## Contents

1 This documentation ........................................... 3

2 Installing OpenFAST .......................................... 5
   2.1 Download binaries ......................................... 5
   2.2 Compile from source ...................................... 5
   2.3 Appendix .................................................. 13

3 Testing OpenFAST ............................................. 17
   3.1 Configuring the test suite .............................. 17
   3.2 Test specific documentation ......................... 18
   3.3 Continuous integration ................................ 27

4 User Documentation ........................................... 29
   4.1 API changes between versions ....................... 29
   4.2 AeroDyn Users Guide and Theory Manual ......... 31
   4.4 FAST v8 and the transition to OpenFAST ............ 71
   4.5 C++ API Users Guide .................................. 77

5 Developer Documentation ................................... 81
   5.1 API Reference ........................................... 82
   5.2 Development Philosophy ................................ 82
   5.3 Development Guidelines ............................... 82

6 Licensing ....................................................... 95

7 Getting Help .................................................. 97

8 Acknowledgements ............................................ 99

Bibliography ..................................................... 101
OpenFAST is a multi-physics, multi-fidelity tool for simulating the coupled dynamic response of wind turbines. Practically speaking, OpenFAST is the framework (or “glue code”) that couples computational modules for aerodynamics, hydrodynamics for offshore structures, control and electrical system (servo) dynamics, and structural dynamics to enable coupled nonlinear aero-hydro-servo-elastic simulation in the time domain. OpenFAST enables the analysis of a range of wind turbine configurations, including two- or three-blade horizontal-axis rotor, pitch or stall regulation, rigid or teetering hub, upwind or downwind rotor, and lattice or tubular tower. The wind turbine can be modeled on land or offshore on fixed-bottom or floating substructures.

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

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

- OpenFAST Github Organization
- Github Repository
- Nightly Tests

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

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

The majority of this documentation is divided into two parts:

*User Documentation*

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

*Developer Documentation*

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

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

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

2.1 Download binaries

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

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

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

2.2 Compile from source

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

### 2.2.1 Dependencies

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

#### Build tools

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

<table>
<thead>
<tr>
<th>Package</th>
<th>Applicable systems</th>
<th>Minimum version</th>
<th>Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMake</td>
<td>All</td>
<td>3.0</td>
<td><a href="https://cmake.org">https://cmake.org</a></td>
</tr>
<tr>
<td>GNU Make</td>
<td>macOS, Linux</td>
<td>1.8</td>
<td><a href="https://www.gnu.org/software/make/">https://www.gnu.org/software/make/</a></td>
</tr>
<tr>
<td>GNU Compiler Collection</td>
<td>macOS, Linux</td>
<td>4.6.0</td>
<td><a href="https://gcc.gnu.org">https://gcc.gnu.org</a></td>
</tr>
</tbody>
</table>

#### Math libraries

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

<table>
<thead>
<tr>
<th>Library</th>
<th>Maintainer</th>
<th>Paid/Free</th>
<th>Open Source?</th>
<th>Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLAS</td>
<td>NetLib</td>
<td>Free</td>
<td>Yes</td>
<td><a href="http://www.netlib.org/blas/">http://www.netlib.org/blas/</a></td>
</tr>
<tr>
<td>LAPACK</td>
<td>NetLib</td>
<td>Free</td>
<td>Yes</td>
<td><a href="http://www.netlib.org/lapack/">http://www.netlib.org/lapack/</a></td>
</tr>
<tr>
<td>BLAS/LAPACK</td>
<td>OpenBLAS</td>
<td>Free</td>
<td>Yes</td>
<td><a href="https://www.openblas.net">https://www.openblas.net</a></td>
</tr>
<tr>
<td>MKL</td>
<td>Intel</td>
<td>Paid</td>
<td>No</td>
<td><a href="https://software.intel.com/en-us/mkl">https://software.intel.com/en-us/mkl</a></td>
</tr>
</tbody>
</table>

#### Dependencies for the test suite

The following packages are required to run the test suite:

- Python 3
- MatPlotLib - used for generating error plots

#### Dependencies for the C++ API

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

- HDF5 (provided by HDF5_ROOT)
- yaml-cpp (provided by YAML_ROOT)
2.2.2 Get the code

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

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

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

- “master” branch - Stable release
- “dev” branch - Latest updates

2.2.3 Visual Studio Solution for Windows

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

Building OpenFAST on Windows with Visual Studio

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

Prerequisites

1. A version of Visual Studio (VS).
   - Currently VS 2013 Professional and VS 2015 Community Edition have been tested with OpenFAST.
   - A list of Intel Fortran compatible VS versions and specific installation notes are found [here](#).
   - The included C/C++ project files for MAP++ and the Registry are compatible with VS 2013, but will upgrade seemlessly to a newer version of VS.
   - If you download and install Visual Studio 2015 Community Edition, you will need to be sure and select the C/C++ component using the Customize option.

2. Intel Fortran Compiler
   - Currently only version 2017.1 has been tested with OpenFAST, but any newer version should be compatible.
   - You can download an Intel Fortran compiler [here](#).
   - Only install Intel Fortran after you have completed your Visual Studio installation.

3. Git for Windows
   - Download and install git for Windows.

4. Python 3.x for Windows (for regression/unit testing)
   - The testing framework of OpenFAST requires the use of Python.
   - Please see Section 3 on testing OpenFAST for further information on this topic.
   - We have been working with Continuum’s Anaconda installation of Python 3.6 for Windows.
Compiling OpenFAST

1. Open a command prompt, or git bash shell from the Start menu
2. Create a directory where you will clone OpenFAST repository (change code to your preferred name)
   - mkdir code
   - cd code
3. Clone the OpenFAST repository
   - git clone https://github.com/openfast/openfast.git

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

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

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

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

2.2.4 Understanding CMake

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

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

Running CMake

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

CMake can be executed in a few ways:

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

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

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

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

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

cmake .. -DBUILD_SHARED_LIBS=ON
```

The command line curses interface can be invoked similarly:

```
ccmake ..
```

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

**OpenFAST CMake options**

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

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

```
BUILD_DOCUMENTATION - Build documentation (Default: OFF)
BUILD_OPENFAST_CPP_API - Enable building OpenFAST - C++ API (Default: OFF)
BUILD_OPENFAST_SIMULINK_API - Enable building OpenFAST for use with Simulink
BUILD_SHARED_LIBS - Enable building shared libraries (Default: OFF)
BUILD_TESTING - Build the testing tree (Default: OFF)
CMAKE_BUILD_TYPE - Choose the build type: Debug Release (Default: Release)
CMAKE_Fortran_MODULE_DIRECTORY - Set the Fortran Modules directory
CMAKE_INSTALL_PREFIX - Install path prefix, prepended onto install directories.
DOUBLE_PRECISION - Treat REAL as double precision (Default: ON)
FPE_TRAP_ENABLED - Enable Floating Point Exception (FPE) trap in compiler options (Default: OFF)
GENERATE_TYPES - Use the openfast-registry to autogenerate types
GENERATE_OPTS - Choose the build type: Debug Release (Default: Release)
GENERATE_OPTS - Choose the build type: Debug Release (Default: Release)
ORCA_DLL_LOAD - Enable OrcaFlex library load (Default: OFF)
USE_DLL_INTERFACE - Enable runtime loading of dynamic libraries (Default: ON)
```

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

### 2.2. Compile from source
# Initial configuration with the default settings

cmake ..

# Change the build to Debug mode rather than Release

cmake .. -DCMAKE_BUILD_TYPE=Debug

# Use dynamic linking rather than static linking

cmake .. -DBUILD_SHARED_LIBS=ON

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

```
cmake .. -DCMAKE_BUILD_TYPE=Debug -DBUILD_SHARED_LIBS=ON
```

## CMAKE_BUILD_TYPE

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

<table>
<thead>
<tr>
<th>CMAKE_BUILD_TYPE</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Release</td>
<td>-O3 optimization level</td>
</tr>
<tr>
<td>RelWithDebInfo</td>
<td>-O2 optimization level with -g flag for debug info</td>
</tr>
<tr>
<td>MinSizeRel</td>
<td>-O1 optimization level</td>
</tr>
<tr>
<td>Debug</td>
<td>No optimization and -g flag for debug info; additional debugging flags: -fcheck=all -pedantic -fbacktrace</td>
</tr>
</tbody>
</table>

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

This flag can be set with the following command:

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

## CMAKE_INSTALL_PREFIX

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

This flag can be set with the following command:

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

### Setting the build tool

CMake can target a variety of build tools or *generators*. To obtain a list of available generators on the current system, run with the empty generator flag, select the target from the list, and rerun with the generator flag populated:
OpenFAST Documentation, Release v2.3.0

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

cmake .. -G

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

# Choose one from the list above and pass it as an argument after -G
# NOTE: wrap this is in quotes!

cmake .. -G"Sublime Text 2 - Ninja"

---

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

---

**Math libraries**

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

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

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

---

2.2.5 **CMake with Make for Linux/macOS**

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

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

```bash
# Clone the repository from GitHub using git

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

# Move into the OpenFAST directory

cd OpenFAST

# Create the build directory and move into it

mkdir build
```

---

2.2. Compile from source

11
cd build

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

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

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

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

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

2.2.6 CMake with Visual Studio for Windows

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

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

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

# Move into the OpenFAST directory
cd OpenFAST

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

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

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

Visual Studio will open a solution containing all of the OpenFAST projects, and any module library, module driver, or glue-code can be compiled from there. The compiled binaries are located within a directory determined by the Visual Studio build type (Release, Debug, or RelWithDebInfo) in openfast/build/glue-codes/ and openfast/build/modules/. For example, the OpenFAST executable will be located at openfast/build/glue-codes/Release/openfast.exe when compiling in Release mode.
The **CMake-generated Visual Studio build is not currently fully functional.** Any configurations made to the Solution in the Visual Studio UI will be lost when CMake is executed, and this can happen whenever a change is made to the structure of the file system or if the CMake configuration is changed. It is recommended that this method **not** be used for debugging or active development on Windows. Instead, see *Visual Studio Solution for Windows.*

### 2.3 Appendix

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

#### 2.3.1 Building OpenFAST with Spack

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

**Dependencies**

OpenFAST has the following dependencies:

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

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

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

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

- GCC 6.3.0
- CMake 3.6.1
- pkg-config 0.29.2

**Step 1**

Checkout the official Spack repo from github (we will checkout into `~`):

```bash
cd ~ && git clone https://github.com/LLNL/spack.git
```

**Step 2**

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

```bash
export SPACK_ROOT=~/.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:

```bash
mkdir ${SPACK_ROOT}/var/spack/repos/builtin/packages/openfast
cd ${SPACK_ROOT}/var/spack/repos/builtin/packages/openfast
```

**Step 4**

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

```bash
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:

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

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

```bash
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:

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

The executables and libraries will be located at

```bash
spack location -i openfast
```

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

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

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

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

Installing prerequisites

1. Download and install Cygwin 64-bit. You will need to Run as Administrator to complete the installation process.
   - Choose Install from internet
   - Choose the default install location
   - Choose the default package download location
   - Choose Direct connection
   - Choose a download site
   - See next step for select packages. Alternately, you can skip this step and run setup-x86_64.exe anytime later to select and install required software.

2. Select packages necessary for compiling OpenFAST. Choose binary packages and not the source option.
   - Choose Category view, we will be installing packages from Devel and Math
   - From Devel mark the following packages for installation
     - cmake
     - cmake-doc
     - cmake-gui
     - cygwin-devel
     - gcc-core
     - gcc-fortran
     - gcc-g++
     - git
     - make
     - makedepend
   - From Math mark the following packages for installation
     - liblapack-devel
     - libopenblas
   - To run the test suite, install these optional packages from Python:
     - python3
     - Python3-numpy
   - Click Next and accept all additional packages that the setup process requests to install to satisfy dependencies

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

Compiling OpenFAST

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

- If you would like to run openfast.exe from the cmd terminal, then you must add the C:\cygwin64\lib\lapack and C:\cygwin64\home\<USERNAME>\software\bin to your %PATH% variable in environment setting. Replace <USERNAME> with your account name on Windows system.

- It is suggested to compile with optimization level 2 for Cygwin. Do this by changing the build mode in the cmake command

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

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

### 3.1 Configuring the test suite

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

- r-test
- pFUnit

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

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

3.2.1 Unit test

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

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

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

Dependencies

The following packages are required for unit testing:

- Python 3+
- CMake
- pFUnit - Included in OpenFAST repo through a git-submodule

Compiling

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

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

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

Executing

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

```
>>>$ ./openfast/build/unit_tests/beamdyn_utest
............
Time:        0.018 seconds
OK
(14 tests)
```

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

```
>>>$ ./unit_tests/beamdyn_utest
.....F....... 
Time:       0.008 seconds
```
Failure in:
test_BD_CrvMatrixH_suite.test_BD_CrvMatrixH
  Location:
  [test_BD_CrvMatrixH.F90:48]
simple rotation with known parameters: Pi on xaxis expected +0.5000000 but found: +0.
  → -4554637; difference: |+0.4453627E-01| > tolerance:+0.1000000E-13; first
  → difference at element [1, 1].

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

Error termination. Backtrace:
#0 0x1073b958c
#1 0x1073ba295
#2 0x1073bb1b6
#3 0x106ecdd4f
#4 0x1063fabee
#5 0x10706691e

Adding unit tests

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

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

```
openfast/
  - modules/
    - beamdyn/
      - src/
        | - BeamDyn.f90
        | - BeamDyn_Subs.f90
      - tests/
        - test_BD_Subroutine1.F90
        - test_BD_Subroutine2.F90
        - test_BD_Subroutine3.F90
```

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

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

Then, modify the new CMakeLists.txt with the appropriate list of test subroutines and module name variables.
For reference, a template unit test file is included at openfast/unit_tests/test_SUBROUTINE.F90. Each
unit test should fully test the target code block. If full test coverage is not easily achievable, it may be an indication
that refactoring would be beneficial.

