OpenFAST Documentation

Release v2.3.0

National Renewable Energy Laboratory

Oct 02, 2020

CONTENTS

OpenFAST is a multi-physics, multi-fidelity tool for simulating the coupled dynamic response of wind turbines. Practically speaking, OpenFAST is the framework (or "glue code") that couples computational modules for aerodynamics, hydrodynamics for offshore structures, control and electrical system (servo) dynamics, and structural dynamics to enable coupled nonlinear aero-hydro-servo-elastic simulation in the time domain. OpenFAST enables the analysis of a range of wind turbine configurations, including two- or three-blade horizontal-axis rotor, pitch or stall regulation, rigid or teetering hub, upwind or downwind rotor, and lattice or tubular tower. The wind turbine can be modeled on land or offshore on fixed-bottom or floating substructures.

Established in 2017, OpenFAST is an open-source software package that builds on FAST v8 (see *FAST v8 and the transition to OpenFAST*). The glue code and underlying modules are mostly written in Fortran (adhering to the 2003 standard), and modules can also be written in C or C++. It was created with the goal of being a community model developed and used by research laboratories, academia, and industry. It is managed by a dedicated team at the National Renewable Energy Lab. Our objective is to ensure that OpenFAST is well tested, well documented, and self-sustaining software. To that end, we are continually improving the documentation and test coverage for existing code, and we expect that new capabilities will include adequate testing and documentation. If you'd like to contribute, see the *Developer Documentation* and any open GitHub issues with the Help Wanted tag.

The following links provide more insight into OpenFAST as a software package:

- OpenFAST Github Organization
- Github Repository

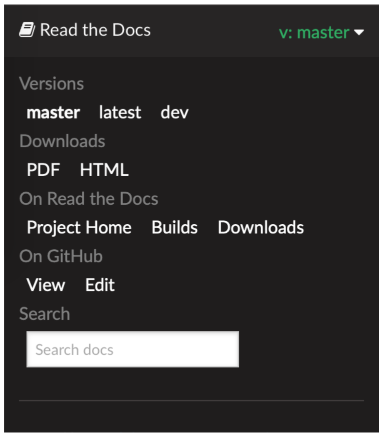
Documentation Directory

CHAPTER

ONE

THIS DOCUMENTATION

OpenFAST documentation is hosted on readthedocs, and is automatically generated from both the master and dev branches whenever new commits are added. Clicking on the bar on the lower left corner of the page reveals a panel (see image below) containing options to select the branch of the repository, download the documentation other formats (PFD, HTML, EPub), and link to other relevant websites.



While OpenFAST developer documentation is being enhanced here, developers are encouraged to consult the legacy FAST v8 Programmer's Handbook. Instructions on obtaining and installing OpenFAST are available in *Installing OpenFAST*, and documentation for verifying an installation with the automated tests is at *Testing OpenFAST*.

The majority of this documentation is divided into two parts:

User Documentation

Directed towards end-users, this part provides detailed documentation regarding usage of the OpenFAST and its underlying modules, as well as theory and verification documentation.

Developer Documentation

The developer guide is targeted towards users wishing to extend the functionality provided within OpenFAST. Here you will find details regarding the code structure, API supported by various classes, and links to source code documentation extracted using Doxygen.

CHAPTER TWO

INSTALLING OPENFAST

Guidelines and procedures for obtaining precompiled binaries or compiling OpenFAST from source code are described here. While there are multiple ways to achieve the same outcome, the OpenFAST team has developed a comprehensive and well thought out system for compiling the source code. Thus, the methods described here are the only officially supported and maintained paths for obtaining an OpenFAST executable.

For Windows users only, precompiled binaries are available as described in the *Download binaries* section. For all platforms, OpenFAST is configured to build with with CMake and a system-appropriate build tool. Background on CMake is given in *Understanding CMake*, and procedures for configuring and compiling are given in *CMake with Make for Linux/macOS* and *CMake with Visual Studio for Windows*. Finally, an alternative and more appropriate option for compiling on Windows while doing active software development is given in *Visual Studio Solution for Windows*.

2.1 Download binaries

Each tagged release is accompanied by precompiled binaries for Windows systems. DLL's for MAP and the DISCON controllers are also included. The following architecture and precision combinations are currently available:

- 32 bit single precision
- 64 bit single precision
- 64 bit double precision

All precompiled binaries can be found in the `Assets` dropdown in the GitHub Releases. Click here to download the latest binaries.

Note that the precompiled binaries require either the Intel fortran compiler or the Intel MKL redistributable libraries, which are not by default included with the binaries. To configure the libraries, download the installers from here and run the MSI file(s) to install the libraries. Note that if you have a Command Prompt open, you will need to close it after installing the libraries in order for the changes to take effect.

2.2 Compile from source

For compiling from source code, the NREL OpenFAST team has developed an approach that uses CMake to generate build files for all platforms. Currently, CMake support for Visual Studio while doing active development is not well supported, so OpenFAST maintains a Visual Studio solution giving Windows developers a better option for developing code, compiling and debugging in a streamlined manner. See *Visual Studio Solution for Windows* for more information.

2.2.1 Dependencies

Compiling OpenFAST from source requires additional libraries and tools that are not distributed with the OpenFAST repository. In many cases, these tools can be installed with a system's package manager (e.g. `homebrew` for macOS, `yum` for CentOS/Red Hat, or `apt` for Debian-based systems like Ubuntu). If binaries are downloaded or compiled manually, be sure they are installed in a standard location for your system so that the other components of the OpenFAST build system can find the dependencies.

Build tools

An environment-specific build system is required and will consist of a combination of the packages listed in the table below.

Package	Applicable systems	Minimum version	Link
CMake	All	3.0	https://cmake.org
GNU Make	macOS, Linux	1.8	https://www.gnu.org/software/make/
Visual Studio	Windows	2015	https://visualstudio.microsoft.com>
GNU Compiler Collection (gfortran, gcc, g++)	macOS, Linux	4.6.0	https://gcc.gnu.org
Intel Parallel Studio (ifort, icc)	All	2013	https://software.intel.com/en-us/parallel-studio-xe/

Math libraries

Math libraries with the BLAS and LAPACK interfaces are also required. These can be obtained as free, open source libraries or paid, closed source versions. Some packages contain separate libraries for each interface while others have the interfaces bundles into a single binary. The most common options are listed in the table below.

Library	Maintainer	Paid/Free	Open Source?	Link
BLAS	NetLib	Free	Yes	http://www.netlib.org/blas/
LAPACK	NetLib	Free	Yes	http://www.netlib.org/lapack/
BLAS/LAPACK	OpenBLAS	Free	Yes	https://www.openblas.net
MKL	Intel	Paid	No	https://software.intel.com/en-us/mkl

Dependencies for the test suite

The following packages are required to run the test suite:

- Python 3
- MatPlotLib - used for generating error plots

Dependencies for the C++ API

When using the C++ API, the following packages are required:

- HDF5 (provided by `HDF5_ROOT`)
- yaml-cpp (provided by `YAML_ROOT`)

2.2.2 Get the code

OpenFAST can be cloned (i.e., downloaded) from its Github repository via the command line:

```
git clone https://github.com/OpenFAST/OpenFAST.git
```

An archive of the source code can also be downloaded directly from these links:

- "master" branch - Stable release
- "dev" branch - Latest updates

2.2.3 Visual Studio Solution for Windows

A complete Visual Studio solution is maintained for working with the OpenFAST on Windows systems. The procedure for configuring the system and proceding with the build process are documentated in the following section:

Building OpenFAST on Windows with Visual Studio

These instructions are specifically for the standalone Visual Studio project at *openfast/vs-build*. Separate CMake documentation is provided for Windows users at Section 2.2.6.

Prerequisites

1. A version of Visual Studio (VS).
   - Currently VS 2013 Professional and VS 2015 Community Edition have been tested with OpenFAST.
   - A list of Intel Fortran compatible VS versions and specific installation notes are found here.
   - The included C/C++ project files for MAP++ and the Registry are compatible with VS 2013, but will upgrade seemlessly to a newer version of VS.
   - If you download and install Visual Studio 2015 Community Edition, you will need to be sure and select the `C/C++ component` using the `Customize` option.
2. Intel Fortran Compiler
   - Currently only version 2017.1 has been tested with OpenFAST, but any newer version should be compatible.
   - You can download an Intel Fortran compiler here.
   - Only install Intel Fortran after you have completed your Visual Studio installation.
3. Git for Windows
   - Download and install git for Windows.
4. Python 3.x for Windows (for regression/unit testing)

- The testing framework of OpenFAST requires the use of Python.
- Please see Section 3 on testing OpenFAST for further information on this topic.
- We have been working with Continuum's Anaconda installation of Python 3.6 for Windows.

Compiling OpenFAST

1. Open `A command prompt`, or `git bash` shell from the `Start` menu
2. Create a directory where you will clone OpenFAST repository (change `code` to your preferred name)

```
mkdir code
cd code
```

3. Clone the OpenFAST repository

```
git clone https://github.com/openfast/openfast.git
```

This will create a directory called `openfast` within the `code` directory.

4. Using Windows Explorer, navigate to the directory `openfast\vs-build\FAST` and double-click on the `FAST.sln` Visual Studio solution file. This will open Visual Studio with the FAST solution and its associated projects.

NOTE: If you are using Visual Studio 2015 or newer, you will be asked to upgrade both the `Fast_Registry.vcxproj` and the `MAP_dll.vcxproj` files to a newer format. Go ahead and accept the upgrade on those files.

5. Select the desired Solution Configuration, such as `Release`, and the desired Solution Platform, such as `x64` by using the drop down boxes located below the menubar.
6. Build the solution using the `Build->Build Solution` menu option.
7. If the solution built without errors, the executable will be located under the `openfast\build\bin` folder.

2.2.4 Understanding CMake

To more fully understand CMake and its methodology, visit this guide on running CMake.

CMake is a build configuration system that creates files as input to a build tool like GNU Make, Visual Studio, or Ninja. CMake does not compile code or run compilers directly, but rather creates the environment needed for another tool to run compilers and create binaries. A CMake project is described by a series of files called `CMakeLists.txt` located in directories throughout the project. The main CMake file for OpenFAST is located at `openfast/CMakeLists.txt` and each module and glue-code has its own `CMakeLists.txt`; for example, AeroDyn and BeamDyn have one at `openfast/modules/aerodyn/CMakeLists.txt` and `openfast/modules/beamdyn/CMakeLists.txt`, respectively.

Running CMake

Running CMake and a build tool will create many files (text files and binaries) used in the various stages of the build. For this reason, a `build` folder should be created to contain all of the generated files associated with the build process. Here, an important file called `CMakeCache.txt` contains the user-defined settings for the CMake configuration. This file functions like memory storage for the build. It is initially created the first time the CMake command is run and populated with the initial settings. Then, any subsequent changes to the settings will be updated and stored there.

CMake can be executed in a few ways:

- Command line interace: `cmake`
- Command line curses interface: `ccmake`
- Official CMake GUI

The CMake GUI is only distributed for Windows, but it can be built from source for other platforms. OpenFAST's build process focuses on the command line execution of CMake for both the Linux/macOS and Windows terminals. The command line syntax to run CMake for OpenFAST is generally:

```
cmake <path-to-primary-CMakeLists.txt> [options]

Options
    -D <var>[:<type>]=<value>     = Create or update a cmake cache entry.
```

For example, a common CMake command issued from the `openfast/build` directory is:

```
# cmake <path-to-primary-CMakeLists.txt> [options]
#   where
#     <path-to-primary-CMakeLists.txt> is ".."
#     [options] can be
#       -DBUILD_SHARED_LIBS:BOOL=ON or
#       -DBUILD_SHARED_LIBS=ON

cmake .. -DBUILD_SHARED_LIBS=ON
```

The command line curses interface can be invoked similarly:

```
ccmake ..
```

The interface will be rendered in the terminal window and all navigation happens through keyboard inputs.

OpenFAST CMake options

CMake has a large number of general configuration variables available. A good resource for useful CMake variables is at this link: GitLab CMake variables. The CMake API documentation is also helpful for searching through variables and determining the resulting action. Note that the CMake process should be well understood before customizing the general options.

The CMake options specific to OpenFAST and their default settings are:

```
BUILD_DOCUMENTATION            - Build documentation (Default: OFF)
BUILD_OPENFAST_CPP_API         - Enable building OpenFAST - C++ API (Default: OFF)
BUILD_OPENFAST_SIMULINK_API    - Enable building OpenFAST for use with Simulink
BUILD_SHARED_LIBS              - Enable building shared libraries (Default: OFF)
BUILD_TESTING                  - Build the testing tree (Default: OFF)
CMAKE_BUILD_TYPE               - Choose the build type: Debug Release (Default:
↪Release)
```

(continues on next page)

(continued from previous page)

```
CMAKE_Fortran_MODULE_DIRECTORY - Set the Fortran Modules directory
CMAKE_INSTALL_PREFIX           - Install path prefix, prepended onto install␣
↪directories.
DOUBLE_PRECISION               - Treat REAL as double precision (Default: ON)
FPE_TRAP_ENABLED               - Enable Floating Point Exception (FPE) trap in␣
↪compiler options (Default: OFF)
GENERATE_TYPES                 - Use the openfast-regsitry to autogenerate types␣
↪modules
ORCA_DLL_LOAD                  - Enable OrcaFlex library load (Default: OFF)
USE_DLL_INTERFACE              - Enable runtime loading of dynamic libraries␣
↪(Default: ON)
OPENMP                         - Enable OpenMP parallelization in FVW (Default: OFF)
```

Additional system-specific options may exist for a given system, but those should not impact the OpenFAST configuration. As mentioned above, the configuration variables are set initially but can be changed at any time. For example, the defaults may be accepted to initially configure the project, but then the settings may be configured individually:

```
# Initial configuration with the default settings
cmake ..

# Change the build to Debug mode rather than Release
cmake .. -DCMAKE_BUILD_TYPE=Debug

# Use dynamic linking rather than static linking
cmake .. -DBUILD_SHARED_LIBS=ON
```

The commands above are equivalent to having run this command the first time:

```
# Initial configuration in Debug mode with dynamic linking
cmake .. -DCMAKE_BUILD_TYPE=Debug -DBUILD_SHARED_LIBS=ON
```

CMAKE_BUILD_TYPE

This option allows to set the compiler optimization level and debug information. The value and its effect are listed in the table below.

CMAKE_BUILD_TYPE	Effect
Release	`-O3` optimization level
RelWithDebInfo	`-O2` optimization level with `-g` flag for debug info
MinSizeRel	`-O1` optimization level
Debug	No optimization and *-g* flag for debug info; additional debugging flags: `-fcheck=all -pedantic -fbacktrace`

Use `Debug` during active development to add debug symbols for use with a debugger. This build type also adds flags for generating runtime checks that would otherwise result in undefined behavior. `MinSizeRel` adds basic optimizations and targets a minimal size for the generated executable. The next level, `RelWithDebInfo`, enables vectorization and other more agressive optimizations. It also adds debugging symbols and results in a larger executable size. Finally, use `Release` for best performance at the cost of increased compile time.

This flag can be set with the following command:

```
cmake .. -DCMAKE_BUILD_TYPE=RelWithDebInfo
```

CMAKE_INSTALL_PREFIX

This flag sets the location of the compiled binaries when the build tool runs the `install` command. It should be a full path in a carefully chosen location. The binaries will be copied into `include`, `lib`, and `bin` subfolders under the value of this flag. The default is to install binaries within the repository in a folder called `install`.

This flag can be set with the following command:

```
cmake .. -DCMAKE_INSTALL_PREFIX="/usr/local/"
```

Setting the build tool

CMake can target a variety of build tools or *generators*. To obtain a list of available generators on the current system, run with the empty generator flag, select the target from the list, and rerun with the generator flag populated:

```
# Run with the empty -G flag to get a list of available generators
cmake .. -G

# CMake Error: No generator specified for -G
#
# Generators
# * Unix Makefiles               = Generates standard UNIX makefiles.
# Ninja                        = Generates build.ninja files.
# Xcode                        = Generate Xcode project files.
# CodeBlocks - Ninja           = Generates CodeBlocks project files.
# CodeBlocks - Unix Makefiles  = Generates CodeBlocks project files.
# CodeLite - Ninja             = Generates CodeLite project files.
# CodeLite - Unix Makefiles    = Generates CodeLite project files.
# Sublime Text 2 - Ninja       = Generates Sublime Text 2 project files.
# Sublime Text 2 - Unix Makefiles
#                              = Generates Sublime Text 2 project files.
# Kate - Ninja                 = Generates Kate project files.
# Kate - Unix Makefiles        = Generates Kate project files.
# Eclipse CDT4 - Ninja         = Generates Eclipse CDT 4.0 project files.
# Eclipse CDT4 - Unix Makefiles= Generates Eclipse CDT 4.0 project files.

# Choose one from the list above and pass it as an argument after -G
#   NOTE: wrap this is in quotes!
cmake .. -G"Sublime Text 2 - Ninja"
```

Note: If the chosen generator name contains spaces, be sure to wrap it in quotes.

Math libraries

The CMake project is configured to search for the required math libraries in default locations. However, if math libraries are not found, they can be specified directly to CMake. The two required libraries are `BLAS` and `LAPACK`, and their location can be passed to CMake with this command syntax:

```
cmake .. -DBLAS_LIBRARIES="/path/to/blas" -DLAPACK_LIBRARIES="/path/to/lapack"
```

The paths given should be to the directory which contains the libraries, not to the libraries themselves.

2.2.5 CMake with Make for Linux/macOS

After reading *Understanding CMake*, proceed with configuring OpenFAST. The CMake project is well developed for Linux and macOS systems, so the default settings should work as given. These settings should only be changed when a custom build is required.

The procedure for configuring CMake and compiling with GNU Make on Linux and macOS systems is given below.

```
# Clone the repository from GitHub using git
git clone https://github.com/OpenFAST/OpenFAST.git

# Move into the OpenFAST directory
cd OpenFAST

# Create the build directory and move into it
mkdir build
cd build

# Execute CMake with the default options;
# this step creates the Makefiles
cmake ..

# Execute the Make-help command to list all available targets
make help

# Choose a particular target or give no target to compile everything
make
```

Tip: Compile in parallel by adding "-jN" where N is the number of parallel processes to use

This will build the OpenFAST project in the `build` directory. Binaries are located in `openfast/build/glue-codes/` and `openfast/build/modules/`. Since all build-related files are located in the `build` directory, a new fresh build process can be accomplished by simply deleting the build directory and starting again.

2.2.6 CMake with Visual Studio for Windows

After reading *Understanding CMake*, proceed with configuring OpenFAST. The result of this configuration process will be a Visual Studio solution which will be fully functional for compiling any of the targets within OpenFAST. However, this method lacks support for continued active development. Specifically, any settings that are configured in the Visual Studio solution directly will be lost any time CMake is executed. Therefore, this method should only be used to compile binaries, and the procure described in *Visual Studio Solution for Windows* should be used for active OpenFAST development on Windows.

The procedure for configuring CMake and compiling with Visual Studio on Windows systems is given below.

```
# Clone the repository from GitHub using git
git clone https://github.com/OpenFAST/OpenFAST.git

# Move into the OpenFAST directory
cd OpenFAST

# Create the build directory and move into it
mkdir build
cd build
```

(continues on next page)

(continued from previous page)

```
# Execute CMake with the default options and a specific Visual Studio version
# and build architecture. For a list of available CMake generators, run
# ``cmake .. -G``.
# This step creates the Visual Studio solution.
cmake .. -G "Visual Studio 14 2015 Win64"

# Open the generated Visual Studio solution
start OpenFAST.sln
```

Visual Studio will open a solution containing all of the OpenFAST projects, and any module library, module driver, or glue-code can be compiled from there. The compiled binaries are located within a directory determined by the Visual Studio build type (Release, Debug, or RelWithDebInfo) in `openfast/build/glue-codes/` and `openfast/build/modules/`. For example, the OpenFAST executable will be located at `openfast/build/glue-codes/Release/openfast.exe` when compiling in *Release* mode.

The CMake-generated Visual Studio build is not currently fully functional. Any configurations made to the Solution in the Visual Studio UI will be lost when CMake is executed, and this can happen whenever a change is made to the structure of the file system or if the CMake configuration is changed. It is recommended that this method **not** be used for debugging or active development on Windows. Instead, see *Visual Studio Solution for Windows*.

2.3 Appendix

The following are additional methods for installation which may not be fully test or may be deprecated in the future.

2.3.1 Building OpenFAST with Spack

The process to build and install OpenFAST with Spack on Linux or macOS is described here.

Dependencies

OpenFAST has the following dependencies:

- LAPACK libraries. Users should set `BLAS_LIBRARIES` and `LAPACK_LIBRARIES` appropriately for CMake if the library isn't found in standard paths. Use *BLASLIB* as an example when using Intel MKL.
- For the optional C++ API, HDF5 (provided by `HDF5_ROOT`) and yaml-cpp (provided by `YAML_ROOT`)
- For the optional testing framework, Python 3+ and Numpy

Building OpenFAST Semi-Automatically Using Spack on macOS or Linux

The following describes how to build OpenFAST and its dependencies mostly automatically on macOS using Spack. This can also be used as a template to build OpenFAST on any Linux system with Spack.

These instructions were developed on macOS 10.11 with the following tools installed via Homebrew:

- GCC 6.3.0
- CMake 3.6.1
- pkg-config 0.29.2

Step 1

Checkout the official Spack repo from github (we will checkout into `${HOME}`):

```
cd ${HOME} && git clone https://github.com/LLNL/spack.git
```

Step 2

Add Spack shell support to your `.profile` by adding the lines:

```
export SPACK_ROOT=${HOME}/spack
. $SPACK_ROOT/share/spack/setup-env.sh
```

Step 3

Copy the https://raw.githubusercontent.com/OpenFAST/openfast/dev/share/spack/package.py file to your installation of Spack:

```
mkdir ${SPACK_ROOT}/var/spack/repos/builtin/packages/openfast
cd ${SPACK_ROOT}/var/spack/repos/builtin/packages/openfast
wget --no-check-certificate https://raw.githubusercontent.com/OpenFAST/openfast/dev/
↪share/spack/package.py
```

Step 4

Try `spack info openfast` to see if Spack works. If it does, check the compilers you have available by:

```
machine:~ user$ spack compilers
==> Available compilers
-- gcc ---------------------------------------------------------
gcc@6.3.0  gcc@4.2.1

-- clang -------------------------------------------------------
clang@8.0.0-apple  clang@7.3.0-apple
```

Step 5

Install OpenFAST with your chosen version of GCC:

```
spack install openfast %gcc@6.3.0
```

To install OpenFAST with the C++ API, do:

```
spack install openfast+cxx %gcc@6.3.0
```

That should be it! Spack will automatically use the most up-to-date dependencies unless otherwise specified. For example to constrain OpenFAST to use some specific versions of dependencies you could issue the Spack install command:

```
spack install openfast %gcc@6.3.0 ^hdf5@1.8.16
```

The executables and libraries will be located at

```
spack location -i openfast
```

Add the appropriate paths to your `PATH` and `LD_LIBRARY_PATH` to run OpenFAST.

2.3.2 Building OpenFAST on Windows with CMake and Cygwin 64-bit

WARNING: This build process takes a significantly long amount of time. If GNU tools are not required, it is recommended that Windows users see one of the following sections:

- *Download binaries*
- *CMake with Visual Studio for Windows*
- *Building OpenFAST on Windows with Visual Studio*.

Installing prerequisites

1. Download and install Cygwin 64-bit. You will need to `Run as Administrator` to complete the installation process.
   - Choose `Install from internet`
   - Choose the default install location
   - Choose the default package download location
   - Choose `Direct connection`
   - Choose a download site
   - See next step for `select packages`. Alternately, you can skip this step and run `setup-x86_64.exe` anytime later to select and install required software.
2. Select packages necessary for compiling `OpenFAST`. Choose `binary` packages and not the source option.
   - Choose `Category` view, we will be installing packages from `Devel` and `Math`
   - From `Devel` mark the following packages for installation
     - `cmake`
     - `cmake-doc`
     - `cmake-gui`
     - `cygwin-devel`
     - `gcc-core`
     - `gcc-fortran`
     - `gcc-g++`
     - `git`
     - `make`
     - `makedepend`
   - From `Math` mark the following packages for installation

  - liblapack-devel
  - libopenblas
- To run the test suite, install these optional packages from Python:
  - python3
  - Python3-numpy
- Click Next and accept all additional packages that the setup process requests to install to satisfy dependencies

3. It is *recommended* that you reboot the machine after installing Cygwin and all the necessary packages.

Compiling OpenFAST

From here, pick up from the Linux with CMake instructions at *CMake with Make for Linux/macOS*.

Other tips

- If you would like to run openfast.exe from the cmd terminal, then you must add the C:\cygwin64\lib\lapack and C:\cygwin64\home\<USERNAME>\software\bin to your %PATH% variable in environment setting. Replace <USERNAME> with your account name on Windows system.
- It is suggested to compile with optimization level 2 for Cygwin. Do this by changing the build mode in the cmake command

```
cmake .. -DCMAKE_BUILD_TYPE=RelWithDebInfo
```

CHAPTER

THREE

TESTING OPENFAST

The OpenFAST test suite consists of glue code and module level regression tests and unit tests. The regression tests compare locally generated solutions to a set of baseline solutions. The unit tests ensure that individual subroutines are functioning as intended.

All of the necessary files corresponding to the regression tests are contained in the `reg_tests` directory. The unit test framework is housed in `unit_tests` while the actual tests are contained in the directory corresponding to the tested module.

3.1 Configuring the test suite

Portions of the test suite are linked to the OpenFAST repository through a *git submodule*. Specifically,

- r-test
- pFUnit

Tip: Be sure to clone the repo with the `--recursive` flag or execute `git submodule update --init --recursive` after cloning.

The test suite can be configured with CMake similar to OpenFAST. The default CMake configuration is suitable for most systems, but may need customization for particular build environments. See the *Understanding CMake* section for more details on configuring the CMake targets. While the unit tests must be built with CMake due to its external dependencies, the regression test may be executed without CMake.

3.2 Test specific documentation

3.2.1 Unit test

In a software package as dynamic and collaborative as OpenFAST, confidence in multiple layers of code is best accomplished with a strong system of unit tests. Through robust testing practices, the entire OpenFAST community can understand the intention behind code blocks and debug or expand functionality quicker and with more confidence and stability.

Unit testing in OpenFAST modules is accomplished through pFUnit. This framework provides a Fortran abstraction to the popular xUnit structure. pFUnit is compiled along with OpenFAST through CMake when the CMake variable `BUILD_TESTING` is turned on.

The BeamDyn module has been unit tested and should serve as a reference for future development and testing.

Dependencies

The following packages are required for unit testing:

- Python 3.7+
- CMake
- pFUnit - Included in OpenFAST repo through a git-submodule

Compiling

Compiling the unit tests is handled with CMake similar to compiling OpenFAST in general. After configuring CMake with `BUILD_TESTING` turned on, new build targets are created for each module included in the unit test framework named `[module]_utest`. Then, `make` the target to test:

```
cmake .. -DBUILD_TESTING=ON
make beamdyn_utest
```

This creates a unit test executable at `openfast/build/unit_tests/beamdyn_utest`.

Executing

To execute a module's unit test, simply run the unit test binary. For example:

```
>>>$ ./openfast/build/unit_tests/beamdyn_utest
.............
Time:         0.018 seconds

 OK
 (14 tests)
```

pFUnit will display a `.` for each unit test successfully completed and a `F` for each failing test. If any tests do fail, the failure criteria will be displayed listing which particular value caused the failure. Failure cases display the following output:

```
>>>$ ./unit_tests/beamdyn_utest
.....F.......
Time:         0.008 seconds

Failure
in:
test_BD_CrvMatrixH_suite.test_BD_CrvMatrixH
    Location:
[test_BD_CrvMatrixH.F90:48]
simple rotation with known parameters: Pi on xaxis expected +0.5000000 but found: +0.
↪4554637;  difference: |+0.4453627E-01| > tolerance:+0.1000000E-13;  first␣
↪difference at element [1, 1].

FAILURES!!!
Tests run: 13, Failures: 1, Errors: 0
Note: The following floating-point exceptions are signalling: IEEE_INVALID_FLAG IEEE_
↪DIVIDE_BY_ZERO
ERROR STOP *** Encountered 1 or more failures/errors during testing. ***

Error termination. Backtrace:
```

(continues on next page)

(continued from previous page)

```
#0  0x1073b958c
#1  0x1073ba295
#2  0x1073bb1b6
#3  0x106ecdd4f
#4  0x1063fabee
#5  0x10706691e
```

Adding unit tests

Unit tests should be included for each new, *testable* code block (subroutine or function). What is testable is the discretion of the developer, but an element of the pull request review process will be evaluating test coverage.

New unit tests can be added to a `tests` directory alongside the `src` directory included in each module. For example, the BeamDyn module directory is structured as

```
openfast/
  └── modules/
        └── beamdyn/
              ├── src/
              │     ├── BeamDyn.f90
              │     └── BeamDyn_Subs.f90
              └── tests/
                    ├── test_BD_Subroutine1.F90
                    ├── test_BD_Subroutine2.F90
                    └── test_BD_Subroutine3.F90
```

Each unit test must be contained in a unique file called `test_[SUBROUTINE].F90` where `[SUBROUTINE]` is the code block being tested. Finally, update the CMake configuration for building a module's unit test executable by copying the BeamDyn CMake configuration into a new module directory:

```
cp -r openfast/unit_tests/beamdyn openfast/unit_tests/[module]
```

Then, modify the new `CMakeLists.txt` with the appropriate list of test subroutines and module name variables.

For reference, a template unit test file is included at `openfast/unit_tests/test_SUBROUTINE.F90`. Each unit test should fully test the target code block. If full test coverage is not easily achievable, it may be an indication that refactoring would be beneficial.

Some useful topics to consider when developing and testing for OpenFAST are:

- Test driven development
- Separation of concerns
- pFUnit usage

3.2.2 Regression test

The regression test executes a series of test cases which intend to fully describe OpenFAST and its module's capabilities. Jump to one of the following sections for instructions on running the regression tests:

- *Executing with Python driver*
- *Executing with CTest*
- *Regression test examples*
- *Windows with Visual Studio regression test*

Each locally computed result is compared to a static set of baseline results. To account for system, hardware, and compiler differences, the regression test attempts to match the current machine and compiler type to the appropriate solution set from these combinations:

Operating System	Compiler	Hardware
macOS	**GNU**	**2017 MacbookPro**
CentOS 7	Intel	NREL Eagle - Intel Skylake
CentOS 7	GNU	NREL Eagle - Intel Skylake
Windows 10	Intel	Dell Precision 3530

The compiler versions, specific math libraries, and more info on hardware used to generate the baseline solutions are documented in the r-test repository documentation. Currently, the regression test supports only double precision builds.

The regression test system can be executed with CMake (through its included test driver, CTest) or manually with a custom Python driver. Both systems provide similar functionality with respect to testing, but CTest integration provides access to multithreading, automation, and test reporting via CDash. Both modes of execution require some configuration as described in the following sections.

In both modes of execution a directory is created in the build directory called `reg_tests` where all of the input files for the test cases are copied and all of the locally generated outputs are stored. Ultimately, both CTest and the manual execution program call a series of Python scripts and libraries in `reg_tests` and `reg_tests/lib`. One such script is `lib/pass_fail.py` which reads the output files and computes a norm on each channel reported. If the maximum norm is greater than the given tolerance, that particular test is reported as failed. The failure criteria is outlined below.

```
difference = abs(testData - baselineData)
for i in nChannels:
    if channelRange < 1:
        norm[i] = MaxNorm( difference[:,i] )
    else:
        norm[i] = MaxNorm( difference[:,i] ) / channelRange

if max(norm) < tolerance:
    pass = True
else:
    pass = False
```

Dependencies

The following packages are required for regression testing:

- Python 3.7+
- Numpy
- CMake and CTest (Optional)
- Bokeh 1.4 (Optional)

Executing with Python driver

The regression test can be executed manually with the included driver at `openfast/reg_tests/manualRegressionTest.py`. This program reads a case list file at `openfast/reg_tests/r-test/glue-codes/openfast/CaseList.md`. Cases can be removed or ignored by starting that line with a `#`. The driver program includes multiple optional flags which can be obtained by executing with the help option:

```
>>>$ python manualRegressionTest.py -h
usage: manualRegressionTest.py [-h] [-p [Plotting-Flag]] [-n [No-Execution]]
                            [-v [Verbose-Flag]] [-case [Case-Name]]
                            OpenFAST System-Name Compiler-Id Test-Tolerance

Executes OpenFAST and a regression test for a single test case.

positional arguments:
OpenFAST               path to the OpenFAST executable
System-Name            current system's name: [Darwin,Linux,Windows]
Compiler-Id            compiler's id: [Intel,GNU]
Test-Tolerance         tolerance defining pass or failure in the regression
                         test

optional arguments:
-h, --help             show this help message and exit
-p [Plotting-Flag], -plot [Plotting-Flag]
                         bool to include plots in failed cases
-n [No-Execution], -no-exec [No-Execution]
                         bool to prevent execution of the test cases
-v [Verbose-Flag], -verbose [Verbose-Flag]
                         bool to include verbose system output
-case [Case-Name]      single case name to execute
```

Note: For the NREL 5MW turbine test cases, an external ServoDyn controller must be compiled and included in the appropriate directory or all NREL 5MW cases will fail without starting. More information is available in the documentation for the r-test repository, but be aware that these three DISCON controllers must exist

```
openfast/build/reg_tests/glue-codes/openfast/5MW_Baseline/ServoData/DISCON.dll
openfast/build/reg_tests/glue-codes/openfast/5MW_Baseline/ServoData/DISCON_ITIBarge.
↪dll
openfast/build/reg_tests/glue-codes/openfast/5MW_Baseline/ServoData/DISCON_OC3Hywind.
↪dll
```

Executing with CTest

CTest is included with CMake and is primarily a set of preconfigured targets and commands. To use the CTest driver for the regression test, execute CMake as described in *Installing OpenFAST*, but with this additional flag: `-DBUILD_TESTING=ON`.

The regression test specific CMake variables are

```
BUILD_TESTING
CTEST_OPENFAST_EXECUTABLE
CTEST_[MODULE]_EXECUTABLE where [MODULE] is the module name
CTEST_PLOT_ERRORS
CTEST_REGRESSION_TOL
```

Some additional resources that are required for the full regression test suite are included in the CMake project. Specifically, external ServoDyn controllers must be compiled for a given system and placed in a particular location. Thus, be sure to execute the build command with the `install` target:

```
# Configure CMake with testing enabled and accept the default
# values for all other test-specific CMake variables
cmake .. -DBUILD_TESTING=ON

# Build and install
make install
```

Note: REMINDER: For the NREL 5MW turbine test cases, an external ServoDyn controller must be compiled and included in the appropriate directory or all NREL 5MW cases will fail without starting. More information is available in the documentation for the r-test repository, but be aware that these three DISCON controllers must exist

```
openfast/build/reg_tests/glue-codes/openfast/5MW_Baseline/ServoData/DISCON.dll
openfast/build/reg_tests/glue-codes/openfast/5MW_Baseline/ServoData/DISCON_ITIBarge.
↪dll
openfast/build/reg_tests/glue-codes/openfast/5MW_Baseline/ServoData/DISCON_OC3Hywind.
↪dll
```

After CMake configuration and compiling, the automated regression test can be executed by running either of the commands `make test` or `ctest` from the `build` directory. If the entire OpenFAST package is to be built, CMake will configure CTest to find the new binary at `openfast/build/glue-codes/openfast/openfast`. However, if the intention is to build only the test suite, the OpenFAST binary should be specified in the CMake configuration under the `CTEST_OPENFAST_EXECUTABLE` flag. There is also a corresponding `CTEST_[MODULE]_NAME` flag for each module included in the regression test.

When driven by CTest, the regression test can be executed by running various forms of the command `ctest` from the build directory. The basic commands are:

```
# Run the entire regression test
ctest

# Disable actual execution of tests;
# this is helpful in formulating a particular ctest command
ctest -N

# Run the entire regression test with verbose output
ctest -V
```

(continues on next page)

(continued from previous page)

```
# Run tests by name where TestName is a regular expression (regex)
ctest -R [TestName]

# Run all tests with N tests executing in parallel
ctest -j [N]
```

Each regression test case contains a series of labels associating all of the modules used. The labeling can be seen in the test instantiation in `reg_tests/CTestList.cmake` or with the command:

```
# Print all available test labels
ctest --print-labels
```

The test cases corresponding to a particular label can be executed with this command:

```
# Filter the test cases corresponding to a particular label
ctest -L [Label]
```

Flags can be compounded making useful variations such as

```
# Run all cases that use AeroDyn14 with verbose output
ctest -V -L aerodyn14

# Run all cases that use AeroDyn14 in 16 concurrent processes
ctest -j 16 -L aerodyn14

# Run the case with name "5MW_DLL_Potential_WTurb" with verbose output
ctest -V -R 5MW_DLL_Potential_WTurb

# List all tests with the "beamdyn" label
ctest -N -L beamdyn

# List the labels included in all tests matching the regex "bd"
ctest -N -R bd --print-labels
```

The automated regression test writes new files only into the build directory. Specifically, all locally generated solutions are located in the corresponding glue-code or module within `openfast/build/reg_tests`. The baseline solutions contained in `openfast/reg_tests/r-test` are strictly read and are not modified by the automated process.

Regression test examples

The following examples illustrate methods of running the regression tests on Unix-based systems. However, similar procedures can be used on Windows with CMake and CTest. An alternate method of running the regression tests on Windows is given in *Detailed example of running on Windows*.

Compile OpenFAST and execute with CTest

The following example assumes the user is starting completely from scratch. The commands below download the source code, configure the OpenFAST project with CMake, compile all executables, and execute the full regression test suite.

```
# Download the source code from GitHub
#    Note: The default branch is 'master'
git clone --recursive https://github.com/openfast/openfast.git
cd openfast

# If necessary, switch to another target branch and update r-test
git checkout dev
git submodule update

# Create the build and install directories and move into build
mkdir build install && cd build

# Configure CMake for testing
# - BUILD_TESTING - turn ON
# - CTEST_OPENFAST_EXECUTABLE - accept the default
# - CTEST_[MODULE]_EXECUTABLE - accept the default
cmake .. -DBUILD_TESTING=ON

# Compile and install
make install

# Execute the full test suite with 4 concurrent processes
ctest -j4
```

Configure with CMake and a given executable

This example assumes the user has a fully functional OpenFAST executable available along with any necessary libraries, but does not have the source code repository downloaded. This might be the case when executables are distributed within an organization or downloaded from an OpenFAST Release. Here, nothing will be compiled, but the test suite will be configured with CMake for use with the CTest command.

```
# Download the source code from GitHub
#    Note: The default branch is 'master'
git clone --recursive https://github.com/openfast/openfast.git
cd openfast

# If necessary, switch to another target branch and update r-test
git checkout dev
git submodule update

# Create the build directory and move into it
mkdir build && cd build
```

(continues on next page)

(continued from previous page)

```
# Configure CMake with openfast/reg_tests/CMakeLists.txt for testing
# - BUILD_TESTING - turn ON
# - CTEST_OPENFAST_EXECUTABLE - provide a path
# - CTEST_[MODULE]_EXECUTABLE - provide a path
cmake ../reg_tests \
    -DBUILD_TESTING=ON \
    -DCTEST_OPENFAST_EXECUTABLE=/home/user/Desktop/openfast_executable \
    -DCTEST_BEAMDYN_EXECUTABLE=/home/user/Desktop/beamdyn_driver

# Install required files
make install

# Execute the full test suite with 4 concurrent processes
ctest -j4
```

Python driver with a given executable

This example assumes the user has a fully functional OpenFAST executable available along with any necessary libraries, but does not have the source code repository downloaded. This might be the case when executables are distributed within an organization or downloaded from an OpenFAST Release. Nothing will be compiled, but the test suite will be executed with the included Python driver.

```
# Download the source code from GitHub
#    Note: The default branch is 'master'
git clone --recursive https://github.com/openfast/openfast.git
cd openfast

# If necessary, switch to another target branch and update r-test
git checkout dev
git submodule update

# Execute the Python driver
cd reg_tests
python manualRegressionTest.py -h
# usage: manualRegressionTest.py [-h] [-p [Plotting-Flag]] [-n [No-Execution]]
#                                [-v [Verbose-Flag]] [-case [Case-Name]]
#                                OpenFAST System-Name Compiler-Id Test-Tolerance
#
# Executes OpenFAST and a regression test for a single test case.
#
# positional arguments:
#   OpenFAST              path to the OpenFAST executable
#   System-Name           current system's name: [Darwin,Linux,Windows]
#   Compiler-Id           compiler's id: [Intel,GNU]
#   Test-Tolerance        tolerance defining pass or failure in the regression
#                         test
#
# optional arguments:
#   -h, --help            show this help message and exit
#   -p [Plotting-Flag], -plot [Plotting-Flag]
#                         bool to include plots in failed cases
#   -n [No-Execution], -no-exec [No-Execution]
#                         bool to prevent execution of the test cases
#   -v [Verbose-Flag], -verbose [Verbose-Flag]
```

(continues on next page)

(continued from previous page)

```
#                           bool to include verbose system output
#   -case [Case-Name]       single case name to execute

python manualRegressionTest.py \
    ..\build\bin\openfast_x64_Double.exe \
    Windows \
    Intel \
    1e-5
```

Detailed example of running on Windows

The *Python driver with a given executable* example can be used for running the regression tests on a Windows computer. However, a more detailed, step-by-step description is given in *Windows with Visual Studio regression test*.

Windows with Visual Studio regression test

1) Clone the openfast repo and initialize the testing database

    a) Open a git command shell window (like git bash)

    b) Change your working directory to the location above where you want your local repo to be located (the repo will be placed into a folder called openfast at this location)

    c. Type: `git clone https://github.com/openfast/openfast.git` (this creates a local version of the openfast repo on your computer). You should see something like this:

```
Cloning into 'openfast'...
remote: Counting objects: 23801, done.
remote: Compressing objects: 100% (80/80), done.
remote: Total 23801 (delta 73), reused 102 (delta 50), pack-reused 23670
Receiving objects: 100% (23801/23801), 92.10 MiB  18.99 MiB/s, done.
Resolving deltas: 100% (13328/13328), done.
Checking connectivity... done.
```

    d) Type: `cd openfast` (change your working directory to the openfast folder)

    e) Type: `git checkout dev` (this places your local repo on the correct branch of the openfast repo)

    f) Type: `git submodule update --init --recursive` (this downloads the testing database to your computer) You should see something like this:

```
Submodule 'reg_tests/r-test' (https://github.com/openfast/r-test.git)␣
↪registered for path 'reg_tests/r-test'
Cloning into 'reg_tests/r-test'...
remote: Counting objects: 3608, done.
remote: Compressing objects: 100% (121/121), done.
remote: Total 3608 (delta 22), reused 161 (delta 21), pack-reused 3442
Receiving objects: 100% (3608/3608), 154.52 MiB  26.29 MiB/s, done.
Resolving deltas: 100% (2578/2578), done.
Checking connectivity... done.
Submodule path 'reg_tests/r-test': checked out
↪'b808f1f3c1331fe5d03c5aaa4167532c2492d378'
```

2) Build The Regression Testing DISCON DLLs

   a) Open the Visual Studio Solution (`Discon.sln`) located in `openfast\vs-build\Discon` folder
   b) Choose Release and x64 for the Solutions Configuration and Solutions Platform, respectively
   c) From the menu bar select `Build->Build Solution`
   d) You should now see the files `Discon.dll`, `Discon_ITIBarge.dll`, and `Discon_OC3Hywind.dll` in your `openfast\reg_tests\r-test\glue-codes\fast\5MW_Baseline\ServoData` folder.

3) Build OpenFAST using Visual Studio
   a) Open the Visual Studio Solution (`FAST.sln`) located in `openfast\vs-build\FAST` folder
   b) Choose Release_Double and x64 for the Solutions Configuration and Solutions Platform, respectively
   c) From the menu bar select `Build->Build Solution`
      i) If this is the first time you have tried to build openfast, you will get build errors!!! [continue to steps (ii) and (iii), otherwise if FAST builds successfully, continue to step (3d)]
      ii) **Cancel build using the menubar `Build->Cancel`** [VS is confused about the build-order/dependency of the project files in FASTlib., but canceling and restarting VS, it somehow as enough info from the partial build to get this right, now]
      iii) Close your Visual Studio and then Repeat Steps (a) through (c)
   d) You should now see the file `openfast_x64_Double.exe` in your `openfast\build\bin` folder.

4) Prepare regression tests
   a) Create a subdirectory called `reg_tests` in your `openfast\build` folder.
   b) Copy the contents of `openfast\reg_tests\r-test` to `openfast\build\reg_tests`.

5) Execute the OpenFAST regression Tests
      a) Open a command prompt which is configured for Python [like Anaconda3]
      b) Change your working directory to `openfast\reg_tests`
      c) **Type: `python manualRegressionTest.py ..\build\bin\openfast_x64_Double.exe Windows`**
         You should see this: `executing AWT_YFix_WSt`
      d) The tests will continue to execute one-by-one until you finally see something like this:

```
executing AWT_YFix_WSt                           PASS
executing AWT_WSt_StartUp_HighSpShutDown         PASS
executing AWT_YFree_WSt                          PASS
executing AWT_YFree_WTurb                        PASS
executing AWT_WSt_StartUpShutDown                PASS
executing AOC_WSt                                PASS
executing AOC_YFree_WTurb                        PASS
executing AOC_YFix_WSt                           PASS
executing UAE_Dnwind_YRamp_WSt                   PASS
executing UAE_Upwind_Rigid_WRamp_PwrCurve        PASS
executing WP_VSP_WTurb_PitchFail                 PASS
executing WP_VSP_ECD                             PASS
executing WP_VSP_WTurb                           PASS
executing WP_Stationary_Linear                   PASS
executing SWRT_YFree_VS_EDG01                    PASS
executing SWRT_YFree_VS_EDC01                    PASS
executing SWRT_YFree_VS_WTurb                    PASS
executing 5MW_Land_DLL_WTurb                     PASS
```

(continues on next page)

(continued from previous page)

```
executing 5MW_OC3Mnpl_DLL_WTurb_WavesIrr         PASS
executing 5MW_OC3Trpd_DLL_WSt_WavesReg           PASS
executing 5MW_OC4Jckt_DLL_WTurb_WavesIrr_MGrowth PASS
executing 5MW_ITIBarge_DLL_WTurb_WavesIrr        PASS
executing 5MW_TLP_DLL_WTurb_WavesIrr_WavesMulti  PASS
executing 5MW_OC3Spar_DLL_WTurb_WavesIrr         PASS
executing 5MW_OC4Semi_WSt_WavesWN                PASS
executing 5MW_Land_BD_DLL_WTurb                  PASS
```

e) If an individual test succeeds you will see `PASS` otherwise you will see `FAIL` after that test's name

3.3 Continuous integration

A TravisCI configuration file is included with the OpenFAST source code at `openfast/.travis.yml`. The continuous integration infrastructure is still under development, but the status for all branches and pull requests can be found on the TravisCI OpenFAST page.

For development and testing purposes, a version of the TravisCI test can be run locally with Docker. The code snippet below outlines starting a TravisCI image on Docker.

```
# Running a travis ci image on docker locally

# Run this on your local machine's command line
BUILDID="build-1"
INSTANCE="travisci/ci-garnet:packer-1512502276-986baf0"
docker run --name $BUILDID -dit $INSTANCE /sbin/init
docker exec -it $BUILDID bash -l

# Now you're inside your docker image
sudo apt-get update
sudo apt-get install python3-pip
sudo -E apt-get -yq --no-install-suggests --no-install-recommends install gfortran␣
↪libblas-dev liblapack-dev
git clone --depth=50 https://github.com/OpenFAST/openfast.git OpenFAST/openfast
cd OpenFAST/openfast

# Modify this line for the commit or pull request to build
git fetch origin +refs/pull/203/merge:

git checkout -qf FETCH_HEAD
git submodule update --init --recursive

export FC=/usr/bin/gfortran-7
export DOUBLE_PRECISION=ON
export TRAVIS_BUILD_INTEL=YES
export TRAVIS_COMPILER=gcc
export CC=gcc
gcc --version
pyenv shell 3.6.3

source ~/.bashrc
pip3 install numpy
mkdir build && cd build
```

(continues on next page)

(continued from previous page)

```
cmake .. -DBUILD_TESTING=ON -DDOUBLE_PRECISION=$DOUBLE_PRECISION -DBUILD_SHARED_
↪LIBS=ON
make -j 8 install
```

CHAPTER

FOUR

USER DOCUMENTATION

This section contains documentation for the OpenFAST module-coupling environment and its underlying modules. Documentation covers usage of models, underlying theory, and in some cases module verification.

We are in the process of transitioning legacy FAST v8 documentation, which can be found at https://nwtc.nrel.gov/. Details on the transition from FAST v8 to OpenFAST may be found in Section 4.7

4.1 API changes between versions

This page lists the main changes in the OpenFAST API (input files) between different versions.

The changes are tabulated according to the module input file, line number, and flag name. The line number corresponds to the resulting line number after all changes are implemented. Thus, be sure to implement each in order so that subsequent line numbers are correct.

4.1.1 OpenFAST v2.3.0 to OpenFAST *dev*

Added in OpenFAST *dev*			
Module	Line	Flag Name	Example Value
HydroDyn	53	ExctnMod	0 ExctnMod - Wave Excitation model {0: None, 1: DFT, 2: state-space} (-)
OpenFAST	44	CalcSteady	true CalcSteady - Calculate a steady-state periodic operating point before linearization? [unused if Linearize=False] (flag)
OpenFAST	45	TrimCase	3 TrimCase - Controller parameter to be trimmed {1:yaw; 2:torque; 3:pitch} [used only if CalcSteady=True] (-)
OpenFAST	46	TrimTol	0.0001 TrimTol - Tolerance for the rotational speed convergence [used only if CalcSteady=True] (-)
OpenFAST	47	TrimGain	0.001 TrimGain - Proportional gain for the rotational speed error (>0) [used only if CalcSteady=True] (rad/(rad/s) for yaw or pitch; Nm/(rad/s) for torque)
OpenFAST	48	Twr_Kdmp	0 Twr_Kdmp - Damping factor for the tower [used only if CalcSteady=True] (N/(m/s))
OpenFAST	49	Bld_Kdmp	0 Bld_Kdmp - Damping factor for the blades [used only if CalcSteady=True] (N/(m/s))
InflowWind	48	InitPosition(x)	0.0 InitPosition(x) - Initial offset in +x direction (shift of wind box) [Only used with WindType = 5] (m)
AeroDyn	13	CompAA	False CompAA - Flag to compute AeroAcoustics calculation [only used when WakeMod=1 or 2]
AeroDyn	14	AA_InputFile	"unused" AA_InputFile - Aeroacoustics input file
AeroDyn	35	[separator line]	====== OLAF – cOnvecting LAgrangian Filaments (Free Vortex Wake) Theory Options ================== [used only when WakeMod=3]
AeroDyn	36	OLAFInputFileName	"Elliptic_OLAF.dat" OLAFInputFileName - Input file for OLAF [used only when WakeMod=3]
AirFoilTables	4*	BL_file	"unused" BL_file - The file name including the boundary layer characteristics of the profile. Ignored if the aeroacoustic module is not called.

*non-comment line count

Additional nodal output channels added for *AeroDyn15*, *BeamDyn*, and *ElastoDyn*.

4.1.2 OpenFAST v2.2.0 to OpenFAST v2.3.0

Removed in OpenFAST v2.3.0			
Module	Line	Flag Name	Example Value
AeroDyn Airfoil Input File - Airfoil Tables	2	Ctrl	0 Ctrl ! Control setting (must be 0 for current AirfoilInfo)

Added in OpenFAST v2.3.0			
Module	Line	Flag Name	Example Value
AeroDyn Airfoil Input File - Airfoil Tables	2	UserProp	0 UserProp ! User property (control) setting
AeroDyn	37	AFTabMod	1 AFTabMod - Interpolation method for multiple airfoil tables {1=1D interpolation on AoA (first table only); 2=2D interpolation on AoA and Re; 3=2D interpolation on AoA and UserProp} (-)

4.1.3 OpenFAST v2.1.0 to OpenFAST v2.2.0

No changes required.

4.1.4 OpenFAST v2.0.0 to OpenFAST v2.1.0

Added in OpenFAST v2.1.0			
Module	Line	Flag Name	Example Value
BeamDyn driver	21	GlbRotBladeT0	True GlbRotBladeT0 - Reference orientation for BeamDyn calculations is aligned with initial blade root?

4.1.5 OpenFAST v1.0.0 to OpenFAST v2.0.0

Removed in OpenFAST v2.0.0			
Module	Line	Flag Name	Example Value
BeamDyn	5	analysis_type	analysis_type - 1: Static analysis; 2: Dynamic analysis

Added in OpenFAST v2.0.0			
Module	Line	Flag Name	Example Value
AeroDyn	22	SkewModFactor	"default" SkewModFactor - Constant used in Pitt/Peters skewed wake model {or "default" is 15/32*pi} (-) [used only when SkewMod=2; unused when WakeMod=0]
AeroDyn	30	Section header	====== Dynamic Blade-Element/Momentum Theory Options == [used only when WakeMod=2]
AeroDyn	31	DBEMT_Mod	2 DBEMT_Mod - Type of dynamic BEMT (DBEMT) model {1=constant tau1, 2=time-dependent tau1} (-) [used only when WakeMod=2]
AeroDyn	32	tau1_const	4 tau1_const - Time constant for DBEMT (s) [used only when WakeMod=2 and DBEMT_Mod=1]
BeamDyn	5	QuasiStaticInit	True QuasiStaticInit - Use quasi-static pre-conditioning with centripetal accelerations in initialization (flag) [dynamic solve only]
BeamDyn	11	load_retries	DEFAULT load_retries - Number of factored load retries before quitting the simulation
BeamDyn	14	tngt_stf_fd	DEFAULT tngt_stf_fd - Flag to use finite differenced tangent stiffness matrix (-)
BeamDyn	15	tngt_stf_comp	DEFAULT tngt_stf_comp - Flag to compare analytical finite differenced tangent stiffness matrix (-)
BeamDyn	16	tngt_stf_pert	DEFAULT tngt_stf_pert - perturbation size for finite differencing (-)
BeamDyn	17	tngt_stf_difftol	DEFAULT tngt_stf_difftol - Maximum allowable relative difference between analytical and fd tangent stiffness (-)
BeamDyn	18	RotStates	True RotStates - Orient states in the rotating frame during linearization? (flag) [used only when linearizing]

4.1.6 FAST v8.16 to OpenFAST v1.0.0

The transition from FAST v8 to OpenFAST is described in detail at *FAST v8 and the transition to OpenFAST*.

Removed in OpenFAST v1.0.0			
Module	Line	Flag Name	Example Value
OpenFAST	18	CompSub	0 CompSub - Compute sub-structural dynamics (switch) {0=None; 1=SubDyn}

Added in OpenFAST v1.0.0			
Module	Line	Flag Name	Example Value
Open-FAST	18	Comp-Sub	0 CompSub - Compute sub-structural dynamics (switch) {0=None; 1=SubDyn; 2=External Platform MCKF}
Aero-Dyn	12	Cavity-Check	False CavitCheck - Perform cavitation check? (flag)
Aero-Dyn	17	Patm	9999.9 Patm - Atmospheric pressure (Pa) [used only when CavitCheck=True]
Aero-Dyn	18	Pvap	9999.9 Pvap - Vapor pressure of fluid (Pa) [used only when CavitCheck=True]
Aero-Dyn	19	Fluid-Depth	9999.9 FluidDepth - Water depth above mid-hub height (m) [used only when Cav-itCheck=True]

4.2 AeroDyn Users Guide and Theory Manual

4.2.1 Introduction

AeroDyn is a time-domain wind turbine aerodynamics module that is coupled in the OpenFAST multi-physics engineering tool to enable aero-elastic simulation of horizontal-axis turbines. AeroDyn can also be driven as a standalone code to compute wind turbine aerodynamic response uncoupled from OpenFAST. When coupled to OpenFAST, AeroDyn can also be linearized as part of the linearization of the full coupled solution (linearization is not available in standalone mode). AeroDyn was originally developed for modeling wind turbine aerodynamics. However, the module equally applies to the hydrodynamics of marine hydrokinetic (MHK) turbines (the terms “wind turbine”, “tower”, “aerodynamics” etc. in this document imply “MHK turbine”, “MHK support structure”, “hydrodynamics” etc. for MHK turbines). Additional physics important for MHK turbines, not applicable to wind turbines, computed by AeroDyn include a cavitation check. This documentation pertains version of AeroDyn in the OpenFAST github repository. The AeroDyn version released of OpenFAST 1.0.0 is most closely related to AeroDyn version 15 in the legacy version numbering. AeroDyn version 15 was a complete overhaul from earlier version of AeroDyn. AeroDyn version 15 and newer follows the requirements of the FAST modularization framework.

AeroDyn calculates aerodynamic loads on both the blades and tower. Aerodynamic calculations within AeroDyn are based on the principles of actuator lines, where the three-dimensional (3D) flow around a body is approximated by local two-dimensional (2D) flow at cross sections, and the distributed pressure and shear stresses are approximated by lift forces, drag forces, and pitching moments lumped at a node in a 2D cross section. Analysis nodes are distributed along the length of each blade and tower, the 2D forces and moment at each node are computed as distributed loads per unit length, and the total 3D aerodynamic loads are found by integrating the 2D distributed loads along the length. When AeroDyn is coupled to OpenFAST, the blade and tower analysis node discretization may be independent from the discretization of the nodes in the structural modules. The actuator line approximations restrict the validity of the model to slender structures and 3D behavior is either neglected, captured through corrections inherent in the model (e.g., tip-loss, hub-loss, or skewed-wake corrections), or captured in the input data (e.g., rotational augmentation corrections applied to airfoil data).

AeroDyn assumes the turbine geometry consists of a one-, two-, or three-bladed rotor atop a single tower. While the undeflected tower is assumed to be straight and vertical, an undeflected blade may consider out-of-plane curvature and in-plane sweep. For blades, the 2D cross sections where the aerodynamic analysis take place may follow the out-of-plane curvature, but in-plane sweep is assumed to be accomplished by shearing, rather than rotation of the 2D cross section. Aerodynamic imbalances are possible through the use of geometrical differences between each blade.

When AeroDyn is coupled to OpenFAST, AeroDyn receives the instantaneous (possibly displaced/deflected) structural position, orientation, and velocities of analysis nodes in the tower, hub, and blades. As with curvature and sweep, the 2D cross sections where the blade aerodynamic analysis takes place will follow the out-of-plane deflection, but in-plane deflection is assumed to be accomplished by shearing, rather than rotation of the 2D cross section. AeroDyn also receives the local freestream (undisturbed) fluid velocities at the tower and blade nodes. (Fluid and structural calculations take place outside of the AeroDyn module and are passed as inputs to AeroDyn by the driver code.) The fluid and structural motions are provided at each coupling time step and then AeroDyn computes the aerodynamic loads on the blade and tower nodes and returns them back to OpenFAST as part of the aero-elastic calculation. In standalone mode, the inputs to AeroDyn are prescribed by a simple driver code, without aero-elastic coupling.

AeroDyn consists of four submodels: (1) rotor wake/induction, (2) blade airfoil aerodynamics, (3) tower influence on the fluid local to the blade nodes, and (4) tower drag. Nacelle, hub, and tail-vane fluid influence and loading, aeroacoustics, and wake and array effects between multiple turbines in a wind plant, are not yet available in AeroDyn v15 and newer.

For operating wind and MHK turbine rotors, AeroDyn calculates the influence of the wake via induction factors based on the quasi-steady Blade-Element/Momentum (BEM) theory, which requires an iterative nonlinear solve (implemented via Brent's method). By quasi-steady, it is meant that the induction reacts instantaneously to loading changes. The induction calculation, and resulting inflow velocities and angles, are based on flow local to each analysis node of each blade, based on the relative velocity between the fluid and structure (including the effects of local inflow skew, shear, turbulence, tower flow disturbances, and structural motion, depending on features enabled). The Glauert's empirical correction (with Buhl's modification) replaces the linear momentum balance at high axial induction factors. In the BEM solution, Prandtl tip-loss, Prandtl hub-loss, and Pitt and Peters skewed-wake are all 3D corrections that can optionally be applied. When the skewed-wake correction is enabled, it is applied after the BEM iteration. Additionally, the calculation of tangential induction (from the angular momentum balance), the use of drag in the axial-induction calculation, and the use of drag in the tangential-induction calculation are all terms that can optionally be included in the BEM iteration (even when drag is not used in the BEM iteration, drag is still used to calculate the nodal loads once the induction has been found). The wake/induction calculation can be bypassed altogether for the purposes of modeling rotors that are parked or idling, in which case the inflow velocity and angle are determined purely geometrically. During linearization analyses with AeroDyn coupled to OpenFAST and BEM enabled, the wake can be assumed to be frozen (i.e., the axial and tangential induces velocities, $-V_x a$ and $V_y a'$, are fixed at their operating-point values during linearization) or the induction can be recalculated during linearization using BEM theory. Dynamic wake that accounts for induction dynamics as a result of transient conditions are not yet available in AeroDyn v15 and newer.

