

# GETTING STARTED...

## Author

Bart Doekemeijer

*PhD Candidate at the Delft University of Technology*

## Latest update

December 19, 2019

## Description

This document contains a basic introduction to working with SOWFA on the DCSC cluster at the Delft University of Technology. Most of the SOWFA examples are universal, and should be useable on any cluster, but might need some alteration in the *runscript.solve\** files. The example cases include wind farm simulations with ADM/ALMAAdvanced rotor models, with and without turbulent inflow (precursors), setting up such precursors, and simulations with a wind farm controller code coupled to SOWFA, where the controller is written in MATLAB.

The example cases presented in this workshop can be used to base your own, new simulations on. However, not all example cases are necessarily realistic, and they are presented exclusively as *example* cases.

## Know before workshop

- Understand basic linux commands (ls, cd, cp, mv, less)

## Preparation before workshop

- Get cluster access (hpc06.tudelft.nl)
- Download and install [PuTTY](#) (Windows users only)
- Download 3mE cluster monitoring tool from [Github](#)
- Download SOWFA\_tools from [Github](#)
- Download DTU10MW turbine files for SOWFA from [Github](#)
- Download and install [ParaView](#) visualization software
- Download and install [Notepad++](#) or another editor
- Download and install [Webdrive](#) and/or [Filezilla](#) to connect to the cluster

**Bart Doekemeijer**

*Delft, December 19, 2019*

# HPC CLUSTER AND SOWFA PRECURSOR CHEATSHEET

## Job scheduler commands

Command	Description
qstat	Show all jobs currently running
qstat -u <netid>	Show all jobs of that user
qstat -l <department>	Show all jobs submitted to that cluster, e.g., 'dcsc' or 'pme'
qstat -f <job_id>	Show details of a particular job
qdel <job_id>	Cancel a particular job
qsub <job_filename>	Submit a particular job to the cluster

## Precursor post-processing guidelines

Here are the basic instructions for running and coupling a precursor to SOWFA windPlantSolver simulations.

### Generating the precursor data

1. Run your precursor for a long period of time so that turbulent structures can arise, and then let a quasi-equilibrium form. This is typically 20,000 seconds. You do not need to output anything during these simulations.
2. Update your controlDict to include the function "sampling/boundaryDataPre". (for example: controlDict.2). You can also add other sampling functions for visualization, but this is optional. In the boundaryDataPre file, you can select which domain side to record. Since the inflow is typically from the west, the default option is to record the west domain patch.
3. Continue/restart your simulation from 20,000 seconds to 22,000 seconds using this new controlDict. This will generate you 2,000 seconds of precursor data. You should see a folder called boundaryDataPre in your postProcessing folder.

### Preparing the precursor data

4. In the postProcessing folder, you will see a folder called SourceHistory and a folder called boundaryDataPre.
5. The folder 'boundaryDataPre' will contain folders 20000.5, 20001, 20001.5, and so on. However, we are missing the very first folder. Therefore, we have to copy folder '20000.5' to a new folder called '20000'.

```
cd postProcessing
cp -r boundaryDataPre/20000.5 boundaryDataPre/20000
```

6. Now, copy the contents from "SOWFA\tools\boundaryDataConversion" to the postProcessing folder, and run the script makeBoundaryDataFiles.west.sh (note: you may have to change the first line of makeBoundaryDataFiles/data.py and of

makeBoundaryDataFiles/points.py to fit your cluster). This will generate a folder called boundaryData containing the inflow data for the 2,000 seconds of simulation with correct formatting. This usually takes about 30 minutes on our cluster.

7. Copy the contents from “SOWFA\tools\sourceDataConversion” to the postProcessing folder, and run the script sourceHistoryRead.py. This will create a file called sources in your postProcessing folder.
8. Create a new folder in your main case directory (i.e., next to the folders constant and system) called drivingData. Copy the file *sources* and the folder *boundaryData* to drivingData.
9. In your main case directory, gather the state information from the various processors at time = 20,000 using OpenFOAM. This will be your initial condition for the wind farm simulation. Use this command:

```
reconstructPar -time 20000 -fields '(k kappat nuSgs p_rgh qwall Rwall T U)'
```

#### Coupling to a wind farm simulation

10. In your wind farm simulation folder, open the file runscript.preprocess. Then update the precursorDir to the directory of your precursor case. This folder should contain drivingData (containing the sources file and the boundaryData folder) and the folder 20000 (containing your initial conditions). Also, set startTime=20000, and make sure this matches with your controlDict.
11. Setup the simulation settings in setUp. Run runscript.preprocess and then submit (qsub) a HPC job for runscript.solve.1.

# PART I: INSTALLING SOWFA

We first present how the user can install SOWFA on their user account on the DCSC cluster. Note that the TUDelft SOWFA repository contains a function called *install\_SOWFA\_tud*, which is a bash script that immediately installs SOWFA for the user. However, to provide insight, each step of that script is presented in this workshop. This part contains three items: the installation of SOWFA without ZeroMQ (which is the wind-farm-controller coupling code), the installation of ZeroMQ, and finally the installation of SOWFA with ZeroMQ.

## 1. Installation of SOWFA without ZeroMQ (5 min)

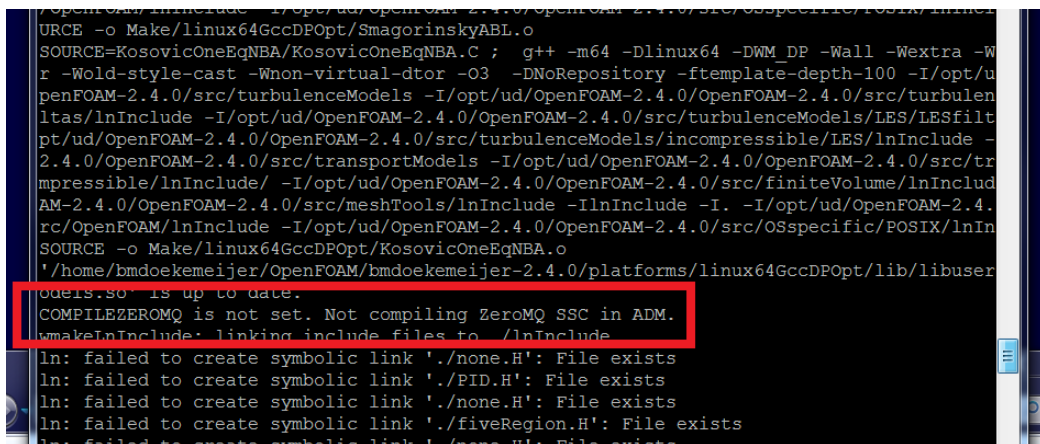
- a. Load the OpenFOAM 2.4.0 module from the cluster. Now go to your home directory. Create an empty folder called OpenFOAM, if non-existent. Then download the SOWFA source code from Barts Github repository and put it in the \$WM\_PROJECT\_USER\_DIR folder.

```
module load openfoam/2.4.0
cd ~
mkdir OpenFOAM
git clone https://github.com/Bartdoekemeijer/SOWFA $WM_PROJECT_USER_DIR
```

