
OpenFAST Documentation

Release 1.0

National Renewable Energy Laboratory

Feb 19, 2019

Contents

1	Overview	1
2	This documentation	3
3	Getting Started	5
4	Installing OpenFAST	7
4.1	Obtaining the OpenFAST source code	7
4.2	Building OpenFAST with CMake on Linux and Mac	7
4.3	Building OpenFAST with Spack	9
4.4	Building OpenFAST on Windows with CMake and Cygwin 64-bit	11
4.5	Building OpenFAST on Windows with Visual Studio	14
5	Testing OpenFAST	17
5.1	Configuring the test suite	17
5.2	Test specific documentation	18
6	User Documentation	27
6.1	AeroDyn Users Guide and Theory Manual	27
6.2	BeamDyn User Guide and Theory Manual	43
6.3	HydroDyn Users Guide and Theory Manual	67
6.4	FAST v8 and the transition to OpenFAST	96
6.5	C++ API Users Guide	102
7	Developer Documentation	105
7.1	OpenFAST-development philosophy	105
7.2	Workflow for interacting with the OpenFAST github.com repo	106
7.3	Building this documentation locally	108
7.4	Doxygen API documentation	109
8	Important Links	111
9	Licensing	113
10	Getting Help	115
11	Acknowledgements	117

OpenFAST is an open-source wind turbine simulation tool that was established in 2017 with the FAST v8 code as its starting point (see *FAST v8 and the transition to OpenFAST*). 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.

OpenFAST and its underlying modules are mostly written in Fortran (adhering to the 2003 standard), but modules can be written in C/C++. OpenFAST was created with the goal of being a community model, with developers and users from research laboratories, academia, and industry. Our goal is also to ensure that OpenFAST is sustainable software that is well tested and well documented. 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.

OpenFAST is under development; our team at NREL is now enhancing this documentation and automated unit/regression testing. During this transition period, users can find FAST v8 documentation at <https://nwtc.nrel.gov/>.

CHAPTER 2

This documentation

OpenFAST documentation is built using [Sphinx](#), which uses [reStructuredText](#) as its markup language. Online documentation is hosted on [readthedocs](#), where one can choose between documentation generated from the OpenFAST *master* or *dev* github branches (if viewing this on <http://openfast.readthedocs.io> click on Read the Docs “box” on the lower left corner of the browser screen for options).

This documentation is divided into two parts:

User Documentation

Directed towards end-users, this part provides detailed documentation regarding installation and usage of the OpenFAST and its underlying modules, as well as theory and verification documentation. Also included are instructions for using the automated test suite, which serves as a suite of examples.

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.

Note: If viewing this on <http://openfast.readthedocs.io>, one can get this documentation in PDF form via Read the Docs “box” on the lower left corner of the browser screen.

CHAPTER 3

Getting Started

Get the code:

OpenFAST can be cloned (i.e., downloaded) from its [Github Repository](https://github.com/OpenFAST/OpenFAST.git), e.g., from the command line:

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

It can also be downloaded directly from <https://github.com/OpenFAST/OpenFAST.git>.

Compile the code:

See [Installing OpenFAST](#), for installation instructions (including dependency requirements) for CMake, Spack, and Visual Studio and for multiple platforms (Linux, Mac, Windows). As an example, from the command line in a Mac or Linux environment:

```
cd OpenFAST
mkdir build && cd build
cmake ../
make
```

Note that one can see all of the *make* targets via

```
make help
```

Use the code:

See [User Documentation](#), which is under construction. In the interim, users may refer to the FAST v8 documentation at <https://nwtc.nrel.gov/>.

Develop the code:

See [Developer Documentation](#), which is under construction. In the interim, developers may consult the FAST v8 Programmer's Handbook.

Installing OpenFAST

The following pages provide instructions for building OpenFAST and/or its modules from source code. The developer team is moving towards a CMake-only approach that well supports Window Visual Studio users, but at this time we provide a separate build path for those users.

4.1 Obtaining the OpenFAST source code

OpenFAST can be cloned (i.e., downloaded) from its [Github Repository](https://github.com/OpenFAST/OpenFAST). For example, from a command line:

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

It can also be downloaded directly from <https://github.com/OpenFAST/OpenFAST.git>.

4.2 Building OpenFAST with CMake on Linux and Mac

We describe here how to install OpenFAST (or any of its modules) using the [CMake](#) build system on Linux or Mac OS systems. Separate [CMake](#) documentation is provided for Windows Cygwin users: see [Section 4.4](#). Also, some template build scripts are available in `openfast/share`.

4.2.1 Required software for building OpenFAST

In order to build OpenFAST using CMake, one needs the following minimum set of packages installed:

- Fortran compiler (GNU compiler version above 4.6.0 or Intel compiler version above 11)
- C/C++ compiler
- GNU Make (version 3.81 or later)
- CMake (version 2.8.12 or later)

4.2.2 OpenFAST third-party-library (TPL) dependencies

OpenFAST has the following dependencies:

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

4.2.3 CMake build instructions

If one has the appropriate TPLs, CMake, and git installed, obtaining and building OpenFAST can be accomplished as follows:

```
# obtain the source code; e.g., from the command line using git:
git clone https://github.com/OpenFAST/OpenFAST.git

# go to the OpenFAST directory
cd OpenFAST

# create a directory called `build`
mkdir build

# go to the build directory
cd build

# execute CMake with the default options, which will create a Makefile
cmake ../

# execute a make command (with no target provided, equivalent to `make all`
make
```

This will build the OpenFAST suite in the `build` directory, which can be deleted for a clean build.

There are many Makefile targets (besides `all`), which can be listed via `help`:

```
# list available make targets
make help

# make a specific target, e.g.
make beamdyn_driver
```

Current CMake options

Below is a list of current CMake options including their default settings (which will effect, e.g., the targets in a resulting Makefile).

- `BUILD_DOCUMENTATION` - Build documentation (Default: OFF)
- `BUILD_FAST_CPP_API` - Enable building OpenFAST - C++ API (Default: OFF)
- `BUILD_SHARED_LIBS` - Enable building shared libraries (Default: OFF)
- `CMAKE_BUILD_TYPE` - Choose the build type: Debug Release (Default: Release)
- `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)
- `ORCA_DLL_LOAD` - Enable OrcaFlex library load (Default: OFF)
- `USE_DLL_INTERFACE` - Enable runtime loading of dynamic libraries (Default: ON)

CMake options can be executed from the command line as, e.g., the CMake command above could be executed as

```
# e.g., to enable Makefile for local building of sphinx-based documentation
cmake -DBUILD_DOCUMENTATION:BOOL=ON ..

# e.g., to compile OpenFAST in single precision
cmake -D:DOUBLE_PRECISION:BOOL=OFF ..
```

Parallel build

GNU Make has a parallel build option with the `-jobs` or `-j` flag, and the OpenFAST CMake configuration handles setting up the dependencies for Make so the build can be parallelized. However, it is important to note that the only parallel portion of the build process is in compiling the modules. Due to some interdependency between modules, the max parallel level is around 12. The remaining portion of the build, mainly compiling the OpenFAST library itself, takes a considerable amount of time and cannot be parallelized.

An example parallel build command is `make -j 8`.

4.3 Building OpenFAST with Spack

The process to build and install OpenFAST with [Spack](#) on Linux or macOS is described here.

4.3.1 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

4.3.2 Building OpenFAST Semi-Automatically Using Spack on macOS or Linux

The following describes how to build OpenFAST and its dependencies mostly automatically on your Mac 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.

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

4.4.1 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 `http://www.gtlib.gatech.edu` as the 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`
 - 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.

4.4.2 Compiling OpenFAST

1. Open Cygwin64 Terminal 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. Create a build directory

```
cd OpenFAST
mkdir build
cd build
```

5. Run cmake. Note that this step is necessary only if you change compiler settings, or add new files to any of the CMakeLists.txt. Modification of .f90 files do not require you to run cmake again, just re-run make command (see next item) to recompile with latest source code modifications.

```
FC=gfortran cmake ../
```

Sample output is shown below:

```
$ FC=gfortran cmake ../
-- The Fortran compiler identification is GNU 5.4.0
-- The C compiler identification is GNU 5.4.0
-- Check for working Fortran compiler: /usr/bin/gfortran.exe
-- Check for working Fortran compiler: /usr/bin/gfortran.exe -- works
-- Detecting Fortran compiler ABI info
-- Detecting Fortran compiler ABI info - done
-- Checking whether /usr/bin/gfortran.exe supports Fortran 90
-- Checking whether /usr/bin/gfortran.exe supports Fortran 90 -- yes
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Looking for Fortran sgemm
-- Looking for Fortran sgemm - found
-- Looking for pthread.h
-- Looking for pthread.h - found
-- Looking for pthread_create
-- Looking for pthread_create - found
-- Found Threads: TRUE
-- A library with BLAS API found.
-- A library with BLAS API found.
-- Looking for Fortran cheev
-- Looking for Fortran cheev - found
-- A library with LAPACK API found.
-- Setting system file as: src/SysGnuLinux.f90
-- Configuring done
```



```
-- Generating done
-- Build files have been written to: /home/sanantha/code/OpenFAST/build
```

6. Compile OpenFAST

```
make
```

Grab a cup of coffee as this takes a while on Cygwin. Once the compilation is completed, the OpenFAST executable is present in `OpenFAST/build/glue-codes/fast/openfast.exe`

7. Test the executable

```
$ glue-codes/fast/openfast.exe -h

*****
FAST (v8.17.00a-bjj, 27-Aug-2016)

Copyright (C) 2016 National Renewable Energy Laboratory

This program comes with ABSOLUTELY NO WARRANTY. See the "license.txt" file,
↳distributed with this
software for details.
*****

Running FAST (v8.17.00a-bjj, 27-Aug-2016), compiled as a 64-bit application using,
↳double
precision
linked with NWTC Subroutine Library (v2.11.00, 12-Nov-2016)

Syntax is:

    FAST_x64.exe [-h] <InputFile>

where:

    -h generates this help message.
    <InputFile> is the name of the required primary input file.

Note: values enclosed in square brackets [] are optional. Do not enter the brackets.

FAST_InitializeAll:The required input file was not specified on the command line.

FAST encountered an error during module initialization.
Simulation error level: FATAL ERROR

Aborting FAST.
```

““

4.4.3 Other tips

- You can specify an installation location during your `cmake` process so that the executable, libraries, and headers (e.g., MAP and OpenFOAM headers) are installed in a common location that you can use to update your environment variables.

```
# 1. Create an installation location mkdir -p ~/software

# 2. Instruct CMake to use the custom install location FC=gfortran cmake
-DCMAKE_INSTALL_PREFIX:PATH=$HOME/software ../

# 3. Compile OpenFAST executable make

# 4. Install OpenFAST to custom install location make install \`\`\`
```

With this step, you can execute `make install` after `make` (see step 6 above). Now the `openfast.exe` and other executables (e.g., `aerodyn.exe`) are available in `~/software/bin/` directory.

- If you desire to be able to run `openfast.exe` from the cmd window, 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.
- In addition to `openfast.exe`, the current CMake setup also allows the user to compile other executables or libraries without compiling the entire codebase. Use `make help` to see what targets are available and then do `make <TARGET>` to choose your desired target. For example, `make aerodyn` will compile only the `aerodyn.exe` executable and its dependencies without compiling the remaining targets.

4.5 Building OpenFAST on Windows with Visual Studio

4.5.1 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 seamlessly 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 5](#) on testing OpenFAST for further information on this topic.
 - We have been working with Continuum's [Anaconda](#) installation of Python 3.6 for Windows.

4.5.2 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.

NOTE: If this is the first time building OpenFAST, you will encounter many error messages. This is due to a known issue with Visual Studio's ability to understand the OpenFAST dependency structure, which includes Registry-generated Fortran files. You can resolve this issue by simply closing Visual Studio and then reopening the FAST.sln solution file. After you have reopened Visual Studio, build the solution again via the Build->Build Solution menu option.

7. If the solution built without errors, the executable will be located under the openfast\build\bin folder.

Testing OpenFAST

The OpenFAST test suite consists of system and module level regression tests and unit tests. The regression test compares 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 test 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.

5.1 Configuring the test suite

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

- `r-test`
- `pFUnit`

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

The test suite can be built with `CMake` similar to OpenFAST. The default `CMake` configuration is useful, but may need customization for particular build environments. See the installation documentation at [Section 4](#) 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 building with `CMake`. [Section 5.2.2](#) and [Section 5.2.1](#) have more information on regression testing and unit testing, respectively.

5.2 Test specific documentation

5.2.1 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

- Python 3+
- CMake and CTest
- Numpy and matplotlib (Optional)
- pFUnit (Included in unit test suite)

Compiling the unit tests

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

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

This creates a binary unit test executable at `openfast/build/unit_tests/[module]_utest`.

Executing the unit tests

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
(13 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:
#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 a portion 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-local/
|   |-- 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 with the appropriate list of test subroutines in `openfast/unit_tests/CMakeLists.txt` using the following format

```

set(testlist
    test_SUBROUTINE1
    test_SUBROUTINE2
    test_SUBROUTINE3
)
# it is important to keep the quotes around "${testlist}" in the call below
build_utest("module_name" "${testlist}")

```

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](#)

5.2.2 Regression test

The regression test executes a series of test cases which intend to fully describe OpenFAST and its module's capabilities. 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

- macOS with GNU compiler (default)
- Red Hat Enterprise Linux with Intel compiler
- Windows with Intel compiler

The compiler versions, specific math libraries, and hardware used to generate these baseline solutions are documented in the [r-test repository documentation](#). Currently, the regression test supports only double precision solutions, so it is required to build OpenFAST in double precision for testing. All baseline solutions are generated with a double precision build.

The regression test system can be executed with CMake and CTest or manually with an included 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 outlined below.

In both modes of execution a subdirectory 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 a preset tolerance, that particular test is reported as failed. The failure criteria is outlined in pseudocode 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:
    success
```

Dependencies

- Python 3+
- Numpy
- CMake and CTest (Optional)

- matplotlib (Optional)

Manual driver configuration

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

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](#).

CTest configuration

CTest is included with CMake and is mostly a set of preconfigured targets and commands. To use the CTest driver for the regression test, CMake must be run with one of two `CMakeLists.txt`'s:

- `openfast/CMakeList.txt`
- `openfast/reg_tests/CMakeLists.txt`

CMake variables can be configured in the [CMake GUI](#) or through the command line interface with `ccmake`.

The regression test specific CMake variables are

- `BUILD_TESTING`
- `CTEST_OPENFAST_EXECUTABLE`
- `CTEST_[MODULE]_EXECUTABLE`
- `CTEST_PLOT_ERRORS`
- `CTEST_REGRESSION_TOL`

IT IS IMPORTANT to verify that NREL 5MW turbine external controllers are compiled and placed in the correct location. 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/fast/5MW_Baseline/ServoDyn/DISCON.dll
openfast/build/reg_tests/glue-codes/fast/5MW_Baseline/ServoDyn/DISCON_ITIBarge.dll
openfast/build/reg_tests/glue-codes/fast/5MW_Baseline/ServoDyn/DISCON_OC3Hywind.dll
```

This can be accomplished manually with the CMake projects included with the DISCON source codes at `openfast/reg_tests/r-test/glue-codes/fast/5MW_Baseline/ServoDyn/` or during CMake configuration by setting the `CMAKE_INSTALL_PREFIX` CMake variable. If using this method, the install prefix variable should point to an existing and appropriate location for CMake to place the compiled binaries. This is important because the NREL 5MW turbine external controller CMake projects are preconfigured to install themselves in the appropriate location in the build directory. Then, it is important to execute `make install` rather than simply `make`. If `CMAKE_INSTALL_PREFIX` is not appropriately configured, the install step may fail or openfast binaries may be placed in some inappropriate default location.

After CMake configuration, 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/fast/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.

Running the regression test with CTest

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

- `ctest` - Run the entire regression test
- `ctest -V` - Run the entire regression test with verbose output
- `ctest -R [TestName]` - Run a test by name
- `ctest -j [N]` - Run all tests with N tests executing in parallel
- `ctest -N` - List all of the tests available

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` and called directly with

- `ctest -L [Label]`

These flags can be compounded making useful variations of `ctest` such as

- `ctest -V -L aerodyn14` - Runs all cases that use AeroDyn14 with verbose output
- `ctest -j 16 -L aerodyn14` - Runs all cases that use AeroDyn14 in 16 concurrent processes
- `ctest -V -R 5MW_DLL_Potential_WTurb` - Runs the case with name “5MW_DLL_Potential_WTurb”
- `ctest -N -L beamdyn` - Lists all tests with the “beamdyn” label

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 not modified by the automated process.

Regression test from scratch

- Build OpenFAST and the test suite

```
git clone --recursive https://github.com/openfast/openfast.git
# The default git branch is 'master'. If necessary, switch to your target branch:
# git checkout dev
mkdir build && cd build
# Configure CMake with openfast/CMakeLists.txt
# - BUILD_TESTING
# - CTEST_OPENFAST_EXECUTABLE
# - CTEST_[MODULE]_EXECUTABLE
cmake ..
make install
ctest
```

- Build only the test suite

```
git clone --recursive https://github.com/openfast/openfast.git
# The default git branch is 'master'. If necessary, switch to your target branch:
# git checkout dev
mkdir build && cd build
```

```
# Configure CMake with openfast/reg_tests/CMakeLists.txt
# - BUILD_TESTING
# - CTEST_OPENFAST_EXECUTABLE
# - CTEST_[MODULE]_EXECUTABLE
cmake ../reg_tests
make install
ctest
```

- *Windows With Visual Studio Regression Test*

Follow the link above for a detailed procedure. It is summarized below though excluding the procedure to build OpenFAST itself.

```
git clone --recursive https://github.com/openfast/openfast.git
cd openfast

## Build the ServoDyn external controller libraries
# Open the Visual Studio Solution (DISCON.sln) located in 'openfast\vs-build\DISCON'
# Choose Release and x64 for the Solutions Configuration and Solutions Platform
# Build Solution

## Execute the OpenFAST regression Tests
# Open a command prompt which is configured for Python (like Anaconda)
cd openfast\reg_tests
python manualRegressionTest.py ../build/bin/openfast_x64.exe Windows Intel 1e-5
```

5.2.3 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.
```

- (a) Type: `cd openfast` (change your working directory to the openfast folder)
- (b) Type: `git checkout dev` (this places your local repo on the correct branch of the openfast repo)
- (c) 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'
```

4. 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.

5. 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.exe in your openfast\build\bin folder

6. 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.exe Windows Intel**
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
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

- (a) If an individual test succeeds you will see PASS otherwise you will see FAIL after that test's name

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 6.4](#)

6.1 AeroDyn Users Guide and Theory Manual

6.1.1 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 takes 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 induced 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 6.1.2](#) describes the AeroDyn input files. [Section 6.1.3](#) discusses the output files generated by AeroDyn; these include an echo file, summary file, and the results file. [Section 6.1.4](#) provides modeling guidance when using AeroDyn. Example input files are included in [Section 6.1.5](#). A summary of available output channels are found [Section 6.1.5](#).

6.1.2 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 6.1.5](#).

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 6.1.5](#).

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 precone) 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,

$$U(Z) = \text{WndSpeed} \times \left(\frac{Z}{\text{HubHt}} \right)^{\text{ShearExp}} \quad (6.1)$$

where `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 `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

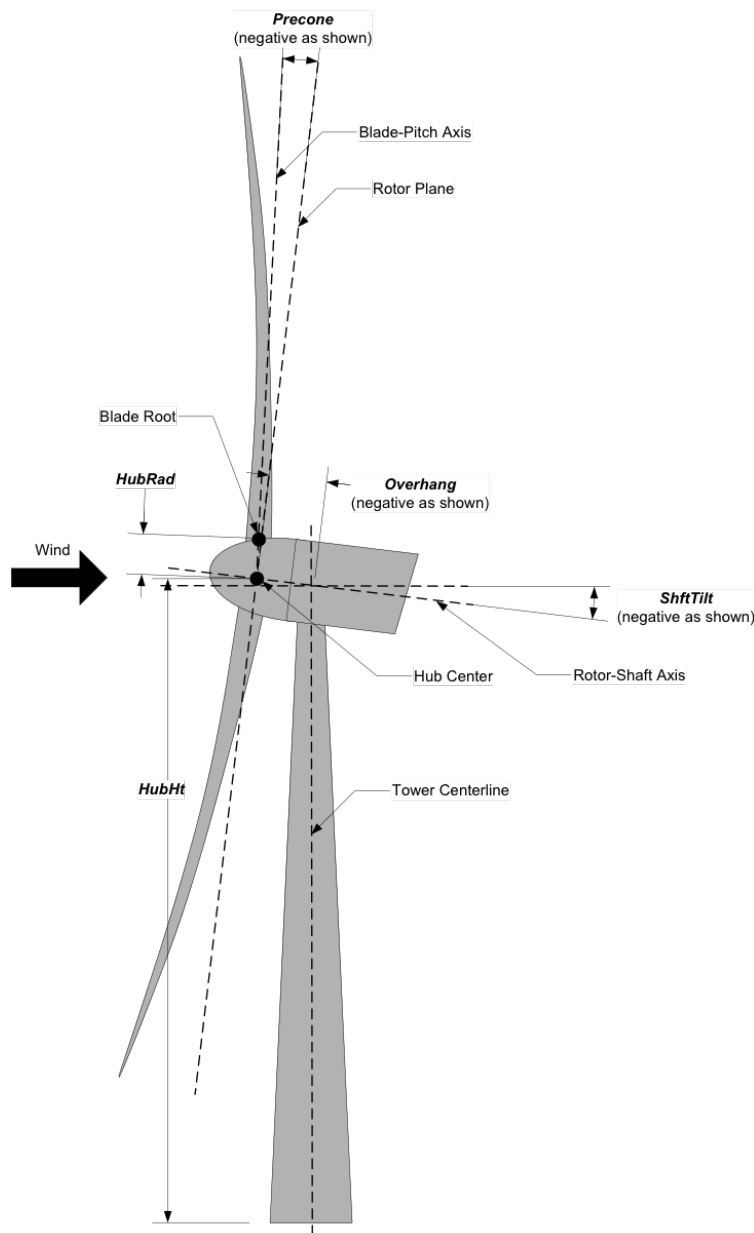


Fig. 6.1: AeroDyn Driver Turbine Geometry

step, which must match the time step for the aerodynamic calculations (DTAero) as specified in the primary AeroDyn input file, and T_{max} is the total simulation time.

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 6.1.5](#).

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 (FAST 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 BEM theory model. 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.

Environmental Conditions

`AirDens` specifies the fluid density and must be a value greater than zero; a typical value is around 1.225 kg/m³ for air (wind turbines) and 1025 kg/m³ 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 m²/s for air (wind turbines) and 1.004E-6 m²/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

The input parameters in this section are only used when `WakeMod = 1`.

`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.**

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.

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 6.1.2](#).