The blade airfoil aerodynamics can be steady or unsteady, except in the case that a cavitation check is requested for MHK, in which case only steady aerodynamics are supported. In the steady model, the supplied static airfoil data — including the lift force, drag force, and optional pitching moment and minimum pressure coefficients versus angle of attack (AoA) — are used directly to calculate nodal loads. The AirfoilPrep preprocessor can be used to generate the needed static airfoil data based on uncorrected 2D data (based, e.g., on airfoil tests in a wind tunnel or XFoil), including features to blend data between different airfoils, apply 3D rotational augmentation, and extrapolate to high AoA. The unsteady airfoil aerodynamic (UA) models account for flow hysteresis, including unsteady attached flow, trailing-edge flow separation, dynamic stall, and flow reattachment. The UA models can be considered as 2D dynamic corrections to the static airfoil response as a result of time-varying inflow velocities and angles. Three semi-empirical UA models are available: the original theoretical developments of Beddoes-Leishman (B-L), extensions to the B-L developed by González, and extensions to the B-L model developed by Minnema/Pierce. **While all of the UA models are documented in this manual, the original B-L model is not yet functional. Testing has shown that the González and Minnema/Pierce models produce reasonable hysteresis of the normal force, tangential force, and pitching-moment coefficients if the UA model parameters are set appropriately for a given airfoil, Reynolds number, and/or Mach number. However, the results will differ a bit from earlier versions of AeroDyn, (which was based on the Minnema/Pierce extensions to B-L) even if the default UA model parameters are used, due to differences in the UA model logic between the versions. We recommend that users run test cases with uniform wind inflow and fixed yaw error (e.g., through the standalone AeroDyn driver) to examine the accuracy of the normal force, tangential force, and pitching-moment coefficient hysteresis and to adjust the UA model parameters appropriately.** The airfoil-, Reynolds-, and Mach-dependent parameters of the UA models may be derived from the static airfoil data. These UA models are valid for small to moderate AoA under normal rotor operation; the steady model is more appropriate under parked or idling conditions. The static airfoil data is always used in the BEM iteration; when UA is enabled, it is applied after the BEM iteration and after the skewed-wake correction. The UA models are not set up to support linearization, so, UA must be disabled during linearization analyses with

AeroDyn coupled to OpenFAST. The interpolation of airfoil data based on Reynolds number or aerodynamic-control setting (e.g., flaps) is not yet available in AeroDyn v15 and newer.

The influence of the tower on the fluid flow local to the blade is based on a potential-flow and/or a tower-shadow model. The potential-flow model uses the analytical potential-flow solution for flow around a cylinder to model the tower dam effect on upwind rotors. In this model, the freestream (undisturbed) flow at each blade node is disturbed based on the location of the blade node relative to the tower and the tower diameter, including lower velocities upstream and downstream of the tower, higher velocities to the left and right of the tower, and cross-stream flow. The Bak correction can optionally be included in the potential-flow model, which augments the tower upstream disturbance and improves the tower wake for downwind rotors based on the tower drag coefficient. The tower shadow model can also be enabled to account for the tower wake deficit on downwind rotors. This model includes an axial flow deficit on the freestream fluid at each blade node dependent on the location of the blade node relative to the tower and the tower diameter and drag coefficient, based on the work of Powles. Both tower-influence models are quasi-steady models, in that the disturbance is applied directly to the freestream fluid at the blade nodes without dynamics, and are applied within the BEM iteration.

The aerodynamic load on the tower is based directly on the tower diameter and drag coefficient and the local relative fluid velocity between the freestream (undisturbed) flow and structure at each tower analysis node (including the effects of local shear, turbulence, and structural motion, depending on features enabled). The tower drag load calculation is quasi-steady and independent from the tower influence on flow models.

The primary AeroDyn input file defines modeling options, environmental conditions (except freestream flow), airfoils, tower nodal discretization and properties, as well as output file specifications. Airfoil data properties are read from dedicated inputs files (one for each airfoil) and include coefficients of lift force, drag force, and optional pitching moment and minimum pressure versus AoA, as well as UA model parameters. (Minimum pressure coefficients versus AoA are also included in the airfoil input files in case that a cavitation check is requested.) Blade nodal discretization, geometry, twist, chord, and airfoil identifier are likewise read from separate input files (one for each blade).

Section 4.2.2 describes the AeroDyn input files. Section 4.2.3 discusses the output files generated by AeroDyn; these include an echo file, summary file, and the results file. Section 4.2.4 provides modeling guidance when using AeroDyn. Example input files are included in Section 4.2.5. A summary of available output channels are found Section 4.2.5.

4.2.2 Input Files

The user configures the aerodynamic model parameters via a primary AeroDyn input file, as well as separate input files for airfoil and blade data. When used in standalone mode, an additional driver input file is required. This driver file specifies initialization inputs normally provided to AeroDyn by OpenFAST, as well as the per-time-step inputs to AeroDyn.

As an example, the `driver.dvr` file is the main driver, the `input.dat` is the primary input file, the `blade.dat` file contains the blade geometry data, and the `airfoil.dat` file contains the airfoil angle of attack, lift, drag, moment coefficients, and pressure coefficients. Example input files are included in Section 4.2.5.

No lines should be added or removed from the input files, except in tables where the number of rows is specified and comment lines in the AeroDyn airfoil data files.

Units

AeroDyn uses the SI system (kg, m, s, N). Angles are assumed to be in radians unless otherwise specified.

AeroDyn Driver Input File

The driver input file is only needed for the standalone version of AeroDyn and contains inputs normally generated by OpenFAST, and necessary to control the aerodynamic simulation for uncoupled models. A sample AeroDyn driver input file is given in Section 4.2.5.

Set the `Echo` flag in this file to TRUE if you wish to have the `AeroDyn_Driver` executable echo the contents of the driver input file (useful for debugging errors in the driver file). The echo file has the naming convention of *OutFileRoot.ech*, where `OutFileRoot` is specified in the I/O SETTINGS section of the driver input file below. `AD_InputFile` is the filename of the primary AeroDyn input file. This name should be in quotations and can contain an absolute path or a relative path.

The TURBINE DATA section defines the AeroDyn-required turbine geometry for a rigid turbine, see Figure 1. `NumBlades` specifies the number of blades; only one-, two-, or three-bladed rotors are permitted. `HubRad` specifies the radius to the blade root from the center-of-rotation along the (possibly preconed) blade-pitch axis; `HubRad` must be greater than zero. `HubHt` specifies the elevation of the hub center above the ground (or above the mean sea level (MSL) for offshore wind turbines or above the seabed for MHK turbines). `Overhang` specifies the distance along the (possibly tilted) rotor shaft between the tower centerline and hub center; `Overhang` is positive downwind, so use a negative number for upwind rotors. `ShftTilt` is the angle (in degrees) between the rotor shaft and the horizontal plane. Positive `ShftTilt` means that the downwind end of the shaft is the highest; upwind turbines have negative `ShftTilt` for improved tower clearance. `Precone` is the angle (in degrees) between a flat rotor disk and the cone swept by the blades, positive downwind; upwind turbines have negative `Precone` for improved tower clearance.

The I/O SETTINGS section controls the creation of the results file. If `OutFileRoot` is specified, the results file will have the filename *OutFileRoot.#.out*, where the '#' character is an integer number corresponding to a test case line found in the COMBINED-CASE ANALYSIS section described below. If an empty string is provided for `OutFileRoot`, then the driver file's root name will be used instead. If `TabDel` is `TRUE`, a TAB character is used between columns in the output file; if FALSE, fixed-width is used otherwise. `OutFmt` is any valid Fortran numeric format string, which is used for text output, excluding the time channel. The resulting field should be 10 characters, but AeroDyn does not check `OutFmt` for validity. If you want a sound generated on program exit, set `Beep` to true.

The COMBINED-CASE ANALYSIS section allows you to execute `NumCases` number of simulations for the given TURBINE DATA with a single driver input file. There will be one row in the subsequent table for each of the `NumCases` specified (plus two table header lines). The information within each row of the table fully specifies each simulation. Each row contains the following columns: `WndSpeed`, `ShearExp`, `RotSpd`, `Pitch`, `Yaw`, `dT`, and `Tmax`. The local undisturbed wind speed for any given blade or tower node is determined using,

$$U(Z) = \mathrm{WndSpeed} \times \left(\frac{Z}{\mathrm{HubHt}}\right)^{\mathrm{ShearExp}} \tag{4.1}$$

where $\mathrm{WndSpeed}$ is the steady wind speed (fluid flow speed in the case of an MHK turbine) located at elevation HubHt, Z is the instantaneous elevation of the blade or tower node above the ground (or above the MSL for offshore wind turbines or above the seabed for MHK turbines), and $\mathrm{ShearExp}$ is the power-law shear exponent. The fixed rotor speed (in rpm) is given by `RotSpd` (positive clockwise looking downwind), the fixed blade-pitch angle (in degrees) is given by `Pitch` (positive to feather, leading edge upwind), and the fixed nacelle-yaw angle (in degrees) is given by `Yaw` (positive rotation of the nacelle about the vertical tower axis, counterclockwise when looking downward). While the flow speed and direction in the AeroDyn driver is uniform and fixed (depending only on elevation above ground), `Yaw` and `ShftTilt` (from the TURBINE DATA section above) can introduce skewed flow. `dT` is the simulation time step, which must match the time step for the aerodynamic calculations (`DTAero`) as specified in the primary AeroDyn input file, and `Tmax` is the total simulation time.

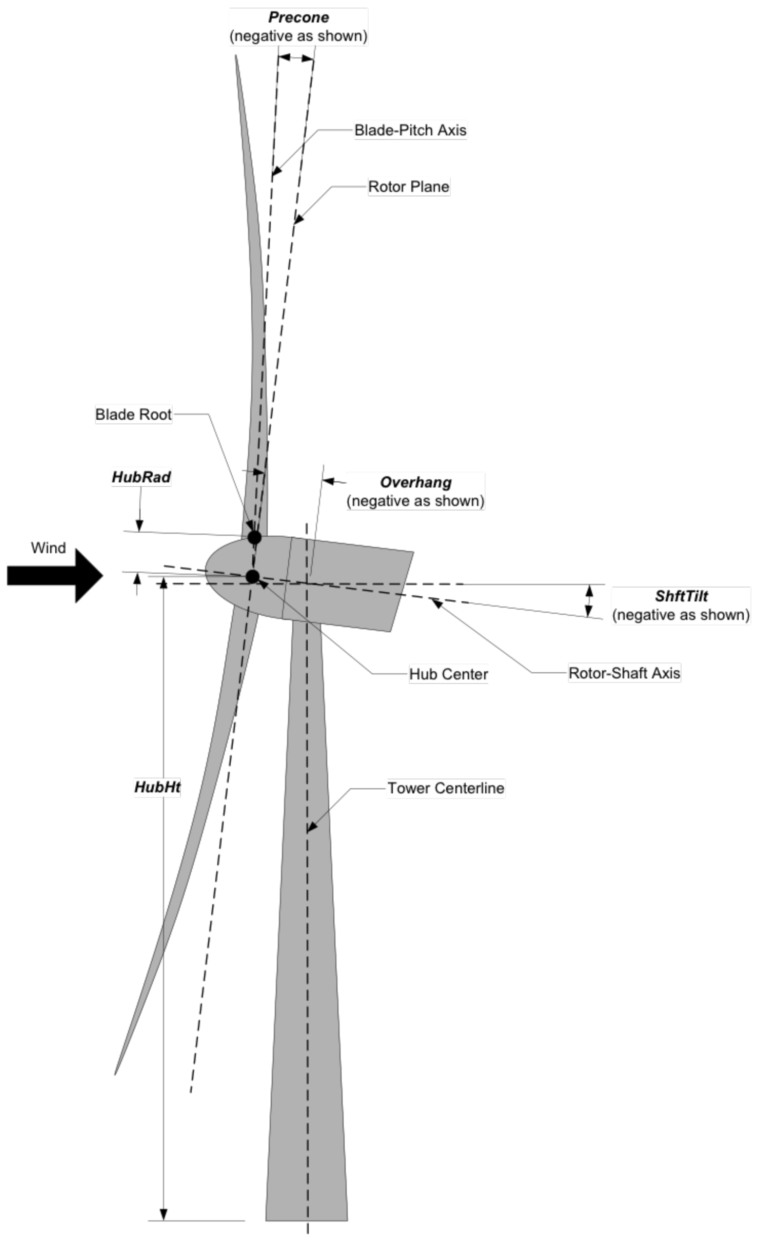


Fig. 4.1: AeroDyn Driver Turbine Geometry

AeroDyn Primary Input File

The primary AeroDyn input file defines modeling options, environmental conditions (except freestream flow), airfoils, tower nodal discretization and properties, as well as output file specifications.

The file is organized into several functional sections. Each section corresponds to an aspect of the aerodynamics model. A sample AeroDyn primary input file is given in Section 4.2.5.

The input file begins with two lines of header information which is for your use, but is not used by the software.

General Options

Set the `Echo` flag to TRUE if you wish to have AeroDyn echo the contents of the AeroDyn primary, airfoil, and blade input files (useful for debugging errors in the input files). The echo file has the naming convention of *OutRootFile.AD.ech*. `OutRootFile` is either specified in the I/O SETTINGS section of the driver input file when running AeroDyn standalone, or by the OpenFAST program when running a coupled simulation.

`DTAero` sets the time step for the aerodynamic calculations. For accuracy and numerical stability, we recommend that `DTAero` be set such that there are at least 200 azimuth steps per rotor revolution. However, when AeroDyn is coupled to OpenFAST, OpenFAST may require time steps much smaller than this rule of thumb. If UA is enabled while using very small time steps, you may need to recompile AeroDyn in double precision to avoid numerical problems in the UA routines. The keyword `DEFAULT` for `DTAero` may be used to indicate that AeroDyn should employ the time step prescribed by the driver code (OpenFAST or the standalone driver program).

Set `WakeMod` to 0 if you want to disable rotor wake/induction effects or 1 to include these effects using the (quasi-steady) BEM theory model. When `WakeMod` is set to 2, a dynamic BEM theory model (DBEMT) is used (also referred to as dynamic inflow or dynamic wake model). When `WakeMod` is set to 3, the free vortex wake model is used, also referred to as OLAF (see Section 4.3). `WakeMod` cannot be set to 2 or 3 during linearization analyses.

Set `AFAeroMod` to 1 to include steady blade airfoil aerodynamics or 2 to enable UA; `AFAeroMod` must be 1 during linearization analyses with AeroDyn coupled to OpenFAST.

Set `TwrPotent` to 0 to disable the potential-flow influence of the tower on the fluid flow local to the blade, 1 to enable the standard potential-flow model, or 2 to include the Bak correction in the potential-flow model.

Set the `TwrShadow` flag to TRUE to include the influence of the tower on the flow local to the blade based on the downstream tower shadow model or FALSE to disable these effects. If the tower influence from potential flow and tower shadow are both enabled, the two influences will be superimposed.

Set the `TwrAero` flag to TRUE to calculate fluid drag loads on the tower or FALSE to disable these effects.

During linearization analyses with AeroDyn coupled OpenFAST and BEM enabled (`WakeMod = 1`), set the `FrozenWake` flag to TRUE to employ frozen-wake assumptions during linearization (i.e. to fix the axial and tangential induces velocities, and, at their operating-point values during linearization) or FALSE to recalculate the induction during linearization using BEM theory.

Set the `CavitCheck` flag to TRUE to perform a cavitation check for MHK turbines or FALSE to disable this calculation. If `CavitCheck` is TRUE, `AFAeroMod` must be set to 1 because the cavitation check does not function with unsteady airfoil aerodynamics.

Set the `CompAA` flag to TRUE to run aero-acoustic calculations. This option is only available for `WakeMod = 1` or `2`. See section Section 4.4 for information on how to use this feature.

The `AA_InputFile` is used to specify the input file for the aeroacoustics sub-module. See Section 4.4 for information on how to use this feature.

Environmental Conditions

AirDens specifies the fluid density and must be a value greater than zero; a typical value is around 1.225 kg/m^3 for air (wind turbines) and 1025 kg/m^3 for seawater (MHK turbines). KinVisc specifies the kinematic viscosity of the air (used in the Reynolds number calculation); a typical value is around 1.460E-5 m^2/s for air (wind turbines) and 1.004E-6 m^2/s for seawater (MHK turbines). SpdSound is the speed of sound in air (used to calculate the Mach number within the unsteady airfoil aerodynamics calculations); a typical value is around 340.3 m/s. The last three parameters in this section are only used when CavitCheck = TRUE for MHK turbines. Patm is the atmospheric pressure above the free surface; typically around 101,325 Pa. Pvap is the vapor pressure of the fluid; for seawater this is typically around 2,000 Pa. FluidDepth is the distance from the hub center to the free surface.

Blade-Element/Momentum Theory Options

The input parameters in this section are not used when WakeMod = 0.

SkewMod determines the skewed-wake correction model. Set SkewMod to 1 to use the uncoupled BEM solution technique without an additional skewed-wake correction. Set SkewMod to 2 to include the Pitt/Peters correction model. **The coupled model ``SkewMod= 3`` is not available in this version of AeroDyn.**

SkewModFactor is used only when SkewMod = 1. Enter a scaling factor to use in the Pitt/Peters correction model, or enter "default" to use the default value of $\frac{15\pi}{32}$.

Set TipLoss to TRUE to include the Prandtl tip-loss model or FALSE to disable it. Likewise, set HubLoss to TRUE to include the Prandtl hub-loss model or FALSE to disable it.

Set TanInd to TRUE to include tangential induction (from the angular momentum balance) in the BEM solution or FALSE to neglect it. Set AIDrag to TRUE to include drag in the axial-induction calculation or FALSE to neglect it. If TanInd = TRUE, set TIDrag to TRUE to include drag in the tangential-induction calculation or FALSE to neglect it. Even when drag is not used in the BEM iteration, drag is still used to calculate the nodal loads once the induction has been found,

IndToler sets the convergence threshold for the iterative nonlinear solve of the BEM solution. The nonlinear solve is in terms of the inflow angle, but IndToler represents the tolerance of the nondimensional residual equation, with no physical association possible. When the keyword DEFAULT is used in place of a numerical value, IndToler will be set to 5E-5 when AeroDyn is compiled in single precision and to 5E-10 when AeroDyn is compiled in double precision; we recommend using these defaults. MaxIter determines the maximum number of iterations steps in the BEM solve. If the residual value of the BEM solve is not less than or equal to IndToler in MaxIter, AeroDyn will exit the BEM solver and return an error message.

Dynamic Blade-Element/Momentum Theory Options

The input parameters in this section are used only when WakeMod = 2.

Set DBEMT_Mod to 1 for the constant-tau1 model, or set DBEMT_Mod to 2 to use a model where tau1 varies with time.

If DBEMT_Mod=1 (constant-tau1 model), set tau1_const to the time constant to use for DBEMT.

OLAF – cOnvecting LAgrangian Filaments (Free Vortex Wake) Theory Options

The input parameters in this section are used only when `WakeMod = 3`.

The settings for the free vortex wake model are set in the OLAF input file described in Section 4.3.4. `OLAFInputFileName` is the filename for this input file.

Unsteady Airfoil Aerodynamics Options

The input parameters in this section are only used when `AFAeroMod = 2`.

`UAMod` determines the UA model. Setting `UAMod` to 1 enables original theoretical developments of B-L, 2 enables the extensions to B-L developed by González, and 3 enables the extensions to B-L developed by Minnema/Pierce. **While all of the UA models are documented in this manual, the original B-L model is not yet functional. Testing has shown that the González and Minnema/Pierce models produce reasonable hysteresis of the normal force, tangential force, and pitching-moment coefficients if the UA model parameters are set appropriately for a given airfoil, Reynolds number, and/or Mach number. However, the results will differ a bit from earlier versions of AeroDyn, (which was based on the Minnema/Pierce extensions to B-L) even if the default UA model parameters are used, due to differences in the UA model logic between the versions. We recommend that users run test cases with uniform inflow and fixed yaw error (e.g., through the standalone AeroDyn driver) to examine the accuracy of the normal force, tangential force, and pitching-moment coefficient hysteresis and to adjust the UA model parameters appropriately.**

`FLookup` determines how the nondimensional separation distance value, *f'*, will be calculated. When `FLookup` is set to TRUE, *f'* is determined via a lookup into the static lift-force coefficient and drag-force coefficient data. **Using best-fit exponential equations (``FLookup = FALSE``) is not yet available, so ``FLookup`` must be ``TRUE`` in this version of AeroDyn.**

Airfoil Information

This section defines the airfoil data input file information. The airfoil data input files themselves (one for each airfoil) include tables containing coefficients of lift force, drag force, and optionally pitching moment, and minimum pressure versus AoA, as well as UA model parameters, and are described in Section 4.2.2.

The first 5 lines in the AIRFOIL INFORMATION section relate to the format of the tables of static airfoil coefficients within each of the airfoil input files. `InCol_Alfa`, `InCol_Cl`, `InCol_Cd`, `InCol_Cm,` and `InCol_Cpmin` are column numbers in the tables containing the AoA, lift-force coefficient, drag-force coefficient, pitching-moment coefficient, and minimum pressure coefficient, respectively (normally these are 1, 2, 3, 4, and 5, respectively). If pitching-moment terms are neglected with `UseBlCm = FALSE`, `InCol_Cm` may be set to zero, and if the cavitation check is disabled with `CavitCheck = FALSE`, `InCol_Cpmin` may be set to zero.

Specify the number of airfoil data input files to be used using `NumAFfiles`, followed by `NumAFfiles` lines of filenames. The file names should be in quotations and can contain an absolute path or a relative path e.g., “C:\airfoils\S809_CLN_298.dat” or “airfoils\S809_CLN_298.dat”. If you use relative paths, it is relative to the location of the current working directory. The blade data input files will reference these airfoil data using their line identifier, where the first airfoil file is numbered 1 and the last airfoil file is numbered `NumAFfiles`.

Rotor/Blade Properties

Set `UseBlCm` to TRUE to include pitching-moment terms in the blade airfoil aerodynamics or FALSE to neglect them; if `UseBlCm = TRUE`, pitching-moment coefficient data must be included in the airfoil data tables with `InCol_Cm` not equal to zero.

The blade nodal discretization, geometry, twist, chord, and airfoil identifier are set in separate input files for each blade, described in Section 4.2.2. `ADBlFile(1)` is the filename for blade 1, `ADBlFile(2)` is the filename for blade 2, and `ADBlFile(3)` is the filename for blade 3, respectively; the latter is not used for two-bladed rotors and the latter two are not used for one-bladed rotors. The file names should be in quotations and can contain an absolute path or a relative path. The data in each file need not be identical, which permits modeling of aerodynamic imbalances.

Tower Influence and Aerodynamics

The input parameters in this section pertain to the tower influence and/or tower drag calculations and are only used when `TwrPotent` > 0, `TwrShadow = TRUE`, or `TwrAero = TRUE`.

`NumTwrNds` is the user-specified number of tower analysis nodes and determines the number of rows in the subsequent table (after two table header lines). `NumTwrNds` must be greater than or equal to two; the higher the number, the finer the resolution and longer the computational time; we recommend that `NumTwrNds` be between 10 and 20 to balance accuracy with computational expense. For each node, `TwrElev` specifies the local elevation of the tower node above ground (or above MSL for offshore wind turbines or above the seabed for MHK turbines), `TwrDiam` specifies the local tower diameter, and `TwrCd` specifies the local tower drag-force coefficient. `TwrElev` must be entered in monotonically increasing order—from the lowest (tower-base) to the highest (tower-top) elevation. See Figure 2.

Outputs

Specifying `SumPrint` to TRUE causes AeroDyn to generate a summary file with name `OutFileRoot**.AD.sum*. ``OutFileRoot` is either specified in the I/O SETTINGS section of the driver input file when running AeroDyn standalone, or by the OpenFAST program when running a coupled simulation. See section 5.2 for summary file details.

AeroDyn can output aerodynamic and kinematic quantities at up to nine nodes along the tower and up to nine nodes along each blade. `NBlOuts` specifies the number of blade nodes that output is requested for (0 to 9) and `BlOutNd` on the next line is a list `NBlOuts` long of node numbers between 1 and `NumBlNds` (corresponding to a row number in the blade analysis node table in the blade data input files), separated by any combination of commas, semicolons, spaces, and/or tabs. All blades have the same output node numbers. `NTwOuts` specifies the number of tower nodes that output is requested for (0 to 9) and `TwOutNd` on the next line is a list `NTwOuts` long of node numbers between 1 and `NumTwrNds` (corresponding to a row number in the tower analysis node table above), separated by any combination of commas, semicolons, spaces, and/or tabs. The outputs specified in the `OutList` section determine which quantities are actually output at these nodes.

The `OutList` section controls output quantities generated by AeroDyn. Enter one or more lines containing quoted strings that in turn contain one or more output parameter names. Separate output parameter names by any combination of commas, semicolons, spaces, and/or tabs. If you prefix a parameter name with a minus sign, "-", underscore, "_", or the characters "m" or "M", AeroDyn will multiply the value for that channel by –1 before writing the data. The parameters are written in the order they are listed in the input file. AeroDyn allows you to use multiple lines so that you can break your list into meaningful groups and so the lines can be shorter. You may enter comments after the closing quote on any of the lines. Entering a line with the string "END" at the beginning of the line or at the beginning of a quoted string found at the beginning of the line will cause AeroDyn to quit scanning for more lines of channel names. Blade and tower node-related quantities are generated for the requested nodes identified through the `BlOutNd` and `TwOutNd` lists above. If AeroDyn encounters an unknown/invalid channel name, it warns the users but will remove the suspect channel from the output file. Please refer to Appendix E for a complete list of possible output parameters.

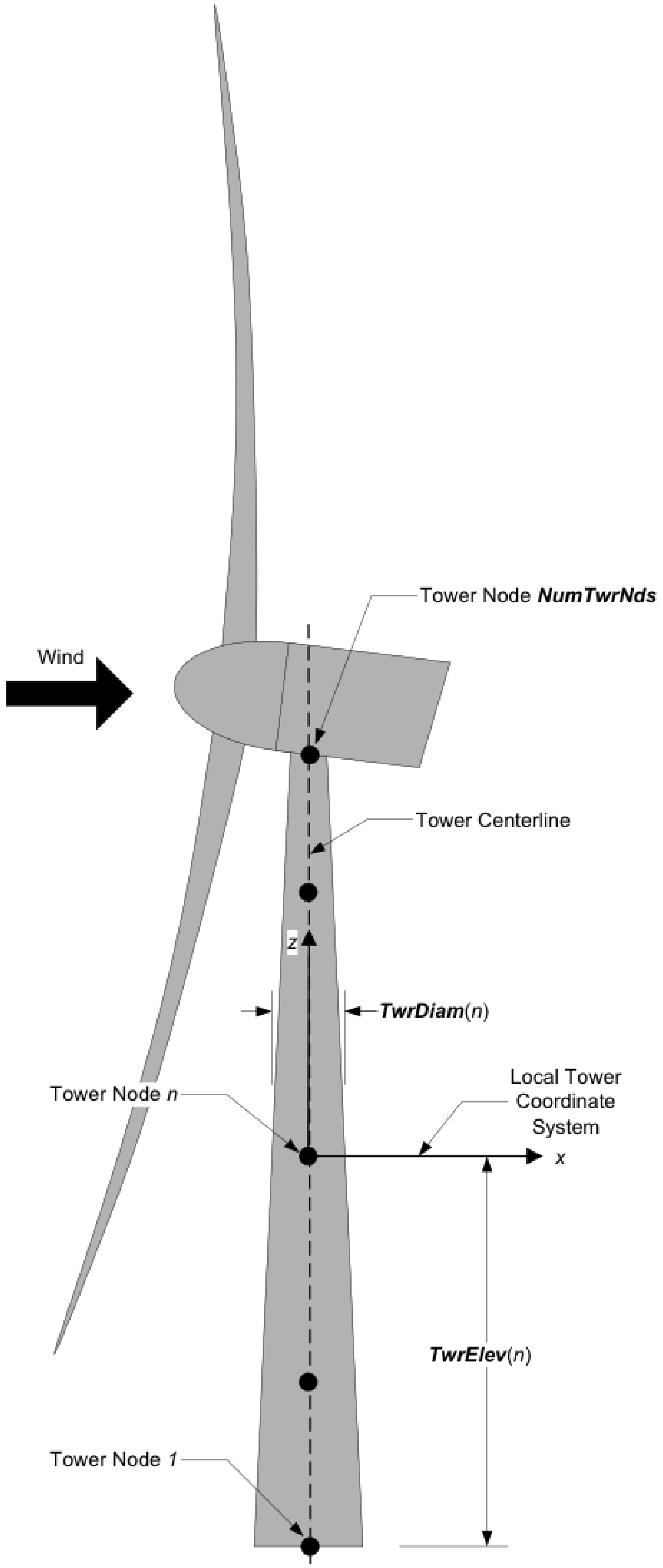


Fig. 4.2: AeroDyn Tower Geometry

Nodal Outputs

In addition to the named outputs in Section 4.2.2 above, AeroDyn allows for outputting the full set blade node motions and loads (tower nodes unavailable at present). Please refer to the AeroDyn_Nodes tab in the Excel file `OutListParameters.xlsx` for a complete list of possible output parameters.

This section follows the *END* statement from normal Outputs section described above, and includes a separator description line followed by the following optinos.

BldNd_BladesOut specifies the number of blades to output. Possible values are 0 through the number of blades AeroDyn is modeling. If the value is set to 1, only blade 1 will be output, and if the value is 2, blades 1 and 2 will be output.

BldNd_BlOutNd specifies which nodes to output. This is currently unused.

The **OutList** section controls the nodal output quantities generated by AeroDyn. In this section, the user specifies the name of the channel family to output. The output name for each channel is then created internally by AeroDyn by combining the blade number, node number, and channel family name. For example, if the user specifies **AxInd** as the channel family name, the output channels will be named with the convention of **B**β**N###AxInd** where β is the blade number, and **###** is the three digit node number.

Sample Nodal Outputs section

This sample includes the `END` statement from the regular outputs section.

```
END of input file (the word "END" must appear in the first 3 columns of this last␣
↪OutList line)
---------------------- NODE OUTPUTS --------------------------------------------
         3   BldNd_BladesOut  - Blades to output
        99   BldNd_BlOutNd   - Blade nodes on each blade (currently unused)
             OutList     - The next line(s) contains a list of output parameters. ␣
↪See OutListParameters.xlsx, AeroDyn_Nodes tab for a listing of available output␣
↪channels, (-)
"VUndx"     - x-component of undisturbed wind velocity at each node
"VUndy"     - y-component of undisturbed wind velocity at each node
"VUndz"     - z-component of undisturbed wind velocity at each node
"VDisx"     - x-component of disturbed wind velocity at each node
"VDisy"     - y-component of disturbed wind velocity at each node
"VDisz"     - z-component of disturbed wind velocity at each node
"STVx"      - x-component of structural translational velocity at each node
"STVy"      - y-component of structural translational velocity at each node
"STVz"      - z-component of structural translational velocity at each node
"VRel"      - Relvative wind speed at each node
"DynP"      - Dynamic pressure at each node
"Re"        - Reynolds number (in millions) at each node
"M"         - Mach number at each node
"Vindx"     - Axial induced wind velocity at each node
"Vindy"     - Tangential induced wind velocity at each node
"AxInd"     - Axial induction factor at each node
"TnInd"     - Tangential induction factor at each node
"Alpha"     - Angle of attack at each node
"Theta"     - Pitch+Twist angle at each node
"Phi"       - Inflow angle at each node
"Curve"     - Curvature angle at each node
"Cl"        - Lift force coefficient at each node
"Cd"        - Drag force coefficient at each node
"Cm"        - Pitching moment coefficient at each node
```

(continues on next page)

(continued from previous page)

```
"Cx"        - Normal force (to plane) coefficient at each node
"Cy"        - Tangential force (to plane) coefficient at each node
"Cn"        - Normal force (to chord) coefficient at each node
"Ct"        - Tangential force (to chord) coefficient at each node
"Fl"        - Lift force per unit length at each node
"Fd"        - Drag force per unit length at each node
"Mm"        - Pitching moment per unit length at each node
"Fx"        - Normal force (to plane) per unit length at each node
"Fy"        - Tangential force (to plane) per unit length at each node
"Fn"        - Normal force (to chord) per unit length at each node
"Ft"        - Tangential force (to chord) per unit length at each node
"Clrnc"     - Tower clearance at each node (based on the absolute distance to the␣
↪nearest point in the tower from blade node B#N# minus the local tower radius, in␣
↪the deflected configuration); please note that this clearance is only approximate␣
↪because the calculation assumes that the blade is a line with no volume (however,␣
↪the calculation does use the local tower radius); when blade node B#N# is above the␣
↪tower top (or below the tower base), the absolute distance to the tower top (or␣
↪base) minus the local tower radius, in the deflected configuration, is output
"Vx"        - Local axial velocity
"Vy"        - Local tangential velocity
"GeomPhi"   - Geometric phi? If phi was solved using normal BEMT equations, GeomPhi =␣
↪1; otherwise, if it was solved geometrically, GeomPhi = 0.
"Chi"       - Skew angle (used in skewed wake correction) -- not available for OLAF
"UA_Flag"   - Flag indicating if UA is turned on for this node. -- not available for␣
↪OLAF
"CpMin"     - Pressure coefficient
"SgCav"     - Cavitation number
"SigCr"     - Critical cavitation number
"Gam"       - Gamma -- circulation on blade
"Cl_Static" - Static portion of lift force coefficient at each node, without ␣
↪unsteady effects -- not available for BEMT/DBEMT
"Cd_Static" - Static portion of drag force coefficient at each node, without unsteady␣
↪effects -- not available for BEMT/DBEMT
"Cm_Static" - Static portion of pitching moment coefficient at each node, without␣
↪unsteady effects -- not available for BEMT/DBEMT
"Uin"       - Axial induced velocity in rotating hub coordinates. Axial aligned with␣
↪hub axis.     rotor plane polar hub rotating coordinates
"Uit"       - Tangential induced velocity in rotating hub coordinates. Tangential to␣
↪the rotation plane. Perpendicular to blade aziumth.     rotor plane polar hub␣
↪rotating coordinates
"Uir"       - Radial induced velocity in rotating hub coordinates. Radial outwards in␣
↪rotation plane. Aligned with blade azimuth.     rotor plane polar hub rotating␣
↪coordinates
END of input file (the word "END" must appear in the first 3 columns of this last␣
↪OutList line)
-------------------------------------------------------------------------------------
↪-
```

Airfoil Data Input File

The airfoil data input files themselves (one for each airfoil) include tables containing coefficients of lift force, drag force, and pitching moment versus AoA, as well as UA model parameters. In these files, any line whose first non-blank character is an exclamation point (!) is ignored (for inserting comment lines). The non-comment lines should appear within the file in order, but comment lines may be intermixed as desired for reading clarity. A sample airfoil data input file is given Section 4.2.5.

`InterpOrd` is the order the static airfoil data is interpolated when AeroDyn uses table look-up to find the lift-, drag-, and optional pitching-moment, and minimum pressure coefficients as a function of AoA. When `InterpOrd` is 1, linear interpolation is used; when `InterpOrd` is 3, the data will be interpolated with cubic splines; if the keyword `DEFAULT` is entered in place of a numerical value, `InterpOrd` is set to 3.

`NonDimArea` is the nondimensional airfoil area (normalized by the local `BlChord` squared), but is currently unused by AeroDyn. `NumCoords` is the number of points to define the exterior shape of the airfoil, plus one point to define the aerodynamic center, and determines the number of rows in the subsequent table; `NumCoords` must be exactly zero or greater than or equal to three. For each point, the nondimensional *X* and *Y* coordinates are specified in the table, `X_Coord` and `Y_Coord` (normalized by the local `BlChord`). The first point must always locate the aerodynamic center (reference point for the airfoil lift and drag forces, likely not on the surface of the airfoil); the remaining points should define the exterior shape of the airfoil. The airfoil shape is currently unused by AeroDyn, but when AeroDyn is coupled to OpenFAST, the airfoil shape will be used by OpenFAST for blade surface visualization when enabled.

Specify the number of Reynolds number- or aerodynamic-control setting-dependent tables of data for the given airfoil via the `NumTabs` setting. The remaining parameters in the airfoil data input files are entered separately for each table.

`Re` and `UserProp` are the Reynolds number (in millions) and aerodynamic-control (or user property) setting for the included table. These values are used only when the `AFTabMod` parameter in the primary AeroDyn input file is set to use 2D interpolation based on `Re` or `UserProp`. If 1D interpolation (based only on angle of attack) is used, only the first table in the file will be used.

Set `InclUAdata` to TRUE if you are including the 32 UA model parameters (required when `AFAeroMod = 2` in the AeroDyn primary input file):

- `alpha0` specifies the zero-lift AoA (in degrees);
- `alpha1` specifies the AoA (in degrees) larger than `alpha0` for which *f* equals 0.7; approximately the positive stall angle;
- `alpha2` specifies the AoA (in degrees) less than `alpha0` for which *f* equals 0.7; approximately the negative stall angle;
- `eta_e` is the recovery factor and typically has a value in the range [0.85 to 0.95] for `UAMod = 1`; if the keyword `DEFAULT` is entered in place of a numerical value, `eta_e` is set to 0.9 for `UAMod = 1`, but `eta_e` is set to 1.0 for other `UAMod` values and whenever `FLookup = TRUE`;
- `C_nalpha` is the slope of the 2D normal force coefficient curve in the linear region;
- `T_f0` is the initial value of the time constant associated with *Df* in the expressions of *Df* and *f'*; if the keyword `DEFAULT` is entered in place of a numerical value, `T_f0` is set to 3.0;
- `T_V0` is the initial value of the time constant associated with the vortex lift decay process, used in the expression of `Cvn`; it depends on Reynolds number, Mach number, and airfoil; if the keyword `DEFAULT` is entered in place of a numerical value, `T_V0` is set to 6.0;
- `T_p` is the boundary-layer leading edge pressure gradient time constant in the expression for *Dp* and should be tuned based on airfoil experimental data; if the keyword `DEFAULT` is entered in place of a numerical value, `T_p` is set to 1.7;
- `T_VL` is the time constant associated with the vortex advection process, representing the nondimensional time in semi-chords needed for a vortex to travel from the leading to trailing edges, and used in the expression of

Cvn; it depends on Reynolds number, Mach number (weakly), and airfoil; valued values are in the range [6 to 13]; if the keyword `DEFAULT` is entered in place of a numerical value, `T_VL` is set to 11.0;

- `b1` is a constant in the expression of ϕ_α^c and ϕ_q^c; this value is relatively insensitive for thin airfoils, but may be different for turbine airfoils; if the keyword `DEFAULT` is entered in place of a numerical value, `b1` is set to 0.14, based on experimental results;
- `b2` is a constant in the expression of ϕ_α^c and ϕ_q^c; this value is relatively insensitive for thin airfoils, but may be different for turbine airfoils; if the keyword `DEFAULT` is entered in place of a numerical value, `b2` is set to 0.53, based on experimental results;
- `b5` is a constant in the expression of $K_q^{'''}$, Cm_q^{nc}, and K_{m_q}; if the keyword `DEFAULT` is entered in place of a numerical value, `b5` is set to 5, based on experimental results;
- `A1` is a constant in the expression ϕ_α^c and ϕ_q^c; this value is relatively insensitive for thin airfoils, but may be different for turbine airfoils; if the keyword `DEFAULT` is entered in place of a numerical value, `A1` is set to 0.3, based on experimental results;
- `A2` is a constant in the expression ϕ_α^c and ϕ_q^c; this value is relatively insensitive for thin airfoils, but may be different for turbine airfoils; if the keyword `DEFAULT` is entered in place of a numerical value, `A2` is set to 0.7, based on experimental results;
- `A5` is a constant in the expression $K_q^{'''}$, Cm_q^{nc}, and K_{m_q}; if the keyword `DEFAULT` is entered in place of a numerical value, `A5` is set to 1, based on experimental results;
- `S1` is the constant in the best fit curve of *f* for `alpha0` $\leq$ AoA $\leq$ `alpha1` for `UAMod = 1` (and is unused otherwise); by definition, it depends on the airfoil;
- `S2` is the constant in the best fit curve of *f* for AoA > `alpha1` for `UAMod = 1` (and is unused otherwise); by definition, it depends on the airfoil;
- `S3` is the constant in the best fit curve of *f* for `alpha2` $\leq$ AoA $\leq$ `alpha0` for `UAMod = 1` (and is unused otherwise); by definition, it depends on the airfoil;
- `S4` is the constant in the best fit curve of *f* for AoA < `alpha2` for `UAMod = 1` (and is unused otherwise); by definition, it depends on the airfoil;
- `Cn1` is the critical value of C_n' at leading-edge separation for positive AoA and should be extracted from airfoil data at a given Reynolds number and Mach number; `Cn1` can be calculated from the static value of *Cn* at either the break in the pitching moment or the loss of chord force at the onset of stall; `Cn1` is close to the condition of maximum lift of the airfoil at low Mach numbers;
- `Cn2` is the critical value of C_n' at leading-edge separation for negative AoA and should be extracted from airfoil data at a given Reynolds number and Mach number; `Cn2` can be calculated from the static value of *Cn* at either the break in the pitching moment or the loss of chord force at the onset of stall; `Cn2` is close to the condition of maximum lift of the airfoil at low Mach numbers;
- `St_sh` is the Strouhal's shedding frequency; if the keyword `DEFAULT` is entered in place of a numerical value, `St_sh` is set to 0.19;
- `Cd0` is the drag-force coefficient at zero-lift AoA;
- `Cm0` is the pitching-moment coefficient about the quarter-chord location at zero-lift AoA, positive for nose up;
- `k0` is a constant in the best fit curve of $\hat{x}_{cp}$ and equals for $\hat{x}_{AC} - 0.25$ `UAMod = 1` (and is unused otherwise);
- `k1` is a constant in the best fit curve of $\hat{x}_{cp}$ for `UAMod = 1` (and is unused otherwise);
- `k2` is a constant in the best fit curve of $\hat{x}_{cp}$ for `UAMod = 1` (and is unused otherwise);
- `k3` is a constant in the best fit curve of $\hat{x}_{cp}$ for `UAMod = 1` (and is unused otherwise);
- `k1_hat` is a constant in the expression of *Cc* due to leading-edge vortex effects for `UAMod = 1` (and is unused otherwise);

- `x_cp_bar` is a constant in the expression of $\hat{x}^{\nu}_{cp}$ for `UAMod = 1` (and is unused otherwise); if the keyword `DEFAULT` is entered in place of a numerical value, `x_cp_bar` is set to 0.2; and
- `UACutOut` is the AoA (in degrees) in absolute value above which UA are disabled; if the keyword `DEFAULT` is entered in place of a numerical value, `UACutOut` is set to 45.
- `filtCutOff` is the cut-off frequency (-3 dB corner frequency) (in Hz) of the low-pass filter applied to the AoA input to UA, as well as to the pitch rate and pitch acceleration derived from AoA within UA; if the keyword `DEFAULT` is entered in place of a numerical value, `filtCutOff` is set to 20.

`NumAlf` is the number of distinct AoA entries and determines the number of rows in the subsequent table of static airfoil coefficients; `NumAlf` must be greater than or equal to one (`NumAlf = 1` implies constant coefficients, regardless of the AoA).

AeroDyn will interpolate on AoA using the data provided via linear interpolation or via cubic splines, depending on the setting of input `InterpOrd` above. If `AFTabMod` is set to `1`, only the first airfoil table in each file will be used. If `AFTabMod` is set to `2`, AeroDyn will find the airfoil table that bounds the computed Reynolds number, and linearly interpolate between the tables, using the logarithm of the Reynolds numbers.

For each AoA, you must set the AoA (in degrees), `alpha`, the lift-force coefficient, `Coefs`(:,1), the drag-force coefficient, `Coefs(:,2)`, and optionally the pitching-moment coefficient, `Coefs(:,3)`, and minimum pressure coefficient, `Coefs(:,4)`, but the column order depends on the settings of `InCol_Alfa`, `InCol_Cl`, `InCol_Cd`, `InCol_Cm`, and `InCol_Cpmin` in the AIRFOIL INFORMATION section of the AeroDyn primary input file. AoA must be entered in monotonically increasing order—from lowest to highest AoA—and the first row should be for AoA = –180 and the last should be for AoA = +180 (unless `NumAlf = 1`, in which case AoA is unused). If pitching-moment terms are neglected with `UseBlCm = FALSE` in the ROTOR/BLADE PROPERTIES section of the AeroDyn primary input file, the column containing pitching-moment coefficients may be absent from the file. Likewise, if the cavitation check is neglected with `CavitCheck = FALSE` in the GENERAL OPTIONS section of the AeroDyn primary input file, the column containing the minimum pressure coefficients may be absent from the file.

Blade Data Input File

The blade data input file contains the nodal discretization, geometry, twist, chord, and airfoil identifier for a blade. Separate files are used for each blade, which permits modeling of aerodynamic imbalances. A sample blade data input file is given in Section 4.2.5.

The input file begins with two lines of header information which is for your use, but is not used by the software.

`NumBlNds` is the user-specified number of blade analysis nodes and determines the number of rows in the subsequent table (after two table header lines). `NumBlNds` must be greater than or equal to two; the higher the number, the finer the resolution and longer the computational time; we recommend that `NumBlNds` be between 10 and 20 to balance accuracy with computational expense. Even though `NumBlNds` is defined in each blade file, all blades must have the same number of nodes. For each node:

- `BlSpn` specifies the local span of the blade node along the (possibly preconed) blade-pitch axis from the root; `BlSpn` must be entered in monotonically increasing order—from the most inboard to the most outboard—and the first node must be zero, and when AeroDyn is coupled to OpenFAST, the last node should be located at the blade tip;
- `BlCrvAC` specifies the local out-of-plane offset (when the blade-pitch angle is zero) of the aerodynamic center (reference point for the airfoil lift and drag forces), normal to the blade-pitch axis, as a result of blade curvature; `BlCrvAC` is positive downwind; upwind turbines have negative `BlCrvAC` for improved tower clearance;
- `BlSwpAC` specifies the local in-plane offset (when the blade-pitch angle is zero) of the aerodynamic center (reference point for the airfoil lift and drag forces), normal to the blade-pitch axis, as a result of blade sweep; positive `BlSwpAC` is opposite the direction of rotation;

- `BlCrvAng` specifies the local angle (in degrees) from the blade-pitch axis of a vector normal to the plane of the airfoil, as a result of blade out-of-plane curvature (when the blade-pitch angle is zero); `BlCrvAng` is positive downwind; upwind turbines have negative `BlCrvAng` for improved tower clearance;
- `BlTwist` specifies the local aerodynamic twist angle (in degrees) of the airfoil; it is the orientation of the local chord about the vector normal to the plane of the airfoil, positive to feather, leading edge upwind; the blade-pitch angle will be added to the local twist;
- `BlChord` specifies the local chord length; and
- `BlAFID` specifies which airfoil data the local blade node is associated with; valid values are numbers between 1 and `NumAFfiles` (corresponding to a row number in the airfoil file table in the AeroDyn primary input file); multiple blade nodes can use the same airfoil data.

See Fig. 4.3. Twist is shown in Fig. 4.4 of Section 4.2.5.

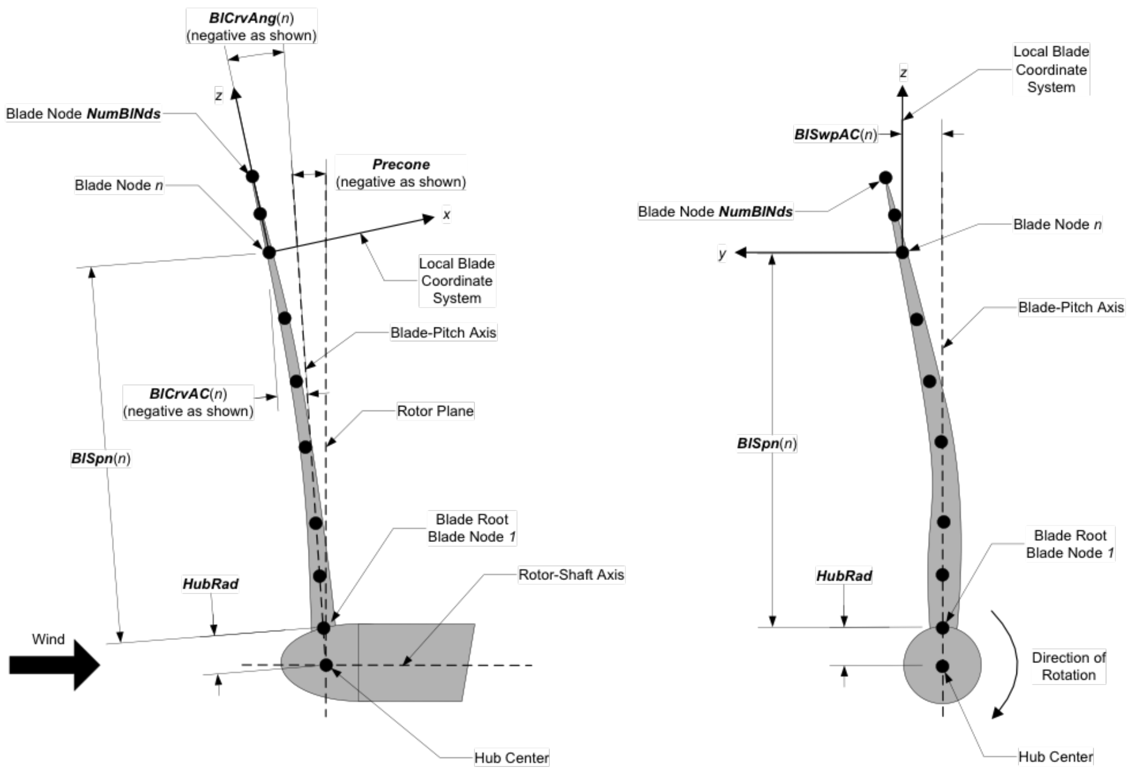


Fig. 4.3: AeroDyn Blade Geometry – Left: Side View; Right: Front View (Looking Downwind)

4.2.3 Output Files

AeroDyn produces three types of output files: an echo file, a summary file, and a time-series results file. The following sections detail the purpose and contents of these files.

Echo Files

If you set the `Echo` flag to `TRUE` in the AeroDyn driver file or the AeroDyn primary input file, the contents of those files will be echoed to a file with the naming conventions, *OutFileRoot.ech* for the driver input file and *OutFileRoot.AD.ech* for the AeroDyn primary input file. `OutFileRoot` is either specified in the I/O SETTINGS section of the driver input file when running AeroDyn standalone, or by the FAST program when running a coupled simulation. The echo files are helpful for debugging your input files. The contents of an echo file will be truncated if AeroDyn encounters an error while parsing an input file. The error usually corresponds to the line after the last successfully echoed line.

Summary File

AeroDyn generates a summary file with the naming convention, *OutFileRoot.AD.sum* if the `SumPrint` parameter is set to `TRUE`. `OutFileRoot` is either specified in the I/O SETTINGS section of the driver input file when running AeroDyn standalone, or by the FAST program when running a coupled simulation. This file summarizes key information about your aerodynamics model, including which features have been enabled and what outputs have been selected.

Results Files

In standalone mode, the AeroDyn time-series results (a separate file for each case) are written to text-based files with the naming convention *OutFileRoot.#.out*, where `OutFileRoot` is specified in the I/O SETTINGS section of the driver input file and the '#' character is an integer number corresponding to a test case line found in the COMBINED-CASE ANALYSIS section. If AeroDyn is coupled to FAST, then FAST will generate a master results file that includes the AeroDyn results and AeroDyn will not write out its own results. The results are in table format, where each column is a data channel (the first column always being the simulation time), and each row corresponds to a simulation output time step. The data channels are specified in the OUTPUTS section of the AeroDyn primary input file. The column format of the AeroDyn-generated files is specified using the `OutFmt` parameter of the driver input file.

4.2.4 Modeling Considerations

AeroDyn was designed as an extremely flexible tool for modeling a wide-range of aerodynamic conditions and turbine configurations. This section provides some general guidance to help you construct models that are compatible with AeroDyn.

Please refer to the theory of Section 7 for detailed information about the implementation approach we have followed in AeroDyn.

Standalone AeroDyn Driver

The standalone AeroDyn driver code is very useful for computing turbine aerodynamics independent of aero-elastic coupling. The standalone AeroDyn driver code essentially replaces the functionality previously available in the separate wind turbine rotor-performance tool WT_Perf. For example, the standalone AeroDyn driver code can be used to compute the surfaces of power coefficient (C_P), thrust coefficient (C_T), and/or torque coefficient (C_Q) as a function of tip-speed ratio (TSR) and blade-pitch angle for a given rotor. Moreover, the standalone AeroDyn driver code is more powerful than WT_Perf in that the standalone AeroDyn driver can capture time-varying dynamics as a result of nacelle-yaw error, shaft tilt, and/or wind shear.

Environmental Conditions

For air, typical values for `AirDens`, `KinVisc`, `SpdSound`, and `Patm` are around 1.225 kg/m^3, 1.460E-5 m^2/s, 340.3 m/s, and 101,325 Pa, respectively. For seawater, typical values for `AirDens`, `KinVisc`, and `Pvap` are around 1025 kg/m^3, 1.004E-6 m^2/s, and 2000 Pa, respectively.

Temporal and Spatial Discretization

For accuracy and numerical stability, we recommend that `DTAero` be set such that there are at least 200 azimuth steps per rotor revolution. However, when AeroDyn is coupled to FAST, FAST may require time steps much smaller than this rule of thumb. If UA is enabled while using very small time steps, you may need to recompile AeroDyn in double precision to avoid numerical problems in the UA routines.

For the blade and tower spatial discretization, using higher number of analysis nodes will result in a more accurate solution at the expense of longer computational time. When AeroDyn is coupled to FAST, the blade and tower analysis node discretization may be independent from the discretization of the nodes in the structural modules.

We recommend that `NumBlNds` be between 10 and 20 to balance accuracy with computational expense for the rotor aerodynamic load calculation. It may be beneficial to use a finer resolution of nodes where large gradients are expected in the aerodynamic loads e.g. near the blade tip. Aerodynamic imbalances are possible through the use of geometrical differences between each blade.

When the tower potential-flow (`TwrPotent > 0`), tower shadow (`TwrShadow = TRUE`), and/or the tower aerodynamic load (`TwrAero = TRUE`) models are enabled, we also recommend that `NumTwrNds` be between 10 and 20 to balance accuracy with computational expense. Normally the local elevation of the tower node above ground (or above MSL for offshore wind turbines or above the seabed for MHK turbines) (`TwrElev`), must be entered in monotonically increasing order from the lowest (tower-base) to the highest (tower-top) elevation. However, when AeroDyn is coupled to FAST, the tower-base node in AeroDyn cannot be set lower than the lowest point where wind is specified in the InflowWind module. To avoid truncating the lower section of the tower in AeroDyn, we recommend that the wind be specified in InflowWind as low to the ground (or MSL for offshore wind turbines or above the seabed for MHK turbines) as possible (this is a particular issue for full-field wind file formats).

Model Options Under Operational and Parked/Idling Conditions

To model an operational rotor, we recommend to include the dynamic BEM model (`WakeMod = 2`) and UA (`AFAeroMod = 2`). Normally, the Pitt and Peters skewed-wake (`SkewMod = 2`), Prandtl tip-loss (`TipLoss = TRUE`), Prandtl hub-loss (`HubLoss = TRUE`), and tangential induction (`TanInd = TRUE`) models should all be enabled, but `SkewMod = 2` is invalid for very large yaw errors (much greater than 45 degrees). The nonlinear solve in the BEM solution is in terms of the inflow angle, but `IndToler` represents the tolerance of the nondimensional residual equation, with no physical association possible; we recommend setting `IndToler` to `DEFAULT`.

While all of the UA models are documented in this manual, the original B-L model is not yet functional. Testing has shown that the González and Minnema/Pierce models produce reasonable hysteresis of the normal force, tangential force, and pitching-moment coefficients if the UA model parameters are set appropriately for a given airfoil, Reynolds *number, and/or Mach number. However, the results will differ a bit from earlier versions of AeroDyn, (which was based on the Minnema/Pierce extensions to B-L) even if the default UA model parameters are used, due to differences in the UA model logic between the versions. We recommend that users run test cases with uniform inflow and fixed yaw error (e.g., through the standalone AeroDyn driver) to examine the accuracy of the normal force, tangential force, and pitching-moment coefficient hysteresis and to adjust the UA model parameters appropriately.*

To model a parked or idling rotor, we recommend to disable induction (`WakeMod = 0`) and UA (`AFAeroMod = 1`), in which case the inflow velocity and angle are determined purely geometrically and the airfoil data is determined statically.

The direct aerodynamic load on the tower often dominates the aerodynamic load on the rotor for parked or idling conditions above the cut-out wind speed, in which case we recommend that `TwrAero = TRUE`. Otherwise, `TwrAero = FALSE` may be satisfactory.

We recommend to include the influence of the tower on the fluid local to the blade for both operational and parked/idling rotors. We recommend that `TwrPotent > 0` for upwind rotors and that `TwrPotent = 2` or `TwrShadow = TRUE` for downwind rotors.

Linearization

When coupled to FAST, AeroDyn can be linearized as part of the linearization of the full coupled solution. When induction is enabled (`WakeMod = 1`), we recommend to base the linearized solution on the frozen-wake assumption, by setting `FrozenWake = TRUE`. The UA models are not set up to support linearization, so, UA must be disabled during linearization by setting `AFAeroMod = 1`.

4.2.5 Appendix

AeroDyn Input Files

In this appendix we describe the AeroDyn input-file structure and provide examples.

1) AeroDyn Driver Input File `(driver input file example)`:

The driver input file is only needed for the standalone version of AeroDyn and contains inputs normally generated by OpenFAST, and necessary to control the aerodynamic simulation for uncoupled models.

2) AeroDyn Primary Input File `(primary input file example)`:

The primary AeroDyn input file defines modeling options, environmental conditions (except freestream flow), airfoils, tower nodal discretization and properties, as well as output file specifications.

The file is organized into several functional sections. Each section corresponds to an aspect of the aerodynamics model.

The input file begins with two lines of header information which is for your use, but is not used by the software.

3) Airfoil Data Input File `(airfoil data input file example)`:

The airfoil data input files themselves (one for each airfoil) include tables containing coefficients of lift force, drag force, and pitching moment versus AoA, as well as UA model parameters. In these files, any line whose first non-blank character is an exclamation point (!) is ignored (for inserting comment lines). The non-comment lines should appear within the file in order, but comment lines may be intermixed as desired for reading clarity.

4) Balde Data Input File `(blade data input file example)`:

The blade data input file contains the nodal discretization, geometry, twist, chord, and airfoil identifier for a blade. Separate files are used for each blade, which permits modeling of aerodynamic imbalances.

AeroDyn List of Output Channels

This is a list of all possible output parameters for the AeroDyn module. The names are grouped by meaning, but can be ordered in the OUTPUTS section of the AeroDyn input file as you see fit. `BN`, refers to output node of blade , where is a number in the range [1,3] and is a number in the range [1,9], corresponding to entry in the `BlOutNd` list. `TwN` refers to output node of the tower and is in the range [1,9], corresponding to entry in the `TwOutNd` list.

The local tower coordinate system is shown in Fig. 4.2 and the local blade coordinate system is shown in Fig. 4.4 below. Figure Fig. 4.4 also shows the direction of the local angles and force components.