- b. The SOWFA source files have now been downloaded to your \$WM\_PROJECT\_USER\_DIR folder, which is something like /home/bmdoekemeijer/OpenFOAM/bmdoekemeijer-2.4.0/. Now, we compile the standard SOWFA library (without ZeroMQ), as follows:

```
cd $WM_PROJECT_USER_DIR
./Allwmake
```

- c. Compiling the SOWFA libraries will take a couple minutes. You might get some warnings about deprecated versions and unused variables, but that's fine. The code will also tell you whether you are compiling SOWFA with or without ZeroMQ, which is the library needed for the MATLAB-coupled simulations. Right now, we are compiling SOWFA without ZeroMQ.



```
/opt/ud/OpenFOAM-2.4.0/src/OSspecific/POSIX/InInclude
URCE -o Make/linux64GccDPOpt/SmagorinskyABL.o
SOURCE=KosovicOneEqNBA/KosovicOneEqNBA.C ; g++ -m64 -Dlinux64 -DWM_DP -Wall -Wextra -W
r -Wold-style-cast -Wnon-virtual-dtor -O3 -DNoRepository -ftemplate-depth-100 -I/opt/u
penFOAM-2.4.0/src/turbulenceModels -I/opt/ud/OpenFOAM-2.4.0/OpenFOAM-2.4.0/src/turbulen
ltas/InInclude -I/opt/ud/OpenFOAM-2.4.0/OpenFOAM-2.4.0/src/turbulenceModels/LES/LESfilt
pt/ud/OpenFOAM-2.4.0/OpenFOAM-2.4.0/src/turbulenceModels/incompressible/LES/InInclude -
2.4.0/OpenFOAM-2.4.0/src/transportModels -I/opt/ud/OpenFOAM-2.4.0/OpenFOAM-2.4.0/src/tr
mpressible/InInclude/ -I/opt/ud/OpenFOAM-2.4.0/OpenFOAM-2.4.0/src/finiteVolume/InInclud
AM-2.4.0/OpenFOAM-2.4.0/src/meshTools/InInclude -IInInclude -I. -I/opt/ud/OpenFOAM-2.4.
rc/OpenFOAM/InInclude -I/opt/ud/OpenFOAM-2.4.0/OpenFOAM-2.4.0/src/OSspecific/POSIX/InIn
SOURCE -o Make/linux64GccDPOpt/KosovicOneEqNBA.o
'/home/bmdoekemeijer/OpenFOAM/bmdoekemeijer-2.4.0/platforms/linux64GccDPOpt/lib/libuser
oers.so' is up to date.
COMPILEZEROMQ is not set. Not compiling ZeroMQ SSC in ADM.
wmakeInInclude: linking include files to ./InInclude
ln: failed to create symbolic link './none.H': File exists
ln: failed to create symbolic link './PID.H': File exists
ln: failed to create symbolic link './none.H': File exists
ln: failed to create symbolic link './fiveRegion.H': File exists
ln: failed to create symbolic link './none.H': File exists
```

- d. If all goes well, SOWFA should now be installed. To double-check, try ./Allwmake one more time. No errors should come up:

```

[bmdoekemeijer@hpc06:bmdoekemeijer-2.4.0]$ ./Allwmake
Building with OpenFOAM-2.4.0
'/home/bmdoekemeijer/OpenFOAM/bmdoekemeijer-2.4.0/platforms/linux64GccDPOpt/lib/libuser
incompressibleLESModels.so' is up to date.
COMPILEZEROMQ is not set. Not compiling ZeroMQ SSC in ADM.
'/home/bmdoekemeijer/OpenFOAM/bmdoekemeijer-2.4.0/platforms/linux64GccDPOpt/lib/libuser
TurbineModelsStandard.so' is up to date.
OPENFAST DIR is not set. Not compiling OpenFAST interface.
'/home/bmdoekemeijer/OpenFOAM/bmdoekemeijer-2.4.0/platforms/linux64GccDPOpt/lib/libuser
sampling.so' is up to date.
'/home/bmdoekemeijer/OpenFOAM/bmdoekemeijer-2.4.0/platforms/linux64GccDPOpt/lib/libuser
fileFormats.so' is up to date.
fileFormats.so' is up to date.
'/home/bmdoekemeijer/OpenFOAM/bmdoekemeijer-2.4.0/platforms/linux64GccDPOpt/lib/libuser
finiteVolume.so' is up to date.
'/home/bmdoekemeijer/OpenFOAM/bmdoekemeijer-2.4.0/platforms/linux64GccDPOpt/lib/libuser
utilityFunctionObjects.so' is up to date.
make: '/home/bmdoekemeijer/OpenFOAM/bmdoekemeijer-2.4.0/platforms/linux64GccDPOpt/bin/s
etFieldsABL' is up to date.
make: '/home/bmdoekemeijer/OpenFOAM/bmdoekemeijer-2.4.0/platforms/linux64GccDPOpt/bin/A
BLSolver' is up to date.
make: '/home/bmdoekemeijer/OpenFOAM/bmdoekemeijer-2.4.0/platforms/linux64GccDPOpt/bin/A
BLTerrainSolver' is up to date.
make: '/home/bmdoekemeijer/OpenFOAM/bmdoekemeijer-2.4.0/platforms/linux64GccDPOpt/bin/w
indPlantSolver.ALM' is up to date.
make: '/home/bmdoekemeijer/OpenFOAM/bmdoekemeijer-2.4.0/platforms/linux64GccDPOpt/bin/w
indPlantSolver.ALMAAdvanced' is up to date.
OPENFAST DIR is not set. Not compiling OpenFAST solver.
make: '/home/bmdoekemeijer/OpenFOAM/bmdoekemeijer-2.4.0/platforms/linux64GccDPOpt/bin/w
indPlantSolver.ADM' is up to date.
make: '/home/bmdoekemeijer/OpenFOAM/bmdoekemeijer-2.4.0/platforms/linux64GccDPOpt/bin/p
isoFoamTurbine.ALM' is up to date.
make: '/home/bmdoekemeijer/OpenFOAM/bmdoekemeijer-2.4.0/platforms/linux64GccDPOpt/bin/p
isoFoamTurbine.ALMAAdvanced' is up to date.
OPENFAST DIR is not set. Not compiling OpenFAST solver.
make: '/home/bmdoekemeijer/OpenFOAM/bmdoekemeijer-2.4.0/platforms/linux64GccDPOpt/bin/p
isoFoamTurbine.ADM' is up to date.
make: '/home/bmdoekemeijer/OpenFOAM/bmdoekemeijer-2.4.0/platforms/linux64GccDPOpt/bin/t
urbineTestHarness.ALM' is up to date.
make: '/home/bmdoekemeijer/OpenFOAM/bmdoekemeijer-2.4.0/platforms/linux64GccDPOpt/bin/t
urbineTestHarness.ALMAAdvanced' is up to date.
[bmdoekemeijer@hpc06:bmdoekemeijer-2.4.0]$

```

e. SOWFA is now installed successfully.