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

- Test driven development

3.2. Test specific documentation
3.2.2 Regression test

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

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

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

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Compiler</th>
<th>Hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>macOS</td>
<td>GNU</td>
<td>2017 MacbookPro</td>
</tr>
<tr>
<td>CentOS 7</td>
<td>Intel</td>
<td>NREL Eagle - Intel Skylake</td>
</tr>
<tr>
<td>CentOS 7</td>
<td>GNU</td>
<td>NREL Eagle - Intel Skylake</td>
</tr>
<tr>
<td>Windows 10</td>
<td>Intel</td>
<td>Dell Precision 3530</td>
</tr>
</tbody>
</table>

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

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

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

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

**Dependencies**

The following packages are required for regression testing:
• Python 3+
• Numpy
• CMake and CTest (Optional)
• Bokeh 1.4 (Optional)

**Executing with Python driver**

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

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

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

**positional arguments:**

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

**optional arguments:**

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

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

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

**Executing with CTest**

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

3.2. Test specific documentation 21
The regression test specific CMake variables are

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

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

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

# Build and install
make install
```

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

```bash
openfast/build/reg_tests/glue-codes/openfast/5MW_Baseline/ServoData/DISCO.dll
openfast/build/reg_tests/glue-codes/openfast/5MW_Baseline/ServoData/DISCO_ITIBarge.dll
openfast/build/reg_tests/glue-codes/openfast/5MW_Baseline/ServoData/DISCO_OC3Hywind.dll
```

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

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

```bash
# Run the entire regression test
cctest

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

# Run the entire regression test with verbose output
cctest -V

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

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

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

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

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

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

Flags can be compounded making useful variations such as

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

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

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

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

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

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

**Regression test examples**

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

**Compile OpenFAST and execute with CTest**

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

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

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

# Create the build and install directories and move into build
mkdir build install && cd build
```
Configure CMake for testing

- BUILD_TESTING - turn ON
- CTEST_OPENFAST_EXECUTABLE - accept the default
- CTEST_[MODULE]_EXECUTABLE - accept the default

```bash
make .. -DBUILD_TESTING=ON
```

# Compile and install
make install

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

## Configure with CMake and a given executable

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

### Download the source code from GitHub

- Note: The default branch is 'master'

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

- If necessary, switch to another target branch and update r-test

```bash
git checkout dev
git submodule update
```

- Create the build directory and move into it

```bash
mkdir build && cd build
```

### Configure CMake with openfast/reg_tests/CMakeLists.txt for testing

- BUILD_TESTING - turn ON
- CTEST_OPENFAST_EXECUTABLE - provide a path
- CTEST_[MODULE]_EXECUTABLE - provide a path

```bash
cmake ../reg_tests

-DDBUILD_TESTING=ON
-DCTEST_OPENFAST_EXECUTABLE=/home/user/Desktop/openfast_executable
-DCTEST_BEAMDYN_EXECUTABLE=/home/user/Desktop/beamdyn_driver
```

- Install required files

```bash
make install
```

- Execute the full test suite with 4 concurrent processes

```bash
ctest -j4
```

## Python driver with a given executable

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

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

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

# Execute the Python driver
```
cd reg_tests
python manualRegressionTest.py -h
```

```
usage: manualRegressionTest.py [-h] [-p [Plotting-Flag]] [-n [No-Execution]]
[-v [Verbose-Flag]] [-case [Case-Name]]

OpenFAST System-Name Compiler-Id Test-Tolerance
```

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

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

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

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

### Detailed example of running on Windows

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

### Windows with Visual Studio regression test

1. Clone the openfast repo and initialize the testing database
   
   (a) Open a git command shell window (like git bash)
   
   (b) Change your working directory to the location above where you want your local repo to be located (the repo will be placed into a folder called openfast at this location)
c. Type: `git clone https://github.com/openfast/openfast.git` (this creates a local version of the openfast repo on your computer). You should see something like this:

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

(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_Double.exe in your openfast\build\bin folder.

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

7. Execute the OpenFAST regression Tests
   (a) Open a command prompt which is configured for Python [like Anaconda3]
   (b) Change your working directory to `openfast\reg_tests`
   (c) Type: `python manualRegressionTest.py ..\build\bin\openfast_x64_Double.exe Windows Intel 1e-5`
       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:

<table>
<thead>
<tr>
<th>Test</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWT_YFix_WSt</td>
<td>PASS</td>
</tr>
<tr>
<td>AWT_WSt_StartUp_HighSpShutDown</td>
<td>PASS</td>
</tr>
<tr>
<td>AWT_YFree_WSt</td>
<td>PASS</td>
</tr>
<tr>
<td>AWT_YFree_WTurb</td>
<td>PASS</td>
</tr>
<tr>
<td>AWT_WSt_StartUpShutDown</td>
<td>PASS</td>
</tr>
<tr>
<td>AOC_WSt</td>
<td>PASS</td>
</tr>
<tr>
<td>AOC_YFree_WTurb</td>
<td>PASS</td>
</tr>
<tr>
<td>AOC_YFix_WSt</td>
<td>PASS</td>
</tr>
<tr>
<td>UAE_Dnwind_YRamp_WSt</td>
<td>PASS</td>
</tr>
<tr>
<td>UAE_Upwind_Rigid_WRamp_PwrCurve</td>
<td>PASS</td>
</tr>
<tr>
<td>WP_VSP_WTurb_PitchFail</td>
<td>PASS</td>
</tr>
<tr>
<td>WP_VSP_ECD</td>
<td>PASS</td>
</tr>
<tr>
<td>WP_VSP_WTurb</td>
<td>PASS</td>
</tr>
<tr>
<td>WP_Stationary_Linear</td>
<td>PASS</td>
</tr>
<tr>
<td>SWRT_YFree_VS_EDG01</td>
<td>PASS</td>
</tr>
<tr>
<td>SWRT_YFree_VS_EDC01</td>
<td>PASS</td>
</tr>
<tr>
<td>SWRT_YFree_VS_WTurb</td>
<td>PASS</td>
</tr>
<tr>
<td>SMW_Land_DLL_WTurb</td>
<td>PASS</td>
</tr>
<tr>
<td>SMW_OC3Mnp1_DLL_WTurb_WavesIrr</td>
<td>PASS</td>
</tr>
<tr>
<td>SMW_OC3Trpd_DLL_WSt_WavesReg</td>
<td>PASS</td>
</tr>
<tr>
<td>SMW_OC4Jckt_DLL_WTurb_WavesIrr_MGrowth</td>
<td>PASS</td>
</tr>
<tr>
<td>SMW_ITIBarge_DLL_WTurb_WavesIrr</td>
<td>PASS</td>
</tr>
<tr>
<td>SMW_TLP_DLL_WTurb_WavesIrr_WavesMulti</td>
<td>PASS</td>
</tr>
<tr>
<td>SMW_OC3Spar_DLL_WTurb_WavesIrr</td>
<td>PASS</td>
</tr>
<tr>
<td>SMW_OC4Semi_WSt_WavesWN</td>
<td>PASS</td>
</tr>
<tr>
<td>SMW_Land_BD_DLL_WTurb</td>
<td>PASS</td>
</tr>
</tbody>
</table>

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

### 3.3 Continuous integration

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

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

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

# Run this on your local machine's command line
BUILDID="build-1"
```

### 3.3. Continuous integration
```bash
INSTANCE="travisci/ci-garnet:packer-1512502276-986baf0"
docker run --name $BUILDID -dit $INSTANCE /sbin/init
docker exec -it $BUILDID bash -l

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

# Modify this line for the commit or pull request to build
git fetch origin +refs/pull/203/merge:
git checkout -qf FETCH_HEAD
git submodule update --init --recursive

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

source ~/.bashrc
pip3 install numpy
mkdir build && cd build
cmake .. -DBUILD_TESTING=ON -DDOUBLE_PRECISION=$DOUBLE_PRECISION -DBUILD_SHARED_
   -DLIBS=ON
make -j 8 install
```
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/](https://nwtc.nrel.gov/). Details on the transition from FAST v8 to OpenFAST may be found in Section 4.4.

### 4.1 API changes between versions

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

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

#### 4.1.1 OpenFAST v2.2.0 to OpenFAST v2.3.0

<table>
<thead>
<tr>
<th>Removed in OpenFAST v2.3.0</th>
<th>Line</th>
<th>Flag Name</th>
<th>Example Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AeroDyn Airfoil Input File - Airfoil Tables</td>
<td>2</td>
<td>Ctrl</td>
<td>0 Ctrl ! Control setting (must be 0 for current AirfoilInfo)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Added in OpenFAST v2.3.0</th>
<th>Line</th>
<th>Flag Name</th>
<th>Example Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AeroDyn Airfoil Input File - Airfoil Tables</td>
<td>2</td>
<td>User-Prop</td>
<td>0 UserProp ! User property (control) setting</td>
</tr>
<tr>
<td>AeroDyn</td>
<td>37</td>
<td>AFTab-Mod</td>
<td>1 AFTabMod - Interpolation method for multiple airfoil tables {1=1D interpolation on AoA (first table only); 2=2D interpolation on AoA and Re; 3=2D interpolation on AoA and UserProp} (-)</td>
</tr>
</tbody>
</table>
4.1.2 OpenFAST v2.1.0 to OpenFAST v2.2.0

No changes required.

4.1.3 OpenFAST v2.0.0 to OpenFAST v2.1.0

No changes required.

4.1.4 OpenFAST v1.0.0 to OpenFAST v2.0.0

<table>
<thead>
<tr>
<th>Removed in OpenFAST v2.0.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module</td>
</tr>
<tr>
<td>--------</td>
</tr>
<tr>
<td>BeamDyn</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Added in OpenFAST v2.0.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module</td>
</tr>
<tr>
<td>--------</td>
</tr>
<tr>
<td>AeroDyn</td>
</tr>
<tr>
<td>AeroDyn</td>
</tr>
<tr>
<td>AeroDyn</td>
</tr>
<tr>
<td>AeroDyn</td>
</tr>
<tr>
<td>BeamDyn</td>
</tr>
<tr>
<td>BeamDyn</td>
</tr>
<tr>
<td>BeamDyn</td>
</tr>
<tr>
<td>BeamDyn</td>
</tr>
<tr>
<td>BeamDyn</td>
</tr>
<tr>
<td>BeamDyn</td>
</tr>
<tr>
<td>BeamDyn</td>
</tr>
</tbody>
</table>

4.1.5 FAST v8.16 to OpenFAST v1.0.0

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

<table>
<thead>
<tr>
<th>Removed in OpenFAST v1.0.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module</td>
</tr>
<tr>
<td>--------</td>
</tr>
<tr>
<td>OpenFAST</td>
</tr>
</tbody>
</table>
4.2 AeroDyn Users Guide and Theory Manual

4.2.1 Introduction

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

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

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

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

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

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

During linearization analyses with AeroDyn coupled to OpenFAST and BEM enabled, the wake can be frozen (i.e., the axial and tangential induces velocities, \( -V_x a \) and \( V_y a' \), are fixed at their operating-point values during linearization) or the induction can be recalculated during linearization using BEM theory. Dynamic wake that accounts for induction dynamics as a result of transient conditions are not yet available in AeroDyn v15 and newer.

The blade airfoil aerodynamics can be steady or unsteady, except in the case that a cavitation check is requested for MHK, in which case only steady aerodynamics are supported. In the steady model, the supplied static airfoil data — including the lift force, drag force, and optional pitching moment and minimum pressure coefficients versus angle of attack (AoA) — are used directly to calculate nodal loads. The AirfoilPrep preprocessor can be used to generate the needed static airfoil data based on uncorrected 2D data (based, e.g., on airfoil tests in a wind tunnel or XFOil), including features to blend data between different airfoils, apply 3D rotational augmentation, and extrapolate to high AoA. The unsteady airfoil aerodynamic (UA) models account for flow hysteresis, including unsteady attached flow, trailing-edge flow separation, dynamic stall, and flow reattachment. The UA models can be considered as 2D dynamic corrections to the static airfoil response as a result of time-varying inflow velocities and angles. Three semi-empirical UA models are available: the original theoretical developments of Beddoes-Leishman (B-L), extensions to the B-L developed by González, and extensions to the B-L model developed by Minnema/Pierce. While all of the UA models are documented in this manual, the original B-L model is not yet functional. Testing has shown that the González and Minnema/Pierce models produce reasonable hysteresis of the normal force, tangential force, and pitching-moment coefficients if the UA model parameters are set appropriately for a given airfoil, Reynolds number, and/or Mach number. However, the results will differ a bit from earlier versions of AeroDyn, (which was based on the Minnema/Pierce extensions to B-L) even if the default UA model parameters are used, due to differences in the UA model logic between the versions. We recommend that users run test cases with uniform wind inflow and fixed yaw error (e.g., through the standalone AeroDyn driver) to examine the accuracy of the normal force, tangential force, and pitching-moment coefficient hysteresis and to adjust the UA model parameters appropriately. The airfoil-, Reynolds-, and Mach-dependent parameters of the UA models may be derived from the static airfoil data. These UA models are valid for small to moderate AoA under normal rotor operation; the steady model is more appropriate under parked or idling conditions. The static airfoil data is always used in the BEM iteration; when UA is enabled, it is applied after the BEM iteration and after the skewed-wake correction. The UA models are not set up to support linearization, so, UA must be disabled during linearization analyses with AeroDyn coupled to OpenFAST. The interpolation of airfoil data based on Reynolds number or aerodynamic-control
setting (e.g., flaps) is not yet available in AeroDyn v15 and newer.

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

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

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

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

4.2.2 Input Files

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

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

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

Units

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

AeroDyn Driver Input File

The driver input file is only needed for the standalone version of AeroDyn and contains inputs normally generated by OpenFAST, and necessary to control the aerodynamic simulation for uncoupled models. A sample AeroDyn driver input file is given in Section 4.2.5.
Set the `Echo` flag in this file to `TRUE` if you wish to have the `AeroDyn_Driver` executable echo the contents of the driver input file (useful for debugging errors in the driver file). The echo file has the naming convention of `OutFileRoot.ech`, where `OutFileRoot` is specified in the I/O SETTINGS section of the driver input file below. `AD_InputFile` is the filename of the primary AeroDyn input file. This name should be in quotations and can contain an absolute path or a relative path.

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

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

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

\[
U(Z) = \text{WndSpeed} \times \left( \frac{Z}{\text{HubHt}} \right)^{\text{ShearExp}}
\]

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

**AeroDyn Primary Input File**

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

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

The input file begins with two lines of header information which is for your use, but is not used by the software.
Fig. 4.1: AeroDyn Driver Turbine Geometry
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 OutRoot-File.AD.ech. OutRootFile is either specified in the I/O SETTINGS section of the driver input file when running AeroDyn standalone, or by the OpenFAST program when running a coupled simulation.

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

Set WakeMod to 0 if you want to disable rotor wake/induction effects or 1 to include these effects using the BEM theory model. When WakeMod is set to 2, a dynamic BEM theory model (DBEMT) is used. WakeMod cannot be set to 2 during linearization analyses.

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

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

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

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

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

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

Environmental Conditions

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

Blade-Element/Momentum Theory Options

The input parameters in this section are not used when WakeMod = 0.
SkewMod determines the skewed-wake correction model. Set SkewMod to 1 to use the uncoupled BEM solution technique without an additional skewed-wake correction. Set SkewMod to 2 to include the Pitt/Peters correction model. The coupled model “SkewMod= 3” is not available in this version of AeroDyn.