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 6.1.2](#). `ADB1File(1)` is the filename for blade 1, `ADB1File(2)` is the filename for blade 2, and `ADB1File(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. `NB1Outs` specifies the number of blade nodes that output is requested for (0 to 9) and `BlOutNd` on the next line is a list `NB1Outs` 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

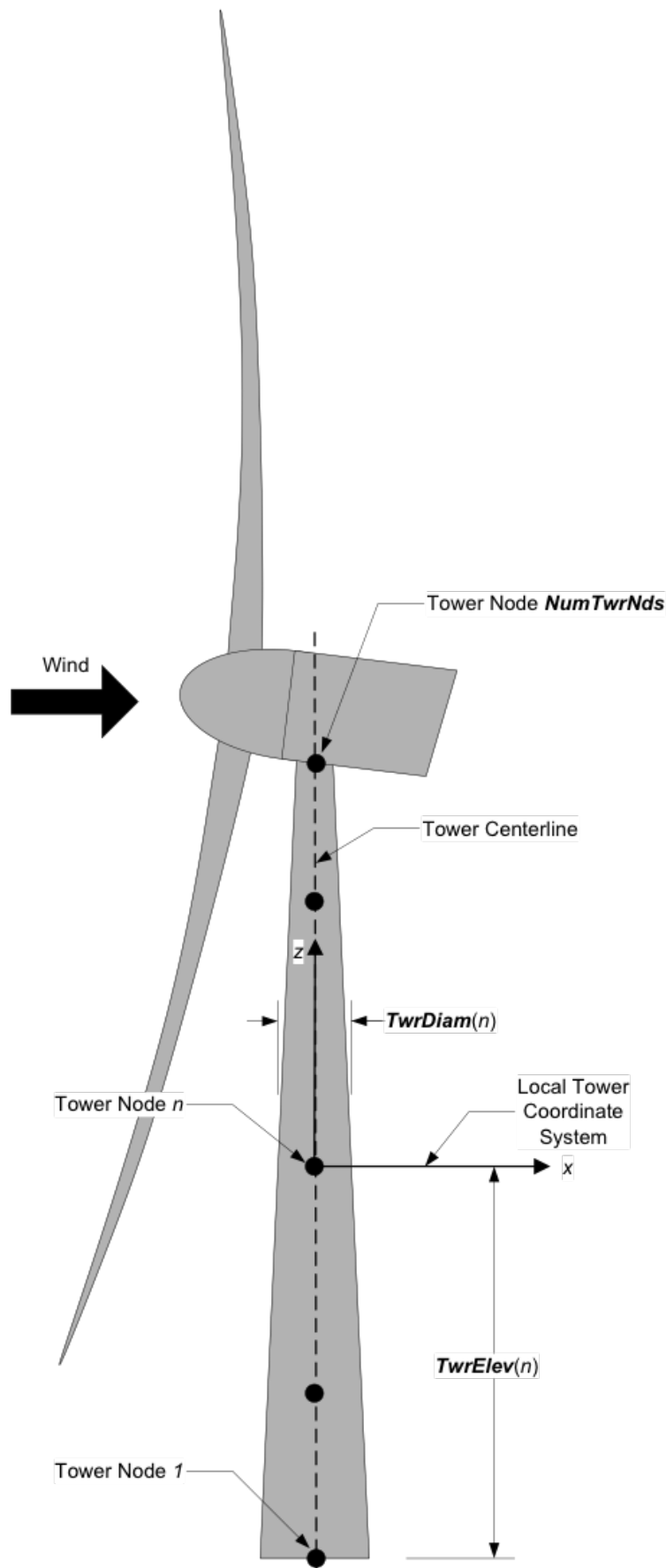


Fig. 6.2: AeroDyn Tower Geometry

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.

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 6.1.5](#).

`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. **Currently, AeroDyn can only use the first table in any given airfoil file, so you should set “NumTabs = 1” and you will need to make separate airfoil data input files and run separate simulations if you need to analyze data for different Reynolds numbers or aerodynamic-control settings.** The remaining parameters in the airfoil data input files are entered separately for each table.

`Re` and `Ctrl` are the Reynolds number (in millions) and aerodynamic-control setting for the included table, **but are both currently unused by AeroDyn.**

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 C_{vn} ; 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 C_{vn} ; 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_{mq} ; 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_{mq} ; 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 $\alpha_{00} \leq \text{AoA} \leq \alpha_{01}$ 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 $\text{AoA} > \alpha_{01}$ 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 $\alpha_{02} \leq \text{AoA} \leq \alpha_{00}$ 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 $\text{AoA} < \alpha_{02}$ 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;

- C_{d0} is the drag-force coefficient at zero-lift AoA;
- C_{m0} is the pitching-moment coefficient about the quarter-chord location at zero-lift AoA, positive for nose up;
- k_0 is a constant in the best fit curve of \hat{x}_{cp} and equals for $\hat{x}_{AC} - 0.25 \text{ UAMod} = 1$ (and is unused otherwise);
- k_1 is a constant in the best fit curve of \hat{x}_{cp} for $\text{UAMod} = 1$ (and is unused otherwise);
- k_2 is a constant in the best fit curve of \hat{x}_{cp} for $\text{UAMod} = 1$ (and is unused otherwise);
- k_3 is a constant in the best fit curve of \hat{x}_{cp} for $\text{UAMod} = 1$ (and is unused otherwise);
- k_{l_hat} is a constant in the expression of C_c due to leading-edge vortex effects for $\text{UAMod} = 1$ (and is unused otherwise);
- x_{cp_bar} is a constant in the expression of \hat{x}_{cp}^ν for $\text{UAMod} = 1$ (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 ($\text{NumAlf} = 1$ implies constant coefficients, regardless of the AoA). AeroDyn will interpolate the data provided via linear interpolation or via cubic splines, depending on the setting of input `InterpOrd` above. For each AoA, you must set the AoA (in degrees), α , the lift-force coefficient, $\text{Coefs}(:,1)$, the drag-force coefficient, $\text{Coefs}(:,2)$, and optionally the pitching-moment coefficient, $\text{Coefs}(:,3)$, and minimum pressure coefficient, $\text{Coefs}(:,4)$, 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 $\text{NumAlf} = 1$, 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 6.1.5](#).

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. 6.3. Twist is shown in Fig. 6.16 of Section 6.1.5.

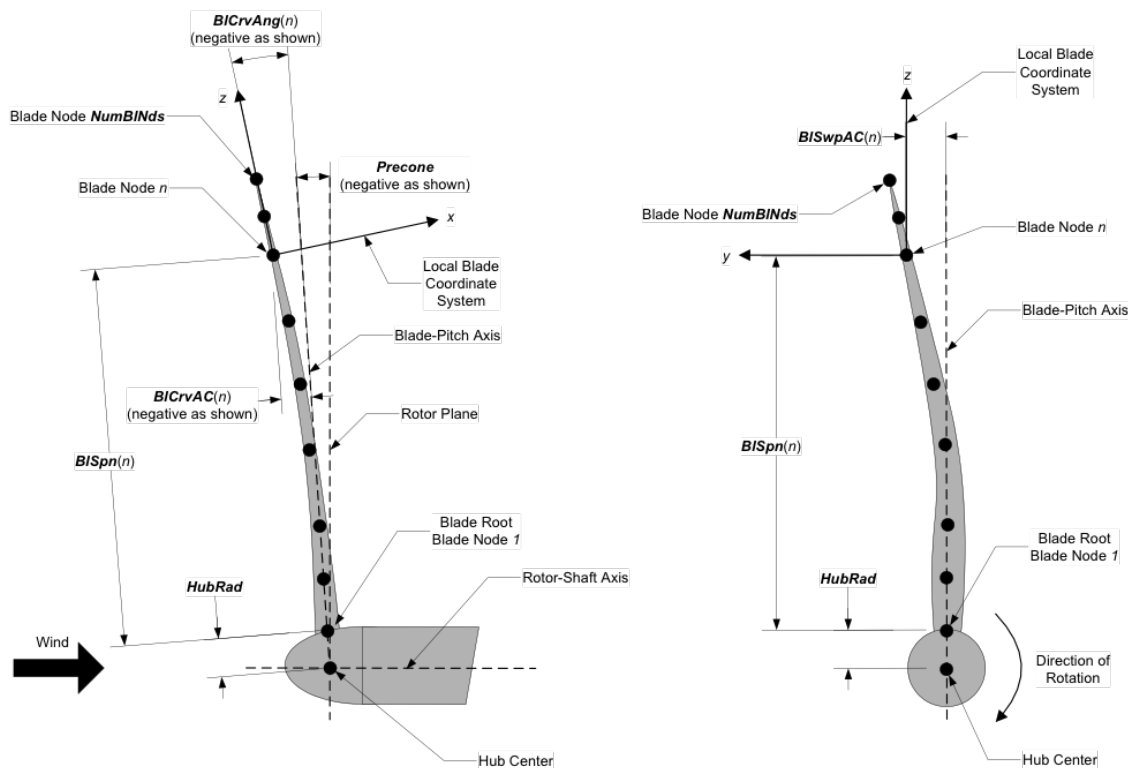


Fig. 6.3: AeroDyn Blade Geometry – Left: Side View; Right: Front View (Looking Downwind)

6.1.3 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.

6.1.4 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/m³, 1.460E-5 m²/s, 340.3 m/s, and 101,325 Pa, respectively. For seawater, typical values for `AirDens`, `KinVisc`, and `Pvap` are around 1025 kg/m³, 1.004E-6 m²/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 induction (`WakeMod = 1`) 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, Reynold's 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`.

6.1.5 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) Blade 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. $B\alpha N\beta$, refers to output node β of blade α , where α is a number in the range [1,3] and β is a number in the range [1,9], corresponding to entry β in the `BlOutNd` list. $TwN\beta$ refers to output node β of the tower and is in the range [1,9], corresponding to entry β in the `TwOutNd` list.

The local tower coordinate system is shown in Fig. 6.2 and the local blade coordinate system is shown in Fig. 6.16 below. Figure Fig. 6.16 also shows the direction of the local angles and force components.

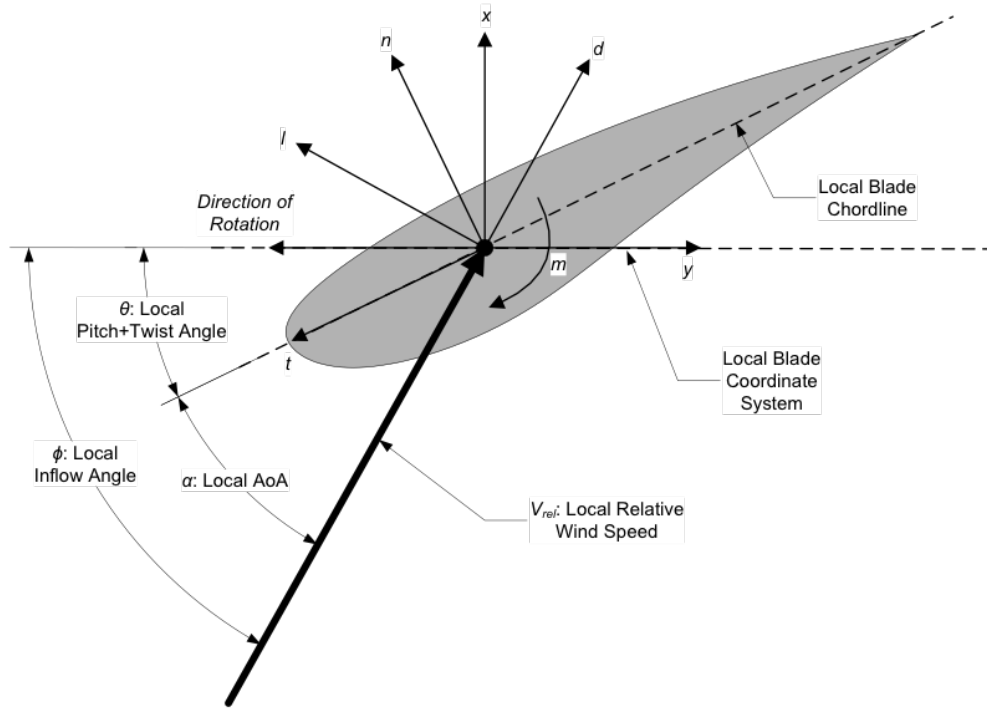


Fig. 6.4: AeroDyn Local Blade Coordinate System (Looking Toward the Tip, from the Root) – l: Lift, d: Drag, m: Pitching, x: Normal (to Plane), y: Tangential (to Plane), n: Normal (to Chord), and t: Tangential (to Chord)

6.2 BeamDyn User Guide and Theory Manual

6.2.1 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 is 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). LSFEs 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

Channel Name(s)	Units	Description
<i>Tower</i>		
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
<i>Blade</i>		
BaAzimuth	(deg)	Azimuth angle of Ba
BaPitch	(deg)	Pitch angle of Ba
BaNβClrc ¹	(m)	Tower clearance at BaNβ ¹
BaNβVUndx, BaNβVUndy, BaNβVUndz	(m/s), (m/s), (m/s)	Undisturbed wind velocity at BaNβ in the local blade coordinate system
BaNβVDissx, BaNβVDisy, BaNβVDisz	(m/s), (m/s), (m/s)	Disturbed wind velocity at BaNβ in the local blade coordinate system
BaNβSTVx, BaNβSTVy, BaNβSTVz	(m/s), (m/s), (m/s)	Structural translational velocity at BaNβ in the local blade coordinate system
BaNβVrel	(m/s)	Relative wind speed at BaNβ
BaNβDynP	(Pa)	Dynamic pressure at BaNβ
BaNβRe	(-)	Reynolds number (in millions) at BaNβ
BaNβM	(-)	Mach number at BaNβ
BaNβVIndx, BaNβVIndy	(m/s), (m/s)	Axial and tangential induced wind velocity at BaNβ
BaNβAxInd, BaNβTnInd	(-), (-)	Axial and tangential induction factors at BaNβ
BaNβAlpha, BaNβTheta, BaNβPhi, BaNβCurve	(deg), (deg), (deg), (deg)	AoA, pitch+twist angle, inflow angle, and curvature angle at BaNβ
BaNβCl, BaNβCd, BaNβCm, BaNβCpmin	(-), (-), (-), (-)	Lift force, drag force, pitching moment, minimum pressure,
BaNβCx, BaNβCy ² , BaNβCn, BaNβCt	(-), (-), (-), (-)	normal force (to plane), tangential force (to plane) ² , normal force (to

¹ BaNβClrc is based on the absolute distance to the nearest point in the tower from BaNβ 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 BaNβ 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. 6.5: AeroDyn Output Channel List

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 7th-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 refer 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.



Fig. 6.6: Coupled interaction between BeamDyn and FAST

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

ear 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 6.2.7](#). A summary of available output channels is found in Appendix [BeamDyn List of Output Channels](#).

6.2.2 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.

6.2.3 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 only needed for the stand-alone version of BeamDyn and 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 6.2.7](#).

Simulation Control Parameters

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

G_x , G_y , and G_z 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. $\text{GlbPos}(1)$, $\text{GlbPos}(2)$, $\text{GlbPos}(3)$ specifies three components of the initial global position vector along X , Y , and Z directions resolved in the global coordinate system, see Figure Fig. 6.7. 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. 6.7: Global and blade coordinate systems in BeamDyn.

Blade Floating Reference Frame Parameters

This section specifies the parameters that defines 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 frame is assumed to be in a constant rigid-body rotation mode about the origin of the global coordinate system, that is,

$$v_{rt} = \omega_r \times r_t \quad (6.2)$$

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. 6.7. 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 axes 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. (6.2).

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):

```
u%RootMotion%RotationVel(:, :) = 0.0D0
u%RootMotion%RotationVel(1,1) = IniVelo(5)
u%RootMotion%RotationVel(2,1) = IniVelo(6)
u%RootMotion%RotationVel(3,1) = IniVelo(4)
u%RootMotion%TranslationVel(:, :) = 0.0D0
u%RootMotion%TranslationVel(:, 1) = &
MATMUL(BD_Tilde(real(u%RootMotion%RotationVel(:, 1), BDKi)), temp_rr)
```

where `IniVelo(5)`, `IniVelo(6)`, and `IniVelo(4)` are the three components of the root angular velocity vector about X , Y , and Z axising in the global coordinate system, respectively; `temp_rr` is the global position vector at instant t . The first index in the `u%RootMotion%RotationVel(:, :)` and the `u%RootMotion%TranslationVel(:, :)` arrays range from 1 to 3 for load vector components along three directions and the second index of each array are set to 1, denoting the root FE node. Note that the internal BeamDyn variables (here `IniVelo`) are based on the internal BD coordinate system described in section FIXME.

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

$$\begin{aligned} a_0 &= \omega_r \times (\omega_r \times (r_0 + P)) \\ v_0 &= v_{r0} + \omega_r \times P \\ \omega_0 &= \omega_r \end{aligned} \quad (6.3)$$

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.

Applied Load

This section defines the applied loads, including distributed 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. 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:

```
! Define concentrated force vector
u%PointLoad%Force(:, :) = 0.0D0
! Define concentrated moment vector
u%PointLoad%Moment(:, :) = 0.0D0
```

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 `node_total`, where the latter is the total number of FE nodes. For example, a time-dependent sinusoidal force acting along the *X* direction applied at the 2nd FE node can be defined as

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

with `1.0D+03` being the amplitude and `6.0` being the period.

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

```
IF (p%quadrature .EQ. 1) THEN
  DO i=1, p%ngp*p%elem_total+2
    u%DistrLoad%Force(1:3, i) = InitInput%DistrLoad(1:3)
    u%DistrLoad%Moment(1:3, i) = InitInput%DistrLoad(4:6)
  ENDDO
ELSEIF (p%quadrature .EQ. 2) THEN
  DO i=1, p%ngp
    u%DistrLoad%Force(1:3, i) = InitInput%DistrLoad(1:3)
    u%DistrLoad%Moment(1:3, i) = InitInput%DistrLoad(4:6)
  ENDDO
ENDIF
```

where `p%ngp` is the number of quadrature points, `InitInput%DistrLoad(:)` is the constant uniformly distributed load BeamDyn reads from the driver input file, and `p%elem_total` is the total number of elements. The user can modify `InitInput%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 (i.e., `p%quadrature .EQ. 1`), 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 6.2.7](#).

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).

`Analysis_Type` specifies the type of an analysis. In the current version, there are two options: 1) static analysis, and 2) dynamic analysis. If BeamDyn is run in coupled FAST mode, this entry can be only set to 2, i.e., for dynamic analysis.

`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).

`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 6.2.5](#) for more details.

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:

$$kp_total = \sum_{i=1}^{member_total} n_i - member_total + 1 \quad (6.4)$$

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. 6.8 and Fig. 6.9 show two cases for member and key-point definition.

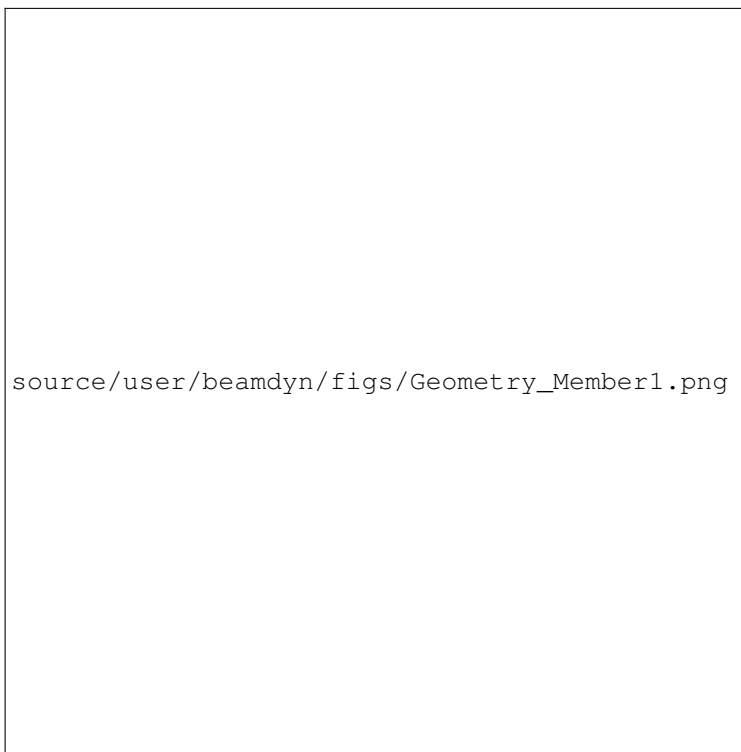


Fig. 6.8: Member and key point definition: one member defined by four key points;

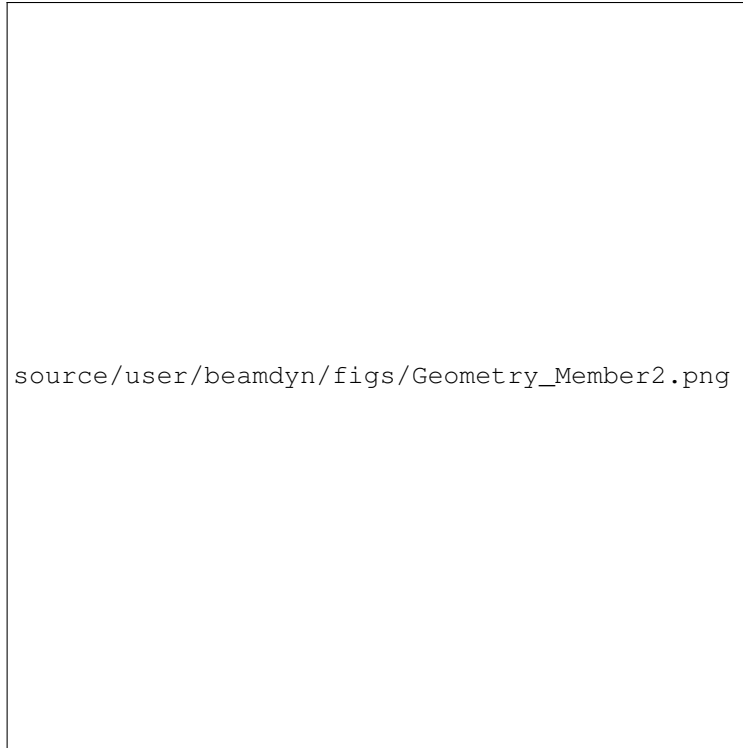


Fig. 6.9: Member and key point definition: two members defined by six key points.

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. 6.10 for more details on the blade geometry definition.

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

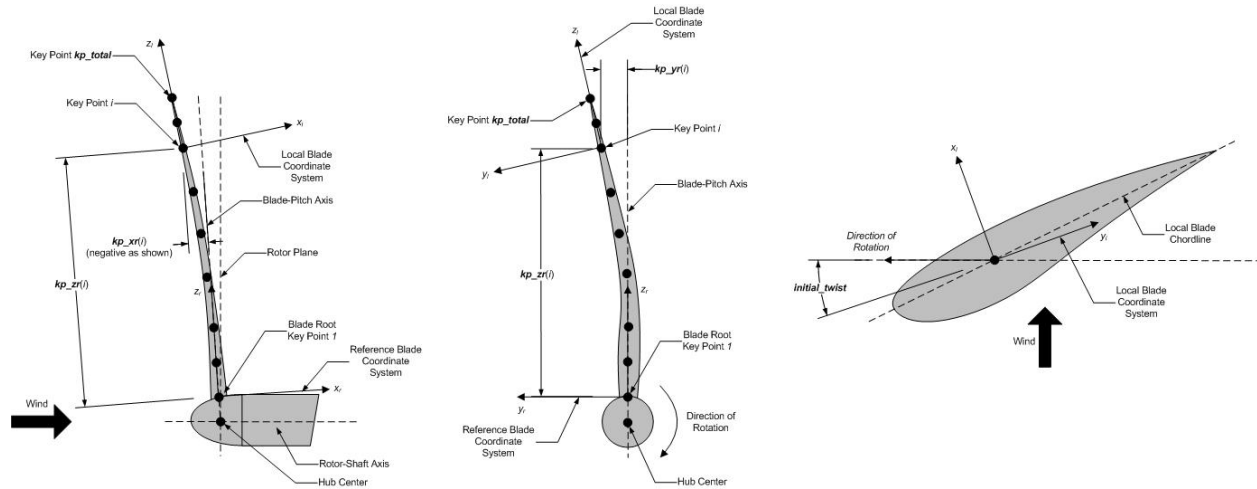


Fig. 6.10: BeamDyn Blade Geometry - Top: Side View; Middle: Front View (Looking Downwind); Bottom: Cross Section View (Looking Toward the Tip, from the Root)

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 6.2.4](#) 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 `node_total` (total number of FE nodes), separated by any combination of commas, semicolons, spaces, and/or tabs. The nodal positions are given in the summary file, if output.

