WEIS

Release 2.0

NREL WEIS Team

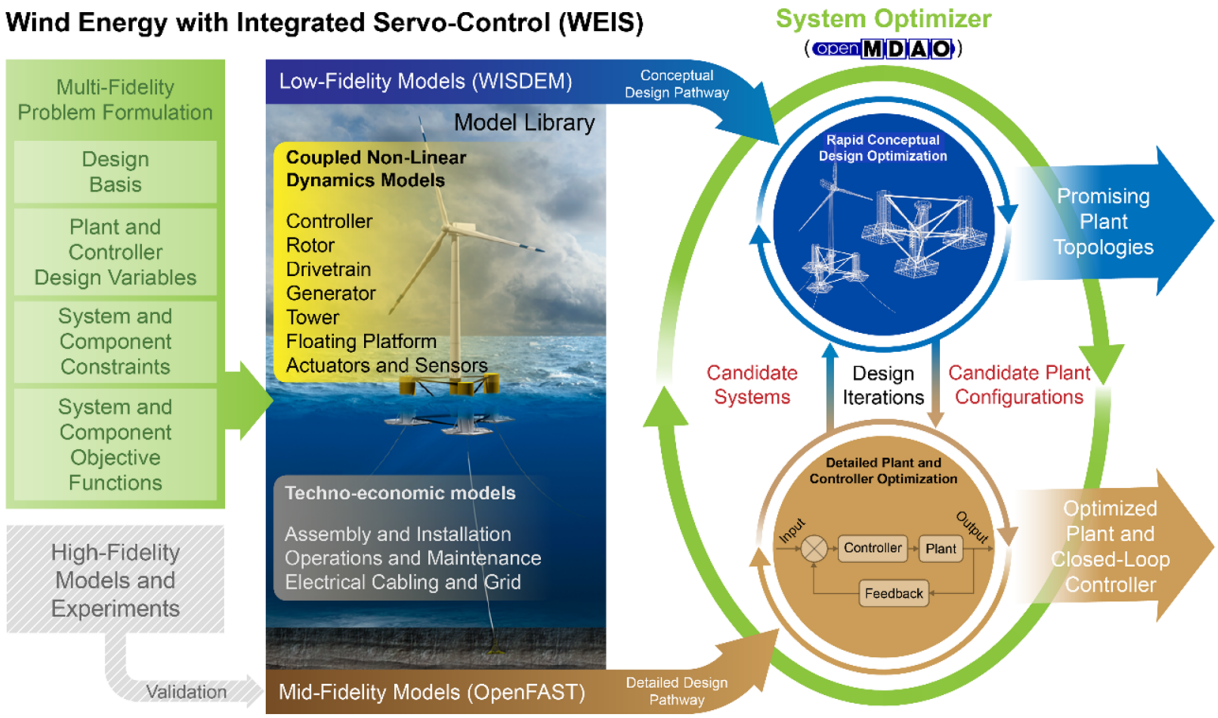
Jun 12, 2023

CONTENTS

WEIS, Wind Energy with Integrated Servo-control, performs multifidelity co-design of wind turbines. WEIS is a framework that combines multiple NREL-developed tools to enable design optimization of floating offshore wind turbines.

Important Links:

- Source Code Repository



CHAPTER

ONE

USING WEIS

1.1 WEIS Installation

To install WEIS, please follow the up-to-date instructions contained in the README.md at the root level of this repo, or on the WEIS GitHub page.

1.1.1 Installing SNOPT for use within WEIS

SNOPT is available for purchase here. Upon purchase, you should receive a zip file. Within the zip file, there is a folder called `src`. To use SNOPT within WEIS, paste all files from `src` except snopth.f into `WEIS/pyoptsparse/pyoptsparse/pySNOPT/source`. If you do this step before you install WEIS, SNOPT will be automatically compiled within pyOptSparse and should be usable. Otherwise, you can simply re-install WEIS following the same installation instructions after removing all `build` directories from the WEIS, WISDEM, and pyOptSparse directories.

