not included in TowerSE.

Theory for the finite element code is available at the website: Frame3DD. The RNA (rotor/nacelle/assembly) affects the stiffness of the structure and top loads. It is assumed that the RNA is a rigid body with respect to the tower modes. The RNA mass properties are transferred to the tower top using the generalized parallel axis theorem. Two different buckling approaches are implemented. A shell buckling method from Eurocode [EuropeanCfStandardisation93] and a global buckling method from Germanischer Lloyd [Llo05]. The implementation of the Eurocode buckling is modified slightly so as to produce continuously differentiable output. Since the tower is typically reinforced at shorter distances than the full tower length, the user may specify the reinforcement length. Hoop stress is estimated using the Eurocode method. Axial and shear stress calculations are done for cylindrical shell sections and are combined with hoop stress into a von Mises stress. Fatigue uses supplied damage equivalent moments, which are converted to stress for the given geometry. Using the stress, and inputs for the number of cycles and slope of the S-N curve allows for a damage calculation.

Computation of drag loads is done assuming drag over a smooth circular cylinder as a function of Reynolds number [Ros61]. Wave drag loads are computed using Morrison’s equation. Morrison’s equation predicts the hydrodynamic loads on a cylinder with three terms. These terms correspond to a drag force and the inertial forces due to wave motion and cylinder motion. The current analysis neglects the motion of the tower. With that assumption the two remaining forces per unit length are given as

\[ F'_{\text{max}} = \frac{\pi}{4} \rho_{\text{water}} A_{\text{current}} c_m d^2 \]
\[ F''_{\text{max}} = \frac{1}{2} \rho_{\text{water}} U_{\text{current}}^2 c_d d \]

The calculation of the resulting drag is separated from the actual velocity distributions, which are handled in the commonse.environment module. The environment model provides default implementations for power-law wind profiles, logarithmic-law wind profiles, and linear wave theory. A textbook model is used for soil stiffness properties [AONeilP79].

Discretization

As a user, the key concept to understand in order to take full advantage of TowerSE is its discretization. This is presented in some examples and bullet points below, for the standard land-based tower and the extension of that to include a monopile.

Land-Based Tower

The tower assumed to be divided into sections (“cans”), where the outer diameter varies linearly from the base to the top of each section and the wall thickness is constant. This is consistent with rolling a steel plate of constant thickness that is cut into a trapezoid and rolled into a tapered cylinder, or frustum. Sections are created by the grid point nodes in the reference_axis section of the yaml ontology or in the non-dimensional tower_s coordinates that vary from 0 to 1 applied to tower_height when using the TowerSE Python code directly. This means that the outer diameter is specified at each node, but the wall thickness is specified at each section, so internally the wall thickness vector is always one element less than the diameter vector. When using Python, this is done directly, with examples provided in the 05_tower_monopile case. When using the yaml ontology is used for input, there are some other considerations that users should be aware of:

- The tower height determined by the reference_axis is re-scaled to meet the user-specified hub height. The tower grid nodes are similarly stretched.
• The foundation height is taken from the bottom z-coordinate of the reference_axis.

• The section wall thickness is taken as the average of the grid point values specified in the internal_structure.layers.thicknesses entry. This can either be an accepted approximation, such as in the NREL 5-MW example, or accounted for by creating tiny sections at each true section interface where the thickness changes but the outer diameter stays the same, as the IEA 15-MW example does.

To demonstrate, with the following yaml input,

```yaml
assembly:
  hub_height: 90.
...
components:
  tower:
    outer_shape_bem:
      reference_axis: &ref_axis_tower
    x:
      grid: [0.0, 1.0]
      values: [0.0, 0.0]
    y:
      grid: [0.0, 1.0]
      values: [0.0, 0.0]
    z:
      grid: &grid_tower [0., 0.5, 1.]
      values: [0., 40.0, 80.0]
    outer_diameter:
      grid: *grid_tower
      values: [6.0, 5.0, 4.0]
    internal_structure_2d_fem:
      reference_axis: *ref_axis_tower
    layers:
      - name: tower_wall
        material: steel
        thickness:
          grid: *grid_tower
          values: [0.03, 0.02, 0.01]
```