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

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

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

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

Dynamic Blade-Element/Momentum Theory Options

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

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

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

Unsteady Airfoil Aerodynamics Options

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

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

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

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

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

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

Rotor/Blade Properties

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

The blade nodal discretization, geometry, twist, chord, and airfoil identifier are set in separate input files for each blade, described in Section 4.2.2. 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. NBlOuts specifies the number of blade nodes that output is requested for (0 to 9) and BlOutNd
on the next line is a list `NBOuts` long of node numbers between 1 and `NumBlNds` (corresponding to a row number in the blade analysis node table in the blade data input files), separated by any combination of commas, semicolons, spaces, and/or tabs. All blades have the same output node numbers. `NTwOuts` specifies the number of tower nodes that output is requested for (0 to 9) and `TwOutNd` on the next line is a list `NTwOuts` long of node numbers between 1 and `NumTwrNds` (corresponding to a row number in the tower analysis node table above), separated by any combination of commas, semicolons, spaces, and/or tabs. The outputs specified in the `OutList` section determine which quantities are actually output at these nodes.

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

**Airfoil Data Input File**

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

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

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

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

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

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

- `alpha0` specifies the zero-lift AoA (in degrees);
- `alpha1` specifies the AoA (in degrees) larger than `alpha0` for which \( f \) equals 0.7; approximately the positive stall angle;
Fig. 4.2: AeroDyn Tower Geometry
• alpha2 specifies the AoA (in degrees) less than alpha0 for which f equals 0.7; approximately the negative stall angle;

• eta_e is the recovery factor and typically has a value in the range [0.85 to 0.95] for UAMod = 1; if the keyword DEFAULT is entered in place of a numerical value, eta_e is set to 0.9 for UAMod = 1, but eta_e is set to 1.0 for other UAMod values and whenever FLookup = TRUE;

• C_nalpha is the slope of the 2D normal force coefficient curve in the linear region;

• T_f0 is the initial value of the time constant associated with Df in the expressions of Df and f'; if the keyword DEFAULT is entered in place of a numerical value, T_f0 is set to 3.0;

• T_V0 is the initial value of the time constant associated with the vortex lift decay process, used in the expression of Cvn; it depends on Reynolds number, Mach number, and airfoil; if the keyword DEFAULT is entered in place of a numerical value, T_V0 is set to 6.0;

• T_p is the boundary-layer leading edge pressure gradient time constant in the expression for Dp and should be tuned based on airfoil experimental data; if the keyword DEFAULT is entered in place of a numerical value, T_p is set to 1.7;

• T_VL is the time constant associated with the vortex advection process, representing the nondimensional time in semi-chords needed for a vortex to travel from the leading to trailing edges, and used in the expression of Cvn; it depends on Reynolds number, Mach number, 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 â and φ 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 â and φ 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"c, 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 â and φ q; 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 â and φ q; 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"c, 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 alpha0 ≤ AoA ≤ alpha1 for UAMod = 1 (and is unused otherwise); by definition, it depends on the airfoil;

• S2 is the constant in the best fit curve of f for AoA > alpha1 for UAMod = 1 (and is unused otherwise); by definition, it depends on the airfoil;

• S3 is the constant in the best fit curve of f for alpha2 ≤ AoA ≤ alpha0 for UAMod = 1 (and is unused otherwise); by definition, it depends on the airfoil;

• S4 is the constant in the best fit curve of f for AoA < alpha2 for UAMod = 1 (and is unused otherwise); by definition, it depends on the airfoil;

• Cn1 is the critical value of C' n at leading-edge separation for positive AoA and should be extracted from airfoil data at a given Reynolds number and Mach number; Cn1 can be calculated from the static value of Cn at either
the break in the pitching moment or the loss of chord force at the onset of stall; \( \text{Cn1} \) is close to the condition of maximum lift of the airfoil at low Mach numbers;

- \( \text{Cn2} \) 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; \( \text{Cn2} \) can be calculated from the static value of \( C_n \) at either the break in the pitching moment or the loss of chord force at the onset of stall; \( \text{Cn2} \) is close to the condition of maximum lift of the airfoil at low Mach numbers;

- \( \text{St}_{\text{sh}} \) is the Strouhal’s shedding frequency; if the keyword DEFAULT is entered in place of a numerical value, \( \text{St}_{\text{sh}} \) is set to 0.19;

- \( \text{Cd0} \) is the drag-force coefficient at zero-lift AoA;

- \( \text{Cm0} \) 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 \( U_{\text{Mod}} = 1 \) (and is unused otherwise);

- \( k_1 \) is a constant in the best fit curve of \( \hat{x}_{cp} \) for \( U_{\text{Mod}} = 1 \) (and is unused otherwise);

- \( k_2 \) is a constant in the best fit curve of \( \hat{x}_{cp} \) for \( U_{\text{Mod}} = 1 \) (and is unused otherwise);

- \( k_3 \) is a constant in the best fit curve of \( \hat{x}_{cp} \) for \( U_{\text{Mod}} = 1 \) (and is unused otherwise);

- \( k_1_{\text{hat}} \) is a constant in the expression of \( C_c \) due to leading-edge vortex effects for \( U_{\text{Mod}} = 1 \) (and is unused otherwise);

- \( x_{\text{cp bar}} \) is a constant in the expression of \( \hat{x}_{cp} \) for \( U_{\text{Mod}} = 1 \) (and is unused otherwise); if the keyword DEFAULT is entered in place of a numerical value, \( x_{\text{cp bar}} \) is set to 0.2; and

- \( \text{UACutOut} \) 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, \( \text{UACutOut} \) is set to 45.

- \( \text{filtCutOff} \) 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, \( \text{filtCutOff} \) is set to 20.

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

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

For each AoA, you must set the AoA (in degrees), \( \alpha \), the lift-force coefficient, \( \text{Coefs(:,1)} \), 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 \( \text{InCol_Alfa}, \text{InCol_Cl}, \text{InCol_Cd}, \text{InCol_Cm}, \text{and InCol_Cpmin} \) 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 \( \text{AoA} = -180 \) and the last should be for \( \text{AoA} = +180 \) (unless \( \text{NumAlf} = 1 \), in which case AoA is unused). If pitching-moment terms are neglected with \( \text{UseBlCm} = \text{FALSE} \) 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 \( \text{CavitCheck} = \text{FALSE} \) in the GENERAL OPTIONS section of the AeroDyn primary input file, the column containing the minimum pressure coefficients may be absent from the file.

**Blade Data Input File**

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

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

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

- **BlSpn** specifies the local span of the blade node along the (possibly preconed) blade-pitch axis from the root; **BlSpn** must be entered in monotonically increasing order—from the most inboard to the most outboard—and the first node must be zero, and when AeroDyn is coupled to OpenFAST, the last node should be located at the blade tip;

- **BlCrvAC** specifies the local out-of-plane offset (when the blade-pitch angle is zero) of the aerodynamic center (reference point for the airfoil lift and drag forces), normal to the blade-pitch axis, as a result of blade curvature; **BlCrvAC** is positive downwind; upwind turbines have negative **BlCrvAC** for improved tower clearance;

- **BlSwpAC** specifies the local in-plane offset (when the blade-pitch angle is zero) of the aerodynamic center (reference point for the airfoil lift and drag forces), normal to the blade-pitch axis, as a result of blade sweep; positive **BlSwpAC** is opposite the direction of rotation;

- **BlCrvAng** specifies the local angle (in degrees) from the blade-pitch axis of a vector normal to the plane of the airfoil, as a result of blade out-of-plane curvature (when the blade-pitch angle is zero); **BlCrvAng** is positive downwind; upwind turbines have negative **BlCrvAng** for improved tower clearance;

- **BlTwist** specifies the local aerodynamic twist angle (in degrees) of the airfoil; it is the orientation of the local chord about the vector normal to the plane of the airfoil, positive to feather, leading edge upwind; the blade-pitch angle will be added to the local twist;

- **BlChord** specifies the local chord length; and

- **BlAFID** specifies which airfoil data the local blade node is associated with; valid values are numbers between 1 and **NumAFfiles** (corresponding to a row number in the airfoil file table in the AeroDyn primary input file); multiple blade nodes can use the same airfoil data.

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

### 4.2.3 Output Files

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

**Echo Files**

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

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

Results Files

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

4.2.4 Modeling Considerations

AeroDyn was designed as an extremely flexible tool for modeling a wide-range of aerodynamic conditions and turbine configurations. This section provides some general guidance to help you construct models that are compatible with AeroDyn.
Please refer to the theory of Section 7 for detailed information about the implementation approach we have followed in AeroDyn.

**Standalone AeroDyn Driver**

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

**Environmental Conditions**

For air, typical values for $\text{AirDens}$, $\text{KinVisc}$, $\text{SpdSound}$, and $\text{Patm}$ are around $1.225 \text{ kg/m}^3$, $1.460\times10^{-5} \text{ m}^2/\text{s}$, $340.3 \text{ m/s}$, and $101,325 \text{ Pa}$, respectively. For seawater, typical values for $\text{AirDens}$, $\text{KinVisc}$, and $\text{Pvap}$ are around $1025 \text{ kg/m}^3$, $1.004\times10^{-6} \text{ m}^2/\text{s}$, and $2000 \text{ Pa}$, respectively.

**Temporal and Spatial Discretization**

For accuracy and numerical stability, we recommend that $\text{DTAero}$ 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 $\text{NumBlNds}$ 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 ($\text{TwrPotent} > 0$), tower shadow ($\text{TwrShadow} = \text{TRUE}$), and/or the tower aerodynamic load ($\text{TwrAero} = \text{TRUE}$) models are enabled, we also recommend that $\text{NumTwrNds}$ 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) ($\text{TwrElev}$), 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 ($\text{WakeMod} = 1$) and UA ($\text{AFAeroMod} = 2$). Normally, the Pitt and Peters skewed-wake ($\text{SkewMod} = 2$), Prandtl tip-loss ($\text{TipLoss} = \text{TRUE}$), Prandtl hub-loss ($\text{HubLoss} = \text{TRUE}$), and tangential induction ($\text{TanInd} = \text{TRUE}$) models should all be enabled, but $\text{SkewMod} = 2$ 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 $\text{IndToler}$ represents the tolerance of the nondimensional residual equation, with no physical association possible; we recommend setting $\text{IndToler}$ to DEFAULT.
While all of the UA models are documented in this manual, the original B-L model is not yet functional. Testing has shown that the González and Minnema/Pierce models produce reasonable hysteresis of the normal force, tangential force, and pitching-moment coefficients if the UA model parameters are set appropriately for a given airfoil, Reynolds number, and/or Mach number. However, the results will differ a bit from earlier versions of AeroDyn, (which was based on the Minnema/Pierce extensions to B-L) even if the default UA model parameters are used, due to differences in the UA model logic between the versions. We recommend that users run test cases with uniform inflow and fixed yaw error (e.g., through the standalone AeroDyn driver) to examine the accuracy of the normal force, tangential force, and pitching-moment coefficient hysteresis and to adjust the UA model parameters appropriately.

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

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

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

Linearization

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

4.2.5 Appendix

AeroDyn Input Files

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

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

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

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

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

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

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

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

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

4) 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 $\beta$ of blade $\alpha$, where $\alpha$ is a number in the range [1,3] and $\beta$ is a number in the range [1,9], corresponding to entry $\beta$ in the $B\text{OutNd}$ list. $T_{\alpha N\beta}$ refers to output node $\beta$ of the tower and is in the range [1,9], corresponding to entry $\beta$ in the $T\text{wOutNd}$ list.

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

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

**4.3 BeamDyn User Guide and Theory Manual**

**4.3.1 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][WS13][WSJJ14][WJSJ15]. The module has been coupled into the FAST aero-hydro-servo-elastic wind turbine multi-physics engineering tool where it used to model blade structural dynamics. The BeamDyn module follows the requirements of the FAST modularization
<table>
<thead>
<tr>
<th>Channel Name(s)</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TwNj1VUndx, TwNj1VUndy, TwNj1VUndz</td>
<td>(m/s), (m/s), (m/s)</td>
<td>Undisturbed wind velocity at TwNj in the local tower coordinate system</td>
</tr>
<tr>
<td>TwNj1STVx, TwNj1STVy, TwNj1STVz</td>
<td>(m/s), (m/s), (m/s)</td>
<td>Structural translational velocity at TwNj in the local tower coordinate system</td>
</tr>
<tr>
<td>TwNj1Vrel</td>
<td>(m/s)</td>
<td>Relative wind speed at TwNj</td>
</tr>
<tr>
<td>TwNj1Re</td>
<td>(-)</td>
<td>Reynolds number (in millions) at TwNj</td>
</tr>
<tr>
<td>TwNj1M</td>
<td>(-)</td>
<td>Mach number at TwNj</td>
</tr>
<tr>
<td>TwNj1Fdx, TwNj1Fdy</td>
<td>(N/m), (N/m)</td>
<td>Drag force per unit length at TwNj in the local tower coordinate system</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Blade</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BuAzimuth</td>
<td>(deg)</td>
<td>Azimuth angle of Bu</td>
</tr>
<tr>
<td>BuPitch</td>
<td>(deg)</td>
<td>Pitch angle of Bu</td>
</tr>
<tr>
<td>BuNj1Clrc</td>
<td>(m)</td>
<td>Tower clearance at BuNj</td>
</tr>
<tr>
<td>BuNj1VUndx, BuNj1VUndy, BuNj1VUndz</td>
<td>(m/s), (m/s), (m/s)</td>
<td>Undisturbed wind velocity at BuNj in the local blade coordinate system</td>
</tr>
<tr>
<td>BuNj1VDisx, BuNj1VDisy, BuNj1VDisz</td>
<td>(m/s), (m/s), (m/s)</td>
<td>Disturbed wind velocity at BuNj in the local blade coordinate system</td>
</tr>
<tr>
<td>BuNj1STVx, BuNj1STVy, BuNj1STVz</td>
<td>(m/s), (m/s), (m/s)</td>
<td>Structural translational velocity at BuNj in the local blade coordinate system</td>
</tr>
<tr>
<td>BuNj1Vrel</td>
<td>(m/s)</td>
<td>Relative wind speed at BuNj</td>
</tr>
<tr>
<td>BuNj1DynP</td>
<td>(Pa)</td>
<td>Dynamic pressure at BuNj</td>
</tr>
<tr>
<td>BuNj1Re</td>
<td>(-)</td>
<td>Reynolds number (in millions) at BuNj</td>
</tr>
<tr>
<td>BuNj1M</td>
<td>(-)</td>
<td>Mach number at BuNj</td>
</tr>
<tr>
<td>BuNj1VIndx, BuNj1VIndy</td>
<td>(m/s), (m/s)</td>
<td>Axial and tangential induced wind velocity at BuNj</td>
</tr>
<tr>
<td>BuNj1AxInd, BuNj1TalInd</td>
<td>(-), (-)</td>
<td>Axial and tangential induction factors at BuNj</td>
</tr>
<tr>
<td>BuNj1Alpha, BuNj1Theta, BuNj1Phi, BuNj1Curv</td>
<td>(deg), (deg), (deg), (deg)</td>
<td>AoA, pitch-twist angle, inflow angle, and curvature angle at BuNj</td>
</tr>
<tr>
<td>BuNj1C1, BuNj1Cdl, BuNj1Cm, BuNj1Cmin</td>
<td>(-), (-), (-), (-), (-)</td>
<td>Lift force, drag force, pitching moment, minimum pressure, normal force (to plane), tangential force (to plane), normal force (to plane)</td>
</tr>
</tbody>
</table>

1 BuNj1Clrc is based on the absolute distance to the nearest point in the tower from BuNj 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 BuNj1 is above the tower top (or below the tower base), the absolute distance to the tower top (or base) minus the local tower radius, in the deflected configuration, is output.