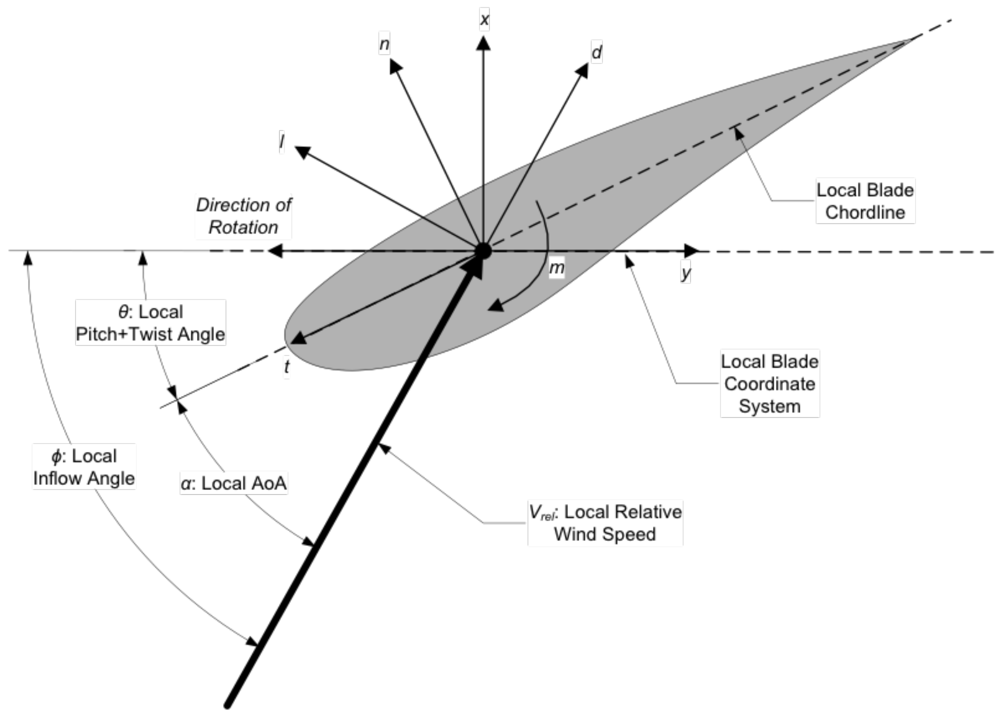
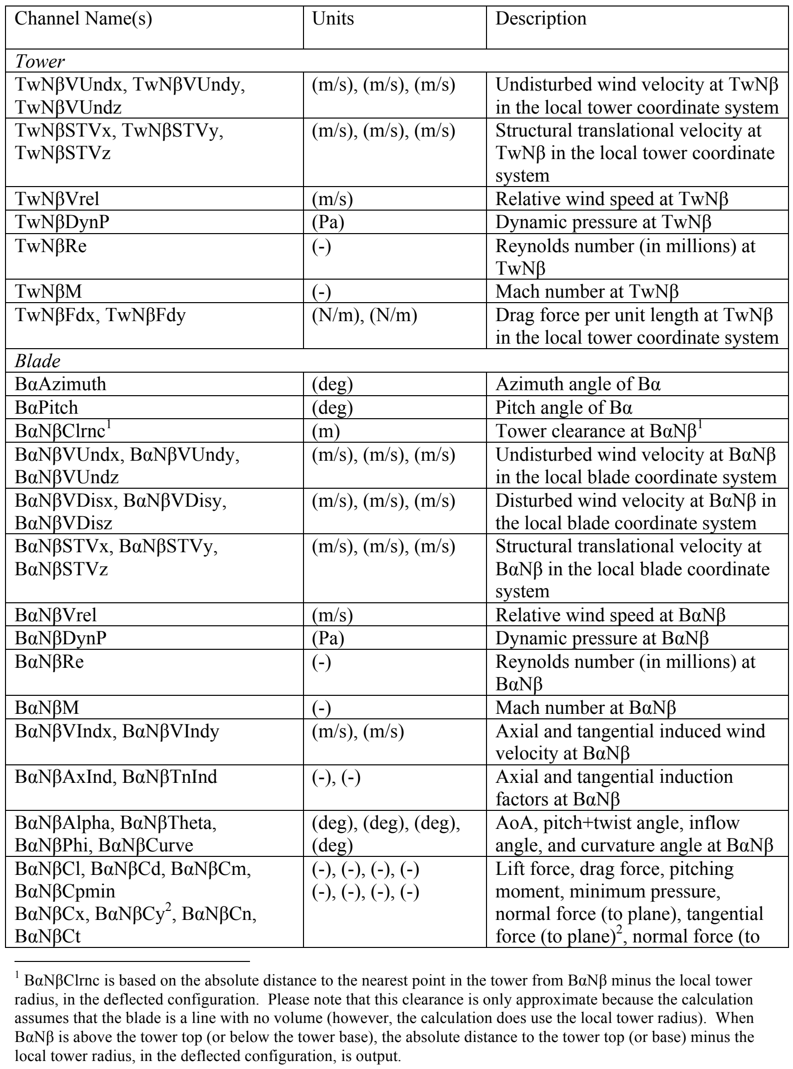

x
n
d
l
Direction of Rotation
Local Blade Chordline
m
y
θ: Local Pitch+Twist Angle
t
Local Blade Coordinate System
ϕ: Local Inflow Angle
α: Local AoA
V_{rel}: Local Relative Wind Speed

Fig. 4.4: AeroDyn Local Blade Coordinate System (Looking Toward the Tip, from the Root) – l: Lift, d: Drag, m: Pitching, x: Normal (to Plane), y: Tangential (to Plane), n: Normal (to Chord), and t: Tangential (to Chord)

4.3 OLAF User's Guide and Theory Manual (Free Vortex Wake in AeroDyn15)

4.3.1 Introduction

Over the past few decades, substantial reductions in the cost of wind energy have come from large increases in rotor size. One important consideration for such large turbines is increased blade flexibility. In particular, large blade deflections may lead to a swept area that deviates significantly from the rotor plane. Such deviations violate assumptions used by common aerodynamic models, such as the blade element momentum (BEM) method. Such methods rely on actuator-disk assumptions that are only valid for axisymmetric rotor loads contained in a plane. Large blade deflections may also cause near wake of the turbine to diverge from a uniform helical shape. Further, interactions between turbine blades and the local near wake may increase, thus violating assumptions of models that do not account for the position and dynamics of the near wake. Additionally, highly flexible blades will likely cause increased unsteadiness and three-dimensionality of aerodynamic effects, increasing the importance of accurate and robust dynamic stall models. There are many other complex wind turbine situations that violate simple engineering assumptions. Such



Channel Name(s)	Units	Description
Tower		
TwNβVUndx, TwNβVUndy, TwNβVUndz	(m/s), (m/s), (m/s)	Undisturbed wind velocity at TwNβ in the local tower coordinate system
TwNβSTVx, TwNβSTVy, TwNβSTVz	(m/s), (m/s), (m/s)	Structural translational velocity at TwNβ in the local tower coordinate system
TwNβVrel	(m/s)	Relative wind speed at TwNβ
TwNβDynP	(Pa)	Dynamic pressure at TwNβ
TwNβRe	(-)	Reynolds number (in millions) at TwNβ
TwNβM	(-)	Mach number at TwNβ
TwNβFdx, TwNβFdy	(N/m), (N/m)	Drag force per unit length at TwNβ in the local tower coordinate system
Blade		
BαAzimuth	(deg)	Azimuth angle of Bα
BαPitch	(deg)	Pitch angle of Bα
BαNβClrnc[1]	(m)	Tower clearance at BαNβ[1]
BαNβVUndx, BαNβVUndy, BαNβVUndz	(m/s), (m/s), (m/s)	Undisturbed wind velocity at BαNβ in the local blade coordinate system
BαNβVDisx, BαNβVDisy, BαNβVDisz	(m/s), (m/s), (m/s)	Disturbed wind velocity at BαNβ in the local blade coordinate system
BαNβSTVx, BαNβSTVy, BαNβSTVz	(m/s), (m/s), (m/s)	Structural translational velocity at BαNβ in the local blade coordinate system
BαNβVrel	(m/s)	Relative wind speed at BαNβ
BαNβDynP	(Pa)	Dynamic pressure at BαNβ
BαNβRe	(-)	Reynolds number (in millions) at BαNβ
BαNβM	(-)	Mach number at BαNβ
BαNβVIndx, BαNβVIndy	(m/s), (m/s)	Axial and tangential induced wind velocity at BαNβ
BαNβAxInd, BαNβTnInd	(-), (-)	Axial and tangential induction factors at BαNβ
BαNβAlpha, BαNβTheta, BαNβPhi, BαNβCurve	(deg), (deg), (deg), (deg)	AoA, pitch+twist angle, inflow angle, and curvature angle at BαNβ
BαNβCl, BαNβCd, BαNβCm, BαNβCpmin BαNβCx, BαNβCy[2], BαNβCn, BαNβCt	(-), (-), (-), (-) (-), (-), (-), (-)	Lift force, drag force, pitching moment, minimum pressure, normal force (to plane), tangential force (to plane)[2], normal force (to

[1] BαNβClrnc is based on the absolute distance to the nearest point in the tower from BαNβ minus the local tower radius, in the deflected configuration. Please note that this clearance is only approximate because the calculation assumes that the blade is a line with no volume (however, the calculation does use the local tower radius). When BαNβ is above the tower top (or below the tower base), the absolute distance to the tower top (or base) minus the local tower radius, in the deflected configuration, is output.

Fig. 4.5: AeroDyn Output Channel List

situations include obtaining accurate aerodynamic loads for nonstraight blade geometries (e.g., built-in curvature or sweep); skewed flow caused by yawed inflow or turbine tilt; and large rotor motion as a result of placing the turbine atop a compliant offshore floating platform.

Higher-fidelity aerodynamic models are necessary to account for the increased complexity of flexible and floating rotors. Although computational fluid dynamics (CFD) methods are able to capture such features, their computational cost limits the number of simulations that can be feasibly performed, which is an important consideration in load analysis for turbine design. FVW methods are less computationally expensive than CFD methods while modeling similarly complex physics. As opposed to the BEM methods, FVW methods do not rely on ad-hoc engineering models to account for dynamic inflow, skewed wake, tip losses, or ground effects. These effects are inherently part of the model. Numerous vorticity-based tools have been implemented, ranging from the early treatments by Rosenhead ([olaf-Ros31]), the formulation of vortex particle methods by Winckelmans and Leonard ([olaf-WL93]), to the recent mixed Eulerian-Lagrangian compressible formulations of Papadakis ([olaf-Pap14]). Examples of long-standing codes that have been applied in the field of wind energy are GENUVP ([olaf-Vou06]), using vortex particles methods, and AWSM ([olaf-vG03]), using vortex filament methods. Both tools have successfully been coupled to structural solvers. The method was extended by Branlard et al. ([olaf-BPG+15]) to consistently use vortex methods to perform aero-elastic simulations of wind turbines in sheared and turbulent inflow. Most formulations rely on a lifting-line representation of the blades, but recently, a viscous-inviscid representation was used in combination with a structural solver ([olaf-SGarciaSorensenS17]).

cOnvecting LAgrangian Filaments (OLAF) is a free vortex wake (FVW) module used to compute the aerodynamic forces on moving two- or three-bladed horizontal-axis wind turbines. This module has been incorporated into the National Renewable Energy Laboratory physics-based engineering tool, OpenFAST, which solves the aero-hydro-servo-elastic dynamics of individual wind turbines. OLAF is incorporated into the OpenFAST module, *AeroDyn15*, as an alternative to the traditional blade-element momentum (BEM) option, as shown in Figures Fig. 4.6 and Fig. 4.7.

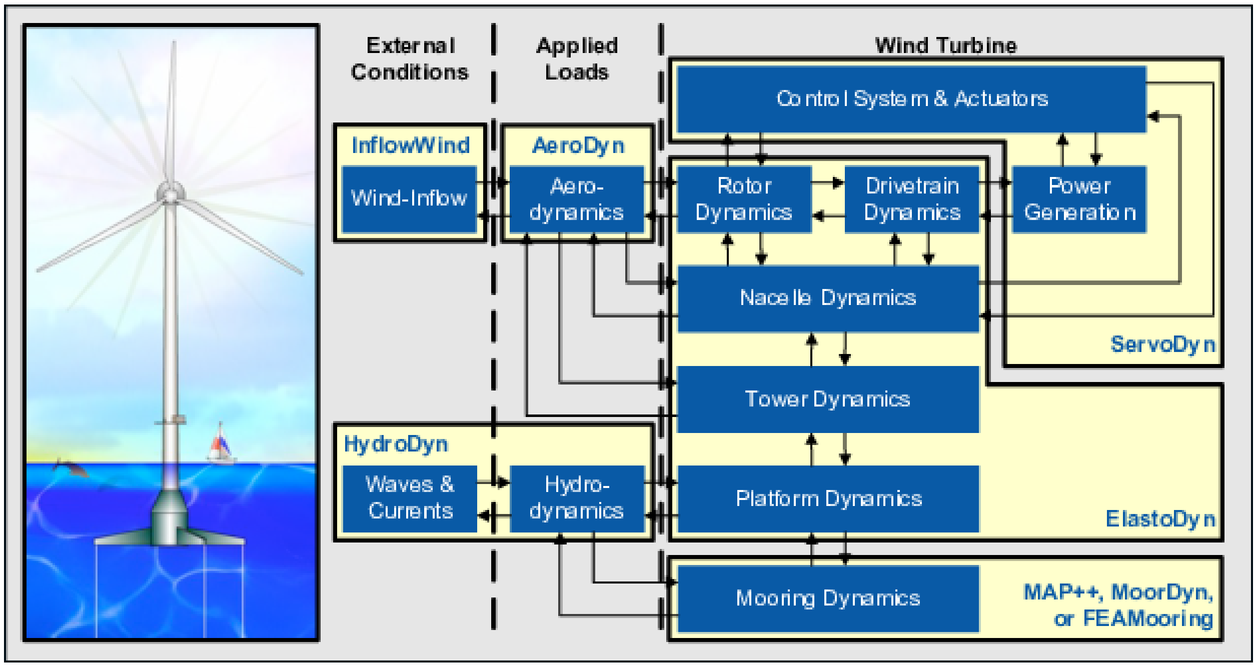


Fig. 4.6: OpenFAST schematic

Incorporating the OLAF module within OpenFAST allows for the modeling of highly flexible turbines along with the aero-hydro-servo-elastic response capabilities of OpenFAST. The OLAF module follows the requirements of the OpenFAST modularization framework ([olaf-SJJ15][olaf-Jon13]).

The OLAF module uses a lifting-line representation of the blades, which is characterized by a distribution of bound circulation. The spatial and time variation of the bound circulation results in free vorticity being emitted in the wake.

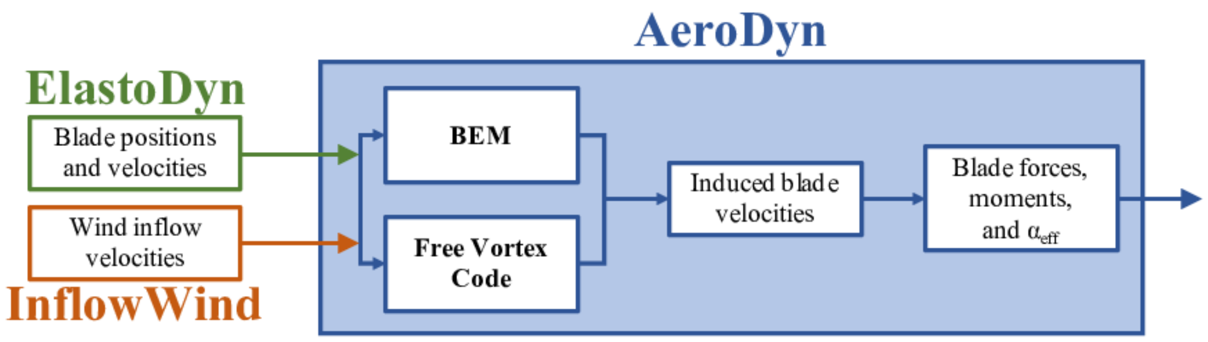


Fig. 4.7: OLAF and BEM integration with *AeroDyn15*

OLAF solves for the turbine wake in a time-accurate manner, which allows the vortices to convect, stretch, and diffuse. The OLAF model is based on a Lagrangian approach, in which the turbine wake is discretized into Lagrangian markers. There are many methods of representing the wake with Lagrangian markers ([olaf-Bra17]). In this work, a hybrid lattice/filament method is used, as depicted in Figure Fig. 4.8.

Here, the position of the Lagrangian markers is defined in terms of wake age, ζ, and azimuthal position, ψ. A lattice method is used in the near wake of the blade. The near wake spans over a user-specified angle or distance for nonrotating cases. Though past research has indicated that a near-wake region of 30° is sufficient ([olaf-Lei06][olaf-ALR02]), it has been shown that a larger near wake is required for high thrust and other challenging conditions. After the near wake region, the wake is assumed to instantaneously roll up into a tip vortex and a root vortex, which are assumed to be the most dominant features for the remainder of the wake ([olaf-LBB02]). Each Lagrangian marker is connected to adjacent markers by straight-line vortex filaments, approximated to second-order accuracy ([olaf-GL02]). The wake is discretized based on the spanwise location of the blade sections and a specified time step (dt), which may be different from the time step of AeroDyn. After an optional initialization period, the wake is allowed to move and distort, thus changing the wake structure as the markers are convected downstream. To limit computational expense, the root and tip vortices are truncated after a specified distance (**WakeLength**) downstream from the turbine. The wake truncation violates Helmholtz's first law and hence introduces an erroneous boundary condition. To alleviate this, the wake is "frozen" in a buffer zone between a specified buffer distance, **FreeWakeLength**, and **WakeLength**. In this buffer zone, the markers convect at the average ambient velocity. In this way, truncation error is minimized~([olaf-LBB02]). The buffer zone is typically chosen as the convected distance over one rotor revolution.

As part of OpenFAST, induced velocities at the lifting line/blade are transferred to *AeroDyn15* and used to compute the effective blade angle of attack at each blade section, which is then used to compute the aerodynamic forces on the blades. The OLAF method returns the same information as the BEM method, but allows for more accurate calculations in areas where BEM assumptions are violated, such as those discussed above. As the OLAF method is more computationally expensive than BEM, both methods remain available in OpenFAST, and the user may specify in the *AeroDyn15* input file which method is used.

The OLAF input file defines the wake convection and circulation solution methods; wake size and length options; Lagrangian marker regularization (viscous core) method; and other simulation and output parameters. The extents of the near and far wakes are specified by a nondimensional length in terms of rotor diameter. Different regularization functions for the vortex elements are available. Additionally, different methods to compute the regularization parameters of the bound and wake vorticity may be selected. In particular, viscous diffusion may be accounted for by dynamically changing the regularization parameter. Wake visualization output options are also available.

This document is organized as follows. Section 4.3.3 covers downloading, compiling, and running OLAF. Section 4.3.4 describes the OLAF input file and modifications to the *AeroDyn15* input file. Section 4.3.5 details the OLAF output file. Section 4.3.6 provides an overview of the OLAF theory, including the free vortex wake method as well as integration into the *AeroDyn15* module. Example input files and a list of output channels are detailed in Appendices A, B, and C.

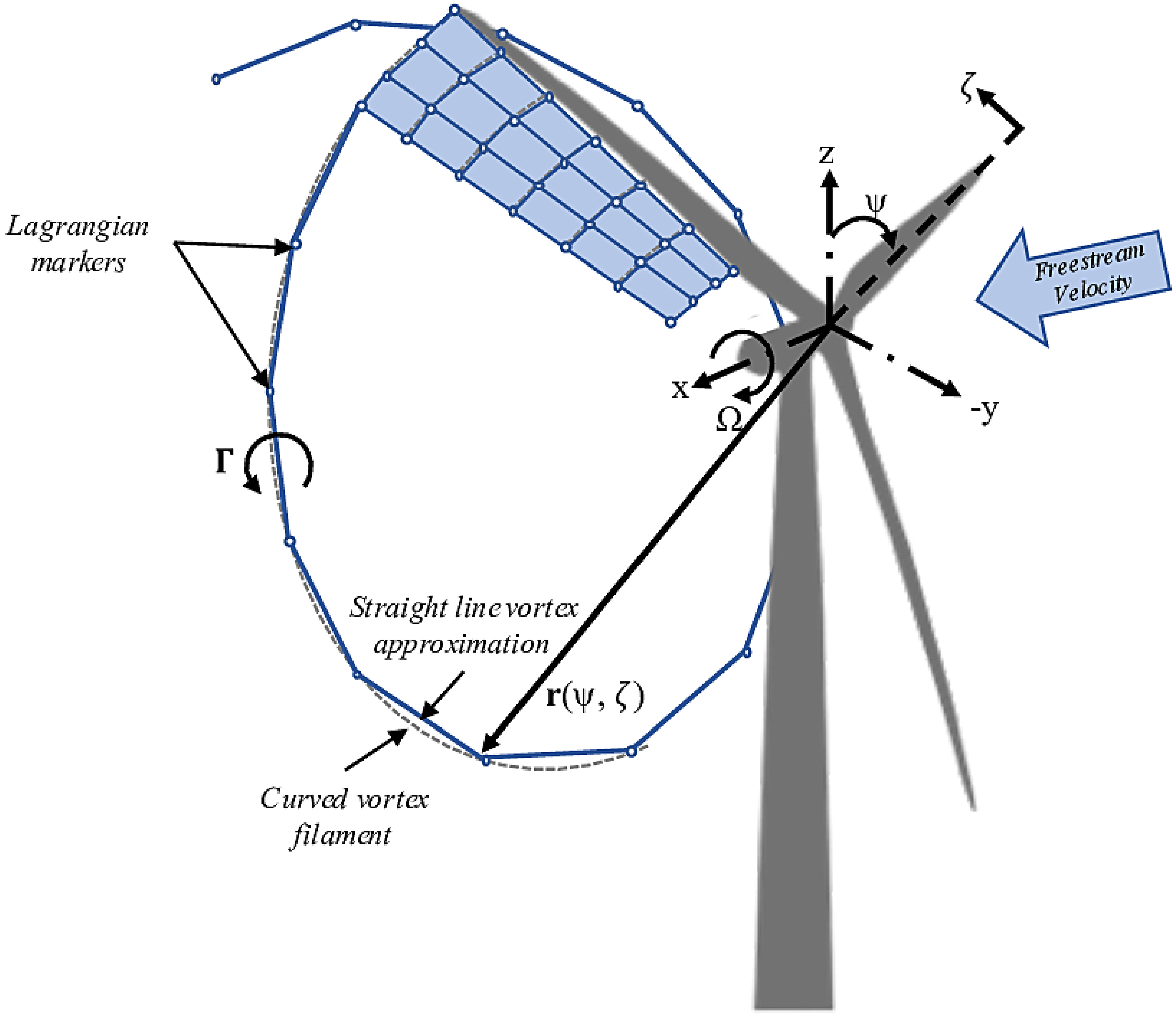


Fig. 4.8: Evolution of near-wake lattice, blade-tip vortex, and Lagrangian markers

4.3.2 List of Symbols

BEM	blade-element momentum
CFD	computational fluid dynamics
DOE	U.S. Department of Energy
F_v	core radius factor
t	time
FVW	free vortex wake
N	number of rotor revolutions before wake cutoff condition
$\vec{r}$	vector between point of interest and vortex segment
$\vec{r}(\psi, \zeta)$	position vector of Lagrangian markers
r_c	core radius
r_{c0}	initial core radius
OLAF	cOnvecting LAgrangian Filaments
α	numerical constant $= 1.25643$
Γ	circulation strength
δ	measure of viscous diffusion
ϵ	measure of strain
$\Delta\psi$	step size for blade rotation
Ω	rotational speed of wind turbine
ζ	vortex wake age
ζ_0	vortex wake age offset
ν	kinematic viscosity
ψ	azimuth blade position

4.3.3 Running OLAF

As OLAF is a module of OpenFAST, the process of downloading, compiling, and running OLAF is the same as that for OpenFAST. Such instructions are available in the *Installing OpenFAST* documentation.

Note: To improve the speed of FVW module, the user may wish to compile with *OpenMP*. To do so, add the *-DOPENMP=ON* option with CMake.

4.3.4 Input Files

No lines should be added or removed from the input files, except in tables where the number of rows is specified.

Units

OLAF uses the International System of Units (e.g., kg, m, s, N). Angles are assumed to be in degrees unless otherwise specified.

OLAF Primary Input File

The primary OLAF input file defines general free wake options, circulation model selection and specification, near- and far-wake length, and wake visualization options. Each section within the file corresponds to an aspect of the OLAF model. For most parameters, the user may specify the value "default" (with or without quotes), in which case a default value, defined below, is used by the program.

See Section 4.3.9 for a sample OLAF primary input file.

General Options

IntMethod [switch] specifies which integration method will be used to convect the Lagrangian markers. There are four options: 1) fourth-order Runge-Kutta *[1]*, 2) fourth-order Adams-Bashforth *[2]*, 3) fourth-order Adams-Bashforth-Moulton *[3]*, and 4) first-order forward Euler *[5]*. The default option is *[5]*. These methods are specified in Section 4.3.6.

DTfvw [sec] specifies the time interval at which the module will update the wake. The time interval must be a multiple of the time step used by *AeroDyn15*. The blade circulation is updated at each intermediate time step based on the intermediate blades positions and wind velocities. The default value is dt_{aero}, where dt_{aero} is the time step used by AeroDyn.

FreeWakeStart [sec] specifies at what time the wake evolution is classified as "free." Before this point is reached, the Lagrangian markers are simply convected with the freestream velocity. After this point, induced velocities are computed and affect the marker convection. If a time less than or equal to zero is given, the wake is "free" from the beginning of the simulation. The default value is 0.

FullCircStart [sec] specifies at what time the blade circulation reaches its full strength. If this value is specified to be > 0, the circulation is multiplied by a factor of 0 at $t = 0$ and linearly increasing to a factor of 1 for $t >$ *FullCircStart*. The default value is 0.

Circulation Specifications

CircSolvMethod [switch] specifies which circulation method is used. There are three options: 1) C_l-based iterative procedure *[1]*, 2) no-flow through *[2]*, and 3) prescribed *[3]*. The default option is *[1]*. These methods are described in Section 4.3.6.

CircSolvConvCrit [-] specifies the dimensionless convergence criteria used for solving the circulation. This variable is only used if *CircSolvMethod = [1]*. The default value is 0.001, corresponding to 0.1% error in the circulation between two iterations.

CircSolvRelaxation [-] specifies the relaxation factor used to solve the circulation. This variable is only used if *CircSolvMethod = [1]*. The default value is 0.1.

CircSolvMaxIter [-] specifies the maximum number of iterations used to solve the circulation. This variable is only used if *CircSolvMethod = [1]*. The default value is 30.

PrescribedCircFile [quoted string] specifies the file containing the prescribed blade circulation. This option is only used if *CircSolvMethod = [3]*. The circulation file format is a delimited file with one header line and two columns. The first column is the dimensionless radial position [r/R]; the second column is the bound circulation value in [m^2/s]. The radial positions do not need to match the AeroDyn node locations. A sample prescribed circulation file is given in Section 4.3.10.

Wake Extent and Discretization Options

nNWPanel [-] specifies the number of FVW time steps (**DTfvw**) for which the near-wake lattice is computed. In the future, this value will be defined as an azimuthal span in degrees or a downstream distance in rotor diameter.

WakeLength [D] specifies the length, in rotor diameters, of the far wake. The default value is 8.[1]

FreeWakeLength [D] specifies the length, in rotor diameters, for which the turbine wake is convected as "free." If *FreeWakeLength* is greater than *WakeLength*, then the entire wake is free. Otherwise, the Lagrangian markers located within the buffer zone delimited by *FreeWakeLength* and *WakeLength* are convected with the average velocity. The default value is 6.[2]

FWShedVorticity [flag] specifies whether shed vorticity is included in the far wake. The default value is *[False]*, specifying that the far wake consists only of the trailed vorticity from the root and tip vortices.

Wake Regularization and Diffusion Options

DiffusionMethod [switch] specifies which diffusion method is used to account for viscous diffusion. There are two options: 1) no diffusion *[0]* and 2) the core-spreading method *[1]*. The default option is *[0]*.

RegDetMethod [switch] specifies which method is used to determine the regularization parameters. There are two options: 1) manual *[0]* and 2) optimized *[1]*. The manual option requires the user to specify the parameters listed in this subsection. The optimized option determines the parameters for the user. The default option is *[0]*.

RegFunction [switch] specifies the regularization function used to remove the singularity of the vortex elements, as specified in Section 4.3.6. There are five options: 1) no correction *[0]*, 2) the Rankine method *[1]*, 3) the Lamb-Oseen method *[2]*, 4) the Vatistas method *[3]*, and 5) the denominator offset method *[4]*. The functions are given in . The default option is *[3]*.

WakeRegMethod [switch] specifies the method of determining viscous core radius (i.e., the regularization parameter). There are three options: 1) constant *[1]*, 2) stretching *[2]*, and 3) age *[3]*. The methods are described in Section 4.3.6. The default option is *[1]*.

WakeRegParam [m] specifies the wake regularization parameter, which is the regularization value used at the initialization of a vortex element. If the regularization method is "constant", this value is used throughout the wake.

BladeRegParam [m] specifies the bound vorticity regularization parameter, which is the regularization value used for the vorticity elements bound to the blades.

CoreSpreadEddyVisc [-] specifies the eddy viscosity parameter δ. The parameter is used for the core-spreading method (*DiffusionMethod = [1]*) and the regularization method with age (*WakeRegMethod = [3]*). The variable δ is described in Section 4.3.6. The default value is 100.

Wake Treatment Options

TwrShadowOnWake [flag] specifies whether the tower potential flow and tower shadow have an influence on the wake convection. The tower shadow model, when activated in AeroDyn, always has an influence on the lifting line, hence the induction and loads on the blade. This option only concerns the wake. The default option is *[False]*.

ShearVorticityModel [switch] specifies whether shear vorticity is modeled in addition to the sheared inflow prescribed by *InflowWind*. There are two options: 1) no treatment *[0]* and 2) mirrored vorticity *[1]*. The mirrored vorticity accounts for the ground effect. Dedicated options to account for the shear vorticity will be implemented at a later time. The shear velocity profile is handled by *InflowWind* irrespective of this input. The default option is *[0]*.

[1] At present, this variable is called nFWPanel and specified as the number of far wake panels. This will be changed soon.

[2] At present, this variable is called nFWPanelFree and specified as the number of free far wake panels. This will be changed soon.

Speedup Options

VelocityMethod [switch] specifies the method used to determine the velocity. There are two options: 1) Biot-Savart law applied to the vortex segments *[1]* and 2) tree formulation using a particle representation *[2]*. The default option is *[1]*.

TreeBranchFactor [-] specifies the dimensionless distance, in branch radius, above which a multipole calculation is used instead of a direct evaluation. This option is only used in conjunction with the tree code (*VelocityMethod = [2]*).

PartPerSegment [-] specifies the number of particles that are used when a vortex segment is represented by vortex particles. The default value is 1.

Output Options

WrVTK [flag] specifies if Visualization Toolkit (VTK) visualization files are to be written out. *WrVTK = [0]* does not write out any VTK files. *WrVTK = [1]* outputs a VTK file at every time step. The outputs are written in the folder, `vtk_fvw`. The parameters *WrVTK*, *VTKCoord*, and *VTK_fps* are independent of the glue code VTK output options.

VTKBlades [-] specifies how many blade VTK files are to be written out. $VTKBlades = n$ outputs VTK files for n blades, with 0 being an acceptable value. The default value is 1.

VTKCoord [switch] specifies in which coordinate system the VTK files are written. There are two options: 1) global coordinate system *[1]* and 2) hub coordinate system *[2]*. The default option is *[1]*.

VTK_fps [1/sec] specifies the output frequency of the VTK files. The provided value is rounded to the nearest allowable multiple of the time step. The default value is $1/dt_{\text{fvw}}$. Specifying *VTK_fps = [all]*, is equivalent to using the value $1/dt_{\text{aero}}$.

AeroDyn15 Input File

Input file modifications

As OLAF is incorporated into the *AeroDyn15* module, a wake computation option has been added to the *AeroDyn15* input file and a line has been added. These additions are as follows.

WakeMod specifies the type of wake model that is used. *WakeMod = [3]* has been added to allow the user to switch from the traditional BEM method to the OLAF method.

FVWFile [string] specifies the OLAF module file, the path is relative to the AeroDyn file, unless an absolute path is provided.

Relevant sections

The BEM options (e.g. tip-loss, skew, and dynamic models) are read and discarded when *WakeMod = [3]*. The following sections and parameters remain relevant and are used by the vortex code:

- general options (e.g., airfoil and tower modeling);
- environmental conditions;
- dynamic stall model options;
- airfoil and blade information;
- tower aerodynamics; and
- outputs.

4.3.5 Output Files

The OLAF module itself does not produce its own output file. However, additional output channels are made available in *AeroDyn15*. As such, the *AeroDyn15* output file is briefly described as well as the outputs made available with OLAF. Visualization files are generated by using the parameter, **WrVTK**. This parameter is available in the OLAF input file, in which case the VTK files are written to the folder `vtk_fvw`, or the primary `.fst` file, in which case the VTK files are written to the folder `vtk`.

Results File

OpenFAST generates a master results file that includes the *AeroDyn15* results. The results are in table format, where each column is a data channel, and each row corresponds to a simulation-output time step. The data channels are specified in the *OUTPUTS* section in the *AeroDyn15* primary input file. The column format of the AeroDyn-generated files is specified using the **OutFmt** parameter of the OpenFAST driver input file.

4.3.6 OLAF Theory

This section details the OLAF method and provides an overview of the computational method, followed by a brief explanation of its integration with OpenFAST.

Introduction - Vorticity Formulation

The vorticity equation for incompressible homogeneous flows in the absence of non-conservative force is given by Eq. (4.2)

$$\frac{d\vec{\omega}}{dt} = \frac{\partial\vec{\omega}}{\partial t} + \underbrace{(\vec{u}\cdot\nabla)}_{\text{convection}}\vec{\omega} = \underbrace{(\vec{\omega}\cdot\nabla)\vec{u}}_{\text{strain}} + \underbrace{\nu\Delta\vec{\omega}}_{\text{diffusion}} \tag{4.2}$$

Here, $\vec{\omega}$ is the vorticity, $\vec{u}$ is the velocity, and ν is the viscosity. In free vortex wake methods, the vorticity equation is used to describe the evolution of the wake vorticity. Different approximations are introduced to ease its resolution, such as projecting the vorticity onto a discrete number of vortex elements (here vortex filaments), and separately treating the convection and diffusion steps, known as viscous-splitting. Several complications arise from the method; in particular, the discretization requires a regularization of the vorticity field (or velocity field) to ensure a smooth approximation.

The forces exerted by the blades onto the flow are expressed in vorticity formulation as well. This vorticity is bound to the blade and has a circulation associated with the lift force. A lifting-line formulation is used here to model the bound vorticity.

The different models of the implemented free vortex code are described in the following sections.

Discretization - Projection

The numerical method uses a finite number of states to model the continuous vorticity distribution. To achieve this, the vorticity distribution is projected onto basis function which is referred to as vortex elements. Vortex filaments are here used as elements that represents the vorticity field. A vortex filament is delimited by two points and hence assumes a direction formed by these two points. A vorticity tube is oriented along the unit vector $\vec{e}_x$ of cross section dS and length l. It can then be approximated by a vortex filament of length l oriented along the same direction. The total vorticity of the tube and the vortex filaments are the same and related by:

$$\vec{\omega}\, dS = \vec{\Gamma} \tag{4.3}$$

where $\vec{\Gamma}$ is the circulation intensity of the vortex filament. If the vorticity tubes are complex and occupy a large volume, the projection onto vortex filaments is difficult and the projection onto vortex particle is more appropriate. Assuming the wake is confined to a thin vorticity layer which defines a velocity jump of know direction, it is possible to approximate the wake vorticity sheet as a mesh of vortex filaments. This is the basis of vortex filament wake methods. Vortex filaments are a singular representation of the vorticity field, as they occupy a line instead of a volume. To better represent the vorticity field, the filaments are "inflated", a process referred to as regularization (see Section 4.3.6). The regularization of the vorticity field also regularizes the velocity field and avoids the singularities that would otherwise occur.

Lifting-Line Representation

The code relies on a lifting-line formulation. Lifting-line methods effectively lump the loads at each cross-section of the blade onto the mean line of the blade and do not account directly for the geometry of each cross-section. In the vorticity-based version of the lifting-line method, the blade is represented by a line of varying circulation. The line follows the motion of the blade and is referred to as "bound" circulation. The bound circulation does not follow the same dynamic equation as the free vorticity of the wake. Instead, the intensity is linked to airfoil lift via the Kutta-Joukowski theorem. Spanwise variation of the bound circulation results in vorticity being emitted into the the wake. This is referred to as "trailed vorticity". Time changes of the bound circulation are also emitted in the wake, referred to as "shed" vorticity. The subsequent paragraphs describe the representation of the bound vorticity.

Lifting-Line Panels and Emitted Wake Panels

The lifting-line and wake representation is illustrated in Fig. 4.9. The blade lifting-line is discretized into a finite number of panels, each of them forming a four sided vortex rings. The spanwise discretization follows the discretization of the AeroDyn blade input file. The number of spanwise panels, n_{LL}, is one less than the total number of AeroDyn nodes, **NumBlNds**. The sides of the panels coincide with the lifting-line and the trailing edge of the blade. The lifting-line is currently defined as the 1/4 chord location from the leading edge (LE). More details on the panelling is provided in Section 4.3.6. At a given time step, the circulation of each lifting-line panel is determined according to one of the three methods developed in Section 4.3.6. At the end of the time step, the circulation of each lifting-line panel is emitted into the wake, forming free vorticity panels. To satisfy the Kutta condition, the circulation of the first near wake panel and the bound circulation are equivalent (see Fig. 4.9 b). The wake panels model the thin shear layer resulting from the continuation of the blade boundary layer. This shear layer can be modelled using a continuous distribution of vortex doublets. A constant doublet strength is assumed on each panel, which in turn is equivalent to a vortex ring of constant circulation.

The current implementation stores the positions and circulations of the panel corner points. In the vortex ring formulation, the boundary between two panels corresponds to a vortex segment of intensity equal to the difference of circulation between the two panels. The convention used to define the segment intensity based on the panels intensity is shown in Fig. 4.9 c. Since the circulation of the bound panels and the first row of near wake panels are equal, the vortex segments located on the trailing edge have no circulation.

Panelling

The definitions used for the panelling of the blade are given in Fig. 4.9 d, following the notations of van Garrel ([olaf-vG03]). The leading edge and trailing edge (TE) locations are directly obtained from the AeroDyn mesh. At two spanwise locations, the LE and TE define the corner points: $\vec{x}_1$, $\vec{x}_2$, $\vec{x}_3$, and $\vec{x}_4$. The current implementation assumes that the aerodynamic center, the lifting-line, and the 1/4 chord location all coincide. For a given panel, the lifting-line is then delimited by the points $\vec{x}_9 = 3/4\,\vec{x}_1 + 1/4\,\vec{x}_2$ and $\vec{x}_{10} = 3/4\,\vec{x}_4 + 1/4\,\vec{x}_3$. The mid points of the four panel sides are noted $\vec{x}_5$, $\vec{x}_6$, $\vec{x}_7$, and $\vec{x}_8$. The lifting-line vector ($\vec{dl}$) as well as the vectors tangential ($\vec{T}$) and

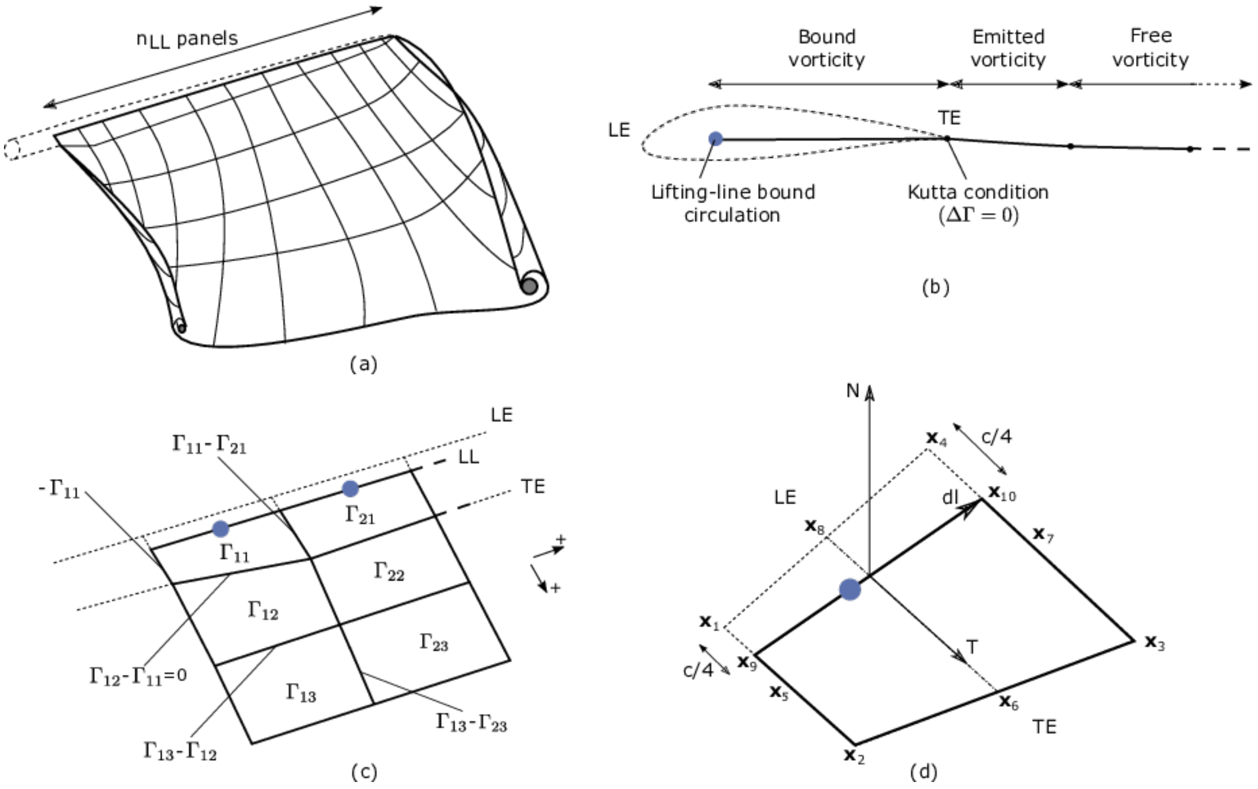


Fig. 4.9: Wake and lifting-line vorticity discretized into vortex ring panels. (a) Overview. (b) Cross-sectional view, defining the leading-edge, trailing edge, and lifting-line. (c) Circulation of panels and corresponding circulation for vorticity segments between panels. (d) Geometrical quantities for a lifting-line panel.

normal ($\vec{N}$) to the panel are defined as:

$$\vec{dl} = \vec{x}_{10} - \vec{x}_9, \qquad \vec{T} = \frac{\vec{x}_6 - \vec{x}_8}{|\vec{x}_6 - \vec{x}_8|}, \qquad \vec{N} = \frac{\vec{T} \times \vec{dl}}{|\vec{T} \times \vec{dl}|} \tag{4.4}$$

The area of the panel is obtained as $dA = |(\vec{x}_6 - \vec{x}_8) \times (\vec{x}_7 - \vec{x}_5)|$. For **CircSolvMethod=[1]**, the control points are located on the lifting-line at the location $\vec{x}_9 + \eta_j \vec{dl}$. The factor η_j is determined based on the full-cosine approximation of van Garrel. This is based on the spanwise widths of the current panel, w_j, and the neighboring panels w_{j-1} and w_{j+1}:

$$\begin{aligned}
\eta_1 &= \frac{w_1}{w_1 + w_2}, \\
\eta_j &= \frac{1}{4}\left[\frac{w_{j-1}}{w_{j-1} + w_j} + \frac{w_j}{w_j + w_{j+1}} + 1\right],\ j = 2..n-1, \\
\eta_n &= \frac{w_{n-1}}{w_{n-1} + w_n}
\end{aligned}$$

For an equidistant spacing, this discretization places the control points at the middle of the lifting-line ($\eta = 0.5$). Theoretical circulation results for an elliptic wing with a cosine spacing are retrieved with such discretization since it places the control points closer to stronger trailing segments at the wing extremities (see e.g. [olaf-Ker00]).

Circulation Solving Methods

Three methods are implemented to determine the bound circulation strength. They are selected using the input **CircSolvMethod**, and are presented in the following sections.

Cl-Based Iterative Method

The Cl-based iterative method determines the circulation within a nonlinear iterative solver that makes use of the polar data at each control point located on the lifting line. The algorithm ensures that the lift obtained using the angle of attack and the polar data matches the lift obtained with the Kutta-Joukowski theorem. At present, it is the preferred method to compute the circulation along the blade span. It is selected with **CircSolvMethod=[1]**. The method is described in the work from van Garrel ([olaf-vG03]). The algorithm is implemented in at iterative approach using the following steps:

1. The circulation distribution from the previous time step is used as a guessed circulation, Γ_{prev}.
2. The velocity at each control points j is computed as the sum of the wind velocity, the structural velocity, and the velocity induced by all the vorticity in the domain, evaluated at the control point location.

$$\vec{v}_j = \vec{V}_0 - \vec{V}_{\text{elast}} + \vec{v}_{\omega,\text{free}} + \vec{v}_{\Gamma_{ll}}$$

   $\vec{v}_{\omega,\text{free}}$ is the velocity induced by all free vortex filaments, as introduced in Eq. (4.11) . The contribution of $\vec{v}_{\Gamma_{ll}}$ comes from the lifting-line panels and the first row of near wake panels, for which the circulation is set to Γ_{prev}

3. The circulation for all lifting-line panels j is obtained as follows.

$$\Gamma_{ll,j} = \frac{1}{2} C_{l,j}(\alpha_j) \frac{\left[(\vec{v}_j \cdot \vec{N})^2 + (\vec{v}_j \cdot \vec{T})^2\right]^2 dA}{\sqrt{\left[(\vec{v}_j \times \vec{dl}) \cdot \vec{N}\right]^2 + \left[(\vec{v}_j \times \vec{dl}) \cdot \vec{T}\right]^2}}, \quad \text{with} \quad \alpha_j = \operatorname{atan}\left(\frac{\vec{v}_j \cdot \vec{N}}{\vec{v}_j \cdot \vec{T}}\right)$$

   The function $C_{l,j}$ is the lift coefficient obtained from the polar data of blade section j and α_j is the angle of attack at the control point.

4. The new circulation is set using the relaxation factor k_{relax} (**CircSolvRelaxation**):

$$\Gamma_{\text{new}} = \Gamma_{\text{prev}} + k_{\text{relax}} \Delta\Gamma, \qquad \Delta\Gamma = \Gamma_{ll} - \Gamma_{\text{prev}}$$

5. Convergence is checked using the criterion k_{crit} (**CircSolvConvCrit**):

$$\frac{\max(|\Delta\Gamma|}{\text{mean}(|\Gamma_{\text{new}}|)} < k_{\text{crit}}$$

If convergence is not reached, steps 2-5 are repeated using Γ_{new} as the guessed circulation Γ_{prev}.

No-flow-through Method

A Weissinger-L-based representation ([olaf-Wei47]) of the lifting surface is also available ([olaf-BL94][olaf-Gup06][olaf-Rib07]). In this method, the circulation is solved by satisfying a no-flow through condition at the 1/4-chord points. It is selected with **CircSolvMethod=[2]**.

Prescribed Circulation

The final available method prescribes a constant circulation. A user specified spanwise distribution of circulation is prescribed onto the blades. It is selected with **CircSolvMethod=[3]**.

Free Vorticity Convection

The governing equation of motion for a vortex filament is given by the convection equation of a Lagrangian marker:

$$\frac{d\vec{r}}{dt} = \vec{V}(\vec{r}, t) \tag{4.5}$$

where $\vec{r}$ is the position of a Lagrangian marker. The Lagrangian markers are the end points of the vortex filaments. The Lagrangian convection of the filaments stretches the filaments and thus automatically accounts for strain in the vorticity equation.

At present, a first-order forward Euler method is used to numerically solve the left-hand side of Eq. (4.5) for the vortex filament location (**IntMethod=[5]**). This is an explicit method solved using Eq. (4.6).

$$\vec{r} = \vec{r} + \vec{V}\Delta t \tag{4.6}$$

Free Vorticity Convection in Polar Coordinates

The governing equation of motion for a vortex filament is given by:

$$\frac{d\vec{r}(\psi, \zeta)}{dt} = \vec{V}[\vec{r}(\psi, \zeta), t] \tag{4.7}$$

Using the chain rule, Eq. (4.7) is rewritten as:

$$\frac{\partial\vec{r}(\psi, \zeta)}{\partial\psi} + \frac{\partial\vec{r}(\psi, \zeta)}{\partial\zeta} = \frac{\vec{V}[\vec{r}(\psi, \zeta), t]}{\Omega} \tag{4.8}$$

where $d\psi/dt = \Omega$ and $d\psi = d\zeta$ ([olaf-LBB02]). Here, $\vec{r}(\psi, \zeta)$ is the position vector of a Lagrangian marker, and $\vec{V}[\vec{r}(\psi, \zeta)]$ is the velocity.

Induced Velocity and Velocity Field

The velocity term on the right-hand side of Eq. (4.5) is a nonlinear function of the vortex position, representing a combination of the freestream and induced velocities ([olaf-Han08]). The induced velocities at point $\vec{x}$, caused by each straight-line filament, are computed using the Biot-Savart law, which considers the locations of the Lagrangian markers and the intensity of the vortex elements ([olaf-LBB02]):

$$d\vec{v}(\vec{x}) = \frac{\Gamma}{4\pi}\frac{\vec{dl}\times\vec{r}}{r^3} \tag{4.9}$$

Here, Γ is the circulation strength of the filament, $\vec{dl}$ is an elementary length along the filament, $\vec{r}$ is the vector between a point on the filament and the control point $\vec{x}$, and $r = |\vec{r}|$ is the norm of the vector. The integration of the Biot-Savart law along the filament length, delimited by the points $\vec{x}_1$ and $\vec{x}_2$ leads to:

$$\vec{v}(\vec{x}) = F_\nu \frac{\Gamma}{4\pi}\frac{(r_1+r_2)}{r_1r_2(r_1r_2+\vec{r}_1\cdot\vec{r}_2)}\vec{r}_1\times\vec{r}_2 \tag{4.10}$$

with $\vec{r}_1 = \vec{x} - \vec{x}_1$ and $\vec{r}_2 = \vec{x} - \vec{x}_2$. The factor F_ν is a regularization parameter, discussed in Section 4.3.6. r_0 is the filament length, where $\vec{r}_0 = \vec{x}_2 - \vec{x}_1$. The distance orthogonal to the filament is:

$$\rho = \frac{|\vec{r}_1\times\vec{r}_2|}{r_0}$$

The velocity at any point of the domain is obtained by superposition of the velocity induced by all vortex filaments, and by superposition of the primary flow, $\vec{V}_0$, (here assumed divergence free):

$$\vec{V}(\vec{x}) = \vec{V}_0(\vec{x}) + \vec{v}_\omega(\vec{x}), \quad \text{with} \quad \vec{v}_\omega(\vec{x}) = \sum_k \vec{v}_k(\vec{x}) \tag{4.11}$$

where the sum is over all the vortex filaments, each of intensity Γ_k. The intensity of each filament is determined by spanwise and time changes of the bound circulation, as discussed in Section 4.3.6. In tree-based methods, the sum over all vortex elements is reduced by lumping together the elements that are far away from the control points.

Regularization

Regularization and viscous diffusion

The singularity that occurs in Eq. (4.9) greatly affects the numerical accuracy of vortex methods. By regularizing the "1-over-r" kernel of the Biot-Savart law, it is possible to obtain a numerical method that converges to the Navier-Stokes equations. The regularization is used to improve the regularity of the discrete vorticity field, as compared to the "true" continuous vorticity field. This regularization is usually obtained by convolution with a smooth function. In this case, the regularization of the vorticity field and the velocity field are the same. Some engineering models also perform regularization by directly introducing additional terms in the denominator of the Biot-Savart velocity kernel. The factor, F_ν, was introduced in Eq. (4.10) to account for this regularization.

In the convergence proofs of vortex methods, regularization and viscous diffusion are two distinct aspects. It is common practice in vortex filament methods to blur the notion of regularization with the notion of viscous diffusion. Indeed, for a physical vortex filament, viscous effects prevent the singularity from occurring and diffuse the vortex strength with time. The circular zone where the velocity drops to zero around the vortex is referred to as the vortex core. A length increase of the vortex segment will result in a vortex core radius decrease, and vice versa. Diffusion, on the other hand, continually spreads the vortex radially.

Because of the previously mentioned analogy, practitioners of vortex filament methods often refer to regularization as "viscous-core" models and regularization parameters as "core-radii." Additionally, viscous diffusion is often introduced by modifying the regularization parameter in space and time instead of solving the diffusion from the vorticity equation. The distinction is made explicit in this document when clarification is required, but a loose terminology is used when the context is clear.

Determination of the regularization parameter

The regularization parameter is both a function of the physics being modeled (blade boundary layer and wake) and the choice of discretization. Contributing factors are the chord length, the boundary layer height, and the volume that each vortex filament is approximating. Currently the choice is left to the user (**RegDetMethod=[0]**). Empirical results for a rotating blade are found in the work of Gupta ([olaf-Gup06]). As a guideline, the regularization parameter may be chosen as twice the average spanwise discretization of the blade. This guideline is implemented when the user chooses **RegDetMethod=[1]**. Further refinement of this option will be considered in the future.

Implemented regularization functions

Several regularization functions have been developed ([olaf-Ran58][olaf-Scu75][olaf-VKM91]). At present, five options are available: 1) No correction, 2) the Rankine method, 3) the Lamb-Oseen method, 4) the Vatistas method, or 5) the denominator offset method. If no correction method is used, (**RegFunction=[0]**), $F_\nu = 1$. The remaining methods are detailed in the following sections. Here, r_c is the regularization parameter (**WakeRegParam**) and ρ is the distance to the filament. Both variables are expressed in meters.

Rankine

The Rankine method ([olaf-Ran58]) is the simplest regularization model. With this method, the Rankine vortex has a finite core with a solid body rotation near the vortex center and a potential vortex away from the center. If this method is used (**RegFunction=[1]**), the viscous core correction is given by Eq. (4.12).

$$F_\nu = \begin{cases} \rho^2/r_c^2 & 0 < \rho < 1 \\ 1 & \rho > 1 \end{cases} \tag{4.12}$$

Here, r_c is the viscous core radius of a vortex filament, detailed in Section 4.3.6.

Lamb-Oseen

If the Lamb-Oseen method is used [**RegFunction=[2]**], the viscous core correction is given by Eq. (4.13).

$$F_\nu = \left[1 - \exp(-\frac{\rho^2}{r_c^2})\right] \tag{4.13}$$

Vatistas

If the Vatistas method is used [**RegFunction=[3]**], the viscous core correction is given by Eq. (4.14).

$$F_\nu = \frac{\rho^2}{(\rho^{2n} + r_c^{2n})^{1/n}} = \frac{(\rho/r_c)^2}{(1 + (\rho/r_c)^{2n})^{1/n}} \tag{4.14}$$

Here, ρ is the distance from a vortex segment to an arbitrary point ([olaf-Abe16]). Research from rotorcraft applications suggests a value of $n = 2$, which is used in this work ([olaf-BL93]).

Denominator Offset/Cut-Off

If the denominator offfset method is used [**RegFunction=[4]**], the viscous core correction is given by Eq. (4.15)

$$\vec{v}(\vec{x}) = \frac{\Gamma}{4\pi} \frac{(r_1 + r_2)}{r_1 r_2 (r_1 r_2 + \vec{r}_1 \cdot \vec{r}_2) + r_c^2 r_0^2} \vec{r}_1 \times \vec{r}_2 \tag{4.15}$$

Here, the singularity is removed by introducing an additive factor in the denominator of Eq. (4.10), proportional to the filament length r_0. In this case, $F_\nu = 1$. This method is found in the work of van Garrel ([olaf-vG03]).

Time Evolution of the Regularization Parameter–Core Spreading Method

There are four available methods by which the regularization parameter may evolve with time: 1) constant value, 2) stretching, 3) wake age, or 4) stretching and wake age. The three latter methods blend the notions of viscous diffusion and regularization. The notation r_{c0} used in this section corresponds to input file parameter value **WakeRegParam**.

Constant

If a constant value is selected, (**WakeRegMethod=[1]**), the value of r_c remains unchanged for all Lagrangian markers throughout the simulation and is taken as the value given with the parameter **WakeRegParam** in meters.

$$r_c(\zeta) = r_{c0} \tag{4.16}$$

Here, ζ is the vortex wake age, measured from its emission time.

Stretching

If the stretching method is selected, (**WakeRegMethod=[2]**), the viscous core radius is modeled by Eq. (4.17).

$$r_c(\zeta, \epsilon) = r_{c0}(1+\epsilon)^{-1} \tag{4.17}$$

$$\epsilon = \frac{\Delta l}{l}$$

Here, ϵ is the vortex-filament strain, l is the filament length, and Δl is the change of length between two time steps. The integral in Eq. (4.17) represents strain effects.

Wake Age / Core-Spreading

If the wake age method is selected, (**WakeRegMethod=[3]**), the viscous core radius is modeled by Eq. (4.18).

$$r_c(\zeta) = \sqrt{r_{c0}^2 + 4\alpha\delta\nu\zeta} \tag{4.18}$$

where $\alpha = 1.25643$, ν is kinematic viscosity, and δ is a viscous diffusion parameter (typically between 1 and $1,000$). The parameter δ is provided in the input file as **CoreSpreadEddyVisc**. Here, the term $4\alpha\delta\nu\zeta$, accounts for viscous effects as the wake propagates downstream. The higher the background turbulence, the more diffusion of the vorticity with time, and the higher the value of δ should be. This method partially accounts for viscous diffusion of the vorticity while neglecting the interaction between the wake vorticity itself or between the wake vorticity and the background flow. It is often referred to as the core-spreading method. Setting **DiffusionMethod=[1]** is the same as using the wake age method (**WakeRegMethod=[3]**).

Stretching and Wake Age

If the stretching and wake-age method is selected (**WakeRegMethod=[4]**), the viscous core radius is modeled by Eq. (4.19).

$$r_c(\zeta, \epsilon) = \sqrt{r_{c0}^2 + 4\alpha\delta\nu\zeta\left(1+\epsilon\right)^{-1}} \tag{4.19}$$

Diffusion

The viscous-splitting assumption is used to solve for the convection and diffusion of the vorticity separately. The diffusion term $\nu\Delta\vec{\omega}$ represents molecular diffusion. This term allows for viscous connection of vorticity lines. Also, turbulent flows will diffuse the vorticity in a similar manner based on a turbulent eddy viscosity.

The parameter **DiffusionMethod** is used to switch between viscous diffusion methods. Currently, only the core-spreading method is implemented. The method is described in Section 4.3.6 since it is equivalent to the increase of the regularization parameter with the wake age.

4.3.7 State-Space Representation and Integration with OpenFAST

State, Constraint, Input, and Output Variables

The OLAF module has been integrated into the latest version of OpenFAST via *AeroDyn15*, following the OpenFAST modularization framework ([olaf-Jon13][olaf-SJJ15]). To follow the OpenFAST framework, the vortex code is written as a module, and its formulation comprises state, constraint, and output equations. The data manipulated by the module include the following vectors: constant parameters, $\vec{p}$; inputs, $\vec{u}$; constrained state, $\vec{z}$; states, $\vec{x}$; and outputs, $\vec{y}$. The vectors are defined as follows:

- Parameters, $\vec{p}$ – a set of internal system values that are independent of the states and inputs. The parameters can be fully defined at initialization and characterize the system state and output equations.
- Inputs, $\vec{u}$ – a set of values supplied to the module that, along with the states, are needed to calculate future states and the system output.
- Constraint states, $\vec{z}$ – algebraic variables that are calculated using a nonlinear solver, based on values from the current time step.
- States, $\vec{x}$ – a set of internal values of the module. They are influenced by the inputs and used to calculate future state values and output. Continuous states are employed, meaning that the states are differentiable in time and characterized by continuous time-differential equations.
- Outputs, $\vec{y}$ – a set of values calculated and returned by the module that depend on the states, inputs, and/or parameters through output equations.

The parameters of the vortex code include:

- Fluid characteristics: kinematic viscosity, ν.
- Airfoil characteristics: chord c and polar data – $C_l(\alpha)$, $C_d(\alpha)$, $C_m(\alpha)$).
- Algorithmic methods and parameters, e.g., regularization, viscous diffusion, discretization, wake geometry, and acceleration.

The inputs of the vortex code are:

- Position, orientation, translational velocity, and rotational velocity of the different nodes of the lifting lines ($\vec{r}_{ll}$, Λ_{ll}, $\vec{\dot{r}}_{ll}$, and $\vec{\omega}_{ll}$, respectively), gathered into the vector, $\vec{x}_{\text{elast},ll}$, for conciseness. These quantities are handled using the mesh-mapping functionality and data structure of OpenFAST.

- Disturbed velocity field at requested locations, written $\vec{V_0} = [\vec{V}_{0,ll}, \vec{V}_{0,m}]$. Locations are requested for lifting-line points, $\vec{r}_{ll}$, and Lagrangian markers, $\vec{r}_m$. Based on the parameters, this disturbed velocity field may contain the following influences: freestream, shear, veer, turbulence, tower, and nacelle disturbance. The locations where the velocity field is requested are typically the location of the Lagrangian markers.

The constraint states are:

- The circulation intensity along the lifting lines, Γ_{ll}.

The continuous states are:

- The position of the Lagrangian markers, $\vec{r}_m$
- The vorticity associated with each vortex element, $\vec{\omega}_e$. For a projection of the vorticity onto vortex segments, this corresponds to the circulation, $\vec{\Gamma}_e$. For each segment, $\vec{\Gamma}_e = \Gamma_e \vec{dl}_e = \vec{\omega}_e dV_e$, with $\vec{dl}_e$ and dV_e, the vortex segment length and its equivalent vortex volume.

The outputs are[1]:

- The induced velocity at the lifting-line nodes, $\vec{v}_{i,ll}$
- The locations where the undisturbed wind is computed, $\vec{r}_r$ (typically $\vec{r_r} = \vec{r}_m$).