The 80m tower would be re-scaled to 90m with two sections of 45m each, one with a wall thickness of 25mm and the other with a thickness of 15mm. To be able to set the wall thickness directly, do instead something similar to:

```yaml
components:
  tower:
    outer_shape_bem:
      reference_axis: &ref_axis_tower
    z:
      grid: &grid_tower [0., 0.5, 0.501, 1.]
      values: [0., 40.0, 40.001, 80.0]
    outer_diameter:
      grid: *grid_tower
      values: [6.0, 5.0, 5.0, 4.0]
    internal_structure_2d_fem:
      reference_axis: *ref_axis_tower
    layers:
      - name: tower_wall
        material: steel
```

(continues on next page)
Offshore Tower with Monopile

The monopile discretization adheres to the same pattern as the tower discretization, with some additional assumptions. Chief among these is that the transition piece height, where the monopile mates with the tower, is taken as the bottom z-coordinate of the tower (same point as the tower foundation height). The monopile reference_axis is shifted such that this is always true. From this transition point, the monopile extends into the water column and the beneath the sea floor according to the grid and length in the monopile reference_axis (the suctionpile_depth parameter is an output of with this approach). If the monopile does not reach the sea floor, the suction pile depth will be output as negative value so that it can be trapped as a design constraint. For gravity-based foundations, the user should set the monopile length to meet the sea floor exactly.

Due to numerical limitations in the soil boundary condition representation, the submerged pile segment will be represented as a single section that terminates at the mudline (sea floor). If the user specifies multiple sections in the submerged pile or a single section that extends into the water column, these will be forcibly altered to comply with this condition.

In the following example,

```
components:
  tower:
    outer_shape_bem:
      reference_axis:
        z:
          grid: [0., 0.5, 1.]
          values: [20., 60.0, 100.0]
  monopile:
    outer_shape_bem:
      reference_axis:
        z:
          grid: [0., 0.3846, 0.8462, 1.0]
          values: [-55.0, -30.0, 0.0, 10.0]
  env:
    water_depth: 30.0
```

The monopile grid would be shifted to meet the transition piece height of 20m, and the 65m monopile length would extend through the 30m water column to a 15m embedded pile depth in the soil.

Documentation

Referenced Tower Modules

5.9 Publications

A list of relevant publications related to WISDEM is provided here. The publications are ranked in chronological order.

- 2020


• 2019


• 2018


• 2017


• 2016

• 2015

• 2014
1. Ning SA. A simple solution method for the blade element momentum equations with guaranteed convergence. Wind Energy 17(9), 2014. DOI: 10.1002/we.1636


• 2011

5.10 Known issues within WISDEM

This doc page serves as a non-exhaustive record of any issues relevant to the usage of WISDEM. Some of these items are features that would be nice to have, but are not necessarily in the pipeline of development.

5.10.1 General issues

The components within WISDEM do not provide analytic gradients, so any gradient-based optimization using WISDEM uses finite-differenced gradients at the top level.

Many duplicate variables and names exist within WISDEM. The naming should be consistent throughout and data not duplicated. Many variable names can be changed for clarity.
5.10.2 Common pitfalls

Depending on the type of design study being done, not all of the disciplines available must be included. For example, if you are interested in the aerodynamic performance of the turbine blades, you can turn off all calculations related to the drivetrain, tower, costs, etc, to save on computational expense.

If you use an existing set of .yaml files and adapt them for your turbine analysis, make sure to update all values as needed. For example, if using a land-based 5MW reference turbine as a starting point for an offshore 5MW turbine, you will need to change the tower foundation properties.

5.10.3 Improvements yet-to-be-implemented

More complete documentation and comments throughout the code would be beneficial.

Portions of the codebase were written before external packages existed and the codebase could be simplified to take advantage of some built-in Scipy and OpenMDAO features now.

5.11 How to contribute code to WISDEM

Note: This section will be expanded in the future.

WISDEM is an open-source tool, thus we welcome users to submit additions or fixes to the code to make it better for everybody.

5.11.1 Issues

If you have an issue with WISDEM, a bug to report, or a feature to request, please submit an issue on the GitHub repository. This lets other users know about the issue. If you are comfortable fixing the issue, please do so and submit a pull request.