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 6.2.7](#) for a complete list of possible output parameters and their names.

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 6.2.7](#). 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:

$$\underline{\mathcal{F}}^{Damp} = \underline{\mu} \underline{S} \dot{\underline{\epsilon}} \quad (6.5)$$

where $\underline{\mathcal{F}}^{Damp}$ is the damping force, \underline{S} is the 6×6 cross-sectional stiffness matrix, $\dot{\underline{\epsilon}}$ is the strain rate, and $\underline{\mu}$ is the damping coefficient matrix defined as

$$\underline{\mu} = \begin{bmatrix} \mu_{11} & 0 & 0 & 0 & 0 & 0 \\ 0 & \mu_{22} & 0 & 0 & 0 & 0 \\ 0 & 0 & \mu_{33} & 0 & 0 & 0 \\ 0 & 0 & 0 & \mu_{44} & 0 & 0 \\ 0 & 0 & 0 & 0 & \mu_{55} & 0 \\ 0 & 0 & 0 & 0 & 0 & \mu_{66} \end{bmatrix} \quad (6.6)$$

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:

$$\begin{bmatrix} K_{ShrFlp} & 0 & 0 & 0 & 0 & 0 \\ 0 & K_{ShrEdg} & 0 & 0 & 0 & 0 \\ 0 & 0 & EA & 0 & 0 & 0 \\ 0 & 0 & 0 & EI_{Edg} & 0 & 0 \\ 0 & 0 & 0 & 0 & EI_{Flp} & 0 \\ 0 & 0 & 0 & 0 & 0 & GJ \end{bmatrix} \quad (6.7)$$

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:

$$\begin{bmatrix} m & 0 & 0 & 0 & 0 & -mY_{cm} \\ 0 & m & 0 & 0 & 0 & mX_{cm} \\ 0 & 0 & m & mY_{cm} & -mX_{cm} & 0 \\ 0 & 0 & mY_{cm} & i_{Edg} & -i_{cp} & 0 \\ 0 & 0 & -mX_{cm} & -i_{cp} & i_{Flp} & 0 \\ -mY_{cm} & mX_{cm} & 0 & 0 & 0 & i_{plr} \end{bmatrix} \quad (6.8)$$

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)

$$i_{plr} = i_{Edg} + i_{Flp} \quad (6.9)$$

6.2.4 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 6.1](#) in [Section 6.2.5](#).

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.

6.2.5 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 $\underline{\underline{A}}$. Note that sometimes the underlines only denote the dimension of the corresponding matrix.

Coordinate Systems

[Fig. 6.10](#) (in [Section 6.2.3](#)) and [Fig. 6.11](#) show the coordinate system used in BeamDyn.

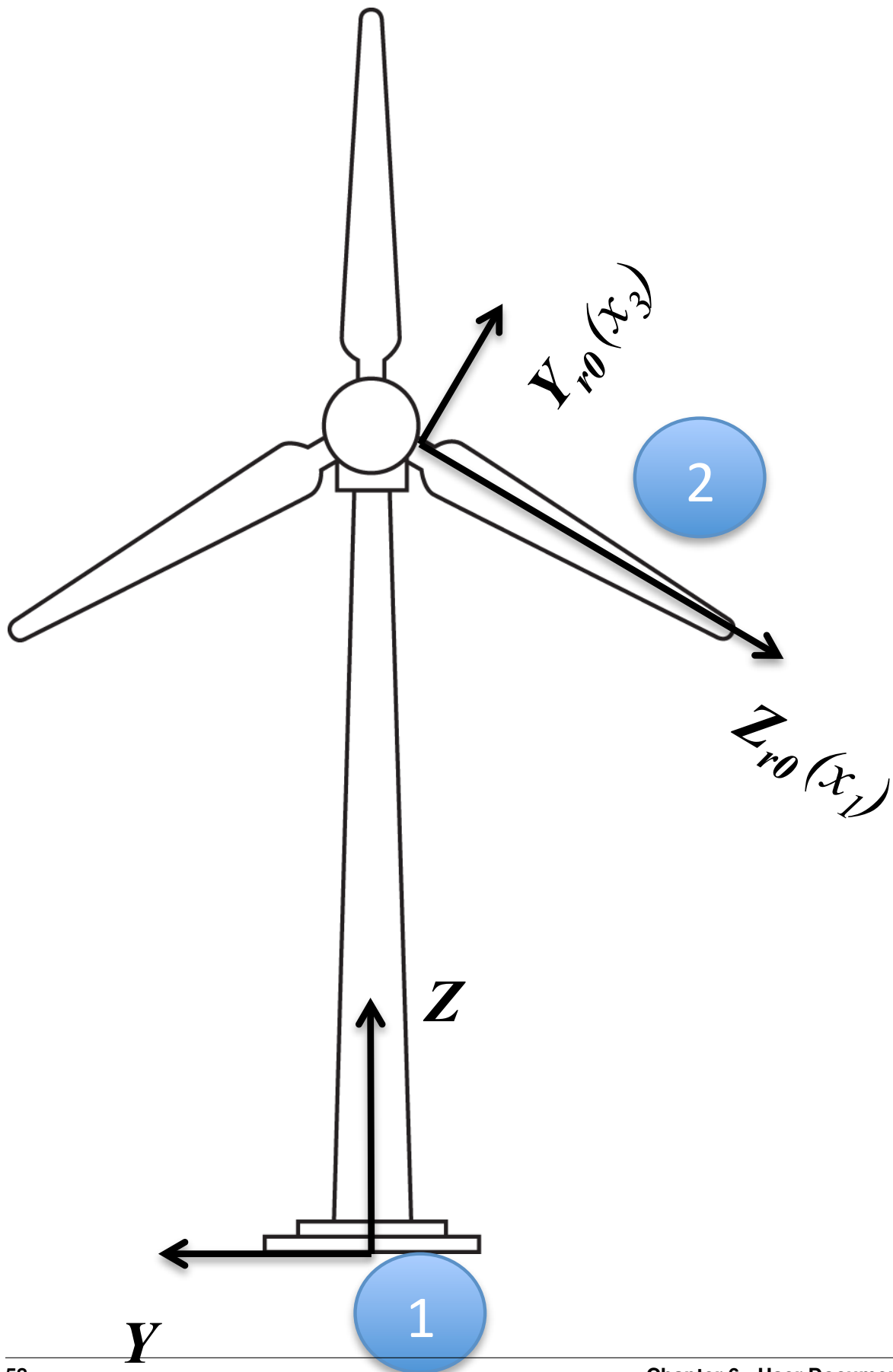


Fig. 6.11: Global, blade reference, and internal coordinate systems in BeamDyn. Illustration by Al Hicks, NREL.

Global Coordinate System

The global coordinate system is denoted as X , Y , and Z in Fig. 6.11. 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. 6.11. 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. 6.11 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 6.1.

Table 6.1: 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 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.

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]

$$\begin{aligned} \dot{\underline{h}} - \underline{F}' &= \underline{f} \\ \dot{\underline{g}} + \dot{\underline{u}}\underline{h} - \underline{M}' + (\tilde{x}'_0 + \tilde{u}')^T \underline{F} &= \underline{m} \end{aligned} \quad (6.10)$$

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 $(\tilde{\bullet})$ defines a skew-symmetric tensor corresponding to the given vector. In the literature, it is also termed as “cross-product matrix”. For example,

$$\tilde{n} = \begin{bmatrix} 0 & -n_3 & n_2 \\ n_3 & 0 & -n_1 \\ -n_2 & n_1 & 0 \end{bmatrix}$$

The constitutive equations relate the velocities to the momenta and the 1D strain measures to the sectional resultants as

$$\begin{Bmatrix} \underline{h} \\ \underline{g} \end{Bmatrix} = \underline{\underline{M}} \begin{Bmatrix} \dot{\underline{u}} \\ \underline{\omega} \end{Bmatrix} \quad (6.11)$$

$$\begin{Bmatrix} \underline{F} \\ \underline{M} \end{Bmatrix} = \underline{\underline{C}} \begin{Bmatrix} \underline{\epsilon} \\ \underline{\kappa} \end{Bmatrix}$$

where $\underline{\underline{M}}$ and $\underline{\underline{C}}$ 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{R} as $\underline{\omega} = \text{axial}(\dot{\underline{R}} \underline{R}^T)$. The axial vector \underline{a} associated with a second-order tensor \underline{A} is denoted $\underline{a} = \text{axial}(\underline{A})$ and its components are defined as

$$\underline{a} = \text{axial}(\underline{A}) = \begin{Bmatrix} a_1 \\ a_2 \\ a_3 \end{Bmatrix} = \frac{1}{2} \begin{Bmatrix} A_{32} - A_{23} \\ A_{13} - A_{31} \\ A_{21} - A_{12} \end{Bmatrix} \quad (6.12)$$

The 1D strain measures are defined as

$$\begin{Bmatrix} \underline{\epsilon} \\ \underline{\kappa} \end{Bmatrix} = \begin{Bmatrix} \underline{x}'_0 + \underline{u}' - (\underline{R} \underline{R}_0) \bar{\underline{r}}_1 \\ \underline{k} \end{Bmatrix} \quad (6.13)$$

where $\underline{k} = \text{axial}[(\underline{R} \underline{R}_0)'(\underline{R} \underline{R}_0)^T]$ is the sectional curvature vector resolved in the inertial basis; \underline{R}_0 is the initial rotation tensor; and $\bar{\underline{r}}_1$ is the unit vector along x_1 direction in the inertial basis. These three sets of equations, including equations of motion Eq. (6.10), constitutive equations Eq. (6.11), and kinematical equations Eq. (6.13), 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-freedom at each node: three displacement components and three rotation components. Here we use \underline{q} to denote the elemental displacement array as $\underline{q} = [\underline{u}^T \ \underline{c}^T]^T$ where \underline{u} is the displacement and \underline{c} is the rotation-parameter vector. The acceleration array can thus be defined as $\underline{a} = [\underline{\ddot{u}}^T \ \underline{\dot{\omega}}^T]^T$. For nonlinear finite-element analysis, the discretized and incremental forms of displacement, velocity, and acceleration are written as

$$\begin{aligned} \underline{q}(x_1) &= \underline{N} \hat{\underline{q}} \quad \Delta \underline{q}^T = [\Delta \underline{u}^T \ \Delta \underline{c}^T] \\ \underline{v}(x_1) &= \underline{N} \hat{\underline{v}} \quad \Delta \underline{v}^T = [\Delta \dot{\underline{u}}^T \ \Delta \dot{\underline{\omega}}^T] \\ \underline{a}(x_1) &= \underline{N} \hat{\underline{a}} \quad \Delta \underline{a}^T = [\Delta \ddot{\underline{u}}^T \ \Delta \dot{\underline{\omega}}^T] \end{aligned} \quad (6.14)$$

where \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

$$\begin{aligned} \underline{u}(\xi) &= h^k(\xi) \hat{\underline{u}}^k \\ \underline{u}'(\xi) &= h^{k'}(\xi) \hat{\underline{u}}^k \end{aligned} \quad (6.15)$$

where $h^k(\xi)$, the component of shape function matrix \underline{N} , is the p^{th} -order polynomial Lagrangian-interpolant shape function of node k , $k = \{1, 2, \dots, p+1\}$, \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

$$\begin{aligned}\underline{c}(\xi) &= h^k(\xi)\hat{c}^k \\ \underline{c}'(\xi) &= h^{k'}(\xi)\hat{c}^k\end{aligned}\tag{6.16}$$

where \underline{c} is the rotation field in an element and \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, \hat{r}^k , by removing the reference rotation, \hat{c}^1 , from the finite rotation at each node, $\hat{r}^k = (\hat{c}^{1-}) \oplus \hat{c}^k$. It is noted that the minus sign on \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 $\underline{R}(\hat{r}^k) = \underline{R}^T(\hat{c}^1) \underline{R}(\hat{c}^k)$.

Step 2: Interpolate the relative-rotation field: $\underline{r}(\xi) = h^k(\xi)\hat{r}^k$ and $\underline{r}'(\xi) = h^{k'}(\xi)\hat{r}^k$. Find the curvature field $\underline{\kappa}(\xi) = \underline{R}(\hat{c}^1)\underline{H}(\underline{r})\underline{r}'$, where \underline{H} is the tangent tensor that relates the curvature vector $\underline{\kappa}$ and rotation vector \underline{c} as

$$\underline{\kappa} = \underline{H} \underline{c}'\tag{6.17}$$

Step 3: Restore the rigid-body rotation removed in Step 1: $\underline{c}(\xi) = \hat{c}^1 \oplus \underline{r}(\xi)$.

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{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. 6.12 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].

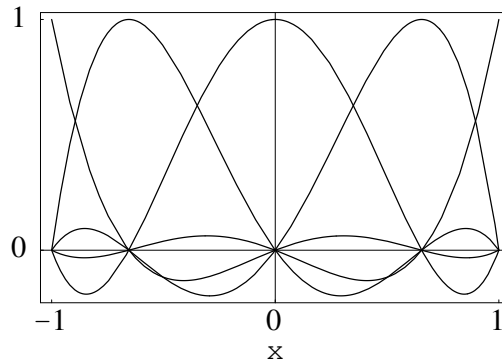


Fig. 6.12: Representative $p+1$ Lagrangian-interpolant shape functions in the element natural coordinates for a fourth-order LSFEs, where nodes are located at the Gauss-Lobatto-Legendre points.

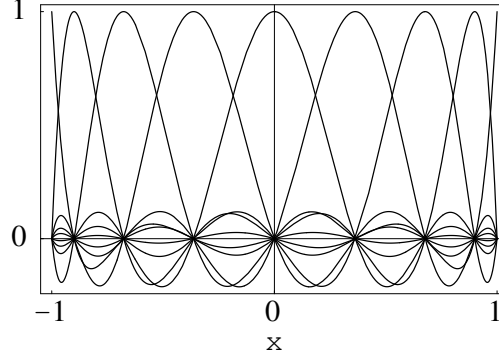


Fig. 6.13: Representative $p + 1$ Lagrangian-interpolant shape functions in the element natural coordinates for a eighth-order LSFs, where nodes are located at the Gauss-Lobatto-Legendre points.

Wiener-Milenković Rotation Parameter

In BeamDyn, the 3D rotations are represented as Wiener-Milenković parameters defined in the following equation:

$$\underline{c} = 4 \tan\left(\frac{\phi}{4}\right) \bar{n} \quad (6.18)$$

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:

$$\underline{r} = \begin{cases} 4(q_0 \underline{p} + p_0 \underline{q} + \tilde{p} \underline{q}) / (\Delta_1 + \Delta_2), & \text{if } \Delta_2 \geq 0 \\ -4(q_0 \underline{p} + p_0 \underline{q} + \tilde{p} \underline{q}) / (\Delta_1 - \Delta_2), & \text{if } \Delta_2 < 0 \end{cases} \quad (6.19)$$

where \underline{p} , \underline{q} , and \underline{r} are the vectorial parameterization of three finite rotations such that $\underline{R}(\underline{r}) = \underline{R}(\underline{p})\underline{R}(\underline{q})$, $p_0 = 2 - \underline{p}^T \underline{p} / 8$, $q_0 = 2 - \underline{q}^T \underline{q} / 8$, $\Delta_1 = (4 - p_0)(4 - q_0)$, 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

$$\underline{R}(\underline{c}) = \frac{1}{(4 - c_0)^2} \begin{bmatrix} c_0^2 + c_1^2 - c_2^2 - c_3^2 & 2(c_1 c_2 - c_0 c_3) & 2(c_1 c_3 + c_0 c_2) \\ 2(c_1 c_2 + c_0 c_3) & c_0^2 - c_1^2 + c_2^2 - c_3^2 & 2(c_2 c_3 - c_0 c_1) \\ 2(c_1 c_3 - c_0 c_2) & 2(c_2 c_3 + c_0 c_1) & c_0^2 - c_1^2 - c_2^2 + c_3^2 \end{bmatrix} \quad (6.20)$$

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

$$\underline{R} = (\underline{DCM})^T \quad (6.21)$$

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. (6.10) are in the form of

$$\underline{\hat{M}}\Delta\hat{a} + \underline{\hat{G}}\Delta\hat{v} + \underline{\hat{K}}\Delta\hat{q} = \underline{\hat{F}}^{ext} - \underline{\hat{F}} \quad (6.22)$$

where the $\underline{\hat{M}}$, $\underline{\hat{G}}$, and $\underline{\hat{K}}$ are the elemental mass, gyroscopic, and stiffness matrices, respectively; $\underline{\hat{F}}$ and $\underline{\hat{F}}^{ext}$ are the elemental forces and externally applied loads, respectively. They are defined for an element of length l along x_1 as follows

$$\begin{aligned} \underline{\hat{M}} &= \int_0^l \underline{N}^T \underline{\mathcal{M}} \underline{N} dx_1 \\ \underline{\hat{G}} &= \int_0^l \underline{N}^T \underline{\mathcal{G}}^I \underline{N} dx_1 \\ \underline{\hat{K}} &= \int_0^l \left[\underline{N}^T (\underline{\mathcal{K}}^I + \underline{\mathcal{Q}}) \underline{N} + \underline{N}^T \underline{\mathcal{P}} \underline{N}' + \underline{N}'^T \underline{\mathcal{C}} \underline{N}' + \underline{N}'^T \underline{\mathcal{Q}} \underline{N} \right] dx_1 \\ \underline{\hat{F}} &= \int_0^l (\underline{N}^T \underline{\mathcal{F}}^I + \underline{N}^T \underline{\mathcal{F}}^D + \underline{N}'^T \underline{\mathcal{F}}^C) dx_1 \\ \underline{\hat{F}}^{ext} &= \int_0^l \underline{N}^T \underline{\mathcal{F}}^{ext} dx_1 \end{aligned} \quad (6.23)$$

where $\underline{\mathcal{F}}^{ext}$ is the applied load vector. The new matrix notations in Eqs. (6.23) to are briefly introduced here. $\underline{\mathcal{F}}^C$ and $\underline{\mathcal{F}}^D$ are elastic forces obtained from Eq. (6.10) as

$$\begin{aligned} \underline{\mathcal{F}}^C &= \left\{ \begin{matrix} \underline{F} \\ \underline{M} \end{matrix} \right\} = \underline{\mathcal{C}} \left\{ \begin{matrix} \underline{\epsilon} \\ \underline{\kappa} \end{matrix} \right\} \\ \underline{\mathcal{F}}^D &= \begin{bmatrix} \underline{0} \\ (\underline{x}'_0 + \underline{u}')^T \end{bmatrix} \underline{\mathcal{F}}^C \equiv \underline{\Upsilon} \underline{\mathcal{F}}^C \end{aligned} \quad (6.24)$$

where $\underline{0}$ denotes a 3×3 null matrix. The $\underline{\mathcal{G}}^I$, $\underline{\mathcal{K}}^I$, $\underline{\mathcal{Q}}$, $\underline{\mathcal{P}}$, $\underline{\mathcal{Q}}$, and $\underline{\mathcal{F}}^I$ in Eqs. (6.23) are defined as

$$\begin{aligned} \underline{\mathcal{G}}^I &= \begin{bmatrix} \underline{0} & (\widetilde{\omega m \eta})^T + \widetilde{\omega m \tilde{\eta}}^T \\ \underline{0} & \widetilde{\omega \underline{\rho}} - \underline{\widetilde{\rho \omega}} \end{bmatrix} \\ \underline{\mathcal{K}}^I &= \begin{bmatrix} \underline{0} & \dot{\widetilde{\omega m \eta}}^T + \widetilde{\omega \dot{\omega m \eta}}^T \\ \underline{0} & \ddot{\widetilde{m \eta}} + \underline{\widetilde{\rho \dot{\omega}}} - \underline{\widetilde{\rho \dot{\omega}}} + \widetilde{\omega \underline{\rho \dot{\omega}}} - \widetilde{\omega \underline{\rho \dot{\omega}}} \end{bmatrix} \\ \underline{\mathcal{Q}} &= \begin{bmatrix} \underline{0} & \underline{\mathcal{C}}_{11} \tilde{E}_1 - \tilde{F} \\ \underline{0} & \underline{\mathcal{C}}_{21} \tilde{E}_1 - \tilde{M} \end{bmatrix} \\ \underline{\mathcal{P}} &= \begin{bmatrix} \underline{0} & \underline{0} \\ \tilde{F} + (\underline{\mathcal{C}}_{11} \tilde{E}_1)^T & (\underline{\mathcal{C}}_{21} \tilde{E}_1)^T \end{bmatrix} \\ \underline{\mathcal{Q}} &= \underline{\Upsilon} \underline{\mathcal{Q}} \\ \underline{\mathcal{F}}^I &= \left\{ \begin{matrix} m \ddot{\underline{u}} + (\dot{\widetilde{\omega}} + \widetilde{\omega \dot{\omega}}) m \underline{\eta} \\ m \tilde{\eta} \ddot{\underline{u}} + \underline{\widetilde{\rho \dot{\omega}}} + \widetilde{\omega \underline{\rho \dot{\omega}}} \end{matrix} \right\} \end{aligned} \quad (6.25)$$

where m is the mass density per unit length, $\underline{\eta}$ is the location of the sectional center of mass, $\underline{\rho}$ is the moment of inertia tensor, and the following notations were introduced to simplify the above expressions

$$\begin{aligned} \underline{E}_1 &= \underline{x}'_0 + \underline{u}' \\ \underline{\mathcal{C}} &= \begin{bmatrix} \underline{\mathcal{C}}_{11} & \underline{\mathcal{C}}_{12} \\ \underline{\mathcal{C}}_{21} & \underline{\mathcal{C}}_{22} \end{bmatrix} \end{aligned} \quad (6.26)$$

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

$$\underline{f}_d = \underline{\mu} \underline{C} \begin{Bmatrix} \dot{\underline{e}} \\ \dot{\underline{\kappa}} \end{Bmatrix} \quad (6.27)$$

where $\underline{\mu}$ 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

$$\begin{aligned} \underline{\mathcal{F}}^C &= \begin{Bmatrix} \underline{F}_d \\ \underline{M}_d \end{Bmatrix} \\ \underline{\mathcal{F}}^D &= \begin{Bmatrix} \underline{0} \\ (\tilde{x}'_0 + \tilde{u}')^T \underline{F}_d \end{Bmatrix} \end{aligned} \quad (6.28)$$

The linearization of the structural damping forces are as follows:

$$\begin{aligned} \Delta \underline{\mathcal{F}}^C &= \underline{\mathcal{S}}_d \begin{Bmatrix} \Delta \underline{u}' \\ \Delta \underline{c}' \end{Bmatrix} + \underline{\mathcal{Q}}_d \begin{Bmatrix} \Delta \underline{u} \\ \Delta \underline{c} \end{Bmatrix} + \underline{\mathcal{G}}_d \begin{Bmatrix} \Delta \dot{\underline{u}} \\ \Delta \dot{\underline{\omega}} \end{Bmatrix} + \underline{\mu} \underline{C} \begin{Bmatrix} \Delta \dot{\underline{u}}' \\ \Delta \dot{\underline{\omega}}' \end{Bmatrix} \\ \Delta \underline{\mathcal{F}}^D &= \underline{\mathcal{P}}_d \begin{Bmatrix} \Delta \underline{u}' \\ \Delta \underline{c}' \end{Bmatrix} + \underline{\mathcal{Q}}_d \begin{Bmatrix} \Delta \underline{u} \\ \Delta \underline{c} \end{Bmatrix} + \underline{\mathcal{X}}_d \begin{Bmatrix} \Delta \dot{\underline{u}} \\ \Delta \dot{\underline{\omega}} \end{Bmatrix} + \underline{\mathcal{Y}}_d \begin{Bmatrix} \Delta \dot{\underline{u}}' \\ \Delta \dot{\underline{\omega}}' \end{Bmatrix} \end{aligned} \quad (6.29)$$

where the newly introduced matrices are defined as

$$\begin{aligned} \underline{\mathcal{S}}_d &= \underline{\mu} \underline{C} \begin{bmatrix} \tilde{\omega}^T & \underline{0} \\ \underline{0} & \tilde{\omega}^T \end{bmatrix} \\ \underline{\mathcal{Q}}_d &= \begin{bmatrix} \underline{0} & \underline{\mu} \underline{C}_{11} (\dot{\underline{u}}' - \tilde{\omega} \tilde{E}_1) - \tilde{F}_d \\ \underline{0} & \underline{\mu} \underline{C}_{21} (\dot{\underline{u}}' - \tilde{\omega} \tilde{E}_1) - \tilde{M}_d \end{bmatrix} \\ \underline{\mathcal{G}}_d &= \begin{bmatrix} \underline{0} & \underline{C}_{11}^T \underline{\mu}^T \tilde{E}_1 \\ \underline{0} & \underline{C}_{12}^T \underline{\mu}^T \tilde{E}_1 \end{bmatrix} \\ \underline{\mathcal{P}}_d &= \begin{bmatrix} \underline{0} & \underline{0} \\ \tilde{F}_d + \tilde{E}_1^T \underline{\mu} \underline{C}_{11} \tilde{\omega}^T & \tilde{E}_1^T \underline{\mu} \underline{C}_{12} \tilde{\omega}^T \end{bmatrix} \\ \underline{\mathcal{Q}}_d &= \begin{bmatrix} \underline{0} & \underline{0} \\ \underline{0} & \tilde{E}_1^T \underline{O}_{12} \end{bmatrix} \\ \underline{\mathcal{X}}_d &= \begin{bmatrix} \underline{0} & \underline{0} \\ \underline{0} & \tilde{E}_1^T \underline{G}_{12} \end{bmatrix} \\ \underline{\mathcal{Y}}_d &= \begin{bmatrix} \underline{0} & \underline{0} \\ \tilde{E}_1^T \underline{\mu} \underline{C}_{11} & \tilde{E}_1^T \underline{\mu} \underline{C}_{12} \end{bmatrix} \end{aligned} \quad (6.30)$$

where \underline{O}_{12} and \underline{G}_{12} are the 3×3 sub matrices of $\underline{\mathcal{Q}}$ and $\underline{\mathcal{G}}$ as \underline{C}_{12} in Eq. (6.26).