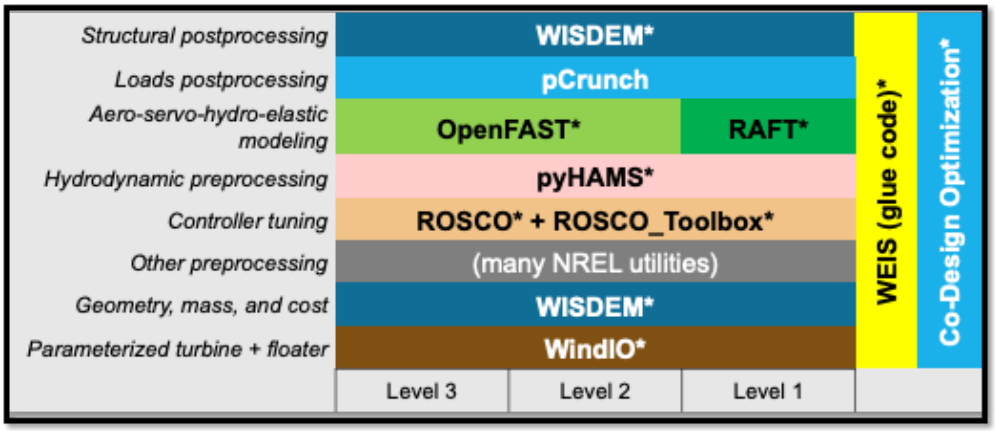
1.2 How WEIS works

WEIS is a stack of software that can be used for floating wind turbine design and analysis. The turbine's components (blade, tower, platform, mooring, and more) are parameterized in a geometry input. The system engineering tool WISDEM determines component masses, costs, and engineering parameters that can be used to determine modeling inputs. A variety of pre-processing tools are used to convert these system engineering models into simulation models.

The modeling_options.yaml determines how the turbine is simulated. We currently support OpenFAST and RAFT.

The analysis_options.yaml determine how to run simulations, either a single run for analysis, a sweep or design of experiments, or an optimization.

A full description of the inputs can be found here (will point to rst when available).



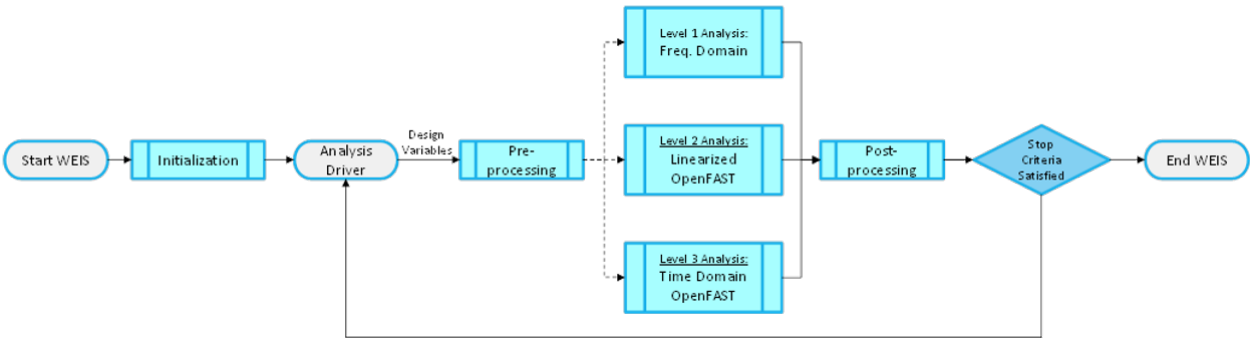
WEIS is "glued" together using openmdao components and groups.