## 2. Installation of ZeroMQ (20 min)

If we want to install SOWFA with the ZeroMQ libraries, we first need to compile the ZeroMQ libraries by itself.

- a. To do this, we need a newer version of CMake than currently available on the cluster. Let's first create a new directory in OpenFOAM, download CMake 3.11.2, and untar it:

```
mkdir -p $HOME/OpenFOAM/zeroMQ
cd $HOME/OpenFOAM/zeroMQ
wget https://cmake.org/files/v3.11/cmake-3.11.2.tar.gz
tar xf cmake-3.11.2.tar.gz && cd cmake-3.11.2
```

- b. Now, we prepare installation using *bootstrap*, and then we compile the source code of CMake in parallel using the *make* command. Finally, we use *make install*. This all will take approximately 10 minutes in total.

```
./bootstrap --prefix=$HOME/OpenFOAM/zeroMQ/cmake-3.11.2/install
make -j 8 && make install
```

we now installed CMake 3.11.2 in a local user folder. Let's add it to our path so that we can call it using the *cmake* command.

```
export PATH=$HOME/OpenFOAM/zeroMQ/cmake-3.11.2/install/bin:$PATH
cmake --version
```

- c. With a newer version of CMake, we can now download and install ZeroMQ. Let's copy the latest version of the ZeroMQ library to our OpenFOAM folder

```
git clone https://github.com/zeromq/libzmq $HOME/OpenFOAM/zeroMQ/libzmq
```

We now make a separate folder for the CMake build information within. Inside this folder we use the *cmake* command to prepare the source files for compilation. Finally, we can compile the source code using the *make* command. This will take a couple of minutes.

```
mkdir $HOME/OpenFOAM/zeroMQ/libzmq/cmake-build
cd $HOME/OpenFOAM/zeroMQ/libzmq/cmake-build
cmake -DCMAKE_INSTALL_PREFIX:PATH=$HOME/OpenFOAM/zeroMQ/libzmq/install ..
make -j 8
make test
make install
```

- d. We now have installed the ZeroMQ library. Be sure to add it to your *PATH* and *LD\_LIBRARY\_PATH* environment variables whenever you are (re)compiling SOWFA with ZeroMQ, and when you are running simulations with ZeroMQ enabled. You can do this as follows.

```
export ZEROMQ_INCLUDE=$HOME/OpenFOAM/zeroMQ/libzmq/install/include
export ZEROMQ_LIB=$HOME/OpenFOAM/zeroMQ/libzmq/install/lib64

export LD_LIBRARY_PATH=$HOME/OpenFOAM/zeroMQ/libzmq/install/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH=$HOME/OpenFOAM/zeroMQ/libzmq/install/lib64:$LD_LIBRARY_PATH
```

### 3. Installation of SOWFA with ZeroMQ (5 min)

The installation of SOWFA with ZeroMQ is very similar to the installation of SOWFA without ZeroMQ, but for some initial environment variables.

- a. Load the OpenFOAM 2.4.0 from the cluster and download SOWFA to \$WM\_PROJECT\_USER\_DIR folder.

```
module load openfoam/2.4.0
cd ~
mkdir OpenFOAM
cd OpenFOAM
git clone https://github.com/Bartdoekemeijer/SOWFA $WM_PROJECT_USER_DIR
```

- b. If you have a pre-existing installation, then first clean the old files, as

```
cd $WM_PROJECT_USER_DIR
./Allwclean
```

- c. Now we load the ZeroMQ library paths and enable ZeroMQ compilation in SOWFA. We can enable the compilation of ZeroMQ in SOWFA using the variable *COMPILEZEROMQ*.

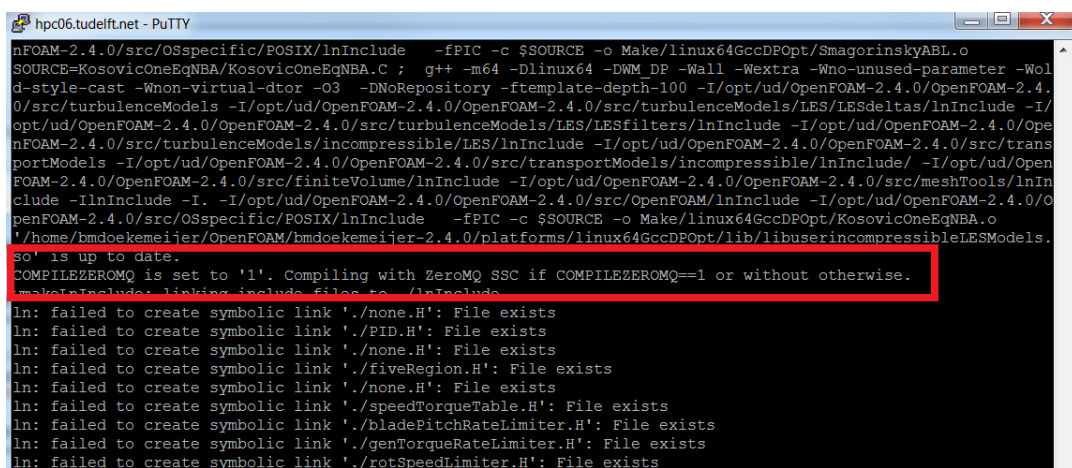
```
export ZEROMQ_INCLUDE=$HOME/OpenFOAM/zeroMQ/libzmq/install/include
export ZEROMQ_LIB=$HOME/OpenFOAM/zeroMQ/libzmq/install/lib64

export LD_LIBRARY_PATH=$HOME/OpenFOAM/zeroMQ/libzmq/install/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH=$HOME/OpenFOAM/zeroMQ/libzmq/install/lib64:$LD_LIBRARY_PATH
export COMPILEZEROMQ=1
```

- d. Now, compile the standard SOWFA library with ZeroMQ using the default command:

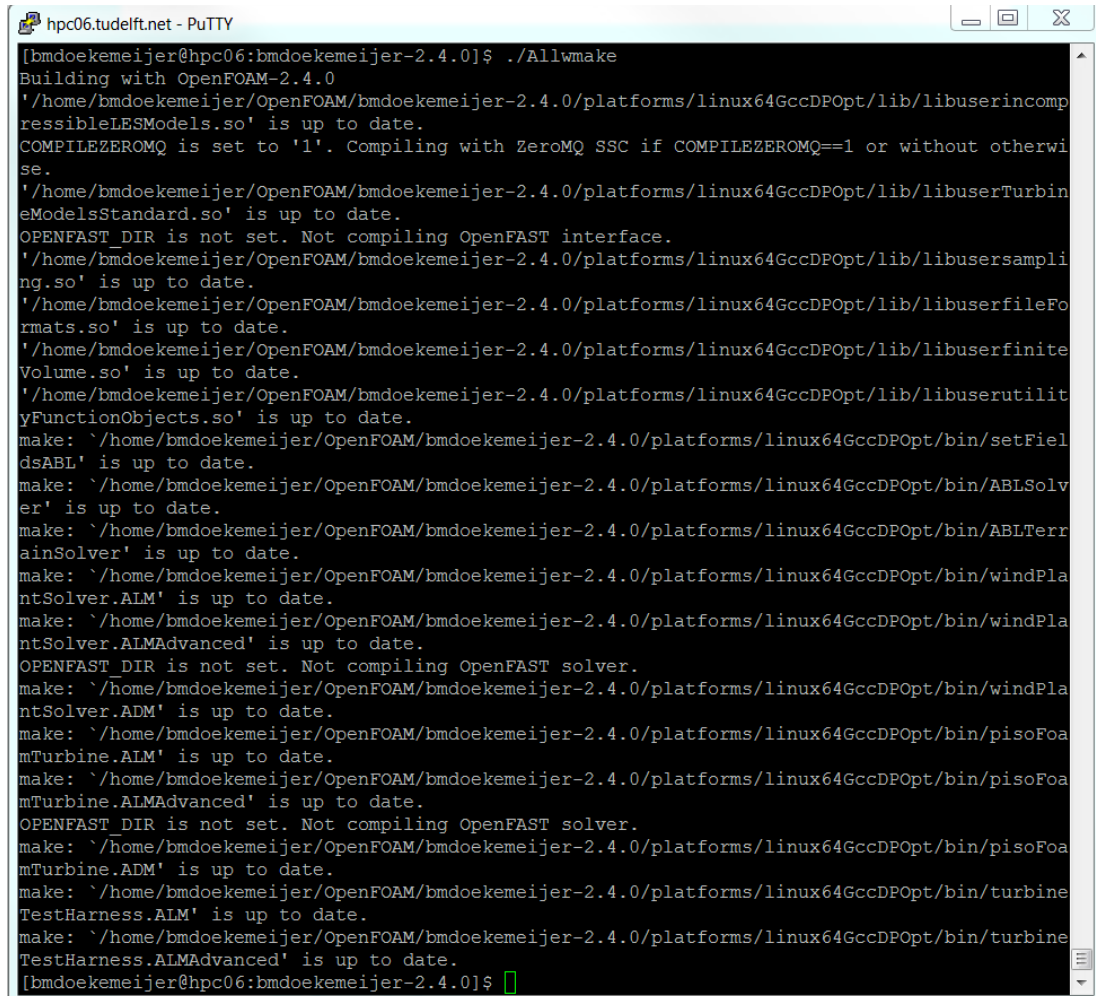
```
cd $WM_PROJECT_USER_DIR
./Allwmake
```

- e. Compiling the SOWFA libraries will take a couple minutes. You might get some warnings about deprecated versions and unused variables, but that's fine. The code will also tell you whether you are compiling SOWFA with or without ZeroMQ, which is the library needed for the MATLAB-coupled simulations. Right now, we are compiling SOWFA with ZeroMQ.



```
hpc06.tudelft.net - PuTTY
nFOAM-2.4.0/src/OSspecific/POSIX/lnInclude -fPIC -c $SOURCE -o Make/linux64GccDPopt/SmagorinskyABL.o
SOURCE=KosovicOneEqNBA/KosovicOneEqNBA.C ; g++ -m64 -Dlinux64 -DWM_DP -Wall -Wextra -Wno-unused-parameter -Wold-style-cast -Wnon-virtual-dtor -O3 -DNoRepository -ftemplate-depth-100 -I/opt/ud/OpenFOAM-2.4.0/OpenFOAM-2.4.0/src/turbulenceModels -I/opt/ud/OpenFOAM-2.4.0/OpenFOAM-2.4.0/src/turbulenceModels/LES/LESdeltas/lnInclude -I/opt/ud/OpenFOAM-2.4.0/OpenFOAM-2.4.0/src/turbulenceModels/LES/LESfilters/lnInclude -I/opt/ud/OpenFOAM-2.4.0/OpenFOAM-2.4.0/src/turbulenceModels/incompressible/LES/lnInclude -I/opt/ud/OpenFOAM-2.4.0/OpenFOAM-2.4.0/src/transportModels -I/opt/ud/OpenFOAM-2.4.0/OpenFOAM-2.4.0/src/transportModels/incompressible/lnInclude/ -I/opt/ud/OpenFOAM-2.4.0/OpenFOAM-2.4.0/src/finiteVolume/lnInclude -I/opt/ud/OpenFOAM-2.4.0/OpenFOAM-2.4.0/src/meshTools/lnInclude -IlnInclude -I. -I/opt/ud/OpenFOAM-2.4.0/OpenFOAM-2.4.0/src/OpenFOAM/lnInclude -I/opt/ud/OpenFOAM-2.4.0/OpenFOAM-2.4.0/src/OSspecific/POSIX/lnInclude -fPIC -c $SOURCE -o Make/linux64GccDPopt/KosovicOneEqNBA.o
'/home/bmdoekemeijer/OpenFOAM/bmdoekemeijer-2.4.0/platforms/linux64GccDPopt/lib/libuserincompressibleLESModels.so' is up to date.
COMPILEZEROMQ is set to '1'. Compiling with ZeroMQ SSC if COMPILEZEROMQ==1 or without otherwise.
makeInInclude: linking include files to /lnInclude
ln: failed to create symbolic link './none.H': File exists
ln: failed to create symbolic link './PID.H': File exists
ln: failed to create symbolic link './none.H': File exists
ln: failed to create symbolic link './fiveRegion.H': File exists
ln: failed to create symbolic link './none.H': File exists
ln: failed to create symbolic link './speedTorqueTable.H': File exists
ln: failed to create symbolic link './bladePitchRateLimiter.H': File exists
ln: failed to create symbolic link './genTorqueRateLimiter.H': File exists
ln: failed to create symbolic link './rotSpeedLimiter.H': File exists
```