Convergence Criterion and Generalized- α Time Integrator

The system of nonlinear equations in Eqs. (6.10) are solved using the Newton-Raphson method with the linearized form in Eq. (6.22). In the present implementation, an energy-like stopping criterion has been chosen, which is calculated as

$$\|\Delta \mathbf{U}^{(i)T} ({}^{t+\Delta t} \mathbf{R} - {}^{t+\Delta t} \mathbf{F}^{(i-1)})\| \leq \|\epsilon_E (\Delta \mathbf{U}^{(1)T} ({}^{t+\Delta t} \mathbf{R} - {}^t \mathbf{F}))\| \quad (6.31)$$

where $\|\cdot\|$ denotes the Euclidean norm, $\Delta\mathbf{U}$ 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. (6.10) can be recast in a compact form

$$\underline{\mathcal{F}}^I - \underline{\mathcal{F}}^{C'} + \underline{\mathcal{F}}^D = \underline{\mathcal{F}}^{ext} \quad (6.32)$$

with all the vectors defined in Section [sec:LinearProcess]. At the blade root, the governing equation is revised as

$$\underline{\mathcal{F}}^I - \underline{\mathcal{F}}^{C'} + \underline{\mathcal{F}}^D = \underline{\mathcal{F}}^{ext} + \underline{\mathcal{F}}^R \quad (6.33)$$

where $\underline{\mathcal{F}}^R = [\underline{F}^R \quad \underline{M}^R]^T$ is the reaction force vector and it can be solved from Eq. (6.33) 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. (6.32) are recast as

$$\underline{\mathcal{F}}^A + \underline{\mathcal{F}}^V - \underline{\mathcal{F}}^{C'} + \underline{\mathcal{F}}^D = \underline{\mathcal{F}}^{ext} \quad (6.34)$$

where the inertial force vector $\underline{\mathcal{F}}^I$ is split into $\underline{\mathcal{F}}^A$ and $\underline{\mathcal{F}}^V$:

$$\begin{aligned} \underline{\mathcal{F}}^A &= \begin{Bmatrix} m\ddot{\underline{u}} + \dot{\omega}m\eta \\ m\ddot{\eta}\dot{\underline{u}} + \underline{\rho}\dot{\underline{\omega}} \end{Bmatrix} \\ \underline{\mathcal{F}}^V &= \begin{Bmatrix} \dot{\omega}\dot{\omega}m\eta \\ \dot{\omega}\underline{\rho}\dot{\underline{\omega}} \end{Bmatrix} \end{aligned} \quad (6.35)$$

The blade loads are thus defined as

$$\underline{\mathcal{F}}^{BF} \equiv \underline{\mathcal{F}}^V - \underline{\mathcal{F}}^{C'} + \underline{\mathcal{F}}^D \quad (6.36)$$

We note that if structural damping is considered in the analysis, the $\underline{\mathcal{F}}_d^C$ and $\underline{\mathcal{F}}_d^D$ are incorporated into the internal elastic forces, $\underline{\mathcal{F}}^C$ and $\underline{\mathcal{F}}^D$, for calculation.

6.2.6 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.

6.2.7 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 dynamic example), (NREL 5MW static example): This file includes information on the analysis type (static vs. dynamic), 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 calls the appropriate inputs and primary input file (i.e., here one 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. $N\beta$, refers to output node β , where β is a number in the range [1,9], corresponding to entry β in the `OutNd` list. 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.

6.3 HydroDyn Users Guide and Theory Manual

6.3.1 Introduction

HydroDyn is a time-domain hydrodynamics module that has been coupled into the OpenFAST wind turbine multi-physics engineering tool to enable aero-hydro-servo-elastic simulation of offshore wind turbines. HydroDyn is applicable to both fixed-bottom and floating offshore substructures. This latest release of HydroDyn follows the requirements of the OpenFAST modularization framework and couples to OpenFAST. HydroDyn can also be driven as a standalone code to compute hydrodynamic loading uncoupled from OpenFAST.

HydroDyn allows for multiple approaches for calculating the hydrodynamic loads on a structure: a potential-flow theory solution, a strip-theory solution, or a hybrid combination of the two. Waves generated internally within HydroDyn can be regular (periodic) or irregular (stochastic) and long-crested (unidirectional) or short-crested (with wave energy spread across a range of directions). Wave elevations or full wave kinematics can also be generated externally and used within HydroDyn. Internally, HydroDyn generates waves analytically for finite depth using first-order (linear Airy) or first- plus second-order wave theory [Sharma and Dean, 1981] with the option to include directional spreading. Wave kinematics are computed in the domain between the flat seabed and still-water level (SWL). Optionally, wave stretching allows for the wave kinematics and hydrodynamic loads to be computed at all nodes within the fluid domain up to the instantaneous free surface (above SWL in a wave crest and below SWL in a wave trough), unlike models without wave stretching, which compute wave kinematics and loads at nodes between the seabed and SWL regardless of the instantaneous free surface. The second-order hydrodynamic implementations include time-domain calculations of difference- (mean- and slow-drift-) and sum-frequency terms. To minimize computational expense, Fast Fourier Transforms (FFTs) are applied in the summation of all wave frequency components.

The potential-flow solution is applicable to substructures or members of substructures that are large relative to a typical wavelength. The potential-flow solution involves either frequency-to-time-domain transforms or fluid-impulse theory (FIT). In the former, potential-flow hydrodynamic loads include linear hydrostatic restoring, the added mass and damping contributions from linear wave radiation (including free-surface memory effects), and the incident-wave excitation from first- and second-order diffraction (Froude-Kriloff and scattering). The hydrodynamic coefficients (first and second order) required for the potential-flow solution are frequency dependent and must be supplied by a separate frequency-domain panel code (e.g., WAMIT) from a pre-computation step. The radiation memory effect can be calculated either through direct time-domain convolution or through a linear state-space approach, with a state-space model derived through the `SS_Fitting` preprocessor. The second-order terms can be derived from the full difference- and sum-frequency quadratic transfer functions (QTFs) or the difference-frequency terms can be estimated via Standing et al.'s extension to Newman's approximation, based only on first-order coefficients. The use of FIT is not yet documented in this manual.

The strip-theory solution may be preferable for substructures or members of substructures that are small in diameter relative to a typical wavelength. Strip-theory hydrodynamic loads can be applied across multiple interconnected members, each with possible incline and taper, and are derived directly from the undisturbed wave and current kinematics at the undisplaced position of the substructure. The strip-theory loads include the relative form of Morison's equation for the distributed fluid-inertia, added-mass, and viscous-drag components. Additional distributed load components include axial loads from tapered members and static buoyancy loads. Hydrodynamic loads are also applied as lumped loads on member endpoints (called joints). It is also possible to include flooding or ballasting of members, and the effects of marine growth. The hydrodynamic coefficients required for this solution come through user-specified dynamic-pressure, added-mass, and viscous-drag coefficients.

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. 6.14: BeamDyn Output Channel List

For some substructures and sea conditions, the hydrodynamic loads from a potential-flow theory should be augmented with the loads brought about by flow separation. For this, the viscous-drag component of the strip-theory solution may be included with the potential-flow theory solution. Another option available is to supply a global damping matrix (linear or quadratic) to the system to represent this effect. When HydroDyn is coupled to OpenFAST, HydroDyn receives the position, orientation, velocities, and accelerations of the (rigid or flexible) substructure at each coupling time step and then computes the hydrodynamic loads and returns them back to OpenFAST. At this time, OpenFAST's ElastoDyn structural-dynamics module assumes for a floating platform that the substructure (floating platform) is a six degree-of-freedom (DOF) rigid body. For fixed-bottom offshore wind turbines, OpenFAST's SubDyn module allows for structural flexibility of multi-member substructures and the coupling to HydroDyn includes hydro-elastic effects.

The primary HydroDyn input file defines the substructure geometry, hydrodynamic coefficients, incident wave kinematics and current, potential-flow solution options, flooding/ballasting and marine growth, and auxiliary parameters. The geometry of strip-theory members is defined by joint coordinates of the undisplaced substructure in the global reference system, with the origin at the intersection of the undeflected tower centerline with mean sea level (MSL). A member connects two joints; multiple members can use a common joint. The hydrodynamic loads are computed at nodes, which are the resultant of member refinement into multiple (MDivSize input) elements (nodes are located at the ends of each element), and they are calculated by the module. Member properties include outer diameter, thickness, and dynamic-pressure, added-mass and viscous-drag coefficients. Member properties are specified at the joints; if properties change from one joint to the other, they will be linearly interpolated for the inner nodes.

[Section 6.3.2](#) describes the HydroDyn input files. [Section 6.3.3](#) discusses the output files generated by HydroDyn; these include echo files, wave-elevation outputs, a summary file, and the results file. [Section 6.3.4](#) provides modeling guidance when using HydroDyn. The HydroDyn theory is covered in [Section 6.3.5](#), and [Section 6.3.6](#) contains a list of references. Example input files are included in [Section 6.3.7](#). A summary of available output channels are found [Section 6.3.7](#).

6.3.2 Input Files

The user configures the hydrodynamic model parameters via a primary HydroDyn input file, as well as separate input files for the potential flow model and state-space radiation model. When used in standalone mode, an additional driver input file is required. This driver file specifies initialization inputs normally provided to HydroDyn by OpenFAST, as well as the per-time-step inputs to HydroDyn.

As an example, the `driver.dvr` file is the main driver, the `input.dat` is the primary input file. If a potential flow model is being used, WAMIT output files with a common root name, for example "platform", are required. These files contain the linear, nondimensionalized, hydrostatic restoring matrix (.hst extension), the frequency-dependent hydrodynamic added mass matrix and damping matrix (.1 extension), and the frequency- and direction-dependent wave excitation force vector per unit wave amplitude (.3 extension). If the state-space radiation model is being used, a `platform.ss` file contains the state-space radiation model for the platform. Example input files are included in [Section 6.3.7](#).

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

Units

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

HydroDyn Driver Input File

The driver input file is only needed for the standalone version of HydroDyn and contains inputs normally generated by OpenFAST, and necessary to control the hydrodynamic simulation for uncoupled models. A sample HydroDyn driver input file is given in [Section 6.3.7](#).

Set the `Echo` flag in this file to `TRUE` if you wish to have the `HydroDyn_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 `OutRootName.dvr.ech`, where `OutRootName` is specified in the `HYDRODYN` section of the driver input file. Set the gravity constant using the `Gravity` parameter. `HydroDyn` expects a magnitude, so in SI units this would be set to $9.80665 \frac{m}{s^2}$. `HDInputFile` is the filename of the primary `HydroDyn` input file. This name should be in quotations and can contain an absolute path or a relative path. All `HydroDyn`-generated output files will be prefixed with `OutRootName`. If this parameter includes a file path, the output will be generated in that folder. `NSteps` specifies the number of simulation time steps, and `TimeInterval` specifies the time between steps. Setting `WAMITInputsMod = 0` forces all WAMIT reference point (WRP) input motions to zero for all time. If you set `WAMITInputsMod = 1`, then you must set the steady-state inputs in the `WAMIT STEADY STATE INPUTS` section of the file. Setting `WAMITInputsMod = 2`, requires the time-series input file whose name is specified via the `WAMITInputsFile` parameter. The WAMIT inputs file is a text-formatted file. This file has no header lines. Each data row corresponds to a given time step, and the whitespace separated columns of floating point values represent the necessary motion inputs as follows:

- Column 1 : Time step value (s)
- Columns 2-4 : Translational displacements along X, Y, and Z (m)
- Columns 5-7 : Rotational displacements about X, Y, and Z [small angle] ($radians$)
- Columns 8-10: Translational velocities along X, Y, and Z ($\frac{m}{s}$)
- Columns 11-13: Rotational velocities about X, Y, and Z ($\frac{radians}{s}$)
- Columns 14-16: Translational accelerations along X, Y, and Z ($\frac{m}{s^2}$)
- Columns 17-19: Rotational accelerations about X, Y, and Z ($\frac{radians}{s^2}$)

All motions are specified in the global inertial-frame coordinate system.

In a similar fashion, the input motions for the Morison members (strip-theory model) are set to zero if `MorisonInputsMod = 0`. If you select `MorisonInputsMod = 1` then the motions at each substructure joint are set to the steady-state values given in the `MORISON STEADY STATE INPUTS` section. Currently, option 2 is unavailable for the Morison inputs.

The standalone `HydroDyn` does not check for physical consistency between motions specified for the WRP and Morison members in the driver file.

Setting `WaveElevSeriesFlag` to `TRUE` enables the outputting of a grid of wave elevations to a text-based file with the name `OutRootName.WaveElev.out`. The grid consists of `WaveElevNX` by `WaveElevNY` wave elevations (centered at $X = 0, Y = 0$ i.e., $(0,0)$) with a dX and dY spacing in the global inertial-frame coordinate system. These wave elevations are distinct and output separately from the wave elevations determined by `NWaveElev` in the `HydroDyn` primary input file, such that the total number of wave elevation outputs is `NWaveElev + (WaveElevNX × WaveElevNY)`. The wave-elevation output file `OutRootName.WaveElev.out` contains the total wave elevation, which is the sum of the first- and second-order terms (when the second-order wave kinematics are optionally enabled).

HydroDyn Primary Input File

The `HydroDyn` input file defines the substructure geometry, hydrodynamic coefficients, incident wave kinematics and current, potential-flow solution options, flooding/ballasting and marine growth, and auxiliary parameters. The geometry of strip-theory members is defined by joint coordinates of the undisplaced substructure in the global reference system, with the origin at the intersection of the undeflected tower centerline with MSL. A member connects two joints; multiple members can use a common joint. The hydrodynamic loads are computed at nodes, which are the resultant of member refinement into multiple (`MDivSize` input) elements (nodes are located at the ends of each element), and they are calculated by the module. Member properties include outer diameter, thickness, and dynamic-pressure, added-mass and viscous-drag coefficients. Member properties are specified at the joints; if properties change

from one joint to the other, they will be linearly interpolated for the inner nodes. The file is organized into several functional sections. Each section corresponds to an aspect of the hydrodynamics model or the submerged substructure. A sample HydroDyn primary input file is given in [Section 6.3.7](#).

If this manual refers to an ID in a table entry, this is an integer identifier for the table entry, and these IDs do not need to be consecutive or increasing, but they must be unique for a given table entry. The input file begins with two lines of header information which is for your use, but is not used by the software. On the next line, set the `Echo` flag to `TRUE` if you wish to have HydroDyn echo the contents of the HydroDyn input file (useful for debugging errors in the input file). The echo file has the naming convention of `OutRootName.HD.ech`. `OutRootName` is either specified in the `HYDRODYN` section of the driver input file when running HydroDyn standalone, or by OpenFAST when running a coupled simulation.

Environmental Conditions

`WtrDens` specifies the water density and must be a value greater than or equal to zero; a typical value of seawater is around 1025 kg/m³. `WtrDpth` specifies the water depth (depth of the flat seabed), based on the reference MSL, and must be a value greater than zero. `MSL2SWL` is the offset between the MSL and SWL, positive upward. This parameter is useful when simulating the effect of tides or storm-surge sea-level variations without having to alter the substructure geometry information. This parameter is unused with `WaveMod` = 6 and must be set to zero if you are using a potential-flow model (`PotMod` = 1 or 2).

Waves

The `WAVES` section of the input file controls the internal generation of first-order waves or the use of externally generated waves, used by both the strip-theory and potential-flow solutions. The wave spectrum settings in this section only pertain to the first-order wave frequency components. When second-order terms are optionally enabled—see the `2ND-ORDER WAVES` and `2ND-ORDER FLOATING PLATFORM FORCES` sections below—the second-order terms are calculated using the first-order wave-component amplitudes and extra energy is added to the wave spectrum (at the difference and sum frequencies).

`WaveMod` specifies the incident wave kinematics model. The options are: * 0: none = still water * 1: regular (periodic) waves * 1P#: regular (periodic) waves with user-specified phase, for example 1P20.0 for regular waves with a 20 phase (without P#, the phase will be random, based on `WaveSeed`); 0 phase represents a cosine function, starting at the peak and decreasing in time * 2: Irregular (stochastic) waves based on the JONSWAP or Pierson-Moskowitz frequency spectrum * 3: Irregular (stochastic) waves based on a white-noise frequency spectrum * 4: Irregular (stochastic) waves based on a user-defined frequency spectrum from routine `UserWaveSpcrm()`; see Appendix D for compiling instructions * 5: Externally generated wave-elevation time series * 6: Externally generated full wave-kinematics time series

Option 4 requires that the `UserWaveSpcrm()` subroutine of the `Waves.f90` source file be implemented by the user, and will require recompiling either the standalone HydroDyn program or OpenFAST. Option 5 allows the use of externally generated wave-elevation time series, from which the hydrodynamic loads in the potential-flow solution or the wave kinematics used in the strip-theory solution are derived internally. Option 6 allows the use of full externally generated wave kinematics for use with the strip-theory solution (but not the potential-flow solution). With options 5 and 6, the externally generated wave data is provided through input files, all of which have the root name given by the `WvKinFile` parameter below.

This version does not include the ability to model stretching of internally generated incident wave kinematics to the instantaneous free surface; you must set `WaveStMod` = 0.

`WaveTMax` sets the length of the incident wave kinematics time series, but it also determines the frequency step used in the inverse FFT, from which the internal wave time series are derived ($\Delta\omega = 2\pi/\text{WaveTMax}$). If `WaveTMax` is less than the total simulation time, HydroDyn implements repeating wave kinematics that have a period of `WaveTMax`; `WaveTMax` must not be less than the total simulation time when `WaveMod` = 5. `WaveDT` determines the time step for the wave kinematics time series, but it also determines the maximum frequency in the inverse FFT ($\omega_{\text{max}} =$

π/WaveDT). When modeling irregular sea states, we recommend that `WaveTMax` be set to at least 1 hour (3600 s) and that `WaveDT` be a value in the range between 0.1 and 1.0 s to ensure sufficient resolution of the wave spectrum and wave kinematics. When HydroDyn is coupled to OpenFAST, `WaveDT` may be specified arbitrarily independently from the glue code time step of OpenFAST (the wave kinematics will be interpolated in time as necessary); `WaveDT` must equal the glue code time step of OpenFAST when `WaveMod` = 6.

For internally generated waves, the wave height (crest-to-trough, twice the amplitude) for regular waves and the significant wave height for irregular waves is set using `WaveHs` (only used when `WaveMod` = 1, 2, or 3). The wave period for regular waves and the peak-spectral wave period for irregular waves is controlled with the `WaveTp` parameter (only used when `WaveMod` = 1 or 2). `WavePkShp` is the peak-shape parameter of JONSWAP irregular wave spectrum (only used when `WaveMod` = 2). Set `WavePkShp` to DEFAULT to obtain the value recommended in the IEC 61400-3 Annex B, derived based on the peak-spectral period and significant wave height [IEC, 2009]. Set `WavePkShp` to 1.0 for the Pierson-Moskowitz spectrum.

`WvLowCoff` and `WvHiCoff` control the lower and upper cut-off frequencies (in rad/s) of the first-order wave spectrum; the first-order wave-component amplitudes are zeroed below and above these cut-off frequencies, respectively. `WvLowCoff` may be set lower than the low-energy limit of the first-order wave spectrum to minimize computational expense. Setting a proper upper cut-off frequency (`WvHiCoff`) also minimizes computational expense and is important to prevent nonphysical effects when approaching of the breaking-wave limit and to avoid nonphysical wave forces at high frequencies (i.e., at short wavelengths) when using a strip-theory solution. `WvLowCoff` and `WvHiCoff` are unused when `WaveMod` = 0, 1, or 6.

`WaveDir` (unused when `WaveMod` = 0 or 6) is the mean wave propagation heading direction (in degrees), and must be in the range $[-180, 180]$. A heading of 0 corresponds to wave propagation in the positive X-axis direction. And a heading of 90 corresponds to wave propagation in the positive Y-axis direction. `WaveDirMod` specifies the wave directional spreading model (only used when `WaveMod` = 2, 3, or 4). Setting `WaveDirMod` to 0 disables directional spreading, resulting in long-crested (plane-progressive) sea states propagating in the `WaveDir` direction. Setting `WaveDirMod` to 1 enables the modeling of short-crested sea states, with a mean propagation direction of `WaveDir`, through the commonly used cosine spreading function (COS2S) to define the directional spreading spectrum, based on the spreading coefficient (S) defined via `WaveDirSpread`. The wave directional spreading spectrum is discretized with an equal-energy method using `WaveNDir` number of equal-energy bins. `WaveNDir` is an odd-valued integer greater or equal to 1 (1 or 3 or 5...), but HydroDyn may slightly increase the specified value of `WaveNDir` to ensure that there is the same number of wave components within each direction bin; setting `WaveNDir` = 1 is equivalent to setting `WaveDirMod` = 0. The range of the directional spread (in degrees) is defined via `WaveDirSpread`. The equal-energy method assumes that the directional spreading spectrum is the product of a frequency spectrum and a spreading function i.e. $S(\omega, \beta) = S(\omega)D(\beta)$. Directional spreading is not permitted when using Newman's approximation of the second-order difference-frequency potential-flow loads.