5.11.2 Documentation

When you add or modify code, make sure to provide relevant documentation that explains the new code. This should be done in code via comments, but also in the Sphinx documentation as well if you add a new feature or capability. Look at the .rst files in the docs section of the repo or click on view source on any of the doc pages to see some examples.

5.11.3 Testing

When you add code or functionality, add tests that cover the new or modified code. These may be units tests for individual components or regression tests for entire models that use the new functionality.

Each discipline sub-directory contains tests in the test folder. For example, wisdem/test/test_ccblade hosts the tests for CCBlade within WISDEM. Look at test_ccblade.py within that folder for a simple unit test that you can mimic when you add new components.
5.11.4 Pull requests

Once you have added or modified code, submit a pull request via the GitHub interface. This will automatically go through all of the tests in the repo to make sure everything is functioning properly. This also automatically does a coverage test to ensure that any added code is covered in a test. The main developers of WISDEM will then merge in the request or provide feedback on how to improve the contribution.

5.12 How to write docs for WISDEM code

5.12.1 Introduction

This page describes how to add, improve, and update any of the WISDEM documentation. The WISDEM documentation can be divided into two categories, a documentation for WISDEM submodules, and then WISDEM usage documentation, tutorials and guides.

For the former, the code documentation, is a documentation related to the code itself such as API and code usage. It is located within the each submodule’s docs/wisdem folder.

For the latter, general documentation (like this one here) the source files are contained in the docs repository outside of the docs/wisdem folder.

5.12.2 Getting started with the docs

When adding or updating the documentation please try to follow the these guidelines:

- files and folders should follow the existing naming convention
- images, figures and files should be placed in specific folders

To update the repo, you need to commit and push your changes. Use the following commands, but with a more descriptive commit message:

```
git commit -am "Updated docs"
git push
```

5.12.3 Sphinx and rst

In all cases documentation is generated using the Sphinx documentation generator.

The source files or the documentation itself is written in reStructuredText. A primer on the rst syntax can be found here. In general http://www.sphinx-doc.org is very helpful for syntax and examples.

**Note:** When viewing the documentation in a browser you can always view the source by clicking the Show Source link. This is also a great way of getting examples.

The sphinx system uses Makefiles to generate the documentation from the source .rst files.

In any case, to build the documentation, navigate to the docs folder and run the following command from the command line:

```
make html
```
5.12.4 General guidelines for formatting

Headings

When contributing to any documentation please use the following character for heading levels:

```
Sample heading 1
================
Sample heading 2
----------------
Sample heading 3
~~~~~~~~~~~~~~~~
Sample heading 4
****************
```

Note: Make sure the character underlines the entire heading or Sphinx will generate warnings when building.

Tables

Tables can be difficult to get “right” in html and especially when compiled to LaTeX. Using the simple version of tables often leads to imbalanced column widths and building LaTeX documents often results in bad tables. To try to mitigate this issue another table type should be used:

```
.. tabularcolumns:: |>{\raggedright\arraybackslash}\X{1}{5}|>{\raggedright\arraybackslash}\X{1}{5}|>{\raggedright\arraybackslash}\X{3}{5}|