Fig. 4.5: AeroDyn Output Channel List
The model underlying BeamDyn is the geometrically exact beam theory (GBET) \cite{Hod06}. GBET supports full geometric nonlinearity and large deflection, with bending, torsion, shear, and extensional degree-of-freedom (DOFs); anisotropic composite material couplings (using full $6 \times 6$ 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) \cite{Pat84}. For smooth solutions, LSFEs have exponential convergence rates compared to low-order elements that have algebraic convergence \cite{SG03} \cite{WS13}. Two spatial numerical integration schemes are implemented for the finite element inner products: reduced Gauss quadrature and trapezoidal-rule integration. Trapezoidal-rule integration is appropriate when a large number of sectional properties are specified along the beam axis, for example, in a long wind turbine blade with material properties that vary dramatically over the length. Time integration of the BeamDyn equations of motion is achieved through the implicit generalized-\(\alpha\) 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 \cite{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-elas-to-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 \cite{fig:FlowChart} for the coupled interactions between BeamDyn and other modules in FAST. When coupled to FAST, BeamDyn replaces the more simplified blade structural model of ElastoDyn that is still available as an option, but is only applicable to straight isotropic blades dominated by bending. When uncoupled from FAST, the root motion (boundary condition) and applied loads are specified via a stand-alone BeamDyn driver code.

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

This document is organized as follows. Section Running BeamDyn details how to obtain the BeamDyn and FAST software archives and run either the stand-alone version of BeamDyn or BeamDyn coupled to FAST. Section Input Files describes the BeamDyn input files. Section Output Files discusses the output files generated by BeamDyn. Section BeamDyn Theory summarizes the BeamDyn theory. Section Future Work outlines potential future work. Example input files are shown in Appendix Section 4.3.7. A summary of available output channels is found in Appendix BeamDyn List of Output Channels.
Fig. 4.6: Coupled interaction between BeamDyn and FAST
4.3.2 Running BeamDyn

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

Downloading the BeamDyn Software

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

Stand-Alone BeamDyn Archive

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

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

FAST Archive

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

Running BeamDyn

Running the Stand-Alone BeamDyn Program

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

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

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

where, Dvr_5MW_Dynamic.inp is the name of the BeamDyn driver input file, as described in Section BeamDyn Driver Input File.
Running BeamDyn Coupled to FAST

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

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

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

### 4.3.3 Input Files

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

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

#### Units

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

#### BeamDyn Driver Input File

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

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

A sample BeamDyn driver input file is given in Section 4.3.7.

#### Simulation Control Parameters

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

#### Gravity Parameters

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

#### Inertial Frame Parameters

This section defines the relation between two inertial frames, the global coordinate system and initial blade reference coordinate system. GlbPos(1), GlbPos(2), and GlbPos(3) specify three components of the initial global position vector along X, Y, and Z directions resolved in the global coordinate system, see Figure Fig. 4.7. And the
following $3 \times 3$ direction cosine matrix ($\text{GlbDCM}$) relates the rotations from the global coordinate system to the initial blade reference coordinate system.

![Floating blade reference frame at instant $t$](image)

**Fig. 4.7: Global and blade coordinate systems in BeamDyn.**

### Blade Floating Reference Frame Parameters

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

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

(4.2)

where $v_{rt}$ is the root (origin of the floating blade reference frame) translational velocity vector; $\omega_r$ is the constant root (origin of the floating blade reference frame) angular velocity vector; and $r_t$ is the global position vector introduced in the previous section at instant $t$, see Fig. 4.7. The floating blade reference frame coincides with the initial floating blade reference frame at the beginning $t = 0$. $\text{RootVel}(4)$, $\text{RootVel}(5)$, and $\text{RootVel}(6)$ specify the three components of the constant root angular velocity vector about $X$, $Y$, and $Z$ axes in global coordinate system, respectively. $\text{RootVel}(1)$, $\text{RootVel}(2)$, and $\text{RootVel}(3)$, which are the three components of the root translational velocity vector along $X$, $Y$, and $Z$ directions in global coordinate system, respectively, are calculated based on Eq. (4.2).

BeamDyn can handle more complicated root motions by changing, for example, the $\text{BD\_InputSolve}$ subroutine in the $\text{Driver\_Beam.f90}$ (requiring a recompile of stand-alone BeamDyn).

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

\[ a_0 = \omega_r \times (\omega_r \times (r_0 + P)) \]
\[ v_0 = r_0 + \omega_r \times P \]
\[ \omega_0 = \omega_r. \]

(4.3)

where \( a_0 \) is the initial translational acceleration vector along the blade; \( v_0 \) and \( \omega_0 \) the initial translational and angular velocity vectors along the blade, respectively; and \( P \) is the position vector along the blade relative to the root. Note that these equations are actually implemented with a call to the NWTC Library’s mesh mapping routines.

### Applied Load

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

The first six entries \( \text{DistrLoad}(i), i \in [1, 6] \), 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 \( \text{TipLoad}(i), i \in [1, 6] \), specify three components of concentrated tip force vector and three components of concentrated tip moment vector in the global coordinate system, respectively.

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

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

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

BeamDyn is capable of handling more complex loading cases, e.g., time-dependent loads, through customization of the source code (requiring a recompile of stand-alone BeamDyn). The user can define such loads in the \text{BD_InputSolve} subroutine in the \text{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:

\[
\begin{align*}
\text{u\%PointLoad}\text{\%Force(1:3,u\%PointLoad\%NNodes)} &= \text{u\%PointLoad}\text{\%Force(1:3,u\%PointLoad \_\%NNodes)} + \text{DvrData}\text{\%TipLoad(1:3)} \\
\text{u\%PointLoad}\text{\%Moment(1:3,u\%PointLoad\%NNodes)} &= \text{u\%PointLoad}\text{\%Moment(1:3,u\%PointLoad \_\%NNodes)} + \text{DvrData}\text{\%TipLoad(4:6)}
\end{align*}
\]

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 \( \text{u\%PointLoad\%NNodes} \), where the latter is the total number of FE nodes. Note that \( \text{u\%PointLoad}\text{\%Force(1:3,:) and u\%PointLoad}\text{\%Moment(1:3,:)} \) have been populated with the point-load loads read from the BeamDyn driver input file using the call to \text{Transfer_Point_to_POINT} earlier in the subroutine.

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

\[
\begin{align*}
\text{u\%PointLoad}\text{\%Force(:,:) } &= 0.0D0 \\
\text{u\%PointLoad}\text{\%Force(1,2)} &= 1.0D+03 \times \text{SIN}((2.0 \times \pi) \times t/6.0) \\
\text{u\%PointLoad}\text{\%Moment(:,:) } &= 0.0D0
\end{align*}
\]

with \( 1.0D+03 \) being the amplitude and \( 6.0 \) being the period. Note that this particular implementation overrides the tip-load and point-loads defined in the driver input file.
Similar to the concentrated load, the distributed loads can be defined in the same subroutine:

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

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

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

### Primary Input File

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

### BeamDyn Primary Input File

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

The file is organized into several functional sections. Each section corresponds to an aspect of the BeamDyn model. A sample BeamDyn primary input file is given in Section 4.3.7.

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

### Simulation Controls

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

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

`rhoinf` specifies the numerical damping parameter (spectral radius of the amplification matrix) in the range of \([0.0, 1.0]\) used in the generalized-\(\alpha\) time integrator implemented in BeamDyn for dynamic analysis. For \(rhoinf = 1.0\), no numerical damping is introduced and the generalized-\(\alpha\) 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) Gaussian quadrature; and 2) Trapezoidal quadrature. We note that in the current version, Gaussian quadrature is implemented in reduced form to improve efficiency and avoid shear locking. In the trapezoidal quadrature, only one member (FE element) can be defined in the following GEOMETRY section of the primary input file. Trapezoidal quadrature is appropriate when the number of “blade input stations” (described below) is significantly greater than the order of the LSFE.
Refine specifies a refinement parameter used in trapezoidal quadrature. An integer value greater than unity will split the space between two input stations into “Refine factor” of segments. The keyword “DEFAULT” may be used to set it to 1, i.e., no refinement is needed. This entry is not used in Gauss quadrature.

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

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

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

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

Stop_Tol specifies a tolerance parameter used in convergence criteria of a nonlinear solution that is used for the termination of the iteration. The keyword “DEFAULT” sets Stop_Tol = 1.0E-05. Please refer to Section 4.3.5 for more details.

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

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

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

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

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

**Geometry Parameter**

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

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

\texttt{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, \texttt{order\_elem} below, may well be sufficient.

\texttt{kp\_total} specifies the total number of key points used to define the beam members. The following section contains \texttt{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:

\[
kp\_total = \sum_{i=1}^{\text{member\_total}} n_i - \text{member\_total} + 1 \quad (4.4)
\]

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. 4.8 and Fig. 4.9 show two cases for member and key-point definition.

**Fig. 4.8:** Member and key point definition: one member defined by four key points;

1  Member\_Total - Total number of member (-)  
4  KP\_Total - Total number of key point (-)  
1  4

**Fig. 4.9:** Member and key point definition: two members defined by six key points;

2  Member\_Total - Total number of member (-)  
6  KP\_Total - Total number of key point (-)  
1  4  
2  3

The next section defines the key-point information, preceded by two header lines. Each key point is defined by three physical coordinates (\texttt{kp\_xr}, \texttt{kp\_yr}, \texttt{kp\_zr}) in the IEC standard blade coordinate system (the blade reference coordinate system) along with a structural twist angle (\texttt{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 \(Z\) axis. The key points are entered sequentially (from the root to tip) and there should be a total of \texttt{kp\_total} lines for BeamDyn to read in the information, after two header lines. Please refer to Figure Fig. 4.10 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 directly. PitchJ, PitchK, and PitchC specify the pitch actuator inertial, stiffness, and damping coefficient, respectively. In future releases, specifying UsePitchAct = TRUE will enable a second-order pitch actuator, whereby the pitch angular orientation, velocity, and acceleration are determined by the actuator based on the input blade-pitch angle prescribed by the driver code.

Outputs

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

Specifying SumPrint = TRUE causes BeamDyn to generate a summary file with name InputFile.sum. See Section 4.3.4 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 of NNodeOuts long of node numbers between 1 and the number of nodes on the output mesh, separated by any combination of commas, semicolons, spaces, and/or tabs. The nodal positions are given in the summary file, if output. For Gassian quadrature, the number of nodes on the output mesh is the total number of FE nodes; for trapezoidal quadrature, this is the number of quadrature nodes.

The **OutList** block contains a list of output parameters. Enter one or more lines containing quoted strings that in turn contain one or more output parameter names. Separate output parameter names by any combination of commas, semicolons, spaces, and/or tabs. If you prefix a parameter name with a minus sign, “-”, underscore, “_”, or the characters “m” or “M”, BeamDyn will multiply the value for that channel by -1 before writing the data. The parameters are written in the order they are listed in the input file. BeamDyn allows you to use multiple lines so that you can break your list into meaningful groups and so the lines can be shorter. You may enter comments after the closing quote on any of the lines. Entering a line with the string “END” at the beginning of the line or at the beginning of a quoted string found at the beginning of the line will cause BeamDyn to quit scanning for more lines of channel names. Node-related quantities are generated for the requested nodes identified through the OutNd list above. If BeamDyn encounters an unknown/invalid channel name, it warns the users but will remove the suspect channel from the output file. Please refer to Appendix Section 4.3.7 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 4.3.7. 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, \( \mu_{ii} \) 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 \( 6 \times 6 \) damping matrix at each cross section is scaled from the \( 6 \times 6 \) stiffness matrix by these diagonal entries of a \( 6 \times 6 \) scaling matrix:

\[
\mathbf{F}_{\text{Damp}} = \mu \mathbf{S} \dot{\mathbf{\varepsilon}}
\]

where \( \mathbf{F}_{\text{Damp}} \) is the damping force, \( \mathbf{S} \) is the \( 6 \times 6 \) cross-sectional stiffness matrix, \( \dot{\mathbf{\varepsilon}} \) is the strain rate, and \( \mu \) is the damping coefficient matrix defined as

\[
\mu = \begin{bmatrix}
\mu_{11} & 0 & 0 & 0 & 0 & 0 \\
0 & \mu_{22} & 0 & 0 & 0 & 0 \\
0 & 0 & \mu_{33} & 0 & 0 & 0 \\
0 & 0 & 0 & \mu_{44} & 0 & 0 \\
0 & 0 & 0 & 0 & \mu_{55} & 0 \\
0 & 0 & 0 & 0 & 0 & \mu_{66}
\end{bmatrix}
\]

\[(4.5)\]
Distributed Properties

This section specifies the cross-sectional properties at each of the Station_Total stations. For each station, a non-dimensional parameter \( \eta \) specifies the station location along the local blade reference axis ranging from \([0, 1]\). 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 \( \eta \), there are two \( 6 \times 6 \) matrices providing the structural and inertial properties for each 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:

\[
\begin{bmatrix}
K_{ShrFlp} & 0 & 0 & 0 & 0 & 0 \\
0 & K_{ShrEdg} & 0 & 0 & 0 & 0 \\
0 & 0 & EA & 0 & 0 & 0 \\
0 & 0 & 0 & EI_{Edg} & 0 & 0 \\
0 & 0 & 0 & 0 & EI_{Flp} & 0 \\
0 & 0 & 0 & 0 & 0 & GJ
\end{bmatrix}
\] (4.7)

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:

\[
\begin{bmatrix}
m & 0 & 0 & 0 & 0 & -mY_{cm} \\
0 & m & 0 & 0 & 0 & mX_{cm} \\
0 & 0 & m & mY_{cm} & -mX_{cm} & 0 \\
0 & 0 & mY_{cm} & i_{Edg} & -i_{cp} & 0 \\
0 & 0 & -mX_{cm} & i_{cp} & i_{Flp} & 0 \\
-mY_{cm} & mX_{cm} & 0 & 0 & 0 & i_{plr}
\end{bmatrix}
\] (4.8)

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)

\[
i_{plr} = i_{Edg} + i_{Flp}
\] (4.9)

4.3.4 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 $r_0$ that follows the IEC standard, is related to the internal BD coordinate system by Table 4.1 in Section 4.3.5.

Results File

The BeamDyn time-series results are written to a text-based file with the naming convention `DriverInputFile.out` where `DriverInputFile` is the name of the driver input file when BeamDyn is run in the stand-alone mode. If BeamDyn is coupled to FAST, then FAST will generate a master results file that includes the BeamDyn results.
The results in `DriverInputFile.out` are in table format, where each column is a data channel (the first column always being the simulation time), and each row corresponds to a simulation time step. The data channel are specified in the OUTPUT section of the primary input file. The column format of the BeamDyn-generated file is specified using the `OutFmt` parameters of the primary input file.