- f. If all goes well, SOWFA should now be installed. To double-check, try `./Allwmake` one more time. No errors should come up:



```
hpc06.tudelft.net - PuTTY
[bmdoekemeijer@hpc06:bmdoekemeijer-2.4.0]$ ./Allwmake
Building with OpenFOAM-2.4.0
'/home/bmdoekemeijer/OpenFOAM/bmdoekemeijer-2.4.0/platforms/linux64GccDPOpt/lib/libuserincomp
ressibleLESModels.so' is up to date.
COMPILEZEROMQ is set to '1'. Compiling with ZeroMQ SSC if COMPILEZEROMQ==1 or without otherwi
se.
'/home/bmdoekemeijer/OpenFOAM/bmdoekemeijer-2.4.0/platforms/linux64GccDPOpt/lib/libuserTurbin
eModelsStandard.so' is up to date.
OPENFAST_DIR is not set. Not compiling OpenFAST interface.
'/home/bmdoekemeijer/OpenFOAM/bmdoekemeijer-2.4.0/platforms/linux64GccDPOpt/lib/libusersampli
ng.so' is up to date.
'/home/bmdoekemeijer/OpenFOAM/bmdoekemeijer-2.4.0/platforms/linux64GccDPOpt/lib/libuserfileFo
rmats.so' is up to date.
'/home/bmdoekemeijer/OpenFOAM/bmdoekemeijer-2.4.0/platforms/linux64GccDPOpt/lib/libuserfinite
Volume.so' is up to date.
'/home/bmdoekemeijer/OpenFOAM/bmdoekemeijer-2.4.0/platforms/linux64GccDPOpt/lib/libuserutilit
yFunctionObjects.so' is up to date.
make: '/home/bmdoekemeijer/OpenFOAM/bmdoekemeijer-2.4.0/platforms/linux64GccDPOpt/bin/setFiel
dsABL' is up to date.
make: '/home/bmdoekemeijer/OpenFOAM/bmdoekemeijer-2.4.0/platforms/linux64GccDPOpt/bin/ABLSolv
er' is up to date.
make: '/home/bmdoekemeijer/OpenFOAM/bmdoekemeijer-2.4.0/platforms/linux64GccDPOpt/bin/ABLTerr
ainSolver' is up to date.
make: '/home/bmdoekemeijer/OpenFOAM/bmdoekemeijer-2.4.0/platforms/linux64GccDPOpt/bin/windPla
ntSolver.ALM' is up to date.
make: '/home/bmdoekemeijer/OpenFOAM/bmdoekemeijer-2.4.0/platforms/linux64GccDPOpt/bin/windPla
ntSolver.ALMAdvanced' is up to date.
OPENFAST_DIR is not set. Not compiling OpenFAST solver.
make: '/home/bmdoekemeijer/OpenFOAM/bmdoekemeijer-2.4.0/platforms/linux64GccDPOpt/bin/windPla
ntSolver.ADM' is up to date.
make: '/home/bmdoekemeijer/OpenFOAM/bmdoekemeijer-2.4.0/platforms/linux64GccDPOpt/bin/pisoFoa
mTurbine.ALM' is up to date.
make: '/home/bmdoekemeijer/OpenFOAM/bmdoekemeijer-2.4.0/platforms/linux64GccDPOpt/bin/pisoFoa
mTurbine.ALMAdvanced' is up to date.
OPENFAST_DIR is not set. Not compiling OpenFAST solver.
make: '/home/bmdoekemeijer/OpenFOAM/bmdoekemeijer-2.4.0/platforms/linux64GccDPOpt/bin/pisoFoa
mTurbine.ADM' is up to date.
make: '/home/bmdoekemeijer/OpenFOAM/bmdoekemeijer-2.4.0/platforms/linux64GccDPOpt/bin/turbine
TestHarness.ALM' is up to date.
make: '/home/bmdoekemeijer/OpenFOAM/bmdoekemeijer-2.4.0/platforms/linux64GccDPOpt/bin/turbine
TestHarness.ALMAdvanced' is up to date.
[bmdoekemeijer@hpc06:bmdoekemeijer-2.4.0]$
```

- g. SOWFA is now installed successfully. **Note that for any SOWFA run, you now need to define the ZeroMQ user libraries in `PATH` and `LD_LIBRARY_PATH`, even when you are not using ZeroMQ in that particular run.**



## PART II: WORKING WITH PISOFOAM IN SOWFA

The repository provides a wide variety of exampleCases, such that any user can build their own case by simply adjusting an existing case, rather than building cases from the ground up. This part explains all the example cases with PisoFoamTurbine. pisoFoamTurbine is the wind farm solver for simulations with uniform inflow conditions.

### Example.01.piso.NREL5MW.ADM

*Wind farm simulation (ADM) without precursor, without refinements, with NREL 5MW*

- Each SOWFA simulation has its own folder with all the necessary input files. There's no GUI. To get started, let's create a folder called simulationCases and (recursively) copy an example simulation from the exampleCases folder.

```
cd $WM_PROJECT_USER_DIR
cd simulationCases
cp -r ../exampleCases/example.01.piso.NREL5MW.ADM .
```

- Each SOWFA run is created in a separate folder, which has all the necessary inputs files within it. In example.ADM, we see the following files:
  - **0.original** – This folder contains the initial system states (flow field, pressure field). It also contains the boundary conditions of each wall. For example, open “U”, and you will see the internal field (uniform with value \$U0mag along the x-direction) and the boundary fields.
  - **constant** – This folder contains the turbine settings (turbineArrayProperties, turbineProperties, airfoilProperties), some of the solver settings (turbulenceProperties, transportProperties, LESProperties), and the numerical mesh properties (polyMesh). You can change the turbine properties, number of turbines, their positions, initial control settings, etc. in this folder.
  - **system** – This folder contains the controlDict.1 file, which sets the initial time, end time and timestep of the SOWFA solver. It also assigns the flow outputs (e.g., 2D flow VTK slices, point measurements)
  - **system/sampling/sliceDataInstaneous** – Here we define what flow measurements we extract, and at what rate.
  - **runscript.\*** – These files are to preprocess the SOWFA case, run the actual simulation, and remove all the postprocessed/output data to reset the case to a clean folder.
  - **setUp** – a file containing the settings of the mesh, number of cells, wind speed, wind direction, etc. This file basically has a central point to collect the important settings of the simulation. Sometimes, these settings are directly inserted into the corresponding SOWFA case files (e.g., check out constant/polyMesh/blockMeshDict).
- Let's change the domain size and the number of cells, to speed up the simulation.

- Edit *setUp* and adjust xMax, yMax, zMax, and nx, ny, nz.
- Edit constant/turbineArrayProperties, set *bladeEpsilon* as twice the cell size (rule of thumb).

- Preprocess the case.

```
./runscript.removeAll
module load openfoam/2.4.0
./runscript.preprocess
```

Basically, the preprocessing will do the following here

1. It will copy controlDict.1 to controlDict
2. It will create a clean initial state folder ("0"), copied from 0.original
3. It will use blockMesh to generate a numerical mesh
4. It will use renumberMesh to numerically condition the mesh for better performance
5. It will use decomposePar to distribute the simulation to multiple processors for parallel jobs
6. It will use checkMesh to validate whether the mesh is correct

- Run pisoFoamTurbine.ADM locally to check if it works

```
pisoFoamTurbine.ADM
```

- If it works, now submit a parallelized 8-core job to the cluster. First edit runscript.solve.1 to fit to your needs. Then, submit it using

```
qsub runscript.solve.1
```

- Let it run for a while. Once the case has run for a couple minutes, cancel it.

```
qstat -l dcsc
qdel <jobid.hpc06>
```

- View the output data

- Turbine outputs are written to the turbineOutput folder. You can open these files with MATLAB, e.g., using the SOWFA\_tools functions.
- Flow fields are written to postprocessing/sliceDataInstantaneous. You can read these with MATLAB or with ParaView.