State, Constraint, and Output Equations

An overview of the states, constraints, and output equations is given here. More details are provided in Section 4.3.6. The constraint equation is used to determine the circulation distribution along the span of each lifting line. For the van Garrel method, this circulation is a function of the angle of attack along the blade and the airfoil coefficients. The angle of attack at a given lifting-line node is a function of the undisturbed velocity, $\vec{v}_{0,ll}$, and the velocity induced by the vorticity, $\vec{v}_{i,ll}$, at that point. Part of the induced velocity is caused by the vorticity being shed and trailed at the current time step, which in turn is a function of the circulation distribution along the lifting line. This constraint equation may be written as:

$$\vec{Z} = \vec{0} = \vec{\Gamma}_{ll} - \vec{\Gamma}_p \bigg(\vec{\alpha}(\vec{x}, \vec{u}), \vec{p} \bigg)$$

where $\vec{\Gamma}_p$ is the function that returns the circulation along the blade span, according to one of the methods presented in Section 4.3.6.

The state equation specifies the time evolution of the vorticity and the convection of the Lagrangian markers:

$$\frac{d\vec{\omega}_e}{dt} = \bigg[(\vec{\omega} \cdot \nabla)\vec{v} + \nu \nabla^2 \vec{\omega} \bigg]_e$$

$$\frac{d\vec{r}_m}{dt} = \vec{V}(\vec{r}_m) = \vec{V}_0(\vec{r}_m) + \vec{v}_\omega(\vec{r}_m) = \vec{V}_0(\vec{r}_m) + \vec{V}_\omega(\vec{r}_m, \vec{r}_m, \vec{\omega}) \tag{4.20}$$

Here,

- $\vec{v}_\omega$ is the velocity induced by the vorticity in the domain;
- $\vec{V}_\omega(\vec{r}, \vec{r}_m, \vec{\omega})$ is the function that computes this induced velocity at a given point, $\vec{r}$, based on the location of the Lagrangian markers and the intensity of the vortex elements;
- the subscript e indicates that a quantity is applied to an element; and
- the vorticity, $\vec{\omega}$, is recovered from the vorticity of the vortex elements by means of discrete convolutions.

[1] The loads on the lifting line are not an output of the vortex code; their calculation is handled by a separate submodule of *AeroDyn*.

For vortex-segment simulations, the viscous-splitting algorithm is used, and the convection step (Eq. (4.20)) is the main state equation being solved for. The vorticity stretching is automatically accounted for, and the diffusion is performed *a posteriori*. The velocity function, $\vec{V}_\omega$, uses the Biot-Savart law. The output equation is:

$$\vec{y}_1 = \vec{v}_{i,ll} = \vec{V}_\omega(\vec{r}_{ll}, \vec{r}_m, \vec{\omega})$$
$$\vec{y}_2 = \vec{r}_r$$

Integration with AeroDyn15

The vortex code has been integrated as a submodule of the aerodynamic module of OpenFAST, *AeroDyn15*. The data workflow between the different modules and submodules of OpenFAST is illustrated in Fig. 4.10. AeroDyn inputs such as BEM options (e.g., tip-loss factor), skew model, and dynamic inflow are discarded when the vortex code is used. The environmental conditions, tower shadow, and dynamic stall model options are used. This integration required a restructuring of the *AeroDyn15* module to isolate the parts of the code related to tower shadow modeling, induction computation, lifting-line-forces computations, and dynamic stall. The dynamic stall model is adapted when used in conjunction with the vortex code to ensure the effect of shed vorticity is not accounted for twice. The interface between *AeroDyn15* and the inflow module, *InflowWind*, was accommodated to include the additionally requested points by the vortex code.

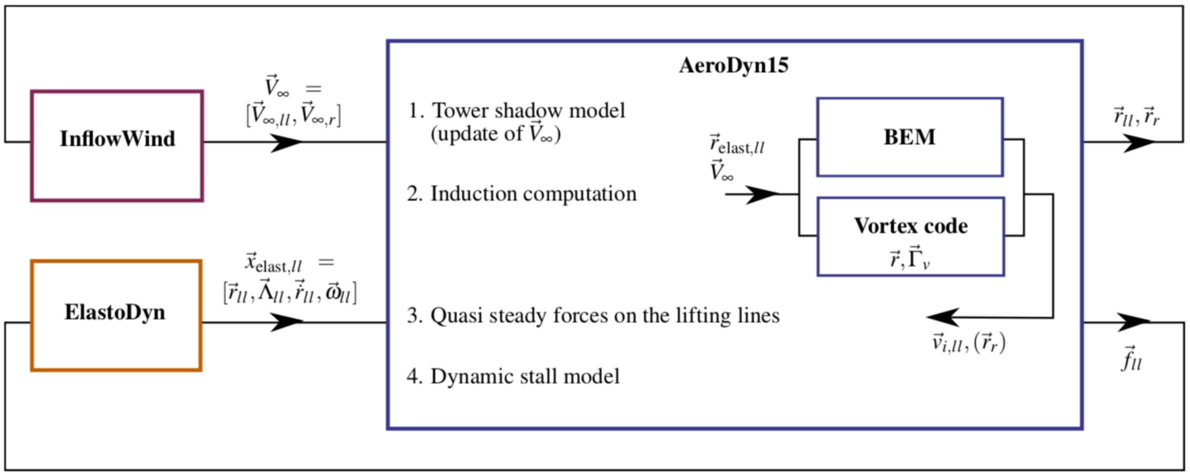

AeroDyn15
InflowWind
ElastoDyn
$\vec{V}_\infty = [\vec{V}_{\infty,ll}, \vec{V}_{\infty,r}]$
$\vec{x}_{elast,ll} = [\vec{r}_{ll}, \vec{\Lambda}_{ll}, \dot{\vec{r}}_{ll}, \vec{\omega}_{ll}]$
1. Tower shadow model (update of $\vec{V}_\infty$)
2. Induction computation
3. Quasi steady forces on the lifting lines
4. Dynamic stall model
$\vec{r}_{elast,ll}$
$\vec{V}_\infty$
BEM
Vortex code
$\vec{r}, \vec{\Gamma}_v$
$\vec{v}_{i,ll}, (\vec{r}_r)$
$\vec{r}_{ll}, \vec{r}_r$
$\vec{f}_{ll}$

Fig. 4.10: OpenFAST-OLAF code integration workflow

4.3.8 Future Work

This first implementation phase focused on single-turbine capabilities, fulfilling the basic requirements for the design of large and novel rotor concepts. Future development work will turn toward the implementation of features enabling multiple-turbine simulations on medium-to-large-scale computational clusters. The reduction of the computational time will also be of focus. This may be achieved using tree techniques such as the fast multipole method. Further algorithmic options, such as vortex amalgamation in the far wake, will be considered to speed up the simulation. The framework presented in this manual is compatible with grid-free or grid-based vortex particle formulations. Such particle-based implementations will also be envisaged in the future. Further validation of the code against measurements and higher-order tools will be pursued. Applications to cases known to be challenging for the BEM algorithm will also be investigated, such as highly flexible rotors, offshore floating turbines, small-scale wind farms, multiple-rotor turbines, or kites.

The following list contains future work on OLAF software:

- Lagrangian particles
- Multiple turbines, integration into FAST.Farm
- Code speed-up
- Dedicated dynamic stall model

4.3.9 Appendix A: OLAF Primary Input File

Check the regression test cases for updates to this input file.

```
--------------------------- OLAF (cOnvecting LAgrangian Filaments) INPUT FILE --------
↪---------
Free wake input file for the Helix test case
--------------------------- GENERAL OPTIONS ------------------------------------------
↪---------
5       IntMethod          Integration method {5: Forward Euler 1st order, default: 5}
↪ (switch)
0.2     DTfvw              Time interval for wake propagation. {default: dtaero} (s)
5       FreeWakeStart      Time when wake is free. (-) value = always free. {default:␣
↪0.0} (s)
2.0     FullCircStart      Time at which full circulation is reached. {default: 0.0}␣
↪(s)
--------------------------- CIRCULATION SPECIFICATIONS -------------------------------
↪---------
1       CircSolvingMethod  Circulation solving method {1: Cl-Based, 2: No-Flow␣
↪Through, 3: Prescribed, default: 1 }(switch)
0.01    CircSolvConvCrit   Convergence criteria {default: 0.001} [only if␣
↪CircSolvingMethod=1] (-)
0.1     CircSolvRelaxation Relaxation factor {default: 0.1} [only if␣
↪CircSolvingMethod=1] (-)
30      CircSolvMaxIter    Maximum number of iterations for circulation solving
↪{default: 30} (-)
 "NA"   PrescribedCircFile File containing prescribed circulation [only if␣
↪CircSolvingMethod=3] (quoted string)
===============================================================================================
--------------------------- WAKE OPTIONS ---------------------------------------------
↪---------
------------------- WAKE EXTENT AND DISCRETIZATION -----------------------------------
↪---------
50      nNWPanel           Number of near-wake panels [integer] (-)
400     WakeLength         Total wake distance [integer] (number of time steps)
default FreeWakeLength     Wake length that is free [integer] (number of time steps)
↪{default: WakeLength}
False   FWShedVorticity    Include shed vorticity in the far wake {default: false}
------------------- WAKE REGULARIZATIONS AND DIFFUSION -------------------------------
↪----------
0       DiffusionMethod    Diffusion method to account for viscous effects {0: None,␣
↪1: Core Spreading, "default": 0}
0       RegDeterMethod     Method to determine the regularization parameters {0: ␣
↪Manual, 1: Optimized, default: 0 }
2       RegFunction        Viscous diffusion function {0: None, 1: Rankine, 2:␣
↪LambOseen, 3: Vatistas, 4: Denominator, "default": 3} (switch)
0       WakeRegMethod      Wake regularization method {1: Constant, 2: Stretching, 3:␣
↪Age, default: 1} (switch)
2.0     WakeRegFactor      Wake regularization factor (m)
2.0     WingRegFactor      Wing regularization factor (m)
```

(continues on next page)

(continued from previous page)

```
100     CoreSpreadEddyVisc Eddy viscosity in core spreading methods, typical values 1-
↪1000
------------------- WAKE TREATMENT OPTIONS ------------------------------------------
↪--------
False   TwrShadowOnWake    Include tower flow disturbance effects on wake convection
↪{default:false} [only if TwrPotent or TwrShadow]
0       ShearModel         Shear Model {0: No treatment, 1: Mirrored vorticity,␣
↪default: 0}
------------------- SPEEDUP OPTIONS -------------------------------------------------
↪---------
2       VelocityMethod     Method to determine the velocity {1:Biot-Savart Segment,␣
↪2:Particle tree, default: 1}
1.5     TreeBranchFactor   Branch radius fraction above which a multipole calculation␣
↪is used {default: 2.0} [only if VelocityMethod=2]
1       PartPerSegment     Number of particles per segment [only if VelocityMethod=2]
==============================================================================================
--------------------------- OUTPUT OPTIONS  ------------------------------------------
↪---------
1       WrVTk              Outputs Visualization Toolkit (VTK) (independent of .fst␣
↪option) {0: NoVTK, 1: Write VTK at each time step} (flag)
1       nVTKBlades         Number of blades for which VTK files are exported {0: No␣
↪VTK per blade, n: VTK for blade 1 to n} (-)
2       VTKCoord           Coordinate system used for VTK export. {1: Global, 2: Hub,
↪"default": 1}
1       VTK_fps            Frame rate for VTK output (frames per second) {"all" for␣
↪all glue code timesteps, "default" for all OLAF timesteps} [used only if WrVTK=1]
-------------------------------------------------------------------------------------
↪----------
```

4.3.10 Appendix B: Prescribed Circulation Input File

Check the regression tests for updated versions of this file.

```
r/R [-],  Gamma [m^2/s]
0.048488,     0.000000
0.087326,     0.442312
0.126163,     6.909277
0.165000,    23.678557
0.203837,    55.650700
0.242674,    74.091529
0.281512,    84.205843
0.320349,    88.740429
0.359186,    89.730814
0.398023,    88.568114
0.436860,    87.114743
0.475698,    86.110557
0.514535,    85.705529
0.553372,    85.215829
0.592209,    84.547371
0.631047,    83.774329
0.669884,    82.889157
0.708721,    81.635600
0.747558,    79.788700
0.786395,    77.195200
0.825233,    73.765100
```

(continues on next page)

(continued from previous page)

```
0.864070,    69.275900
0.902907,    62.965400
0.941744,    53.603300
0.980581,    39.854000
```

4.3.11 Appendix C: OLAF List of Output Channels

This is a list of all possible output parameters from the OLAF module. The names are grouped by meaning, but can be ordered in the OUTPUTS section of the *AeroDyn15* primary input file, as the user sees fit. $N\beta$ refers to output node, β, where β is a number in the range [1,9], corresponding to entry, β, in the **OutNd** list. $B\alpha$ is prefixed to each output name, where α is a number in the range [1,3], corresponding to the blade number.

Table 4.1: Available OLAF Output Channels

Channel Name(s)	Units	Description
$B\alpha N\beta Gam$	m^2/s	Circulation along the blade

4.4 Aeroacoustics Noise Model of OpenFAST

4.4.1 List of Acronyms

BPM	Brooks-Pope-Marcolini airfoil noise model
dB	decibels
dBA	A-weighted decibels
deg	degrees
Hz	hertz
IEA	International Energy Agency
kg	kilograms
kHz	kilohertz
LFC	low-frequency correction
m	meters
N	newtons
NREL	National Renewable Energy Laboratory
rad	radians

continues on next page

Table 4.2 – continued from previous page

s	seconds
SPL	sound pressure level
TBL	turbulent boundary layer
TBL-TE	turbulent boundary layer – trailing edge
TNO	a Netherlands organization for applied scientific research
TE	trailing edge
TI	turbulent inflow
TUM	Technical University of Munich

4.4.2 List of Symbols

l	low frequency	
h	high frequency	
p	airfoil pressure side	
s	airfoil suction side	
t	turbulence	
0	reference	
1	parallel to airfoil chord	
2	normal to airfoil chord	
3	blade spanwise direction	
α	angle of attack	[rad]
β^2	Prandtl-Glauert correction factor	[-]
δ	airfoil boundary layer thickness	[-]
δ^*	airfoil boundary layer displacement thickness	[-]
θ	airfoil boundary layer momentum thickness	[-]
$\Theta_e,\ \Phi_e$	angles between emitter and observer	[rad]
ρ	air density	[kg/m^3]

continues on next page

Table 4.3 – continued from previous page

ω	radial frequency	[rad/s]
A_w	A-weight	[dB]
c	speed of sound	[m/s]
c_i	chord at blade spanwise position i	[m]
d	blade span at station i	[m]
$\overline{D}$	directivity function	[-]
f	frequency	[Hz]
G	empirical function	[-]
h	height of the trailing edge thickness	[m]
H	airfoil kinematic shape factor	[-]
I	turbulence intensity	[-]
k	wave number	[m^{-1}]
$\overline{k},\ \widehat{k}$	nondimensional wave number	[-]
$\Delta K_1,\ K_1, K_2$	empirical parameters of the BPM model	[-]
l	spanwise extent of the separation zone from blade tip	[m]
L	lift force	[N]
L_t	length scale	[m]
M	Mach number	[-]
M_c	Mach number past the trailing edge	[-]
r_e	effective observer distance	[m]
Re	Reynolds number	[-]
S^2	Sears function	[-]
St	Strouhal number	[-]
t_x	relative thickness of the airfoil at chordwise position x	[-]
U	local inflow velocity	[m/s]

continues on next page

Table 4.3 – continued from previous page

y	blade spanwise position	[m]
z	height above the ground	[m]
z_0	ground surface roughness	[m]

4.4.3 Introduction

The increasing penetration of wind energy into the electricity mix has been possible thanks to a constantly growing installed capacity, which has so far been mostly located on land. Land-based installations are, however, increasingly constrained by local ordinances and an often-limiting factor that comprises maximum allowable levels of noise. To further increase the number of land-based installations, it is important to develop accurate modeling tools to estimate the noise generated by wind turbines. This allows for a more accurate assessment of the noise emissions and the possibility to design quieter wind turbines.

Wind turbines emit two main sources of noise:

- Aeroacoustics noise from the interaction between rotor blades and the turbulent atmospheric boundary layer
- Mechanical noise from the nacelle component, mostly the gearbox, generator, and yaw mechanism.

This work targets the first class of noise generation and aims at providing a set of open-source models to estimate the aeroacoustics noise generated by an arbitrary wind turbine rotor. The models are implemented in Fortran and are fully coupled to the aeroservoelastic wind turbine simulator OpenFAST. The code is available in the GitHub repository of OpenFAST.[1] The code builds on the implementation of NAFNoise and the documentation presented in [aa-MM03] and [aa-Mor05]. OpenFAST is implemented as a modularization framework and the aeroacoustics model is implemented as a submodule of AeroDyn ([aa-MH05]).

The set of models is described in Section 4.4.4 and exercised on the noise estimate of the International Energy Agency (IEA) land-based reference wind turbine in Section 4.4.5. In Section 4.4.5, we also show a comparison to results obtained running the noise models implemented at the Technical University of Munich. This documentation closes with conclusions, an outlook on future work, and appendices, where the input files to OpenFAST are presented.

4.4.4 Aeroacoustics Noise Models

The aeroacoustics noise of wind turbine rotors emanates from pressure oscillations that are generated along the blades and propagate in the atmosphere. This source of noise has been historically simulated with models characterized by different fidelity levels. At lower fidelity, models correlated aeroacoustics noise with rotor thrust and torque ([aa-Low70][aa-Vit81]). At higher fidelity, three-dimensional incompressible computational fluid dynamics models are coupled with the Ffowcs Williams-Hawkings model to propagate pressure oscillations generated along the surface of the rotor blades to the far field ([aa-KGW+18]). The latter models are often only suitable to estimate noise at low frequency because capturing noise in the audible range, which is commonly defined between 20 (hertz) Hz and 20 kilohertz (kHz), requires a very fine space-time discretization with enormous computational costs.

For the audible range, a variety of models is available in the public domain, and [aa-SBCB18] offers the most recent literature review. These models have inputs that match the inputs and outputs of modern aeroservoelastic solvers, such as OpenFAST, and have therefore often been coupled together. Further, the computational costs of these acoustic models are similar to the costs of modern aeroservoelastic solvers, which has facilitated the coupling.

Models have targeted different noise generation mechanisms following the distinction defined by [aa-BPM89], and the mechanism of turbulent inflow noise. The latter represents a broadband noise source that is generated when a body of arbitrary shape experiences an unsteady lift because of the presence of an incident turbulent flow. For an airfoil, this

[1] https://github.com/OpenFAST/openfast

phenomenon can be interpreted as leading-edge noise. Turbulent inflow noise was the topic of multiple investigations over the past decades and, as a result, multiple models have been published ([aa-SBCB18]). The BPM model includes five mechanisms of noise generation for an airfoil immersed in a flow:

1. Turbulent boundary layer – trailing edge (TBL-TE)
2. Separation stall
3. Laminar boundary layer – vortex shedding
4. Tip vortex
5. Trailing-edge bluntness – vortex shedding.

For the five mechanisms, semiempirical models were initially defined for the NACA 0012 airfoil. The BPM model is still a popular model for wind turbine noise prediction, and subsequent studies have improved the model by removing some of the assumptions originally adopted. Recent studies have especially focused on the TBL-TE mechanism, which is commonly the dominant noise source of modern wind turbines. As a result, each noise source defined in the BPM model now has a variety of permutations.

The following subsections describe the details of each mechanism and the models implemented in this model of OpenFAST.

Turbulent Inflow

A body of any arbitrary shape, when immersed in a turbulent flow, generates surface pressure fluctuations. Over the years, several formulations of the turbulent inflow noise model have been developed ([aa-SBCB18]). In this model of OpenFAST, the formulation defined in [aa-MGM04] is adopted. The formulation is based on the model of Amiet ([aa-Ami75][aa-PA76]) and is presented in Section 4.4.4. Additionally, the user can activate the correction defined by [aa-MH05], which builds upon the Amiet model and accounts for the thickness of the airfoils adopted along the blade span. This second model is named Simplified Guidati and is presented in Section 4.4.4.

Amiet model

The formulation is based on work from [aa-Ami75] and [aa-PA76], and it represents the blade as a flat plate and neglects the shape of the airfoil.

The model starts by first computing the wave number, k_1, for a given frequency f:

$$k_1 = \frac{2\mathrm{f}}{U_1} \tag{4.21}$$

where U_1 is the incident inflow velocity on the profile. From k_1, the wave numbers $\overline{k}_1$ and $\widehat{k}_1$ are computed:

$$\overline{k}_1 = \frac{k_1 c_i}{2} \tag{4.22}$$

$$\widehat{k}_1 = \frac{k_1}{k_e} \tag{4.23}$$

where c_i is the local chord, and k_e is the wave number range of energy containing eddies, defined as:

$$k_e = \frac{3}{4L_t}. \tag{4.24}$$

$\mathrm{L_t}$ is the turbulent length scale, and many different formulations have been proposed over the years. As default implementation, L_t is defined following the formulation proposed in [aa-ZHS05]:

$$L_t = 25 z^{0.35} z_0^{-0.063} \tag{4.25}$$

where z is the height above the ground of the leading edge of section i at a given instant, t, while z_0 is the surface roughness. Note that setting L_t appropriately is a challenge, and advanced users of this model may want to validate this formulation against experimental data.

The value of sound pressure level (SPL) is expressed in one-third octave bands at the given frequency, f, originated at the given blade station, i, which can be computed as:

$$\mathrm{SPL}_{\mathrm{TI}} = 10\log_{10}\left(\rho^2 c^4 \frac{L_t d}{2r_e^2} M^5 I_1^2 \frac{\widehat{k}_1^3}{\left(1+\widehat{k}_1^2\right)^{\frac{7}{3}}} \overline{D}\right) + 78.4 \tag{4.26}$$

where ρ is the air density, c the speed of sound, d the blade element span, r_e the effective distance between leading edge and observer, M the Mach number, I_1 the turbulence intensity of the airfoil inflow, and $\overline{D}$ the directivity term. $\overline{D}$ is different below ($\overline{D}_l$) and above ($\overline{D}_h$) a certain frequency, which is named "cut-off" and defined as:

$$f_{\mathrm{co}} = \frac{10U_1}{\pi c_i}. \tag{4.27}$$

The formulations of $\overline{D}_h$ and $\overline{D}_l$ are presented in Section 4.4.4.

The current implementation offers two approaches to estimate I_1. The first one is through a user-defined grid of I_1; see Section 4.4.7. The second option is to have the code reconstructing I_1 from the turbulent wind grid, where the code computes the airfoil relative position of each blade section, i, at every time instant and, given the rotor speed, reconstructs the inflow component, I_1, of the turbulence intensity.

Two corrections to this model are also implemented. The first one comprises a correction for the angle of attack, α, in which the effect is neglected in the original formulation from [aa-Ami75] and Amiet and Peterson (1976). This correction is formulated as:

$$\mathrm{SPL}_{\mathrm{TI}} = \mathrm{SPL}_{\mathrm{TI}} + 10\log_{10}\left(1+9a^2\right). \tag{4.28}$$

The second correction is called low-frequency correction (LFC), and is formulated as:

$$S^2 = \left(\frac{2\pi\overline{k}_1}{\beta^2} + \left(1+2.4\frac{\overline{k}_1}{\beta^2}\right)^{-1}\right)^{-1} \tag{4.29}$$

$$LFC = 10S^2 M\overline{k}_1^2\beta^{-2} \tag{4.30}$$

$$\mathrm{SPL}_{\mathrm{TI}} = \mathrm{SPL}_{\mathrm{TI}} + 10\log_{10}\left(\frac{\mathrm{LFC}}{1+LFC}\right). \tag{4.31}$$

In (4.29) and (4.30), S^2 represents the squared Sears function, and β^2 is the Prandtl-Glauert correction factor, which is defined as:

$$\beta^2 = 1 - M^2. \tag{4.32}$$

It is worth stressing that numerous alternative formulations of the turbulent inflow noise model exist ([aa-SBCB18]), where the main differences comprise different definitions of L_t and k_1.

Simplified Guidati

Sound spectra are often overpredicted by the Amiet model implemented here. Guidatai ([aa-GBW+97]) derived a correction to the sound pressure levels by adding a term considering shape and camber of the airfoil profiles, but the method proved computationally too expensive for wind turbine simulations. Moriarty et al. ([aa-MGM05]) proposed a simplified model based on geometric characteristics of six wind turbine airfoils. The validity of the correction is limited to Mach numbers on the order of 0.1 0.2 and Strouhal number St below 75. St is defined based on airfoil chord and mean inflow velocity:

$$St = \frac{fc_i}{U_1}. \tag{4.33}$$

The formula for the correction to the noise spectra is provided in Eq. 4 in [aa-MGM05]:

$$t = t_{1\%} + t_{10\%} \tag{4.34}$$

$$\Delta SPL_{\mathrm{TI}} = -\left(1.123t + 5.317t^2\right)\left(2\pi St + 5\right) \tag{4.35}$$

where $t_{x\%}$ is the relative thickness of the profile at x position along the chord (i.e., 0% being the leading edge and 100% the trailing edge).

It should be highlighted here that a validation campaign was conducted in a wind tunnel on two-dimensional airfoils ([aa-MGM04]), returning a fairly poor match between the Simplified Guidati model and the experimental results. Therefore, a correction of +10 decibels (dB) on the SPL levels across the whole frequency spectrum was proposed. This correction is still implemented, but a validation at turbine level should assess the accuracy of the models for turbulent inflow. It should also be noted that the code currently does not check whether Mach and Strouhal numbers are within the range of validity of this model.

Turbulent Boundary Layer – Trailing Edge

Airfoils immersed in a flow develop a boundary layer, which at high Reynolds numbers is turbulent. When the turbulence passes over the trailing edge, noise is generated. This noise source was named TBL-TE in [aa-BPM89] and it is a relevant source of aeroacoustics noise for modern wind turbine rotors. Two formulations of TBL-TE noise are implemented in the code: (1) the original formulation from the BPM model, described in Section 4.4.4, and (2) a more recent model developed at the Dutch research institute, TNO, described in Section 4.4.4. Both models take as input the characteristics of the airfoil boundary layer. These must be provided by the user and are discussed in Section 4.4.7.

BPM

The SPL of the TBL-TE noise in the BPM model is made from three contributions:

$$\mathrm{SPL}_{TBL-TE} = 10\log_{10}\left(10^{\frac{\mathrm{SPL}_p}{10}} + 10^{\frac{\mathrm{SPL}_s}{10}} + 10^{\frac{\mathrm{SPL}_\alpha}{10}}\right) \tag{4.36}$$

where the subscripts $_\mathrm{p}$, $_\mathrm{s}$, and refer to the contributions of pressure side, suction side, and angle of attack, respectively. The equations describing the three contributions are described in great detail in Section 5.1.2, in [aa-BPM89], and are summarized here.

For the suction and pressure contributions, the equations are:

$$\mathrm{SPL}_p = 10\log_{10}\left(\frac{\delta_p^* M^5 d\overline{D}_h}{r_e^2}\right) + A\left(\frac{\mathrm{St}_p}{\mathrm{St}_1}\right) + (K_1 - 3) + \Delta K_1 \tag{4.37}$$

$$\mathrm{SPL}_s = 10\log_{10}\left(\frac{\delta_s^* M^5 d\overline{D}_h}{r_e^2}\right) + A\left(\frac{\mathrm{St}_s}{\mathrm{St}_1}\right) + (K_1 - 3)\,. \tag{4.38}$$

The terms in the equations, which are also described in the nomenclature at the beginning of this document, list δ^* as the boundary layer displacement thickness on either side of the airfoil, St, as the Strouhal number based on δ^*, and A, A', B, ΔK_1, K_1, and K_2 as empirical functions based on St.

For the angle-of-attack contribution, a distinction is made above and below the stall angle, which in the original BPM model is set equal to 12.5 degrees, whereas it is here assumed to be the actual stall angle of attack of the airfoil at blade station i. Below stall, SPL$_\alpha$ is equal to:

$$\text{SPL}_\alpha = 10\log_{10}\left(\frac{\delta_s^* M^5 d\overline{D}_h}{r_e^2}\right) + B\left(\frac{\text{St}_s}{\text{St}_2}\right) + K_2. \tag{4.39}$$

At angles of attack above the stall point, the flow along the profile is fully separated and noise radiates from the whole chord. SPL$_p$ and SPL$_s$ are then set equal to -∞, whereas SPL$_\alpha$ becomes:

$$\text{SPL}_\alpha = 10\log_{10}\left(\frac{\delta_s^* M^5 d\overline{D}_l}{r_e^2}\right) + A'\left(\frac{\text{St}_s}{\text{St}_2}\right) + K_{2.} \tag{4.40}$$

Notably, above stall the low-frequency directivity $\overline{D}_l$ is adopted in Eqs. 18 and 19 (see Section 4.4.4).

TNO model

The TNO model is a more recent model to simulate the noise emitted by the vortices shed at the trailing edge of the blades and was formulated by Parchen ([aa-Par98]). The implementation adopted here is the one described in Moriarty et al. (2005). The TNO model uses the spectrum of the wave number, $\overline{k}$, of unsteady surface pressures to estimate the far-field noise. The spectrum, P, is assumed to be:

$$\begin{aligned} P\left(k_1, k_3, \omega\right) = 4\rho_0^2 \frac{k_1^2}{k_1^2+k_3^2}\int_0^{10\frac{\omega}{Mc}} L_2\overline{u_2^2}\left(\frac{\partial U_1}{\partial x_2}\right)^2 \phi_{22}\left(k_1,k_3,\omega\right) \\ \phi_m\left(\omega - U_c\left(x_2\right)k_1\right)e^{\left(-2|\overline{k}|x_2\right)}dx_2. \end{aligned} \tag{4.41}$$

In the equation, the indices 1, 2, and 3 refer to the directions parallel to the airfoil chord, normal to the airfoil chord, and along span, respectively; ϕ_{22} is the vertical velocity fluctuation spectrum; ϕ_m is the moving axis spectrum; and U_c is the convection velocity of the eddies along the trailing edge. Lastly, L_2 is the vertical correlation length, perpendicular to the chord length, which indicates the vertical extension of the vortices that convect over the trailing edge. In this work, L_2 is assumed equal to the mixing length, L_m (Moriarty et al. 2005). This decision is partially arbitrary, and dedicated research should better assess the correct integral length to be adopted within the TNO model.

From P, the far-field spectrum, $S\left(\omega\right)$, is computed as:

$$S\left(\omega\right) = \frac{d\overline{D}_h}{4\pi r_e^2}\int_0^\delta \frac{\omega}{ck_1}P\left(k_1, 0, \omega\right)\text{dk}_1. \tag{4.42}$$

The implementation of the TNO model is identical to the one described in [aa-MGM05]. The inputs to the model are generated from the boundary layer characteristics provided by the user (see Section 4.4.7).

Laminar Boundary Layer – Vortex Shedding

Another source of airfoil self-noise noise included in the BPM model is the noise generated by a feedback loop between vortices being shed at the trailing edge and instability waves in the laminar boundary layer. This noise is typically distributed on a narrow band of frequencies and occurs when the boundary layer of the airfoil remains laminar. This may occur in the inboard region of smaller wind turbines, where the Reynolds number can be smaller than 1 million, but hardly occurs in modern rotors that operate at a Reynolds number one order of magnitude larger. The formula to estimate the noise spectrum in a one-third-octave presentation is:

$$\begin{aligned} \text{SPL}_{LBL-VS} = 10\log_{10}\left(\frac{\delta_p M^5 d\overline{D}_h}{r_e^2}\right) + G_1\left(\frac{St'}{St'_{\text{peak}}}\right) \\ +G_2\left[\frac{\text{Re}_c}{\left(\text{Re}_c\right)_0}\right] + G_3\left(\alpha_*\right) \end{aligned} \tag{4.43}$$

where G represents empirical functions, St'_{peak} is the peak Strouhal number function of Re_c, which is the Reynolds number at chord, c_i. The subscript $_0$ refers to a reference Reynolds number that is a function of the angle of attack (Brooks et al. 1989).

Tip Vortex

The vortices generated at blade tips are another source of noise of the BPM model. Although rarely relevant in modern wind turbines, the possibility to include this noise source is offered. The sound pressure level is estimated as:

$$\text{SPL}_{\text{Tip}} = 10 \log_{10} \left(\frac{M^2 M_{\max}^2 l^2 \overline{D}_h}{r_e^2} \right) - 30.5 \left(\log_{10} St^{''} + 0.3 \right)^2 + 126 \tag{4.44}$$

where $M_{\max} = M_{\max} \left(\alpha_{\text{tip}} \right)$ is the maximum Mach number, measured near the blade tip within the separated flow region that is assumed to depend on α_{tip}, which is the angle of attack at the tip; l is the spanwise extent of the separation zone; and St''' is the Strouhal number based on l. For a round shape of the tip, l is estimated as:

$$l = c_i 0.008 \alpha_{\text{tip}} \tag{4.45}$$

where α_{tip} is the angle of attack of the tip region to the incoming flow. For a square tip, the BPM model estimates l based on the quantity, α'_{tip}, which is defined as:

$$\alpha^{'}_{\text{tip}} = \left[\left(\frac{\frac{\partial L'}{\partial y}}{\left(\frac{\partial L'}{\partial y} \right)_{\text{ref}}} \right)_{y \to tip} \right] \alpha_{\text{tip}} \tag{4.46}$$

where L' is the lift per unit span along the blade at position y. For α'_{tip} between 0 and 2 degrees, l becomes:

$$l = c_i \left(0.0230 + 0.0169 \alpha^{'}_{\text{tip}} \right), \tag{4.47}$$

while for α'_{tip} larger than 2 degrees, l is:

$$l = c_i \left(0.0378 + 0.0095 \alpha^{'}_{\text{tip}} \right). \tag{4.48}$$

However, it must be noted that, unfortunately, α_{tip} is not a reliable output of standard aeroelastic models and the impossibility to accurately determine α_{tip} weakens the formulation of the tip vortex noise.

Trailing-Edge Bluntness – Vortex Shedding

Lastly, wind turbine blades are often characterized by a finite height of the trailing edge, which generates noise as a result of vortex shedding. The frequency and amplitude of this noise source depends on the geometry of the trailing edge and is typically characterized by a tonal nature. Adopting flatback and truncated airfoils far outboard along the blade may strengthen this noise source. When this noise source is activated, the user is asked to provide the distribution along the blade span of the blunt thickness of the trailing edge, h, and the solid angle between the suction and pressure sides of the airfoil, Ψ (see Section 4.4.7). h and Ψ are inputs to the equation:

$$\begin{aligned} \text{SPL}_{TEB-VS} = 10 \log_{10} \left(\frac{\delta_p^* M^5 d \overline{D}_h}{r_e^2} \right) + G_4 \left(\frac{h}{\delta_{\text{avg}}^*}, \Psi \right) \\ + G_5 \left(\frac{h}{\delta_{\text{avg}}^*}, \Psi, \frac{St''}{St''_{\text{peak}}} \right). \end{aligned} \tag{4.49}$$

In the equation, δ_{avg}^* is the average displacement thickness for both sides of the airfoil. Note that this noise source is very sensitive to h and Ψ, which, therefore, should be estimated accurately.

Directivity

The position of one or more observers is specified by the user, as described in Section 4.4.7. The directivity from the BPM model is adopted in this implementation ([aa-BPM89]). The directivity term, $\overline{D}$, corrects the SPL depending on the relative position of the observer to the emitter. The position is described by the spanwise directivity angle, Φ_e, and by the chordwise directivity angle, Θ_e, which are schematically represented in Fig. 4.11 and defined as:

$$\Phi_e = \text{atan}\left(\frac{z_e}{y_e}\right) \tag{4.50}$$

$$\Theta_e = \text{atan}\left(\frac{y_e \bullet \cos\left(\Phi_e\right) + z_e \bullet \sin\left(\Phi_e\right)}{x_e}\right) \tag{4.51}$$

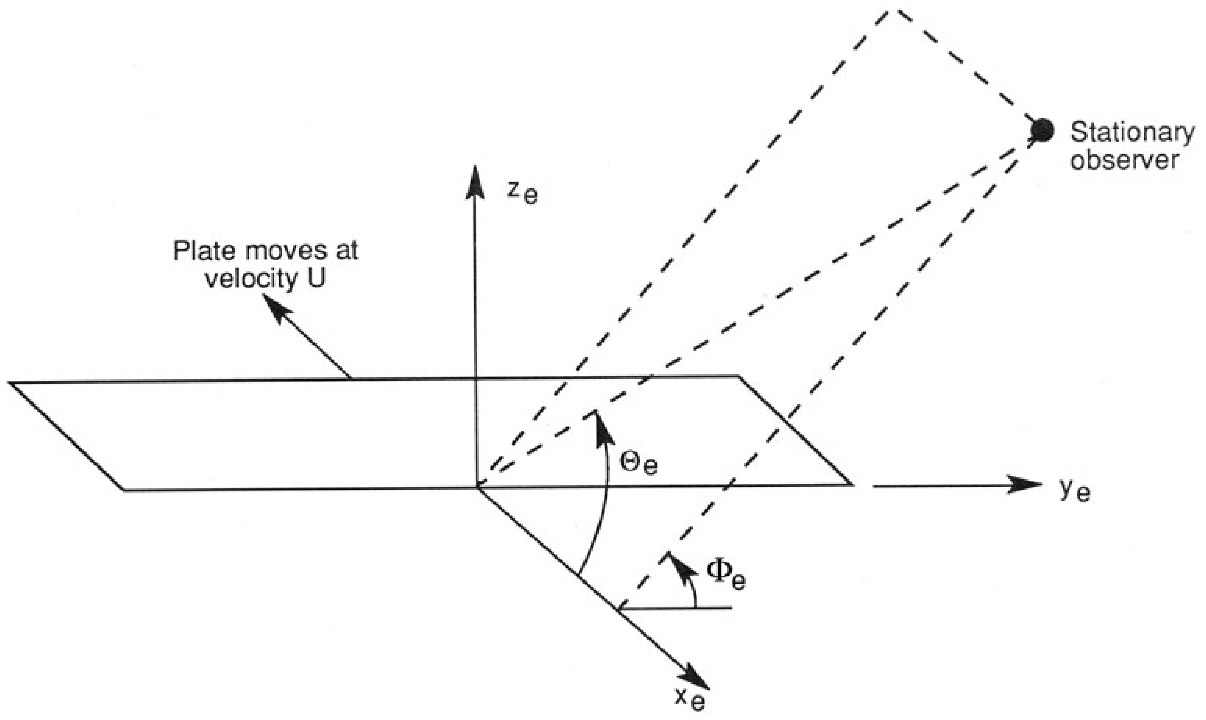


Fig. 4.11: Angles used in the directivity function ([aa-BPM89][aa-MM03])

The reference axis is located at each blade node and x_e is aligned with the chord, y_e is aligned with the span pointing to the blade tip, and z_e is aligned toward the airfoil suction side. Note that in OpenFAST the local airfoil-oriented reference system is used, and a rotation is applied.

Given the angles Θ_e and Φ_e, at high frequency, $\overline{D}$ takes the expression:

$$\overline{D}_h\left(\Theta_e, \Phi_e\right) = \frac{2\sin^2\left(\frac{\Theta_e}{2}\right)\sin^2\Phi_e}{\left(1 + M\cos\Theta_e\right)\left(1 + \left(M - M_c\right)\cos\Theta_e\right)^2} \tag{4.52}$$

where M_c represents the Mach number past the trailing edge and that is here for simplicity assumed equal to 80% of free-stream M. At low frequency, the equation becomes:

$$\overline{D}_l\left(\Theta_e, \Phi_e\right) = \frac{\sin^2\ \Theta_e\ \sin^2\Phi_e}{\left(1 + M\cos\Theta_e\right)^4}. \tag{4.53}$$

Each model distinguishes a different value between low and high frequency. For the TI noise model, the shift between low and high frequency is defined based on $\overline{k}_1$. For the TBL-TE noise, the model differences instead shift between below and above stall, where $\overline{D}_h$and $\overline{D}_l$ are used, respectively.

A-Weighting

The code offers the possibility to weigh the aeroacoustics outputs by A-weighting, which is an experimental coefficient that aims to take into account the sensitivity of human hearing to different frequencies. The A-weight, A_w, is computed as:

$$A_w = \frac{10\log\left(1.562339\frac{f^4}{(f^2+107.65265^2)(f^2+737.86223^2)}\right)}{\log 10} + \frac{10\log\left(2.422881e16\frac{f^4}{(f^2+20.598997^2)^2(f^2+12194.22^2)^2}\right)}{\log 10} \tag{4.54}$$

The A-weighting is a function of frequency and is added to the values of sound pressure levels:

$$SPL_{A_w} = SPL + A_w \tag{4.55}$$

4.4.5 Model Verification

Reference Wind Turbine

The noise model of OpenFAST is exercised by simulating the aeroacoustics noise emissions of the IEA Wind Task 37 land-based reference wind turbine ([aa-BTD+19]). The main characteristics of the reference wind turbine are presented in Table 4.4.

Table 4.4: Main Characteristics of the IEA Wind Task 37 Land-Based Reference Wind Turbine

Data	Value	Data	Value
Wind class	International Electrotechnical Commision 3A	Rated electrical power	3.37 megawatts
Rated aerodynamic power	3.6 megawatts	Drivetrain & generator efficiency	93.60%
Rotor diameter	130 meters	Hub height	110 meters
Cut-in wind speed	4 meters/second	Cut-out wind speed	25 meters/second
Rotor cone angle	3 degrees	Nacelle tilt angle	5 degrees
Max blade tip speed	80 meters/second	Rated tip-speed ratio	8.16
Maximum aerodynamic Cp	0.481	Rated rotor speed	11.75 revolutions per minute

The OpenFAST model of the wind turbine is available at https://github.com/OpenFAST/r-test and is optionally coupled to the Reference OpenSource Controller.[2]

Code-to-Code Comparison

A detailed code-to-code comparison was conducted to verify the implementation of the noise models linked to OpenFAST with the implementation available at the Wind Energy Institute of the Technical University of Munich, Germany. The latter is described in Sucameli ([aa-SBCB18]) and is implemented in the wind turbine design framework Cp-Max, which adopts the multibody-based aeroservoelastic solver Cp-Lambda.

The comparison is conducted for the main noise sources—turbulent inflow and the TBL-TE noise—for both the single airfoil profile and full turbine. This helped resolve a few implementation mistakes and small inconsistencies. The comparison is performed with a steady wind of 8 meters per second (m/s), no shear, a rated pitch angle of 1.17 degrees (deg), and a fixed rotor speed of 10.04 revolutions per minute (rpm). A fixed value of 0.1 is assumed for the incident turbulent intensity, I_1.

Fig. 4.12 shows the predictions in terms of SPL for the Amiet model with the angle-of-attack correction from OpenFAST, the Simplified Guidati model generated by OpenFAST, and the Amiet model from Cp-Max.

The two implementations of the turbulent inflow Amiet model return a perfect match between OpenFAST and Cp-Max. The chosen scenario sees the blade operating at optimal angles of attack and, therefore, the effect of the angle of attack correction is negligible. The plots also show the great difference between the Amiet model and the Simplified Guidati model. It may be useful to keep in mind that the Simplified Guidati model has, in the past, been corrected with a factor of +10 dB, which is applied here.

[2] https://github.com/NREL/ROSCO

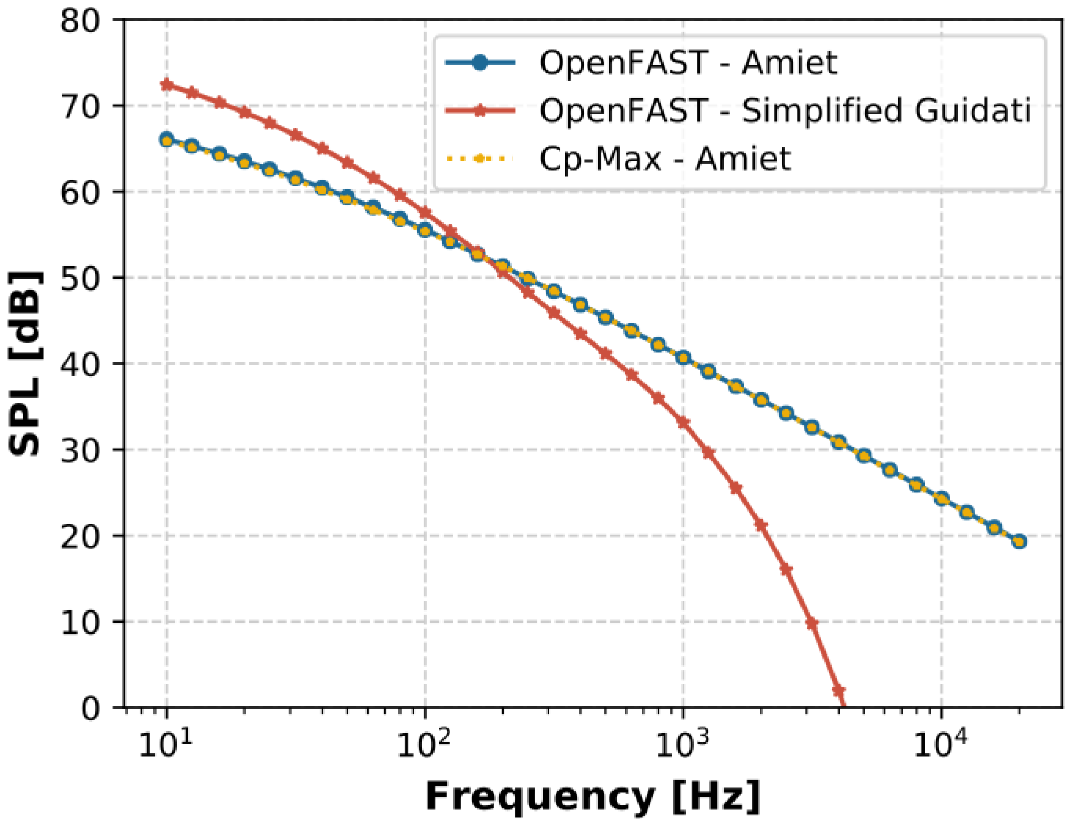


Fig. 4.12: Code-to-code comparison for the TI models

For the same inflow and rotor conditions, the BPM and TNO TBL-TE noise models are compared in Fig. 4.13. The match is again satisfactory, although slightly larger differences emerge that are attributed to differences in the angles of attack between the two aeroelastic solvers and in different integration schemes in the TNO formulations.

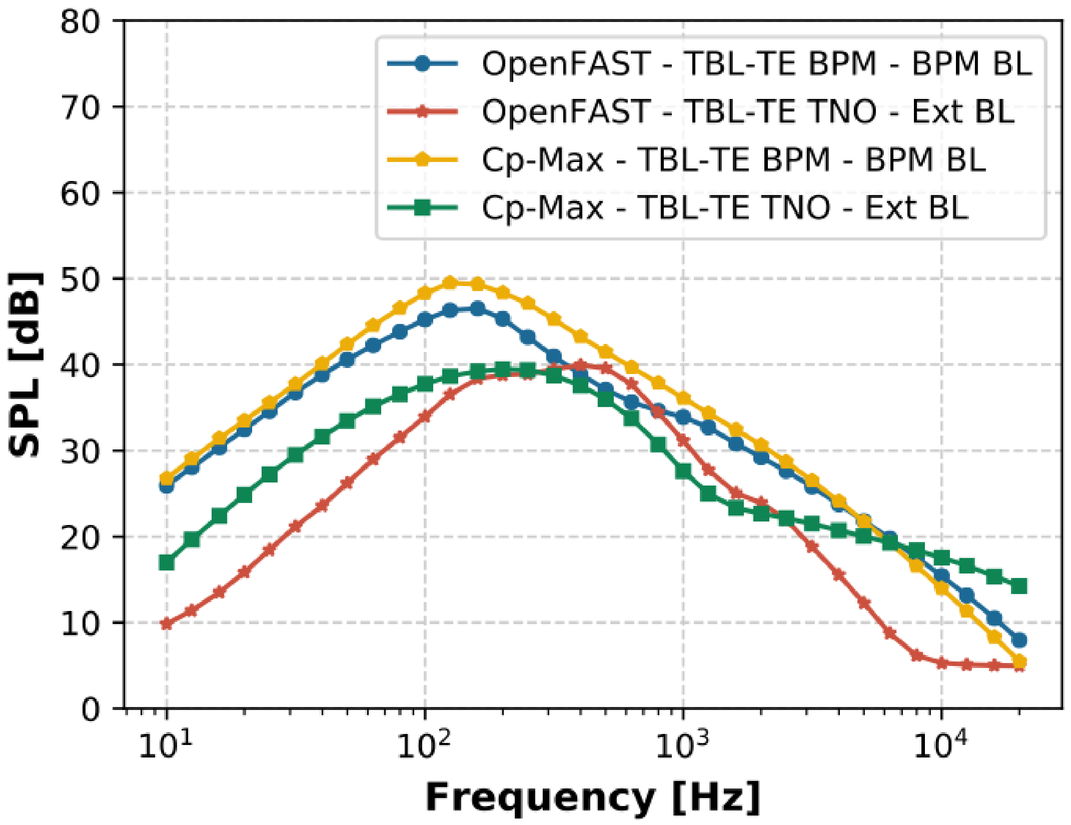


Fig. 4.13: Code-to-code comparison for the BPM and TNO TBL-TE models. The boundary layer properties are estimated from either the BPM model (BPM BL) or defined by the user (Ext BL)

The last comparison looked at the directivity models and the overall sound pressure levels at various observer locations. Simulations are run distributing 200 observers in a horizontal square of 500 meters (m) by 500 m (see Fig. 4.14). The noise is computed from the Amiet and the BPM turbulent boundary layer-trailing edge models. The code-to-code comparison returns similar predictions between OpenFAST and Cp-Max. The comparison is shown in Fig. 4.15.

The main conclusion of this code-to-code comparison is that, to the best of authors' knowledge, the models are now implemented correctly and generate similar SPL and overall SPL levels for any arbitrary observer. Nonetheless, it is clear that all of the presented models are imperfect, and improvements could be made both at the theoretical implementation levels.

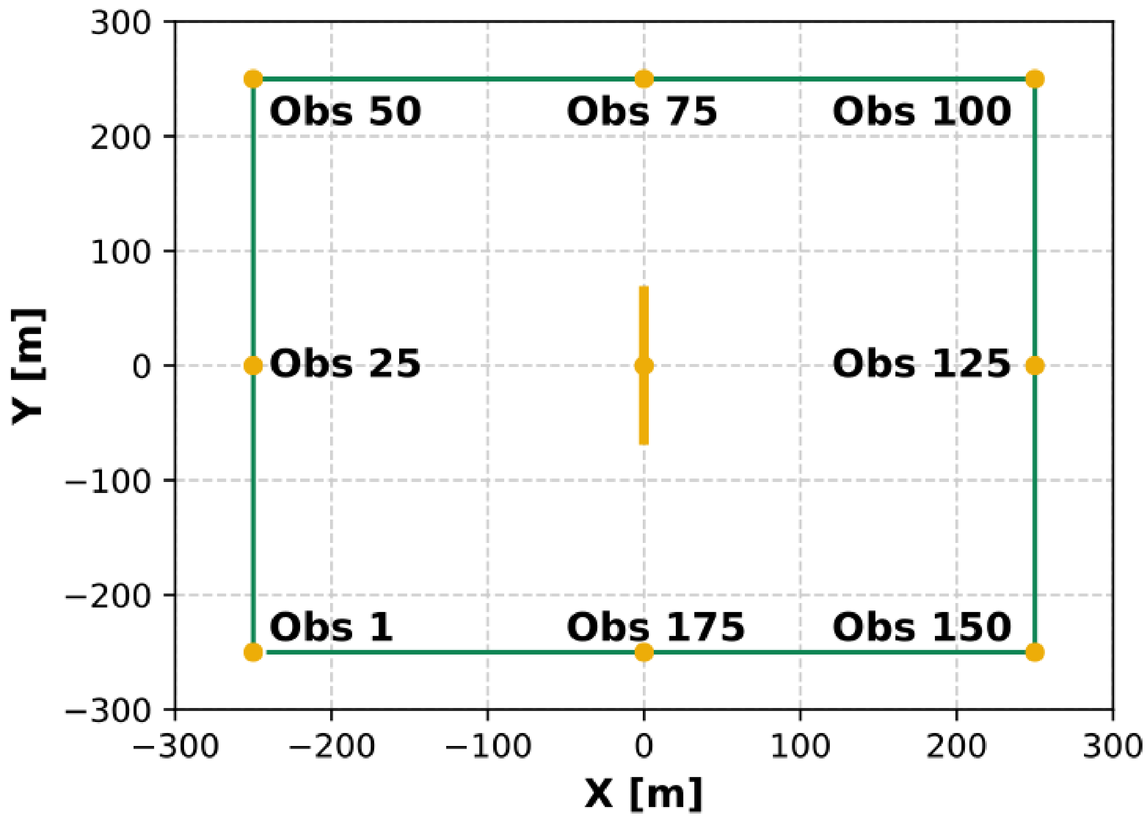


Fig. 4.14: Location and numbering of the observers

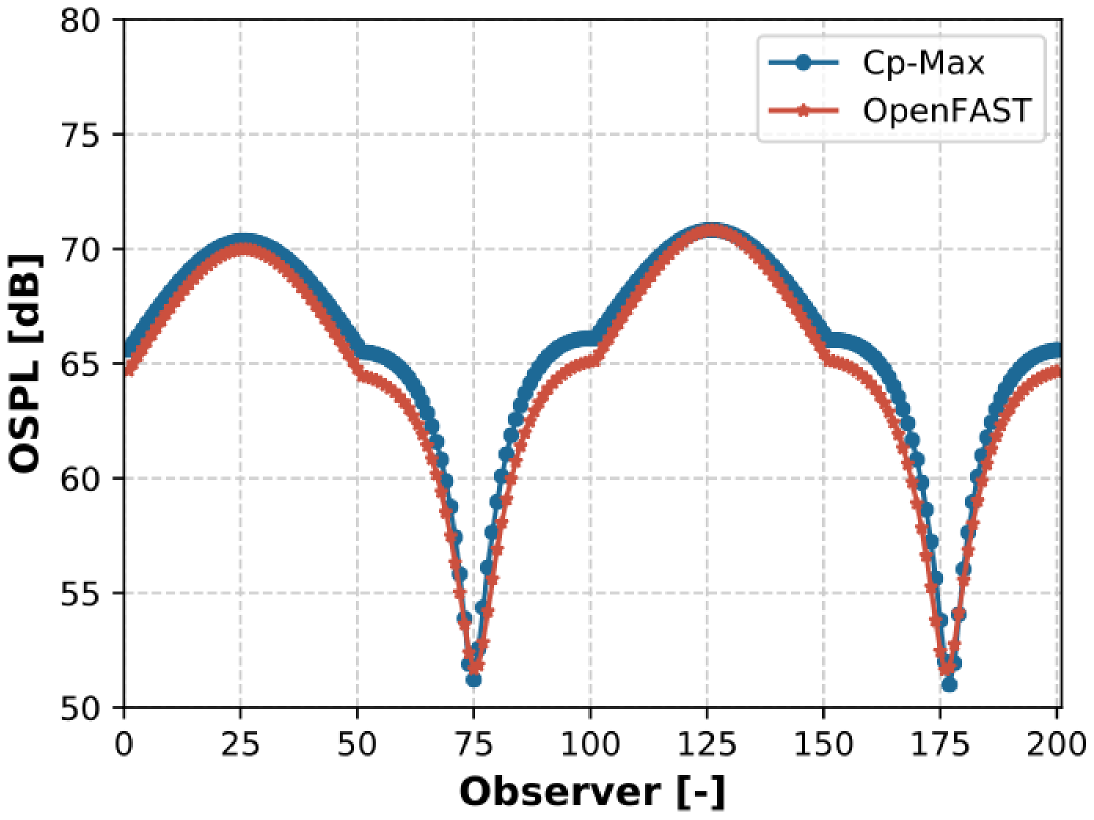


Fig. 4.15: Comparison of overall sound pressure levels for the observers distributed, as shown in the previous figure

Model Usage

The aeroacoustics model of OpenFAST has four options for the outputs:

1. Overall sound pressure level (dB/A-weighted decibels [dBA])—one value per time step per observer is generated
2. Total sound pressure level spectra (dB/dBA)—one spectrum per time step per observer is generated between 10 Hz and 20 kHz
3. Mechanism-dependent sound pressure level spectra (dB/dBA)—one spectrum per active noise mechanism per time step per observer is generated between 10 Hz and 20 kHz.
4. Overall sound pressure level (dB/A-weighted decibels [dBA])—one value per blade per node per time step per observer is generated

The overall SPL from the first option can be used to plot directivity maps of the noise. An example, which was generated using a Python script,[3] is shown in Fig. 4.16. The noise map, which shows the overall SPL averaged over 1 rotor revolution, is generated for a steady wind speed of 8 m/s, a fixed rotor speed of 10.04 rpm, and a 1.17-deg pitch angle. In a horizontal circle of 500 m in diameter, 1681 observers are placed at a 2-m height. Only the Simplified Guidati and the BPM TBL-TE noise models are activated.

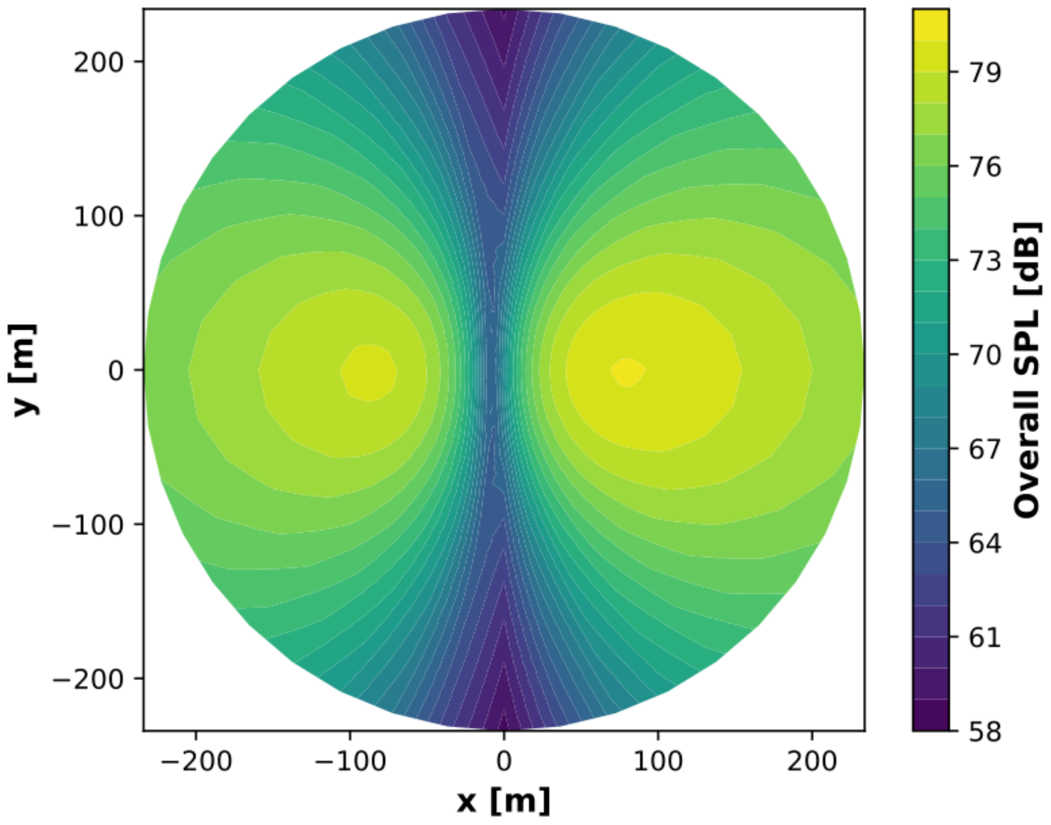


Fig. 4.16: Map of the overall SPL of the reference wind turbine at a 2-m height from Simplified Guidati and BPM TBL-TE noise models. The wind turbine is located at x=0, y=0. A steady wind of 8 m/s blows from left (-x) to right (+x).

[3] https://github.com/OpenFAST/python-toolbox

The second output can be used to generate SPL spectra. These spectra can be computed for various observers and optionally A-weighted to account for human hearing. Fig. 4.17 shows the total SPL spectra computed for the same rotor conditions of the previous example. The A-weight greatly reduces the curve at frequency below 1,000 Hz while slightly increasing those between 1 kHz and 8 kHz.