WEIS works best by running examples:

- 01_aeroelasticse can be used to set up batches of aeroelastic simulations in OpenFAST
- 02_control_opt can be used to run WEIS from an existing OpenFAST model, and optimize control parameters only
- 03_NREL5MW_OC3 and 04_NREL5MW_OC4_semi can be used to simulate the OC spar and semi, respectively, from WISDEM geometries.
- 05_IEA-3.4-130-RWT runs the design load cases (DLCs) for the fixed-bottom IEA-3.4 turbine
- 06_IEA-15-240-RWT contains several examples for running the IEA-15MW with the VolturnUS platform, including tower and structural controller optimization routines
- 15_RAFT_Studies contains an example for optimizing a the IEA-15MW with the VolturnUS platform in RAFT

More documentation specific to these examples can be found there, with more to follow.

This documentation only covers a summary of WEIS's functionality. WEIS can be adapted to solve a wide variety of problems. If you have questions or would like to discuss WEIS's functionality further, please email dzalkind (at) nrel (dot) gov.



CHAPTER

TWO

OTHER USEFUL DOCS

2.1 How to contribute code to WEIS

WEIS is an open-source tool, thus we welcome users to submit additions or fixes to the code to make it better for everybody.

2.1.1 Issues

If you have an issue with WEIS, 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.

2.1.2 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 and also in the Sphinx documentation 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.

There is currently very little documentation for WEIS, so you have a lot of flexibility in terms of where you place your new documentation. Do not stress too much about the outline of the information you create, simply that it exists within the repo. We will reorganize the documentation content at a later date.

2.1.3 Using Git subtrees

The WEIS repo contains copies of other codes created by using the `git subtree` commands. Below are some details about how to add external codes and update them.

Adding a subtree for the first time

To add an external code for the first time, using OpenFAST as an example, type:

```
$ git remote add OpenFAST https://github.com/OpenFAST/openfast
$ git fetch OpenFAST
$ git subtree add -P OpenFAST OpenFAST/dev --squash
```

The `--squash` is important so WEIS doesn't get filled up with commits from the subtree repos.

Updating a subtree

Once a subtree code exists in this repo, we can update it like this. This first two lines are needed only if you don't have the remote for the particular subtree yet. If you already have the remote, only the last line is needed.

```
$ git remote add OpenFAST https://github.com/OpenFAST/openfast
$ git fetch OpenFAST
$ git subtree pull --prefix OpenFAST https://github.com/OpenFAST/openfast dev --squash --
↪message="Updating to latest OpenFAST develop"
```

Changes to these subtree codes **should only be made to their original repos**, *not* to this WEIS repo. Once those individual repos have been updated, use the previous `git subtree pull` command to pull in those updates to the WEIS repo. Once the upstream repos have your code changes, those changes have been pulled into your branch, you can then submit a PR for WEIS.

Troubleshooting subtree updates

If you run into trouble using `git subtree`, specifically if you see `git: 'subtree' is not a git command.`, try using your system git instead of any conda-installed git. Specifically, try using `/usr/bin/git subtree` for any subtree commands. If that doesn't work for you, please open an issue on this repo so we can track it.

Sometimes when updating a subtree, you might get an error that contains `could not rev-parse split hash`, which suggests that the subtree and the more recent branch have diverged in some way. To fix this, manually remove the entire folder created by the subtree command (e.g. `rm -rf ROSCO`.) Then, follow the steps to add a repo subtree for the first time. By deleting then re-adding the subtree repo, you ensure that all the changes are correctly added to the WEIS repo.

2.1.4 Testing

When you add code or functionality, add tests that cover the new or modified code. These may be units tests for individual code blocks or regression tests for entire models that use the new functionality. These tests should be a balance between minimizing computational cost and maximizing code coverage. This ensures continued functionality of WEIS while keeping development time short.

Any Python file with `test` in its name within the `weis` package directory is tested with each commit to WEIS. This is done through GitHub Actions and you can see the automated testing progress on the GitHub repo under the `Actions` tab, located here. If any test fails, this information is passed on to GitHub and a red X will be shown next to the commit. Otherwise, if all tests pass, a green check mark appears to signify the code changes are valid.