### 4.3.5 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 $\mathbf{u}$, an over bar denotes unit vector $\mathbf{n}$, and a double underline denotes a tensor $\mathbf{A}$. Note that sometimes the underlines only denote the dimension of the corresponding matrix.

#### Coordinate Systems

**Fig. 4.10 (in Section 4.3.3) and Fig. 4.11** show the coordinate system used in BeamDyn.

**Global Coordinate System**

The global coordinate system is denoted as $X$, $Y$, and $Z$ in **Fig. 4.11**. 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. 4.11**. 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 $X_{r1}$, $Y_{r1}$, and $Z_{r1}$ in **Fig. 4.11** 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 4.1**.

<table>
<thead>
<tr>
<th>Blade Frame</th>
<th>$X_{r0}$</th>
<th>$Y_{r0}$</th>
<th>$Z_{r0}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>BD Frame</td>
<td>$x_2$</td>
<td>$x_3$</td>
<td>$x_1$</td>
</tr>
</tbody>
</table>
Fig. 4.11: Global, blade reference, and internal coordinate systems in BeamDyn. Illustration by Al Hicks, NREL.
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 $\mathbf{Z}_l$ 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” refers 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]

$$\begin{align*}
\dot{h} - F' &= f \\
\dot{g} + \dot{u}h - M' + (\ddot{\mathbf{x}}_0 + \ddot{\mathbf{u}})'F &= m
\end{align*}$$

(4.10)

where $h$ and $g$ are the linear and angular momenta resolved in the inertial coordinate system, respectively; $F$ and $M$ are the beam’s sectional force and moment resultants, respectively; $\mathbf{u}$ is the one-dimensional (1D) displacement of a point on the reference line; $\mathbf{x}_0$ is the position vector of a point along the beam’s reference line; and $f$ and $m$ 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 $(\bullet)$ 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,

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

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

$$
\begin{align*}
\begin{bmatrix}
h \\ g
\end{bmatrix} &= \mathbf{M} \begin{bmatrix}
\dot{u} \\ \omega
\end{bmatrix} \\
\begin{bmatrix}
F \\ M
\end{bmatrix} &= \mathbf{C} \begin{bmatrix}
\varepsilon \\ \kappa
\end{bmatrix}
\end{align*}
$$

(4.11)

where $\mathbf{M}$ and $\mathbf{C}$ are the $6 \times 6$ sectional mass and stiffness matrices, respectively (note that they are not really tensors; $\varepsilon$ and $\kappa$ are the 1D strains and curvatures, respectively; and, $\omega$ is the angular velocity vector that is defined by the rotation tensor $R$ as $\omega = \text{axial}(R \ R^T)$. The axial vector $\mathbf{a}$ associated with a second-order tensor $\mathbf{A}$ is denoted $\mathbf{a} = \text{axial}(\mathbf{A})$ and its components are defined as

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

(4.12)

The 1D strain measures are defined as

$$
\begin{bmatrix}
\varepsilon \\ \kappa
\end{bmatrix} = \begin{bmatrix}
\mathbf{x}'_0 + \mathbf{u}' - (R \ R^T)\tilde{\mathbf{n}}_1 \\ k
\end{bmatrix}
$$

(4.13)
where \( k = \text{axial}([RR_0]'[RR_0]^T) \) is the sectional curvature vector resolved in the inertial basis; \( R_0 \) is the initial rotation tensor; and \( \hat{e}_1 \) is the unit vector along \( x_1 \) direction in the inertial basis. These three sets of equations, including equations of motion Eq. (4.10), constitutive equations Eq. (4.11), and kinematical equations Eq. (4.13), 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-freedoms at each node: three displacement components and three rotation components. Here we use \( q \) to denote the elemental displacement array as \( q = [u^T \ c^T] \) where \( u \) is the displacement and \( c \) is the rotation-parameter vector. The acceleration array can thus be defined as \( \ddot{q} = [\ddot{u}^T \ \ddot{c}^T] \). For nonlinear finite-element analysis, the discretized and incremental forms of displacement, velocity, and acceleration are written as

\[
\begin{align*}
q(x_1) &= \bar{N} \dot{q} \quad \Delta q^T = [\Delta u^T \ \Delta c^T] \\
\ddot{q}(x_1) &= \bar{N} \ddot{u} \quad \Delta \ddot{q}^T = [\Delta \ddot{u}^T \ \Delta \ddot{c}^T] \\
\dddot{q}(x_1) &= \bar{N} \dddot{u} \quad \Delta \dddot{q}^T = [\Delta \dddot{u}^T \ \Delta \dddot{c}^T]
\end{align*}
\] (4.14)

where \( \bar{N} \) is the shape function matrix and \((\cdot)^\dagger\) denotes a column matrix of nodal values.

The displacement fields in an element are approximated as

\[
\begin{align*}
u(\xi) &= h_k(\xi) \tilde{u}_k \\
\nu'(\xi) &= h_k'(\xi) \tilde{u}_k
\end{align*}
\] (4.15)

where \( h_k(\xi) \), the component of shape function matrix \( \bar{N} \), is the \( p \)-th-order polynomial Lagrangian-interpolant shape function of node \( k, k \in \{1, 2, ..., p + 1\}, \tilde{u}_k \) is the \( k \)-th nodal value, and \( \xi \in [-1, 1] \) is the element natural coordinate. However, as discussed in [JelenicC99], the 3D rotation field cannot simply be interpolated as the displacement field in the form of

\[
\begin{align*}
\zeta(\xi) &= h_k(\xi) \tilde{\zeta}_k \\
\zeta'(\xi) &= h_k'(\xi) \tilde{\zeta}_k
\end{align*}
\] (4.16)

where \( \zeta \) is the rotation field in an element and \( \tilde{\zeta}_k \) 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-body motion.

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, \( \hat{\zeta}^k \), by removing the reference rotation, \( \zeta^1 \), from the finite rotation at each node, \( \hat{\zeta}^k = (\zeta^k - \zeta^1) \). It is noted that the minus sign on \( \zeta^1 \) denotes that the relative rotation is calculated by removing the reference rotation from each node. The composition in that equation is an equivalent of \( R(\hat{\zeta}^k) = R^T(\zeta^1) R(\zeta^k) \).

**Step 2:** Interpolate the relative-rotation field: \( \zeta(\xi) = h_k(\xi) \hat{\zeta}_k \) and \( \zeta'(\xi) = h_k'(\xi) \hat{\zeta}_k \). Find the curvature field \( \zeta(\xi) = R(\zeta^1) H(\zeta^1) \zeta' \), where \( H \) is the tangent tensor that relates the curvature vector \( \hat{k} \) and rotation vector \( \zeta \) as

\[
\hat{k} = H \zeta'
\] (4.17)

**Step 3:** Restore the rigid-body rotation removed in Step 1: \( \zeta(\xi) = \zeta^1 \oplus \zeta(\xi) \).
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 $N$) are $p^{th}$-order Lagrangian interpolants, where nodes are located at the $p + 1$ Gauss-Lobatto-Legendre (GLL) points in the $[-1, 1]$ element natural-coordinate domain. Fig. 4.12 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. 4.12: 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.](image1)

![Fig. 4.13: Representative $p + 1$ Lagrangian-interpolant shape functions in the element natural coordinates for a eighth-order LSFEs, where nodes are located at the Gauss-Lobatto-Legendre points.](image2)

**Wiener-Milenković Rotation Parameter**

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

$$\xi = 4 \tan \left( \frac{\phi}{4} \right) \hat{n}$$  \hspace{1cm} (4.18)

where $\phi$ is the rotation angle and $\hat{n}$ 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 $\pi$ as:

$$\xi = \begin{cases} 
4(q_0p + p_0q + \tilde{p}_0q)/(\Delta_1 + \Delta_2), & \text{if } \Delta_2 \geq 0 \\
-4(q_0p + p_0q + \tilde{p}_0q)/(\Delta_1 - \Delta_2), & \text{if } \Delta_2 < 0 
\end{cases}$$  \hspace{1cm} (4.19)
where \( p, q, \) and \( r \) are the vectorial parameterization of three finite rotations such that \( R(r) = R(p)R(q), \) \( p_0 = 2 - p^T \hat{p}/8, q_0 = 2 - q^T \hat{q}/8, \Delta_1 = (4 - p_0)(4 - q_0), \) and \( \Delta_2 = p_0q_0 - \hat{p}^T \hat{q}. \) 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

\[
\hat{R}(\zeta) = \frac{1}{(4 - c_0)^2} \begin{bmatrix}
  c_0^2 + c_1^2 + c_2^2 - c_3^2 & 2(c_1c_2 - c_0c_3) & 2(c_1c_3 - c_0c_2) \\
  2(c_1c_2 + c_0c_3) & c_0^2 - c_1^2 + c_2^2 - c_3^2 & 2(c_2c_3 - c_0c_1) \\
  2(c_2c_3 + c_0c_1) & 2(c_1c_3 + c_0c_2) & c_0^2 - c_1^2 - c_2^2 + c_3^2
\end{bmatrix}
\]  

(4.20)

where \( \zeta = [c_1 \ c_2 \ c_3]^T \) is the Wiener-Milenković parameter and \( c_0 = 2 - \frac{1}{8} \zeta^T \zeta. \) The relation between rotation tensor and direction cosine matrix (DCM) is

\[
\hat{R} = (DCM)^T
\]  

(4.21)

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. (4.10) are in the form of

\[
\begin{aligned}
\hat{M}\Delta\ddot{\zeta} + \hat{G}\Delta\dot{\zeta} + \hat{K}\Delta\zeta &= \hat{F}^\text{ext} - \hat{\bar{F}} \\
\end{aligned}
\]  

(4.22)

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

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

(4.23)

where \( \mathcal{F}^\text{ext} \) is the applied load vector. The new matrix notations in Eqs. (4.23) to are briefly introduced here, \( \mathcal{F}^C \) and \( \mathcal{F}^D \) are elastic forces obtained from Eq. (4.10) as

\[
\begin{aligned}
\mathcal{F}^C &= \left\{ \begin{array}{c}
\mathcal{F} \\
M
\end{array} \right\} = \mathcal{C} \left\{ \begin{array}{c}
\zeta \\
\dot{\zeta}
\end{array} \right\} \\
\mathcal{F}^D &= \left\{ \begin{array}{c}
0 \\
(\dot{x}_0 + \ddot{u})^T
\end{array} \right\} \mathcal{F}^C = \Lambda \mathcal{F}^C
\end{aligned}
\]  

(4.24)

where $0$ denotes a $3 \times 3$ null matrix. The $\mathcal{G}^I$, $\mathcal{K}^I$, $\mathcal{Q}$, $\mathcal{P}$, $\mathcal{O}$, and $\mathcal{F}^I$ in Eqs. (4.23) are defined as

\[
\begin{align*}
\mathcal{G}^I &= \begin{bmatrix}
0 & (\ddot{\omega} m \dot{\eta} + \dot{\omega} m \dot{\eta})^T \\
0 & \dot{\omega} m \dot{\eta}^T + \ddot{\omega} m \dot{\eta}^T
\end{bmatrix} \\
\mathcal{K}^I &= \begin{bmatrix}
0 & \dot{u} m \dot{\eta} - \ddot{\omega} \dot{\eta} + \dot{\omega} \dot{\eta} - \dot{\omega} \dot{\eta} \\
0 & \ddot{u} \dot{\eta} + \dot{\omega} \dot{\eta} - \ddot{\omega} \dot{\eta} + \dot{\omega} \dot{\eta}
\end{bmatrix} \\
\mathcal{Q} &= \begin{bmatrix}
0 & C_{11} \dot{E}_1 - \dot{F} \\
0 & C_{21} \dot{E}_1 - \dot{M}
\end{bmatrix} \\
\mathcal{P} &= \begin{bmatrix}
\dot{F} + (C_{11} \dot{E}_1)^T & (C_{21} \dot{E}_1)^T \\
0 & 0
\end{bmatrix} \\
\mathcal{O} &= \chi \mathcal{O} \\
\mathcal{F}^I &= \begin{bmatrix}
\dot{m} \ddot{u} + (\dot{\omega} + \ddot{\omega}) m \dot{\eta} \\
\ddot{\omega} m \ddot{u} + \dot{\omega} m \ddot{\eta}
\end{bmatrix}
\end{align*}
\]  

(4.25)

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

\[
\begin{align*}
\dot{E}_1 &= \dot{E}_0 + \dot{u}' \\
\mathcal{C} &= \begin{bmatrix}
C_{11} & C_{12} \\
C_{21} & C_{22}
\end{bmatrix}
\end{align*}
\]  

(4.26)

**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

\[
\mathbf{f}_d = \mu \mathcal{C} \begin{bmatrix}
\dot{\varepsilon} \\
\dot{\kappa}
\end{bmatrix}
\]  

(4.27)

where $\mu$ is a user-defined damping-coefficient diagonal matrix. The damping force can be recast in two separate parts, like $\mathcal{F}^C_d$ and $\mathcal{F}^D_d$ in the elastic force, as

\[
\begin{align*}
\mathcal{F}^C_d &= \begin{bmatrix}
\mathbf{E}_d \\
\mathbf{M}_d
\end{bmatrix} \\
\mathcal{F}^D_d &= \begin{bmatrix}
(\ddot{x}_0 + \dot{u}')^T \mathbf{E}_d \\
(\ddot{x}_0 + \dot{u}')^T \mathbf{E}_d
\end{bmatrix}
\end{align*}
\]  

(4.28)

The linearization of the structural damping forces are as follows:

\[
\begin{align*}
\Delta \mathcal{F}^C_d &= \mathcal{S}_d \begin{bmatrix}
\Delta u' \\
\Delta \dot{\kappa}
\end{bmatrix} + \mathcal{Q}_t \begin{bmatrix}
\Delta \dot{u} \\
\Delta \omega
\end{bmatrix} + \mathcal{T}_d \begin{bmatrix}
\Delta \ddot{u} \\
\Delta \ddot{\omega}
\end{bmatrix} + \mu \mathcal{C} \begin{bmatrix}
\Delta \dot{u}' \\
\Delta \dot{\kappa}'
\end{bmatrix} \\
\Delta \mathcal{F}^D_d &= \mathcal{P}_d \begin{bmatrix}
\Delta u' \\
\Delta \dot{\kappa}
\end{bmatrix} + \mathcal{Q}_d \begin{bmatrix}
\Delta \dot{u} \\
\Delta \omega
\end{bmatrix} + \mathcal{X}_d \begin{bmatrix}
\Delta \ddot{u} \\
\Delta \ddot{\omega}
\end{bmatrix} + \mathcal{Y}_d \begin{bmatrix}
\Delta \dot{u}' \\
\Delta \dot{\kappa}'
\end{bmatrix}
\end{align*}
\]  

(4.29)
where the newly introduced matrices are defined as

\[
\begin{align*}
\mathbf{S}_d &= \mu \mathbf{C} \begin{bmatrix} \bar{\omega}^T & 0 \\ 0 & \bar{\omega}^T \end{bmatrix} \\
\mathbf{O}_d &= \begin{bmatrix} 0 \\ \mu C_{11} (\dot{u} - \bar{\omega} \bar{E}_1) - \bar{F}_d \\ 0 \\ \mu C_{12} (\dot{u} - \bar{\omega} \bar{E}_1) - \bar{M}_d \end{bmatrix} \\
\mathbf{G}_d &= \begin{bmatrix} 0 \\ C_{11} \mu T \bar{\omega}^T \bar{E}_1 \\ 0 \\ C_{12} \mu T \bar{E}_1 \end{bmatrix} \\
\mathbf{P}_d &= \begin{bmatrix} \bar{F}_d + \bar{E}_1^T \mu C_{11} \bar{\omega}^T \bar{E}_1^T \mu C_{12} \bar{\omega}^T \\ 0 \\ \bar{E}_1 \end{bmatrix} \\
\mathbf{Q}_d &= \begin{bmatrix} 0 \\ 0 \\ \bar{E}_1 \end{bmatrix} \\
\mathbf{R}_d &= \begin{bmatrix} 0 \\ 0 \\ \bar{E}_1 \end{bmatrix} \\
\mathbf{Q}_d &= \begin{bmatrix} 0 \\ 0 \\ \bar{E}_1 \end{bmatrix} \\
\mathbf{G}_d &= \begin{bmatrix} 0 \\ 0 \\ \bar{E}_1 \end{bmatrix} \\
\mathbf{S}_d &= \mu \mathbf{C} \begin{bmatrix} \bar{\omega}^T & 0 \\ 0 & \bar{\omega}^T \end{bmatrix}
\end{align*}
\]

(4.30)

where \(O_{12}\) and \(G_{12}\) are the \(3 \times 3\) sub matrices of \(O\) and \(G\) as \(C_{12}\) in Eq. (4.26).