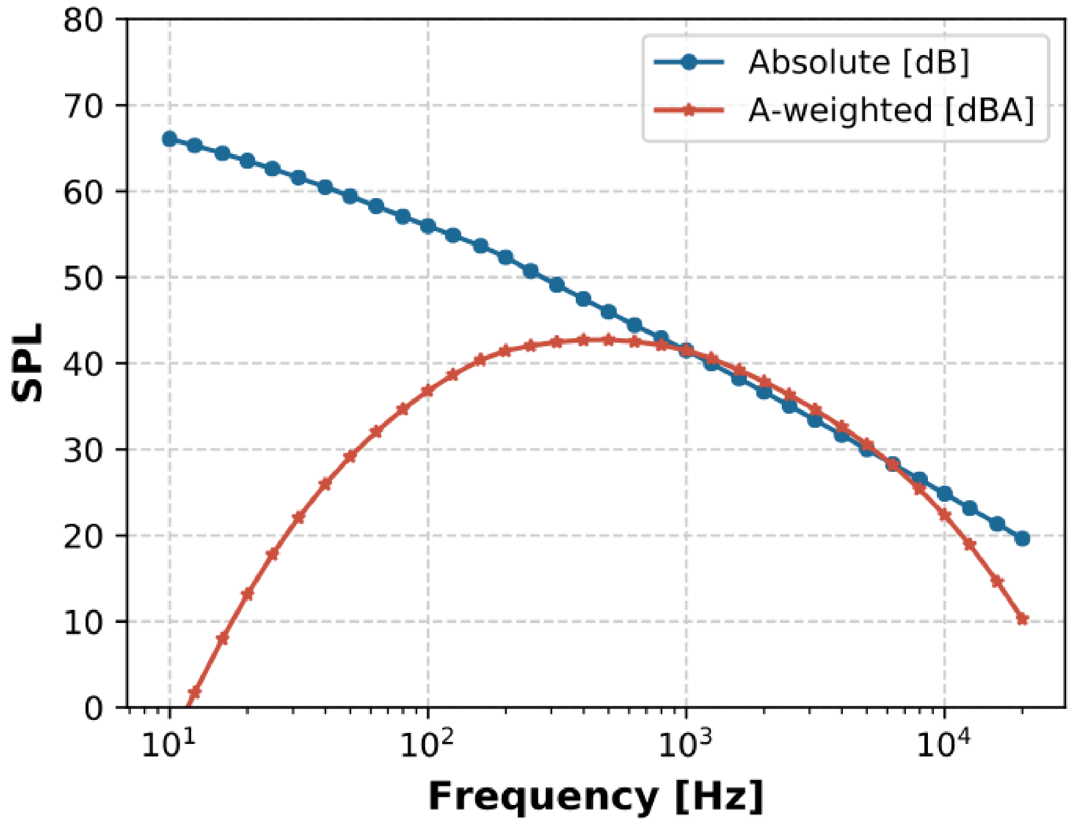


Fig. 4.17: Comparison between absolute and A-weighted SPL

The third output distinguishes the SPL spectrum per mechanism. Fig. 4.18 shows the various SPL spectra estimated by each noise model for the same rotor conditions reported earlier. The total spectrum is visibly dominated by the turbulent inflow, TBL-TE, and trailing-edge bluntness noise mechanisms. Notably, the latter is extremely sensitive to its inputs, Ψ and h. The reference wind turbine is a purely numerical model, and these quantities have been arbitrarily set. Users should pay attention to these inputs when calling the trailing-edge bluntness model. Consistent with literature, the laminar boundary layer-vortex shedding and tip vortex noise mechanisms have negative dB values and are, therefore, not visible. Notably, these spectra are not A-weighted, but users can activate the flag and obtain A-weighted spectra.

Finally, the fourth output can be used to visualize the noise emission across the rotor. Fig. 4.19 shows the noise generation of the rotor as seen from an observer located 175 meters downwind at a height of 2 meters. The map is generated by plotting the overall SPL generated by one blade during one rotor revolution. The plot shows that higher noise is observed when the blade is descending (the rotor from behind is seen rotating counterclockwise). This effect, which matches the results shown in [aa-MM03], is explained by the asymmetry of (4.51). Noise is indeed higher when the observer faces the leading edge of an airfoil (high Θ_e), than when it faces the trailing edge (low Θ_e).

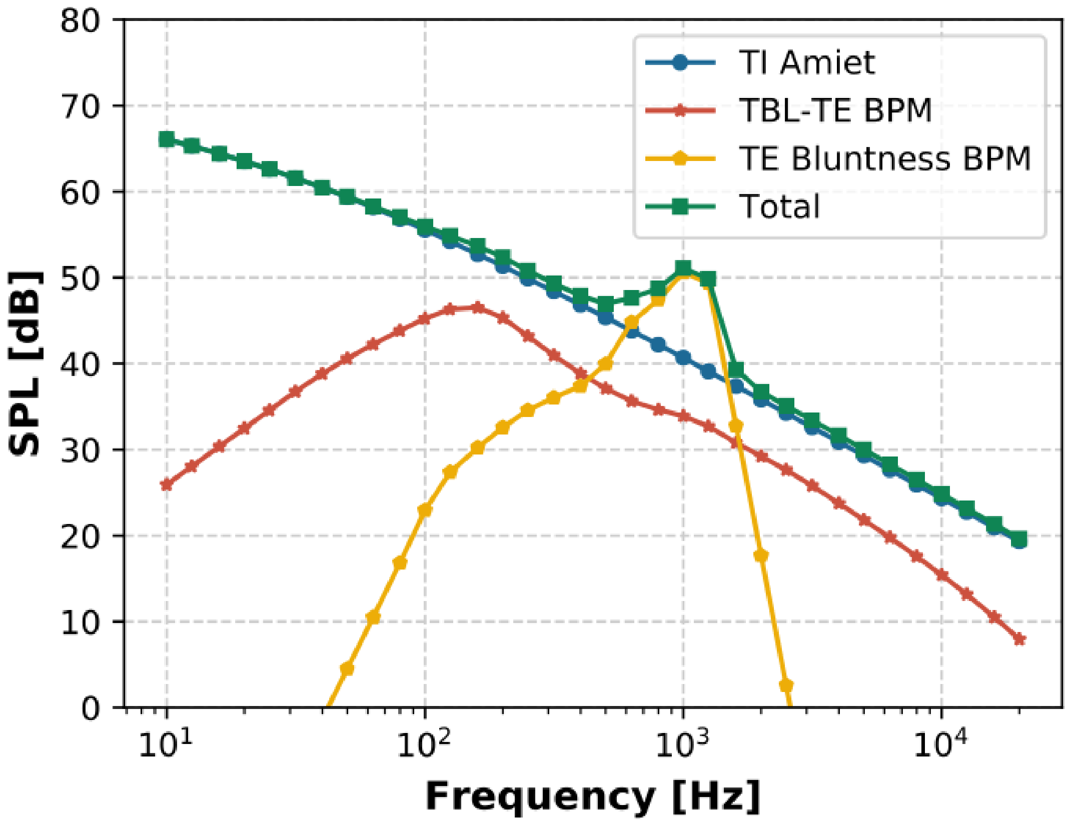


Fig. 4.18: Nonweighted SPL spectra of the various noise mechanisms

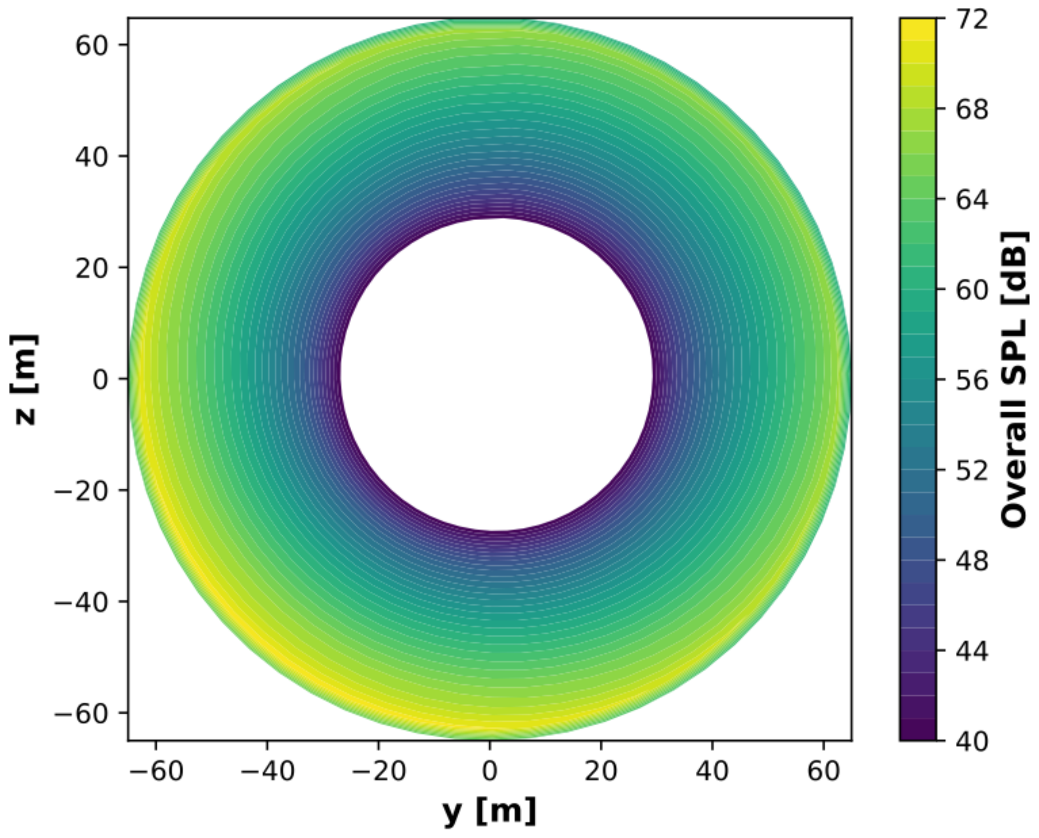


Fig. 4.19: Map of the overall SPL of the rotor of the reference wind turbine from Simplified Guidati and BPM TBL-TE noise models. The observer is located 175 meters downwind at a height of 2 meters.

4.4.6 Conclusions

This document describes a set of frequency-based aeroacoustics models coupled to the open-source aeroservoelastic solver OpenFAST. The goal of these models is to predict the aeroacoustics emissions of wind turbine rotors. The document shows a code-to-code comparison between the models coupled to OpenFAST and the models implemented at the Technical University of Munich and coupled to the aeroservoelastic solver Cp-Lambda. The comparison is performed simulating the aeroacoustics emissions of the IEA Wind Task 37 land-based reference wind turbine. The results show a good agreement between the two implementations. The same turbine model is later used to exercise the aeroacoustics model showcasing its capabilities. Finally, the appendices describe the entries of the input files of OpenFAST to run the aeroacoustics analysis.

Future work will focus on the validation of the aeroacoustics models. In parallel, propagation models will be investigated and implemented. Finally, attention will be dedicated to infrasound noise and to the time-domain models that can simulate it.

4.4.7 Using the Aeroacoustics Model in AeroDyn

A live version of this documentation is available at https://openfast.readthedocs.io/. To run the aeroacoustics model, the flag **CompAA** needs to be set to **True** at line 13 of the AeroDyn15 main input file in the inputs block **General Options**. When the flag is set to **True**, the following line must include the name of the file containing the inputs to the aeroacoustics model, which is discussed in Section 4.4.7.

```
1  ------- AERODYN v15.03.\* INPUT FILE ------------------------------------------------
2  IEA Wind Task 37 land-based reference wind turbine
3  ====== General Options ===================================================
4  False          Echo        - Echo the input to "<rootname>.AD.ech"? (flag)
5  "default"      DT_AA       - Time interval for aerodynamic calculations {or "default"}
   ↪ (s)
6  1              WakeMod     - Type of wake/induction model (switch) {0=none, 1=BEMT}
7  2              AFAeroMod   - Type of blade airfoil aerodynamics model (switch
8  0              TwrPotent   - Type tower influence on wind around the tower (switch)
9  False          TwrShadow   - Calculate tower influence on wind (flag)
10 False          TwrAero     - Calculate tower aerodynamic loads? (flag)
11 False          FrozenWake  - Assume frozen wake during linearization? (flag
12 False          CavitCheck  - Perform cavitation check? (flag)
13 True           CompAA      - Flag to compute AeroAcoustics calculation
14 "AeroAcousticsInput.dat"   AA_InputFile
15 ====== Environmental Conditions ==========================================
16 1.225.         AirDens     - Air density (kg/m^3)
17
18 File continues...
```

Main Input File

The aeroacoustics main input file comprises a series of inputs and flags that should be set appropriately depending on the analysis that should be run. These are split into the subfields General Options, Aeroacoustics Models, Observer Input, and Outputs.

Starting from the General Options, these are:

- **Echo** – True/False: option to rewrite the input file with the correct template
- **DT_AA** – Float: time step of the aeroacoustics computations. Only multiples of the time step **DTAero** of AeroDyn can be used. If set to default, the time step DTAero is adopted.
- **AAStart** – Float: time after which the AeroAcoustics module is run.

- **BldPrcnt** – Float: percentage value of blade span measured from blade tip that contributes to the noise emissions; 100% corresponds to the entire blade from tip to root.

The field Aeroacoustics Models lists all the flags for the actual noise models:

- **TIMod** – Integer 0/1/2: flag to set the turbulent inflow noise model; 0 turns it off, 1 corresponds to the Amiet model discussed in Section 4.4.4, and 2 corresponds to the Simplified Guidati model presented in Section 4.4.4.
- **TICalcMeth** – Integer 1/2: flag to set the calculation method for the incident turbulence intensity. When set to 1, incident turbulence intensity is defined in a user-defined grid; see Section 4.4.7. When set to 2, incident turbulence intensity is estimated from the time history of the incident flow.
- **TICalcTabFile** – String: name of the text file with the user-defined turbulence intensity grid; see Section 4.4.7.
- **SurfRoughness** – Float: value of z_0 used to estimate L_t in the Amiet model.
- **TBLTEMod** – Integer 0/1/2: flag to set the TBL-TE noise model; 0 turns off the model, 1 uses the Brooks-Pope-Marcolini (BPM) airfoil noise model (see Section 4.4.4), and 2 uses the TNO model described in Section 4.4.4.
- **BLMod** – Integer 1/2: flag to set the calculation method for the boundary layer characteristics; 1 uses the simplified equations from the BPM model, 2 loads the files as described in Section 4.4.7. Only used if **TBLTEMod** is different than zero.
- **TripMod** – Integer 0/1/2: if BLMod is set to 1, different semiempirical parameters are used for a nontripped boundary layer (**TRipMod=0**), heavily tripped boundary layer (**TRipMod=1**), or lightly tripped boundary layer (**TRipMod=2**); 2 is typically used for operational wind turbines, whereas 1 is often used for wind tunnel airfoil models.
- **LamMod** – Integer 0/1: flag to activate the laminar boundary layer – vortex shedding model, presented in Section 4.4.4.
- **TipMod** – Integer 0/1: flag to activate the tip vortex model, presented in Section 4.4.4.
- **RoundedTip** – True/False: if **TipMod=1**, this flag switches between a round tip (True) and a square tip (False), see Section 4.4.4.
- **Alprat** – Float: value of the slope of the lift coefficient curve at blade tip; see Section 4.4.4.
- **BluntMod** – Integer 0/1: flag to activate (**BluntMod=1**) the trailing-edge bluntness – vortex shedding model, see Section 4.4.4. If the flag is set to 1, the trailing-edge geometry must be specified in the file(s) listed in the field Blade Properties.

Next, the field Blade Properties lists three file names, often but not necessarily identical, which contain the distributed properties describing the detailed geometry of the trailing edge. These are described in Section 4.4.7.

The field Observer Locations contains the path to the file where the number of observers (NrObsLoc) and the respective locations are specified; see Section 4.4.7.

Finally, the set Outputs contains a few options for the output data:

- **AWeighting** – True/False: flag to set whether the sound pressure levels are reported with (True) or without (False) the A-weighting correction; see Section 4.4.5.
- **NAAOutFile** – Integer 1/2/3: flag to set the desired output file. When set to 1, a value of overall sound pressure level at every **DT_AA** time step per observer is printed to file. When set to 2, the first output is accompanied by a second file where the total sound pressure level spectrum is printed per time step per observer. When set to 3, the two first outputs are accompanied by a third file where the sound pressure level spectrum per noise mechanism is printed per time step per observer. When set to 4, a fourth file is generated with the values of overall sound pressure levels per node, per blade, per observer, and per time step.
- The following line contains the file name used to store the outputs. The file name is attached with a 1, 2, 3, and 4 flag based on the **NAAOutFile** options.

The file must be closed by an END command.

```
------- AeroAcoustics Module v1.00.* INPUT FILE -------------------------------------
↪----------
IEA task 37 RWT turbine -- https://github.com/IEAWindTask37/IEA-3.4-130-RWT
======  General Options ␣
↪==============================================================================
False         Echo          - Echo the input to "<rootname>.AD.ech"?  (flag)
0.1           DT_AA         - Time interval for aeroacoustics calculations (s), must be␣
↪a multiple of DT_Aero from AeroDyn15  (or "default")
0             AAStart       - Time after which the AeroAcoustics module is run (s)
70            BldPrcnt      - Percentage of the blade span, starting from the tip, that␣
↪will contribute to the overall noise levels. (float)
======  Aeroacoustic Models ␣
↪==============================================================================
2             TIMod         - Turbulent Inflow noise model  {0: none, 1: Amiet 2: Amiet␣
↪+ Simplified Guidati} (switch)
1             TICalcMeth    - Method to estimate turbulence intensity incident to the␣
↪profile {1: given table, 2: computed on the fly} (switch) [Only used if TIMod!=0]
"TIGrid_InVerify.txt"   TICalcTabFile - Name of the file containing the table for␣
↪incident turbulence intensity (-) [Only used if TiCalcMeth == 1]
0.5           SurfRoughness- Surface roughness value used to estimate the turbulent␣
↪length scale in Amiet model (m)
1             TBLTEMod      - Turbulent Boundary Layer-Trailing Edge noise calculation
↪{0: none, 1:BPM, 2: TNO} (switch)
1             BLMod         - Calculation method for boundary layer properties,  {1:␣
↪BPM, 2: Pretabulated} (switch)
1             TripMod       - Boundary layer trip model {0:no trip, 1: heavy trip, 2:␣
↪light trip} (switch) [Only used if BLMod=1]
0             LamMod        - Laminar boundary layer noise model {0:none, 1: BPM} ␣
↪(switch)
0             TipMod        - Tip vortex noise model {0:none, 1: BPM}  (switch)
True          RoundedTip    - Logical indicating rounded tip (flag) [Only used if␣
↪TipMod=1]
1.0           Alprat        - Tip lift curve slope (Default = 1.0) [Only used if␣
↪TipMod=1]
0             BluntMod      - Trailing-edge-bluntness - Vortex-shedding model {0:none,␣
↪1: BPM}  (switch)
"AABlade1.dat"    AABlFile(1)    - Name of file containing distributed aerodynamic␣
↪properties for Blade #1 (-)
"AABlade1.dat"    AABlFile(2)    - Name of file containing distributed aerodynamic␣
↪properties for Blade #2 (-)
"AABlade1.dat"    AABlFile(3)    - Name of file containing distributed aerodynamic␣
↪properties for Blade #3 (-)
======  Observer Input ␣
↪==================================================================
"AA_ObserverLocations.dat"    ObserverLocations        - Name of file containing all␣
↪observer locations X Y Z (-)
======  Outputs ␣
↪================================================================================
False               AWeighting  - A-weighting Flag (flag)
3                  NrOutFile  - Number of Output files. 1 for Time Dependent Overall␣
↪SPL, 2 for both 1 and Frequency and Time Dependent SPL as well, or 3 for both 1 and␣
↪2 and Acoustics mechanism dependent, 4 for 1-3 and the overall sound pressure␣
↪levels per blade per node per observer
"IEA_LB_RWT-AeroAcoustics_"    AAOutFile   - No Extension needed the resulting file␣
↪will have .out Name of file containing
END of input file (the word "END" must appear in the first 3 columns of this last␣
↪OutList line)
```

(continues on next page)

(continued from previous page)

```
-----------------------------------------------------------------------------------------
↪-
```

Boundary Layer Inputs

When the flag **BLMod** is set equal to 2, pretabulated properties of the boundary layer must be provided and are used by the turbulent boundary layer – trailing-edge noise models. The file name is to be specified in the field BL_file among the inputs of the file with the airfoil polar coefficients. One airfoil file must be specified per aerodynamic station.

```
! ------------ AirfoilInfo v1.01.x Input File ----------------------------------
! AeroElasticSE FAST driver
!
!
! ------------------------------------------------------------------------------
DEFAULT                  InterpOrd   ! Interpolation order to use for quasi-steady␣
↪table lookup {1=linear; 3=cubic spline; "default"} [default=3]
1                        NonDimArea  ! The non-dimensional area of the airfoil (area/
↪chord^2) (set to 1.0 if unsure or unneeded)
@"AF20_Coords.txt"       NumCoords   ! The number of coordinates in the airfoil shape␣
↪file. Set to zero if coordinates not included.
AF20_BL.txt              BL_file     ! The file name including the boundary layer␣
↪characteristics of the profile. Ignored if the aeroacoustic module is not called.
1                        NumTabs     ! Number of airfoil tables in this file.  Each␣
↪table must have lines for Re and Ctrl.
! ------------------------------------------------------------------------------
! data for table 1
```

The file, in this example named **AF20_BL.txt**, contains 8 inputs, which are tabulated for a given number of Reynolds numbers, ReListBL, and a given number of angles of attack, aoaListBL. The inputs, which are defined nondimensionally and must be provided for the suction and pressure side of the airfoil above and below the trailing edge, are:

- **Ue_Vinf** – flow velocity at the top of the boundary layer
- **Dstar** – δ^*, boundary layer displacement thickness
- **Delta** – δ, nominal boundary layer thickness
- **Cf** – friction coefficient.

In the following example, the file was generated thanks to a Python script[4] that runs the boundary layer solver, XFoil. Notably, XFoil, by default, does not return δ, but the boundary layer momentum thickness, θ. δ can be reconstructed using the expression from [aa-DG87]:

$$\delta = \theta \bullet \left(3.15 + \frac{1.72}{H-1}\right) + \delta^* \tag{4.56}$$

where H is the kinematic shape factor, which is also among the standard outputs of XFoil. Because it is usually impossible to obtain these values for the whole ranges of Reynolds numbers and angles of attack, the code is set to adopt the last available values and print to screen a warning.

```
! Boundary layer characteristics at the trailing edge for the airfoil coordinates of /
↪Users/pbortolo/work/2_openfast/noise/verifyAA/OpenFAST_IEA_LB_RWT/Airfoils/AF20_
↪Coords.txt
! Legend: aoa - angle of attack (deg), Re - Reynolds number (-, millions), PS -␣
↪pressure side, SS - suction side,  Ue_Vinf - edge velocity (-), Dstar -␣
↪displacement thickness (-), Delta - nominal boundary layer thickness (-) Cf  -␣
↪friction coefficient (-)
```

(continues on next page)

[4] https://github.com/OpenFAST/python-toolbox

(continued from previous page)

```
4           ReListBL   -  Number of Reynolds numbers (it corresponds to the number of␣
↪tables)
30          aoaListBL  -  Number of angles of attack (it corresponds to the number of␣
↪rows in each table)
0.50          -  Re
aoa             Ue_Vinf_SS           Ue_Vinf_PS          Dstar_SS            Dstar_PS    ␣
↪        Delta_SS            Delta_PS            Cf_SS               Cf_PS
(deg)           (-)                  (-)                 (-)                 (-)         ␣
↪        (-)                 (-)                 (-)                 (-)
-5.00000        8.39390e-01         -8.37360e-01        7.43700e-03         1.07730e-02␣
↪        2.75094e-02         5.15849e-02         1.13200e-03         1.58200e-03
-3.96552        8.42050e-01         -8.40230e-01        8.26600e-03         9.29500e-03␣
↪        2.98650e-02         4.87153e-02         1.04400e-03         1.85700e-03
-2.93103        8.45320e-01         -8.43690e-01        9.08800e-03         8.10000e-03␣
↪        3.19790e-02         4.70045e-02         9.58000e-04         2.16500e-03
-1.89655        8.48230e-01         -8.46710e-01        9.97400e-03         7.33700e-03␣
↪        3.44024e-02         4.50456e-02         8.90000e-04         2.35800e-03
-0.86207        8.51550e-01         -8.50140e-01        1.09130e-02         6.54100e-03␣
↪        3.68822e-02         4.30884e-02         8.26000e-04         2.59900e-03
 0.17241        8.55000e-01         -8.53670e-01        1.18900e-02         5.92900e-03␣
↪        3.96199e-02         4.27416e-02         7.79000e-04         2.87100e-03
 1.20690        8.63820e-01         -1.04207e+00        1.22130e-02         9.89500e-03␣
↪        4.18890e-02         1.68156e-02         8.18000e-04        -1.77000e-04
 2.24138        8.61500e-01         -8.60210e-01        1.40420e-02         4.88700e-03␣
↪        4.51813e-02         3.93105e-02         6.78000e-04         3.28700e-03
 3.27586        8.64430e-01         -8.63080e-01        1.52900e-02         4.57300e-03␣
↪        4.85938e-02         3.82233e-02         6.39000e-04         3.44000e-03
 4.31034        8.67960e-01         -8.66600e-01        1.65660e-02         4.09100e-03␣
↪        5.17768e-02         3.63749e-02         5.96000e-04         3.69000e-03
 5.34483        8.72300e-01         -8.70850e-01        1.81000e-02         3.81700e-03␣
↪        5.43379e-02         3.52278e-02         5.09000e-04         3.86300e-03
 6.37931        8.77930e-01         -8.76410e-01        1.98500e-02         3.39700e-03␣
↪        5.69109e-02         3.31481e-02         4.18000e-04         4.13900e-03
 7.41379        8.86840e-01         -8.85140e-01        2.22250e-02         3.15000e-03␣
↪        5.81316e-02         3.19040e-02         2.64000e-04         4.36900e-03
 8.44828        9.00620e-01         -8.98660e-01        2.54290e-02         2.75900e-03␣
↪        5.91946e-02         2.95298e-02         1.01000e-04         4.76300e-03
 9.48276        9.20300e-01         -9.17700e-01        2.99830e-02         2.48300e-03␣
↪        6.07767e-02         2.75551e-02         5.00000e-06         5.16000e-03
10.51724        9.48080e-01         -9.44440e-01        3.80160e-02         2.13200e-03␣
↪        6.65531e-02         2.48447e-02        -1.60000e-05         5.76800e-03
11.55172        9.89560e-01         -9.84930e-01        5.83630e-02         1.85700e-03␣
↪        8.76076e-02         2.18890e-02        -1.50000e-05         6.49000e-03
12.58621        1.02883e+00         -1.02353e+00        8.80990e-02         1.66700e-03␣
↪        1.21588e-01         2.00072e-02        -1.30000e-05         7.20200e-03
13.62069        1.05789e+00         -1.05226e+00        1.18914e-01         1.51000e-03␣
↪        1.57264e-01         1.78004e-02        -1.10000e-05         7.74800e-03
14.65517        1.07975e+00         -1.07394e+00        1.48726e-01         1.41900e-03␣
↪        1.91423e-01         1.65710e-02        -1.00000e-05         8.15600e-03
15.68966        1.09657e+00         -1.09067e+00        1.76430e-01         1.34400e-03␣
↪        2.22657e-01         1.56180e-02        -9.00000e-06         8.50600e-03
16.72414        1.11040e+00         -1.10441e+00        2.02883e-01         1.26100e-03␣
↪        2.52158e-01         1.43276e-02        -9.00000e-06         8.80900e-03
17.75862        1.12290e+00         -1.11682e+00        2.29606e-01         1.20600e-03␣
↪        2.81695e-01         1.35432e-02        -8.00000e-06         9.07600e-03
18.79310        1.13461e+00         -1.12844e+00        2.55478e-01         1.15500e-03␣
↪        3.10143e-01         1.28744e-02        -8.00000e-06         9.34700e-03
```

(continues on next page)

(continued from previous page)

```
19.82759        1.14605e+00        -1.13974e+00        2.80923e-01        1.08200e-03␣
↪        3.37970e-01        1.16844e-02        -8.00000e-06        9.61200e-03
20.86207        1.15722e+00        -1.15073e+00        3.05117e-01        1.03800e-03␣
↪        3.64240e-01        1.10866e-02        -7.00000e-06        9.87000e-03
21.89655        1.16808e+00        -1.16138e+00        3.27770e-01        9.81000e-04␣
↪        3.88826e-01        1.02373e-02        -7.00000e-06        1.01370e-02
22.93103        1.17845e+00        -1.17148e+00        3.48909e-01        9.33000e-04␣
↪        4.11299e-01        9.52780e-03        -7.00000e-06        1.03870e-02
23.96552        1.18930e+00        -1.18205e+00        3.70277e-01        8.93000e-04␣
↪        4.34300e-01        9.01762e-03        -7.00000e-06        1.06550e-02
25.00000        1.19987e+00        -1.19227e+00        3.90503e-01        8.36000e-04␣
↪        4.55921e-01        8.12755e-03        -7.00000e-06        1.09080e-02
1.00       -  Re
aoa             Ue_Vinf_SS          Ue_Vinf_PS          Dstar_SS            Dstar_PS     ␣
↪        Delta_SS            Delta_PS            Cf_SS               Cf_PS
(deg)            (-)                 (-)                 (-)                 (-)         ␣
↪        (-)                 (-)                 (-)                 (-)
-5.00000        8.34300e-01        -8.32480e-01        6.49600e-03        7.74600e-03␣
↪        2.28566e-02        3.97467e-02        8.39000e-04        1.54900e-03
-3.96552        8.37330e-01        -8.35790e-01        7.10100e-03        6.55800e-03␣
↪        2.45059e-02        3.67266e-02        7.84000e-04        1.80000e-03
-2.93103        8.40670e-01        -8.39370e-01        7.75600e-03        5.65600e-03␣
↪        2.62162e-02        3.42658e-02        7.27000e-04        2.03700e-03
-1.89655        8.44170e-01        -8.43070e-01        8.45300e-03        4.96000e-03␣
↪        2.79616e-02        3.22259e-02        6.72000e-04        2.25700e-03
-0.86207        8.47840e-01        -8.46890e-01        9.21600e-03        4.45100e-03␣
↪        2.98142e-02        3.07238e-02        6.18000e-04        2.45400e-03
 0.17241        8.51730e-01        -8.50900e-01        1.00790e-02        3.95100e-03␣
↪        3.18738e-02        2.89503e-02        5.65000e-04        2.66300e-03
 1.20690        8.55470e-01        -8.54730e-01        1.09340e-02        3.54400e-03␣
↪        3.37289e-02        2.74209e-02        5.12000e-04        2.86100e-03
 2.24138        8.59040e-01        -8.58320e-01        1.18130e-02        3.25200e-03␣
↪        3.55603e-02        2.64490e-02        4.62000e-04        3.03800e-03
 3.27586        8.63480e-01        -8.62770e-01        1.29500e-02        2.91700e-03␣
↪        3.78947e-02        2.47691e-02        4.08000e-04        3.23200e-03
 4.31034        8.67590e-01        -8.66830e-01        1.40320e-02        2.69800e-03␣
↪        3.97441e-02        2.39342e-02        3.50000e-04        3.40400e-03
 5.34483        8.72380e-01        -8.71540e-01        1.53110e-02        2.43000e-03␣
↪        4.18407e-02        2.22446e-02        2.92000e-04        3.59200e-03
 6.37931        8.78360e-01        -8.77360e-01        1.68420e-02        2.23600e-03␣
↪        4.38267e-02        2.12352e-02        2.20000e-04        3.78300e-03
 7.41379        8.86030e-01        -8.84810e-01        1.87390e-02        2.00100e-03␣
↪        4.60113e-02        1.94428e-02        1.44000e-04        4.00100e-03
 8.44828        8.96310e-01        -8.94850e-01        2.13480e-02        1.83100e-03␣
↪        4.88127e-02        1.83696e-02        5.90000e-05        4.24200e-03
 9.48276        9.25990e-01        -9.23230e-01        2.81520e-02        1.56900e-03␣
↪        5.51012e-02        1.62260e-02        -1.00000e-06        4.73700e-03
10.51724        9.66170e-01        -9.62320e-01        4.28900e-02        1.36700e-03␣
↪        7.03103e-02        1.45187e-02        -9.00000e-06        5.34800e-03
11.55172        1.00255e+00        -9.97860e-01        6.33540e-02        1.21700e-03␣
↪        9.26255e-02        1.29836e-02        -7.00000e-06        5.90200e-03
12.58621        1.03100e+00        -1.02578e+00        8.62500e-02        1.10600e-03␣
↪        1.18923e-01        1.16999e-02        -6.00000e-06        6.34900e-03
13.62069        1.05406e+00        -1.04857e+00        1.10634e-01        1.04100e-03␣
↪        1.47132e-01        1.09721e-02        -6.00000e-06        6.70700e-03
14.65517        1.07334e+00        -1.06769e+00        1.35720e-01        9.66000e-04␣
↪        1.76016e-01        9.96935e-03        -5.00000e-06        7.01900e-03
```

(continues on next page)

(continued from previous page)

```
15.68966        1.08881e+00       -1.08308e+00       1.60129e-01       9.17000e-04␣
↪       2.03832e-01       9.33244e-03       -5.00000e-06       7.27400e-03
16.72414        1.10158e+00       -1.09579e+00       1.83765e-01       8.82000e-04␣
↪       2.30423e-01       8.89329e-03       -5.00000e-06       7.49000e-03
17.75862        1.11342e+00       -1.10758e+00       2.08205e-01       8.32000e-04␣
↪       2.57695e-01       8.20477e-03       -4.00000e-06       7.69800e-03
18.79310        1.12407e+00       -1.11817e+00       2.32504e-01       8.01000e-04␣
↪       2.84583e-01       7.81234e-03       -4.00000e-06       7.88600e-03
19.82759        1.13501e+00       -1.12904e+00       2.57953e-01       7.76000e-04␣
↪       3.12682e-01       7.52201e-03       -4.00000e-06       8.07500e-03
20.86207        1.14614e+00       -1.14008e+00       2.83630e-01       7.33000e-04␣
↪       3.41005e-01       6.90325e-03       -4.00000e-06       8.27100e-03
21.89655        1.15868e+00       -1.15248e+00       3.10888e-01       7.07000e-04␣
↪       3.71055e-01       6.60979e-03       -4.00000e-06       8.48600e-03
22.93103        1.17050e+00       -1.16410e+00       3.35623e-01       6.81000e-04␣
↪       3.98279e-01       6.28286e-03       -3.00000e-06       8.69100e-03
23.96552        1.18348e+00       -1.17683e+00       3.61314e-01       6.45000e-04␣
↪       4.26528e-01       5.81057e-03       -3.00000e-06       8.91700e-03
25.00000        1.19753e+00       -1.19058e+00       3.87323e-01       6.21000e-04␣
↪       4.54991e-01       5.52432e-03       -3.00000e-06       9.15800e-03
5.00       -  Re
aoa             Ue_Vinf_SS          Ue_Vinf_PS         Dstar_SS          Dstar_PS     ␣
↪       Delta_SS          Delta_PS          Cf_SS             Cf_PS
(deg)          (-)                 (-)               (-)               (-)           ␣
↪       (-)               (-)               (-)               (-)
-5.00000        8.23420e-01       -8.21880e-01       4.67200e-03       4.76700e-03␣
↪       1.77334e-02       2.96859e-02       6.92000e-04       1.41000e-03
-3.96552        8.25550e-01       -8.24400e-01       5.04400e-03       4.14000e-03␣
↪       1.88321e-02       2.75480e-02       6.57000e-04       1.55000e-03
-2.93103        8.27930e-01       -8.27220e-01       5.46200e-03       3.53900e-03␣
↪       2.00407e-02       2.52464e-02       6.21000e-04       1.70500e-03
-1.89655        8.30490e-01       -8.30120e-01       5.91700e-03       3.10400e-03␣
↪       2.13254e-02       2.34284e-02       5.86000e-04       1.84000e-03
-0.86207        8.33100e-01       -8.33000e-01       6.40000e-03       2.77600e-03␣
↪       2.26264e-02       2.19701e-02       5.50000e-04       1.95800e-03
 0.17241        8.35520e-01       -8.35690e-01       6.86100e-03       2.45300e-03␣
↪       2.37731e-02       2.03359e-02       5.15000e-04       2.08300e-03
 1.20690        8.38270e-01       -8.38660e-01       7.40600e-03       2.17500e-03␣
↪       2.51176e-02       1.87906e-02       4.79000e-04       2.20700e-03
 2.24138        8.41350e-01       -8.41880e-01       8.04900e-03       1.95800e-03␣
↪       2.66635e-02       1.75032e-02       4.40000e-04       2.31900e-03
 3.27586        8.43950e-01       -8.44520e-01       8.65200e-03       1.80300e-03␣
↪       2.79650e-02       1.65339e-02       4.03000e-04       2.40900e-03
 4.31034        8.48180e-01       -8.48810e-01       9.58300e-03       1.61000e-03␣
↪       3.00737e-02       1.51804e-02       3.59000e-04       2.53200e-03
 5.34483        8.53570e-01       -8.54090e-01       1.08300e-02       1.48600e-03␣
↪       3.27612e-02       1.43249e-02       3.08000e-04       2.63700e-03
 6.37931        8.72880e-01       -8.73060e-01       1.51570e-02       1.28200e-03␣
↪       4.16833e-02       1.28096e-02       1.92000e-04       2.88700e-03
 7.41379        8.92130e-01       -8.91760e-01       1.98220e-02       1.14700e-03␣
↪       4.87740e-02       1.17767e-02       8.30000e-05       3.11600e-03
 8.44828        9.17360e-01       -9.16020e-01       2.50640e-02       9.92000e-04␣
↪       5.31945e-02       1.04181e-02       2.00000e-06       3.41900e-03
 9.48276        9.42910e-01       -9.40410e-01       3.17040e-02       8.85000e-04␣
↪       5.85499e-02       9.42477e-03       -1.00000e-06       3.70700e-03
10.51724        9.64800e-01       -9.61630e-01       4.02300e-02       7.96000e-04␣
↪       6.64893e-02       8.47323e-03       -2.00000e-06       3.96100e-03
```

(continues on next page)

(continued from previous page)

```
11.55172        9.86420e-01       -9.82570e-01       5.11880e-02        7.23000e-04␣
↪       7.76623e-02       7.65452e-03       -2.00000e-06       4.20700e-03
12.58621        1.00657e+00       -1.00210e+00       6.43270e-02        6.71000e-04␣
↪       9.20001e-02       7.06023e-03       -2.00000e-06       4.43100e-03
13.62069        1.02475e+00       -1.01984e+00       7.93340e-02        6.16000e-04␣
↪       1.09051e-01       6.35528e-03       -1.00000e-06       4.64000e-03
14.65517        1.04370e+00       -1.03850e+00       9.84840e-02        5.79000e-04␣
↪       1.31195e-01       5.91001e-03       -1.00000e-06       4.84500e-03
15.68966        1.06004e+00       -1.05467e+00       1.18503e-01        5.43000e-04␣
↪       1.54410e-01       5.44594e-03       -1.00000e-06       5.02500e-03
16.72414        1.07448e+00       -1.06905e+00       1.39604e-01        5.14000e-04␣
↪       1.78759e-01       5.05912e-03       -1.00000e-06       5.18500e-03
17.75862        1.08720e+00       -1.08175e+00       1.61656e-01        4.93000e-04␣
↪       2.03997e-01       4.79726e-03       -1.00000e-06       5.32500e-03
18.79310        1.09867e+00       -1.09324e+00       1.84226e-01        4.68000e-04␣
↪       2.29525e-01       4.45243e-03       -1.00000e-06       5.45500e-03
19.82759        1.10970e+00       -1.10430e+00       2.08500e-01        4.51000e-04␣
↪       2.56774e-01       4.24858e-03       -1.00000e-06       5.57800e-03
20.86207        1.11936e+00       -1.11397e+00       2.32097e-01        4.34000e-04␣
↪       2.83065e-01       4.03443e-03       -1.00000e-06       5.69000e-03
21.89655        1.12815e+00       -1.12274e+00       2.54679e-01        4.14000e-04␣
↪       3.07965e-01       3.77358e-03       -1.00000e-06       5.79400e-03
22.93103        1.13774e+00       -1.13227e+00       2.78750e-01        4.00000e-04␣
↪       3.34530e-01       3.60784e-03       -1.00000e-06       5.90600e-03
23.96552        1.14721e+00       -1.14164e+00       3.02299e-01        3.84000e-04␣
↪       3.60352e-01       3.41109e-03       -1.00000e-06       6.01800e-03
25.00000        1.15816e+00       -1.15244e+00       3.27151e-01        3.68000e-04␣
↪       3.87710e-01       3.21949e-03       -1.00000e-06       6.14600e-03
10.00      -  Re
aoa             Ue_Vinf_SS         Ue_Vinf_PS         Dstar_SS           Dstar_PS     ␣
↪       Delta_SS          Delta_PS          Cf_SS              Cf_PS
(deg)           (-)                (-)                (-)                (-)          ␣
↪       (-)               (-)               (-)                (-)
-5.00000        8.19760e-01       -8.18060e-01       4.17800e-03        4.54900e-03␣
↪       1.65706e-02       2.88150e-02       6.56000e-04        1.23100e-03
-3.96552        8.21540e-01       -8.20450e-01       4.52500e-03        3.74000e-03␣
↪       1.76308e-02       2.59028e-02       6.23000e-04        1.39100e-03
-2.93103        8.23580e-01       -8.22970e-01       4.89400e-03        3.21700e-03␣
↪       1.87333e-02       2.38284e-02       5.91000e-04        1.51700e-03
-1.89655        8.25560e-01       -8.25320e-01       5.25400e-03        2.85300e-03␣
↪       1.97567e-02       2.22669e-02       5.60000e-04        1.62100e-03
-0.86207        8.27870e-01       -8.28060e-01       5.67900e-03        2.46600e-03␣
↪       2.09522e-02       2.03860e-02       5.28000e-04        1.74400e-03
 0.17241        8.30330e-01       -8.30840e-01       6.14400e-03        2.18100e-03␣
↪       2.22219e-02       1.88758e-02       4.96000e-04        1.84900e-03
 1.20690        8.32880e-01       -8.33650e-01       6.64800e-03        1.94100e-03␣
↪       2.35312e-02       1.74735e-02       4.63000e-04        1.94900e-03
 2.24138        8.35130e-01       -8.36090e-01       7.13000e-03        1.75100e-03␣
↪       2.46910e-02       1.62700e-02       4.31000e-04        2.03800e-03
 3.27586        8.39970e-01       -8.41060e-01       8.09900e-03        1.56800e-03␣
↪       2.72181e-02       1.50508e-02       3.88000e-04        2.14300e-03
 4.31034        8.50470e-01       -8.51560e-01       1.01990e-02        1.37300e-03␣
↪       3.25448e-02       1.36378e-02       3.18000e-04        2.29400e-03
 5.34483        8.64450e-01       -8.65280e-01       1.32660e-02        1.23700e-03␣
↪       3.92329e-02       1.26866e-02       2.31000e-04        2.45100e-03
 6.37931        8.78610e-01       -8.79110e-01       1.65810e-02        1.08900e-03␣
↪       4.49765e-02       1.14397e-02       1.47000e-04        2.62200e-03
```

(continues on next page)

(continued from previous page)

```
 7.41379        8.91030e-01        -8.91080e-01        1.96290e-02        9.93000e-04        4.89936e-02        1.06282e-02        7.60000e-05        2.76500e-03
 8.44828        9.08900e-01        -9.08620e-01        2.35230e-02        8.71000e-04        5.22284e-02        9.45732e-03        2.00000e-06        2.96800e-03
 9.48276        9.32700e-01        -9.30700e-01        2.84210e-02        7.79000e-04        5.52443e-02        8.61055e-03       -0.00000e+00        3.20000e-03
10.51724        9.51380e-01        -9.48770e-01        3.46600e-02        6.96000e-04        6.05165e-02        7.64709e-03       -1.00000e-06        3.39700e-03
11.55172        9.71740e-01        -9.68450e-01        4.35850e-02        6.37000e-04        6.90670e-02        6.98615e-03       -1.00000e-06        3.59500e-03
12.58621        9.91260e-01        -9.87290e-01        5.44080e-02        5.84000e-04        8.03205e-02        6.33577e-03       -1.00000e-06        3.78700e-03
13.62069        1.00996e+00        -1.00542e+00        6.74960e-02        5.36000e-04        9.47613e-02        5.73102e-03       -1.00000e-06        3.97000e-03
14.65517        1.02771e+00        -1.02275e+00        8.31660e-02        5.06000e-04        1.12645e-01        5.35979e-03       -1.00000e-06        4.13700e-03
15.68966        1.04427e+00        -1.03905e+00        1.00836e-01        4.71000e-04        1.33082e-01        4.88548e-03       -1.00000e-06        4.29600e-03
16.72414        1.06019e+00        -1.05485e+00        1.21136e-01        4.45000e-04        1.56673e-01        4.55077e-03       -1.00000e-06        4.44600e-03
17.75862        1.07407e+00        -1.06868e+00        1.42220e-01        4.22000e-04        1.81035e-01        4.24533e-03       -1.00000e-06        4.57900e-03
18.79310        1.08623e+00        -1.08087e+00        1.64037e-01        4.01000e-04        2.06006e-01        3.94306e-03       -0.00000e+00        4.69600e-03
19.82759        1.09748e+00        -1.09215e+00        1.87080e-01        3.86000e-04        2.32142e-01        3.76503e-03       -0.00000e+00        4.80500e-03
20.86207        1.10794e+00        -1.10267e+00        2.10804e-01        3.67000e-04        2.58816e-01        3.50553e-03       -0.00000e+00        4.90800e-03
21.89655        1.11776e+00        -1.11253e+00        2.35256e-01        3.54000e-04        2.86067e-01        3.34709e-03       -0.00000e+00        5.00500e-03
22.93103        1.12664e+00        -1.12138e+00        2.58366e-01        3.43000e-04        3.11568e-01        3.20986e-03       -0.00000e+00        5.09600e-03
23.96552        1.13635e+00        -1.13106e+00        2.83067e-01        3.28000e-04        3.38816e-01        3.02058e-03       -0.00000e+00        5.19400e-03
25.00000        1.14573e+00        -1.14034e+00        3.06604e-01        3.16000e-04        3.64612e-01        2.86692e-03       -0.00000e+00        5.29100e-03
```

Observer Positions

The number and position of observers is set in the file ObserverLocations, which is explained in Section 4.4.7. The positions must be specified in the OpenFAST global inertial frame coordinate system, which is located at the tower base and has the x-axis pointing downwind, the y-axis pointing laterally, and the z-axis pointing vertically upward. A scheme of the coordinate system for the observers is shown in Fig. 4.20.

The International Energy Agency Wind Task 37 land-based reference wind turbine, which is shown in Table 4.4, has a hub height of 110 meters and a rotor radius of 65 meters, and has the International Electrotechnical Commission 61400-11 standards compliant observer located at:

x = 175 [m]

y = 0 [m]

z = 0 [m].

An example of a file listing four observers located at a 2-meter height is shown here:

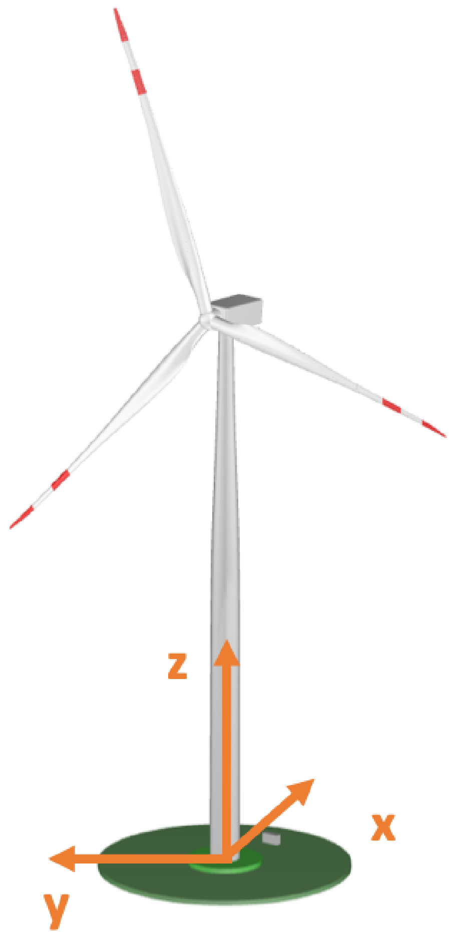


Fig. 4.20: Reference system for the observers

```
4 NrObsLoc - Total Number of observer locations
X Observer location in tower-base coordinate X horizontal (m), Y Observer location in␣
↪tower-base coordinate Y Lateral (m), Z Observer location in tower-base coordinate Z␣
↪Vertical (m)
-200  -200  2
-200  +200  2
+200  -200  2
+200  +200  2
```

Turbulence Grid

When the flag **TICalcMeth** is set equal to 1, the grid of incident turbulent intensity I_1 must be defined by the user. This is done by creating a file called **TIGrid_In.txt**, which mimics a TurbSim output file and contains a grid of turbulence intensity, which is defined as a fraction value. The file defines a grid centered at hub height and oriented with the OpenFAST global inertial frame coordinate system; see Fig. 4.20. A user-defined number of lateral and vertical points equally spaced by a user-defined number of meters must be specified. An example file for a 160 (lateral) by 180 (vertical) meters grid looks like the following:

```
Total Grid points In Y (lateral), Starts from - radius goes to + radius+
4
Total Grid points In Z (vertical), Starts from bottom tip (hub-radius)
3
Grid spacing In Y (lateral)
40
Grid spacing In Z (vertical)
60
0.1200 0.1200 0.1200 0.1200
0.1100 0.1100 0.1100 0.1100
0.1000 0.1000 0.1000 0.1000
```

Trailing-Edge Geometry

When the flag **BluntMod** is set to 1, the detailed geometry of the trailing edge must be defined along the span. Two inputs must be provided, namely the angle, Ψ, between the suction and pressure sides of the profile, right before the trailing-edge point, and the height, h, of the trailing edge. Ψ must be defined in degrees, while h is in meters. Note that the BPM trailing-edge bluntness model is very sensitive to these two parameters, which, however, are often not easy to determine for real blades. Fig. 4.21 shows the two inputs.

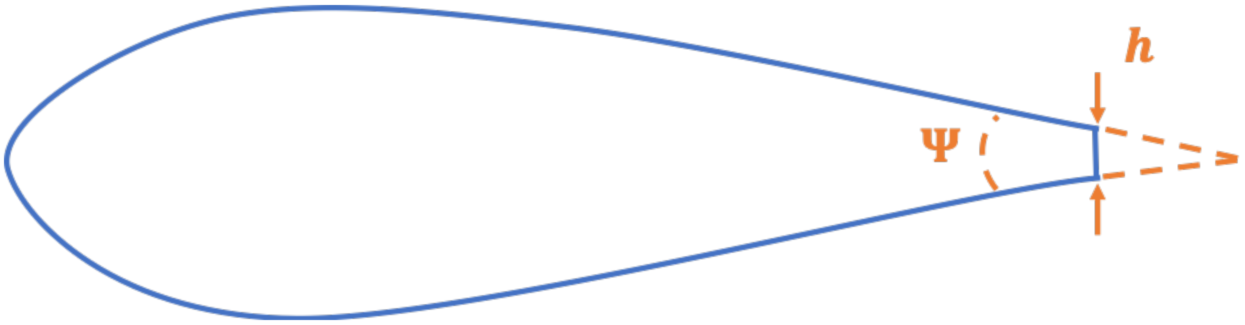


Fig. 4.21: Geometric parameters **Ψ** and **h** of the trailing-edge bluntness

The two distributions must be defined with the same spanwise resolution of the AeroDyn15 blade file, such as:

```
Example aerodynamic blade input properties
====== Blade Properties =========================================
30    NumBlNds    - Number of blade nodes used in the analysis (-)
TEAngle               TEThick
(deg)                 (m)
10.000000E+00         0.03000E+00
10.000000E+00         0.02900E+00
10.000000E+00         0.02800E+00
10.000000E+00         0.02700E+00
10.000000E+00         0.02600E+00
10.000000E+00         0.02500E+00
10.000000E+00         0.02400E+00
10.000000E+00         0.02300E+00
10.000000E+00         0.02200E+00
10.000000E+00         0.02100E+00
10.000000E+00         0.02000E+00
10.000000E+00         0.01900E+00
10.000000E+00         0.01800E+00
10.000000E+00         0.01700E+00
10.000000E+00         0.01600E+00
10.000000E+00         0.01500E+00
10.000000E+00         0.01400E+00
10.000000E+00         0.01300E+00
10.000000E+00         0.01200E+00
10.000000E+00         0.01100E+00
10.000000E+00         0.01000E+00
10.000000E+00         0.01000E+00
10.000000E+00         0.01000E+00
10.000000E+00         0.01000E+00
10.000000E+00         0.01000E+00
10.000000E+00         0.01000E+00
10.000000E+00         0.01000E+00
10.000000E+00         0.01000E+00
10.000000E+00         0.01000E+00
10.000000E+00         0.01000E+00
```

4.5 BeamDyn User Guide and Theory Manual

4.5.1 Introduction

BeamDyn is a time-domain structural-dynamics module for slender structures created by the National Renewable Energy Laboratory (NREL) through support from the U.S. Department of Energy Wind and Water Power Program and the NREL Laboratory Directed Research and Development (LDRD) program through the grant "High-Fidelity Computational Modeling of Wind-Turbine Structural Dynamics", see References [WS13][WYS13][WSJJ14][WJSJ15]. The module has been coupled into the FAST aero-hydro-servo-elastic wind turbine multi-physics engineering tool where it used to model blade structural dynamics. The BeamDyn module follows the requirements of the FAST modularization framework, see References [Jon13]; [GSJ13][SJJ14][JJ13], couples to FAST version 8, and provides new capabilities for modeling initially curved and twisted composite wind turbine blades undergoing large deformation. BeamDyn can also be driven as a stand-alone code to compute the static and dynamic responses of slender structures (blades or otherwise) under prescribed boundary and applied loading conditions uncoupled from FAST.

The model underlying BeamDyn is the geometrically exact beam theory (GEBT) [Hod06]. GEBT supports full geometric nonlinearity and large deflection, with bending, torsion, shear, and extensional degree-of-freedom (DOFs); anisotropic composite material couplings (using full 6×6 mass and stiffness matrices, including bend-twist coupling); and a reference axis that permits blades that are not straight (supporting built-in curve, sweep, and sectional offsets). The GEBT beam equations are discretized in space with Legendre spectral finite elements (LSFEs). LFSEs are *p*-type elements that combine the accuracy of global spectral methods with the geometric modeling flexibility of the *h*-type finite elements (FEs) [Pat84]. For smooth solutions, LSFEs have exponential convergence rates compared to low-order elements that have algebraic convergence [SG03][WS13] . Two spatial numerical integration schemes are implemented for the finite element inner products: reduced Gauss quadrature and trapezoidal-rule integration. Trapezoidal-rule integration is appropriate when a large number of sectional properties are specified along the beam axis, for example, in a long wind turbine blade with material properties that vary dramatically over the length. Time integration of the BeamDyn equations of motion is achieved through the implicit generalized- α solver, with user-specified numerical damping. The combined GEBT-LSFE approach permits users to model a long, flexible, composite wind turbine blade with a single high-order element. Given the theoretical foundation and powerful numerical tools introduced above, BeamDyn can solve the complicated nonlinear composite beam problem in an efficient manner. For example, it was recently shown that a grid-independent dynamic solution of a 50-m composite wind turbine blade and with dozens of cross-section stations could be achieved with a single 7^{th}-order LSFE [WSJJ16].

When coupled with FAST, loads and responses are transferred between BeamDyn, ElastoDyn, ServoDyn, and AeroDyn via the FAST driver program (glue code) to enable aero-elasto-servo interaction at each coupling time step. There is a separate instance of BeamDyn for each blade. At the root node, the inputs to BeamDyn are the six displacements (three translations and three rotations), six velocities, and six accelerations; the root node outputs from BeamDyn are the six reaction loads (three translational forces and three moments). BeamDyn also outputs the blade displacements, velocities, and accelerations along the beam length, which are used by AeroDyn to calculate the local aerodynamic loads (distributed along the length) that are used as inputs for BeamDyn. In addition, BeamDyn can calculate member internal reaction loads, as requested by the user. Please refers to Figure [fig:FlowChart] for the coupled interactions between BeamDyn and other modules in FAST. When coupled to FAST, BeamDyn replaces the more simplified blade structural model of ElastoDyn that is still available as an option, but is only applicable to straight isotropic blades dominated by bending. When uncoupled from FAST, the root motion (boundary condition) and applied loads are specified via a stand-alone BeamDyn driver code.

The BeamDyn input file defines the blade geometry; cross-sectional material mass, stiffness, and damping properties; FE resolution; and other simulation- and output-control parameters. The blade geometry is defined through a curvilinear blade reference axis by a series of key points in three-dimensional (3D) space along with the initial twist angles at these points. Each *member* contains at least three key points for the cubic spline fit implemented in BeamDyn; each member is discretized with a single LSFE with a parameter defining the order of the element. Note that the number of key points defining the member and the order (N) of the LSFE are independent. LSFE nodes, which are located at the $N + 1$ Gauss-Legendre-Lobatto points, are not evenly spaced along the element; node locations are generated by the module based on the mesh information. Blade properties are specified in a non-dimensional coordinate ranging from 0.0 to 1.0 along the blade reference axis and are linearly interpolated between two stations if needed by the spatial integration method. The BeamDyn applied loads can be either distributed loads specified at quadrature points, concentrated loads specified at FE nodes, or a combination of the two. When BeamDyn is coupled to FAST, the blade analysis node discretization may be independent between BeamDyn and AeroDyn.

This document is organized as follows. Section *Running BeamDyn* details how to obtain the BeamDyn and FAST software archives and run either the stand-alone version of BeamDyn or BeamDyn coupled to FAST. Section *Input Files* describes the BeamDyn input files. Section *Output Files* discusses the output files generated by BeamDyn. Section *BeamDyn Theory* summarizes the BeamDyn theory. Section *Future Work* outlines potential future work. Example input files are shown in Appendix Section 4.5.7. A summary of available output channels is found in Appendix *BeamDyn List of Output Channels*.

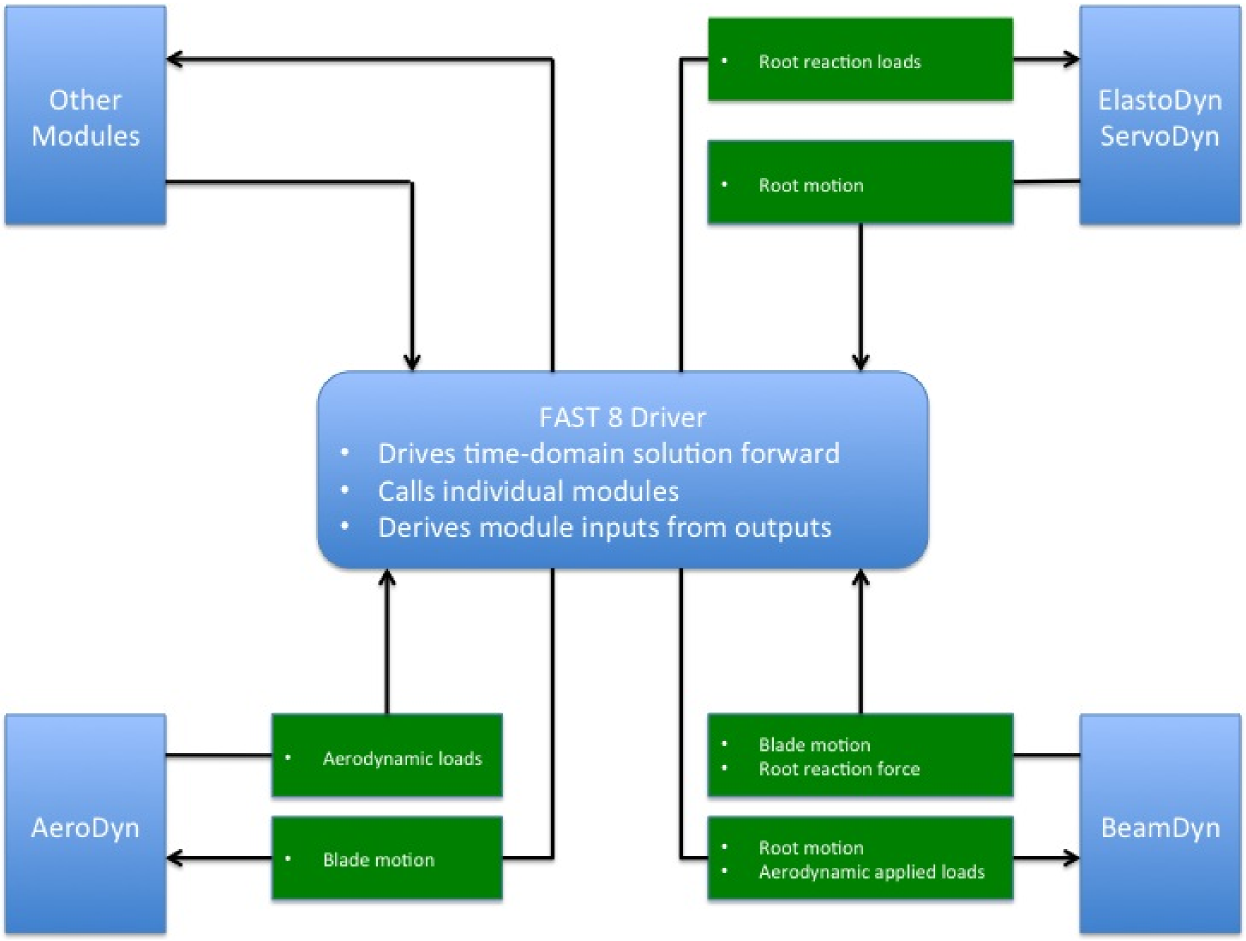


Fig. 4.22: Coupled interaction between BeamDyn and FAST

4.5.2 Running BeamDyn

This section discusses how to obtain and execute BeamDyn from a personal computer. Both the stand-alone version and the FAST-coupled version of the software are considered.