Unit tests

Each discipline sub-directory should contain tests in the `test` folder. For example, `weis/multifidelity/test` hosts the tests for multifidelity optimization within WEIS. Look at `test_simple_models.py` within that folder for a simple unit test that you can mimic when you add new code. Another simple unit test is contained in `weis/aeroelasticse/test` called `test_IECWind.py`.

Unit tests should be short and purposeful, test the smallest reasonable block of code, and quickly point to potential problems in the code. This article has some quick tips on how to write good unit tests.

Regression tests

Regression tests examine much larger portions of the code by examining top-level input and output relationships. Specifically, these tests check the values that the code produces against "truth" values and returns an error if they do not match. As an example, a low-level coding change might alter a default within a subsystem of the model being tested, which might result in a different AEP value for the wind turbine. The regression test would report that the AEP value differs, and thus the tests fail. Of course, it would be challenging to completely diagnose a coding change based on only regression tests, so well-made unit tests can help narrow down a problem much more quickly.

Within WEIS, regression tests live in the `weis/test` folder. Examine `test_aeroelasticse/test_DLC.py` to see an example regression test that checks OpenFAST results obtained through WEIS' wrapper. Specifically, that test compares all of the channel outputs against truth values contained in `.pkl` files within the same folder.

Like unit tests, regression tests should run quickly. They can have unrealistic simulation parameters (1 second time-series) as long as they adequately test the code.

Coveralls

To understand how WEIS is tested, we use a tool called Coveralls, which reports the lines of code that are used during testing. This lets WEIS developers know which functions and methods are tested, as well as where to add tests in the future.

When you push a commit to WEIS, all of the unit and regression tests are ran. Then, the coverage from those tests is reported to Coveralls automatically.

There may be some files that you specifically do not want Coveralls to report on, such as plotting scripts or helper scripts that aren't part of the main WEIS repo. To not include certain files, add them to the *omit* section of the *.coveragеac* file at the root repo level. For example, we currently do not report coverage on the *schema2rest.py* file, which is listed in the *omit* section.

2.1.5 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 WEIS will then merge in the request or provide feedback on how to improve the contribution.

In addition to the full unit and regression test suite, on pull requests additional examples are checked using GitHub Actions using the workflow labeled `run_exhaustive_examples`. These examples are useful for users to adapt, but are computationally expensive, so we do not test them on every commit. Instead, we test them only when code is about to be added to the main WEIS develop or master branches through pull requests. The coverage from these examples are not considered in Coveralls.

The examples that are covered are shown in `weis/test/run_examples.py`. If you add an example to WEIS, make sure to add a call to it in the `run_examples.py` script as well.

2.2 How to write docs for WEIS code

2.2.1 Introduction

This page describes how to add, improve, and update any of the WEIS documentation. The WEIS documentation can be divided into two categories, a documentation for WEIS submodules, and then WEIS usage documentation, tutorials and guides.

2.2.2 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
```

2.2.3 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
```

2.2.4 General guidelines for formatting

Headings

When contributing to any documentation please use the following character for heading levels:

```
Sample heading 1
================

Sample heading 2
----------------
```

(continues on next page)

(continued from previous page)

```
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

Table 2.1: Demo table title

Col 1	Col 2	Col 3
Entry 1	Entry 2	Entry 3

2.2.5 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.

2.2.6 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.

2.2.7 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.

2.2.8 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 WEIS 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 WEIS 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.

2.3 Known issues within WEIS

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.

2.3.1 Running on Eagle

Depending on the method that send batch scripts to Eagle, they may not run correctly in parallel. Specifically, when calling `sbatch submit_job.sh` and the job involves MPI and WEIS, multiple users have reported issues. These issues manifest as the script starting correctly, but then returning `MPI_INIT` errors.

One user could not successfully run jobs submitted via VS Code, but could via terminal. For another user, the regular Windows command terminal worked, but not the Ubuntu subsystem. **If you have any issues regarding running scripts on Eagle, first try a few different terminals to submit sbatch jobs**.

The reason for this error is not known.

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search