**Convergence Criterion and Generalized-\(\alpha\) Time Integrator**

The system of nonlinear equations in Eqs. (4.10) are solved using the Newton-Raphson method with the linearized form in Eq. (4.22). In the present implementation, an energy-like stopping criterion has been chosen, which is calculated as

\[
|\Delta U^{(i)} (t+\Delta t)^T R - (t+\Delta t)^T F^{(i-1)} | \
\leq |\epsilon_E \left( \Delta U^{(1)} (t+\Delta t)^T R - (t+\Delta t)^T F \right) |
\]

(4.31)

where \(|·|\) denotes the absolute value, \(\Delta U\) is the incremental displacement vector, \(R\) is the vector of externally applied nodal point loads, \(F\) is the vector of nodal point forces corresponding to the internal element stresses, and \(\epsilon_E\) 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-\(\alpha\) 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-\(\alpha\) method can be found in [CH93] [Bau10].

**Calculation of Reaction Loads**

Since the root motion of the wind turbine blade, including displacements and rotations, translational and angular velocities, and translational and angular accelerates, are prescribed as inputs to BeamDyn either by the driver (in stand-alone mode) or by FAST glue code (in FAST-coupled mode), the reaction loads at the root are needed to satisfy equality of the governing equations. The reaction loads at the root are also the loads passing from blade to hub in a full turbine analysis.

The governing equations in Eq. (4.10) can be recast in a compact form

\[
\mathbf{F}^I - \mathbf{F}^{C'} + \mathbf{F}^D = \mathbf{F}^{ext}
\]

(4.32)
with all the vectors defined in Section [sec:LinearProcess]. At the blade root, the governing equation is revised as

$$\mathbf{F}^I - \mathbf{F}^C + \mathbf{F}^D = \mathbf{F}^{ext} + \mathbf{F}^R$$

(4.33)

where $\mathbf{F}^R = [\mathbf{F}^R \quad \mathbf{M}^R]^T$ is the reaction force vector and it can be solved from Eq. (4.33) 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. (4.32) are recast as

$$\mathbf{F}^A + \mathbf{F}^V - \mathbf{F}^C + \mathbf{F}^D = \mathbf{F}^{ext}$$

(4.34)

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

$$\mathbf{F}^A = \begin{bmatrix} m\ddot{\mathbf{u}} + \dot{\omega}m \eta \\ m\dot{\eta} \ddot{\mathbf{u}} + \rho \omega^2 \end{bmatrix}$$

$$\mathbf{F}^V = \begin{bmatrix} \ddot{\omega} m \eta \\ \dot{\omega} \rho \omega^2 \end{bmatrix}$$

(4.35)

The blade loads are thus defined as

$$\mathbf{F}^{BF} = \mathbf{F}^{V} - \mathbf{F}^C + \mathbf{F}^D$$

(4.36)

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

**4.3.6 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 $6 \times 6$ 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 $6 \times 6$ mass/stiffness matrices, etc.
- Extending applications in FAST to other slender structures in the wind turbine system, for example, tower, mooring lines, and shaft.
- Developing a simplified form of GEBT with only rotational DOFs (bending, torsion) for computational efficiency.
4.3.7 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_β 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, “Bo” is prefixed to each output name, where α is a number in the range [1,3], corresponding to the blade number. The outputs are expressed in one of the following three coordinate systems:

- **r**: a floating reference coordinate system fixed to the root of the moving beam; when coupled to FAST for blades, this is equivalent to the IEC blade (b) coordinate system.
- **l**: a floating coordinate system local to the deflected beam.
- **g**: the global inertial frame coordinate system; when coupled to FAST, this is equivalent to FAST’s global inertial frame (i) coordinate system.

4.4 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).

4.4.1 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
<table>
<thead>
<tr>
<th>Channel Name(s)</th>
<th>Units</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RootFxr, RootFyr, RootFzr</td>
<td>(N), (N), (N)</td>
<td>Root reaction forces expressed in r</td>
</tr>
<tr>
<td>RootMxr, RootMyr, RootMzr</td>
<td>(N m), (N m), (N m)</td>
<td>Root reaction moments expressed in r</td>
</tr>
<tr>
<td>TipTDxr, TipTDyr, TipTDzr</td>
<td>(m), (m), (m)</td>
<td>Tip translational deflection (relative to the undeflected position) expressed in r</td>
</tr>
<tr>
<td>TipRDxr, TipRDyr, TipRDzr</td>
<td>(-), (-), (-)</td>
<td>Tip angular/rotational deflection Wiener-Milenkovic parameter (relative to the undeflected orientation) expressed in r</td>
</tr>
<tr>
<td>TipTVXg, TipTVYg, TipTVZg</td>
<td>(m/s), (m/s), (m/s)</td>
<td>Tip translational velocities (absolute) expressed in g</td>
</tr>
<tr>
<td>TipRVXg, TipRVYg, TipRVZg</td>
<td>(deg/s), (deg/s), (deg/s)</td>
<td>Tip angular/rotational velocities (absolute) expressed in g</td>
</tr>
<tr>
<td>TipTAXg, TipTAYg, TipTAZg</td>
<td>(m/s$^2$), (m/s$^2$), (m/s$^2$)</td>
<td>Tip translational accelerations (absolute) expressed in g</td>
</tr>
<tr>
<td>TipRAXg, TipRAYg, TipRAZg</td>
<td>(deg/s$^2$), (deg/s$^2$), (deg/s$^2$)</td>
<td>Tip angular/rotational accelerations (absolute) expressed in g</td>
</tr>
<tr>
<td>NβFxl, NβFyl, NβFzl</td>
<td>(N), (N), (N)</td>
<td>Sectional force resultants at Nβ expressed in l</td>
</tr>
<tr>
<td>NβMxl, NβMyl, NβMzl</td>
<td>(N m), (N m), (N m)</td>
<td>Sectional moment resultants at Nβ expressed in l</td>
</tr>
<tr>
<td>NβTDxr, NβTDyr, NβTDzr</td>
<td>(m), (m), (m)</td>
<td>Sectional translational deflection (relative to the undeflected position) at Nβ expressed in r</td>
</tr>
<tr>
<td>NβRDxr, NβRDyr, NβRDzr</td>
<td>(-), (-), (-)</td>
<td>Sectional angular/rotational deflection Wiener-Milenkovic parameter (relative to the undeflected orientation) at Nβ expressed in r</td>
</tr>
<tr>
<td>NβTVXg, NβTVYg, NβTVZg</td>
<td>(m/s), (m/s), (m/s)</td>
<td>Sectional translational velocities (absolute) at Nβ expressed in g</td>
</tr>
<tr>
<td>NβRVXg, NβRVYg, NβRVZg</td>
<td>(deg/s), (deg/s), (deg/s)</td>
<td>Sectional angular/rotational velocities (absolute) at Nβ expressed in g</td>
</tr>
<tr>
<td>NβTAXg, NβTAYg, NβTAZg</td>
<td>(m/s$^2$), (m/s$^2$), (m/s$^2$)</td>
<td>Sectional translational accelerations (absolute) at Nβ expressed in g</td>
</tr>
<tr>
<td>NβRAXg, NβRAYg, NβRAZg</td>
<td>(deg/s$^2$), (deg/s$^2$), (deg/s$^2$)</td>
<td>Sectional angular/rotational accelerations (absolute) at Nβ expressed in g</td>
</tr>
<tr>
<td>NβPFxl, NβPFyl, NβPFzl</td>
<td>(N), (N), (N)</td>
<td>Applied point forces at Nβ expressed in l</td>
</tr>
<tr>
<td>NβPMxl, NβPMyl, NβPMzl</td>
<td>(N m), (N m), (N m)</td>
<td>Applied point moments at Nβ expressed in l</td>
</tr>
<tr>
<td>NβDFxl, NβDFyl, NβDFzl</td>
<td>(N/m), (N/m), (N/m)</td>
<td>Applied distributed forces at Nβ expressed in l</td>
</tr>
<tr>
<td>NβDMxl, NβDMyl, NβDMzl</td>
<td>(N m/m), (N m/m), (N m/m)</td>
<td>Applied distributed moments at Nβ expressed in l</td>
</tr>
</tbody>
</table>

Fig. 4.14: BeamDyn Output Channel List
lattice or tubular tower. The wind turbine can be modeled on land or offshore on fixed-bottom or floating substructures. FAST is based on advanced engineering models derived from fundamental laws, but with appropriate simplifications and assumptions, and supplemented where applicable with computational solutions and test data.

The aerodynamic models use wind-inflow data and solve for the rotor-wake effects and blade-element aerodynamic loads, including dynamic stall. The hydrodynamics models simulate the regular or irregular incident waves and currents and solve for the hydrostatic, radiation, diffraction, and viscous loads on the offshore substructure. The control and electrical system models simulate the controller logic, sensors, and actuators of the blade-pitch, generator-torque, nacelle-yaw, and other control devices, as well as the generator and power-converter components of the electrical drive. The structural-dynamics models apply the control and electrical system reactions, apply the aerodynamic and hydrodynamic loads, adds gravitational loads, and simulate the elasticity of the rotor, drivetrain, and support structure. Coupling between all models is achieved through a modular interface and coupler.

### 4.4.2 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’]
  - abcd1234 is the first 8 characters of the current commit hash
  - dirty denotes that local changes have been made but not committed
- Because all modules are contained in the same repository, the version numbers of each module have been eliminated and now use the OpenFAST version number (starting from OpenFAST v1.0.0) though old documentation may still refer to old version numbers
- The OpenFAST regression test baseline solutions (formerly the Certification Tests or CertTest) reside in a standalone repository: https://github.com/openfast/r-test (starting from OpenFAST v1.0.0)
- Unit testing has been introduced at the subroutine level (starting with BeamDyn from OpenFAST v1.0.0).
- An online documentation system has been established to replace existing documentation of FAST v8: http://openfast.readthedocs.io/ during the transition to OpenFAST, most user-related documentation is still provided through the NWTC Information Portal, https://nwtc.nrel.gov
- Cross platform compiling is accomplished with CMake on macOS, Linux, and Cygwin (Windows) systems
- Visual Studio Projects (VS-Build) are provided for compiling OpenFAST on Windows (starting from OpenFAST v1.0.0), but the development team is working to automate the generation of Visual Studio build files via CMake in a future release

### 4.4. FAST v8 and the transition to OpenFAST
OpenFAST Documentation, Release v2.3.0

- GitHub Issues has been made the primary platform for developers to report and track bugs, request feature enhancements, and to ask questions related to the source code, compiling, and regression/unit testing; general user-related questions on OpenFAST theory and usage should still be handled through the forum at https://wind.nrel.gov/forum/wind

- A new API has been added that provides a high level interface to run OpenFAST through a C++ driver code helping to interface OpenFAST with external programs like CFD solvers written in C++ (starting in OpenFAST v1.0.0)

4.4.3 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 macOS, Linux, and Cygwin (Windows) systems
  - An online documentation system has been established to replace existing documentation of FAST v8: http://openfast.readthedocs.io/
  - GitHub Issues has been made the primary platform for developers to report and track bugs, request feature enhancements, and to ask questions related to the source code, compiling, and regression/unit testing; general user-related questions on OpenFAST theory and usage should still be handled through the forum at https://wind.nrel.gov/forum/wind

- The AeroDyn v15 aerodynamics module has been significantly updated. The blade-element/momentum theory (BEMT) solution algorithm has been improved as follows:
  - BEMT now functions for the case where the undisturbed velocity along the x-direction of the local blade coordinate system (Vx) is less than zero
  - BEMT no longer aborts when a valid value of the inflow angle (ϕ) cannot be found; in this case, the inflow angle is computed geometrically (without induction)
  - The inflow angle (ϕ) is now initialized on the first call instead of defaulting to using ϕ = 0, giving better results during simulation start up
  - When hub- and/or tip-loss are enabled (HubLoss = True and/or TipLoss = True), tangential induction (a’) is set to 0 instead of -1 at the root and/or tip, respectively (axial induction (a) is still set to 1 at the root and/or tip)
  - The BEMT solution has been made more efficient
  - In addition, several bugs in AeroDyn v15 have been fixed, including:
    - Fixed a bug whereby when hub- and/or tip-loss are enabled (HubLoss = True and/or TipLoss = True) along with the Pitt/Peters skewed-wake correction (SkewMod = 2), BEMT no longer modifies the induction factors at the hub and/or tip, respectively
    - Fixed a bug whereby the time series was affected after the linearization analysis with AeroDyn coupled to OpenFAST when frozen wake is enabled (FrozenWake = True)
• The BeamDyn finite-element blade structural-dynamics model has undergone an extensive cleanup of the source code. A bug in an off-diagonal term in the structural damping-induced stiffness (i.e., representing a change in the damping force with beam displacement) has been corrected.