## example.02.piso.NREL5MW.ALMAAdvanced

*Wind farm simulation (ALMAAdvanced); no precursor, no refinements; NREL 5MW*

1. We can repeat the previous simulation case, but now with the ALMAAdvanced turbine model.
2. This example case only differs from **example.01** in the turbine model. This was done by replacing ADM files with the turbine ALMAAdvanced files for the NREL 5MW turbine. The exact folders and files that have been changed are:

```
constant/turbineArrayProperties
constant/turbineProperties
constant/airfoilProperties
```

Note that there are now many more options in the `turbineArrayProperties` file, such as the modelling of the nacelle/tower, how the blades are discretized, forces are projected onto the flow, various *epsilons* to smear the force to avoid numerical instability.

3. Similarly, we now also have to change the solver in **runscript.solve**, since our rotor model is now ALMAdvanced rather than ADM. Our solver becomes

```
solver=pisoFoamTurbine.ALMAAdvanced
```

4. Make sure to also edit the job's name is **runscript.solve**, so that you can keep the jobs apart.

### example.03.piso.DTU10MW.ALMAAdvanced

*Wind farm simulation (ALMAAdvanced); no precursor, no refinements; DTU 10MW turbine*

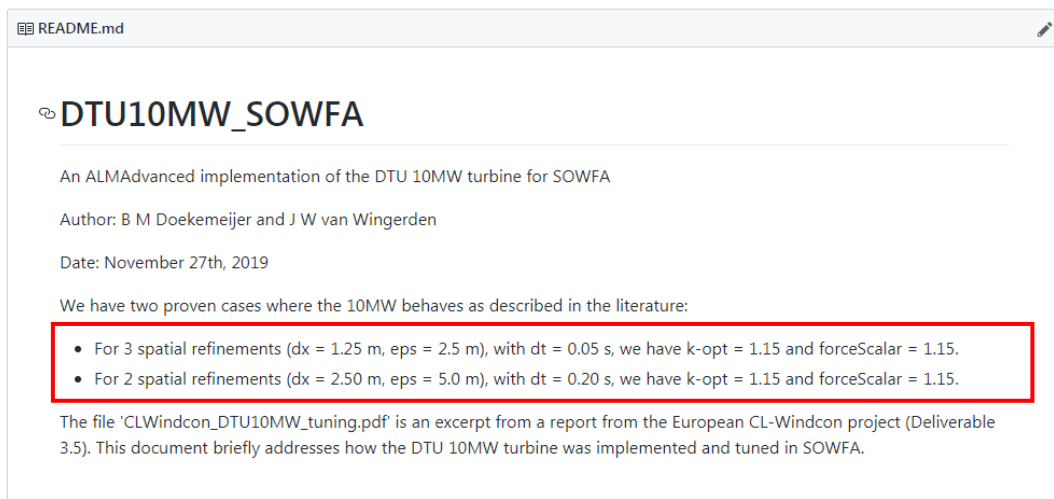
This case is basically identical to example 2, but now we replace the turbine files with those from the [DTU 10MW Github page](#). The files to be modified are the exact same files are in the previous example:

- `constant/turbineArrayProperties`
- `constant/turbineProperties`
- `constant/airfoilProperties`
- `runscript.solve.1`

### example.04.piso.DTU10MW.ALMAAdvanced.refinements

*Wind farm simulation (ALMAAdvanced); no precursor, with refinements; DTU 10MW turbine*

1. Generally, we introduce local mesh refinements in order to better simulate the flow behaviour in the wake and close to the turbine blades. For the DTU 10MW, the [Github manual](#) reads:



Thus, our initial base mesh is 10 m x 10 m x 10 m. To get to  $dx = 2.50$  m, we need to do two refinements.

2. The refinements are defined in `system/topoSetDict.local.*`. We can plot the refinements easily using the [plotRefinements](#) tool from the SOWFA\_tools repository. To enforce the mesh refinements, we modify `runscript.preprocess` to include two refinements.
3. We now have to modify the *epsilon* smearing factor in `turbineArrayProperties` and the sampling time in `system/controlDict.1` to match what is given in the Github page.
4. The job can now be executed as normal: preprocess the job using `runscript.preprocess.piso` and then submit the job using `qsub runscript.solve.piso`. Note that the number of cells may have increased significantly due to refinements, so you may want to parallelize the job to more cores (e.g., 20 or 40 cores).

## example.05.piso.NREL5MW.ADM.timeTableSSC

**Wind farm simulation (ADM) without precursor, without refinements, with NREL 5MW, with timetable SSC**

1. The recently implemented *timeTable Sowfa Super Controller (SSC)* add-on can be used to specify time-varying control signals such as blade pitch angles or yaw angles. A number of changes have to be made to enable the `timeTable_SSC` functionality in SOWFA.
2. Firstly, `constant/turbineArrayProperties` must be modified to include the SOWFA SuperController (SSC) bit. This is a separate set of inputs that can be added to `turbineArrayProperties` to set-up the SSC controller. If this bit is missing, then it will just assume there is no SSC present. It has the following parameters:

```

sscEnabled      true;
nInputsToSSC    3;
nOutputsFromSSC 2;
sscControllerType "timeTableSSC";
sscMeasurementsFunction "default";

```

*sscEnabled* specifies whether the SSC is used or not, *nInputsToSSC* specifies the number of inputs per turbine that SOWFA sends to the SSC code, *nOutputsFromSSC* specifies the number of control signals send to each turbine from the SSC, *sscControllerType* specifies which SSC to use (here: timeTableSSC), and finally *sscMeasurementsFunction* refers to the measurement function, which converts SOWFA code to useful measurements to be sent to the SSC code.

3. In the main folder *SC\_INPUT.txt* specifies the input signal timeseries for all turbines.
4. Now that the SSC is enabled and can read the *SC\_INPUT.txt* file, there is one more thing left to adjust. The local turbine controllers must now be replaced by turbine controllers that will take the outputs from the SSC (e.g., the assigned yaw angle) and enforce it on a turbine level. This can be done in *constant/turbineArrayProperties/NREL5MW*. Specifically, if we are changing both the blade pitch angles and the yaw angles, then we change:

```

BladePitchControllerType "PIDSC";
NacYawControllerType     "yawSC";

```

Note that having the standard turbine controllers with the SSC enabled basically means that the turbines ignore the SSC, but SOWFA runs perfectly fine. If you have the SC turbine controllers but without SSC enabled, you will get weird errors running SOWFA.

5. You preprocess and run SOWFA in the exact same way as in **example.02**. Actually, you do not even need to re-do the preprocessing. The turbine properties are independent from the mesh, and the preprocess script only does mesh-related things. After adjusting all turbine files, simply change the runscript.solve job name and submit it to the cluster.