`WaveSeed(1)` and `WaveSeed(2)` (unused when `WaveMod` = 0, 5, or 6) combined determine the initial seed (starting point) for the internal pseudorandom number generator needed to derive the internal wave kinematics from the wave frequency and direction spectra. If you want to run different time-domain realizations for given boundary conditions (of significant wave height, and peak-spectral period, etc.), you should change one or both seeds between simulations. While the phase of each wave frequency and direction component of the wave spectrum is always based on a uniform distribution (except when using the 1P# `WaveMod` option), the amplitude of the wave frequency spectrum can also be randomized (following a normal distribution) by setting `WaveNDamp` to TRUE. Setting `WaveNDamp` to FALSE means that the amplitude of the wave frequency spectrum always matches the target spectrum. `WaveNDamp` is only used with `WaveMod` = 2, 3, or 4.

When using externally generated wave data (`WaveMod` = 5 or 6), input parameter `WvKinFile` should be set to the root name of the input file(s) (without extension) containing the data.

Using externally generated wave-elevation time series (`WaveMod` = 5) requires a text-formatted input data file with the extension `.Elev` containing two columns of data—the first is time (starting at zero) (in s) and the second is the wave elevation at (0,0) (in m), separated by whitespace. Header lines (identified as those not beginning with a number) are ignored. The time series must be at least `WaveTMax` in length and not less than the total simulation time and the time step must match `WaveDT`. The wave-elevation time series specified is assumed to be of first order and long-crested, but is not checked for physical correctness. When second-order terms are optionally enabled—see the 2ND-ORDER

WAVES and 2ND-ORDER FLOATING PLATFORM FORCES sections below—the second-order terms are calculated using the wave-component amplitudes derived from the provided wave-elevation time series and extra energy is added to the wave spectrum (at the difference and sum frequencies).

Using full externally generated wave kinematics (`WaveMod = 6`) requires eight text-formatted input data files, all without headers. Seven files with extensions `.Vxi`, `.Vyi`, `.Vzi`, `.Axi`, `.Ayi`, `.Azi`, and `.DynP` correspond to the X, Y, and Z velocities (in m/s) and accelerations (in m/s^2) in the global inertial-frame coordinate system and the dynamic pressure (in Pa) time series. Each of these files must have exactly `WaveTMax/DT` rows and N whitespace-separated columns, where N

is the total number of internal HydroDyn analysis nodes (corresponding exactly to those written to the HydroDyn summary file). Time is absent from the files, but is assumed to go from zero to `WaveTMax - WaveDT` in steps of `WaveDT`. To use this feature, it is the burden of the user to generate wave kinematics data at each of HydroDyn's time steps and analysis nodes. HydroDyn will not interpolate the data; as such, when HydroDyn is coupled to OpenFAST, `WaveDT` must equal the glue code time step of OpenFAST. A numerical value (including 0) in a file is assumed to be valid data (with 0 corresponding to 0 m/s, 0 m/s^2 , or 0 Pa); a nonnumeric string will designate that the node is outside of the water at that time step (above the instantaneous water elevation or below the seabed)—externally generated wave kinematics used with `WaveMod = 6` are not limited to the domain between a flat seabed and SWL and may consider wave stretching, higher-order wave theories, or an uneven seabed. All seven files must have nonnumeric strings in the same locations within the file. The eighth file, with extension `.Elev`, must contain the wave elevation (in m) at each of the `NWaveElev` points on the SWL where wave elevations can be output—see below; this data is required for output purposes only and is not used by HydroDyn for other means. This file must have exactly `WaveTMax/DT` rows and `NWaveElev` whitespace-separated columns and only valid numeric data is allowed (the file will have `NWaveElev + (WaveElevNX × WaveElevNY)` columns when HydroDyn is operated in standalone mode). The data in these files is not processed (filtered, etc.) or checked for physical correctness (other than for consistency in the location of the nonnumeric strings). Full externally generated wave kinematics (`WaveMod = 6`) cannot be used in conjunction with the potential-flow solution.

You can generate up to 9 wave elevation outputs. `NWaveElev` determines the number (between 0 and 9), and the whitespace-separated lists of `WaveElevxi` and `WaveElevyi` determine the locations of these `NWaveElev` number of points on the SWL plane in the global inertial-frame coordinate system.

2nd-Order Waves

The 2ND-ORDER WAVES section (unused when `WaveMod = 0` or 6) of the input file allows the option of adding second-order contributions to the wave kinematics used by the strip-theory solution. When second-order terms are optionally enabled, the second-order terms are calculated using the first-order wave-component amplitudes and extra energy is added to the wave spectrum (at the difference and sum frequencies). The second-order terms cannot be computed without also including the first-order terms from the WAVES section above. Enabling the second-order terms allows one to capture some of the nonlinearities of real surface waves, permitting more accurate modeling of sea states and the associated wave loads at the expense of greater computational effort (mostly at HydroDyn initialization).

While the cut-off frequencies in this section apply to both the second-order wave kinematics used by strip theory and the second-order diffraction loads in potential-flow theory, the second-order terms themselves are enabled separately. The second-order wave kinematics used by strip theory are enabled in this section while the second-order diffraction loads in potential-flow theory are enabled in the 2ND-ORDER FLOATING PLATFORM FORCES section below. While the second-order effects are included when enabled, the wave elevations output from HydroDyn will only include the second-order terms when the second-order wave kinematics are enabled in this section.

To use second-order wave kinematics in the strip-theory solution, set `WvDiffQTF` and/or `WvSumQTF` to TRUE. When `WvDiffQTF` is set to TRUE, second-order difference-frequency terms, calculated using the full difference-frequency QTF, are incorporated in the wave kinematics. When `WvSumQTF` is set to TRUE, second-order sum-frequency terms, calculated using the full sum-frequency QTF, are incorporated in the wave kinematics. The full difference- and sum-frequency wave kinematics QTFs are implemented analytically following [Sharma and Dean, 1981], which extends Stokes second-order theory to irregular multidirectional waves. A setting of FALSE disregards the second-order contributions to the wave kinematics in the strip-theory solution.

`WvLowCoffD` and `WvHiCoffD` control the lower and upper cut-off frequencies (in rad/s) of the second-order difference-frequency terms; the second-order difference-frequency terms are zeroed below and above these cut-off frequencies, respectively. The cut-offs apply directly to the physical difference frequencies, not the two individual first-order frequency components of the difference frequencies. When enabling second-order potential-flow theory, a setting of `WvLowCoffD = 0` is advised to avoid eliminating the mean-drift term (second-order wave kinematics do not have a nonzero mean). `WvHiCoffD` need not be set higher than the peak-spectral frequency of the first-order wave spectrum ($\omega_p = 2\pi/\text{WaveTp}$) to minimize computational expense.

Likewise, `WvLowCoffS` and `WvHiCoffS` control the lower and upper cut-off frequencies (in rad/s) of the second-order sum-frequency terms; the second-order sum-frequency terms are zeroed below and above these cut-off frequencies, respectively. The cut-offs apply directly to the physical sum frequencies, not the two individual first-order frequency components of the sum frequencies. `WvLowCoffS` need not be set lower than the peak-spectral frequency of the first-order wave spectrum ($\omega_p = 2\pi/\text{WaveTp}$) to minimize computational expense. Setting a proper upper cut-off frequency (`WvHiCoffS`) also minimizes computational expense and is important to (1) ensure convergence of the second-order summations, (2) avoid unphysical “bumps” in the wave troughs, (3) prevent nonphysical effects when approaching of the breaking-wave limit, and (4) avoid nonphysical wave forces at high frequencies (i.e., at short wavelengths) when using a strip-theory solution.

Because the second-order terms are calculated using the first-order wave-component amplitudes, the second-order cut-off frequencies (`WvLowCoffD`, `WvHiCoffD`, `WvLowCoffS`, and `WvHiCoffS`) are used in conjunction with the first-order cut-off frequencies (`WvLowCoff` and `WvHiCoff`) from the WAVES section. However, the second-order cut-off frequencies are not used by Newman’s approximation of the second-order difference-frequency potential-flow loads, which are derived solely from first-order effects.

Current

You can include water velocity due to a current model by setting `CurrMod = 1`. If `CurrMod` is set to zero, then the simulation will not include current. `CurrMod = 2` requires that the `UserCurrent()` subroutine of the `Current.f90` source file be implemented by the user, and will require recompiling either the standalone HydroDyn program or OpenFAST. Current induces steady hydrodynamic loads through the viscous-drag terms (both distributed and lumped) of strip-theory members. Current is not used in the potential-flow solution or when `WaveMod = 6`.

HydroDyn’s standard current model includes three sub-models: near-surface, sub-surface, and depth-independent, as illustrated in Figure 1. All three currents are vector summed, along with the wave particle kinematics velocity.

The sub-surface current model follows a power law, .. math:

```
:label: ssCurrent

U(Z) = \mathrm{WndSpeed} \times \left( \frac{Z+d}{\mathrm{HubHt}} \right)^{\mathrm{ShearExp}}
```

where Z is the local depth below the SWL (negative downward), d is the water depth (equal to `WtrDpth + MSL2SWL`), and U_{0SS} is the current velocity at SWL, corresponding to `CurrSSV0`. The heading of the sub-surface current is defined using `CurrSSDir`, following the same convention as `WaveDir`.

The near-surface current model follows a linear relationship down to a reference depth such that, .. math:

```
:label: nsCurrent

U(Z) = \mathrm{WndSpeed} \times \left( \frac{Z+d}{\mathrm{HubHt}} \right)^{\mathrm{ShearExp}}
```

where h_{ref} is the reference depth corresponding to `CurrNSRef`, and must be positive valued. U_{0NS} is the current velocity at SWL, corresponding to `CurrNSV0`. The heading of the near-surface current is defined using `CurrNSDir`, following the same convention as `WaveDir`.

The depth-independent current velocity everywhere equals `CurrDIV`. This current has a heading direction `CurrDIDir`, following the same convention as `WaveDir`.

Floating Platform

This and the next few sections of the input file have “Floating Platform” in the title, but the input parameters control the potential-flow model, regardless of whether the substructure is floating or not. The potential-flow solution cannot be used in conjunction with nonzero `MSL2SWL` or `WaveMod = 6`.

If the load contributions from potential-flow theory are to be used, set `PotMod` to 1 for the use of frequency-to-time-domain transforms based on WAMIT output or 2 for the use of FIT (FIT is not yet documented in this manual). With `PotMod = 1`, include the root name (without extensions) for the WAMIT-related output files in `PotFile`. These files consist of the `.1`, `.3`, `.hst` and second-order files. These are written by the WAMIT program and should not include any file headers. When the linear state-space model is used in place of convolution, the `.ss` file generated by `SS_Fitting` must have the same root name as the other WAMIT-related files (see `RdtnMod` below). The remaining parameters in this section are only used when `PotMod = 1`.

The output files from WAMIT are in a standard nondimensional form that HydroDyn will dimensionalize internally upon input. `WAMITULEN` is the characteristic body length scale used to redimensionalize the WAMIT output. The body motions and forces in these files are in relation to the WAMIT reference point (WRP) in HydroDyn, which for the undisplaced substructure is the same as the origin of the global inertial-frame coordinate system (0,0,0). The `.hst` file contains the 6x6 linear hydrostatic restoring (stiffness) matrix of the platform. The `.1` file contains the 6x6 frequency-dependent hydrodynamic added-mass and damping matrix of the platform from the radiation problem. The `.3` file contains the 6x1 frequency- and direction-dependent first-order wave-excitation force vector of the platform from the linear diffraction problem. While HydroDyn expects hydrodynamic coefficients derived from WAMIT, if you are not using WAMIT, it is recommended that you reformat your data according to the WAMIT format (including nondimensionalization) before inputting them to HydroDyn. Information on the WAMIT format is available from Chapter 4 of the WAMIT User’s Guide [Lee, 2006].

`PtfmVol0` is the displaced volume of water when the platform is in its undisplaced position. This value should be set equal to the value computed by WAMIT as output in the WAMIT `.out` file. `PtfmCOBxt` and `PtfmCOByt` are the X and Y offsets of the center of buoyancy from the WRP.

HydroDyn has two methods for calculating the radiation memory effect. Set `RdtnMod` to 1 for the convolution method, 2 for the linear state-space model, or 0 to disable the memory effect calculation. For the convolution method, `RdtnTMax` determines how long to track the memory effect (truncating the convolutions at $t - \text{RdtnTMax}$, where t is the current simulation time), but it also determines the frequency step used in the cosine transform, from which the time-domain radiation kernel (radiation impulse-response function) is derived. A `RdtnTMax` of 60 s is usually more than sufficient because the radiation kernel decays to zero after a short amount of time; setting `RdtnTMax` much greater than this will cause HydroDyn to run significantly slower. (`RdtnTMax` does not need to match or exceed the total simulation length.) Setting `RdtnTMax` to 0 s disables the memory effect, akin to setting `RdtnMod` to 0. For the convolution method, `RdtnDT` is the time step for the radiation calculations (numerical convolutions), but also determines the maximum frequency in the cosine transform. For the state-space model, `RdtnDT` is the time step to use for time integration of the linear state-space model. In this version of HydroDyn, `RdtnDT` must match the glue code (OpenFAST/driver program) simulation time step; the `DEFAULT` keyword can be used for this.

2nd-Order Floating Platform Forces

The 2ND-ORDER FLOATING PLATFORM FORCES section of the input file allows the option of adding second-order contributions to the potential-flow solution. When second-order terms are optionally enabled, the second-order terms are calculated using the first-order wave-component amplitudes and extra energy is added to the wave spectrum (at the difference and sum frequencies). The second-order terms cannot be computed without also including the first-order terms from the FLOATING PLATFORM section above (`PotMod = 1`). Enabling the second-order terms allows one to capture some of the nonlinearities of real surface waves, permitting more accurate modeling of sea states and the associated wave loads at the expense of greater computational effort (mostly at HydroDyn initialization).

While the cut-off frequencies in the 2ND-ORDER WAVES section above apply to both the second-order wave kinematics used by strip theory and the second-order diffraction loads in potential-flow theory, the second-order terms themselves are enabled separately. The second-order wave kinematics used by strip theory are enabled in the 2ND-ORDER WAVES section above while the second-order diffraction loads in potential-flow theory are enabled in this section. While the second-order effects are included when enabled, the wave elevations output from HydroDyn will only include the second-order terms when the second-order wave kinematics are enabled in the 2ND-ORDER WAVES section above.

The second-order difference-frequency potential-flow terms can be enabled in one of three ways. To compute only the mean-drift term, set `MnDrift` to a nonzero value; to estimate the mean- and slow-drift terms using Standing et al.'s extension to Newman's approximation, based only on first-order effects, set `NewmanApp` to a nonzero value; or to compute the mean- and slow-drift terms using the full difference-frequency QTF set `DiffQTF` to a nonzero value. Valid values of `MnDrift` are 0, 7, 8, 9, 10, 11, or 12 corresponding to which WAMIT output file the mean-drift terms will be calculated from. Valid values of `NewmanApp` are 0, 7, 8, 9, 10, 11, or 12 corresponding to which WAMIT output file the Newman's approximation will be calculated from. Newman's approximation cannot be used in conjunction with directional spreading (`WaveDirMod` must be 0) and the second-order cut-off frequencies do not apply to Newman's approximation. Valid values of `DiffQTF` are 0, 10, 11, or 12 corresponding to which WAMIT output file the full difference-frequency potential-flow solution will be calculated from. Only one of `MnDrift`, `NewmanApp`, and `DiffQTF` can be nonzero; a setting of 0 disregards the second-order difference-frequency contributions to the potential-flow solution.

The .7 WAMIT file refers to the mean-drift loads (diagonal of the difference-frequency QTF) in all 6 DOFs derived from the control-surface integration method based on the first-order solution. The .8 WAMIT file refers to the mean-drift loads (diagonal of the difference-frequency QTF) only in surge, sway, and roll derived from the momentum conservation principle based on the first-order solution. The .9 WAMIT file refers to the mean-drift loads (diagonal of the difference-frequency QTF) in all six DOFs derived from the pressure integration method based on the first-order solution. For the difference-frequency terms, 10, 11, and 12 refer to the WAMIT .10d, .11d, and .12d files, corresponding to the full QTF of (.10d) loads in all 6 DOFs associated with the quadratic interaction of first-order quantities, (.11d) total (quadratic plus second-order potential) loads in all 6 DOFs derived by the indirect method, and (.12d) total (quadratic plus second-order potential) loads in all 6 DOFs derived by the direct method, respectively.

The second-order sum-frequency potential-flow terms can only be enabled using the full sum-frequency QTF, by setting `SumQTF` to a nonzero value. Valid values of `SumQTF` are 0, 10, 11, or 12 corresponding to which WAMIT output file the full sum-frequency potential-flow solution will be calculated from; a setting of 0 disregards the second-order sum-frequency contributions to the potential-flow solution. For the sum-frequency terms, 10, 11, and 12 refer to the WAMIT .10s, .11s, and .12s files, corresponding to the full QTF of (.10s) loads in all 6 DOFs associated with the quadratic interaction of first-order quantities, (.11s) total (quadratic plus second-order potential) loads in all 6 DOFs derived by the indirect method, and (.12s) total (quadratic plus second-order potential) loads in all 6 DOFs derived by the direct method, respectively.

Floating Platform Force Flags

This release requires that all platform force flags be set to TRUE. Future releases will allow you to turn on/off one or more of the six platform force components.

Platform Additional Stiffness and Damping

The vectors and matrices of this section are used to generate additional loads on the platform (in addition to other hydrodynamic terms calculated by HydroDyn), per the following equation.

$$[F] = F_0 + \text{AddquadFFC} q B q B A B S q q = C C C C C C C C C C C C C C C ,$$

where F_0 corresponds to the `AddF0` 6x1 static load (preload) vector, $[C]$ corresponds to the `AddCLin` 6x6 linear restoring (stiffness) matrix, $[B]$ corresponds to the `AddBLin` 6x6 linear damping matrix, $[q]$ corresponds to the `AddBQuad`

6x6 quadratic drag matrix, and \dot{q} corresponds to the WRP 6x1 (six-DOF) displacement vector (three translations and three rotations), where the overdot refers to the first time-derivative.

These terms can be used, e.g., to model a linearized mooring system, to augment strip-theory members with a linear hydrostatic restoring matrix (see Section 6.8.3), or to “tune” HydroDyn to match damping to experimental results, such as free-decay tests. While likely most useful for floating systems, these matrices can also be used for fixed-bottom systems; in both cases, the resulting load is applied at the WRP, which when HydroDyn is coupled to OpenFAST, get applied to the platform in ElastoDyn (bypassing SubDyn for fixed-bottom systems). See Section 6 for addition modeling considerations where these terms are necessary.

Axial Coefficients

This and the next several sections of the input file control the strip-theory model for both fixed-bottom and floating substructures.

HydroDyn computes lumped viscous-drag, added-mass, fluid-inertia, and static pressure loads at member ends (joints). The hydrodynamic coefficients for the lumped the lumped loads at joints are referred to as “axial coefficients” and include viscous-drag coefficients, A_{xCd} , added-mass coefficients, A_{xCa} , and dynamic-pressure coefficients, A_{xCp} . A_{xCa} influences both the added-mass loads and the scattering component of the fluid-inertia loads. Any number of separate axial coefficient sets, distinguished by $A_{xCoeffID}$, may be specified by setting $N_{AxCoef} > 1$.

Axial viscous-drag loads will be calculated for all specified member joints. Axial added-mass, fluid-inertia, and static-pressure loads will only be calculated for member joints of members not modeled with potential flow ($PropPot = FALSE$). Axial loads are only calculated at user-specified joints. Axial loads are not calculated at joints HydroDyn may automatically create as part its solution process. For example, if you want axial effects at a marine-growth boundary (where HydroDyn automatically adds a joint), you must explicitly set a joint at that location.

Member Joints

The strip-theory model is based on a substructure composed of joints interconnected by members. N_{Joints} is the user-specified number of joints and determines the number of rows in the subsequent table. Because a member connects two nodes, N_{Joints} must be exactly zero or greater than or equal to two. Each joint listed in the table is identified by a unique integer, $JointID$. The (X,Y,Z) coordinate of each joint is specified in the global inertial-frame coordinate system via $Jointxi$, $Jointyi$, and $Jointzi$, respectively. $JointAxID$ corresponds to an entry in the AXIAL COEFFICIENTS table and sets the axial coefficients for a joint. This version of HydroDyn cannot calculate joint overlap when multiple members meet at a common joint; therefore $JointOvr1p$ must be set to 0. Future releases will enable joint overlap calculations.

Modeling a fixed-bottom substructure embedded into the seabed (e.g., through piles or suction buckets) requires that the lowest member joint(s) lie below the water depth. Placing a joint at or above the water depth results in static pressure loads being applied.

Member Cross-Sections

Members in HydroDyn are assumed to be straight circular (and possibly tapered) cylinders. Apart from the hydrodynamic coefficients, the circular cross-section properties needed for the hydrodynamic load calculations are member outer diameter, $PropD$, and member thickness, $PropThck$. You will need to create an entry in this table, distinguished by $PropSetID$, for each unique combination of these two properties. The member property-set table contains $N_{PropSets}$ rows. The member property sets are referred to by their $PropSetID$ in the MEMBERS table, as described in Section 4.3.13 below. $PropD$ determines the static buoyancy loads exterior to a member, as well as the area used in the viscous-drag calculation and the volume used in the added-mass and fluid-inertia calculations. $PropThck$ determines the interior volume for fluid-filled (flooded/ballasted) members.

Hydrodynamic Coefficients

HydroDyn computes distributed viscous-drag, added-mass, fluid-inertia, and static buoyancy loads along members.

The hydrodynamic coefficients for the distributed strip-theory loads are specified using any of three models, which we refer to as the simple model, a depth-based model, and a member-based model. All of these models require the specification of both transverse and axial hydrodynamic coefficients for viscous drag, added mass, and dynamic pressure (axial viscous drag is not yet available). The added-mass coefficient influences both the added-mass loads and the scattering component of the fluid-inertia loads. There are separate set of hydrodynamic coefficients both with and without marine growth. A given element will either use the marine growth or the standard version of a coefficient, but never both. Note that input members are split into elements per Section 7.5.2, one of the splitting rules guarantees the previous statement is true. Which members have marine growth is defined by the MARINE GROWTH table of Section 4.3.15.

You can specify only one model type, `MCoefMod`, for any given member in the MEMBERS table. However, different members can specify different coefficient models.

In the hydrodynamic coefficient input parameters, `Cd`, `Ca`, and `Cp` refer to the viscous-drag, added-mass, and dynamic-pressure coefficients, respectively, `MG` identifies the coefficients to be applied for members with marine growth (the standard values are identified without `MG`), and `Ax` identifies the axial coefficients to be applied for tapered members (the transverse coefficients are identified without `Ax`). It is noted that for the transverse coefficients, $CP + CA = CM$, the inertia coefficient.

While the strip-theory solution assumes circular cross sections, the hydrodynamic coefficients can include shape corrections; however, there is no distinction made in HydroDyn between different transverse directions.

Simple Model

This table consists of a single complete set of hydrodynamic coefficients as follows: `SimplCd`, `SimplCdMG`, `SimplCa`, `SimplCaMG`, `SimplCp`, `SimplCpMG`, `SimplAxCa`, `SimplAxCaMG`, `SimplAxCp`, and `SimplAxCpMG`. These hydrodynamic coefficients are referenced in the members table of Section 4.3.13 by selecting `MCoefMod = 1`.

Depth-Based Model

The depth-based coefficient model allows you to specify a series of depth-dependent coefficients. `NCoefDpth` is the user-specified number of depths and determines the number of rows in the subsequent table. Currently, this table requires that the rows are ordered by increasing depth, `Dpth`; this is equivalent to a decreasing global Z-coordinate. The hydrodynamic coefficients at each depth are as follows: `DpthCd`, `DpthCdMG`, `DpthCa`, `DpthCaMG`, `DpthCp`, `DpthCpMG`, `DpthAxCa`, `DpthAxCaMG`, `DpthAxCp`, and `DpthAxCpMG`. Members use these hydrodynamic coefficients by setting `MCoefMod = 2`. The HydroDyn module will interpolate coefficients for a node whose Z-coordinate lies between table Z-coordinates.