• A new module for user-specified platform loading (ExtPtfm) has been introduced. ExtPtfm allows the user to specify 6x6 added mass, damping, and stiffness matrices, as well as a 6x1 load vector to define loads to be applied to ElastoDyn’s tower base/platform, e.g., to support the modeling of substructures or foundations through a super-element representation (with super-element derived from external software). ExtPtfm also provides the user with a module to customize with more advanced platform applied loads. Module ExtPtfm can be enabled by setting CompSub to 2 in the FAST primary input file (a new option) and setting SubFile to the name of the file containing the platform matrices and load time history, but setting CompSub to 2 requires one to disable hydrodynamics (by setting CompHydro to 0). Please note that the introduction of option 2 for CompSub represents a minor input file change (the only input file change in OpenFAST v0.1.0), but the MATLAB conversion scripts have not yet been updated.

• In the ServoDyn control and electrical-system module, the units and sign of output parameter YawMom have been corrected

• In the InflowWind wind-inflow module, the ability to use TurbSim-generated tower wind data files in Bladed-style format was corrected

• Minor fixes were made to the error checking in ElastoDyn

v1.0.0 (September 2017)

• Organizational changes:
  – The FAST program has been renamed OpenFAST
  – Version numbering has been updated for OpenFAST (see Section 4.3.2 for details)
  – The OpenFAST regression test baseline solutions (formerly the Certification Tests or CertTest) reside in a standalone repository: https://github.com/openfast/r-test
  – Unit testing has been introduced at the subroutine level (starting with BeamDyn)
  – The online documentation (http://openfast.readthedocs.io/en/latest/index.html) has been extensively updated with additions for installation, testing, user (AeroDyn BeamDyn, transition from FAST v8, release notes), and developer guides, etc
  – The scripts for compiling OpenFAST using CMake on macOS, Linux, and Cygwin (Windows) systems have been updated, including the ability to compile in single precision and building with Spack
  – Visual Studio Projects (VS-Build) are provided for compiling OpenFAST on Windows
  – TurbSim has been included in the OpenFAST repository

• The AeroDyn aerodynamics module has been updated:
  • Added a cavitation check for marine hydrokinetic (MHK) turbines. This includes the additions of new input parameters CavitCheck, Patm, Pvap, and FluidDepth in the AeroDyn primary input file, the addition of the Cpmmin 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 NumCrtctn = 0)
  • Fixed a bug in the Line2-to-Point mapping of loads as per GitHub issue #8 (https://github.com/OpenFAST/openfast/issues/8). Previously, the augmented mesh was being formed using an incorrect projection, thus causing strange transfer of loads in certain cases. This could cause issues in the coupling between ElastoDyn and AeroDyn and/or in the coupling between HydroDyn and SubDyn
  • Added an otherwise undocumented feature for writing binary output without compression to support the new regression testing. The new format is available by setting OutFileFmt to 0 in the FAST primary input file.
  • A new API has been added that provides a high level interface to run OpenFAST through a C++ driver code. The primary purpose of the C++ API is to help interface OpenFAST to external programs like CFD solvers that are typically written in C++.
  • The TurbSim wind-inflow turbulence preprocessor was updated:
    • The API spectra was corrected
    • Several minor bugs were fixed.

### 4.4.4 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.

4.5 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_OPENFAST_CPP_API flag.

A sample glue-code FAST_Prog.cpp is provided as a demonstration of the usage of the C++ API. The glue-code allows for the simulation of multiple turbines using OpenFAST in parallel over multiple processors. The glue-code takes a single input file named cDriver.i (download).

```yaml
# -*- mode: yaml -*-
#
# C++ glue-code for OpenFAST - Example input file
#

#Total number of turbines in the simulation
nTurbinesGlob: 3
#Enable debug outputs if set to true
debug: False
#The simulation will not run if dryRun is set to true
dryRun: False
#Flag indicating whether the simulation starts from scratch or restart
simStart: init # init/trueRestart/restartDriverInitFAST
#Start time of the simulation
tStart: 0.0
#End time of the simulation. tEnd <= tMax
tEnd: 1.0
#Max time of the simulation
tMax: 4.0
#Time step for FAST. All turbines should have the same time step.
dtFAST: 0.00625
#Restart files will be written every so many time steps
nEveryCheckPoint: 160

Turbine0:
  #The position of the turbine base for actuator-line simulations
turbine_base_pos: [0.0, 0.0, 0.0]
  #The number of actuator points along each blade for actuator-line simulations
num_force_pts_blade: 0
  #The number of actuator points along the tower for actuator-line simulations.
num_force_pts_tower: 0
  #The checkpoint file for this turbine when restarting a simulation
restart_filename: "banana"
  #The FAST input file for this turbine
FAST_input_filename: "t1_Test05.fst"
  #A unique turbine id for each turbine
```

4.5. C++ API Users Guide 77
4.5.1 Command line invocation

\texttt{mpiexec -np <N> openfastcpp}

4.5.2 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. \(\text{simStart}\) takes on one of three values:
  - \texttt{init} - Use this option when starting a simulation from \(t=0s\).
  - \texttt{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.
  - \texttt{restartDriverInitFAST} - When the \texttt{restartDriverInitFAST} option is selected, the individual turbine models start from \(t=0s\) and run up to the specified restart time using the inflow data stored at the actuator nodes from a hdf5 file. The C++ API stores the inflow data at the actuator nodes in a hdf5 file at every OpenFAST time step and then reads it back when using this restart option. This restart option is especially useful when the glue code is a CFD solver.

- **tStart**
  Start time of the simulation

- **tEnd**
  End time of the simulation. \(tEnd <= tMax\)
4.5.3 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
CHAPTER 5

Developer Documentation

Our goal as developers of OpenFAST is to ensure that it is well tested, well documented, and self-sustaining software. To that end, we continually work to improve the documentation and test coverage along with feature additions and improvements. This section of the documentation outlines the processes and procedures we have established for external developers to work with the NREL OpenFAST team on code development.

If you’d like to help with general OpenFAST development or work on a particular feature, then first install OpenFAST following the installation instructions for your machine. Next, verify that your installation is valid by running the test suite following the testing instructions. While OpenFAST is compiling, we encourage reading through the Development Philosophy section to understand the general workflow for individual and coordinated development. Finally, be sure to review the GitHub workflow to avoid any merge or code conflicts.

With development happening in parallel between NREL, industry partners, and universities, NREL relies on GitHub to coordinate efforts:

- GitHub Issues is the place to ask usage or development questions, report bugs, and suggest code enhancements
- GitHub Pull Requests is the place for engaging with the OpenFAST team to have your new code merged into the main repository.

For other questions regarding OpenFAST, please contact Mike Sprague.

Tip: The following sections provide valuable guidance on workflow and development tips which make the process more efficient and effective:
- Working with OpenFAST on GitHub
- Code Style
- Debugging OpenFAST
5.1 API Reference

Some subroutines and derived types throughout the source code have in-source documentation which is compiled with Doxygen. Though this portion of the documentation is always under development, the existing API reference can be found in the following pages:

- Main Page
- Index of Types
- Source Files

5.2 Development Philosophy

OpenFAST is intended to be a self-sustaining community developed software. A couple of tenets of this goal are that the code should be reasonably straightforward to comprehend and manageable to improve. With that in mind, we expect that new capabilities will include adequate testing and documentation.

We have the following guidance for developers:

- When fixing a bug, first introduce a unit test that exposes the bug, fix the bug, and submit a Pull Request. See Testing OpenFAST and Working with OpenFAST on GitHub for more information.

- When adding a new feature, create appropriate automated unit and regression tests as described in Testing OpenFAST. The objective is to create a GitHub pull request that provides adequate verification and validation so that the NREL OpenFAST developer team can merge the pull request with confidence that the new feature is “correct” and supports our goal of self-sustaining software. See Pull Requests for more information on submitting a pull request.

- If a code modification affects regression test results in an expected manner, work with the NREL OpenFAST developer team to upgrade the regression test suite via a GitHub issue or pull request at the openfast/r-test repository.

5.3 Development Guidelines

The following sections provide extended guidance on how to develop source code, interacting with the NREL OpenFAST team and other community contributors, and generally debugging and building out features.

5.3.1 Working with OpenFAST on GitHub

The majority of the collaboration and development for OpenFAST takes place on the GitHub repository. There, issues and pull requests are discussed and new versions are released. It is the best mechanism for engaging with the NREL OpenFAST team and other developers throughout the OpenFAST community.

Issues and work assignment

Issues should be opened with proper documentation and data to fully describe the problem or feature gap. It is here that communication and coordination should happen regarding ongoing work for new development, and developers should make clear any intention to complete a task.
Pull Requests

When a code modification is ready for review, a pull request should be submitted along with all appropriate documentation and tests. An NREL OpenFAST team member will assign a reviewer and work with the developer to have the code merged into the main repository.

New pull requests should contain the following:

- A description of the need for modifications
  - If the pull request fixes a bug, the accompanying GitHub issue should be referenced
- A summary of the work implemented
- Regression test results
  - If all tests pass, the summary print out should be provided
  - If any tests fail, an explanation of the failing cases and supporting data like plots should be included
- Updated unit tests, if applicable
- Updated documentation in applicable sections ready for compilation and deployment to readthedocs.

Git workflow and interacting with the main repository

OpenFAST development should follow “Git Flow” when interacting with the github repository. Git Flow is a well-defined and widely adopted workflow for using git that outlines safe methods of pushing and pulling commits to a shared repository. Maintaining Git Flow is critical to prevent remote changes from blocking your local development. This workflow is detailed nicely here and the chart below provides a high level perspective.
OpenFAST Specific Git Flow

It is important to consider how your current work will be affected by other developer’s commits and how your commits will affect other developers. On public branches, avoid using `git rebase` and never `force push`.

In OpenFAST development, the typical workflow follows this procedure:

1. Fork the OpenFAST repository on GitHub
2. Clone your new fork
   ```
   git clone https://github.com/<youruser>/OpenFAST
   ```
3. Add OpenFAST/OpenFAST as a remote named `upstream`

# This adds the remote
```bash
git remote add upstream https://github.com/OpenFAST/OpenFAST
```

# This downloads all the info in the remote, but it doesn’t change
# the local source code
```bash
git fetch --all
```

4. Create a feature branch for active development starting from the OpenFAST dev branch and check it out
```bash
git branch feature/a_great_feature upstream/dev
``` 
```bash
git checkout feature/a_great_feature
```

5. Add new development on feature/a_great_feature
```bash
git add a_file.f90
``` 
```bash
git commit -m "A message"
``` 
```bash
git push origin feature/a_great_feature
```

6. Update your feature branch with upstream
```bash
git pull upstream dev
``` 
```bash
git push origin feature/a_great_feature
```

7. Open a new pull request to merge <youruser>/OpenFAST/feature/a_great_feature into
   OpenFAST/OpenFAST/dev

## 5.3.2 Code Style

OpenFAST and its underlying modules are mostly written in Fortran adhering to the 2003 standard, but modules can be written in C or C++. The NWTC Programmer’s Handbook is the definitive reference for all questions related to
working with the FAST Framework and adding code to OpenFAST.

Generally, code should be written such that it is straightforward to read. Syntactic sugar or brevity should not detract from readability. The exception to this is in situations where performance requires poorly readable code. Here, comment blocks should be used to describe what is not readily apparent in the code. Indentation is typically three spaces and no tabs.

## 5.3.3 Developing Documentation

OpenFAST documentation is hosted on readthedocs. It is automatically generated through the readthedocs build system from both the master and dev branches whenever new commits are added. This documentation uses the restructured text markup language.

### Building this documentation locally

The documentation is compiled with Sphinx, which is a Python based tool. Install it and the other required Python packages listed in `openfast/docs/requirements.txt` with pip or another Python package manager.

These additional packages are optional and are not included in the requirements file:

- Doxygen
- Doxylink
- Graphviz

# Development Guidelines

## 5.3. Development Guidelines

85
• **LaTeX**

Doxygen and Graphviz can be installed directly from their website or with a package manager like `brew`, `yum`, or `apt`.

The result of building the documentation locally will be a set of HTML files and their accompanying required files. The main HTML file will exist `openfast/build/docs/html/index.html`. This file can be opened with any browser to view and navigate the locally-generated documentation as if it were any other web site.

**Pure python build**

If CMake and Make are not available on your system, the documentation can be generated directly with `sphinx`.

**Note:** This method does not generate the API documentation through Doxygen.

First, align your directory structure to the standard OpenFAST build by creating a directory at `openfast/build`. Then, move into `openfast/build` and run this command:

```
# sphinx-build -b <builder-name> <source-directory> <output-directory>
sphinx-build -b html ../docs ./docs/html
```

If this completes successfully, an html file will be created at `build/docs/html/index.html` which can be opened with any web browser.

**Building with CMake and Make**

In the OpenFAST directory, create a `build` directory and move into it. Then, run CMake with this flag: `DBUILD_DOCUMENTATION=ON`. CMake will configure the build system with the necessary files for building the documentation.

Next, run the command to compile the docs:

```
make docs
```

This will first build the Doxygen API documentation and then the Sphinx documentation. If this completes successfully, a html file will be created at `build/docs/html/index.html` which can be opened with any web browser.

The full procedure for configuring and building the documentation is:

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

If any modifications are made to the source files in `openfast/docs/source`, you can simply update the html files by executing the `make` command again.

The table below lists make-targets related to the documentation.

<table>
<thead>
<tr>
<th>Target</th>
<th>Command</th>
<th>Output location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full docs</td>
<td><code>make docs</code></td>
<td><code>openfast/build/docs/html/index.html</code></td>
</tr>
<tr>
<td>Full docs</td>
<td><code>make sphinx</code></td>
<td><code>openfast/build/docs/html/index.html</code></td>
</tr>
<tr>
<td>Doxygen API Reference</td>
<td><code>make doxygen</code></td>
<td></td>
</tr>
<tr>
<td>HTML only</td>
<td><code>make sphinx-html</code></td>
<td><code>openfast/build/docs/html/index.html</code></td>
</tr>
<tr>
<td>PDF only</td>
<td><code>make sphinx-pdf</code></td>
<td><code>openfast/build/docs/latex/Openfast.pdf</code></td>
</tr>
</tbody>
</table>
5.3.4 Types Files and the OpenFAST Registry

Being a modern software project, OpenFAST has a complex system of custom data types. In Fortran, these are known as “derived data types.” Each module contains a unique collection of derived types which may add on to but must comply with the OpenFAST Framework. The module types are generally auto-generated by an included program called *OpenFAST Registry*. The OpenFAST Registry is written in C and adapted from a similar utility used in WRF. Visit the OpenFAST Registry README for more information.

The OpenFAST Registry requires an input file to describe the necessary types for a given module. Generally, all module use a similar naming convention, `<Module>_Registry.txt`, and resulting Fortran code will be in a file called `<Module>_Types.f90`. For example, the AeroDyn OpenFAST Registry input file is located at `openfast/modules/aerodyn/src/AeroDyn_Registry.txt` and the resulting auto-generated Fortran source code is at `openfast/modules/aerodyn/src/AeroDyn_Types.f90`.