## example.06.piso.NREL5MW.ALMAAdvanced.timeTableSSC

***Wind farm simulation (ALMAAdvanced) without precursor, without refinements, with NREL 5MW, with timetable SSC***

1. This is the same simulation as **example.05**, but now the ALMAAdvanced rotor model is used instead of the ADM rotor model. Thus, this example case is easily made by using example.02 and extending it with SSC capabilities, or taking example.05 and replacing the ADM turbine files with the ALMAAdvanced turbine files.

## PART III: MAKING PRECURSOR SIMULATIONS IN SOWFA

The repository provides a wide variety of exampleCases, such that any user can build their own case by simply adjusting an existing case, rather than building cases from the ground up. This part explains all the example cases with ABLSolver. This is the simulation solver that generates a turbulent inflow for wind farm simulations later on (part IV).

### example.07.ABL.precursor.neutral1.5x1x1.lowTI

*Turbulent precursor simulation in a 1.5km x 1.5km x 1km domain with a low turbulence intensity*

1. This is an example precursor simulation of a small domain, for demonstration purposes. The main focus of this simulation setup now lies with the file *setUp*. There are no turbine properties. The folders *0.original*, *constant* and *system* still hold the same meaning. Note that there are now two *runscript.solve* files: *runscript.solve.1* and *runscript.solve.2*. Similarly, there are two *controlDict*s: *controlDict.1* and *controlDict.2* (in *system*). This is for the following reason:

A precursor simulation is first run for approx. 20,000 seconds, so that the flow field can converge and turbulent structures can develop inside the flow. This is done by *runscript.solve.1* and the corresponding *controlDict.1*. Within *runscript.solve.1*, there are two commands: one that initializes the flow field, and one that starts the actual simulation.

```
setFieldsABL # Initializer  
ABLSolver # Solver
```

2. After that, a second simulation is run that continues from 20,000 s to 22,000 s where the inflow plane (e.g., the west plane) is sampled, later to be used as the inflow for a wind farm simulation. This plane sampling is enabled by including the “boundaryDataPre” sampling function in the *controlDict.2* file. This gives 2,000 seconds of sampling data. *Runscript.solve.2* only contains the solver, not the initializer.

```
ABLSolver # Solver
```

3. After the simulations have been completed, the data should be post-processed in a special manner using the SOWFA/tools files. You can find the exact instructions in the precursor cheatsheet.

### example.08.ABL.precursor.neutral3x3x1.lowTI

*Turbulent precursor simulation in a 3km x 3km x 1km domain with a low turbulence intensity*

1. This is a realistic, turbulent precursor simulation which yields a turbulent flowfield with an approximate TI of 5-6%, and a mean wind speed of 8 m/s or 9 m/s, as specified in *setUp*.

## PART IV: WORKING WITH WINDPLANTSOLVER IN SOWFA

The repository provides a wide variety of exampleCases, such that any user can build their own case by simply adjusting an existing case, rather than building cases from the ground up. This part explains all the example cases with windPlantSolver (wps). This solver does the wind farm simulations which are subjected to a turbulent inflow field (one of the precursors from part III).

### example.09.wps.NREL5MW.ADM

*Wind farm simulation (ADM) with precursor, without refinements, with NREL 5MW*

1. The general idea of the folders *constant* and *system* are the same as for the pisoFoamTurbine example cases. However, now, note that the folder *0.original* is missing. This is because of simulation will not start from a uniform flow field, but instead start from the turbulent flow field at time 20,000 s of one of the precursor cases.

Looking into *runscript.preprocess.wps*, we see that the following things happen:

1. It will copy controlDict.1 to controlDict
2. It will copy folder *20000* from the precursor to the case directory
3. It will copy the numerical mesh from the precursor to the case directory
4. It will create a symbolic link with the boundaryData (inflow data)
5. It will change the cyclic boundary conditions (as was necessary for the precursor simulation) to a predefined inflow conditions as is necessary for the wind farm simulation.
6. It will refine the mesh, if necessary.
7. It will use renumberMesh to numerically condition the mesh for better performance
8. It will use decomposePar to distribute the simulation to multiple processors for parallel jobs
9. It will use checkMesh to validate whether the mesh is correct

Thus fundamentally, there are a few important differences with the preprocess.piso. Namely, the initial flow conditions and the numerical mesh are copied from the precursor simulation, instead of generated by scratch. This gives less flexibility in terms of the domain size and discretization. And most importantly, the wind farm is simulated under turbulent inflow.

2. In terms of turbine definitions, nothing has changed between pisoFoamTurbine and windPlantSolver. The turbine properties are defined in the *constant* folder.
3. The wind farm solver is now *windPlantSolver.ADM*. The runscript.solve.wps has been adjusted accordingly. You can submit a job in the same manner using the *qsub* command.

## **example.10.wps.DTU10MW.ALMAAdvanced.refinements**

*Wind farm simulation (ALMAAdvanced) with precursor, with refinements, with DTU 10MW*

1. This simulation differs from **example.09** by replacing the NREL5MW ADM model with the DTU10MW ALMAAdvanced model. Do not forget to update the timestep in controlDict.1 and the solver in runscript.solve.wps. Also, if a higher mesh is desired, be sure to add refinements (as is done here), and update the *epsilon* smearing factors in turbineArrayProperties accordingly (recall:  $\epsilon = 2 * dx$ ).

## **example.11.wps.DTU10MW.ALMAAdvanced.refinements**

*Wind farm simulation (ALMAAdvanced) with precursor, with refinements, with DTU 10MW, with timeTableSSC*

1. This simulation differs from **example.10** by the inclusion of the TimeTableSSC super controller. Thus, the control setpoints for turbines 1 and 2 are assigned in the file SC\_INPUT.txt. Note that the SSC bit has to be added to the turbineArrayProperties file to enable the timeTableSSC functionality, and furthermore the local turbine controllers must be updated to read the SSC-assigned setpoints.



## PART V: WORKING WITH ZEROMQ IN SOWFA

The repository provides a wide variety of exampleCases, such that any user can build their own case by simply adjusting an existing case, rather than building cases from the ground up. This part explains all the example case, both *wps* and *piso*, that use the ZeroMQ functionality in SOWFA to couple a MATLAB-based wind farm control algorithm. Note that this MATLAB-code can also easily be swapped with a Python-based wind farm controller, if the ZeroMQ library is set up properly.