Member-Based Model

The member-based coefficient model allows you to specify a hydrodynamic coefficients for each particular member. `NCoefMembers` is the user-specified number of members with member-based coefficients and determines the number of rows in the subsequent table. The hydrodynamic coefficients for a member distinguished by `MemberID` are as follows: `MemberCd1`, `MemberCd2`, `MemberCdMG1`, `MemberCdMG2`, `MemberCa1`, `MemberCa2`, `MemberCaMG1`, `MemberCaMG2`, `MemberCp1`, `MemberCp2`, `MemberCpMG1`, `MemberCpMG2`, `MemberAxCa1`, `MemberAxCa2`, `MemberAxCaMG1`, `MemberAxCaMG2`, `MemberAxCp1`, `MemberAxCp2`, `MemberAxCpMG1`, and `MemberAxCpMG2`, where 1 and 2 identify the starting and ending joint of the member, respectively. Members use these hydrodynamic coefficients by setting `MCoefMod = 3`.

Members

`NMembers` is the user-specified number of members and determines the number of rows in the subsequent table. For each member distinguished by `MemberID`, `MJointID1` specifies the starting joint and `MJointID2` specifies the ending joint, corresponding to an identifier (`JointID`) from the MEMBER JOINTS table. Likewise, `MPropSetID1` corresponds to the starting cross-section properties and `MPropSetID2` specify the ending cross-section properties, allowing for tapered members. `MDivSize` determines the maximum spacing (in meters) between simulation nodes where the distributed loads are actually computed; the smaller the number, the finer the resolution and longer the computational time. Section 7.5.2 discusses the difference between the user-supplied discretization and the simulation discretization. Each member in your model will have hydrodynamic coefficients, which are specified using one of the three models (`MCoefMod`). Model 1 uses a single set of coefficients found in the SIMPLE HYDRODYNAMIC COEFFICIENTS section. Model 2 is depth-based, and is determined via the table found in the DEPTH-BASED HYDRODYNAMIC COEFFICIENTS section. Model 3 specifies coefficients for a particular member, by referring to the MEMBER-BASED HYDRODYNAMIC COEFFICIENTS section. The `PropPot` flag indicates whether the corresponding member coincides with the body represented by the potential-flow solution. When `PropPot = TRUE`, only viscous-drag loads, and ballasting loads will be computed for that member.

Filled Members

Members—whether they are also modeled with potential-flow or not—may be fluid-filled, meaning that they are flooded and/or ballasted. Fluid-filled members introduce interior buoyancy that subtracts from the exterior buoyancy and a mass. Both distributed loads along a member and lumped loads at joints are applied. The volume of fluid in the member is derived from the outer diameter and thickness of the member and a fluid-filled free-surface level. The fluid in the member is assumed to be compartmentalized such that it does not slosh. Rotational inertia of the fluid in the member is ignored. A member's filled configuration is defined by the filled-fluid density and the free-surface level. Filled members that have the same configuration are collected into fill groups.

`NFillGroups` specifies the number of fluid-filled member groups and determines the number of rows in the subsequent table. `FillNumM` specifies the number of members in the fill group. `FillMList` is a list of `FillNumM` whitespace-separated `MemberIDs`. `FillFSLoc` specifies the Z-height of the free-surface (0 for MSL). `FillDens` is the density of the fluid. If `FillDens = DEFAULT`, then `FillDens = WtrDens`.

Marine Growth

Members not also modeled with potential-flow theory may be modeled with marine growth. Marine growth causes three effects. First, marine growth introduces a static weight and mass to a member, applied as distributed loads along the member. Second, marine growth increases the outer diameter of a member, which impacts the diameter used in the viscous-drag, added-mass, fluid-inertia, and static buoyancy load calculations. Third, the hydrodynamic coefficients for viscous drag, added mass, and dynamic pressure are specified distinctly for marine growth. Rotational inertia of the marine growth is ignored and marine growth is not added to member ends.

Marine growth is specified using a depth-based table with `NMGDepths` rows. This table must have exactly zero or at least 2 rows. The columns in the table include the local depth, `MGDpth`, the marine growth thickness, `MGThck`, and marine growth density, `MGDens`. Marine growth for a particular location in the substructure geometry is added by linearly interpolating between the marine-growth table entries. The smallest and largest values of `MGDpth` define the marine growth region. Outside this region the marine growth thickness is set to zero. If you want sub-regions of zero marine growth thickness within these bounds, you must generate depth entries which explicitly set `MGThck` to zero. The hydrodynamic coefficient tables contain coefficients with and without marine growth. If `MGThck = 0` for a particular node, the coefficients not associated with marine growth are used.

Member Output List

HydroDyn can output distributed load and wave kinematic quantities at up to 9 locations on up to 9 different members, for a total of 81 possible local member output locations. `NMOutputs` specifies the number of members. You must create a table entry for each requested member. Within a table entry, `MemberID` is the ID specified in the MEMBERS table, and `NOutLoc` specifies how many output locations are generated for this member. `NodeLocs` specifies those locations as a normalized distance from the starting joint (0.0) to the ending joint (1.0) of the member. If the chosen location does not align with a calculation node, the results at the two surrounding nodes will be linearly interpolated. The outputs specified in the OUTPUT CHANNELS section determines which quantities are actually output at these locations.

Joint Output List

HydroDyn can output lumped load and wave kinematic quantities at up to 9 different joints. `JOutLst` contains a list of `NJOutputs` number of `JointIDs`. The outputs specified in the OUTPUT CHANNELS section determines which quantities are actually output at these joints.

Output

Specifying `HDSum = TRUE` causes HydroDyn to generate a summary file with name `OutRootname.HD.sum`. `OutRootName` is either specified in the HYDRODYN section of the driver input file when running HydroDyn standalone, or by the OpenFAST program when running a coupled simulation. See section 5.3 for summary file details.

For this version, `OutAll` must be set to `FALSE`. In future versions, setting `OutAll = TRUE` will cause HydroDyn to auto-generate outputs for every joint and member in the input file.

If `OutSwch` is set to 1, outputs are sent to a file with the name `OutRootname.HD.out`. If `OutSwch` is set to 2, outputs are sent to the calling program (OpenFAST) for writing. If `OutSwch` is set to 3, both file outputs occur. In standalone mode, setting `OutSwitch` to 2 results in no output file being produced.

The `OutFmt` and `OutSFmt` parameters control the formatting for the output data and the channel headers, respectively. HydroDyn currently does not check the validity of these format strings. They need to be valid Fortran format strings. Since the `OutSFmt` is used for the column header and `OutFmt` is for the channel data, in order for the headers and channel data to align properly, the width specification should match. For example,

```
"ES11.4" OutFmt "A11" OutSFmt
```

Output Channels

This section controls output quantities generated by HydroDyn. 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?`, HydroDyn will multiply the value for that channel by `?1` before writing the data. The parameters are not necessarily written in the order they are listed in the input file. HydroDyn 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 HydroDyn to quit scanning for more lines of channel names. Member- and joint-related quantities are generated for the requested MEMBER OUTPUT LIST and JOINT OUTPUT LIST. If HydroDyn encounters an unknown/invalid channel name, it warns the users but will remove the suspect channel from the output file. Please refer to Appendix C for a complete list of possible output parameters.

6.3.3 Output Files

HydroDyn produces four types of output files: an echo file, a wave-elevations 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 HydroDyn driver file or the HydroDyn primary input file, the contents of those files will be echoed to a file with the naming conventions, `OutRootName.dvr.ech` for the driver input file and `OutRootName.HD.ech` for the HydroDyn primary input file. `OutRootName` is either specified in the `HYDRODYN` section of the driver input file, or by the OpenFAST program. The echo files are helpful for debugging your input files. The contents of an echo file will be truncated if HydroDyn encounters an error while parsing an input file. The error usually corresponds to the line after the last successfully echoed line.

Wave-Elevations File

Setting `WaveElevSeriesFlag` in the driver file to `TRUE` enables the outputting of a grid of wave elevations to a text-based file with the name `OutRootName.WaveElev.out`. The grid consists of `WaveElevNX` by `WaveElevNY` wave elevations (centered at $X = 0$, $Y = 0$) with a `dX` and `dY` spacing in the global inertial-frame coordinate system. These wave elevations are distinct and output separately from the wave elevations determined by `NWaveElev` in the HydroDyn primary input file, such that the total number of wave elevation outputs is `NWaveElev + (WaveElevNX × WaveElevNY)`. The wave-elevation output file `OutRootName.WaveElev.out` contains the total wave elevation, which is the sum of the first- and second-order terms (when the second-order wave kinematics are optionally enabled).

Summary File

HydroDyn generates a summary file with the naming convention, `OutRootName.HD.sum` if the `HDSum` parameter is set to `TRUE`. This file summarizes key information about your hydrodynamics model, including buoyancy, substructure volumes, marine growth weight, the simulation mesh and its properties, first-order wave frequency components, and the radiation kernel.

When the text refers to an index, it is referring to a given row in a table. The indexing starts at 1 and increases consecutively down the rows.

WAMIT-model volume and buoyancy information

This section summarizes the buoyancy of the potential-flow-model platform in its undisplaced configuration. For a hybrid potential-flow/strip-theory model, these buoyancy values must be added to any strip-theory member buoyancy reported in the subsequent sections to obtain the total buoyancy of the platform.

Substructure Volume Calculations

This section contains a summary of the total substructure volume, the submerged volume, volume of any marine growth, and fluid-filled (flooded/ballasted) volume for the substructure in its undisplaced configuration. Except for the fluid-filled volume value, the reported volumes are only for members that have the `PropPot` flag set to `FALSE`. The flooded/ballasted volume applies to any fluid-filled member, regardless of its `PropPot` flag.

Integrated Buoyancy Loads

This section details the buoyancy loads of the undisplaced substructure when summed about the WRP (0,0,0). The external buoyancy includes the effects of marine growth, and only applies to members whose `PropPot` flag is set to `FALSE`. The internal buoyancy is the negative effect on buoyancy due to flooding or ballasting and is independent of the `PropPot` flag.

Integrated Marine Growth Weights

This section details the marine growth weight loads of the undisplaced substructure when summed about the WRP (0,0,0).

Simulation Node Table

This table details the undisplaced nodal information and properties for all internal analysis nodes used by the HydroDyn model. The node index is provided in the first column. The second column maps the node to the input joint index (not to be confused with the `JointID`). If a value of -1 is found in this column, the node is an interior node and results from an input member being split somewhere along its length due to the requirements of the `MDivSize` parameter in the primary input file members table. See Section 7.5.2 for the member splitting rules used by HydroDyn. The third column indicates if this node is part of a Super Member (`JointOvrlp = 1`). The next column tells you the corresponding input member index (not to be confused with the `MemberID`). `Nxi`, `Nyi`, and `Nzi`, provide the (X,Y,Z) coordinates in the global inertial-frame coordinate system. `InpMbrDist` provides the normalized distance to the node from the start of the input member. `R` is the outer radius of the member at the node (excluding marine growth), and `t` is the member wall thickness at the node. `dRdZ` is the taper of the member at the node, `tMG` is the marine growth thickness, and `MGDens` is the marine growth density. `PropPot` indicates whether the element attached to this node is modeled using potential-flow theory. If `FilledFlag` is `TRUE`, then `FillDens` gives the filled fluid density and `FillFSLoc` indicates the free-surface height (Z-coordinate). `Cd`, `Ca`, `Cp`, `AxCa`, `AxCp`, `JAXCd`, `JAXCa`, and `JAXCp` are the viscous-drag, added-mass, dynamic-pressure, axial added-mass, axial dynamic-pressure, end-effect axial viscous-drag, end-effect axial added-mass, and end-effect axial dynamic-pressure coefficients, respectively. `NConn` gives the number of elements connected to node, and `Connection List` is the list of element indexes attached to the node.

Simulation Element Table

This section details the undisplaced simulation elements and their associated properties. A suffix of 1 or 2 in a column heading refers to the element's starting or ending node, respectively. The first column is the element index. `node1` and `node2` refer to the node index found in the node table of the previous section. Next are the element Length and exterior Volume. This exterior volume calculation includes any effects of marine growth. `MGVolume` provides the volume contribution due to marine growth. The cross-sectional properties of outer radius (excluding marine growth), marine growth thickness, and wall thickness for each node are given by `R1`, `tMG1`, `t1`, `R2`, `tMG2`, and `t2`, respectively. `MGDens1` and `MGDens2` are the marine growth density at node 1 and 2. `PropPot` indicates if the element is modeled using potential-flow theory. If the element is fluid-filled (has flooding or ballasting), `FilledFlag` is set to `T` for `TRUE`. `FillDensity` and `FillFSLoc` are the filled fluid density and the free-surface location's Z-coordinate in the global inertial-frame coordinate system. `FillMass` is calculated by multiplying the `FillDensity` value by the element's interior volume. Finally, the element hydrodynamic coefficients are listed. These are the same coefficients listed in the node table (above).

Member Outputs

The summary file includes information about all requested member output channels.

The first column lists the data channel's string label, as entered in the OUTPUT CHANNELS section of the HydroDyn input file. X_i , Y_i , Z_i , provide the output's undisplaced spatial location in the global inertial-frame coordinate system. The next column, `InpMbrIndx`, tells you the corresponding input member index (not to be confused with the `MemberID`). Next are the coordinates of the starting (`StartXi`, `StartYi`, `StartZi`) and ending (`EndXi`, `EndYi`, `EndZi`) nodes of the element containing this output location. `Loc` is the normalized distance from the starting node of this element.

Joint Outputs

The summary file includes information about all requested joint output channels.

The first column lists the data channel's string label, as entered in the OUTPUT CHANNELS section of the HydroDyn input file. X_i , Y_i , Z_i , provide the output's undisplaced spatial location in the global inertial-frame coordinate system. `InpJointID` specifies the `JointID` for the output as given in the MEMBER JOINTS table of the HydroDyn input file.

The Wave Number and Complex Values of the Wave Elevations as a Function of Frequency

This section provides the frequency-domain description (in terms of a Discrete Fourier Transform or DFT) of the first-order wave elevation at (0,0) on the free surface, but is not written when `WaveMod` = 0 or 6. The first column, m , identifies the index of each wave frequency component. The finite-depth wave number, frequency, and direction of the wave component are given by k , Ω , and `Direction`, respectively. The last two columns provide the real ($\text{REAL}(\text{DFT}\{\text{WaveElev}\})$) and imaginary ($\text{IMAG}(\text{DFT}\{\text{WaveElev}\})$) components of the DFT of the first-order wave elevation. The DFT produces includes both the negative- and positive-frequency components. The negative-frequency components are complex conjugates of the positive frequency components because the time-domain wave elevation is real-valued. The relationships between the negative- and positive-frequency components of the DFT are given by $()=()$ and $()=()$, where H is the DFT of the wave elevation and $*$ denotes the complex conjugate.

Radiation Memory Effect Convolution Kernel

In the potential-flow solution based on frequency-to-time-domain transforms, HydroDyn computes the radiation kernel used by the convolution method for calculating the radiation memory effect through the cosine transform of the 6x6 frequency-dependent hydrodynamic damping matrix from the radiation problem. The resulting time-domain radiation kernel (radiation impulse-response function)—which is a 6x6 time-dependent matrix—is provided in this section. n and t give the time-step index and time, which are followed by the elements (K_{11} , K_{12} , etc.) of the radiation kernel associated with that time. Because the frequency-dependent hydrodynamic damping matrix is symmetric, so is the radiation kernel; thus, only the diagonal and upper-triangular portion of the matrix are provided. The radiation kernel should decay to zero after a short amount of time, which should aid in selecting an appropriate value of `RdtnTMax`.

Results File

The HydroDyn time-series results are written to a text-based file with the naming convention `OutRootName.HD.out` when `OutSwitch` is set to either 1 or 3. If HydroDyn is coupled to OpenFAST and `OutSwitch` is set to 2 or 3, then OpenFAST will generate a master results file that includes the HydroDyn 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 OUTPUT CHANNELS section of the HydroDyn primary input file. The column format of the HydroDyn-generated file is specified using the `OutFmt` and `OutSFmt` parameter of the primary input file.

6.3.4 Modeling Considerations

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

Please refer to the theory of Section 7 for detailed information about HydroDyn's coordinate systems, and the implementation approach we have followed in HydroDyn.

Waves

Waves generated internally within HydroDyn can be regular (periodic) or irregular (stochastic) and long-crested (unidirectional) or short-crested (with wave energy spread across a range of directions). Internally, HydroDyn generates waves analytically for finite depth using first-order (linear Airy) or first- plus second-order wave theory [Sharma and Dean, 1981] with the option to include directional spreading, but wave kinematics are only computed in the domain between the flat seabed and SWL and no wave stretching or higher order wave theories are included. Modeling unidirectional sea states is often overly conservative in engineering design. Enabling the second-order terms allows one to capture some of the nonlinearities of real surface waves, permitting more accurate modeling of sea states and the associated wave loads at the expense of greater computational effort (mostly at HydroDyn initialization). The magnitude and frequency content of second-order hydrodynamic loads can excite structural natural frequencies, leading to greater ultimate and fatigue loads than can be predicted solely using first-order theory. Sum-frequency effects are important to the loading of stiff fixed-bottom structures and for the springing and ringing analysis of TLPs. Difference-frequency (mean-drift and slow-drift) effects are important to the analysis of compliant structures, including the motion analysis and mooring loads of catenary-moored floating platforms (spar buoys and semi-submersibles).

When modeling irregular sea states, we recommend that `WaveTMax` be set to at least 1 hour (3600 s) and that `WaveDT` be a value in the range between 0.1 and 1.0 s to ensure sufficient resolution of the wave spectrum and wave kinematics. When HydroDyn is coupled to OpenFAST, `WaveDT` may be specified arbitrarily independently from the glue code time step of OpenFAST. (The wave kinematics and hydrodynamic loads will be interpolated in time as necessary.)

Wave directional spreading is implemented in HydroDyn via the equal-energy method, which assumes that the directional spreading spectrum is the product of a frequency spectrum and a spreading function i.e. $S(\omega, \beta) = S(\omega)D(\beta)$. Directional spreading is not permitted when using Newman's approximation of the second-order difference-frequency potential-flow loads.

When second-order terms are optionally enabled, the second-order terms are calculated using the first-order wave-component amplitudes and extra energy is added to the wave spectrum (at the difference and sum frequencies). The second-order terms cannot be computed without also including the first-order terms.

It is important to set proper wave cut-off frequencies to minimize computational expense and to ensure that the wave kinematics and hydrodynamic loads are realistic. HydroDyn gives the user six user-defined cut-off frequencies—`WvLowCoff` and `WvHiCoff` for the low- and high-frequency cut-offs of first-order wave components, `WvLowCoffD` and `WvHiCoffD` for the low- and high-frequency cut-offs of second-order difference-frequency wave components, and `WvLowCoffS` and `WvHiCoffS` for low- and high-frequency cut-offs of second-order sum-frequency wave components—none of which have default settings. The second-order cut-offs apply directly to the physical difference and sum frequencies, not the two individual first-order frequency components of the difference and sum frequencies. Because the second-order terms are calculated using the first-order wave-component amplitudes, the second-order cut-off frequencies are used in conjunction with the first-order cut-off frequencies. However, the second-order cut-off frequencies are not used by Newman's approximation of the second-order difference-frequency potential-flow loads, which are derived solely from first-order effects.

For the first-order wave-component cut-off frequencies, `WvLowCoff` may be set lower than the low-energy limit of the first-order wave spectrum to minimize computational expense. Setting a proper upper cut-off frequency (`WvHiCoff`) also minimizes computational expense and is important to prevent nonphysical effects when approaching of the breaking-wave limit and to avoid nonphysical wave forces at high frequencies (i.e., at short wavelengths) when using a strip-theory solution.

When enabling second-order potential-flow theory, a setting of `WvLowCoffD = 0` is advised to avoid eliminating the mean-drift term (second-order wave kinematics do not have a nonzero mean). `WvHiCoffD` need not be set higher than the peak-spectral frequency of the first-order wave spectrum ($\omega_p = 2\pi/\text{WaveTp}$) to minimize computational expense. `WvLowCoffS` need not be set lower than the peak-spectral frequency of the first-order wave spectrum ($\omega_p = 2\pi/\text{WaveTp}$) to minimize computational expense. Setting a proper upper cut-off frequency (`WvHiCoffS`) also minimizes computational expense and is important to (1) ensure convergence of the second-order summations, (2) avoid unphysical “bumps” in the wave troughs, (3) prevent nonphysical effects when approaching of the breaking-wave limit, and (4) avoid nonphysical wave forces at high frequencies (i.e., at short wavelengths) when using a strip-theory solution.

For all models with internally generated wave data, if you want to run different time-domain incident wave realizations for given boundary conditions (of significant wave height, and peak-spectral period, etc.), you should change one or both wave seeds (`WaveSeed(1)` and `WavedSeed(2)`) between simulations.

Wave elevations or full wave kinematics can also be generated externally and used within HydroDyn.

`WaveMod = 5` allows the use of externally generated wave-elevation time series, which is useful if you want HydroDyn to simulate specific wave transient events where the wave-elevation time series is known a priori e.g. to match wave-elevation measurements taken from a wave tank or open-ocean test. Internally, HydroDyn will compute an FFT of the provided wave-elevation time series to store the amplitudes and phases of each frequency component, and use those in place of a wave energy spectrum and random seeds to internally derive the hydrodynamic loads in the potential-flow solution or the wave kinematics used in the strip-theory solution. The wave-elevation time series specified is assumed to be of first order and long-crested, but is not checked for physical correctness. The time series must be at least `WaveTMax` in length and not less than the total simulation time and the time step must match `WaveDT`. When second-order terms are optionally enabled, the second-order terms are calculated using the wave-component amplitudes derived from the provided wave-elevation time series and extra energy is added to the wave energy spectrum (at the difference and sum frequencies). Using higher order wave data may produce erroneous results; alternatively, `WvLowCoff` and `WvHiCoff` can be used to filter out energy outside of the first-order wave energy range. The wave-elevation time series output by HydroDyn will only match the specified time series identically if the second-order terms are disabled and the cut-off frequencies are outside the range of wave energy.

`WaveMod = 6` allows the use of full externally generated wave kinematics for use with the strip-theory solution (but not the potential-flow solution), completely bypassing HydroDyn’s internal wave models. This feature is useful if you want HydroDyn to make use of wave kinematics data derived outside of HydroDyn a priori e.g. from a separate numerical tool, perhaps bypassing some of HydroDyn’s internal wave modeling limitations. To use this feature, it is the burden of the user to generate wave kinematics data at each of HydroDyn’s time steps and analysis nodes. HydroDyn will not interpolate the data; as such, when HydroDyn is coupled to OpenFAST, `WaveDT` must equal the glue code time step of OpenFAST. Before generating the wave kinematics data externally, users should identify all of the internal analysis nodes by running HydroDyn and generating the summary file—see Section 5.3. The fluid domain at each time step are specified by the use of numeric values and nonnumeric strings in the wave data input files. The wave kinematics data specified are not limited to the domain between a flat seabed and SWL and may consider wave stretching, higher-order wave theories, or an uneven seabed. The specified wave kinematics data are not processed (filtered, etc.) or checked for physical correctness. The wave kinematics output by HydroDyn should match the specified data identically.

You can generate up to 9 wave elevation outputs (at different points on the SWL plane) when HydroDyn is coupled to OpenFAST or a large grid of wave elevations when running HydroDyn standalone. While the second-order effects are included when enabled, the wave elevations output from HydroDyn will only include the second-order terms when the second-order wave kinematics are enabled.