Downloading the BeamDyn Software

There are two forms of the BeamDyn software to choose from: stand-alone and coupled to the FAST simulator. Although the user may not necessarily need both forms, he/she would likely need to be familiar with and run the stand-alone model if building a model of the blade from scratch. The stand-alone version is also helpful for model troubleshooting, even if the goal is to conduct aero-hydro-servo-elastic simulations of onshore/offshore wind turbines within FAST.

Stand-Alone BeamDyn Archive

Users can download the stand-alone BeamDyn archive from our Web server at https://nwtc.nrel.gov/BeamDyn. The file has a name similar to `BD_v1.00.00a.exe`, but may have a different version number. The user can then download the self-extracting archive (*.exe*) to expand the archive into a folder he/she specifies.

The archive contains the `bin`, `CertTest`, `Compiling`, `Docs`, and `Source` folders. The `bin` folder includes the main executable file, `BeamDyn_Driver.exe`, which is used to execute the stand-alone BeamDyn program. The `CertTest` folder contains a collection of sample BeamDyn input files and driver input files that can be used as templates for the user's own models. This document may be found in the `Docs` folder. The `Compiling` folder contains files for compiling the stand-alone `BeamDyn_v1.00.00.exe` file with either Visual Studio or gFortran. The Fortran source code is located in the `Source` folder.

FAST Archive

Download the FAST archive, which includes BeamDyn, from our Web server at https://nwtc.nrel.gov/FAST8. The file has a name similar to `FAST_v8.12.00.exe`, but may have a different version number. Run the downloaded self-extracting archive (`.exe`) to expand the archive into a user-specified folder. The FAST executable file is located in the archive's `bin` folder. An example model using the NREL 5-MW reference turbine is located in the `CertTest` folder.

Running BeamDyn

Running the Stand-Alone BeamDyn Program

The stand-alone BeamDyn program, `BeamDyn_Driver.exe`, simulates static and dynamic responses of the user's input model, without coupling to FAST. Unlike the coupled version, the stand-alone software requires the use of a driver file in addition to the primary and blade BeamDyn input files. This driver file specifies inputs normally provided to BeamDyn by FAST, including motions of the blade root and externally applied loads. Both the BeamDyn summary file and the results output file are available when using the stand-alone BeamDyn (see Section *Output Files* for more information regarding the BeamDyn output files).

Run the stand-alone BeamDyn software from a DOS command prompt by typing, for example:

```
>BeamDyn_Driver.exe Dvr_5MW_Dynamic.inp
```

where, `Dvr_5MW_Dynamic.inp` is the name of the BeamDyn driver input file, as described in Section *BeamDyn Driver Input File*.

Running BeamDyn Coupled to FAST

Run the coupled FAST software from a DOS command prompt by typing, for example:

```
>FAST_Win32.exe Test26.fst
```

where `Test26.fst` is the name of the primary FAST input file. This input file has a feature switch to enable or disable the BeamDyn capabilities within FAST, and a corresponding reference to the BeamDyn input file. See the documentation supplied with FAST for further information.

4.5.3 Input Files

Users specify the blade model parameters; including its geometry, cross-sectional properties, and FE and output control parameters; via a primary BeamDyn input file and a blade property input file. When used in stand-alone mode, an additional driver input file is required. This driver file specifies inputs normally provided to BeamDyn by FAST, including simulation range, root motions, and externally applied loads.

No lines should be added or removed from the input files, except in tables where the number of rows is specified.

Units

BeamDyn uses the SI system (kg, m, s, N). Angles are assumed to be in radians unless otherwise specified.

BeamDyn Driver Input File

The driver input file is needed only for the stand-alone version of BeamDyn. It contains inputs that are normally set by FAST and that are necessary to control the simulation for uncoupled models.

The driver input file begins with two lines of header information, which is for the user but is not used by the software. If BeamDyn is run in the stand-alone mode, the results output file will be prefixed with the same name of this driver input file.

A sample BeamDyn driver input file is given in Section 4.5.7.

Simulation Control Parameters

`DynamicSolve` is a logical variable that specifies if BeamDyn should use dynamic analysis (`DynamicSolve = true`) or static analysis (`DynamicSolve = false`). `t_initial` and `t_final` specify the starting time of the simulation and ending time of the simulation, respectively. `dt` specifies the time step size.

Gravity Parameters

`Gx` , `Gy` , and `Gz` specify the components of gravity vector along X, Y, and Z directions in the global coordinate system, respectively. In FAST, this is normally 0, 0, and -9.80665.

Inertial Frame Parameters

This section defines the relation between two inertial frames, the global coordinate system and initial blade reference coordinate system. `GlbPos(1)`, `GlbPos(2)`, and `GlbPos(3)` specify three components of the initial global position vector along X, Y, and Z directions resolved in the global coordinate system, see Figure Fig. 4.23. And the following 3×3 direction cosine matrix (`GlbDCM`) relates the rotations from the global coordinate system to the initial blade reference coordinate system.

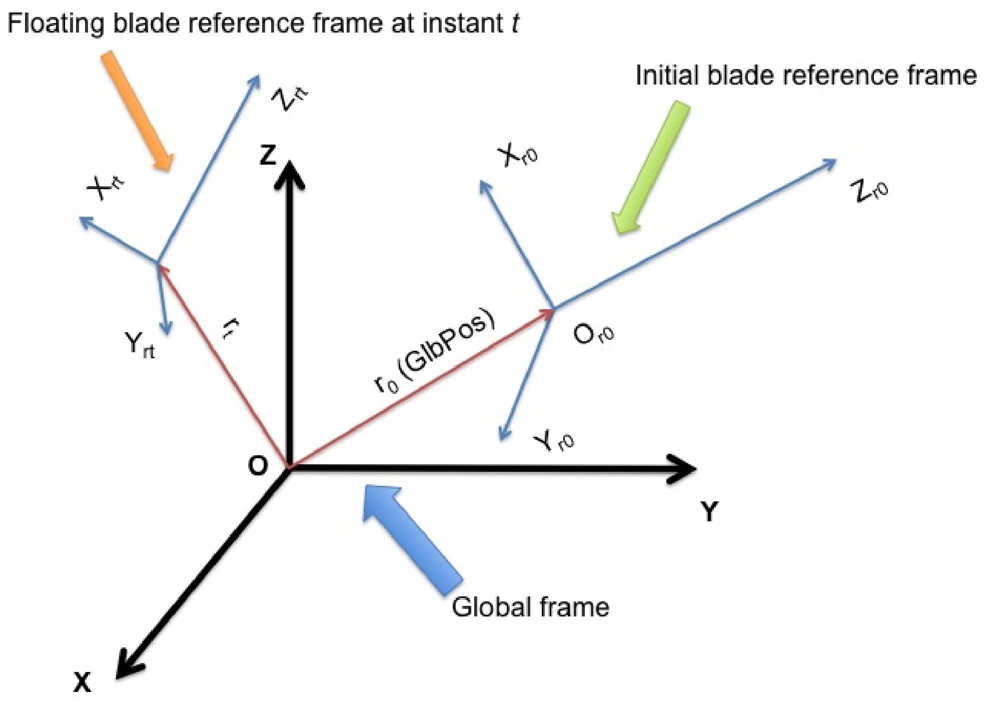


Fig. 4.23: Global and blade coordinate systems in BeamDyn.

Blade Floating Reference Frame Parameters

This section specifies the parameters that define the blade floating reference frame, which is a body-attached floating frame; the blade root is cantilevered at the origin of this frame. Based on the driver input file, the floating blade reference fame is assumed to be in a constant rigid-body rotation mode about the origin of the global coordinate system, that is,

$$v_{rt} = \omega_r \times r_t \tag{4.57}$$

where v_{rt} is the root (origin of the floating blade reference frame) translational velocity vector; ω_r is the constant root (origin of the floating blade reference frame) angular velocity vector; and r_t is the global position vector introduced in the previous section at instant t, see Fig. 4.23. The floating blade reference frame coincides with the initial floating blade reference frame at the beginning $t = 0$. `RootVel(4)`, `RootVel(5)`, and `RootVel(6)` specify the three components of the constant root angular velocity vector about X, Y, and Z axises in global coordinate system, respectively. `RootVel(1)`, `RootVel(2)`, and `RootVel(3)`, which are the three components of the root translational velocity vector along X, Y, and Z directions in global coordinate system, respectively, are calculated based on Eq. (4.57).

BeamDyn can handle more complicated root motions by changing, for example, the `BD_InputSolve` subroutine in the `Driver_Beam.f90` (requiring a recompile of stand-alone BeamDyn).

The blade is initialized in the rigid-body motion mode, i.e., based on the root velocity information defined in this section and the position information defined in the previous section, the motion of other points along the blade are initialized as

$$\begin{aligned} a_0 &= \omega_r \times (\omega_r \times (r_0 + P)) \\ v_0 &= v_{r0} + \omega_r \times P \\ \omega_0 &= \omega_r \end{aligned} \tag{4.58}$$

where a_0 is the initial translational acceleration vector along the blade; v_0 and ω_0 the initial translational and angular velocity vectors along the blade, respectively; and P is the position vector along the blade relative to the root. Note that these equations are actually implemented with a call to the NWTC Library's mesh mapping routines.

Applied Load

This section defines the applied loads, including distributed, point (lumped), and tip-concentrated loads, for the stand-alone analysis.

The first six entries `DistrLoad(i)`, $i \in [1, 6]$, specify three components of uniformly distributed force vector and three components of uniformly distributed moment vector in the global coordinate systems, respectively.

The following six entries `TipLoad(i)`, $i \in [1, 6]$, specify three components of concentrated tip force vector and three components of concentrated tip moment vector in the global coordinate system, respectively.

`NumPointLoads` defines how many point loads along the blade will be applied. The table following this input contains two header lines with seven columns and `NumPointLoads` rows. The first column is the non-dimensional distance along the local blade reference axis, ranging from $[0.0, 1.0]$. The next three columns, `Fx`, `Fy`, and `Fz` specify three components of point-force vector. The remaining three columns, `Mx`, `My`, and `Mz` specify three components of a moment vector.

The distributed load defined in this section is assumed to be uniform along the blade and constant throughout the simulation. The tip load is a constant concentrated load applied at the tip of a blade.

It is noted that all the loads defined in this section are dead loads, i.e., they are not rotating with the blade following the rigid-body rotation defined in the previous section.

BeamDyn is capable of handling more complex loading cases, e.g., time-dependent loads, through customization of the source code (requiring a recompile of stand-alone BeamDyn). The user can define such loads in the `BD_InputSolve` subroutine in the `Driver_Beam.f90` file, which is called every time step. The following section can be modified to define the concentrated load at each FE node:

```
u%PointLoad%Force(1:3,u%PointLoad%NNodes)  = u%PointLoad%Force(1:3,u%PointLoad
↪%NNodes)  + DvrData%TipLoad(1:3)
u%PointLoad%Moment(1:3,u%PointLoad%NNodes) = u%PointLoad%Moment(1:3,u%PointLoad
↪%NNodes) + DvrData%TipLoad(4:6)
```

where the first index in each array ranges from 1 to 3 for load vector components along three global directions and the second index of each array ranges from 1 to `u%PointLoad%NNodes`, where the latter is the total number of FE nodes. Note that `u%PointLoad%Force(1:3,:)` and `u%PointLoad%Moment(1:3,:)` have been populated with the point-load loads read from the BeamDyn driver input file using the call to `Transfer_Point_to_Point` earlier in the subroutine.

For example, a time-dependent sinusoidal force acting along the X direction applied at the 2^{nd} FE node can be defined as

```
u%PointLoad%Force(:,:) = 0.0D0
u%PointLoad%Force(1,2)  = 1.0D+03*SIN((2.0*pi)*t/6.0 )
u%PointLoad%Moment(:,:) = 0.0D0
```

with `1.0D+03` being the amplitude and `6.0` being the period. Note that this particular implementation overrides the tip-load and point-loads defined in the driver input file.

Similar to the concentrated load, the distributed loads can be defined in the same subroutine

```
DO i=1,u%DistrLoad%NNodes
   u%DistrLoad%Force(1:3,i) = DvrData%DistrLoad(1:3)
   u%DistrLoad%Moment(1:3,i)= DvrData%DistrLoad(4:6)
ENDDO
```

where `u%DistrLoad%NNodes` is the number of nodes input to BeamDyn (on the quadrature points), and `DvrData%DistrLoad(:)` is the constant uniformly distributed load BeamDyn reads from the driver input file. The user can modify `DvrData%DistrLoad(:)` to define the loads based on need.

We note that the distributed loads are defined at the quadrature points for numerical integrations. For example, if Gauss quadrature is chosen, then the distributed loads are defined at Gauss points plus the two end points of the beam (root and tip). For trapezoidal quadrature, `p%ngp` stores the number of trapezoidal quadrature points.

Primary Input File

`InputFile` is the file name of the primary BeamDyn input file. This name should be in quotations and can contain an absolute path or a relative path.

BeamDyn Primary Input File

The BeamDyn primary input file defines the blade geometry, LSFE-discretization and simulation options, output channels, and name of the blade input file. The geometry of the blade is defined by key-point coordinates and initial twist angles (in units of degree) in the blade local coordinate system (IEC standard blade system where Z_r is along blade axis from root to tip, X_r directs normally toward the suction side, and Y_r directs normally toward the trailing edge).

The file is organized into several functional sections. Each section corresponds to an aspect of the BeamDyn model.

A sample BeamDyn primary input file is given in Section 4.5.7.

The primary input file begins with two lines of header information, which are for the user but are not used by the software.

Simulation Controls

The user can set the `Echo` flag to `TRUE` to have BeamDyn echo the contents of the BeamDyn input file (useful for debugging errors in the input file).

The `QuasiStaticInit` flag indicates if BeamDyn should perform a quasi-static solution at initialization to better initialize its states. In general, this should be set to true for better numerical performance (it reduces startup transients).

`rhoinf` specifies the numerical damping parameter (spectral radius of the amplification matrix) in the range of $[0.0, 1.0]$ used in the generalized-α time integrator implemented in BeamDyn for dynamic analysis. For `rhoinf = 1.0`, no numerical damping is introduced and the generalized-α scheme is identical to the Newmark scheme; for `rhoinf = 0.0`, maximum numerical damping is introduced. Numerical damping may help produce numerically stable solutions.

`Quadrature` specifies the spatial numerical integration scheme. There are two options: 1) Gauss quadrature; and 2) Trapezoidal quadrature. We note that in the current version, Gauss quadrature is implemented in reduced form to improve efficiency and avoid shear locking. In the trapezoidal quadrature, only one member (FE element) can be defined in the following GEOMETRY section of the primary input file. Trapezoidal quadrature is appropriate when the number of "blade input stations" (described below) is significantly greater than the order of the LSFE.

`Refine` specifies a refinement parameter used in trapezoidal quadrature. An integer value greater than unity will split the space between two input stations into "Refine factor" of segments. The keyword "DEFAULT" may be used to set it to 1, i.e., no refinement is needed. This entry is not used in Gauss quadrature.

`N_Fact` specifies a parameter used in the modified Newton-Raphson scheme. If `N_Fact = 1` a full Newton iteration scheme is used, i.e., the global tangent stiffness matrix is computed and factorized at each iteration; if `N_Fact > 1` a modified Newton iteration scheme is used, i.e., the global stiffness matrix is computed and factorized every `N_Fact` iterations within each time step. The keyword "DEFAULT" sets `N_Fact = 5`.

`DTBeam` specifies the constant time increment of the time-integration in seconds. The keyword "DEFAULT" may be used to indicate that the module should employ the time increment prescribed by the driver code (FAST/stand-alone driver program).

`load_retries` specifies the maximum number of load retries allowed. This option currently works only for static analysis. For every load retry, the applied load is halved to promote convergence of the Newton-Raphson scheme in iteration of smaller load steps as opposed to one single large load step which may cause divergence of the Newton-Raphson scheme. The keyword "DEFAULT" sets `load_retries = 20`.

`NRMax` specifies the maximum number of iterations per time step in the Newton-Raphson scheme. If convergence is not reached within this number of iterations, BeamDyn returns an error message and terminates the simulation. The keyword "DEFAULT" sets `NRMax = 10`.

`Stop_Tol` specifies a tolerance parameter used in convergence criteria of a nonlinear solution that is used for the termination of the iteration. The keyword "DEFAULT" sets `Stop_Tol = 1.0E-05`. Please refer to Section 4.5.5 for more details.

`tngt_stf_fd` is a boolean that sets the flag to compute the tangent stiffness matrix using finite differencing instead of analytical differentiation. The finite differencing is performed using a central scheme. The keyword "DEFAULT" sets `tngt_stf_fd = FALSE`.

`tngt_stf_comp` is a boolean that sets the flag to compare the analytical tangent stiffness matrix against the finite differenced tangent stiffness matrix. Information is written to the terminal regarding the dof where the maximum difference is observed. If `tngt_stf_fd = FALSE` and `tngt_stf_comp = TRUE`, the analytical tangent stiffness matrix is used to solve the system of equations while the finite difference tangent stiffness matrix is used only to perform the comparison of the two matrices. The keyword "DEFAULT" sets `tngt_stf_comp = FALSE`.

`tngt_stf_pert` sets the perturbation size for finite differencing. The "DEFAULT" value based on experience is set to `1e-06`.

`tngt_stf_difftol` is the maximum allowable relative difference between the analytical and finite differenced tangent stiffness matrices. If for any entry in the matrices, the relative difference exceeds this value the simulation will terminate. The "DEFAULT" value is currently set to `1e-01`.

`RotStates` is a flag that indicates if BeamDyn's continuous states should be oriented in the rotating frame during linearization analysis when coupled to OpenFAST. If multi-blade coordinate (MBC3) analysis is performed, `RotStates` must be `true`.

Geometry Parameter

The blade geometry is defined by a curvilinear local blade reference axis. The blade reference axis locates the origin and orientation of each a local coordinate system where the cross-sectional 6x6 stiffness and mass matrices are defined in the BeamDyn blade input file. It should not really matter where in the cross section the 6x6 stiffness and mass matrices are defined relative to, as long as the reference axis is consistently defined and closely follows the natural geometry of the blade.

The blade beam model is composed of several *members* in contiguous series and each member is defined by at least three key points in BeamDyn. A cubic-spline-fit pre-processor implemented in BeamDyn automatically generates the member based on the key points and then interconnects the members into a blade. There is always a shared key point at adjacent members; therefore the total number of key points is related to number of members and key points in each member.

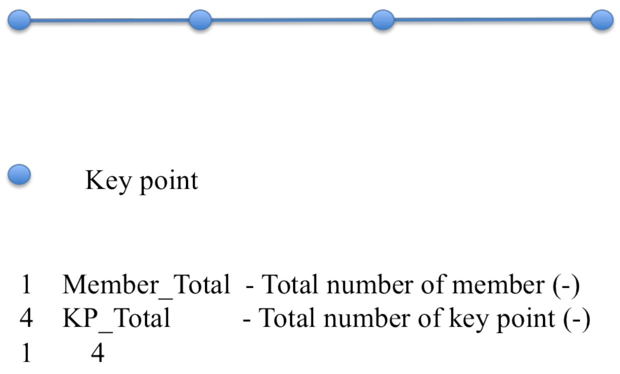
`member_total` specifies the total number of beam members used in the structure. With the LSFE discretization, a single member and a sufficiently high element order, `order_elem` below, may well be sufficient.

`kp_total` specifies the total number of key points used to define the beam members.

The following section contains `member_total` lines. Each line has two integers providing the member number (must be 1, 2, 3, etc., sequentially) and the number of key points in this member, respectively. It is noted that the number of key points in each member is not independent of the total number of key points and they should satisfy the following equality:

$$kp\_total = \sum_{i=1}^{member\_total} n_i - member\_total + 1 \tag{4.59}$$

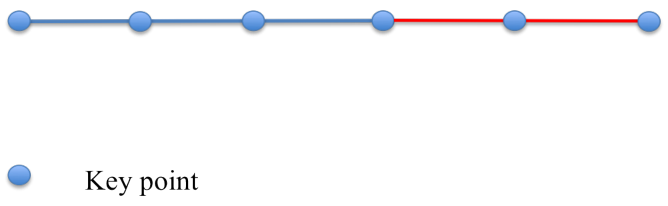
where n_i is the number of key points in the i^{th} member. Because cubic splines are implemented in BeamDyn, n_i must be greater than or equal to three. Figures Fig. 4.24 and Fig. 4.25 show two cases for member and key-point definition.



```
1   Member_Total  - Total number of member (-)
4   KP_Total      - Total number of key point (-)
1     4
```

Fig. 4.24: Member and key point definition: one member defined by four key points;

The next section defines the key-point information, preceded by two header lines. Each key point is defined by three physical coordinates (`kp_xr`, `kp_yr`, `kp_zr`) in the IEC standard blade coordinate system (the blade reference coordinate system) along with a structural twist angle (`initial_twist`) in the unit of degrees. The structural twist angle is also following the IEC standard which is defined as the twist about the negative Z_l axis. The key points are entered sequentially (from the root to tip) and there should be a total of `kp_total` lines for BeamDyn to read in the information, after two header lines. Please refer to Figure Fig. 4.26 for more details on the blade geometry definition.



```
2    Member_Total  - Total number of member (-)
6    KP_Total         - Total number of key point (-)
1      4
2      3
```

Fig. 4.25: Member and key point definition: two members defined by six key points.

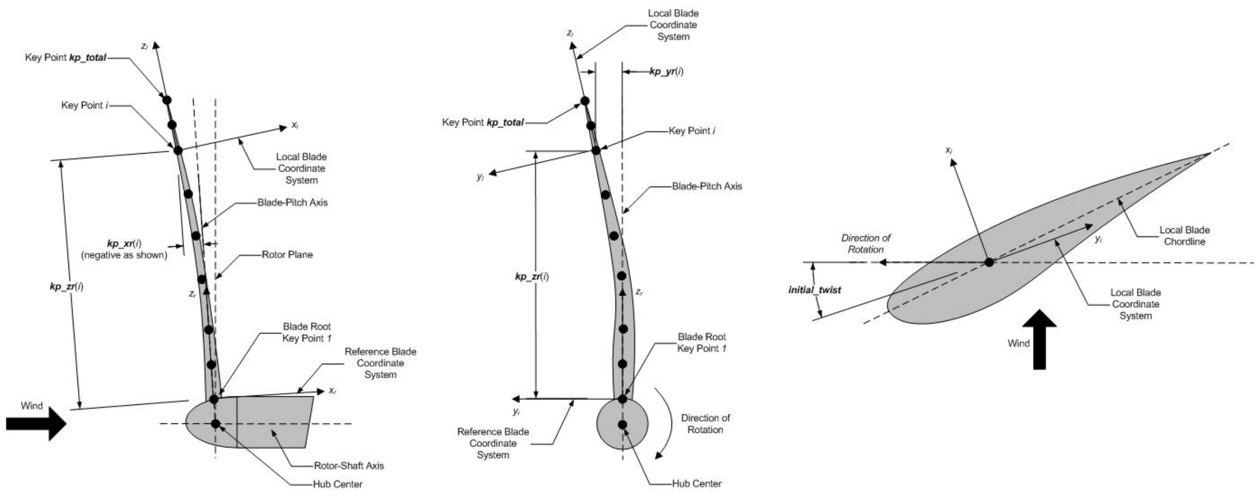


Fig. 4.26: BeamDyn Blade Geometry - Top: Side View; Middle: Front View (Looking Downwind); Bottom: Cross Section View (Looking Toward the Tip, from the Root)

Mesh Parameter

Order_Elem specifies the order of shape functions for each finite element. Each LSFE will have Order_Elem+1 nodes located at the GLL quadrature points. All LSFEs will have the same order. With the LSFE discretization, an increase in accuracy will, in general, be better achieved by increasing Order_Elem (i.e., p-refinement) rather than increasing the number of members (i.e., h-refinement). For Gauss quadrature, Order_Elem should be greater than one.

Material Parameter

BldFile is the file name of the blade input file. This name should be in quotations and can contain an absolute path or a relative path.

Pitch Actuator Parameter

In this release, the pitch actuator implemented in BeamDyn is not available. The UsePitchAct should be set to "FALSE" in this version, whereby the input blade-pitch angle prescribed by the driver code is used to orient the blade directly. PitchJ, PitchK, and PitchC specify the pitch actuator inertial, stiffness, and damping coefficient, respectively. In future releases, specifying UsePitchAct $=$ TRUE will enable a second-order pitch actuator, whereby the pitch angular orientation, velocity, and acceleration are determined by the actuator based on the input blade-pitch angle prescribed by the driver code.

Outputs

In this section of the primary input file, the user sets flags and switches for the desired output behavior.

Specifying SumPrint = TRUE causes BeamDyn to generate a summary file with name InputFile.sum. See Section 4.5.4 for summary file details.

OutFmt parameter controls the formatting of the results within the stand-alone BeamDyn output file. It needs to be a valid Fortran format string, but BeamDyn currently does not check the validity. This input is unused when BeamDyn is used coupled to FAST.

NNodeOuts specifies the number of nodes where output can be written to a file. Currently, BeamDyn can output quantities at a maximum of nine nodes.

OutNd is a list NNodeOuts long of node numbers between 1 and the number of nodes on the output mesh, separated by any combination of commas, semicolons, spaces, and/or tabs. The nodal positions are given in the summary file, if output. For Gassian quadrature, the number of nodes on the output mesh is the total number of FE nodes; for trapezoidal quadrature, this is the number of quadrature nodes.

The OutList block contains a list of output parameters. Enter one or more lines containing quoted strings that in turn contain one or more output parameter names. Separate output parameter names by any combination of commas, semicolons, spaces, and/or tabs. If you prefix a parameter name with a minus sign, "-", underscore, "_", or the characters "m" or "M", BeamDyn will multiply the value for that channel by -1 before writing the data. The parameters are written in the order they are listed in the input file. BeamDyn allows you to use multiple lines so that you can break your list into meaningful groups and so the lines can be shorter. You may enter comments after the closing quote on any of the lines. Entering a line with the string "END" at the beginning of the line or at the beginning of a quoted string found at the beginning of the line will cause BeamDyn to quit scanning for more lines of channel names. Node-related quantities are generated for the requested nodes identified through the OutNd list above. If BeamDyn encounters an unknown/invalid channel name, it warns the users but will remove the suspect channel from the output file. Please refer to Appendix Section 4.5.7 for a complete list of possible output parameters and their names.

Nodal Outputs

In addition to the named outputs in Section 4.5.3 above, BeamDyn allows for outputting the full set blade node motions and loads (tower nodes unavailable at present). Please refer to the BeamDyn_Nodes tab in the Excel file `OutListParameters.xlsx` for a complete list of possible output parameters.

This section follows the *END* statement from normal Outputs section described above, and includes a separator description line followed by the following optinos.

BldNd_BlOutNd specifies which nodes to output. This is currently unused.

The **OutList** section controls the nodal output quantities generated by BeamDyn. In this section, the user specifies the name of the channel family to output. The output name for each channel is then created internally by BeamDyn by combining the blade number, node number, and channel family name. For example, if the user specifies **TDxr** as the channel family name, the output channels will be named with the convention of **B**β**N###TDxr** where β is the blade number, and **###** is the three digit node number.

Sample Nodal Outputs section

This sample includes the `END` statement from the regular outputs section.

```
END of input file (the word "END" must appear in the first 3 columns of this last␣
↪OutList line)
---------------------- NODE OUTPUTS --------------------------------------------
         99   BldNd_BlOutNd   - Blade nodes on each blade (currently unused)
              OutList     - The next line(s) contains a list of output parameters. ␣
↪See OutListParameters.xlsx, BeamDyn_Nodes tab for a listing of available output␣
↪channels, (-)
"FxL"        - Sectional force resultants at each node expressed in l    l: a floating␣
↪coordinate system local to the deflected beam    (N)
"FyL"        - Sectional force resultants at each node expressed in l    l: a floating␣
↪coordinate system local to the deflected beam    (N)
"FzL"        - Sectional force resultants at each node expressed in l    l: a floating␣
↪coordinate system local to the deflected beam    (N)
"MxL"        - Sectional moment resultants at each node expressed in l    l: a␣
↪floating coordinate system local to the deflected beam    (N-m)
"MyL"        - Sectional moment resultants at each node expressed in l    l: a␣
↪floating coordinate system local to the deflected beam    (N-m)
"MzL"        - Sectional moment resultants at each node expressed in l    l: a␣
↪floating coordinate system local to the deflected beam    (N-m)
"Fxr"        - Sectional force resultants at each node expressed in r    r: a floating␣
↪reference coordinate system fixed to the root of the moving beam; when coupled to␣
↪FAST for blades, this is equivalent to the IEC blade (b) coordinate system    (N)
"Fyr"        - Sectional force resultants at each node expressed in r    r: a floating␣
↪reference coordinate system fixed to the root of the moving beam; when coupled to␣
↪FAST for blades, this is equivalent to the IEC blade (b) coordinate system    (N)
"Fzr"        - Sectional force resultants at each node expressed in r    r: a floating␣
↪reference coordinate system fixed to the root of the moving beam; when coupled to␣
↪FAST for blades, this is equivalent to the IEC blade (b) coordinate system    (N)
"Mxr"        - Sectional moment resultants at each node expressed in r    r: a␣
↪floating reference coordinate system fixed to the root of the moving beam; when␣
↪coupled to FAST for blades, this is equivalent to the IEC blade (b) coordinate␣
↪system    (N-m)
"Myr"        - Sectional moment resultants at each node expressed in r    r: a␣
↪floating reference coordinate system fixed to the root of the moving beam; when␣
↪coupled to FAST for blades, this is equivalent to the IEC blade (b) coordinate␣
↪system    (N-m)
```

(continues on next page)

(continued from previous page)

```
"Mzr"       - Sectional moment resultants at each node expressed in r    r: a␣
↪floating reference coordinate system fixed to the root of the moving beam; when␣
↪coupled to FAST for blades, this is equivalent to the IEC blade (b) coordinate␣
↪system    (N-m)
"TDxr"      - Sectional translational deflection (relative to the undeflected␣
↪position) at each node expressed in r    r: a floating reference coordinate system␣
↪fixed to the root of the moving beam; when coupled to FAST for blades, this is␣
↪equivalent to the IEC blade (b) coordinate system    (m)
"TDyr"      - Sectional translational deflection (relative to the undeflected␣
↪position) at each node expressed in r    r: a floating reference coordinate system␣
↪fixed to the root of the moving beam; when coupled to FAST for blades, this is␣
↪equivalent to the IEC blade (b) coordinate system    (m)
"TDzr"      - Sectional translational deflection (relative to the undeflected␣
↪position) at each node expressed in r    r: a floating reference coordinate system␣
↪fixed to the root of the moving beam; when coupled to FAST for blades, this is␣
↪equivalent to the IEC blade (b) coordinate system    (m)
"RDxr"      - Sectional angular/rotational deflection Wiener-Milenkovic parameter␣
↪(relative to the undeflected orientation) at each node expressed in r    r: a␣
↪floating reference coordinate system fixed to the root of the moving beam; when␣
↪coupled to FAST for blades, this is equivalent to the IEC blade (b) coordinate␣
↪system    (-)
"RDyr"      - Sectional angular/rotational deflection Wiener-Milenkovic parameter␣
↪(relative to the undeflected orientation) at each node expressed in r    r: a␣
↪floating reference coordinate system fixed to the root of the moving beam; when␣
↪coupled to FAST for blades, this is equivalent to the IEC blade (b) coordinate␣
↪system    (-)
"RDzr"      - Sectional angular/rotational deflection Wiener-Milenkovic parameter␣
↪(relative to the undeflected orientation) at each node expressed in r    r: a␣
↪floating reference coordinate system fixed to the root of the moving beam; when␣
↪coupled to FAST for blades, this is equivalent to the IEC blade (b) coordinate␣
↪system    (-)
"AbsXg"     - Node position in X (global coordinate)     g: the global inertial frame␣
↪coordinate system; when coupled to FAST, this is equivalent to FAST's global␣
↪inertial frame (i) coordinate system     (m)
"AbsYg"     - Node position in Y (global coordinate)     g: the global inertial frame␣
↪coordinate system; when coupled to FAST, this is equivalent to FAST's global␣
↪inertial frame (i) coordinate system     (m)
"AbsZg"     - Node position in Z (global coordinate)     g: the global inertial frame␣
↪coordinate system; when coupled to FAST, this is equivalent to FAST's global␣
↪inertial frame (i) coordinate system     (m)
"AbsXr"     - Node position in X (relative to root)     r: a floating reference␣
↪coordinate system fixed to the root of the moving beam; when coupled to FAST for␣
↪blades, this is equivalent to the IEC blade (b) coordinate system     (m)
"AbsYr"     - Node position in Y (relative to root)     r: a floating reference␣
↪coordinate system fixed to the root of the moving beam; when coupled to FAST for␣
↪blades, this is equivalent to the IEC blade (b) coordinate system     (m)
"AbsZr"     - Node position in Z (relative to root)     r: a floating reference␣
↪coordinate system fixed to the root of the moving beam; when coupled to FAST for␣
↪blades, this is equivalent to the IEC blade (b) coordinate system     (m)
"TVxg"      - Sectional translational velocities (absolute)    g: the global inertial␣
↪frame coordinate system; when coupled to FAST, this is equivalent to FAST's global␣
↪inertial frame (i) coordinate system     (m/s)
"TVyg"      - Sectional translational velocities (absolute)    g: the global inertial␣
↪frame coordinate system; when coupled to FAST, this is equivalent to FAST's global␣
↪inertial frame (i) coordinate system     (m/s)
"TVzg"      - Sectional translational velocities (absolute)    g: the global inertial␣
↪frame coordinate system; when coupled to FAST, this is equivalent to FAST's global␣
↪inertial frame (i) coordinate system     (m/s)
```

(continues on next page)

(continued from previous page)

```
"TVxl"      - Sectional translational velocities (absolute)    l: a floating␣
↪coordinate system local to the deflected beam    (m/s)
"TVyl"      - Sectional translational velocities (absolute)    l: a floating␣
↪coordinate system local to the deflected beam    (m/s)
"TVzl"      - Sectional translational velocities (absolute)    l: a floating␣
↪coordinate system local to the deflected beam    (m/s)
"TVxr"      - Sectional translational velocities (absolute)    r: a floating␣
↪reference coordinate system fixed to the root of the moving beam; when coupled to␣
↪FAST for blades, this is equivalent to the IEC blade (b) coordinate system     (m/s)
"TVyr"      - Sectional translational velocities (absolute)    r: a floating␣
↪reference coordinate system fixed to the root of the moving beam; when coupled to␣
↪FAST for blades, this is equivalent to the IEC blade (b) coordinate system     (m/s)
"TVzr"      - Sectional translational velocities (absolute)    r: a floating␣
↪reference coordinate system fixed to the root of the moving beam; when coupled to␣
↪FAST for blades, this is equivalent to the IEC blade (b) coordinate system     (m/s)
"RVxg"      - Sectional angular/rotational velocities (absolute)    g: the global␣
↪inertial frame coordinate system; when coupled to FAST, this is equivalent to␣
↪FAST's global inertial frame (i) coordinate system     (deg/s)
"RVyg"      - Sectional angular/rotational velocities (absolute)    g: the global␣
↪inertial frame coordinate system; when coupled to FAST, this is equivalent to␣
↪FAST's global inertial frame (i) coordinate system     (deg/s)
"RVzg"      - Sectional angular/rotational velocities (absolute)    g: the global␣
↪inertial frame coordinate system; when coupled to FAST, this is equivalent to␣
↪FAST's global inertial frame (i) coordinate system     (deg/s)
"RVxl"      - Sectional angular/rotational velocities (absolute)    l: a floating␣
↪coordinate system local to the deflected beam    (deg/s)
"RVyl"      - Sectional angular/rotational velocities (absolute)    l: a floating␣
↪coordinate system local to the deflected beam    (deg/s)
"RVzl"      - Sectional angular/rotational velocities (absolute)    l: a floating␣
↪coordinate system local to the deflected beam    (deg/s)
"RVxr"      - Sectional angular/rotational velocities (absolute)    r: a floating␣
↪reference coordinate system fixed to the root of the moving beam; when coupled to␣
↪FAST for blades, this is equivalent to the IEC blade (b) coordinate system     (deg/
↪s)
"RVyr"      - Sectional angular/rotational velocities (absolute)    r: a floating␣
↪reference coordinate system fixed to the root of the moving beam; when coupled to␣
↪FAST for blades, this is equivalent to the IEC blade (b) coordinate system     (deg/
↪s)
"RVzr"      - Sectional angular/rotational velocities (absolute)    r: a floating␣
↪reference coordinate system fixed to the root of the moving beam; when coupled to␣
↪FAST for blades, this is equivalent to the IEC blade (b) coordinate system     (deg/
↪s)
"TAxl"      - Sectional angular/rotational velocities (absolute)    l: a floating␣
↪coordinate system local to the deflected beam    (m/s^2)
"TAyl"      - Sectional angular/rotational velocities (absolute)    l: a floating␣
↪coordinate system local to the deflected beam    (m/s^2)
"TAzl"      - Sectional angular/rotational velocities (absolute)    l: a floating␣
↪coordinate system local to the deflected beam    (m/s^2)
"TAxr"      - Sectional angular/rotational velocities (absolute)    r: a floating␣
↪reference coordinate system fixed to the root of the moving beam; when coupled to␣
↪FAST for blades, this is equivalent to the IEC blade (b) coordinate system     (m/s^
↪2)
"TAyr"      - Sectional angular/rotational velocities (absolute)    r: a floating␣
↪reference coordinate system fixed to the root of the moving beam; when coupled to␣
↪FAST for blades, this is equivalent to the IEC blade (b) coordinate system     (m/s^
↪2)
"TAzr"      - Sectional angular/rotational velocities (absolute)    r: a floating␣
↪reference coordinate system fixed to the root of the moving beam; when coupled to␣
↪FAST for blades, this is equivalent to the IEC blade (b) coordinate system     (m/s^
↪2)
```

(continues on next page)

(continued from previous page)

```
"RAxl"      - Sectional angular/rotational velocities (absolute)    l: a floating␣
↪coordinate system local to the deflected beam    (deg/s^2)
"RAyl"      - Sectional angular/rotational velocities (absolute)    l: a floating␣
↪coordinate system local to the deflected beam    (deg/s^2)
"RAzl"      - Sectional angular/rotational velocities (absolute)    l: a floating␣
↪coordinate system local to the deflected beam    (deg/s^2)
"RAxr"      - Sectional angular/rotational velocities (absolute)    r: a floating␣
↪reference coordinate system fixed to the root of the moving beam; when coupled to␣
↪FAST for blades, this is equivalent to the IEC blade (b) coordinate system     (deg/
↪s^2)
"RAyr"      - Sectional angular/rotational velocities (absolute)    r: a floating␣
↪reference coordinate system fixed to the root of the moving beam; when coupled to␣
↪FAST for blades, this is equivalent to the IEC blade (b) coordinate system     (deg/
↪s^2)
"RAzr"      - Sectional angular/rotational velocities (absolute)    r: a floating␣
↪reference coordinate system fixed to the root of the moving beam; when coupled to␣
↪FAST for blades, this is equivalent to the IEC blade (b) coordinate system     (deg/
↪s^2)
"PFxL"      - Applied point forces at each node expressed in l    l: a floating␣
↪coordinate system local to the deflected beam    (N)
"PFyL"      - Applied point forces at each node expressed in l    l: a floating␣
↪coordinate system local to the deflected beam    (N)
"PFzL"      - Applied point forces at each node expressed in l    l: a floating␣
↪coordinate system local to the deflected beam    (N)
"PMxL"      - Applied point moments at each node expressed in l    l: a floating␣
↪coordinate system local to the deflected beam    (N-m)
"PMyL"      - Applied point moments at each node expressed in l    l: a floating␣
↪coordinate system local to the deflected beam    (N-m)
"PMzL"      - Applied point moments at each node expressed in l    l: a floating␣
↪coordinate system local to the deflected beam    (N-m)
"DFxL"      - Applied distributed forces at each node expressed in l    l: a floating␣
↪coordinate system local to the deflected beam    (N/m)
"DFyL"      - Applied distributed forces at each node expressed in l    l: a floating␣
↪coordinate system local to the deflected beam    (N/m)
"DFzL"      - Applied distributed forces at each node expressed in l    l: a floating␣
↪coordinate system local to the deflected beam    (N/m)
"DMxL"      - Applied distributed moments at each node expressed in l    l: a␣
↪floating coordinate system local to the deflected beam    (N-m/m)
"DMyL"      - Applied distributed moments at each node expressed in l    l: a␣
↪floating coordinate system local to the deflected beam    (N-m/m)
"DMzL"      - Applied distributed moments at each node expressed in l    l: a␣
↪floating coordinate system local to the deflected beam    (N-m/m)
"DFxR"      - Applied distributed forces at each node expressed in r    r: a floating␣
↪reference coordinate system fixed to the root of the moving beam; when coupled to␣
↪FAST for blades, this is equivalent to the IEC blade (b) coordinate system     (N/m)
"DFyR"      - Applied distributed forces at each node expressed in r    r: a floating␣
↪reference coordinate system fixed to the root of the moving beam; when coupled to␣
↪FAST for blades, this is equivalent to the IEC blade (b) coordinate system     (N/m)
"DFzR"      - Applied distributed forces at each node expressed in r    r: a floating␣
↪reference coordinate system fixed to the root of the moving beam; when coupled to␣
↪FAST for blades, this is equivalent to the IEC blade (b) coordinate system     (N/m)
"DMxR"      - Applied distributed forces at each node expressed in r    r: a floating␣
↪reference coordinate system fixed to the root of the moving beam; when coupled to␣
↪FAST for blades, this is equivalent to the IEC blade (b) coordinate system     (N-m/
↪m)
"DMyR"      - Applied distributed forces at each node expressed in r    r: a floating␣
↪reference coordinate system fixed to the root of the moving beam; when coupled to␣
↪FAST for blades, this is equivalent to the IEC blade (b) coordinate system     (N-m/
↪m)
```

(continues on next page)

(continued from previous page)

```
"DMzR"      - Applied distributed forces at each node expressed in r    r: a floating␣
↪reference coordinate system fixed to the root of the moving beam; when coupled to␣
↪FAST for blades, this is equivalent to the IEC blade (b) coordinate system     (N-m/
↪m)
"FFbxl"     - Gyroscopic force x     l: a floating coordinate system local to the␣
↪deflected beam    (N)
"FFbyl"     - Gyroscopic force y     l: a floating coordinate system local to the␣
↪deflected beam    (N)
"FFbzl"     - Gyroscopic force z     l: a floating coordinate system local to the␣
↪deflected beam    (N)
"FFbxr"     - Gyroscopic force x     r: a floating reference coordinate system fixed␣
↪to the root of the moving beam; when coupled to FAST for blades, this is equivalent␣
↪to the IEC blade (b) coordinate system     (N)
"FFbyr"     - Gyroscopic force y     r: a floating reference coordinate system fixed␣
↪to the root of the moving beam; when coupled to FAST for blades, this is equivalent␣
↪to the IEC blade (b) coordinate system     (N)
"FFbzr"     - Gyroscopic force z     r: a floating reference coordinate system fixed␣
↪to the root of the moving beam; when coupled to FAST for blades, this is equivalent␣
↪to the IEC blade (b) coordinate system     (N)
"MFbxl"     - Gyroscopic moment about x     l: a floating coordinate system local to␣
↪the deflected beam    (N-m)
"MFbyl"     - Gyroscopic moment about y     l: a floating coordinate system local to␣
↪the deflected beam    (N-m)
"MFbzl"     - Gyroscopic moment about z     l: a floating coordinate system local to␣
↪the deflected beam    (N-m)
"MFbxr"     - Gyroscopic moment about x     r: a floating reference coordinate system␣
↪fixed to the root of the moving beam; when coupled to FAST for blades, this is␣
↪equivalent to the IEC blade (b) coordinate system     (N-m)
"MFbyr"     - Gyroscopic moment about y     r: a floating reference coordinate system␣
↪fixed to the root of the moving beam; when coupled to FAST for blades, this is␣
↪equivalent to the IEC blade (b) coordinate system     (N-m)
"MFbzr"     - Gyroscopic moment about z     r: a floating reference coordinate system␣
↪fixed to the root of the moving beam; when coupled to FAST for blades, this is␣
↪equivalent to the IEC blade (b) coordinate system     (N-m)
"FFcxl"     - Elastic restoring force Fc x     l: a floating coordinate system local␣
↪to the deflected beam    (N)
"FFcyl"     - Elastic restoring force Fc y     l: a floating coordinate system local␣
↪to the deflected beam    (N)
"FFczl"     - Elastic restoring force Fc z     l: a floating coordinate system local␣
↪to the deflected beam    (N)
"FFcxr"     - Elastic restoring force Fc x     r: a floating reference coordinate␣
↪system fixed to the root of the moving beam; when coupled to FAST for blades, this␣
↪is equivalent to the IEC blade (b) coordinate system     (N)
"FFcyr"     - Elastic restoring force Fc y     r: a floating reference coordinate␣
↪system fixed to the root of the moving beam; when coupled to FAST for blades, this␣
↪is equivalent to the IEC blade (b) coordinate system     (N)
"FFczr"     - Elastic restoring force Fc z     r: a floating reference coordinate␣
↪system fixed to the root of the moving beam; when coupled to FAST for blades, this␣
↪is equivalent to the IEC blade (b) coordinate system     (N)
"MFcxl"     - Elastic restoring moment Fc about x     l: a floating coordinate system␣
↪local to the deflected beam    (N-m)
"MFcyl"     - Elastic restoring moment Fc about y     l: a floating coordinate system␣
↪local to the deflected beam    (N-m)
"MFczl"     - Elastic restoring moment Fc about z     l: a floating coordinate system␣
↪local to the deflected beam    (N-m)
"MFcxr"     - Elastic restoring moment Fc about x     r: a floating reference␣
↪coordinate system fixed to the root of the moving beam; when coupled to FAST for␣
↪blades, this is equivalent to the IEC blade (b) coordinate system     (N-m)
```

(continues on next page)

(continued from previous page)

```
"MFcyr"     - Elastic restoring moment Fc about y     r: a floating reference coordinate system fixed to the root of the moving beam; when coupled to FAST for blades, this is equivalent to the IEC blade (b) coordinate system     (N-m)
"MFczr"     - Elastic restoring moment Fc about z     r: a floating reference coordinate system fixed to the root of the moving beam; when coupled to FAST for blades, this is equivalent to the IEC blade (b) coordinate system     (N-m)
"FFdxl"     - Elastic restoring force Fd x     l: a floating coordinate system local to the deflected beam     (N)
"FFdyl"     - Elastic restoring force Fd y     l: a floating coordinate system local to the deflected beam     (N)
"FFdzl"     - Elastic restoring force Fd z     l: a floating coordinate system local to the deflected beam     (N)
"FFdxr"     - Elastic restoring force Fd x     r: a floating reference coordinate system fixed to the root of the moving beam; when coupled to FAST for blades, this is equivalent to the IEC blade (b) coordinate system     (N)
"FFdyr"     - Elastic restoring force Fd y     r: a floating reference coordinate system fixed to the root of the moving beam; when coupled to FAST for blades, this is equivalent to the IEC blade (b) coordinate system     (N)
"FFdzr"     - Elastic restoring force Fd z     r: a floating reference coordinate system fixed to the root of the moving beam; when coupled to FAST for blades, this is equivalent to the IEC blade (b) coordinate system     (N)
"MFdxl"     - Elastic restoring moment Fd about x     l: a floating coordinate system local to the deflected beam     (N-m)
"MFdyl"     - Elastic restoring moment Fd about y     l: a floating coordinate system local to the deflected beam     (N-m)
"MFdzl"     - Elastic restoring moment Fd about z     l: a floating coordinate system local to the deflected beam     (N-m)
"MFdxr"     - Elastic restoring moment Fd about x     r: a floating reference coordinate system fixed to the root of the moving beam; when coupled to FAST for blades, this is equivalent to the IEC blade (b) coordinate system     (N-m)
"MFdyr"     - Elastic restoring moment Fd about y     r: a floating reference coordinate system fixed to the root of the moving beam; when coupled to FAST for blades, this is equivalent to the IEC blade (b) coordinate system     (N-m)
"MFdzr"     - Elastic restoring moment Fd about z     r: a floating reference coordinate system fixed to the root of the moving beam; when coupled to FAST for blades, this is equivalent to the IEC blade (b) coordinate system     (N-m)
"FFgxl"     - Gravity force x     l: a floating coordinate system local to the deflected beam     (N)
"FFgyl"     - Gravity force y     l: a floating coordinate system local to the deflected beam     (N)
"FFgzl"     - Gravity force z     l: a floating coordinate system local to the deflected beam     (N)
"FFgxr"     - Gravity force x     r: a floating reference coordinate system fixed to the root of the moving beam; when coupled to FAST for blades, this is equivalent to the IEC blade (b) coordinate system     (N)
"FFgyr"     - Gravity force y     r: a floating reference coordinate system fixed to the root of the moving beam; when coupled to FAST for blades, this is equivalent to the IEC blade (b) coordinate system     (N)
"FFgzr"     - Gravity force z     r: a floating reference coordinate system fixed to the root of the moving beam; when coupled to FAST for blades, this is equivalent to the IEC blade (b) coordinate system     (N)
"MFgxl"     - Gravity moment about x     l: a floating coordinate system local to the deflected beam     (N-m)
"MFgyl"     - Gravity moment about y     l: a floating coordinate system local to the deflected beam     (N-m)
"MFgzl"     - Gravity moment about z     l: a floating coordinate system local to the deflected beam     (N-m)
```

(continues on next page)

(continued from previous page)

```
"MFgxr"      - Gravity moment about x      r: a floating reference coordinate system␣
↪fixed to the root of the moving beam; when coupled to FAST for blades, this is␣
↪equivalent to the IEC blade (b) coordinate system     (N-m)
"MFgyr"      - Gravity moment about y      r: a floating reference coordinate system␣
↪fixed to the root of the moving beam; when coupled to FAST for blades, this is␣
↪equivalent to the IEC blade (b) coordinate system     (N-m)
"MFgzr"      - Gravity moment about z      r: a floating reference coordinate system␣
↪fixed to the root of the moving beam; when coupled to FAST for blades, this is␣
↪equivalent to the IEC blade (b) coordinate system     (N-m)
"FFixl"      - Inertial force x      l: a floating coordinate system local to the␣
↪deflected beam     (N)
"FFiyl"      - Inertial force y      l: a floating coordinate system local to the␣
↪deflected beam     (N)
"FFizl"      - Inertial force z      l: a floating coordinate system local to the␣
↪deflected beam     (N)
"FFixr"      - Inertial force x      r: a floating reference coordinate system fixed to␣
↪the root of the moving beam; when coupled to FAST for blades, this is equivalent to␣
↪the IEC blade (b) coordinate system     (N)
"FFiyr"      - Inertial force y      r: a floating reference coordinate system fixed to␣
↪the root of the moving beam; when coupled to FAST for blades, this is equivalent to␣
↪the IEC blade (b) coordinate system     (N)
"FFizr"      - Inertial force z      r: a floating reference coordinate system fixed to␣
↪the root of the moving beam; when coupled to FAST for blades, this is equivalent to␣
↪the IEC blade (b) coordinate system     (N)
"MFixl"      - Inertial moment about x      l: a floating coordinate system local to␣
↪the deflected beam     (N-m)
"MFiyl"      - Inertial moment about y      l: a floating coordinate system local to␣
↪the deflected beam     (N-m)
"MFizl"      - Inertial moment about z      l: a floating coordinate system local to␣
↪the deflected beam     (N-m)
"MFixr"      - Inertial moment about x      r: a floating reference coordinate system␣
↪fixed to the root of the moving beam; when coupled to FAST for blades, this is␣
↪equivalent to the IEC blade (b) coordinate system     (N-m)
"MFiyr"      - Inertial moment about y      r: a floating reference coordinate system␣
↪fixed to the root of the moving beam; when coupled to FAST for blades, this is␣
↪equivalent to the IEC blade (b) coordinate system     (N-m)
"MFizr"      - Inertial moment about z      r: a floating reference coordinate system␣
↪fixed to the root of the moving beam; when coupled to FAST for blades, this is␣
↪equivalent to the IEC blade (b) coordinate system     (N-m)
END of input file (the word "END" must appear in the first 3 columns of this last␣
↪OutList line)
---------------------------------------------------------------------------------------
↪-
```

Blade Input File

The blade input file defines the cross-sectional properties at various stations along a blade and six damping coefficient for the whole blade. A sample BeamDyn blade input file is given in Section 4.5.7. The blade input file begins with two lines of header information, which is for the user but is not used by the software.

Blade Parameters

`Station_Total` specifies the number cross-sectional stations along the blade axis used in the analysis.

`Damp_Type` specifies if structural damping is considered in the analysis. If `Damp_Type = 0`, then no damping is considered in the analysis and the six damping coefficient in the next section will be ignored. If `Damp_Type = 1`, structural damping will be included in the analysis.

Damping Coefficient

This section specifies six damping coefficients, μ_{ii} with $i \in [1, 6]$, for six DOFs (three translations and three rotations). Viscous damping is implemented in BeamDyn where the damping forces are proportional to the strain rate. These are stiffness-proportional damping coefficients, whereby the 6×6 damping matrix at each cross section is scaled from the 6×6 stiffness matrix by these diagonal entries of a 6×6 scaling matrix:

$$\underline{\mathcal{F}}^{Damp} = \underline{\underline{\mu}}\, \underline{\underline{S}}\, \underline{\dot{\epsilon}} \tag{4.60}$$

where $\underline{\mathcal{F}}^{Damp}$ is the damping force, $\underline{\underline{S}}$ is the 6×6 cross-sectional stiffness matrix, $\underline{\dot{\epsilon}}$ is the strain rate, and $\underline{\underline{\mu}}$ is the damping coefficient matrix defined as

$$\underline{\underline{\mu}} = \begin{bmatrix} \mu_{11} & 0 & 0 & 0 & 0 & 0 \\ 0 & \mu_{22} & 0 & 0 & 0 & 0 \\ 0 & 0 & \mu_{33} & 0 & 0 & 0 \\ 0 & 0 & 0 & \mu_{44} & 0 & 0 \\ 0 & 0 & 0 & 0 & \mu_{55} & 0 \\ 0 & 0 & 0 & 0 & 0 & \mu_{66} \end{bmatrix} \tag{4.61}$$

Distributed Properties

This section specifies the cross-sectional properties at each of the `Station_Total` stations. For each station, a non-dimensional parameter η specifies the station location along the local blade reference axis ranging from $[0.0, 1.0]$. The first and last station parameters must be set to 0.0 (for the blade root) and 1.0 (for the blade tip), respectively.

Following the station location parameter η, there are two 6×6 matrices providing the structural and inertial properties for this cross-section. First is the stiffness matrix and then the mass matrix. We note that these matrices are defined in a local coordinate system along the blade axis with Z_l directing toward the unit tangent vector of the blade reference axis. For a cross-section without coupling effects, for example, the stiffness matrix is given as follows:

$$\begin{bmatrix} K_{ShrFlp} & 0 & 0 & 0 & 0 & 0 \\ 0 & K_{ShrEdg} & 0 & 0 & 0 & 0 \\ 0 & 0 & EA & 0 & 0 & 0 \\ 0 & 0 & 0 & EI_{Edg} & 0 & 0 \\ 0 & 0 & 0 & 0 & EI_{Flp} & 0 \\ 0 & 0 & 0 & 0 & 0 & GJ \end{bmatrix} \tag{4.62}$$

where K_{ShrEdg} and K_{ShrFlp} are the edge and flap shear stiffnesses, respectively; EA is the extension stiffness; EI_{Edg} and EI_{Flp} are the edge and flap stiffnesses, respectively; and GJ is the torsional stiffness. It is pointed out that for a generic cross-section, the sectional property matrices can be derived from a sectional analysis tool, e.g. VABS, BECAS, or NuMAD/BPE.

A generalized sectional mass matrix is given by:

$$\begin{bmatrix} m & 0 & 0 & 0 & 0 & -mY_{cm} \\ 0 & m & 0 & 0 & 0 & mX_{cm} \\ 0 & 0 & m & mY_{cm} & -mX_{cm} & 0 \\ 0 & 0 & mY_{cm} & i_{Edg} & -i_{cp} & 0 \\ 0 & 0 & -mX_{cm} & -i_{cp} & i_{Flp} & 0 \\ -mY_{cm} & mX_{cm} & 0 & 0 & 0 & i_{plr} \end{bmatrix} \tag{4.63}$$

where m is the mass density per unit span; X_{cm} and Y_{cm} are the local coordinates of the sectional center of mass, respectively; i_{Edg} and i_{Flp} are the edge and flap mass moments of inertia per unit span, respectively; i_{plr} is the polar moment of inertia per unit span; and i_{cp} is the sectional cross-product of inertia per unit span. We note that for beam structure, the i_{plr} is given as (although this relationship is not checked by BeamDyn)

$$i_{plr} = i_{Edg} + i_{Flp} \tag{4.64}$$

4.5.4 Output Files

BeamDyn produces three types of output files, depending on the options selected: an echo file, a summary file, and a time-series results file. The following sections detail the purpose and contents of these files.

Echo File

If the user sets the `Echo` flag to TRUE in the BeamDyn primary input file, the contents of this file will be echoed to a file with the naming convention `InputFile.ech`. The echo file is helpful for debugging the input files. The contents of an echo file will be truncated if BeamDyn encounters an error while parsing an input file. The error usually corresponds to the line after the last successfully echoed line.

Summary File

In stand-alone mode, BeamDyn generates a summary file with the naming convention, `InputFile.sum` if the `SumPrint` parameter is set to TRUE. When coupled to FAST, the summary file is named `InputFile.BD.sum`. This file summarizes key information about the simulation, including:

- Blade mass.
- Blade length.
- Blade center of mass.
- Initial global position vector in BD coordinate system.
- Initial global rotation tensor in BD coordinate system.
- Analysis type.
- Numerical damping coefficients.
- Time step size.
- Maximum number of iterations in the Newton-Raphson solution.
- Convergence parameter in the stopping criterion.
- Factorization frequency in the Newton-Raphson solution.

- Numerical integration (quadrature) method.
- FE mesh refinement factor used in trapezoidal quadrature.
- Number of elements.
- Number of FE nodes.
- Initial position vectors of FE nodes in BD coordinate system.
- Initial rotation vectors of FE nodes in BD coordinate system.
- Quadrature point position vectors in BD coordinate system. For Gauss quadrature, the physical coordinates of Gauss points are listed. For trapezoidal quadrature, the physical coordinates of the quadrature points are listed.
- Sectional stiffness and mass matrices at quadrature points in local blade reference coordinate system. These are the data being used in calculations at quadrature points and they can be different from the section in Blade Input File since BeamDyn linearly interpolates the sectional properties into quadrature points based on need.
- Initial displacement vectors of FE nodes in BD coordinate system.
- Initial rotational displacement vectors of FE nodes in BD coordinate system.
- Initial translational velocity vectors of FE nodes in BD coordinate system.
- Initial angular velocity vectors of FE nodes in BD coordinate system.
- Requested output information.

All of these quantities are output in this file in the BD coordinate system, the one being used internally in BeamDyn calculations. The initial blade reference coordinate system, denoted by a subscript $r0$ that follows the IEC standard, is related to the internal BD coordinate system by Table 4.5 in Section 4.5.5.

Results File

The BeamDyn time-series results are written to a text-based file with the naming convention `DriverInputFile.out` where `DriverInputFile` is the name of the driver input file when BeamDyn is run in the stand-alone mode. If BeamDyn is coupled to FAST, then FAST will generate a master results file that includes the BeamDyn results. The results in `DriverInputFile.out` are in table format, where each column is a data channel (the first column always being the simulation time), and each row corresponds to a simulation time step. The data channel are specified in the OUTPUT section of the primary input file. The column format of the BeamDyn-generated file is specified using the `OutFmt` parameters of the primary input file.

4.5.5 BeamDyn Theory

This section focuses on the theory behind the BeamDyn module. The theoretical foundation, numerical tools, and some special handling in the implementation will be introduced. References will be provided in each section detailing the theories and numerical tools.

In this chapter, matrix notation is used to denote vectorial or vectorial-like quantities. For example, an underline denotes a vector $\underline{u}$, an over bar denotes unit vector $\bar{n}$, and a double underline denotes a tensor $\underline{\underline{\Delta}}$. Note that sometimes the underlines only denote the dimension of the corresponding matrix.

Coordinate Systems

Fig. 4.26 (in Section 4.5.3) and Fig. 4.27 show the coordinate system used in BeamDyn.

Global Coordinate System

The global coordinate system is denoted as `X`, `Y`, and `Z` in Fig. 4.27. This is an inertial frame and in FAST its origin is usually placed at the bottom of the tower as shown.