Since the types-modules are autogenerated, any changes to the data types directly should be expressed in the OpenFAST Registry input files so that the changes are not subsequently overwritten.

Compiling the OpenFAST Registry

The OpenFAST Registry is included in the OpenFAST build system through CMake. However, the default is to not compile the OpenFAST Registry executable and instead use the types modules that are included in git while compiling OpenFAST. To include the OpenFAST Registry in the build process and compile the Registry program, configure CMake with the `GENERATE_TYPES` flag:

```
cmake .. -DGENERATE_TYPES=ON
```

With `GENERATE_TYPES` enabled, CMake will configure the `openfast-registry` target to compile as a dependency of all other targets. The OpenFAST Registry executable will be found in `openfast/build/modules/openfast-registry/openfast-registry`.

Regenerating a types module

With the `GENERATE_TYPES` flag enabled, an additional step will be added to modules that are configured can make use of the OpenFAST Registry. The additional step will execute the OpenFAST Registry and regenerate the types module overwriting the existing modules. Any changes to the types module will be evident in git. For modules where the registry input file has not changed, the resulting types module will not change. However, for registry input files that have been modified, the output types module will be recompiled.

Adding a new types module

The process for adding a new types module follows Regenerating a types module closely. Here, an additional step is required to configure CMake to execute the Registry on the new input file and include the resulting types module in the compile step.

First, a new OpenFAST Registry input file must be created. Then, it must be configured to pass through the Registry in the corresponding module's `CMakeLists.txt`:
This is the control statement for allowing the Registry to execute
if (GENERATE_TYPES)

    # Here is the CMake wrapper-function to execute the Registry
    # syntax: generate_f90_types(<Registry input file> <output file location>)
    generate_f90_types(src/AeroDyn_Registry.txt ${CMAKE_CURRENT_LIST_DIR}/src/AeroDyn_Types.f90)
    generate_f90_types(src/New_Registry.txt ${CMAKE_CURRENT_LIST_DIR}/src/New_Types.f90)
endif()

Finally, the resulting types module must be added to the source files for the corresponding module:

```
# AeroDyn lib
set(AD_LIBS_SOURCES
    src/AeroDyn.f90
    src/AeroDyn_IO.f90
    src/AirfoilInfo.f90
    src/BEMT.f90
    src/DBEMT.f90
    src/BEMTUncoupled.f90
    src/UnsteadyAero.f90
    src/fmin_fcn.f90
    src/mod_root1dim.f90
    src/AeroDyn_Types.f90
    src/AirfoilInfo_Types.f90
    src/BEMT_Types.f90
    src/DBEMT_Types.f90
    src/UnsteadyAero_Types.f90

    # Add the new types module here
    src/New_Types.f90
)
```

With CMake properly configured, a message will display during the build process indicating that the OpenFAST Registry is executing:

```
[ 64%] Generating ../../../modules/aerodyn/src/New_Types.f90
----- FAST Registry ------------
----------------------------------------------------------
input file: /Users/rmudafor/Development/openfast/modules/aerodyn/src/New_Registry.txt
# more build process output will follow
```

And finally there should be an indication that the resulting types module is compiled:

```
Scanning dependencies of target aerodynlib
[ 70%] Building Fortran object modules/aerodyn/CMakeFiles/aerodynlib.dir/src/New_—Types.f90.o
```

### 5.3.5 Debugging OpenFAST

Being a Fortran project, OpenFAST can be challenging to debug and the process is unique for each system and environment. Keep in mind that some OpenFAST cases can be quite large in their memory footprint and may take a long time to reach the point of interest in the code. Choosing a test case carefully could save a significant amount time.
It may be helpful to write a small fortran program to verify that all debugging tools are set up properly before diving in to OpenFAST. Be sure to simulate a bug by doing something like accessing an array element that is not allocated and verify that you can catch the bug with a given set of tools.

**Note:** A requirement for all systems is to compile OpenFAST in **debug** mode.

**Debugging on Windows**

Windows developers using Intel tools can use Visual Studio solution included in the OpenFAST repository for debugging. This is a straightforward process with lots of support from Intel.

Otherwise, Windows developers compiling in Unix-style environments should proceed to **Debugging on Linux and macOS**.

**Debugging on Linux and macOS**

First, compile OpenFAST in debug mode by setting `CMAKE_BUILD_TYPE` to “Debug”. This can be done on the command line with:

```
cmake .. -D CMAKE_BUILD_TYPE=Debug
```

or by using `ccmake` to open the command line cmake gui to change it.

The GNU debugger, `gdb`, works well for debugging compiled code. It has a comprehensive command line interface which enables developers to add breakpoints and inspect variables.

Driving the debugger through an IDE can make inspecting the code much more efficient. One IDE known to work well is Visual Studio Code with the Native Debug extension. You can set up a launch configuration so that you can debug a particular OpenFAST case through the IDE. To do this, open the launch configuration and add a block similar to this:

```
{
    "name": "AOC_WSt",
    "type": "gdb",
    "request": "launch",
    "printCalls": false,
    "showDevDebugOutput": false,
    "valuesFormatting": "prettyPrinters",
    "gdbpath": "gdb",
    "target": "${workspaceRoot}/build/glue-codes/openfast/openfast",
    "cwd": "${workspaceRoot}/build/reg_tests/glue-codes/openfast/AOC_WSt/",
    "arguments": "${workspaceRoot}/build/reg_tests/glue-codes/openfast/AOC_WSt/AOC_WSt.fst",
}
```

**macOS-specific configuration**

GDB on macOS needs some configuration before the system allows it to take over a process. It is recommended that `gdb` be installed with homebrew

```
brew info gdb
brew install gdb
```
After that completes, be sure to follow the caveats to finish the installation. For gdb 8.2.1, it looks like this:

```
== Caveats

gdb requires special privileges to access Mach ports.
You will need to codesign the binary. For instructions, see:

https://sourceware.org/gdb/wiki/BuildingOnDarwin

On 10.12 (Sierra) or later with SIP, you need to run this:

```

```

```

echo "set startup-with-shell off" >> ~/.gdbinit
```

For Native Debug on macOS, you have to sort of hack the extension to allow breakpoints in fortran files by adding this line to .vscode/settings.json:

```
{
    "debug.allowBreakpointsEverywhere": true
}
```

### 5.3.6 Performance-Profiling and Optimization

The OpenFAST team has been engaged in performance-profiling and optimization work in an effort to improve the time-to-solution performance for the most computationally expensive use cases. This work is supported by Intel® through its designation of NREL as an Intel® Parallel Computing Center (IPCC).

After initial profiling and hotspot analysis, specific subroutines in the physics modules of OpenFAST were targeted for optimization. Among other takeaways, it was learned that the memory alignment of the derived data types could yield a significant increase in performance. Ultimately, tuning the Intel® tools to perform best on NREL’s hardware and adding high level multithreading yielded a maximum 3.8x time-to-solution improvement for one of the benchmark cases.

**Approach**

The general mechanisms identified for performance improvements in OpenFAST are:

- Intel® compiler suite and Intel® Math Kernel Library (Intel® MKL)
- Algorithmic improvements
- Memory-access optimization enabling more efficient cache usage
- Data type alignment allowing for SIMD vectorization
- Multithreading with OpenMP

To establish a path forward with any of these options, OpenFAST was first profiled with Intel® VTune™ Amplifier which provides a clear breakdown of time spent in the simulation. Then, the optimization report generated from the Intel® Fortran compiler was analyzed to determine area which were not autovectorized. Finally, Intel® Advisor was used to highlight areas of the code which the compiler identified as potentially improved with multithreading.

**Test cases**

Two OpenFAST test cases have been chosen to provide meaningful and realistic timing benchmarks. In addition to real-world turbine and atmospheric models, these cases are computationally expensive and expose the areas where performance improvements would make a difference.
5MW_Land_BD_DLL_WTurb

Download files here.

The physics modules used in this case are:

- BeamDyn
- InflowWind
- AeroDyn 15
- ServoDyn

This is a land based NREL 5-MW turbine simulation using BeamDyn as the structural module. It simulates 20 seconds with a time step size of 0.001 seconds and executes in 3m 55s on NREL's Peregrine supercomputer.

5MW_OC4Jckt_DLL_WTurb_WavesIrr_MGrowth

Download files here.

This is an offshore, fixed-bottom NREL 5-MW turbine simulation with the majority of the computational expense occurring in the HydroDyn wave-dynamics calculation.

The physics modules used in this case are:

- ElastoDyn
- InflowWind
- AeroDyn 15
- ServoDyn
- HydroDyn
- SubDyn

It simulates 60 seconds with a time step size of 0.01 seconds and executes in 20m 27s on NREL's Peregrine supercomputer.

Profiling

The OpenFAST test cases were profiled with Intel® VTune™ Amplifier to identify performance hotspots. Being that the two test cases exercise difference portions of the OpenFAST software, different hotspots were identified. In all cases and environment settings, the majority of the CPU time was spent in fast_solution loop which is a high-level subroutine that coordinates the solution calculation from each physics module.

LAPACK

In the offshore case, the LAPACK usage was identified as a performance load. Within the fast_solution loop, the calls to the LAPACK function dgetrs consume 3.3% of the total CPU time.
BeamDyn

While BeamDyn provides a high-fidelity blade-response calculation, it is a computationally expensive module. Initial profiling highlighted the `bd_elementmatrixga2` subroutine, in particular, as a hotspot. However, initial attempts to improve performance in BeamDyn highlighted needs for algorithmic improvements and refinements to the module’s data structures.

Results

Though work is ongoing, OpenFAST time-to-solution performance has improved and the performance potential is better understood.

Some key outcomes from the first year of the IPCC project are as follows:

- Use of Intel® compiler and MKL library provides dramatic speedup over GCC and LAPACK
  - Additional significant gains are possible through MKL threading for offshore simulations
- Offshore-wind-turbine simulations are poorly load balanced across modules
  - Land-based-turbine configuration better balanced
  - OpenMP Tasks are employed to achieve better load-balancing
- OpenMP module-level parallelism provides significant, but limited speed up due to imbalance across different module tasks
- Core algorithms need significant modification to enable OpenMP and SIMD benefits
Speedup - Intel® Compiler and MKL

By employing the standard Intel® developer tools tech stack, a performance improvement over GNU tools was demonstrated:

<table>
<thead>
<tr>
<th>Compiler</th>
<th>Math Library</th>
<th>5MW_Land_BD_DLL_WTurb</th>
<th>5MW_OC4Jckt_DLL_WTurb_WavesIrr_MGrowth</th>
</tr>
</thead>
<tbody>
<tr>
<td>GNU</td>
<td>LAPACK</td>
<td>2265 s (1.0x)</td>
<td>673 s (1.0x)</td>
</tr>
<tr>
<td>Intel® 17</td>
<td>LAPACK</td>
<td>1650 s (1.4x)</td>
<td>251 s (2.7x)</td>
</tr>
<tr>
<td>Intel® 17</td>
<td>MKL</td>
<td>1235 s (1.8x)</td>
<td>—</td>
</tr>
<tr>
<td>Intel® 17</td>
<td>MKL Multithreaded</td>
<td>722 s (3.1x)</td>
<td>—</td>
</tr>
</tbody>
</table>

Speedup - OpenMP at FAST_Solver

A performance improvement was demonstrated by adding OpenMP directives to the FAST_Solver module. Although the solution scheme is not well balanced, parallelizing mesh mapping and calculation routines resulted in the following speedup:

<table>
<thead>
<tr>
<th>Compiler</th>
<th>Math Library</th>
<th>5MW_Land_BD_DLL_WTurb</th>
<th>5MW_OC4Jckt_DLL_WTurb_WavesIrr_MGrowth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel® 17</td>
<td>MKL - 1 thread</td>
<td>1073 s (2.1x)</td>
<td>100 s (6.7x)</td>
</tr>
<tr>
<td>Intel® 17</td>
<td>MKL - 8 threads</td>
<td>597 s (3.8x)</td>
<td>—</td>
</tr>
</tbody>
</table>

Ongoing Work

The next phase of the OpenFAST performance improvements are focused in two key areas:

1. Implementing the outcomes from previous work throughout OpenFAST modules and glue codes
2. Preparing OpenFAST for efficient execution on Intel®’s next generation platforms

5.3.7 Versioning

OpenFAST follows semantic versioning. In summary, this means that with a version number as MAJOR.MINOR.PATCH, the components will be incremented as follows:

- MAJOR version when introducing incompatible API changes,
- MINOR version when adding functionality in a backwards-compatible manner, and
- PATCH version when making backwards-compatible bug fixes.

For example, OpenFAST-v1.0.0-123-gabcd1234-dirty describes OpenFAST as:
<table>
<thead>
<tr>
<th>Version Component</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>v1.0.0</td>
<td>MAJOR.MINOR.PATCH numbering system; corresponds to a tagged commit made by NREL on GitHub</td>
</tr>
<tr>
<td>123-g</td>
<td>Number of additional commits after the most recent tag for a build (the (-g) is for (git))</td>
</tr>
<tr>
<td>abcd1234</td>
<td>First 8 characters of the current commit hash</td>
</tr>
<tr>
<td>dirty</td>
<td>Denotes that local changes have been made but not committed; omitted if there are no local changes</td>
</tr>
</tbody>
</table>
The OpenFAST software, including its underlying modules, are licensed under Apache License Version 2.0 open-source license.
CHAPTER 7

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.
Acknowledgements

This software is developed and maintained by researchers at the National Renewable Energy Laboratory with funding from U.S. Department of Energy Wind Energy Technology Office through the Atmosphere to electrons (A2e) research initiative.

NREL gratefully acknowledges development contributions from the following organizations:

- Envision Energy USA, Ltd
- Brigham Young University

NREL gratefully acknowledges additional development support through designation as an Intel® Parallel Computing Center (IPCC).


Index

D
ddebug
  input file parameter, 78
dryRun
  input file parameter, 78
dtFAST
  input file parameter, 79

F
FAST_input_filename
  input file parameter, 79

I
input file parameter
ddebug, 78
dryRun, 78
dtFAST, 79
FAST_input_filename, 79
nEveryCheckPoint, 79
nTurbinesGlob, 78
num_force_pts_blade, 79
num_force_pts_tower, 79
restart_filename, 79
simStart, 78
tEnd, 78
tMax, 78
tStart, 78
turb_id, 79
turbine_base_pos, 79

N
nEveryCheckPoint
  input file parameter, 79
nTurbinesGlob
  input file parameter, 78
num_force_pts_blade
  input file parameter, 79
num_force_pts_tower
  input file parameter, 79

R
restart_filename
  input file parameter, 79

S
simStart
  input file parameter, 78

T
tEnd
  input file parameter, 78
tMax
  input file parameter, 78
tStart
  input file parameter, 78
turb_id
  input file parameter, 79
turbine_base_pos
  input file parameter, 79