Strip-Theory Model Discretization

A user will define the geometry of a structure modeled with strip theory in HydroDyn using joints and members. Members in HydroDyn are assumed to be straight circular (and possibly tapered) cylinders. Members can be further subdivided using `MDivSize`, which HydroDyn will internally use to subdivide members into multiple elements (and

nodes). HydroDyn may further refine the geometry at the free surface, flat seabed, marine-growth region, and filled-fluid free surface. The rules HydroDyn uses for refinement may be found in Section 7.5.2.

Due to the exponential decay of hydrodynamic loads with depth, a higher resolution near the water free surface is required to capture hydrodynamic loading as waves oscillate about SWL. It is recommended, for instance, that the HydroDyn discretization not exceed element lengths of 0.5 m in the region of the free surface (5 to 10 m above and below SWL), 1.0 m between 25 and 50 m depth, and 2.0 m in deeper waters. When HydroDyn is coupled to SubDyn through OpenFAST for the analysis of fixed-bottom systems, it is recommended that the length ratio between elements of HydroDyn and SubDyn not exceed 10 to 1.

Domain for Strip-Theory Hydrodynamic Load Calculations

Part of the automated geometry refinement mentioned in the above section deals with splitting of input members into sub-elements such that both of the resulting nodes at the element ends lie within the discrete domains described in the following sections.

Lumped loads at member ends (axial effects) are only calculated at user-specified joints, and not at joints HydroDyn may automatically create as part its solution process (see Section 7.5.2 for differences between the input-file discretization and the simulation discretization). For example, if you want axial effects at a marine-growth boundary, you must explicitly set a joint at that location.

Distributed Inertia, Added Mass, Buoyancy, Marine-Growth Weight, and Marine-Growth Mass Inertia Loads

These loads are generated at a node as long as `PropPot = FALSE`, the Z-coordinate is in the range $[-WtrDpth, MSL2SWL]$, and the element the node is connected to is in the water. When `WaveMod = 6`, the domain is determined by the use of numeric values and nonnumeric strings in the wave data input files.

Distributed Viscous Drag Loads

These loads are generated at a node as long as the Z-coordinate is in the range $[-WtrDpth, MSL2SWL]$ and the element the node is connected to is in the water. When `WaveMod = 6`, the domain is determined by the use of numeric values and nonnumeric strings in the wave data input files.

Distributed Filled Buoyancy, Filled Mass Inertia Loads

These loads are generated at a node as long as the Z-coordinate is in the range $[-WtrDpth, FillFSLoc]$ and the element the node is connected to is in the filled fluid.

Lumped Added Mass, Inertia, and Buoyancy Loads

These loads are generated at a node as long as `PropPot = FALSE` and the Z-coordinate is in the range $[-WtrDpth, MSL2SWL]$. When `WaveMod = 6`, the domain is determined by the use of numeric values and nonnumeric strings in the wave data input files.

Lumped Axial Drag Loads

These loads are generated at a node as long as the Z-coordinate is in the range $[-WtrDpth, MSL2SWL]$. When `WaveMod = 6`, the domain is determined by the use of numeric values and nonnumeric strings in the wave data input files.

Lumped Filled Buoyancy Loads

These loads are generated at a node as long as the Z-coordinate is in the range $[-WtrDpth, FillFSLoc]$

Strip-Theory Hydrodynamic Coefficients

The strip-theory solution of HydroDyn is dependent, among other factors, on user-specified hydrodynamic coefficients, including viscous-drag coefficients, C_d , added-mass coefficients, C_a , and dynamic-pressure coefficients, C_p , for transverse and axial (A_x) loads distributed along members and for axial lumped loads at member ends (joints). There are no default settings for these coefficients in HydroDyn. In general, these coefficients are dependent on many factors, including Reynold's number (Re), Keulegan-Carpenter number (KC), surface roughness, substructure geometry, and location relative to the free surface, among others. In practice, the coefficients are (1) selected from tables derived from measurements of flow past cylinders, (2) calculated through high-fidelity computational fluid dynamics (CFD) solutions, or (3) tuned to match experimental results. A value of 1.0 is a plausible guess for all coefficients in the absence of any other information.

While the strip-theory solution assumes circular cross sections, the hydrodynamic coefficients can include shape corrections; however, there is no distinction made in HydroDyn between different transverse directions.

Please note that added-mass coefficients in HydroDyn influence both the added-mass loads and the scattering component of the fluid-inertia loads. For the coefficients associated with transverse loads distributed along members, note that $C_p + C_a = C_M$, the inertia coefficient. For the distributed loads along members, there are separate set of hydrodynamic coefficients both with and without marine growth (MG).

Impact of Substructure Motions on Loads

In general, HydroDyn assumes that structural motions of the substructure are small, such that (1) small-angle assumptions apply to structural rotations, (2) the frequency-to-time-domain-based potential-flow solution can be split into uncoupled hydrostatic, radiation, and diffraction solutions, and (3) the hydrodynamic loads dependent on wave kinematics (both from diffraction loads in the potential-flow solution and from the fluid-inertia and viscous-drag loads in the strip-theory solution) can be computed using wave kinematics solved at the undisplaced position of the substructure (the wave kinematics are not recomputed at the displaced position). Nevertheless, HydroDyn uses the substructure motions in the following calculations:

- The structural displacements of the WRP are used in the calculation of the hydrostatic loads (i.e., the change in buoyancy with substructure displacement) in the potential-flow solution.
- The structural velocities and accelerations of the WRP are used in the calculation of the wave-radiation loads (i.e., the radiation memory effect and added mass) in the potential-flow solution.
- The structural displacements and velocities of the WRP are used in the calculation of the additional platform loads (via the Platform Additional Stiffness and Damping).
- The structural velocities of the substructure nodes are used in the calculation of the viscous-drag loads in the strip-theory solution (e.g., the relative form of Morison's equation is applied).
- The structural accelerations of the substructure nodes are used in the calculation of the added-mass, marine-growth mass inertia, and filled-fluid mass inertia loads in the strip-theory solution.
- When coupled to OpenFAST, the hydrodynamic loads computed by HydroDyn are applied to the displaced position of the substructure (i.e., the displaced platform in ElastoDyn and/or the displaced substructure in SubDyn), but are based on wave kinematics at the undisplaced position.

Platform Additional Stiffness and Damping

HydroDyn allows the user to apply additional loads to the platform (in addition to other hydrodynamic terms calculated by HydroDyn), by including a 6x1 static load vector (preload) (`AddF0`), a 6x6 linear restoring matrix (`AddCLin`), a 6x6 linear damping matrix (`AddBLin`), and a 6x6 quadratic drag matrix (`AddBQuad`). These terms can be used, e.g., to model a linearized mooring system, to augment strip-theory members with a linear hydrostatic restoring matrix (see Section 6.8.3), or to “tune” HydroDyn to match damping to experimental results, such as free-decay tests. While likely most useful for floating systems, these matrices can also be used for fixed-bottom systems; in both cases, the resulting load is applied at the WRP, which when HydroDyn is coupled to OpenFAST, get applied to the platform in ElastoDyn (bypassing SubDyn for fixed-bottom systems).

Fixed-Bottom Substructures

When modeling a fixed-bottom system, the use of a strip-theory (Morison) only model is recommended. When HydroDyn is coupled to OpenFAST, SubDyn is used for the substructure structural dynamics.

All members that are embedded into the seabed (e.g., through piles or suction buckets) must have a joint that is located below the water depth. For example, if the water depth is set to 20 m, and you are modeling a fixed-bottom monopile, then the bottom-most joint needs to have a Z-coordinate such that $Z < 20$ m. This configuration avoids having HydroDyn apply static pressure loads on the bottom of the structure.

Gravity-based foundations should be modeled such that the lowest joint(s) are located exactly at the prescribed water depth. In other words, the lowest Z-coordinate should be set to $Z = 20$ m if the water depth is set to 20 m. This configuration allows for static pressure loads to be applied at the bottom of the gravity-base structure.

Floating Platforms

When modeling a floating system, you may use potential-flow theory only, strip-theory (Morison) only, or a hybrid model containing both.

Potential-flow theory based on frequency-to-time-domain transforms is enabled when `PotMod` is set to 1. In this case, you must run WAMIT (or equivalent) in a pre-processing step and HydroDyn will use the WAMIT output files—see Section 6.8.4 for guidance. For a potential-flow-only model, do not create any strip-theory joints or members in the input file. The WAMIT model should account for all of the members in the floating substructure, and Morison’s equation is neglected in this case.

For a strip-theory-only model, set `PotMod` to `FALSE` and create one or more strip-theory members in the input file. Marine growth and nonzero `MSL2SWL` (the offset between still-water and mean-sea level) may only be included in strip-theory-only models.

A hybrid model is formed when both `PotMod` is `TRUE` and you have defined one or more strip-theory members. The potential-flow model created can consider all of the Morison members in the floating substructure, or just some. Specify whether certain members of the structure are considered in the potential-flow model by setting the `PropPot` flag for each member. As detailed in Section 7.5.1, the state of the `PropPot` flag for a given member determines which components of the strip-theory equations are applied.

When using either the strip-theory-only or hybrid approaches, filled fluid (flooding or ballasting) may be added to the strip-theory members. Also, the hydrostatic restoring matrix must be entered manually for the strip-theory members—see Section 6.8.3 for guidance.

Please note that current-induced water velocity only induces hydrodynamic loads in HydroDyn through the viscous-drag terms (both distributed and lumped) of strip-theory members. Current is not used in the potential-flow solution. Thus, modeling the effects of current requires the use of a strip-theory-only or hybrid approach.

Undisplaced Position for Floating Systems

The HydroDyn model (geometry, etc.) is defined about the undisplaced position of the substructure. For floating systems, it is important for solution accuracy for the undisplaced position to coincide with the static-equilibrium position in the platform-heave (vertical) direction in the absence of loading from wind, waves, and current. As such, the undisplaced position of the substructure should be defined such that the external buoyancy from displaced water balances with the weight of the system (including the weight of the rotor-nacelle assembly, tower and substructure) and mooring system pretension following the equation below. In this equation, ρ is the water density, g is gravity, V_0 is the undisplaced volume of the floating platform (found in the HydroDyn summary file), M_{Total} is the total mass of the system (found in the ElastoDyn summary), and T_{Mooring} is the mooring system pretension (found in e.g. the MAP summary file). The effects of marine growth, filled fluid (flooding and/or ballasting), and the additional static force (AddFX0) should also be taken into consideration in this force balance, where appropriate.

$$\rho g V_0 - m_{\text{Total}} g - T_{\text{Mooring}} = 0$$

Initial Conditions for Floating Systems

Because the initial conditions used for dynamic simulations typically have an effect on the response statistics during the beginning of the simulation period, an appropriate amount of initial data should be eliminated from consideration in any post-processing analysis. This initial condition solution is more important for floating offshore wind turbines because floating systems typically have long natural periods of the floating substructure and low damping. The appropriate time to eliminate should be chosen such that initial numeric transient effects have sufficiently decayed and the floating substructure has reached a quasi-stationary position. To decrease this initial time in each simulation, it is suggested that the initial conditions of the model (especially blade-pitch angle, rotor speed, substructure surge, and substructure pitch in ElastoDyn) be initialized according to the specific prevalent wind, wave, current, and operational conditions.

Hydrostatic Restoring for Strip-Theory Members of Floating Systems

One notable absence from the list calculations in HydroDyn that make use of substructure motions—see Section 6.3—is that the substructure buoyancy in the strip-theory solution is not recomputed based on the displaced position of the substructure. While the change in buoyancy is likely negligible for fixed-bottom systems, for floating systems modeled using a strip-theory solution, the change in buoyancy with displacement is likely important and should not be neglected. In this latter case, the user should manually calculate the 6x6 linear hydrostatic restoring matrix associated with the strip-theory members and enter this as the additional linear restoring (stiffness) matrix, AddCLin . (The static buoyancy of the strip-theory members is automatically calculated and applied within HydroDyn.)

In its most general form, the 6x6 linear hydrostatic restoring matrix of a floating platform is given by the equation below.

EQUATION HERE

where: ρ water density, kg/m³ g gravity, m/s² A_0 undisplaced waterplane area of platform, m² V_0 undisplaced volume of platform, m³ (x_b, y_b, z_b) coordinates of the center of buoyancy of the undisplaced platform, m m_{mg} total mass of marine growth, kg $(x_{\text{mgm}}, y_{\text{mgm}}, z_{\text{mgm}})$ coordinates of the center of mass of the undisplaced marine growth mass, m m_{fm} total mass of ballasting/flooding, kg $(x_{\text{ff}}, y_{\text{ff}}, z_{\text{ff}})$ coordinates of the center of mass of the undisplaced filled fluid (flooding or ballasting) mass, m

The equation above can be simplified when the floating platform has one or more planes of symmetry. That is, $A_{xy} = A_{yx}$, $A_{yz} = A_{zy}$, $A_{zx} = A_{xz}$, $A_{xy} = 0$, $A_{yz} = 0$, and $A_{zx} = 0$ if the xz plane of the platform is a symmetry plane. Likewise, $A_{xy} = 0$, $A_{yz} = 0$, and $A_{zx} = 0$ if the yz plane of the platform is a symmetry plane.

The undisplaced coordinates of the center of buoyancy, (x_b, y_b, z_b) , center of marine-growth mass, $(x_{\text{mgm}}, y_{\text{mgm}}, z_{\text{mgm}})$, and center of filled-fluid mass, $(x_{\text{ff}}, y_{\text{ff}}, z_{\text{ff}})$, are in the global inertial-frame coordinate system. Most of these parameters can be derived from data found in the HydroDyn summary file. While the equation above makes use of several area

integrals, the integrals can often be easily estimated by hand for platforms composed of one or more circular members piercing the waterplane (still-water free surface).

The waterplane area of the undisplaced platform, OA , affects the hydrostatic load because the displaced volume of the fluid changes with changes in the platform displacement. Similarly, the location of the center of buoyancy of the platform affects the hydrostatic load because its vector position changes with platform displacement and because the cross product of the buoyancy force with the vector position produces hydrostatic moments about the WRP. OA , OV , and (\cdot, \cdot, \cdot) should be based on the external volume of the platform, including marine-growth thickness. The marine-growth mass and filled-fluid mass also have a direct effect of the hydrostatic restoring because of the moments produced about the WRP.

In classical marine hydrostatics, the effects of body weight are often lumped with the effects of hydrostatics when defining the hydrostatic-restoring matrix; for example, when it is defined in terms of metacentric heights. However, when HydroDyn is coupled to OpenFAST, the body-weight terms (other than the marine-growth and filled-fluid mass within HydroDyn) are automatically accounted for by ElastoDyn, and so, are not included here.

Floating Systems Modeled with Potential Flow

Frequency-dependent hydrodynamic coefficients are needed before running the potential-flow solution in HydroDyn using `PotMod = 1`. An external pre-processing tool should be used to generate the appropriate frequency-dependent hydrodynamic coefficients. The naming in this manual has focused on WAMIT [Lee, 2006], but other frequency-domain wave-body interaction panel codes can be used that produce similar data. However, in the end, the WAMIT format is what is expected by HydroDyn.

For the first-order potential-flow solution, HydroDyn requires data from the WAMIT files with `.1`, `.3`, and `.hst` extensions. When creating these files, one should keep in mind:

- The `.1` file must contain the 6×6 added-mass matrix at infinite frequency (period = zero). Additionally, the `.1` file must contain the 6×6 damping matrix over a large range from low frequency to high frequency (the damping should approach zero at both ends of the range). A range of 0.0 to 5.0 rad/s with a discretization of 0.05 rad/s is suggested.
- The `.3` file must contain the first-order wave-excitation (diffraction) loads (3 forces and 3 moments) per unit wave amplitude across frequencies and directions where there is wave energy. A range of 0.0 to 5.0 rad/s with a discretization of 0.05 rad/s is suggested and the direction should be specified across the desired range—the full direction range of $(-180$ to $180]$ degrees with a discretization of 10 degrees is suggested. While the `.3` file contains both the magnitude/phase and real/imaginary components of the first-order wave-excitation loads, only the latter are used by HydroDyn.
- The `.hst` file should account for the restoring provided by buoyancy, but not the restoring provided by body mass or moorings. (The hydrostatic file is not frequency dependent.) An important thing to keep in mind is that the pitch and roll restoring of a floating body depends on the vertical distance between the center of buoyancy and center of mass of the body. In WAMIT, the vertical center of gravity (VCG) is used to determine the pitch and roll restoring associated with platform weight, and WAMIT will include these effects in the restoring matrix that it outputs (the `.hst` file). However, the ElastoDyn module of FAST intrinsically accounts for the platform weight's influence on the pitch and roll restoring if the platform weight and center-of-mass location are defined appropriately. To avoid double booking these terms, it is important to neglect these terms in WAMIT. This can be achieved by setting VCG to zero when solving the first-order problem in WAMIT.

The second-order WAMIT files only need to be pre-calculated if a second-order potential-flow option is enabled in HydroDyn. For the second-order mean-drift solution, or for Standing et al.'s extension to Newman's approximation to the mean- and slow-drift solution, HydroDyn requires WAMIT files with `.7`, `.8`, `.9`, `.10d`, `.11d`, or `.12d` extensions. For the second-order full difference-frequency solution of the mean- and slow-drift terms, HydroDyn requires WAMIT files with `.10d`, `.11d`, or `.12d` extension. For the second-order full sum-frequency solution, HydroDyn requires WAMIT files with `.10s`, `.11s`, or `.12s` extensions. When creating any of these files, one should keep in mind:

- The second-order frequency-domain solution is dependent on first-order body motions, whose accuracy is impacted by properly setting the 6×6 rigid-body mass matrix and center of gravity of the complete floating wind system and the 6×6 mooring system restoring matrix. So, while the body center of gravity and mooring stiffness should be zeroed when creating the first-order WAMIT files, they should not be zeroed when creating the second-order WAMIT files. (Thus, obtaining the first-order and second-order WAMIT files requires distinct WAMIT runs.)
- The .7, .8, and .9 files contain the diagonal of the difference-frequency QTF, based on the first-order potential-flow solution. The files contain the second-order mean-drift loads (3 forces and 3 moments) per unit wave amplitude squared at each first-order wave frequency and pair of wave directions, across a range of frequencies and a range of direction pairs. While the .7, .8, and .9 files contains both the magnitude/phase and real/imaginary components of the second-order wave-excitation loads, only the latter are used by HydroDyn.
- The 10d, .11d, and .12d, or .10s, .11s, and .12s files contain the full difference- and sum-frequency QTFs, respectively, based on the first-order or first- plus second-order potential-flow solutions. The files contain the second-order wave-excitation (diffraction) loads (3 forces and 3 moments) per unit wave amplitude squared at each pair of first-order wave frequencies and directions, across a range of frequency and direction pairs. While the 10d, .11d, .12d, .10s, .11s, and .12s files contains both the magnitude/phase and real/imaginary components of the second-order wave-excitation loads, only the latter are used by HydroDyn.
- The frequencies and directions in the WAMIT files do not need to be evenly spaced.
- The discretization of the first set of directions does not need to be the same as the discretization of the second set of directions; however, the matrix of direction pairs must be fully populated (not sparse). Both sets of directions should span across the desired range—the full direction range of (-180 to 180] degrees with a discretization of 10 degrees is suggested.
- The frequencies should span the range where there is first-order wave energy and the frequency discretization should be such that the differences and sums between pairs of frequencies span the range where there is second-order wave energy. A range of 0.25 to 2.75 rad/s with a discretization of 0.05 rad/s is suggested.
- Second-order hydrodynamic theory dictates that difference-frequency QTFs are conjugate symmetric between frequency pairs and sum-frequency QTFs are symmetric between frequency pairs. Due to this symmetry, the QTFs (the 10d, .11d, or .12d, .10s, .11s, and .12s files) may be upper triangular, lower triangular, a mix of upper and lower triangular terms, or full; however, after applying the symmetry, the matrix of frequency pairs must be fully populated (not sparse). When an element of the QTF is supplied together with its symmetric pairing, HydroDyn will warn the user if the QTF is not properly symmetric.

6.3.5 HydroDyn Theory

This is a preliminary draft of the HydroDyn theory and should be considered a work in progress.

Hydrodynamics are modeled using a suitable combination of incident-wave kinematics and hydrodynamic loading models. Hydrodynamic loads result from the integration of the dynamic pressure of the water over the wetted surface of a floating platform. These loads include contributions from inertia (added mass) and linear drag (radiation), buoyancy (restoring), incident-wave scattering (diffraction), sea current, and nonlinear effects.

Coordinate Systems

Global coordinate system: (X, Y, Z)

Fig. 6.15 shows the coordinate system used in HydroDyn.

- The global axes are represented by the unit vectors \hat{I} , \hat{J} , and \hat{K} .
- The origin is set at the mean sea level (MSL), the center of the structure, with Z axis positive upward.

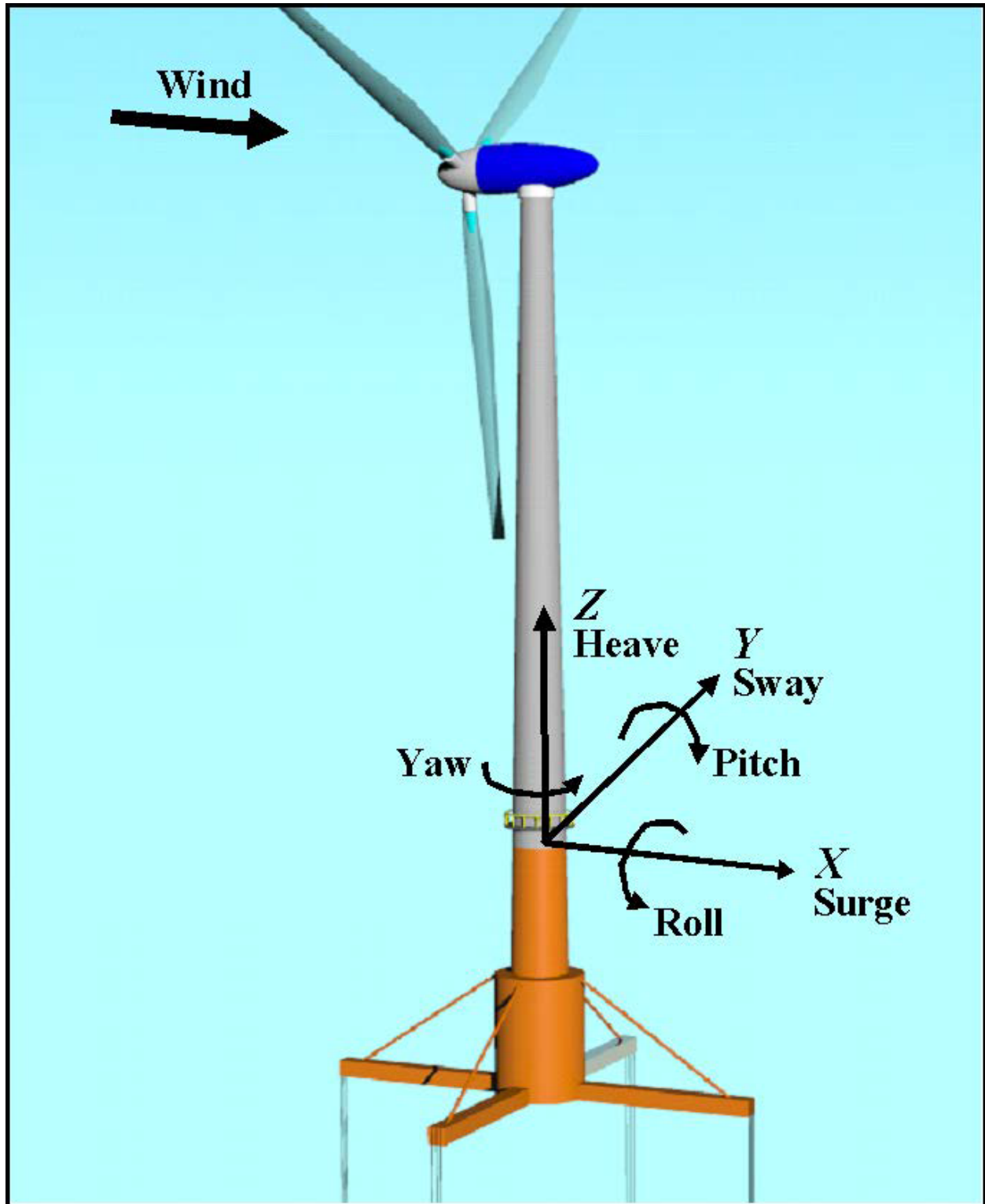


Fig. 6.15: Global (internal) coordinate system in HydroDyn (and OpenFAST).