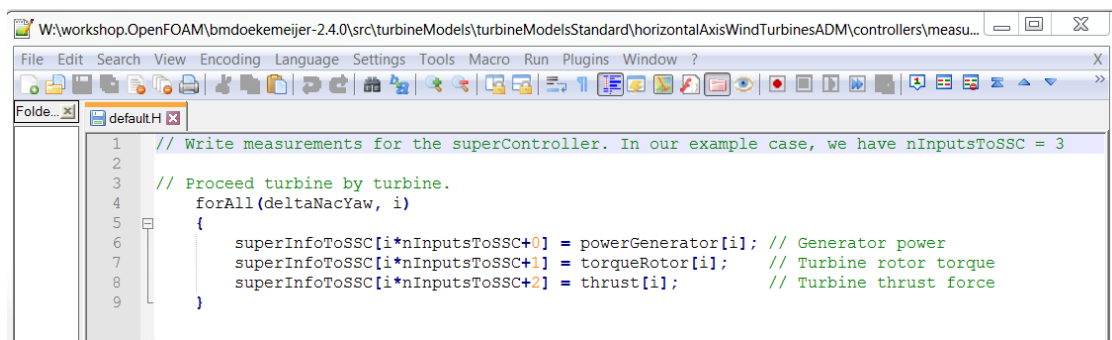
Note that for these simulations, you *must* have compiled SOWFA with the ZeroMQ functionality. You also need the JeroMQ java .jar file that is included in the repository. If you are not working on the DCSC cluster, you may have to compile JeroMQ for your own system.

### example.12.piso.NREL5MW.ADM.zmqSSC

**Wind farm simulation (ADM) without precursor, without refinements, with NREL 5MW, with ZeroMQ-based wind farm controller**

1. The recently implemented *ZeroMQ (zmq) Sowfa Super Controller (SSC)* add-on can be used to do closed-loop wind turbine and wind farm control without being limited to the C-programming language in which SOWFA is written. Communication between SOWFA and the external wind farm controller is done through ZeroMQ, and therefore any programming language that supports ZeroMQ can be used as a supercontroller (e.g., MATLAB, Python, C)
2. To enable ZeroMQ functionality, specify the *sscControllerType* in the *sscProperties* entries in *constant/turbineArrayProperties*. Furthermore, specify the variable *zmqAddress* to specify which network port will be used to do the communication. **Make sure that this is a unique port for each job, and is different from other running jobs.** Moreover, the *sscMeasurementsFunction* specifies which measurements are fed from SOWFA to the wind farm controller at each timestep. To know what the measurements are that belong to the *default* measurements function, check the file in

src\turbineModels\turbineModelsStandard\horizontalAxisWindTurbinesADM\controllers\measurementFunctions:



```
1 // Write measurements for the superController. In our example case, we have nInputsToSSC = 3
2
3 // Proceed turbine by turbine.
4 forAll(deltaNacYaw, i)
5 {
6     superInfoToSSC[i*nInputsToSSC+0] = powerGenerator[i]; // Generator power
7     superInfoToSSC[i*nInputsToSSC+1] = torqueRotor[i];    // Turbine rotor torque
8     superInfoToSSC[i*nInputsToSSC+2] = thrust[i];         // Turbine thrust force
9 }
```

You can change this measurement function or add your own one by adding a .H code in the

measurementFunctions folder, and then adding an *if* statement in the horizontalAxisWindTurbinesADM.C code, similarly to the code for the *default.H* measurement function.

Note that the measurement functions between the ADM supercontroller code and the ALMAdvanced supercontroller code differ slightly, so always check to make sure you are reading the right variables!

3. Make sure the local turbine controllers are also the SC-compatible ones, in constant/turbineProperties/NREL5MWRef

```
BladePitchControllerType "PIDSC";  
NacYawControllerType    "yawSC";
```

4. The wind farm controller code itself is a MATLAB-based wind farm controller here, that is run in parallel with the SOWFA simulations. The main file is located in **ssc/SSC.m**. The function is self-explanatory, and currently is a simple placeholder which just scheduled inputs according to the current time that is sent from SOWFA to MATLAB. You can do much more complicated things, e.g., with the measured power signals in a feedback setting. A more complicated example for the SSC code is given in the example case:

exampleCases/other/Doekemeijer\_RenewableEnergy2019/6turb\_runs/opt/ssc/SSC.m

In this MATLAB code, make sure the port specified in the top lines is the same as the port specified in turbineArrayProperties.

5. Since MATLAB and SOWFA communicate in real-time with each other, they must be run in parallel. Therefore, the runscript.solve.piso code is adapted to also launch the SSC.M wind farm controller at the same time as the SOWFA simulation. This is done with the code:

```
# Run the solver and the MATLAB controller in parallel  
(mpirun -np $cores $solver -parallel > log.$runNumber.$solver 2>&1) &  
(cd ssc; matlab -nodisplay -noFigureWindows -logfile 'SSC_out.log' -r SSC)
```

Note that furthermore the MATLAB module must be loaded on the cluster, and the ZeroMQ libraries must be added to the environment variables.

```
# Load the OpenFOAM module and MATLAB module on the cluster  
echo "Loading the OpenFOAM and MATLAB modules..."  
module load openfoam/2.4.0  
module load matlab  
  
# define the ZeroMQ paths  
export ZEROMQ_INCLUDE=$HOME/OpenFOAM/zeroMQ/libzmq/install/include  
export ZEROMQ_LIB=$HOME/OpenFOAM/zeroMQ/libzmq/install/lib64  
export LD_LIBRARY_PATH=$HOME/OpenFOAM/zeroMQ/libzmq/install/lib:$LD_LIBRARY_PATH  
export LD_LIBRARY_PATH=$HOME/OpenFOAM/zeroMQ/libzmq/install/lib64:$LD_LIBRARY_PATH
```

6. The job can now be preprocessed and launched like any other. Note that if you copied the preprocessed case from **example.05**, then you need not re-do the preprocessing, since the mesh has not changed. Only the turbine properties have changed, which do not affect the preprocessing actions.

### **example.13.piso.DTU10MW.ALMAAdvanced.refinements.zmqSSC**

*Wind farm simulation (ALMAAdvanced) without precursor, with refinements, with DTU 10MW, with ZeroMQ-based wind farm controller*

1. This is largely the same as **example.12**, but now the NREL5MW ADM turbine files have been replaced by the DTU 10MW ALMAAdvanced turbine files. Furthermore, refinements are added so that the turbine implementation becomes accurate (in accordance with the [Github prescriptions](#)). Note that the timestep has therefore also been changed (system/controlDict.1).
2. Since we now have refinements, perhaps you want to use more processors to solve the job in parallel. If you want to use the same discretization/mesh and only want to split the job into more processors, simply
  - a. Delete the existing *processor\** files
  - b. Use *decomposePar -cellDist -force* after updating the *setUp* file to decompose the problem to the new amount of cores.

### **example.14.wps.DTU10MW.ALMAAdvanced.refinements.zmqSSC**

*Wind farm simulation (ALMAAdvanced) with precursor, with refinements, with DTU 10MW, with ZeroMQ-based wind farm controller*

1. This is largely the same as **example.11**, but now the *timeTableSSC* functionality has been replaced by the MATLAB ZeroMQ functionality. Note that the *runscript.solve.wps* has been modified to also launch the MATLAB wind farm controller at the same time as the SOWFA simulation, including the definition of ZeroMQ in the environment variables and loading the MATLAB module on the cluster.
2. Again, the redoing the preprocessing is not necessary compared to **example.11**, if only the turbine properties or the control strategies have been changed. This saves you some time.