BD Coordinate System

The BD coordinate system is denoted as x_1, x_2, and x_3 respectively in Fig. 4.27. This is an inertial frame used internally in BeamDyn (i.e., doesn't rotate with the rotor) and its origin is placed at the initial position of the blade root point.

Blade Reference Coordinate System

The blade reference coordinate system is denoted as X_{rt}, Y_{rt}, and Z_{rt} in Fig. 4.27 at initialization ($t = 0$). The blade reference coordinate system is a floating frame that attaches at the blade root and is rotating with the blade. Its origin is at the blade root and the directions of axes following the IEC standard, i.e., Z_r is pointing along the blade axis from root to tip; Y_r pointing nominally towards the trailing edge of the blade and parallel with the chord line at the zero-twist blade station; and X_r is orthogonal with the Y_r and Z_r axes, such that they form a right-handed coordinate system (pointing nominally downwind). We note that the initial blade reference coordinate system, denoted by subscript $r0$, coincides with the BD coordinate system, which is used internally in BeamDyn and introduced in the previous section. The axis convention relations between the initial blade reference coordinate system and the BD coordinate system can be found in Table 4.5.

Table 4.5: Transformation between blade coordinate system and BD coordinate system.

Blade Frame	X_{r0}	Y_{r0}	Z_{r0}
BD Frame	x_2	x_3	x_1

Local blade coordinate system

The local blade coordinate system is used for some input and output quantities, for example, the cross-sectional mass and stiffness matrices and the the sectional force and moment resultants. This coordinate system is different from the blade reference coordinate system in that its Z_l axis is always tangent to the blade axis as the blade deflects. Note that a subscript l denotes the local blade coordinate system.

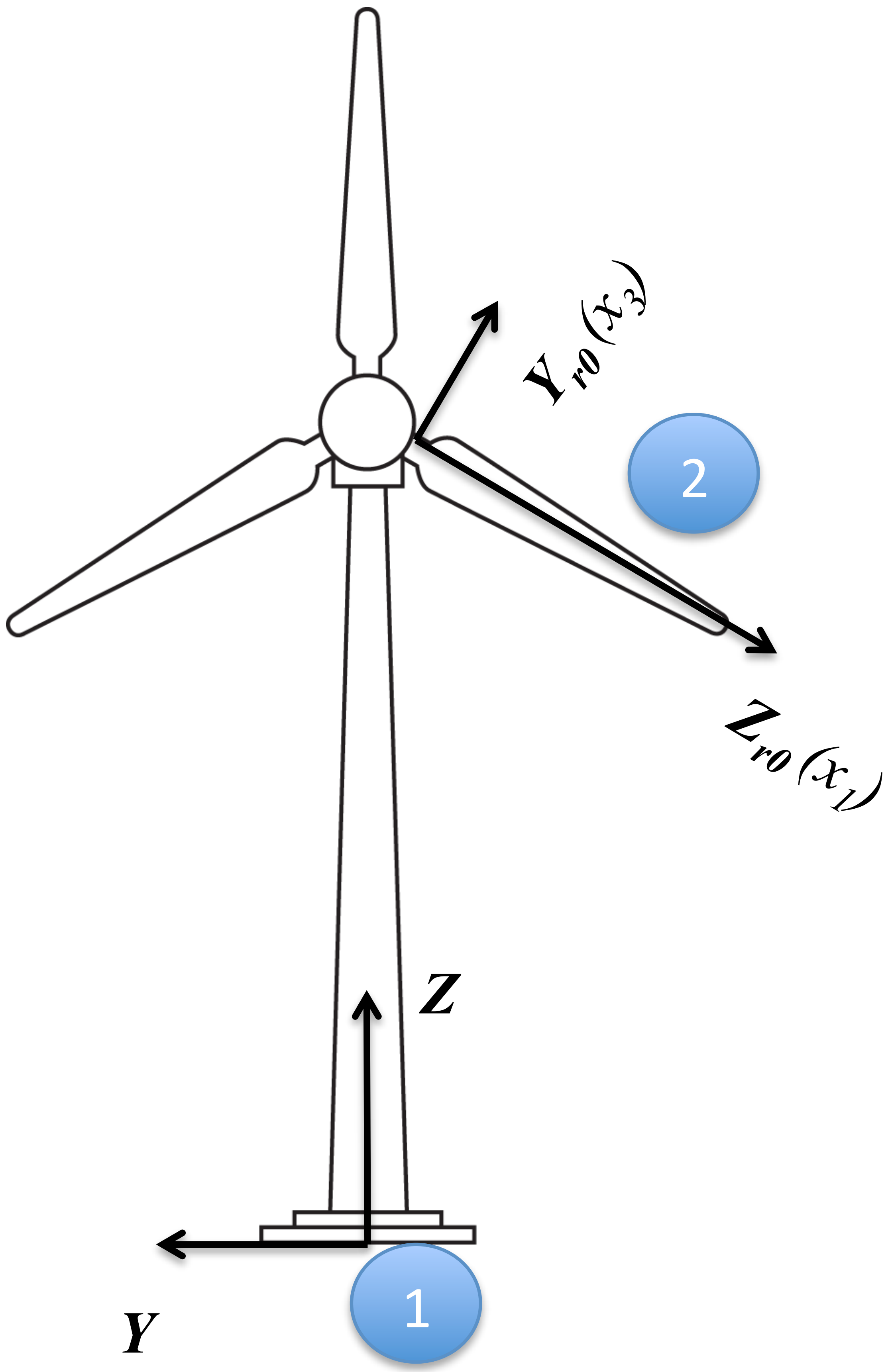


Fig. 4.27: Global, blade reference, and internal coordinate systems in BeamDyn. Illustration by Al Hicks, NREL.

Geometrically Exact Beam Theory

The theoretical foundation of BeamDyn is the geometrically exact beam theory. This theory features the capability of beams that are initially curved and twisted and subjected to large displacement and rotations. Along with a proper two-dimensional (2D) cross-sectional analysis, the coupling effects between all six DOFs, including extension, bending, shear, and torsion, can be captured by GEBT as well . The term, "geometrically exact" refer to the fact that there is no approximation made on the geometries, including both initial and deformed geometries, in formulating the equations [Hod06].

The governing equations of motion for geometrically exact beam theory can be written as [Bau10]

$$\begin{aligned} \dot{\underline{h}} - \underline{F}' &= \underline{f} \\ \dot{\underline{g}} + \dot{\tilde{u}}\underline{h} - \underline{M}' + (\tilde{x}_0' + \tilde{u}')^T \underline{F} &= \underline{m} \end{aligned} \tag{4.65}$$

where $\underline{h}$ and $\underline{g}$ are the linear and angular momenta resolved in the inertial coordinate system, respectively; $\underline{F}$ and $\underline{M}$ are the beam's sectional force and moment resultants, respectively; $\underline{u}$ is the one-dimensional (1D) displacement of a point on the reference line; $\underline{x}_0$ is the position vector of a point along the beam's reference line; and $\underline{f}$ and $\underline{m}$ are the distributed force and moment applied to the beam structure. The notation $(\bullet)'$ indicates a derivative with respect to beam axis x_1 and $(\dot{\bullet})$ indicates a derivative with respect to time. The tilde operator $(\widetilde{\bullet})$ defines a skew-symmetric tensor corresponding to the given vector. In the literature, it is also termed as "cross-product matrix". For example,

$$\widetilde{n} = \begin{bmatrix} 0 & -n_3 & n_2 \\ n_3 & 0 & -n_1 \\ -n_2 & n_1 & 0 \end{bmatrix}$$

The constitutive equations relate the velocities to the momenta and the 1D strain measures to the sectional resultants as

$$\begin{aligned} \begin{Bmatrix} \underline{h} \\ \underline{g} \end{Bmatrix} &= \underline{\underline{\mathcal{M}}} \begin{Bmatrix} \dot{\underline{u}} \\ \underline{\omega} \end{Bmatrix} \\ \begin{Bmatrix} \underline{F} \\ \underline{M} \end{Bmatrix} &= \underline{\underline{\mathcal{C}}} \begin{Bmatrix} \underline{\epsilon} \\ \underline{\kappa} \end{Bmatrix} \end{aligned} \tag{4.66}$$

where $\underline{\underline{\mathcal{M}}}$ and $\underline{\underline{\mathcal{C}}}$ are the 6×6 sectional mass and stiffness matrices, respectively (note that they are not really tensors); $\underline{\epsilon}$ and $\underline{\kappa}$ are the 1D strains and curvatures, respectively; and, $\underline{\omega}$ is the angular velocity vector that is defined by the rotation tensor $\underline{\underline{R}}$ as $\underline{\omega} = axial(\dot{\underline{\underline{R}}}\,\underline{\underline{R}}^T)$. The axial vector $\underline{a}$ associated with a second-order tensor $\underline{\underline{A}}$ is denoted $\underline{a} = axial(\underline{\underline{A}})$ and its components are defined as

$$\underline{a} = axial(\underline{\underline{A}}) = \begin{Bmatrix} a_1 \\ a_2 \\ a_3 \end{Bmatrix} = \frac{1}{2} \begin{Bmatrix} A_{32} - A_{23} \\ A_{13} - A_{31} \\ A_{21} - A_{12} \end{Bmatrix} \tag{4.67}$$

The 1D strain measures are defined as

$$\begin{Bmatrix} \underline{\epsilon} \\ \underline{\kappa} \end{Bmatrix} = \begin{Bmatrix} \underline{x}_0' + \underline{u}' - (\underline{\underline{R}}\,\underline{\underline{R}}_0)\bar{\imath}_1 \\ \underline{k} \end{Bmatrix} \tag{4.68}$$

where $\underline{k} = axial[(\underline{\underline{R}}\underline{\underline{R}}_0)'(\underline{\underline{R}}\underline{\underline{R}}_0)^T]$ is the sectional curvature vector resolved in the inertial basis; $\underline{\underline{R}}_0$ is the initial rotation tensor; and $\bar{\imath}_1$ is the unit vector along x_1 direction in the inertial basis. These three sets of equations, including equations of motion Eq. (4.65), constitutive equations Eq. (4.66), and kinematical equations Eq. (4.68), provide a full mathematical description of the beam elasticity problems.

Numerical Implementation with Legendre Spectral Finite Elements

For a displacement-based finite element implementation, there are six degree-of-freedoms at each node: three displacement components and three rotation components. Here we use $\underline{q}$ to denote the elemental displacement array as $\underline{q} = \left[\underline{u}^T \ \ \underline{c}^T\right]$ where $\underline{u}$ is the displacement and $\underline{c}$ is the rotation-parameter vector. The acceleration array can thus be defined as $\underline{a} = \left[\underline{\ddot{u}}^T \ \ \underline{\dot{\omega}}^T\right]$. For nonlinear finite-element analysis, the discretized and incremental forms of displacement, velocity, and acceleration are written as

$$\begin{aligned} \underline{q}(x_1) &= \underline{\underline{N}}\,\underline{\hat{q}} \quad & \Delta\underline{q}^T &= \left[\Delta\underline{u}^T \ \ \Delta\underline{c}^T\right] \\ \underline{v}(x_1) &= \underline{\underline{N}}\,\underline{\hat{v}} \quad & \Delta\underline{v}^T &= \left[\Delta\underline{\dot{u}}^T \ \ \Delta\underline{\omega}^T\right] \\ \underline{a}(x_1) &= \underline{\underline{N}}\,\underline{\hat{a}} \quad & \Delta\underline{a}^T &= \left[\Delta\underline{\ddot{u}}^T \ \ \Delta\underline{\dot{\omega}}^T\right] \end{aligned} \tag{4.69}$$

where $\underline{\underline{N}}$ is the shape function matrix and $(\hat{\cdot})$ denotes a column matrix of nodal values.

The displacement fields in an element are approximated as

$$\begin{aligned} \underline{u}(\xi) &= h^k(\xi)\underline{\hat{u}}^k \\ \underline{u}'(\xi) &= h^{k\prime}(\xi)\underline{\hat{u}}^k \end{aligned} \tag{4.70}$$

where $h^k(\xi)$, the component of shape function matrix $\underline{\underline{N}}$, is the p^{th}-order polynomial Lagrangian-interpolant shape function of node k, $k = \{1, 2, ..., p+1\}$, $\underline{\hat{u}}^k$ is the k^{th} nodal value, and $\xi \in [-1, 1]$ is the element natural coordinate. However, as discussed in [BEH08], the 3D rotation field cannot simply be interpolated as the displacement field in the form of

$$\begin{aligned} \underline{c}(\xi) &= h^k(\xi)\underline{\hat{c}}^k \\ \underline{c}'(\xi) &= h^{k\prime}(\xi)\underline{\hat{c}}^k \end{aligned} \tag{4.71}$$

where $\underline{c}$ is the rotation field in an element and $\underline{\hat{c}}^k$ is the nodal value at the k^{th} node, for three reasons:

1) rotations do not form a linear space so that they must be "composed" rather than added;
2) a rescaling operation is needed to eliminate the singularity existing in the vectorial rotation parameters;
3) the rotation field lacks objectivity, which, as defined by [JelenicC99], refers to the invariance of strain measures computed through interpolation to the addition of a rigid-bodymotion.

Therefore, we adopt the more robust interpolation approach proposed by [JelenicC99] to deal with the finite rotations. Our approach is described as follows

Step 1: Compute the nodal relative rotations, $\underline{\hat{r}}^k$, by removing the reference rotation, $\underline{\hat{c}}^1$, from the finite rotation at each node, $\underline{\hat{r}}^k = (\underline{\hat{c}}^{1-}) \oplus \underline{\hat{c}}^k$. It is noted that the minus sign on $\underline{\hat{c}}^1$ denotes that the relative rotation is calculated by removing the reference rotation from each node. The composition in that equation is an equivalent of $\underline{\underline{R}}(\underline{\hat{r}}^k) = \underline{\underline{R}}^T(\underline{\hat{c}}^1)\ \underline{\underline{R}}(\underline{c}^k)$.

Step 2: Interpolate the relative-rotation field: $\underline{r}(\xi) = h^k(\xi)\underline{\hat{r}}^k$ and $\underline{r}'(\xi) = h^{k\prime}(\xi)\underline{\hat{r}}^k$. Find the curvature field $\underline{\kappa}(\xi) = \underline{\underline{R}}(\underline{\hat{c}}^1)\underline{\underline{H}}(\underline{r})\underline{r}'$, where $\underline{\underline{H}}$ is the tangent tensor that relates the curvature vector $\underline{k}$ and rotation vector $\underline{c}$ as

$$\underline{k} = \underline{\underline{H}}\ \underline{c}' \tag{4.72}$$

Step 3: Restore the rigid-body rotation removed in Step 1: $\underline{c}(\xi) = \underline{\hat{c}}^1 \oplus \underline{r}(\xi)$.

Note that the relative-rotation field can be computed with respect to any of the nodes of the element; we choose node 1 as the reference node for convenience. In the LSFE approach, shape functions (i.e., those composing $\underline{\underline{N}}$) are p^{th}-order Lagrangian interpolants, where nodes are located at the $p+1$ Gauss-Lobatto-Legendre (GLL) points in the $[-1, 1]$ element natural-coordinate domain. Fig. 4.28 shows representative LSFE basis functions for fourth- and eighth-order elements. Note that nodes are clustered near element endpoints. More details on the LSFE and its applications can be found in References [Pat84][RP87][SG03][SG04].

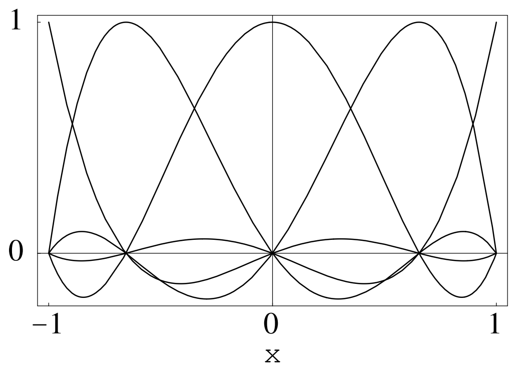


Fig. 4.28: Representative $p+1$ Lagrangian-interpolant shape functions in the element natural coordinates for a fourth-order LSFEs, where nodes are located at the Gauss-Lobatto-Legendre points.

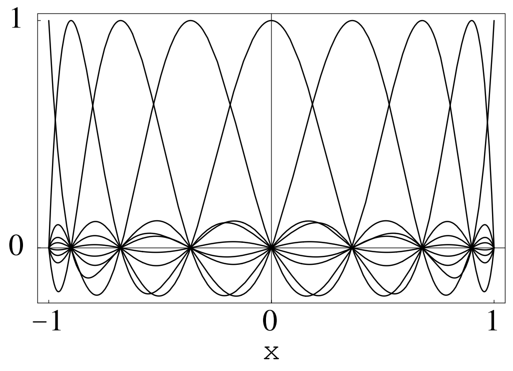


Fig. 4.29: Representative $p+1$ Lagrangian-interpolant shape functions in the element natural coordinates for a eighth-order LSFEs, where nodes are located at the Gauss-Lobatto-Legendre points.

Wiener-Milenković Rotation Parameter

In BeamDyn, the 3D rotations are represented as Wiener-Milenković parameters defined in the following equation:

$$\underline{c} = 4\tan\left(\frac{\phi}{4}\right)\bar{n} \tag{4.73}$$

where ϕ is the rotation angle and $\bar{n}$ is the unit vector of the rotation axis. It can be observed that the valid range for this parameter is $|\phi| < 2\pi$. The singularities existing at integer multiples of $\pm 2\pi$ can be removed by a rescaling operation at π as:

$$\underline{r} = \begin{cases} 4(q_0\underline{p} + p_0\underline{q} + \tilde{p}\underline{q})/(\Delta_1 + \Delta_2), & \text{if } \Delta_2 \geq 0 \\ -4(q_0\underline{p} + p_0\underline{q} + \tilde{p}\underline{q})/(\Delta_1 - \Delta_2), & \text{if } \Delta_2 < 0 \end{cases} \tag{4.74}$$

where $\underline{p}$, $\underline{q}$, and $\underline{r}$ are the vectorial parameterization of three finite rotations such that $\underline{\underline{R}}(\underline{r}) = \underline{\underline{R}}(\underline{p})\underline{\underline{R}}(\underline{q})$, $p_0 = 2 - \underline{p}^T\underline{p}/8$, $q_0 = 2 - \underline{q}^T\underline{q}/8$, $\Delta_1 = (4-p_0)(4-q_0)$, and $\Delta_2 = p_0 q_0 - \underline{p}^T\underline{q}$. It is noted that the rescaling operation could cause a discontinuity of the interpolated rotation field; therefore a more robust interpolation algorithm has been introduced in Section *Numerical Implementation with Legendre Spectral Finite Elements* where the rescaling-independent relative-rotation field is interpolated.

The rotation tensor expressed in terms of Wiener-Milenković parameters is

$$\underline{\underline{R}}(\underline{c}) = \frac{1}{(4-c_0)^2}\begin{bmatrix} c_0^2 + c_1^2 - c_2^2 - c_3^2 & 2(c_1c_2 - c_0c_3) & 2(c_1c_3 + c_0c_2) \\ 2(c_1c_2 + c_0c_3) & c_0^2 - c_1^2 + c_2^2 - c_3^2 & 2(c_2c_3 - c_0c_1) \\ 2(c_1c_3 - c_0c_2) & 2(c_2c_3 + c_0c_1) & c_0^2 - c_1^2 - c_2^2 + c_3^2 \end{bmatrix} \tag{4.75}$$

where $\underline{c} = [c_1 \;\; c_2 \;\; c_3]^T$ is the Wiener-Milenković parameter and $c_0 = 2 - \frac{1}{8}\underline{c}^T\underline{c}$. The relation between rotation tensor and direction cosine matrix (DCM) is

$$\underline{\underline{R}} = (\underline{\underline{DCM}})^T \tag{4.76}$$

Interested users are referred to [BEH08] and [WYS13] for more details on the rotation parameter and its implementation with GEBT.

Linearization Process

The nonlinear governing equations introduced in the previous section are solved by Newton-Raphson method, where a linearization process is needed. The linearization of each term in the governing equations are presented in this section.

According to [Bau10], the linearized governing equations in Eq. (4.65) are in the form of

$$\hat{\underline{\underline{M}}}\Delta\hat{\underline{a}} + \hat{\underline{\underline{G}}}\Delta\hat{\underline{v}} + \hat{\underline{\underline{K}}}\Delta\hat{\underline{q}} = \hat{\underline{F}}^{ext} - \hat{\underline{F}} \tag{4.77}$$

where the $\hat{\underline{\underline{M}}}$, $\hat{\underline{\underline{G}}}$, and $\hat{\underline{\underline{K}}}$ are the elemental mass, gyroscopic, and stiffness matrices, respectively; $\hat{\underline{F}}$ and $\hat{\underline{F}}^{ext}$ are the elemental forces and externally applied loads, respectively. They are defined for an element of length l along x_1 as follows

$$\begin{aligned} \hat{\underline{\underline{M}}} &= \int_0^l \underline{\underline{N}}^T \underline{\underline{\mathcal{M}}}\,\underline{\underline{N}} dx_1 \\ \hat{\underline{\underline{G}}} &= \int_0^l \underline{\underline{N}}^T \underline{\underline{\mathcal{G}}}^I\,\underline{\underline{N}} dx_1 \\ \hat{\underline{\underline{K}}} &= \int_0^l \left[\underline{\underline{N}}^T(\underline{\underline{\mathcal{K}}}^I + \underline{\underline{\mathcal{Q}}})\,\underline{\underline{N}} + \underline{\underline{N}}^T\underline{\underline{\mathcal{P}}}\,\underline{\underline{N}}' + \underline{\underline{N}}'^T\underline{\underline{\mathcal{C}}}\,\underline{\underline{N}}' + \underline{\underline{N}}'^T\underline{\underline{\mathcal{O}}}\,\underline{\underline{N}}\right] dx_1 \\ \hat{\underline{F}} &= \int_0^l (\underline{\underline{N}}^T\underline{\mathcal{F}}^I + \underline{\underline{N}}^T\underline{\mathcal{F}}^D + \underline{\underline{N}}'^T\underline{\mathcal{F}}^C) dx_1 \\ \hat{\underline{F}}^{ext} &= \int_0^l \underline{\underline{N}}^T\underline{\mathcal{F}}^{ext} dx_1 \end{aligned} \tag{4.78}$$

where $\underline{\mathcal{F}}^{ext}$ is the applied load vector. The new matrix notations in Eqs. (4.78) to are briefly introduced here. $\underline{\mathcal{F}}^C$ and $\underline{\mathcal{F}}^D$ are elastic forces obtained from Eq. (4.65) as

$$\begin{aligned}\underline{\mathcal{F}}^C &= \begin{Bmatrix}\underline{F}\\ \underline{M}\end{Bmatrix} = \underline{\underline{\mathcal{C}}}\begin{Bmatrix}\underline{\epsilon}\\ \underline{\kappa}\end{Bmatrix}\\ \underline{\mathcal{F}}^D &= \begin{bmatrix}\underline{\underline{0}} & \underline{\underline{0}}\\ (\tilde{x}_0' + \tilde{u}')^T & \underline{\underline{0}}\end{bmatrix}\underline{\mathcal{F}}^C \equiv \underline{\underline{\Upsilon}}\,\underline{\mathcal{F}}^C\end{aligned} \tag{4.79}$$

where $\underline{\underline{0}}$ denotes a 3×3 null matrix. The $\underline{\underline{\mathcal{G}}}^I$, $\underline{\underline{\mathcal{K}}}^I$, $\underline{\underline{\mathcal{O}}}$, $\underline{\underline{\mathcal{P}}}$, $\underline{\underline{\mathcal{Q}}}$, and $\underline{\mathcal{F}}^I$ in Eqs. (4.78) are defined as

$$\begin{aligned}\underline{\underline{\mathcal{G}}}^I &= \begin{bmatrix}\underline{\underline{0}} & (\widetilde{\tilde{\omega} m \underline{\eta}})^T + \tilde{\omega} m \tilde{\eta}^T\\ \underline{\underline{0}} & \tilde{\omega}\underline{\underline{\varrho}} - \widetilde{\underline{\underline{\varrho}}\underline{\omega}}\end{bmatrix}\\ \underline{\underline{\mathcal{K}}}^I &= \begin{bmatrix}\underline{\underline{0}} & \dot{\tilde{\omega}} m \tilde{\eta}^T + \tilde{\omega}\tilde{\omega} m \tilde{\eta}^T\\ \underline{\underline{0}} & \ddot{\tilde{u}} m \tilde{\eta} + \underline{\underline{\varrho}}\dot{\tilde{\omega}} - \widetilde{\underline{\underline{\varrho}}\dot{\underline{\omega}}} + \tilde{\omega}\underline{\underline{\varrho}}\tilde{\omega} - \tilde{\omega}\widetilde{\underline{\underline{\varrho}}\underline{\omega}}\end{bmatrix}\\ \underline{\underline{\mathcal{O}}} &= \begin{bmatrix}\underline{\underline{0}} & \underline{\underline{C}}_{11}\tilde{E}_1 - \tilde{F}\\ \underline{\underline{0}} & \underline{\underline{C}}_{21}\tilde{E}_1 - \tilde{M}\end{bmatrix}\\ \underline{\underline{\mathcal{P}}} &= \begin{bmatrix}\underline{\underline{0}} & \underline{\underline{0}}\\ \tilde{F} + (\underline{\underline{C}}_{11}\tilde{E}_1)^T & (\underline{\underline{C}}_{21}\tilde{E}_1)^T\end{bmatrix}\\ \underline{\underline{\mathcal{Q}}} &= \underline{\underline{\Upsilon}}\,\underline{\underline{\mathcal{O}}}\\ \underline{\mathcal{F}}^I &= \begin{Bmatrix}m\ddot{\underline{u}} + (\dot{\tilde{\omega}} + \tilde{\omega}\tilde{\omega}) m \underline{\eta}\\ m\tilde{\eta}\ddot{\underline{u}} + \underline{\underline{\varrho}}\dot{\underline{\omega}} + \tilde{\omega}\underline{\underline{\varrho}}\underline{\omega}\end{Bmatrix}\end{aligned} \tag{4.80}$$

where m is the mass density per unit length, $\underline{\eta}$ is the location of the sectional center of mass, $\underline{\underline{\varrho}}$ is the moment of inertia tensor, and the following notations were introduced to simplify the above expressions

$$\begin{aligned}\underline{E}_1 &= \underline{x}_0' + \underline{u}'\\ \underline{\underline{\mathcal{C}}} &= \begin{bmatrix}\underline{\underline{C}}_{11} & \underline{\underline{C}}_{12}\\ \underline{\underline{C}}_{21} & \underline{\underline{C}}_{22}\end{bmatrix}\end{aligned} \tag{4.81}$$

Damping Forces and Linearization

A viscous damping model has been implemented into BeamDyn to account for the structural damping effect. The damping force is defined as

$$\underline{f}_d = \underline{\underline{\mu}}\,\underline{\underline{\mathcal{C}}}\begin{Bmatrix}\dot{\underline{\epsilon}}\\ \dot{\underline{\kappa}}\end{Bmatrix} \tag{4.82}$$

where $\underline{\underline{\mu}}$ is a user-defined damping-coefficient diagonal matrix. The damping force can be recast in two separate parts, like $\underline{\mathcal{F}}^C$ and $\underline{\mathcal{F}}^D$ in the elastic force, as

$$\begin{aligned}\underline{\mathcal{F}}^C_d &= \begin{Bmatrix}\underline{F}_d\\ \underline{M}_d\end{Bmatrix}\\ \underline{\mathcal{F}}^D_d &= \begin{Bmatrix}0\\ (\tilde{x}_0' + \tilde{u}')^T \underline{F}_d\end{Bmatrix}\end{aligned} \tag{4.83}$$

The linearization of the structural damping forces are as follows:

$$\begin{aligned}\Delta\underline{\mathcal{F}}^C_d &= \underline{\underline{\mathcal{S}}}_d\begin{Bmatrix}\Delta\underline{u}'\\ \Delta\underline{c}'\end{Bmatrix} + \underline{\underline{\mathcal{O}}}_d\begin{Bmatrix}\Delta\underline{u}\\ \Delta\underline{c}\end{Bmatrix} + \underline{\underline{\mathcal{G}}}_d\begin{Bmatrix}\Delta\dot{\underline{u}}\\ \Delta\underline{\omega}\end{Bmatrix} + \underline{\underline{\mu}}\,\underline{\underline{\mathcal{C}}}\begin{Bmatrix}\Delta\dot{\underline{u}}'\\ \Delta\underline{\omega}'\end{Bmatrix}\\ \Delta\underline{\mathcal{F}}^D_d &= \underline{\underline{\mathcal{P}}}_d\begin{Bmatrix}\Delta\underline{u}'\\ \Delta\underline{c}'\end{Bmatrix} + \underline{\underline{\mathcal{Q}}}_d\begin{Bmatrix}\Delta\underline{u}\\ \Delta\underline{c}\end{Bmatrix} + \underline{\underline{\mathcal{X}}}_d\begin{Bmatrix}\Delta\dot{\underline{u}}\\ \Delta\underline{\omega}\end{Bmatrix} + \underline{\underline{\mathcal{Y}}}_d\begin{Bmatrix}\Delta\dot{\underline{u}}'\\ \Delta\underline{\omega}'\end{Bmatrix}\end{aligned} \tag{4.84}$$

where the newly introduced matrices are defined as

$$
\begin{aligned}
\underline{\underline{\mathcal{S}}}_d &= \underline{\underline{\mu \mathcal{C}}} \begin{bmatrix} \tilde{\omega}^T & \underline{\underline{0}} \\ \underline{\underline{0}} & \tilde{\omega}^T \end{bmatrix} \\
\underline{\underline{\mathcal{O}}}_d &= \begin{bmatrix} \underline{\underline{0}} & \underline{\underline{\mu C}}_{11}(\dot{\tilde{u}}' - \tilde{\omega}\tilde{E}_1) - \tilde{F}_d \\ \underline{\underline{0}} & \underline{\underline{\mu C}}_{21}(\dot{\tilde{u}}' - \tilde{\omega}\tilde{E}_1) - \tilde{M}_d \end{bmatrix} \\
\underline{\underline{\mathcal{G}}}_d &= \begin{bmatrix} \underline{\underline{0}} & \underline{\underline{C}}_{11}^T \underline{\underline{\mu}}^T \tilde{E}_1 \\ \underline{\underline{0}} & \underline{\underline{C}}_{12}^T \underline{\underline{\mu}}^T \tilde{E}_1 \end{bmatrix} \\
\underline{\underline{\mathcal{P}}}_d &= \begin{bmatrix} \underline{\underline{0}} & \underline{\underline{0}} \\ \tilde{F}_d + \tilde{E}_1^T \underline{\underline{\mu C}}_{11} \tilde{\omega}^T & \tilde{E}_1^T \underline{\underline{\mu C}}_{12} \tilde{\omega}^T \end{bmatrix} \\
\underline{\underline{\mathcal{Q}}}_d &= \begin{bmatrix} \underline{\underline{0}} & \underline{\underline{0}} \\ \underline{\underline{0}} & \tilde{E}_1^T \underline{\underline{O}}_{12} \end{bmatrix} \\
\underline{\underline{\mathcal{X}}}_d &= \begin{bmatrix} \underline{\underline{0}} & \underline{\underline{0}} \\ \underline{\underline{0}} & \tilde{E}_1^T \underline{\underline{G}}_{12} \end{bmatrix} \\
\underline{\underline{\mathcal{Y}}}_d &= \begin{bmatrix} \underline{\underline{0}} & \underline{\underline{0}} \\ \tilde{E}_1^T \underline{\underline{\mu C}}_{11} & \tilde{E}_1^T \underline{\underline{\mu C}}_{12} \end{bmatrix}
\end{aligned}
\tag{4.85}
$$

where $\underline{\underline{O}}_{12}$ and $\underline{\underline{G}}_{12}$ are the 3×3 sub matrices of $\underline{\underline{\mathcal{O}}}$ and $\underline{\underline{\mathcal{G}}}$ as $\underline{\underline{C}}_{12}$ in Eq. (4.81).

Convergence Criterion and Generalized-α Time Integrator

The system of nonlinear equations in Eqs. (4.65) are solved using the Newton-Raphson method with the linearized form in Eq. (4.77). In the present implementation, an energy-like stopping criterion has been chosen, which is calculated as

$$
|\Delta \mathbf{U}^{(i)T} \left({}^{t+\Delta t}\mathbf{R} - {}^{t+\Delta t}\mathbf{F}^{(i-1)} \right)| \leq |\epsilon_E \left(\Delta \mathbf{U}^{(1)T} \left({}^{t+\Delta t}\mathbf{R} - {}^{t}\mathbf{F} \right) \right)|
\tag{4.86}
$$

where $|\cdot|$ denotes the absolute value, $\Delta \mathbf{U}$ is the incremental displacement vector, $\mathbf{R}$ is the vector of externally applied nodal point loads, $\mathbf{F}$ is the vector of nodal point forces corresponding to the internal element stresses, and ϵ_E is the user-defined energy tolerance. The superscript on the left side of a variable denotes the time-step number (in a dynamic analysis), while the one on the right side denotes the Newton-Raphson iteration number. As pointed out by [BC80], this criterion provides a measure of when both the displacements and the forces are near their equilibrium values.

Time integration is performed using the generalized-α scheme in BeamDyn, which is an unconditionally stable (for linear systems), second-order accurate algorithm. The scheme allows for users to choose integration parameters that introduce high-frequency numerical dissipation. More details regarding the generalized-α method can be found in [CH93][Bau10].

Calculation of Reaction Loads

Since the root motion of the wind turbine blade, including displacements and rotations, translational and angular velocities, and translational and angular accelerates, are prescribed as inputs to BeamDyn either by the driver (in stand-alone mode) or by FAST glue code (in FAST-coupled mode), the reaction loads at the root are needed to satisfy equality of the governing equations. The reaction loads at the root are also the loads passing from blade to hub in a full turbine analysis.

The governing equations in Eq. (4.65) can be recast in a compact form

$$
\underline{\mathcal{F}}^I - \underline{\mathcal{F}}^{C\prime} + \underline{\mathcal{F}}^D = \underline{\mathcal{F}}^{ext}
\tag{4.87}
$$

with all the vectors defined in Section [sec:LinearProcess]. At the blade root, the governing equation is revised as

$$\underline{\mathcal{F}}^I - \underline{\mathcal{F}}^{C\prime} + \underline{\mathcal{F}}^D = \underline{\mathcal{F}}^{ext} + \underline{\mathcal{F}}^R \tag{4.88}$$

where $\underline{\mathcal{F}}^R = \begin{bmatrix} \underline{F}^R & \underline{M}^R \end{bmatrix}^T$ is the reaction force vector and it can be solved from Eq. (4.88) given that the motion fields are known at this point.

Calculation of Blade Loads

BeamDyn can also calculate the blade loads at each finite element node along the blade axis. The governing equation in Eq. (4.87) are recast as

$$\underline{\mathcal{F}}^A + \underline{\mathcal{F}}^V - \underline{\mathcal{F}}^{C\prime} + \underline{\mathcal{F}}^D = \underline{\mathcal{F}}^{ext} \tag{4.89}$$

where the inertial force vector $\underline{\mathcal{F}}^I$ is split into $\underline{\mathcal{F}}^A$ and $\underline{\mathcal{F}}^V$:

$$\begin{aligned} \underline{\mathcal{F}}^A &= \begin{Bmatrix} m\ddot{\underline{u}} + \dot{\tilde{\omega}} m \underline{\eta} \\ m\tilde{\eta}\ddot{\underline{u}} + \underline{\underline{\rho}}\dot{\underline{\omega}} \end{Bmatrix} \\ \underline{\mathcal{F}}^V &= \begin{Bmatrix} \tilde{\omega}\tilde{\omega} m \underline{\eta} \\ \tilde{\omega}\underline{\underline{\rho}}\underline{\omega} \end{Bmatrix} \end{aligned} \tag{4.90}$$

The blade loads are thus defined as

$$\underline{\mathcal{F}}^{BF} \equiv \underline{\mathcal{F}}^V - \underline{\mathcal{F}}^{C\prime} + \underline{\mathcal{F}}^D \tag{4.91}$$

We note that if structural damping is considered in the analysis, the $\underline{\mathcal{F}}^C_d$ and $\underline{\mathcal{F}}^D_d$ are incorporated into the internal elastic forces, $\underline{\mathcal{F}}^C$ and $\underline{\mathcal{F}}^D$, for calculation.

4.5.6 Future Work

The following list contains future work on BeamDyn software:

- Eliminating numerical problems in single precision.
- Implementing eigenvalue analysis.
- Improving input options for stand-alone version to make it more user-friendly.
- Implementing GEBT based on modal method for computational efficiency.
- Adding more options for blade cross-sectional properties inputs. For example, for general isotropic beams, engineering parameters including sectional offsets, material properties, etc will be used to generate the 6×6 matrices needed by BeamDyn.
- Writing a general guidance on modeling composite beam structures using BeamDyn, , for example, how to select a time step, how to select the model discretization, how to define the blade reference axis, where to get 6x6 mass/stiffness matrices, etc.
- Extending applications in FAST to other slender structures in the wind turbine system, for example, tower, mooring lines, and shaft.
- Developing a simplified form of GEBT with only rotational DOFs (bending, torsion) for computational efficiency.

4.5.7 Appendix

BeamDyn Input Files

In this appendix we describe the BeamDyn input-file structure and provide examples for the NREL 5MW Reference Wind Turbine.

OpenFAST+BeamDyn and stand-alone BeamDyn (static and dynamic) simulations all require two files:

1) BeamDyn primary input file `(NREL 5MW static example)`: This file includes information on the numerical-solution parameters (e.g., numerical damping, quadrature rules), and the geometric definition of the beam reference line via "members" and "key points". This file also specifies the "blade input file."

2) BeamDyn blade input file `(NREL 5MW example)`:

Stand-alone BeamDyn simulation also require a driver input file; we list here examples for static and dynamic simulations:

3a) BeamDyn driver for dynamic simulations `(NREL 5MW example)`: This file specifies the inputs for a single blade (e.g., forces, orientations, root velocity) and specifies the BeamDyn primary input file.

3b) BeamDyn driver for static simulations `(NREL 5MW example)`: Same as above but for static analysis.

BeamDyn List of Output Channels

This is a list of all possible output parameters for the BeamDyn module. The names are grouped by meaning, but can be ordered in the OUTPUTS section of the BeamDyn primary input file as the user sees fit. Nβ, refers to output node β, where β is a number in the range [1,9], corresponding to entry β in the `OutNd` list. When coupled to FAST, "$B\alpha$" is prefixed to each output name, where α is a number in the range [1,3], corresponding to the blade number. The outputs are expressed in one of the following three coordinate systems:

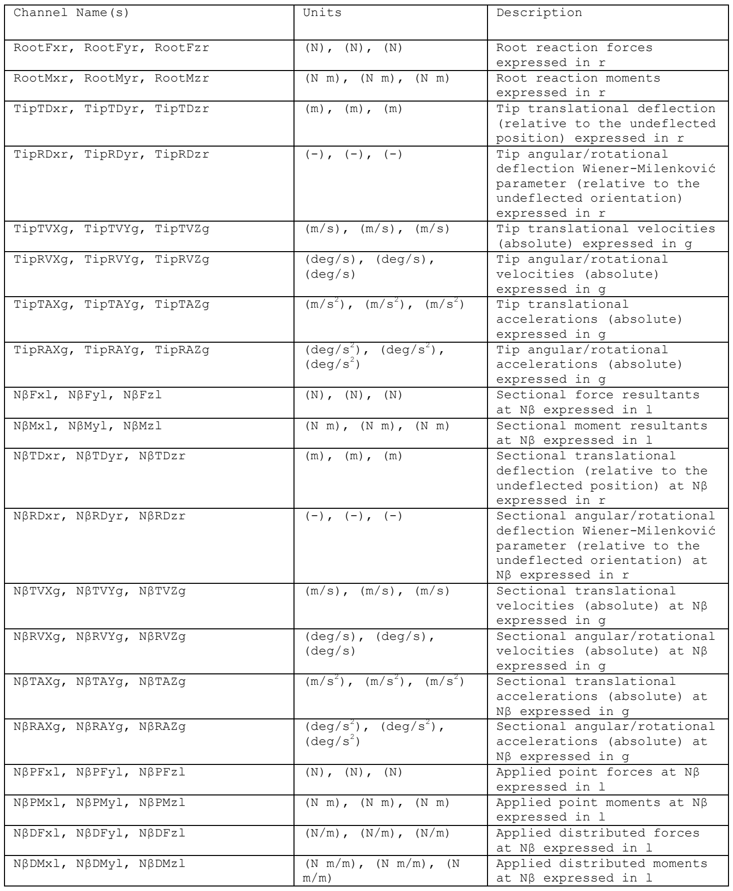
- **r**: a floating reference coordinate system fixed to the root of the moving beam; when coupled to FAST for blades, this is equivalent to the IEC blade (b) coordinate system.
- **l**: a floating coordinate system local to the deflected beam.
- **g**: the global inertial frame coordinate system; when coupled to FAST, this is equivalent to FAST's global inertial frame (i) coordinate system.

4.6 ElastoDyn Users Guide and Theory Manual

4.6.1 Input Files

The user configures the structural model parameters via a primary ElastoDyn input file, as well as separate input files for the tower and *other stuff that will be documented here later.*

No lines should be added or removed from the input files.



Channel Name(s)	Units	Description
RootFxr, RootFyr, RootFzr	(N), (N), (N)	Root reaction forces expressed in r
RootMxr, RootMyr, RootMzr	(N m), (N m), (N m)	Root reaction moments expressed in r
TipTDxr, TipTDyr, TipTDzr	(m), (m), (m)	Tip translational deflection (relative to the undeflected position) expressed in r
TipRDxr, TipRDyr, TipRDzr	(-), (-), (-)	Tip angular/rotational deflection Wiener-Milenković parameter (relative to the undeflected orientation) expressed in r
TipTVXg, TipTVYg, TipTVZg	(m/s), (m/s), (m/s)	Tip translational velocities (absolute) expressed in g
TipRVXg, TipRVYg, TipRVZg	(deg/s), (deg/s), (deg/s)	Tip angular/rotational velocities (absolute) expressed in g
TipTAXg, TipTAYg, TipTAZg	(m/s^2), (m/s^2), (m/s^2)	Tip translational accelerations (absolute) expressed in g
TipRAXg, TipRAYg, TipRAZg	(deg/s^2), (deg/s^2), (deg/s^2)	Tip angular/rotational accelerations (absolute) expressed in g
NβFxl, NβFyl, NβFzl	(N), (N), (N)	Sectional force resultants at Nβ expressed in l
NβMxl, NβMyl, NβMzl	(N m), (N m), (N m)	Sectional moment resultants at Nβ expressed in l
NβTDxr, NβTDyr, NβTDzr	(m), (m), (m)	Sectional translational deflection (relative to the undeflected position) at Nβ expressed in r
NβRDxr, NβRDyr, NβRDzr	(-), (-), (-)	Sectional angular/rotational deflection Wiener-Milenković parameter (relative to the undeflected orientation) at Nβ expressed in r
NβTVXg, NβTVYg, NβTVZg	(m/s), (m/s), (m/s)	Sectional translational velocities (absolute) at Nβ expressed in g
NβRVXg, NβRVYg, NβRVZg	(deg/s), (deg/s), (deg/s)	Sectional angular/rotational velocities (absolute) at Nβ expressed in g
NβTAXg, NβTAYg, NβTAZg	(m/s^2), (m/s^2), (m/s^2)	Sectional translational accelerations (absolute) at Nβ expressed in g
NβRAXg, NβRAYg, NβRAZg	(deg/s^2), (deg/s^2), (deg/s^2)	Sectional angular/rotational accelerations (absolute) at Nβ expressed in g
NβPFxl, NβPFyl, NβPFzl	(N), (N), (N)	Applied point forces at Nβ expressed in l
NβPMxl, NβPMyl, NβPMzl	(N m), (N m), (N m)	Applied point moments at Nβ expressed in l
NβDFxl, NβDFyl, NβDFzl	(N/m), (N/m), (N/m)	Applied distributed forces at Nβ expressed in l
NβDMxl, NβDMyl, NβDMzl	(N m/m), (N m/m), (N m/m)	Applied distributed moments at Nβ expressed in l

Fig. 4.30: BeamDyn Output Channel List

Units

ElastoDyn uses the SI system (kg, m, s, N). Angles are assumed to be in radians unless otherwise specified.

ElastoDyn Primary Input File

The primary ElastoDyn input file defines modeling options and geometries for the OpenFAST structure including the tower, nacelle, drivetrain, and blades (if BeamDyn is not used). It also sets the initial conditions for the structure.

Simulation Control

Set the **Echo** flag to TRUE if you wish to have ElastoDyn echo the contents of the ElastoDyn primary, airfoil, and blade input files (useful for debugging errors in the input files). The echo file has the naming convention of *OutRootFile.ED.ech*. **OutRootFile** is either specified in the I/O SETTINGS section of the driver input file when running ElastoDyn standalone, or by the OpenFAST program when running a coupled simulation.

Method

dT

Environmental Condition

gravity

Degrees of Freedom

FlapDOF1 - First flapwise blade mode DOF (flag)

FlapDOF2 - Second flapwise blade mode DOF (flag)

EdgeDOF - First edgewise blade mode DOF (flag)

TeetDOF - Rotor-teeter DOF (flag) [unused for 3 blades]

DrTrDOF - Drivetrain rotational-flexibility DOF (flag)

GenDOF - Generator DOF (flag)

YawDOF - Yaw DOF (flag)

TwFADOF1 - First fore-aft tower bending-mode DOF (flag)

TwFADOF2 - Second fore-aft tower bending-mode DOF (flag)

TwSSDOF1 - First side-to-side tower bending-mode DOF (flag)

TwSSDOF2 - Second side-to-side tower bending-mode DOF (flag)

PtfmSgDOF - Platform horizontal surge translation DOF (flag)

PtfmSwDOF - Platform horizontal sway translation DOF (flag)

PtfmHvDOF - Platform vertical heave translation DOF (flag)

PtfmRDOF - Platform roll tilt rotation DOF (flag)

PtfmPDOF - Platform pitch tilt rotation DOF (flag)

PtfmYDOF - Platform yaw rotation DOF (flag)

Initial Conditions

OoPDefl - Initial out-of-plane blade-tip displacement (meters)

IPDefl - Initial in-plane blade-tip deflection (meters)

BlPitch(1) - Blade 1 initial pitch (degrees)

BlPitch(2) - Blade 2 initial pitch (degrees)

BlPitch(3) - Blade 3 initial pitch (degrees) [unused for 2 blades]

TeetDefl - Initial or fixed teeter angle (degrees) [unused for 3 blades]

Azimuth - Initial azimuth angle for blade 1 (degrees)

RotSpeed - Initial or fixed rotor speed (rpm)

NacYaw - Initial or fixed nacelle-yaw angle (degrees)

TTDspFA - Initial fore-aft tower-top displacement (meters)

TTDspSS - Initial side-to-side tower-top displacement (meters)

PtfmSurge - Initial or fixed horizontal surge translational displacement of platform (meters)

PtfmSway - Initial or fixed horizontal sway translational displacement of platform (meters)

PtfmHeave - Initial or fixed vertical heave translational displacement of platform (meters)

PtfmRoll - Initial or fixed roll tilt rotational displacement of platform (degrees)

PtfmPitch - Initial or fixed pitch tilt rotational displacement of platform (degrees)

PtfmYaw - Initial or fixed yaw rotational displacement of platform (degrees)

Turbine Configuration

NumBl - Number of blades (-)

TipRad - The distance from the rotor apex to the blade tip (meters)

HubRad - The distance from the rotor apex to the blade root (meters)

PreCone(1) - Blade 1 cone angle (degrees)

PreCone(2) - Blade 2 cone angle (degrees)

PreCone(3) - Blade 3 cone angle (degrees) [unused for 2 blades]

HubCM - Distance from rotor apex to hub mass [positive downwind] (meters)

UndSling - Undersling length [distance from teeter pin to the rotor apex] (meters) [unused for 3 blades]

Delta3 - Delta-3 angle for teetering rotors (degrees) [unused for 3 blades]

AzimB1Up - Azimuth value to use for I/O when blade 1 points up (degrees)

OverHang - Distance from yaw axis to rotor apex [3 blades] or teeter pin [2 blades] (meters)

ShftGagL - Distance from rotor apex [3 blades] or teeter pin [2 blades] to shaft strain gages [positive for upwind rotors] (meters)

ShftTilt - Rotor shaft tilt angle (degrees)

NacCMxn - Downwind distance from the tower-top to the nacelle CM (meters)

NacCMyn - Lateral distance from the tower-top to the nacelle CM (meters)

NacCMzn - Vertical distance from the tower-top to the nacelle CM (meters)

NcIMUxn - Downwind distance from the tower-top to the nacelle IMU (meters)

NcIMUyn - Lateral distance from the tower-top to the nacelle IMU (meters)

NcIMUzn - Vertical distance from the tower-top to the nacelle IMU (meters)

Twr2Shft - Vertical distance from the tower-top to the rotor shaft (meters)

TowerHt - Height of tower above ground level [onshore] or MSL [offshore] (meters)

TowerBsHt - Height of tower base above ground level [onshore] or MSL [offshore] (meters)

PtfmCMxt - Downwind distance from the ground level [onshore] or MSL [offshore] to the platform CM (meters)

PtfmCMyt - Lateral distance from the ground level [onshore] or MSL [offshore] to the platform CM (meters)

PtfmCMzt - Vertical distance from the ground level [onshore] or MSL [offshore] to the platform CM (meters)

PtfmRefzt - Vertical distance from the ground level [onshore] or MSL [offshore] to the platform reference point (meters)

Mass and Inertia

TipMass(1) - Tip-brake mass, blade 1 (kg)

TipMass(2) - Tip-brake mass, blade 2 (kg)

TipMass(3) - Tip-brake mass, blade 3 (kg) [unused for 2 blades]

HubMass - Hub mass (kg)

HubIner - Hub inertia about rotor axis [3 blades] or teeter axis [2 blades] (kg m^2)

GenIner - Generator inertia about HSS (kg m^2)

NacMass - Nacelle mass (kg)

NacYIner - Nacelle inertia about yaw axis (kg m^2)

YawBrMass - Yaw bearing mass (kg)

PtfmMass - Platform mass (kg)

PtfmRIner - Platform inertia for roll tilt rotation about the platform CM (kg m^2)

PtfmPIner - Platform inertia for pitch tilt rotation about the platform CM (kg m^2)

PtfmYIner - Platform inertia for yaw rotation about the platform CM (kg m^2)

Blade

BldNodes - Number of blade nodes (per blade) used for analysis (-)

BldFile(1) - Name of file containing properties for blade 1 (quoted string)

BldFile(2) - Name of file containing properties for blade 2 (quoted string)

BldFile(3) - Name of file containing properties for blade 3 (quoted string) [unused for 2 blades]

Rotor-Teeter

TeetMod - Rotor-teeter spring/damper model {0: none, 1: standard, 2: user-defined from routine UserTeet} (switch) [unused for 3 blades]

TeetDmpP - Rotor-teeter damper position (degrees) [used only for 2 blades and when TeetMod=1]

TeetDmp - Rotor-teeter damping constant (N-m/(rad/s)) [used only for 2 blades and when TeetMod=1]

TeetCDmp - Rotor-teeter rate-independent Coulomb-damping moment (N-m) [used only for 2 blades and when Teet-Mod=1]

TeetSStP - Rotor-teeter soft-stop position (degrees) [used only for 2 blades and when TeetMod=1]

TeetHStP - Rotor-teeter hard-stop position (degrees) [used only for 2 blades and when TeetMod=1]

TeetSSSp - Rotor-teeter soft-stop linear-spring constant (N-m/rad) [used only for 2 blades and when TeetMod=1]

TeetHSSp - Rotor-teeter hard-stop linear-spring constant (N-m/rad) [used only for 2 blades and when TeetMod=1]

Drivetrain

GBoxEff - Gearbox efficiency (%)

GBRatio - Gearbox ratio (-)

DTTorSpr - Drivetrain torsional spring (N-m/rad)

DTTorDmp - Drivetrain torsional damper (N-m/(rad/s))

Furling

Furling - Read in additional model properties for furling turbine (flag) [must currently be FALSE)

FurlFile - Name of file containing furling properties (quoted string) [unused when Furling=False]

Tower

TwrNodes - Number of tower nodes used for analysis (-)

TwrFile - Name of file containing tower properties (quoted string)

Outputs

SumPrint [flag] Set this value to TRUE if you want ElastoDyn to generate a summary file with the name **OutFile-Root**.ED.sum*. **OutFileRoot** is specified by the OpenFAST program when running a coupled simulation.

OutFile [switch] is currently unused. The eventual purpose is to allow output from ElastoDyn to be written to a module output file (option 1), or the main OpenFAST output file (option 2), or both. At present this switch is ignored.

TabDelim [flag] is currently unused. Setting this to True will set the delimeter for text files to the tab character for the ElastoDyn module **OutFile**.

OutFmt [quoted string] is currently unused. ElastoDyn will use this string as the numerical format specifier for output of floating-point values in its local output specified by **OutFile**. The length of this string must not exceed 20 characters and must be enclosed in apostrophes or double quotes. You may not specify an empty string. To ensure that fixed-width column data align properly with the column titles, you should ensure that the width of the field is 10 characters.

Using an E, EN, or ES specifier will guarantee that you will never overflow the field because the number is too big, but such numbers are harder to read. Using an F specifier will give you numbers that are easier to read, but you may overflow the field. Please refer to any Fortran manual for details for format specifiers.

TStart [s] sets the start time for **OutFile**. This is currenlty unused.

DecFact [-] This parameter sets the decimation factor for output. ElastoDyn will output data to **OutFile** only once each DecFact integration time steps. For instance, a value of 5 will cause FAST to generate output only every fifth time step. This value must be an integer greater than zero.

NTwGages [-] The number of strain-gage locations along the tower indicates the number of input values on the next line. Valid values are integers from 0 to 5 (inclusive).

TwrGagNd [-] The virtual strain-gage locations along the tower are assigned to the tower analysis nodes specified on this line. Possible values are 1 to TwrNodes (inclusive), where 1 corresponds to the node closest to the tower base (but not at the base) and a value of TwrNodes corresponds to the node closest to the tower top. The exact elevations of each analysis node in the undeflected tower, relative to the base of the tower, are determined as follows:

Elev. of node J = TwrRBHt + (J – 1⁄2) • [(TowerHt + TwrDraft – TwrRBHt) / TwrNodes] (for J = 1,2,. . . ,TwrNodes)

You must enter at least NTwGages values on this line. If NTwGages is 0, this line will be skipped, but you must have a line taking up space in the input file. You can separate the values with combinations of tabs, spaces, and commas, but you may use only one comma between numbers.

NBlGages [-] specifies the number of strain-gague locations along the blade, and indicates the number of input values expected in **BldGagNd**. This is only used when the blade structure is modeled in ElastoDyn.

BldGagNd [-] specifies the virtual strain-gage locations along the blade that should be output. Possible values are 1 to **BldNodes** (inclusive), where 1 corresponds to the node closest to the blade root (but not at the root) and a value of BldNodes corresponds to the node closest to the blade tip. The node locations are specified by the ElastoDyn blade input files. You must enter at least NBlGages values on this line. If NBlGages is 0, this line will be skipped, but you must have a line taking up space in the input file. You can separate the values with combinations of tabs, spaces, and commas, but you may use only one comma between numbers. This is only used when the blade structure is modeled in ElastoDyn.

The **OutList** section controls output quantities generated by ElastoDyn. Enter one or more lines containing quoted strings that in turn contain one or more output parameter names. Separate output parameter names by any combination of commas, semicolons, spaces, and/or tabs. If you prefix a parameter name with a minus sign, "-", underscore, "_", or the characters "m" or "M", ElastoDyn will multiply the value for that channel by –1 before writing the data. The parameters are written in the order they are listed in the input file. ElastoDyn allows you to use multiple lines so that you can break your list into meaningful groups and so the lines can be shorter. You may enter comments after the closing quote on any of the lines. Entering a line with the string "END" at the beginning of the line or at the beginning of a quoted string found at the beginning of the line will cause ElastoDyn to quit scanning for more lines of channel names. Blade and tower node-related quantities are generated for the requested nodes identified through the **BldGagNd** and **TwrGagNd** lists above. If ElastoDyn encounters an unknown/invalid channel name, it warns the users but will remove the suspect channel from the output file. Please refer to the ElastoDyn tab in the Excel file `OutListParameters.xlsx` for a complete list of possible output parameters.

Nodal Outputs

In addition to the named outputs in Section 4.6.1 above, ElastoDyn allows for outputting the full set blade node motions and loads (tower nodes unavailable at present). Please refer to the ElastoDyn_Nodes tab in the Excel file `OutListParameters.xlsx` for a complete list of possible output parameters.

This section follows the *END* statement from normal Outputs section described above, and includes a separator description line followed by the following optinos.

BldNd_BladesOut specifies the number of blades to output. Possible values are 0 through the number of blades ElastoDyn is modeling. If the value is set to 1, only blade 1 will be output, and if the value is 2, blades 1 and 2 will be output.

BldNd_BlOutNd specifies which nodes to output. This is currently unused.

The **OutList** section controls the nodal output quantities generated by ElastoDyn. In this section, the user specifies the name of the channel family to output. The output name for each channel is then created internally by ElastoDyn by combining the blade number, node number, and channel family name. For example, if the user specifies **TDx** as the channel family name, the output channels will be named with the convention of **B**β**N###TDx** where β is the blade number, and **###** is the three digit node number.

Sample Nodal Outputs section

This sample includes the `END` statement from the regular outputs section.

```
END of input file (the word "END" must appear in the first 3 columns of this last␣
↪OutList line)
---------------------- NODE OUTPUTS --------------------------------------------
          3   BldNd_BladesOut   - Blades to output
         99   BldNd_BlOutNd    - Blade nodes on each blade (currently unused)
              OutList      - The next line(s) contains a list of output parameters. ␣
↪See OutListParameters.xlsx, ElastoDyn_Nodes tab for a listing of available output␣
↪channels, (-)
"ALx"     -  local flapwise acceleration (absolute) of node
"ALy"     - local flapwise acceleration (absolute) of node
"ALz"     - local flapwise acceleration (absolute) of node
"TDx"     - local flapwise (translational) deflection (relative to the undeflected␣
↪position) of node
"TDy"     - local edgewise (translational) deflection (relative to the undeflected␣
↪position) of node
"TDz"     - local axial (translational) deflection (relative to the undeflected␣
↪position) of node
"RDx"     - Local rotational displacement about x-axis (relative to undeflected)
"RDy"     - Local rotational displacement about y-axis (relative to undeflected)
"RDz"     - Local rotational displacement about z-axis (relative to undeflected)
"MLx"     - local edgewise moment at node
"MLy"     - local flapwise moment at node
"MLz"     - local pitching moment at node
"FLx"     - local flapwise shear force at node
"FLy"     - local edgewise shear force at node
"FLz"     - local axial force at node
"MLxNT"   - Edgewise moment in local coordinate system (initial structural twist␣
↪removed)
"MlyNT"   - Flapwise shear moment in local coordinate system (initial structural twist␣
↪removed)
"FLxNT"   - Flapwise shear force in local coordinate system (initial structural twist␣
↪removed)
```

(continues on next page)

(continued from previous page)

```
"FlyNT"  - Edgewise shear force in local coordinate system (initial structural twist␣
↪removed)
END of input file (the word "END" must appear in the first 3 columns of this last␣
↪OutList line)
------------------------------------------------------------------------------------
↪-
```

4.7 FAST v8 and the transition to OpenFAST

This page describes the transition from FAST v8, a computer-aided engineering tool for simulating the coupled dynamic response of wind turbines, to OpenFAST. OpenFAST was established by researchers at the National Renewable Energy Laboratory (NREL) in 2017, who were supported by the U.S. Department of Energy Wind Energy Technology Office (DOE-WETO).

4.7.1 FAST v8

FAST v8 is a computer-aided engineering tool for simulating the coupled dynamic response of wind turbines. FAST joins aerodynamics models, hydrodynamics models for offshore structures, control and electrical system (servo) dynamics models, and structural (elastic) dynamics models to enable coupled nonlinear aero-hydro-servo-elastic simulation in the time domain. The FAST tool enables the analysis of a range of wind turbine configurations, including two- or three-blade horizontal-axis rotor, pitch or stall regulation, rigid or teetering hub, upwind or downwind rotor, and lattice or tubular tower. The wind turbine can be modeled on land or offshore on fixed-bottom or floating substructures. FAST is based on advanced engineering models derived from fundamental laws, but with appropriate simplifications and assumptions, and supplemented where applicable with computational solutions and test data.

The aerodynamic models use wind-inflow data and solve for the rotor-wake effects and blade-element aerodynamic loads, including dynamic stall. The hydrodynamics models simulate the regular or irregular incident waves and currents and solve for the hydrostatic, radiation, diffraction, and viscous loads on the offshore substructure. The control and electrical system models simulate the controller logic, sensors, and actuators of the blade-pitch, generator-torque, nacelle-yaw, and other control devices, as well as the generator and power-converter components of the electrical drive. The structural-dynamics models apply the control and electrical system reactions, apply the aerodynamic and hydrodynamic loads, adds gravitational loads, and simulate the elasticity of the rotor, drivetrain, and support structure. Coupling between all models is achieved through a modular interface and coupler.