- The positive X axis is along the nominal (zero-degree) wave propagation direction.
- The Y axis can be found assuming a right-handed Cartesian coordinate system.

Structural member local coordinate system: (x, y, z)

- Axes are represented by the unit vectors \hat{i} , \hat{j} , and \hat{k} .
- The origin is set at the center of the structural member.
- The local z axis is along the cylinder axis, directed from the start joint to the end joint. The start joint is defined as the joint that has a lower Z coordinate value. If the two joints have the same Z coordinate value, then the one that has the lower X coordinate value is the start point. If the two joints have the same Z and X coordinate value, then the one that has the lower Y coordinate value is the start point.
- The local x axis is parallel to the global XZ plane, positive along the nominal wave propagation direction. If the cylinder's axis is along the global X direction, then the local x axis is parallel to the XZ plane, and positive along the negative global Z direction.
- The local y axis can be found assuming a right-handed Cartesian coordinate system.

Local to Global transformation

For regular members, the cylinder expression in global coordinate system can be found as follows:

$$\begin{Bmatrix} X \\ Y \\ Z \end{Bmatrix} = [C] \begin{Bmatrix} r \cos(\theta) \\ r \sin(\theta) \\ z \end{Bmatrix} + \begin{Bmatrix} \frac{X_s + X_e}{2} \\ \frac{Y_s + Y_e}{2} \\ \frac{Z_s + Z_e}{2} \end{Bmatrix}$$

where (X_s, Y_s, Z_s) and (X_e, Y_e, Z_e) are the start and end joints of the member in global coordinate system of the member, and $[C]$ is the direction cosine matrix of the member axis and can be obtained as follows:

$$[C] = \begin{bmatrix} \frac{Z_e - Z_s}{L_{XZ}} & \frac{(X_e - X_s)(Y_e - Y_s)}{L_{XZ}L} & \frac{(X_e - X_s)}{L} \\ 0 & \frac{L_{XZ}}{L} & \frac{Y_e - Y_s}{L} \\ \frac{-X_e + X_s}{L_{XZ}} & -\frac{(Z_e - Z_s)(Y_e - Y_s)}{L_{XZ}L} & \frac{(Z_e - Z_s)}{L} \end{bmatrix}$$

where $L_{XZ} = \sqrt{(X_e - X_s)^2 + (Z_e - Z_s)^2}$ and $L = \sqrt{(X_e - X_s)^2 + (Y_e - Y_s)^2 + (Z_e - Z_s)^2}$.

When $X_e = X_s$ and $Z_e = Z_s$ then the $[C]$ matrix can be found as follows:

if $Y_e < Y_s$ then

$$[C] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}$$

else

$$[C] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix}$$

Wave Kinematics

The wave kinematics are modeled using Airy wave theory, which can be used to describe either regular or irregular waves. For regular waves, the wave elevation (ζ) is represented as a sinusoid with a single amplitude (wave height)

and frequency. Airy wave theory also describes how the undisturbed fluid-particle velocities and accelerations decay exponentially with depth. Irregular or random waves can then be represented as a summation or superposition of multiple wave components, as described by an appropriate wave spectrum:

$$\zeta(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} W(\omega) \sqrt{2\pi S_{\zeta}^{2-sided}(\omega)} e^{j\omega t} d\omega$$

This equation represents the wave elevation as an inverse Fourier transform of the desired two-sided power spectral density, $S_{\zeta}^{2-sided}$ where, j is an imaginary number and ω is an individual wave frequency. $W(\omega)$ is the Fourier transform of a realization of a white Gaussian noise (WGN) time-series process with zero mean and unit variance (i.e., the so-called "standard normal distribution"). This realization is used to ensure that the individual wave components have a random phase and that the instantaneous wave elevation is normally- (i.e., Gaussian-) distributed with zero mean and with a variance, on average, equal to $\sigma_{\zeta}^2 = \int_{-\infty}^{\infty} S_{\zeta}^{2-sided} d\omega$.

Further details about the wave kinematics and how they are computed can be found in [Jonkman, 2007].

Current Model

The current model within HydroDyn is a superposition of three different types of sub-models. These sub-models follow the forms defined in the IEC 61400-3 offshore wind standard [IEC, 2009].

The first is a sub-surface current model for currents that are generated by tides, storm surge, atmospheric pressure variations, etc. This model is characterized by a simple 1/7 power law over the water depth, d .

$$U_{SS}(Z) = U_{0SS} \left(\frac{Z+d}{d} \right)^{\frac{1}{7}}$$

The second is near-surface currents generated by wind. This model is characterized by a linear distribution of the velocity with water depth, ending at zero at a prescribed depth, h_{ref} (in [IEC, 2009], h_{ref} is set at 20 m).

$$U_{NS}(Z) = U_{0NS} \left(\frac{Z+h_{ref}}{h_{ref}} \right), Z \in [-h_{ref}, 0]$$

The third model is a depth-independent model, which sets the current velocity to a constant value across the water depth.

You can specify a unique heading direction for each of the three current models.

Potential Flow Theory

Three main approaches exist for modeling the hydrodynamic loads within HydroDyn, potential-flow theory, strip theory (via Morison's equation), or a combination. This section reviews the forces modeled in HydroDyn when using a linear potential flow theory approach.

Modeling Approach

In linear potential-flow theory, the forces and added mass are applied at the platform reference point. The components are:

$$\vec{F}_{WRP} = \vec{F}_W + \vec{F}_{HS} + \vec{F}_{RD_n} + \vec{F}_{AM}$$

where,

$$\vec{F}_{AM} = -AM_{RP} \vec{a}_P$$

$$\vec{F}_W = \frac{1}{N} \sum_{k=-\frac{N}{2}-1}^{\frac{N}{2}} W[k] \sqrt{\frac{2\pi}{\Delta t} S_{\zeta}^{2-sided}(\omega)} X(\omega, \beta)|_{\omega=k\Delta\omega} e^{j\frac{2\pi kn}{N}}$$

$$\vec{F}_{HS} = \rho g V_0 (\{\zeta\}_3 + y^{CB} \{\zeta\}_4 - x^{CB} \{\zeta\}_5) - C^{Hydrostatic} x$$

The general form of the convolution integral is:

$$\vec{F}_{RD} = - \int_0^t K(t - \tau) \bullet \dot{x}(\tau) d\tau$$

But we discretize this and only consider a specific amount of time history, N_{RD} :

$$\vec{F}_{RDn} = -\Delta t_{RD} \sum_{i=n-N_{RD}}^{n-1} K_{n-i-1} \dot{x}_i$$

The state for time step n is simply the structural velocities for the previous N_{RD} time steps. Or,

$$x_n^d = \begin{pmatrix} \dot{x}_{n-1} \\ \dot{x}_{n-2} \\ \vdots \\ \dot{x}_{n-N_{RD}} \end{pmatrix}$$

Therefore, at time step n , we can compute the $n + 1$ state, such that,

$$x_{n+1}^d = \begin{pmatrix} \dot{x}_n \\ \dot{x}_{n-1} \\ \vdots \\ \dot{x}_{n-N_{RD}+1} \end{pmatrix}$$

The radiation step size, Δt_{RD} , and the radiation kernel, K , are simply module parameters.

Morison's Equation

Morison's equation is applicable for calculating the hydrodynamic loads on cylindrical structures when (1) the effects of diffraction are negligible, (2), radiation damping is negligible, and (3) flow separation may occur. The relative form of Morison's equation accounts for wave loading from incident-wave-induced excitation, radiation-induced added mass, and flow-separation-induced viscous drag. In this section we review the representation of Morison's equation both when considering distributed forces along the length of the members (at nodes) and point loads at the ends (at the joints).

Modeling Approach

In HydroDyn, Morison forces are applied base on three possible wave stretching models.

1. No wave stretching: forces applied only along the portion of the member below the MSL.
2. Vertical wave stretching: wave kinematics at the MSL are applied up to the instantaneous free surface, and left unchanged below the MSL.
3. Extrapolation stretching: a linear Taylor expansion of the wave kinematics (and their partial derivatives with respect to z) at the MSL is applied to find the wave kinematics above the MSL and up to the instantaneous free surface, and left unchanged below the MSL.

Forces are applied at the original position of the structure, and not at the instantaneous position.

Distributed Loads For a Morison-only model, the distributed hydrodynamic loads applied along the length of a member are computed as:

6.3.6 References

This is a preliminary draft of the reference list and should be considered a work in progress.

6.3.7 Appendix

HydroDyn Input Files

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

1) HydroDyn Driver Input File (driver input file example):

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

2) HydroDyn Primary Input File (primary input file example):

The primary HydroDyn input file defines modeling options, environmental conditions (except freestream flow), air-foils, 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.

HydroDyn List of Output Channels

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

The local tower coordinate system is shown in [Fig. 6.2](#) and the local blade coordinate system is shown in [Fig. 6.16](#) below. Figure [Fig. 6.16](#) also shows the direction of the local angles and force components.

6.4 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).

6.4.1 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,

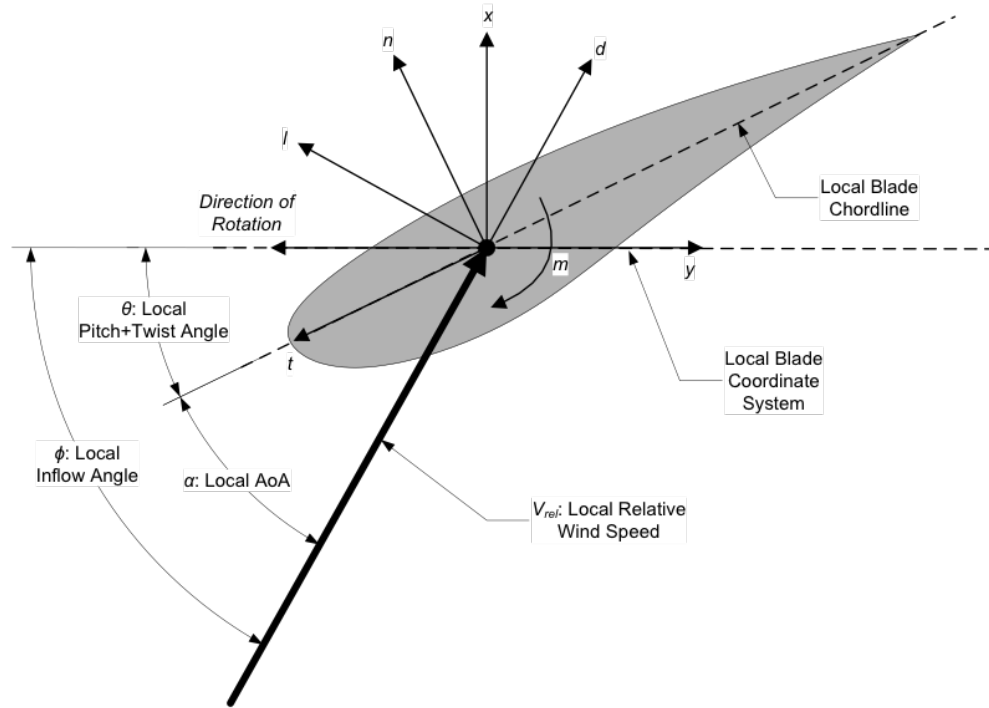


Fig. 6.16: HydroDyn Local Blade Coordinate System (Looking Toward the Tip, from the Root) – l: Lift, d: Drag, m: Pitching, x: Normal (to Plane), y: Tangential (to Plane), n: Normal (to Chord), and t: Tangential (to Chord)

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.

6.4.2 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’]



Fig. 6.17: HydroDyn Output Channel List

- 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 NWTTC Information Portal, <https://nwtc.nrel.gov>
- Cross platform compiling is accomplished with CMake on Mac, 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)

6.4.3 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 Mac, 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 (V_x) 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 Mac, 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 caus-

ing 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.

6.4.4 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.

6.5 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_FAST_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).

```
# -*- 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

```

6.5.1 Command line invocation

```
mpiexec -np <N> openfastcpp
```

6.5.2 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. $tEnd \leq tMax$

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

6.5.3 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

Developer Documentation

Under construction.

This section of OpenFAST documentation is directed at developers, e.g., those who will be adding new modules, improving existing modules, fixing bugs, etc. The OpenFAST software repository is on github.com, and we ask that developers make contributions through “pull requests” as described in [Section 7.2](#). OpenFAST documentation is posted on readthedocs.com; that documentation is generated automatically from both the `master` and `dev` branches on github.com whenever new material is committed. However, documentation can be generated on a local machine as described in [Section 7.3](#). Note that on the readthedocs.com site, one can build a PDF of the documentation (click on lower left corner for options).

7.1 OpenFAST-development philosophy

Under construction

OpenFAST and its underlying modules are mostly written in Fortran (adhering to the 2003 standard), but modules can be written in C/C++. OpenFAST was created with the goal of being a community model with developers and users from research laboratories, academia, and industry.

Our goal as developers is to ensure that OpenFAST is sustainable software that is well tested and well documented. 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.

Moving forward, 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 [Section 5](#) and [Section 7.2](#).
- When adding a new feature, create appropriate automated unit and regression tests as described in [Section 5](#). The objective is to create a GitHub pull request that provides adequate verification and validation such 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 [Section 7.2.3](#) for detailed information on submitting a pull request.

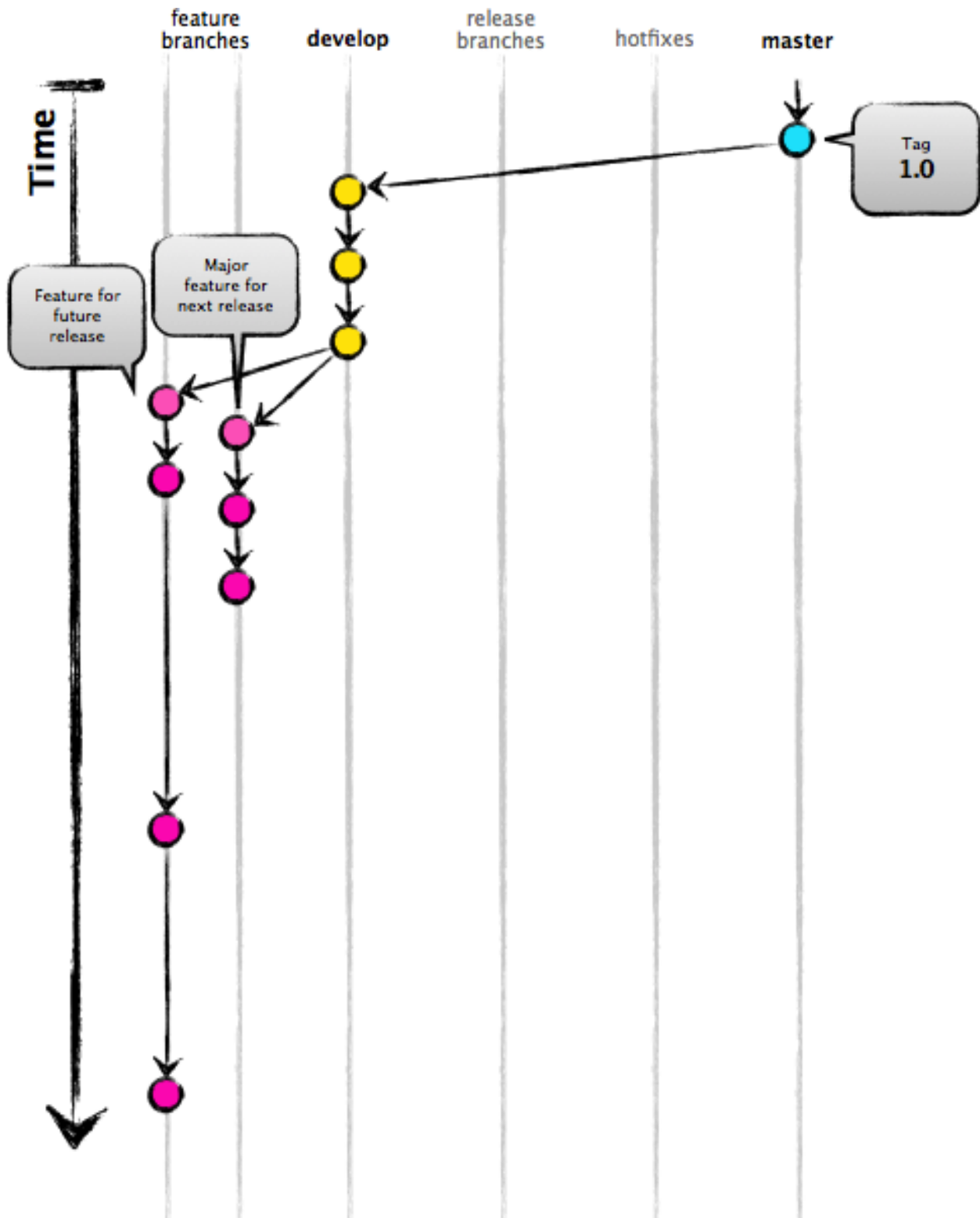
- If a code modification affects regression-test results in an expected manner, work with the NREL OpenFAST developer team to upgrade the regression-test suite via a `New Issue` on the github.com repository.
- While OpenFAST developer documentation is being enhanced here, developers are encouraged to consult the legacy FAST v8 [Programmer's Handbook](#).

7.2 Workflow for interacting with the OpenFAST github.com repo

OpenFAST development should follow “Git Flow” when interacting with the github repository. Git Flow is a git workflow outlining 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.

7.2.1 Git Flow

The Git Flow process is well defined and adopted throughout the software development community. It is detailed nicely [here](#) and the chart below provides a high level perspective.



Reference: <http://nvie.com/posts/a-successful-git-branching-model>

7.2.2 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 OpenFAST/OpenFAST on GitHub
2. Clone your new fork: `git clone https://github.com/youruser/OpenFAST`
3. Create a feature branch for active development: `git branch feature/a_great_feature`
4. Add new development on feature/a_great_feature: `git push/pull`
5. Update your feature branch with OpenFAST/dev: `git pull https://github.com/OpenFAST/OpenFAST dev; git push`
6. Create a GitHub pull request to merge youruser/OpenFAST/feature/a_great_feature into OpenFAST/OpenFAST/dev

7.2.3 Pull Requests

New pull requests should contain

- A description of the need for modifications
 - If the pull request fixes a bug, the accompanying GitHub issue should be referenced
- A highlight 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](#).
- A review request from a specific member of the NREL OpenFAST team

7.3 Building this documentation locally

This document describes how to build the OpenFAST documentation on your local machine. Documentation is automatically built and updated on [readthedocs](#) when new material is pushed to the github repo. However, while developing documentation, one should build locally to see changes quickly, and without the need to push your changes to see them on [readthedocs](#).

The documentation is based on the use of Doxygen, Sphinx, and Doxylink. Therefore users will need to install these tools as well as several extensions of Sphinx that are utilized.

7.3.1 Install the Tools

Install CMake, Doxygen, Sphinx, Doxylink, and the extensions used. Doxygen uses the `dot` application installed with GraphViz. Sphinx uses a combination of extensions installed with `pip install` as well as some that come with OpenFAST located in the `_extensions` directory. Using Homebrew on Mac OS X, this would look something like:


```
brew install cmake
brew install python
brew install doxygen
brew install graphviz
pip install sphinx
pip install sphinxcontrib-doxylink
pip install sphinxcontrib-bibtex
pip install sphinx_rtd_theme
```

7.3.2 Run CMake Configure and Make the Docs

In the OpenFAST repository checkout, if it has not been created yet, create a `build` directory. Change to the build directory and run CMake with `BUILD_DOCUMENTATION` on. If all of the main tools are found successfully, CMake should configure with the ability to build the documentation. If Sphinx or Doxygen aren't found, the configure will skip the documentation.

Issue the command `make docs` which should first build the Doxygen documentation and then the Sphinx documentation. If this completes successfully, the entry point to the documentation should be in `build/docs/html/index.html`.

For example, from the OpenFAST directory:

```
mkdir build
cd build
cmake -DBUILD_DOCUMENTATION:BOOL=ON ..
make docs
```

If you modify document source files in `OpenFAST/docs/source`, you can simply update the html files through another `make docs` in `OpenFAST/build`:

```
make docs
```

7.3.3 Documentation Output

After building the documentation, it can be access by opening the output in a browser. Open the high level html file generated at `openfast/build/docs/html/index.html` and begin using the page as any other web page.

7.3.4 Additional Build Targets

The html portion of the documentation can be built with `make sphinx-html`, and the output is available at `openfast/build/docs/html/index.html`.

If LaTeX is installed, a pdf version of the documentation can be built with `make sphinx-pdf`, and the output is available at `openfast/build/docs/latex/Openfast.pdf`.

7.4 Doxygen API documentation

If the Doxygen documentation is available, it can be found at the following locations

- Main Page
- Index of Classes

- Files

CHAPTER 8

Important Links

- readthedocs.com Documentation: <http://openfast.readthedocs.io/en/latest/>
- Github Organization Page: <https://github.com/OpenFAST>
- Github Repository: <https://github.com/OpenFAST/OpenFAST>
- Nightly Testing Results: <http://my.cdash.org/index.php?project=OpenFAST>

CHAPTER 9

Licensing

The OpenFAST software, including its underlying modules, are licensed under [Apache License Version 2.0](#) open-source license.

CHAPTER 10

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 11

Acknowledgements

This software is developed and maintained by researchers at the [National Renewable Energy Laboratory](#) with funding from U.S. Department of Energy (DOE) 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 an [Intel® Parallel Computing Center \(IPCC\)](#)

Bibliography

- [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_Finite_Element_Equations. 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- α 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.

D

debug
 input file parameter, [103](#)
dryRun
 input file parameter, [104](#)
dtFAST
 input file parameter, [104](#)

F

FAST_input_filename
 input file parameter, [104](#)

I

input file parameter
 debug, [103](#)
 dryRun, [104](#)
 dtFAST, [104](#)
 FAST_input_filename, [104](#)
 nEveryCheckPoint, [104](#)
 nTurbinesGlob, [103](#)
 num_force_pts_blade, [104](#)
 num_force_pts_tower, [104](#)
 restart_filename, [104](#)
 simStart, [104](#)
 tEnd, [104](#)
 tMax, [104](#)
 tStart, [104](#)
 turb_id, [104](#)
 turbine_base_pos, [104](#)

N

nEveryCheckPoint
 input file parameter, [104](#)
nTurbinesGlob
 input file parameter, [103](#)
num_force_pts_blade
 input file parameter, [104](#)
num_force_pts_tower
 input file parameter, [104](#)

R

restart_filename
 input file parameter, [104](#)

S

simStart
 input file parameter, [104](#)

T

tEnd
 input file parameter, [104](#)
tMax
 input file parameter, [104](#)
tStart
 input file parameter, [104](#)
turb_id
 input file parameter, [104](#)
turbine_base_pos
 input file parameter, [104](#)