.. list-table:: Demo table title
   :widths: 15 20 65
   :header-rows: 1

   * - Col 1
     - Col 2
     - Col 3

   * - Entry 1
     - Entry 2
     - Entry 3
```

Note:

- **tabularcolumns**: Controls how LaTeX generates the following table. The **widths** keyword is overridden/omitted for LaTeX when this keyword is specified. The **X{1}{5}** is this column width ratio.
- **widths**: keyword represents columns widths in percentages.
- **header-rows** keyword specifies how many rows are made bold and shaded

The code above generates the following table
5.12.5 Where should files related to documentation live?

Add figures and small files to the docs folder then embed them as desired in the .rst files. For larger files, host them elsewhere and link to them from the docs.

5.12.6 Where should you contribute docs to?

As you begin to determine if you should write a doc page, first search for relevant entries to make sure you don’t duplicate something that already exists. You can then judge if your contribution should be in its own doc page or if it should be added to an existing page. Make sure to think logically about where the information you prepared should live so it is intuitive for other people, especially people just starting out.

Once you have added your .rst file in the repo in a logical place within the file structure, also update the table of contents in the other relevant .rst files as needed. This ensures that your contributions can be easily found.

5.12.7 How to convert existing docs

If you already have something typed up, either in Latex, a basic text file, or another format, it’s usually pretty straightforward to convert this to rst. Pandoc is a helpful automated tool that converts text files near seamlessly.

5.12.8 How to request doc creation

If you think the docs should be modified or expanded, create an issue on the GitHub documentation repository. Do this by going to the WISDEM repo then click on Issues on the lefthand side of the page. There you can see current requests for doc additions as well as adding your own. Feel free to add any issue for any type of doc and members of the WISDEM development team can determine how to approach it. Assign someone or a few people to the issue who you think would be a good fit for that doc.
CHAPTER
SIX

INDICES AND TABLES

• genindex
• modindex
• search


wisdem.airfoilprep, 209
wisdem.airfoilprep.airfoilprep, 210
wisdem.ccblade, 210
wisdem浮动se.column, 266
wisdem浮动se.floating, 264
wisdem浮动se.loading, 266
wisdem浮动se.map_mooring, 266
wisdem浮动se.substructure, 266
wisdem浮动se.visualize, 266
wisdem.nrelcsm.nrel_csm_cost_2015, 283
wisdem.nrelcsm.nrel_csm_mass_2015, 283
INDEX

Symbols

__init__() (wisdem.commonse.utilities.CubicSplineSegment method), 224

A
arc_length_deriv() (in module wisdem.commonse.utilities), 223

B
BulkheadMass (class in wisdem.floatingse.column), 266
BuoyancyTankProperties (class in wisdem.floatingse.column), 266

C
Column (class in wisdem.floatingse.column), 266
ColumnBuckling (class in wisdem.floatingse.column), 266
ColumnGeometry (class in wisdem.floatingse.column), 266
ColumnProperties (class in wisdem.floatingse.column), 266
CubicSplineSegment (class in wisdem.commonse.utilities), 224
linspace_with_deriv() (in module wisdem.commonse.utilities), 222
Loading (class in wisdem.floatingse.loading), 266
LogWind (class in wisdem.commonse.environment), 221

M
MapMooring (class in wisdem.floatingse.map_mooring), 266
module
wisdem.airfoilprep, 209
wisdem.airfoilprep.airfoilprep, 210
wisdem.ccblade, 210
wisdem.floatingse.column, 266
wisdem.floatingse.floating, 264
wisdem.floatingse.loading, 266
wisdem.floatingse.map_mooring, 266
wisdem.floatingse.substructure, 266
wisdem.floatingse.visualize, 266
wisdem.nrelcsm.nrel_csm_cost_2015, 283
wisdem.nrelcsm.nrel_csm_mass_2015, 283

P
PowerWind (class in wisdem.commonse.environment), 221

S
smooth_abs() (in module wisdem.commonse.utilities), 223
smooth_max() (in module wisdem.commonse.utilities), 223
smooth_min() (in module wisdem.commonse.utilities), 223
SoilBase (class in wisdem.commonse.environment), 223
StiffenerMass (class in wisdem.floatingse.column), 266
Substructure (class in wisdem.floatingse.substructure), 266
SubstructureGeometry (class in wisdem.floatingse.substructure), 266

T
TowerSoil (class in wisdem.commonse.environment), 222
trapz_deriv() (in module wisdem.commonse.utilities), 223
TrussIntegerToBoolean (class in wisdem.floatingse.loading), 266

V
Visualize (class in wisdem.floatingse.visualize), 266

W
WaveBase (class in wisdem.commonse.environment), 221
WindBase (class in wisdem.commonse.environment), 221
wisdem.airfoilprep
    module, 209
    wisdem.airfoilprep.airfoilprep
        module, 210
wisdem.ccblade
    module, 210
wisdem.floatingse.column
    module, 266
wisdem.floatingse.floating
    module, 264
wisdem.floatingse.loading
    module, 266
wisdem.floatingse.map_mooring
    module, 266
wisdem.floatingse.substructure
    module, 266
wisdem.floatingse.visualize
    module, 266
wisdem.nrelcsm.nrel_csm_cost_2015
    module, 283
wisdem.nrelcsm.nrel_csm_mass_2015
    module, 283