4.7.2 Transition to OpenFAST

The release of OpenFAST represents a transition to better support an open-source developer community across research laboratories, industry, and academia around FAST-based aero-hydro-servo-elastic engineering models of wind-turbines and wind-plants. OpenFAST aims to provide a solid software-engineering framework for FAST development including well documented source code, extensive automated regression and unit testing, and a robust multi-platform and compiler build system.

OpenFAST includes the following organizational changes relative to FAST v8.16:

- A new GitHub organization has been established at https://github.com/openfast
- The OpenFAST glue codes, modules, module drivers, and compiling tools are contained within a single repository: https://github.com/openfast/openfast
- The FAST program has been renamed OpenFAST (starting from OpenFAST v1.0.0)

- Version numbering has been updated for OpenFAST (starting from OpenFAST v1.0.0), e.g., OpenFAST-v1.0.0-123-gabcd1234-dirty, where:
- v1.0.0 is the major-minor-bugfix numbering system and corresponds to a tagged commit made by NREL on GitHub
- 123-g is the number of additional commits after the most recent tag for a build [the '-g' is for 'git']
- abcd1234 is the first 8 characters of the current commit hash
- dirty denotes that local changes have been made but not committed
- Because all modules are contained in the same repository, the version numbers of each module have been eliminated and now use the OpenFAST version number (starting from OpenFAST v1.0.0) though old documentation may still refer to old version numbers
- The OpenFAST regression test baseline solutions (formerly the Certification Tests or CertTest) reside in a standalone repository: https://github.com/openfast/r-test (starting from OpenFAST v1.0.0)
- Unit testing has been introduced at the subroutine level (starting with BeamDyn from OpenFAST v1.0.0).
- An online documentation system has been established to replace existing documentation of FAST v8: http://openfast.readthedocs.io/; during the transition to OpenFAST, most user-related documentation is still provided through the NWTC Information Portal, https://nwtc.nrel.gov
- Cross platform compiling is accomplished with CMake on macOS, Linux, and Cygwin (Windows) systems
- Visual Studio Projects (VS-Build) are provided for compiling OpenFAST on Windows (starting from OpenFAST v1.0.0), but the development team is working to automate the generation of Visual Studio build files via CMake in a future release
- GitHub Issues has been made the primary platform for developers to report and track bugs, request feature enhancements, and to ask questions related to the source code, compiling, and regression/unit testing; general user-related questions on OpenFAST theory and usage should still be handled through the forum at https://wind.nrel.gov/forum/wind
- A new API has been added that provides a high level interface to run OpenFAST through a C++ driver code helping to interface OpenFAST with external programs like CFD solvers written in C++ (starting in OpenFAST v1.0.0)

4.7.3 Release Notes for OpenFAST

This section outlines significant modifications to OpenFAST made with each tagged release.

v0.1.0 (April 2017)

Algorithmically, OpenFAST v0.1.0 is the release most closely related to FAST v8.16.

- Organizational changes:
  - A new GitHub organization has been established at https://github.com/openfast
  - The OpenFAST glue codes, modules, module drivers, and compiling tools are contained within a single repository: https://github.com/openfast/openfast
  - Cross platform compiling is accomplished with CMake on macOS, Linux, and Cygwin (Windows) systems
  - An online documentation system has been established to replace existing documentation of FAST v8: http://openfast.readthedocs.io/

  - GitHub Issues has been made the primary platform for developers to report and track bugs, request feature enhancements, and to ask questions related to the source code, compiling, and regression/unit testing; general user-related questions on OpenFAST theory and usage should still be handled through the forum at https://wind.nrel.gov/forum/wind

- The AeroDyn v15 aerodynamics module has been significantly updated. The blade-element/momentum theory (BEMT) solution algorithm has been improved as follows:
  - BEMT now functions for the case where the undisturbed velocity along the x-direction of the local blade coordinate system (Vx) is less than zero
  - BEMT no longer aborts when a valid value of the inflow angle (ϕ) cannot be found; in this case, the inflow angle is computed geometrically (without induction)
  - The inflow angle (ϕ) is now initialized on the first call instead of defaulting to using $\phi = 0$, giving better results during simulation start up
  - When hub- and/or tip-loss are enabled (HubLoss = True and/or TipLoss = True), tangential induction (a') is set to 0 instead of -1 at the root and/or tip, respectively (axial induction (a) is still set to 1 at the root and/or tip)
  - The BEMT solution has been made more efficient
  - In addition, several bugs in AeroDyn v15 have been fixed, including:
  - Fixed a bug whereby when hub- and/or tip-loss are enabled (HubLoss = True and/or TipLoss = True) along with the Pitt/Peters skewed-wake correction (SkewMod = 2), BEMT no longer modifies the induction factors at the hub and/or tip, respectively
  - Fixed a bug whereby the time series was affected after the linearization analysis with AeroDyn coupled to OpenFAST when frozen wake is enabled (FrozenWake = True)
- The BeamDyn finite-element blade structural-dynamics model has undergone an extensive cleanup of the source code. A bug in an off-diagonal term in the structural damping-induced stiffness (i.e., representing a change in the damping force with beam displacement) has been corrected.
- A new module for user-specified platform loading (ExtPtfm) has been introduced. ExtPtfm allows the user to specify 6x6 added mass, damping, and stiffness matrices, as well as a 6x1 load vector to define loads to be applied to ElastoDyn's tower base/platform, e.g., to support the modeling of substructures or foundations through a super-element representation (with super-element derived from external software). ExtPtfm also provides the user with a module to customize with more advanced platform applied loads. Module ExtPtfm can be enabled by setting CompSub to 2 in the FAST primary input file (a new option) and setting SubFile to the name of the file containing the platform matrices and load time history, but setting CompSub to 2 requires one to disable hydrodynamics (by setting CompHydro to 0). Please note that the introduction of option 2 for CompSub represents a minor input file change (the only input file change in OpenFAST v0.1.0), but the MATLAB conversion scripts have not yet been updated.
- In the ServoDyn control and electrical-system module, the units and sign of output parameter YawMom have been corrected
- In the InflowWind wind-inflow module, the ability to use TurbSim-generated tower wind data files in Bladed-style format was corrected
- Minor fixes were made to the error checking in ElastoDyn

v1.0.0 (September 2017)

- Organizational changes:
  - The FAST program has been renamed OpenFAST
  - Version numbering has been updated for OpenFAST (see Section 4.3.2 for details)
  - The OpenFAST regression test baseline solutions (formerly the Certification Tests or CertTest) reside in a standalone repository: https://github.com/openfast/r-test
  - Unit testing has been introduced at the subroutine level (starting with BeamDyn)
  - The online documentation (http://openfast.readthedocs.io/en/latest/index.html) has been extensively updated with additions for installation, testing, user (AeroDyn BeamDyn, transition from FAST v8, release notes), and developer guides, etc
  - The scripts for compiling OpenFAST using CMake on macOS, Linux, and Cygwin (Windows) systems have been updated, including the ability to compile in single precision and building with Spack
  - Visual Studio Projects (VS-Build) are provided for compiling OpenFAST on Windows
  - TurbSim has been included in the OpenFAST repository
- The AeroDyn aerodynamics module has been updated:
- Added a cavitation check for marine hydrokinetic (MHK) turbines. This includes the additions of new input parameters CavitCheck, Patm, Pvap, and FluidDepth in the AeroDyn primary input file, the addition of the Cpmin to the airfoil data files (required when CavitCheck = True), and new output channels for the minimum pressure coefficient, critical cavitation, and local cavitation numbers at the blade nodes. Please note that this input file changes represent the only input file change in OpenFAST v1.0.0, but the MATLAB conversion scripts have not yet been updated.
- Fixed a bug in the calculation of wind loads on the tower whereby the tower displacement was used in place of the tower velocity
- Tower strikes detected by the models to calculate the influence of the tower on the wind local to the blade are now treated as fatal errors instead of severe errors
- Fixed minor bugs in the unsteady airfoil aerodynamics model
- The BeamDyn finite-element blade structural-dynamics module has undergone additional changes:
- The source-code has further undergone clean up, including changing the internal coordinate system to match IEC (with the local z axis along the pitch axis)
- Trapezoidal points are now correctly defined by blade stations instead of key points
- The tip rotation outputs were corrected as per GitHub issue #10 (https://github.com/OpenFAST/openfast/issues/10)
- The BeamDyn driver has been fixed for cases involving spinning blades
- BeamDyn no longer produces numerical “spikes” in single precision, so, it is no longer necessary to compile OpenFAST in double precision when using BeamDyn
- The ElastoDyn structural-dynamics model was slightly updated:
- The precision on some module-level outputs used as input to the BeamDyn module were increased from single to double to avoid numerical “spikes” when running BeamDyn in single precision
- Minor fixes were made to the error checking
- The ServoDyn control and electrical system module was slightly updated:

- Fixed the values of the generator torque and electrical power sent from ServoDyn to Bladed-style DLL controllers as per GitHub issue # 40 (https://github.com/OpenFAST/openfast/issues/40)
- Minor fixes were made to the error checking
- The OpenFAST driver/glue code has been updated:
- Correction steps have been added to the OpenFAST driver during the first few time steps to address initialization problems with BeamDyn (even with NumCrctn = 0)
- Fixed a bug in the Line2-to-Point mapping of loads as per GitHub issue #8 (https://github.com/OpenFAST/openfast/issues/8). Previously, the augmented mesh was being formed using an incorrect projection, thus causing strange transfer of loads in certain cases. This could cause issues in the coupling between ElastoDyn and AeroDyn and/or in the coupling between HydroDyn and SubDyn
- Added an otherwise undocumented feature for writing binary output without compression to support the new regression testing. The new format is available by setting OutFileFmt to 0 in the FAST primary input file.
- A new API has been added that provides a high level interface to run OpenFAST through a C++ driver code. The primary purpose of the C++ API is to help interface OpenFAST to external programs like CFD solvers that are typically written in C++.
- The TurbSim wind-inflow turbulence preprocessor was updated:
- The API spectra was corrected
- Several minor bugs were fixed.

4.7.4 OpenFAST: Looking forward

Our goal is to continually improve OpenFAST documentation and to increase the coverage of automated unit and regression testing. In order to increase testing coverage and to maintain robust software, we will require that

- new modules be equipped by the module developer(s) with sufficient module-specific unit and regression testing along with appropriate OpenFAST regression tests;
- bug fixes include appropriate unit tests;
- new features/capabilities include appropriate unit and regression tests. We are in the process of better instrumenting the BeamDyn module with extensive testing as a demonstration of requirements for new modules.

For unit testing, we will employ the pFUnit framework (https://sourceforge.net/projects/pfunit).

For the time being OpenFAST provides project and solution files to support users developing and compiling using Visual Studio. However, the team is continually working to automate the generation of Visual Studio build files via CMake in future releases.

Please contact Michael.A.Sprague@NREL.gov with questions regarding the OpenFAST development plan.

4.8 C++ API Users Guide

The C++ API provides a high level API to run OpenFAST through a C++ gluecode. The primary purpose of the C++ API is to help interface OpenFAST to external programs like CFD solvers that are typically written in C++. The installation of C++ API is enabled via CMake by turning on the `BUILD_OPENFAST_CPP_API` flag.

A sample glue-code FAST_Prog.cpp is provided as a demonstration of the usage of the C++ API. The glue-code allows for the simulation of multiple turbines using OpenFAST in parallel over multiple processors. The glue-code takes a single input file named `cDriver.i` (`download`).

```yaml
# -*- mode: yaml -*-
#
# C++ glue-code for OpenFAST - Example input file
#

#Total number of turbines in the simulation
nTurbinesGlob: 3
#Enable debug outputs if set to true
debug: False
#The simulation will not run if dryRun is set to true
dryRun:  False
#Flag indicating whether the simulation starts from scratch or restart
simStart: init # init/trueRestart/restartDriverInitFAST
#Start time of the simulation
tStart:  0.0
#End time of the simulation. tEnd <= tMax
tEnd:    1.0
#Max time of the simulation
tMax:    4.0
#Time step for FAST. All turbines should have the same time step.
dtFAST:  0.00625
#Restart files will be written every so many time steps
nEveryCheckPoint: 160

Turbine0:
  #The position of the turbine base for actuator-line simulations
  turbine_base_pos: [ 0.0, 0.0, 0.0 ]
  #The number of actuator points along each blade for actuator-line simulations
  num_force_pts_blade: 0
  #The number of actuator points along the tower for actuator-line simulations.
  num_force_pts_tower: 0
  #The checkpoint file for this turbine when restarting a simulation
  restart_filename: "banana"
  #The FAST input file for this turbine
  FAST_input_filename: "t1_Test05.fst"
  #A unique turbine id for each turbine
  turb_id:  1

Turbine1:
  turbine_base_pos: [ 0.0, 0.0, 0.0 ]
  num_force_pts_blade: 0
  num_force_pts_tower: 0
  restart_filename: "banana"
  FAST_input_filename: "t2_Test05.fst"
  turb_id:  2

Turbine2:
  turbine_base_pos: [ 0.0, 0.0, 0.0 ]
  num_force_pts_blade: 0
  num_force_pts_tower: 0
  restart_filename: "banana"
  FAST_input_filename: "t3_Test05.fst"
  turb_id:  3
```

4.8.1 Command line invocation

```
mpiexec -np <N> openfastcpp
```

4.8.2 Common input file options

nTurbinesGlob
Total number of turbines in the simulation. The input file must contain a number of turbine specific sections (*Turbine0*, *Turbine1*, . . . , *Turbine(n-1)*) that is consistent with *nTurbinesGlob*.

debug
Enable debug outputs if set to true

dryRun
The simulation will not run if dryRun is set to true. However, the simulation will read the input files, allocate turbines to processors and prepare to run the individual turbine instances. This flag is useful to test the setup of the simulation before running it.

simStart
Flag indicating whether the simulation starts from scratch or restart. `simStart` takes on one of three values:

- `init` - Use this option when starting a simulation from *t=0s*.
- `trueRestart` - While OpenFAST allows for restart of a turbine simulation, external components like the Bladed style controller may not. Use this option when all components of the simulation are known to restart.
- `restartDriverInitFAST` - When the `restartDriverInitFAST` option is selected, the individual turbine models start from *t=0s* and run up to the specified restart time using the inflow data stored at the actuator nodes from a hdf5 file. The C++ API stores the inflow data at the actuator nodes in a hdf5 file at every OpenFAST time step and then reads it back when using this restart option. This restart option is especially useful when the glue code is a CFD solver.

tStart
Start time of the simulation

tEnd
End time of the simulation. tEnd <= tMax

tMax
Max time of the simulation

dtFAST
Time step for FAST. All turbines should have the same time step.

nEveryCheckPoint
Restart files will be written every so many time steps

4.8.3 Turbine specific input options

`turbine_base_pos`
: The position of the turbine base for actuator-line simulations

`num_force_pts_blade`
: The number of actuator points along each blade for actuator-line simulations

`num_force_pts_tower`
: The number of actuator points along the tower for actuator-line simulations.

`restart_filename`
: The checkpoint file for this turbine when restarting a simulation

`FAST_input_filename`
: The FAST input file for this turbine

`turb_id`
: A unique turbine id for each turbine

CHAPTER

FIVE

DEVELOPER DOCUMENTATION

Our goal as developers of OpenFAST is to ensure that it is well tested, well documented, and self-sustaining software. To that end, we continually work to improve the documentation and test coverage along with feature additions and improvements. This section of the documentation outlines the processes and procedures we have established for external developers to work with the NREL OpenFAST team on code development.

If you'd like to help with general OpenFAST development or work on a particular feature, then first install OpenFAST following the *installation instructions* for your machine. Next, verify that your installation is valid by running the test suite following the *testing instructions*. While OpenFAST is compiling, we encourage reading through the *Development Philosophy* section to understand the general workflow for individual and coordinated development. Finally, be sure to review the *GitHub workflow* to avoid any merge or code conflicts.

With development happening in parallel between NREL, industry partners, and universities, NREL relies on GitHub to coordinate efforts:

- GitHub Issues is the place to ask usage or development questions, report bugs, and suggest code enhancements
- GitHub Pull Requests is the place for engaging with the OpenFAST team to have your new code merged into the main repository.

For other questions regarding OpenFAST, please contact Mike Sprague.

Tip: The following sections provide valuable guidance on workflow and development tips which make the process more efficient and effective:

- *Working with OpenFAST on GitHub*
- *Code Style*
- *Debugging OpenFAST*

5.1 API Reference

Some subroutines and derived types throughout the source code have in-source documentation which is compiled with Doxygen. Though this portion of the documentation is always under development, the existing API reference can be found in the following pages:

- Main Page
- Index of Types
- Source Files

5.2 Development Philosophy

OpenFAST is intended to be a self sustaining community developed software. A couple of tenets of this goal are that the code should be reasonably straightforward to comprehend and manageable to improve. With that in mind, we expect that new capabilities will include adequate testing and documentation.

We have the following guidance for developers:

- When fixing a bug, first introduce a unit test that exposes the bug, fix the bug, and submit a Pull Request. See *Testing OpenFAST* and *Working with OpenFAST on GitHub* for more information.
- When adding a new feature, create appropriate automated unit and regression tests as described in *Testing OpenFAST*. The objective is to create a GitHub pull request that provides adequate verification and validation so that the NREL OpenFAST developer team can merge the pull request with confidence that the new feature is "correct" and supports our goal of self-sustaining software. See *Pull Requests* for more information on submitting a pull request.
- If a code modification affects regression test results in an expected manner, work with the NREL OpenFAST developer team to upgrade the regression test suite via a GitHub issue or pull request at the openfast/r-test repository.

5.3 Development Guidelines

The following sections provide extended guidance on how to develop source code, interacting with the NREL OpenFAST team and other community contributors, and generally debugging and building out features.

5.3.1 Working with OpenFAST on GitHub

The majority of the collaboration and development for OpenFAST takes place on the GitHub repository. There, issues and pull requests are discussed and new versions are released. It is the best mechanism for engaging with the NREL OpenFAST team and other developers throughout the OpenFAST community.

Issues and work assignment

Issues should be opened with proper documentation and data to fully describe the problem or feature gap. It is here that communication and coordination should happen regarding ongoing work for new development, and developers should make clear any intention to complete a task.

Pull Requests

When a code modification is ready for review, a pull request should be submitted along with all appropriate documentation and tests. An NREL OpenFAST team member will assign a reviewer and work with the developer to have the code merged into the main repository.

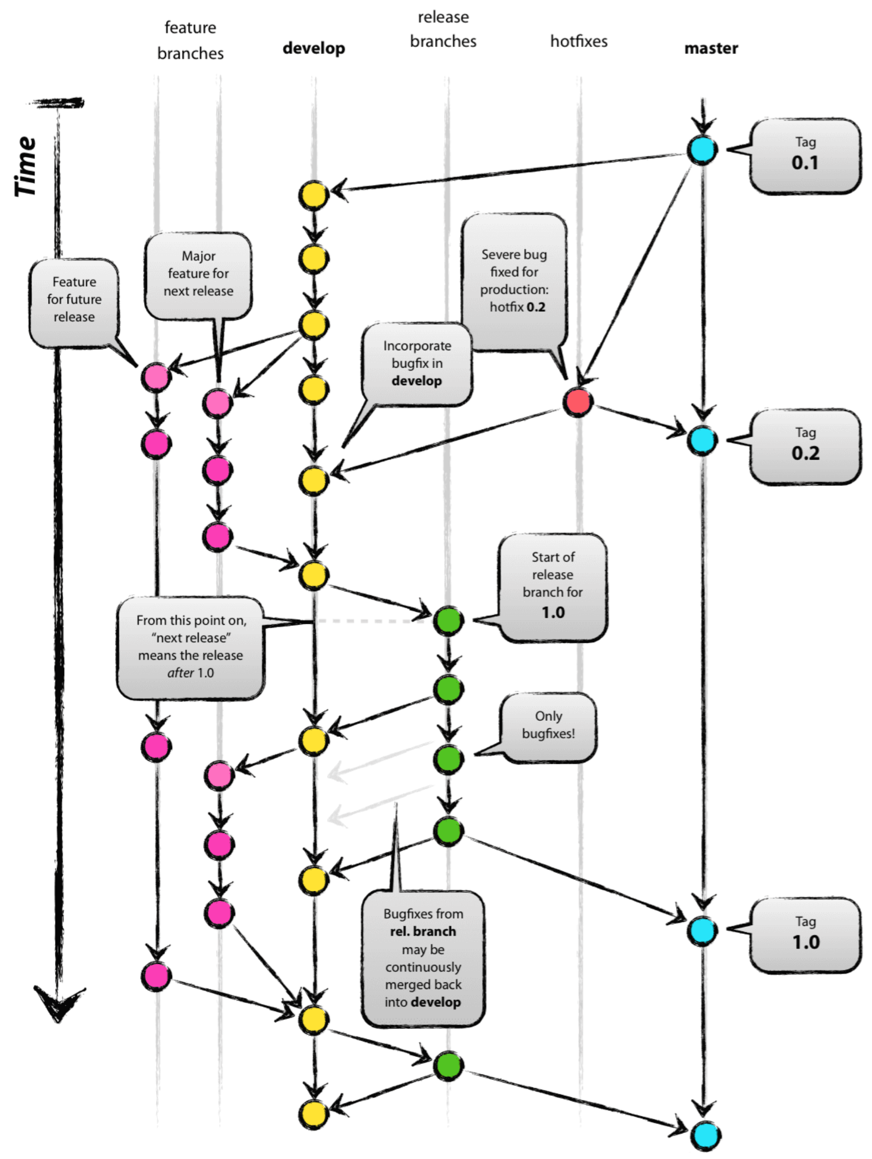
New pull requests should contain the following:

- A description of the need for modifications
  - If the pull request fixes a bug, the accompanying GitHub issue should be referenced
- A summary of the work implemented
- Regression test results

  - If all tests pass, the summary print out should be provided
  - If any tests fail, an explanation of the failing cases and supporting data like plots should be included
- Updated unit tests, if applicable
- Updated documentation in applicable sections ready for compilation and deployment to readthedocs.

Git workflow and interacting with the main repository

OpenFAST development should follow "Git Flow" when interacting with the github repository. Git Flow is a well-defined and widely adopted workflow for using git that outlines safe methods of pushing and pulling commits to a shared repository. Maintaining Git Flow is critical to prevent remote changes from blocking your local development. This workflow is detailed nicely here and the chart below provides a high level perspective.


feature branches
develop
release branches
hotfixes
master
Time
Tag 0.1
Feature for future release
Major feature for next release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Bugfixes from rel. branch may be continuously merged back into develop
Tag 1.0

Reference: http://nvie.com/posts/a-successful-git-branching-model

OpenFAST Specific Git Flow

It is important to consider how your current work will be affected by other developer's commits and how your commits will affect other developers. On public branches, avoid using git rebase and never force push.

In OpenFAST development, the typical workflow follows this procedure:

1. Fork the OpenFAST repository on GitHub
2. Clone your new fork

```
git clone https://github.com/<youruser>/OpenFAST
```

3. Add OpenFAST/OpenFAST as a remote named `upstream`

```
# This adds the remote
git remote add upstream https://github.com/OpenFAST/OpenFAST

# This downloads all the info in the remote, but it doesnt change
# the local source code
git fetch --all
```

4. Create a feature branch for active development starting from the OpenFAST `dev` branch and check it out

```
git branch feature/a_great_feature upstream/dev
git checkout feature/a_great_feature
```

5. Add new development on `feature/a_great_feature`

```
git add a_file.f90
git commit -m "A message"
git push origin feature/a_great_feature
```

6. Update your feature branch with `upstream`

```
git pull upstream dev
git push origin feature/a_great_feature
```

7. Open a new pull request to merge `<youruser>/OpenFAST/feature/a_great_feature` into `OpenFAST/OpenFAST/dev`

5.3.2 Code Style

OpenFAST and its underlying modules are mostly written in Fortran adhering to the 2003 standard, but modules can be written in C or C++. The NWTC Programmer's Handbook is the definitive reference for all questions related to working with the FAST Framework and adding code to OpenFAST.

Generally, code should be written such that it is straightforward to read. Syntactic sugar or brevity should not detract from readability. The exception to this is in situations where performance requires poorly readable code. Here, comment blocks should be used to describe what is not readily apparent in the code. Indentation is typically three spaces and no tabs.

5.3.3 Developing Documentation

OpenFAST documentation is hosted on readthedocs. It is automatically generated through the `readthedocs` build system from both the `master` and `dev` branches whenever new commits are added. This documentation uses the restructured text markup language.

Building this documentation locally

The documentation is compiled with Sphinx, which is a Python based tool. Install it and the other required Python packages listed in `openfast/docs/requirements.txt` with `pip` or another Python package manager.

These additional packages are optional and are not included in the requirements file:

- Doxygen
- Doxylink
- Graphviz
- LaTeX

Doxygen and Graphviz can be installed directly from their website or with a package manager like `brew`, `yum`, or `apt`.

The result of building the documentation locally will be a set of HTML files and their accompanying required files. The main HTML file will exist `openfast/build/docs/html/index.html`. This file can be opened with any browser to view and navigate the locally-generated documentation as if it were any other web site.

Pure python build

If CMake and Make are not available on your system, the documentation can be generated directly with *sphinx*.

Note: This method does not generate the API documentation through Doxygen.

First, align your directory structure to the standard OpenFAST build by creating a directory at `openfast/build`. Then, move into `openfast/build` and run this command:

```
# sphinx-build -b <builder-name> <source-directory> <output-directory>
sphinx-build -b html ../docs ./docs/html
```

If this completes successfully, an html file will be created at `build/docs/html/index.html` which can be opened with any web browser.

Building with CMake and Make

In the OpenFAST directory, create a `build` directory and move into it. Then, run CMake with this flag: `-DBUILD_DOCUMENTATION=ON`. CMake will configure the build system with the necessary files for building the documentation.

Next, run the command to compile the docs:

```
make docs
```

This will first build the Doxygen API documentation and then the Sphinx documentation. If this completes successfully, a html file will be created at `build/docs/html/index.html` which can be opened with any web browser.

The full procedure for configuring and building the documentation is:

```
mkdir build
cd build
cmake .. -DBUILD_DOCUMENTATION=ON
make docs
```

If any modifications are made to the source files in `openfast/docs/source`, you can simply update the html files by executing the `make` command again.

The table below lists make-targets related to the documentation.

Target	Command	Output location
Full docs	make docs	openfast/build/docs/html/index.html
Full docs	make sphinx	openfast/build/docs/html/index.html
Doxygen API Reference	make doxygen	
HTML only	make sphinx-html	openfast/build/docs/html/index.html
PDF only	make sphinx-pdf	openfast/build/docs/latex/Openfast.pdf

Adding documentation

Coming soon. Feel like contributing? Start here!

5.3.4 Types Files and the OpenFAST Registry

Being a modern software project, OpenFAST has a complex system of custom data types. In Fortran, these are known as “derived data types.” Each module contains a unique collection of derived types which may add on to but must comply with the OpenFAST Framework. The module types are generally auto-generated by an included program called *OpenFAST Registry*. The OpenFAST Registry is written in C and adapted from a similar utility used in WRF. Visit the OpenFAST Registry README for more information.

The OpenFAST Registry requires an input file to describe the necessary types for a given module. Generally, all module use a similar naming convention, **<Module>_Registry.txt**, and resulting Fortran code will be in a file called **<Module>_Types.f90**. For example, the AeroDyn OpenFAST Registry input file is located at `openfast/modules/aerodyn/src/AeroDyn_Registry.txt` and the resulting auto-generated Fortran source code is at `openfast/modules/aerodyn/src/AeroDyn_Types.f90`.

Since the types-modules are autogenerated, any changes to the data types directly should be expressed in the OpenFAST Registry input files so that the changes are not subsequently overwritten.

Compiling the OpenFAST Registry

The OpenFAST Registry is included in the OpenFAST build system through CMake. However, the default is to **not** compile the OpenFAST Registry executable and instead use the types modules that are included in *git* while compiling OpenFAST. To include the OpenFAST Registry in the build process and compile the Registry program, configure CMake with the `GENERATE_TYPES` flag:

```
cmake .. -DGENERATE_TYPES=ON
```

With `GENERATE_TYPES` enabled, CMake will configure the *openfast-registry* target to compile as a dependency of all other targets. The OpenFAST Registry executable will be found in `openfast/build/modules/openfast-registry/openfast-registry`.

Regenerating a types module

With the `GENERATE_TYPES` flag enabled, an additional step will be added to modules that are configured can make use of the OpenFAST Registry. The additional step will execute the OpenFAST Registry and regenerate the types module overwriting the existing modules. Any changes to the types module will be evident in *git*. For modules where the registry input file has not changed, the resulting types module will not change. However, for registry input files that have been modified, the output types module will be recompiled.

Adding a new types module

The process for adding a new types module follows *Regenerating a types module* closely. Here, an additional step is required to configure CMake to execute the Registry on the new input file and include the resulting types module in the compile step.

First, a new OpenFAST Registry input file must be created. Then, it must be configured to pass through the Registry in the corresponding module's `CMakeLists.txt`:

```
# This is the control statement for allowing the Registry to execute
if (GENERATE_TYPES)

    # Here is the CMake wrapper-function to execute the Registry
    # syntax: generate_f90_types(<Registry input file> <output file location>)
    generate_f90_types(src/AeroDyn_Registry.txt ${CMAKE_CURRENT_LIST_DIR}/src/AeroDyn_
↪Types.f90)
    generate_f90_types(src/New_Registry.txt ${CMAKE_CURRENT_LIST_DIR}/src/New_Types.
↪f90)
endif()
```

Finally, the resulting types module must be added to the source files for the corresponding module:

```
# AeroDyn lib
set(AD_LIBS_SOURCES
    src/AeroDyn.f90
    src/AeroDyn_IO.f90
    src/AirfoilInfo.f90
    src/BEMT.f90
    src/DBEMT.f90
    src/BEMTUncoupled.f90
    src/UnsteadyAero.f90
    src/fmin_fcn.f90
    src/mod_root1dim.f90
    src/AeroDyn_Types.f90
    src/AirfoilInfo_Types.f90
    src/BEMT_Types.f90
    src/DBEMT_Types.f90
    src/UnsteadyAero_Types.f90

    # Add the new types module here
    src/New_Types.f90
)
```

With CMake properly configured, a message will display during the build process indicating that the OpenFAST Registry is executing:

```
[ 64%] Generating ../../../modules/aerodyn/src/New_Types.f90

----- FAST Registry --------------
------------------------------------------------
input file: /Users/rmudafor/Development/openfast/modules/aerodyn/src/New_Registry.txt
# more build process output will follow
```

And finally there should be an indication that the resulting types module is compiled:

```
Scanning dependencies of target aerodynlib
[ 70%] Building Fortran object modules/aerodyn/CMakeFiles/aerodynlib.dir/src/New_
↪Types.f90.o
```

5.3.5 Debugging OpenFAST

Being a Fortran project, OpenFAST can be challenging to debug and the process is unique for each system and environment. Keep in mind that some OpenFAST cases can be quite large in their memory footprint and may take a long time to reach the point of interest in the code. Choosing a test case carefully could save a significant amount time.

It may by helpful to write a small fortran program to verify that all debugging tools are set up properly before diving in to OpenFAST. Be sure to simulate a bug by doing something like accessing an array element that is not allocated and verify that you can catch the bug with a given set of tools.

Note: A requirement for all systems is to compile OpenFAST in **debug** mode.

Debugging on Windows

Windows developers using Intel tools can use Visual Studio solution included in the OpenFAST repository for debugging. This is a straightforward process with lots of support from Intel.

Otherwise, Windows developers compiling in Unix-style environments should proceed to *Debugging on Linux and macOS*.

Debugging on Linux and macOS

First, compile OpenFAST in debug mode by setting `CMAKE_BUILD_TYPE` to "Debug". This can be done on the command line with:

```
cmake .. -D CMAKE_BUILD_TYPE=Debug
```

or by using `ccmake` to open the command line cmake gui to change it.

The GNU debugger, `gdb`, works well for debugging compiled code. It has a comprehensive command line interface which enables developers to add breakpoints and inspect variables.

Driving the debugger through an IDE can make inspecting the code much more efficient. One IDE known to work well is Visual Studio Code with the Native Debug extension. You can set up a launch configuration so that you can debug a particular OpenFAST case through the IDE. To do this, open the launch configuration and add a block similar to this:

```
{
    "name": "AOC_WSt",
    "type": "gdb",
    "request": "launch",
    "printCalls": false,
    "showDevDebugOutput": false,
    "valuesFormatting": "prettyPrinters",
    "gdbpath": "gdb",
    "target": "${workspaceRoot}/build/glue-codes/openfast/openfast",
    "cwd": "${workspaceRoot}/build/reg_tests/glue-codes/openfast/AOC_WSt/",
    "arguments": "${workspaceRoot}/build/reg_tests/glue-codes/openfast/AOC_WSt/AOC_
↪WSt.fst",
}
```

macOS-specific configuration

GDB on macOS needs some configuration before the system allows it to take over a process. It is recommended that `gdb` be installed with homebrew

```
brew info gdb
brew install gdb
```

After that completes, be sure to follow the caveats to finish the installation. For `gdb 8.2.1`, it looks like this:

```
==> Caveats
gdb requires special privileges to access Mach ports.
You will need to codesign the binary. For instructions, see:

https://sourceware.org/gdb/wiki/BuildingOnDarwin

On 10.12 (Sierra) or later with SIP, you need to run this:

echo "set startup-with-shell off" >> ~/.gdbinit
```

For Native Debug on macOS, you have to sort of hack the extension to allow breakpoints in fortran files by adding this line to `.vscode/settings.json`:

```
{
    "debug.allowBreakpointsEverywhere": true
}
```

5.3.6 Performance-Profiling and Optimization

The OpenFAST team has been engaged in performance-profiling and optimization work in an effort to improve the time-to-solution performance for the most computationally expensive use cases. This work is supported by Intel® through its designation of NREL as an Intel® Parallel Computing Center (IPCC).

After initial profiling and hotspot analysis, specific subroutines in the physics modules of OpenFAST were targeted for optimization. Among other takeaways, it was learned that the memory alignment of the derived data types could yield a significant increase in performance. Ultimately, tuning the Intel® tools to perform best on NREL's hardware and adding high level multithreading yielded a maximum 3.8x time-to-solution improvement for one of the benchmark cases.

Approach

The general mechanisms identified for performance improvements in OpenFAST are:

- Intel® compiler suite and Intel® Math Kernel Library (Intel® MKL)
- Algorithmic improvements
- Memory-access optimization enabling more efficient cache usage
- Data type alignment allowing for SIMD vectorization
- Multithreading with OpenMP

To establish a path forward with any of these options, OpenFAST was first profiled with Intel® VTune™ Amplifier which provides a clear breakdown of time spent in the simulation. Then, the optimization report generated from the Intel® Fortran compiler was analyzed to determine area which were not autovectorized. Finally, Intel® Advisor was used to highlight areas of the code which the compiler identified as potentially improved with multithreading.

Test cases

Two OpenFAST test cases have been chosen to provide meaningful and realistic timing benchmarks. In addition to real-world turbine and atmospheric models, these cases are computationally expensive and expose the areas where performance improvements would make a difference.

5MW_Land_BD_DLL_WTurb

Download files here.

The physics modules used in this case are:

- BeamDyn
- InflowWind
- AeroDyn 15
- ServoDyn

This is a land based NREL 5-MW turbine simulation using BeamDyn as the structural module. It simulates 20 seconds with a time step size of 0.001 seconds and executes in 3m 55s on NREL's Peregrine supercomputer.

5MW_OC4Jckt_DLL_WTurb_WavesIrr_MGrowth

Download files here.

This is an offshore, fixed-bottom NREL 5-MW turbine simulation with the majority of the computational expense occurring in the HydroDyn wave-dynamics calculation.

The physics modules used in this case are:

- ElastoDyn
- InflowWind
- AeroDyn 15
- ServoDyn
- HydroDyn

- SubDyn

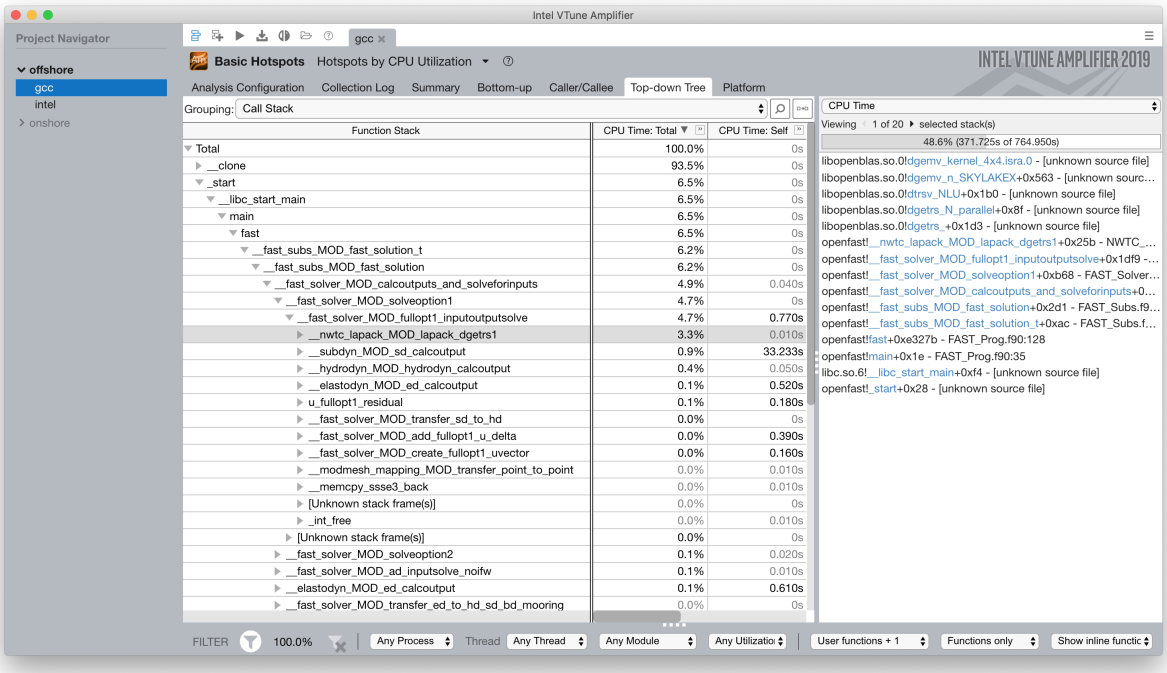
It simulates 60 seconds with a time step size of 0.01 seconds and executes in 20m 27s on NREL's Peregrine supercomputer.

Profiling

The OpenFAST test cases were profiled with Intel® VTune™ Amplifier to identify performance hotspots. Being that the two test cases exercise difference portions of the OpenFAST software, different hotspots were identified. In all cases and environment settings, the majority of the CPU time was spent in *fast_solution* loop which is a high-level subroutine that coordinates the solution calculation from each physics module.

LAPACK

In the offshore case, the LAPACK usage was identified as a performance load. Within the *fast_solution* loop, the calls to the LAPACK function *dgetrs* consume 3.3% of the total CPU time.



Grouping: Call Stack

Function Stack	CPU Time: Total	CPU Time: Self
Total	100.0%	0s
__clone	93.5%	0s
_start	6.5%	0s
__libc_start_main	6.5%	0s
main	6.5%	0s
fast	6.5%	0s
__fast_subs_MOD_fast_solution_t	6.2%	0s
__fast_subs_MOD_fast_solution	6.2%	0s
__fast_solver_MOD_calcoutputs_and_solveforinputs	4.9%	0.040s
__fast_solver_MOD_solveoption1	4.7%	0s
__fast_solver_MOD_fullopt1_inputoutputsolve	4.7%	0.770s
__nwtc_lapack_MOD_lapack_dgetrs1	3.3%	0.010s
__subdyn_MOD_sd_calcoutput	0.9%	33.233s
__hydrodyn_MOD_hydrodyn_calcoutput	0.4%	0.050s
__elastodyn_MOD_ed_calcoutput	0.1%	0.520s
u_fullopt1_residual	0.1%	0.180s
__fast_solver_MOD_transfer_sd_to_hd	0.0%	0s
__fast_solver_MOD_add_fullopt1_u_delta	0.0%	0.390s
__fast_solver_MOD_create_fullopt1_uvector	0.0%	0.160s
__modmesh_mapping_MOD_transfer_point_to_point	0.0%	0.010s
__memcpy_ssse3_back	0.0%	0.010s
[Unknown stack frame(s)]	0.0%	0s
_int_free	0.0%	0.010s
[Unknown stack frame(s)]	0.0%	0s
__fast_solver_MOD_solveoption2	0.1%	0.020s
__fast_solver_MOD_ad_inputsolve_noifw	0.1%	0.010s
__elastodyn_MOD_ed_calcoutput	0.1%	0.610s
__fast_solver_MOD_transfer_ed_to_hd_sd_bd_mooring	0.0%	0s

CPU Time
Viewing 1 of 20 selected stack(s)
48.6% (371.725s of 764.950s)

```
libopenblas.so.0!dgemv_kernel_4x4.isra.0 - [unknown source file]
libopenblas.so.0!dgemv_n_SKYLAKEX+0x563 - [unknown sourc…
libopenblas.so.0!dtrsv_NLU+0x1b0 - [unknown source file]
libopenblas.so.0!dgetrs_N_parallel+0x8f - [unknown source file]
libopenblas.so.0!dgetrs_+0x1d3 - [unknown source file]
openfast!__nwtc_lapack_MOD_lapack_dgetrs1+0x25b - NWTC_…
openfast!__fast_solver_MOD_fullopt1_inputoutputsolve+0x1df9 -…
openfast!__fast_solver_MOD_solveoption1+0xb68 - FAST_Solver…
openfast!__fast_solver_MOD_calcoutputs_and_solveforinputs+0…
openfast!__fast_subs_MOD_fast_solution+0x2d1 - FAST_Subs.f9…
openfast!__fast_subs_MOD_fast_solution_t+0xac - FAST_Subs.f…
openfast!fast+0xe327b - FAST_Prog.f90:128
openfast!main+0x1e - FAST_Prog.f90:35
libc.so.6!__libc_start_main+0xf4 - [unknown source file]
openfast!_start+0x28 - [unknown source file]
```

BeamDyn

While BeamDyn provides a high-fidelity blade-response calculation, it is a computationally expensive module. Initial profiling highlighted the *bd_elementmatrixga2* subroutine, in particular, as a hotspot. However, initial attempts to improve performance in BeamDyn highlighted needs for algorithmic improvements and refinements to the module's data structures.

Results

Though work is ongoing, OpenFAST time-to-solution performance has improved and the performance potential is better understood.

Some keys outcomes from the first year of the IPCC project are as follows:

- Use of Intel® compiler and MKL library provides dramatic speedup over GCC and LAPACK
  - Additional significant gains are possible through MKL threading for offshore simulations
- Offshore-wind-turbine simulations are poorly load balanced across modules
  - Land-based-turbine configuration better balanced
  - OpenMP Tasks are employed to achieve better load-balancing
- OpenMP module-level parallelism provides significant, but limited speed up due to imbalance across different module tasks
- Core algorithms need significant modification to enable OpenMP and SIMD benefits

Speedup - Intel® Compiler and MKL

By employing the standard Intel® developer tools tech stack, a performance improvement over GNU tools was demonstrated:

Compiler	Math Library	5MW_Land_BD_DLL_WTurb	5MW_OC4Jckt_DLL_WTurb_WavesIrr_MGrowth
GNU	LAPACK	2265 s (1.0x)	673 s (1.0x)
Intel® 17	LAPACK	1650 s (1.4x)	251 s (2.7x)
Intel® 17	MKL	1235 s (1.8x)	—
Intel® 17	MKL Multi-threaded	722 s (3.1x)	—

Speedup - OpenMP at FAST_Solver

A performance improvement was domenstrated by adding OpenMP directives to the *FAST_Solver* module. Although the solution scheme is not well balanced, parallelizing mesh mapping and calculation routines resulted in the following speedup:

Compiler	Math Library	5MW_Land_BD_DLL_WTurb	5MW_OC4Jckt_DLL_WTurb_WavesIrr_MGrowth
Intel® 17	MKL - 1 thread	1073 s (2.1x)	100 s (6.7x)
Intel® 17	MKL - 8 threads	597 s (3.8x)	—

Ongoing Work

The next phase of the OpenFAST performance improvements are focused in two key areas:

1. Implementing the outcomes from previous work throughout OpenFAST modules and glue codes
2. Preparing OpenFAST for efficient execution on Intel®'s next generation platforms

5.3.7 Versioning

OpenFAST follows semantic versioning. In summary, this means that with a version number as MAJOR.MINOR.PATCH, the components will be incremented as follows:

- MAJOR version when introducing incompatible API changes,
- MINOR version when adding functionality in a backwards-compatible manner, and
- PATCH version when making backwards-compatible bug fixes.

For example, `OpenFAST-v1.0.0-123-gabcd1234-dirty` describes OpenFAST as:

Version Component	Explanation
v1.0.0	MAJOR.MINOR.PATCH numbering system; corresponds to a tagged commit made by NREL on GitHub
123-g	Number of additional commits after the most recent tag for a build (the `-g` is for `git`)
abcd1234	First 8 characters of the current commit hash
dirty	Denotes that local changes have been made but not committed; omitted if there are no local changes

5.4 Other Documentation

Additional documentation exists that may be useful for developers seeking deeper understanding of the solver and mathematics. This documentation is not generally necessary for most development efforts.

5.4.1 Other documentation

Additional documentation exists that may be useful for developers seeking deeper understanding of the solver and mathematics. This documentation is not generally necessary for most development efforts.

- **`DCM_Interpolation.pdf`** This is a summary of the mathematics used in the interpolation of DCM (direction cosine matrices) using logarithmic mapping and matrix exponentials.
- **`OpenFAST_Algorithms.pdf`** This is a summary of the solve method used in the glue code.
- **`OutListParameters.xlsx`** This Excel file contains the full list of outputs for each module. It is used to generate the Fortran code for the output channel list handling for each module (this code is generally in the _IO.f90 files). The MATLAB script available in the matlab-toolbox repository at *Utilities/GetOutListParameters.m*.

CHAPTER

SIX

LICENSING

The OpenFAST software, including its underlying modules, are licensed under Apache License Version 2.0 open-source license.

CHAPTER SEVEN

GETTING HELP

For possible bugs, enhancement requests, or code questions, please submit an issue at the OpenFAST Github repository.

For OpenFAST usage questions, users should consider the FAST Forum, which provides a large 10+ year legacy of FAST-related Q&A; the forum's search functionality should be used before posting questions to either github issues or the forum.

Users may find the established FAST v8 through the NWTC Information Portal: https://nwtc.nrel.gov/

Please contact Michael.A.Sprague@NREL.gov. with questions regarding the OpenFAST development plan or how to contribute.

CHAPTER EIGHT

ACKNOWLEDGEMENTS

This software is developed and maintained by researchers at the National Renewable Energy Laboratory with funding from U.S. Department of Energy Wind Energy Technology Office through the Atmosphere to electrons (A2e) research initiative.

NREL gratefully acknowledges development contributions from the following organizations:

- Envision Energy USA, Ltd
- Brigham Young University

NREL gratefully acknowledges additional development support through designation as an Intel® Parallel Computing Center (IPCC).

BIBLIOGRAPHY

[olaf-Abe16] H. Abedi. *Development of Vortex Filament Method for Wind Power Aerodynamics*. PhD thesis, Chalmers University of Technology, Gothenburg, Sweden, 2016.

[olaf-ALR02] S. Ananthan, J. G. Leishman, and M. Ramasamy. The role of filament stretching in the free-vortex modeling of rotor wakes. In *58th Annual Forum and Technology Display of the American Helicopter Society International*. Montreal, Canada, 2002.

[olaf-BL93] A. Bagai and J. G. Leishman. Flow visualization of compressible vortex structures using density gradient techniques. *Experiments in Fluids*, 15(6):431–442, 1993.

[olaf-BL94] A. Bagai and J. G. Leishman. Rotor free-wake modeling using a pseudo-implicit technique including comparisons with experimental data. In *50th Annual Forum of the American Helicopter Society*. Washington, D.C., 1994.

[olaf-Bra17] E. Branlard. *Wind Turbine Aerodynamics and Vorticity-Based Methods: Fundamentals and Recent Applications*. Springer International Publishing, 2017. ISBN 978-3-319-55163-0. doi:10.1007/978-3-319-55164-7.

[olaf-BPG+15] E. Branlard, G. Papadakis, M. Gaunaa, G. Winckelmans, and T. J. Larsen. Aeroelastic large eddy simulations using vortex methods: unfrozen turbulent and sheared inflow. *Journal of Physics: Conference Series (Online)*, 2015. doi:10.1088/1742-6596/625/1/012019.

[olaf-Gup06] S. Gupta. *Development of a Time-Accurate Viscous Lagrangian Vortex Wake Model for Wind Turbine Applications*. PhD thesis, Univeristy of Maryland, College Park, MD, 2006.

[olaf-GL02] S. Gupta and J. G. Leishman. Free-vortex filament methods for the analysis of helicopter rotor wakes. *Journal of Aircraft*, 39(5):759–775, 2002.

[olaf-Han08] M. O. L. Hansen. *Aerodynamics of Wind Turbines*. Earthscan, London; Sterling, VA, 2008.

[olaf-Jon13] J. Jonkman. The new modularization framework for the fast wind turbine cae tool. Technical report NREL/CP-5000-57228, National Renewable Energy Laboratory, 2013.

[olaf-Ker00] J. Kerwin. Lecture notes hydrofoil and propellers. Technical Report, M.I.T., 2000.

[olaf-Lei06] J. Leishman. *Principles of Helicopter Aerodynamics*. Cambridge Univ. Press, Cambridge, MA, 2006.

[olaf-LBB02] J. G. Leishman, M. J. Bhagwat, and A. Bagai. Free-vortex filament methods for the analysis of helicopter rotor wakes. *Journal of Aircraft*, 39(5):759–775, 2002.

[olaf-Pap14] G. Papadakis. *Development of a hybrid compressible vortex particle method and application to external problems including helicopter flows*. PhD thesis, National Technical University of Athens, 2014.

[olaf-Ran58] W. J. M. Rankine. *Manual of Applied Mechanics*. Griffen Co., London, 1858.

[olaf-Rib07] M. Ribera. *Helicopter Flight Dynamics Simulation with a Time-Accurate Free-Vortex Wake Model*. PhD thesis, University of Maryland, College Park, MD, 2007.

[olaf-Ros31] L. Rosenhead. The formation of vortices from a surface of discontinuity. *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, 134(823):170–192, 1931. URL: http://www.jstor.org/stable/95835.

[olaf-Scu75] M. P. Scully. *Computation of Helicopter Rotor Wake Geometry and Its Influence on Rotor Harmonic Airloads*. PhD thesis, Massachusetts Institute of Technology, Cambridga, MA, 1975.

[olaf-SGarciaSorensenS17] M. Sessarego, N. Ramos García, J. N. Sørensen, and W. Z. Shen. Development of an aeroelastic code based on three-dimensional viscous-inviscid method for wind turbine computations. *Wind Energy*, 20(7):1145–1170, 2017. doi:10.1002/we.2085.

[olaf-SJJ15] Michael A. Sprague, Jason M. Jonkman, and Bonnie J. Jonkman. Fast modular framework for wind turbine simulation: new algorithms and numerical examples. Technical Report NREL/CP-2C00-63203, National Renewable Energy Laboratory, 2015.

[olaf-vG03] A. van Garrel. Development of a wind turbine aerodynamics simulation module. Technical Report ECN-C–03-079, ECN, 2003.

[olaf-VKM91] G. H. Vatistas, V. Koezel, and W. C. Mih. A simpler model for concentrated vortices. *Experiments in Fluids*, 11(1):73–76, 1991.

[olaf-Vou06] S. G. Voutsinas. Vortex methods in aeronautics: how to make things work. *International Journal of Computational Fluid Dynamics*, 2006.

[olaf-Wei47] J. Weissinger. The lift distribution of swept-back wings. Technical report TM 1120, NACA, 1947.

[olaf-WL93] G. S. Winckelmans and A. Leonard. Contributions to vortex particle methods for the computation of 3-dimensional incompressible unsteady flows. *Journal Of Computational Physics*, 109(2):247–273, 1993.

[aa-Ami75] Roy K. Amiet. Acoustic radiation from an airfoil in a turbulent stream. *Journal of Sound and Vibration*, 41(4):407–420, 1975. doi:10.1016/S0022-460X(75)80105-2.

[aa-BTD+19] Pietro Bortolotti, Helena Canet Tarres, Katherine Dykes, Karl Merz, Latha Sethuraman, David Verelst, and Frederik Zahle. Systems engineering in wind energy - wp2.1 reference wind turbines. Technical Report, IEA Technical Report, 2019. URL: https://www.nrel.gov/docs/fy19osti/73492.pdf.

[aa-BPM89] Thomas F. Brooks, D. Stuart Pope, and Michael A. Marcolini. Airfoil self-noise and prediction. Reference Publication 1218, NASA, 1989.

[aa-DG87] Mark Drela and Michael B. Giles. Viscous-inviscid analysis of transonic and low reynolds number airfoils. *AIAA Journal*, 25(10):1347–1355, 1987. doi:10.2514/3.9789.

[aa-GBW+97] Gianfranco Guidati, Rainer Bareiss, Siegfried Wagner, Rene Parchen, Gianfranco Guidati, Rainer Bareiss, Siegfried Wagner, and Rene Parchen. Simulation and measurement of inflow-turbulence noise on airfoils. In *3rd AIAA/CEAS Aeroacoustics Conference*. 1997. doi:10.2514/6.1997-1698.

[aa-KGW+18] Levin Klein, Jonas Gude, Florian Wenz, Thorsten Lutz, and Ewald Krämer. Advanced computational fluid dynamics (cfd)–multi-body simulation (mbs) coupling to assess low-frequency emissions from wind turbines. *Wind Energy Science Journal*, 3:713–728, 2018. doi:10.5194/wes-3-713-2018.

[aa-Low70] Martin V. Lowson. Theoretical analysis of compressor noise evaluation. *The Journal of the Acoustical Society of America*, 47:371–385, 1970. doi:10.1121/1.1911508.

[aa-MGM04] Patrick Moriarty, Gianfranco Guidati, and Paul Migliore. Recent improvement of a semi-empirical aeroacoustic prediction code for wind turbines. In *10th AIAA/CEAS Aeroacoustics Conference*. 2004. doi:10.2514/6.2004-3041.

[aa-MGM05] Patrick Moriarty, Gianfranco Guidati, and Paul Migliore. Prediction of turbulent inflow and trailing-edge noise for wind turbines. In *11th AIAA/CEAS Aeroacoustics Conference*. 2005. doi:10.2514/6.2005-2881.

[aa-Mor05] Patrick J. Moriarty. Nafnoise user's guide. Technical Report, National Renewable Energy Laboratory, Golden, CO, 2005. URL: https://github.com/NREL/NAFNoise/blob/master/NAFNoise.pdf.

[aa-MH05] Patrick J. Moriarty and A. C. Hansen. Aerodyn theory manual. Technical Report NREL/TP-500-36881, National Renewable Energy Laboratory, Golden, CO, 2005. URL: https://www.nrel.gov/docs/fy05osti/36881.pdf.

[aa-MM03] Patrick J. Moriarty and Paul G. Migliore. Semi-empirical aeroacoustic noise prediction code for wind turbines. Technical Report NREL/TP-500-34478, National Renewable Energy Laboratory, Golden, CO, 2003. URL: https://www.nrel.gov/docs/fy04osti/34478.pdf.

[aa-Par98] René R. Parchen. Progress report DRAW: a prediction scheme for trailing edge noise based on detailed boundary layer characteristics. Technical Report, TNO Institute of Applied Physics, 1998.

[aa-PA76] R. Paterson and R. Amiet. Acoustic radiation and surface pressure characteristics of an airfoil due to incident turbulence. In *3rd Aeroacoustics Conference*. AIAA, 1976. doi:10.2514/6.1976-571.

[aa-SBCB18] CR Sucameli, P Bortolotti, A Croce, and CL Bottasso. Comparison of some wind turbine noise emission models coupled to BEM aerodynamics. *Journal of Physics: Conference Series*, 1037:022038, jun 2018. doi:10.1088/1742-6596/1037/2/022038.

[aa-Vit81] Larry A. Viterna. Method for predicting impulsive noise generated by wind turbine rotors. Technical Report DOE/NASA/20320-36, 1981. URL: https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19820013840.pdf.

[aa-ZHS05] Wei J. Zhu, Nicolai Heilskov, and Wen Zhong Shen. Modeling of aerodynamically generated noise from wind turbines. *Journal of Solar Energy Engineering*, 127(4):517–528, 2005. doi:10.1115/1.2035700.

[BC80] K. J. Bathe and A. P. Cimento. Some practical procedures for the solution of nonlinear finite element equations. *Computer Methods in Applied Mechanics and Engineering*, 22:59–85, 1980. http://web.mit.edu/kjb/www/Publications_Prior_to_1998/Some_Practical_Procedures_for_the_Solution_of_Nonlinear_Fini
doi:10.1016/0045-7825(80)90051-1.

[Bau10] O. A. Bauchau. *Flexible Multibody Dynamics*. Springer, 2010. doi:10.1007/978-94-007-0335-3.

[BEH08] O.A. Bauchau, A. Epple, and S.D. Heo. Interpolation of finite rotations in flexible multibody dynamics simulations. *Proceedings of the Institution of Mechanical Engineers, Part K: Journal of Multi-body Dynamics*, 222:353–366, 2008.

[CH93] J. Chung and G. M. Hulbert. A time integration algorithm for structural dynamics with improved numerical dissipation: the generalized- method. *Journal of Applied Mechanics*, 60:371–375, 1993. doi:10.1115/1.2900803.

[GSJ13] A. Gasmi, M.A. Sprague, and J.M. Jonkman. Numerical stability and accuracy of temporally coupled multi physics modules in wind-turbine cae tools. In *Proceedings of the 51st AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*. Grapevine, Texas, January 2013.

[Hod06] Dewey H. Hodges. *Nonlinear Composite Beam Theory*. AIAA, 2006.

[JelenicC99] G. Jelenić and M. A. Crisfield. Geometrically exact 3d beam theory: implementation of a strain-invariant finite element for statics and dynamics. *Computer Methods in Applied Mechanics and Engineering*, 171:141–171, 1999.

[Jon13] J.M. Jonkman. The new modularization framework for the fast wind turbine cae tool. In *Proceedings of the 51st AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition*. Grapevine, Texas, January 2013.

[JJ13] Jason Jonkman and Bonnie Jonkman. Fast v8. https://nwtc.nrel.gov/FAST8, October 2013. [Online; accessed 29-OCTOBER-2014].

[Pat84] A. T. Patera. A spectral element method for fluid dynamics: laminar flow in a channel expansion. *Journal of Computational Physics*, 54:468–488, 1984.

[RP87] E. M. Ronquist and A. T. Patera. A legendre spectral element method for the stefan problem. *International Journal for Numerical Methods in Engineering*, 24:2273–2299, 1987.

[SG03] M. A. Sprague and T. L. Geers. Spectral elements and field separation for an acoustic fluid subject to cavitation. *Journal of Computational Physics*, 184:149–162, 2003.

[SG04] M. A. Sprague and T. L. Geers. A spectral-element method for modeling cavitation in transient fluid-structure interaction. *International Journal for Numerical Methods in Engineering*, 60:2467–2499, 2004.

[SJJ14] M.A. Sprague, J.M. Jonkman, and B.J. Jonkman. Fast modular wind turbine cae tool: non matching spatial and temporal meshes. In *Proceedings of the 32nd ASME Wind Energy Symposium*. National Harbor, Maryland, January 2014.

[WJSJ15] Q. Wang, N. Johnson, M.A. Sprague, and J. Jonkman. Beamdyn: a high-fidelity wind turbine blade solver in the fast modular framework. In *Proceedings of the 33rd ASME Wind Energy Symposium*. Kissimmee, Florida, January 2015. https://www.nrel.gov/docs/fy15osti/63165.pdf.

[WS13] Q. Wang and M.A. Sprague. A legendre spectral finite element implementation of geometrically exact beam theory. In *Proceedings of the 54th Structures, Structural Dynamics, and Materials Conference*. Boston, Massachusetts, April 2013.

[WSJJ14] Q. Wang, M.A. Sprague, J. Jonkman, and N. Johnson. Nonlinear legendre spectral finite elements for wind turbine blade dynamics. In *Proceedings of the 32nd ASME Wind Energy Symposium*. National Harbor, Maryland, January 2014.

[WSJJ16] Q. Wang, M.A. Sprague, J. Jonkman, and B. Jonkman. Partitioned nonlinear structural analysis of wind turbines using beamdyn. In *Proceedings of the 34th ASME Wind Energy Symposium*. San Diego, California, January 2016.

[WYS13] Q. Wang, W. Yu, and M.A. Sprague. Gemoetrically nonlinear analysis of composite beams using wiener-milenković parameters. In *Proceedings of the 54th Structures, Structural Dynamics, and Materials Conference*. Boston, Massachusetts, April 2013.

INDEX