The Actuator Line Code - User Guide

Markella Zormpa

August 21, 2022

1 Introduction

This is a user guide for the Actuator Line Method, hereafter ALM, aiming to help new users to get started with the code. Before everything, some useful OpenFOAM links are listed in section 2. Section 3 covers some of the basic theory, along with useful references for further reading. Section 4 elaborates on all the different files and dictionaries that need to be adjusted in order to run a wind turbine simulation employing the ALM. This includes the mesh files, the CFD settings, the ALM settings as well as some useful post processing functionalities. OpenFOAM references are included there and basic prior knowledge of OpenFOAM is strongly recommended. A test case is described step-by-step in section 6 with the aim to make the process of setting up your first case easy and quick. Finally, in section 5 the JuPyter notebook used for post-processing will be presented.

In what follows commands will be written in *italics* and files and folders in **bold**.

2 Before starting

Some resources that can be helpful when stuck:

- Jozsef Nagy. For OpenFOAM tutorials.
- Fluid Mechanics 101. Lectures on basic and advanced CFD concepts.
- OpenFOAM User and Programmer's guide available online or in the OpenFOAM installation folder.
- Notes on Computational Fluid Dynamics: General Principles.

3 Theory

The ALM is a momentum source turbine modelling method. It was first developed by Shen and Sorensen in 2002 [2] and has since been used to research wind turbine wakes. The code employs rotating lines that are discretized in collocation points. At each collocation point 2D airfoil theory is assumed and a lift and drag force are calculated. The force is then applied in the flow field as a source term. The algorithm can be seen in figure 1. The three basic steps of the ALM are:

- **Sampling**: Here the velocity $|U_{ref}|$ and angle of attack α are calculated. These are the necessary values needed to calculate the lift and drag from 2D airfoil data look-up tables. The aerofoil concept is a 2D one, and therefore difficulties arise when trying to define those quantities for a 3D blade.
- **Look-up**: This step is determined by the quality of the aerofoil polars. These can be attained most commonly using panel codes like X-Foil [1] (or the wind turbine specific Q-Blade) but also 2D CFD simulations, or experimental data. Under this step, the tip correction is applied, that accounts for the reduction of the blade force as it approaches the tip. The tip-correction employed is another area of ongoing research. Many authors have proposed different tip correction methods and some of them are implemented in the current ALM code.

- **Smearing**: To avoid numerical singularities, that would produce an unrealistic and discontinuous flow field that would most probably diverge, the force calculated in the previous step, are smeared over multiple computational cells. Usually, the function used for smearing is given by

$$\eta(\mathbf{r}) = \frac{1}{\epsilon^3 \pi^{3/2}} exp[-(r/\epsilon)^2] \tag{1}$$

The standard deviation ϵ of the smearing kernel is subject to debate, and various best practices have been proposed in literature. Other authors have used different shapes of kernels to better reproduce the shape of an aerofoil, like, for example an elliptical Gaussian kernel. In the current implementation of the ALM, only the spherical Gaussian kernel of equation 1 is available and the only parameter to be defined is ϵ.

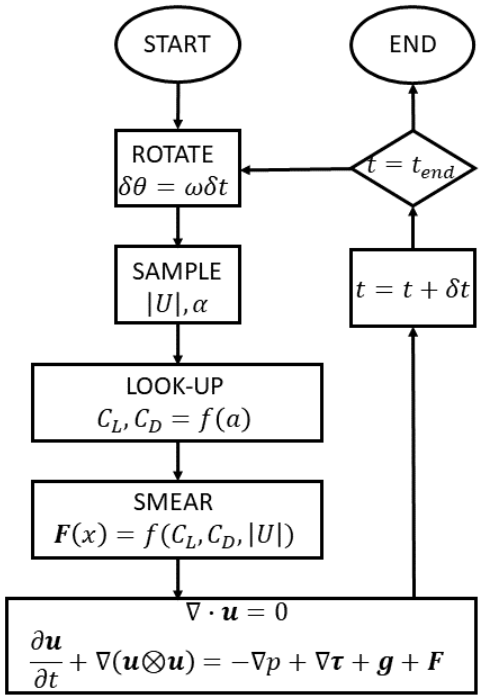

Figure 1: Flow chart of the ALM.

4 Case files

Each case is located in **caseDir** and consists of a number of directories and files, listed in table 1.

In **constant/.** the setup of the simulation is defined. That is, the mesh in **constant/polyMesh/.**, the turbulence model, the transport properties of the acting fluid, as well as the properties, geometry and layout of the turbines to be simulated (see **constant/turbineArrayProperties**, **constant/turbineProperties/.**), together with the aerofoil data that accompany the rotor geometry (**constant/airfoilProperties/.**).

In **system**, the user input parameters are defined. These can vary from the time-step, printing times and precision of output in **system/controlDict**, to solvers in **system/fvSolution**, discretization schemes in **system/fvSchemes**, user-defined source terms in **system/fvOptions**, decomposition properties in **system/decomposeParDict** as well as some essential meshing utilities in **system/blockMesh** and **system/snappyHexMesh**. Another important functionality defined in the **system** folder are the run-time function objects that are essential for data extraction and post processing, called in **system/controlDict** and defined **system/functionObjects/.**.

Finally, in **0.org/*** the initial conditions of the solution variables are defined. The user defined **setUp** file, summarizes some information of the simulation, to make the set-up of the simulation easier and more concentrated.

All the executables need to run under the SLURM protocol and therefore are run using *sbatch* through the **run*** fiels. The **run_preprocess** executable file, contains the mesh generation and decompsition. The **run_solve** contains the main solver. **run_clean** clears the file from an older simulation.

File	Section
caseDir/setUp	4.3
caseDir/run_solve	4.2
caseDir/run_clean	4.2
caseDir/run_preprocess	4.2
caseDir/0.org/{U,p,omega,epsilon,nut}	4.2
caseDir/constant/polyMesh/*	4.1
caseDir/constant/turbineProperties/{turbine1,...,turbineN}	4.4
caseDir/constant/airfoilProperties/{airfoil1,...,airfoilM}	4.4
caseDir/constant/turbineArrayProperties	4.4
caseDir/constant/turbulenceProperties	4.2
caseDir/constant/transportProperties	4.2
caseDir/system/controlDict	4.2
caseDir/system/fvSolution	4.2
caseDir/system/fvSchemes	4.2
caseDir/system/fvOptions	4.4
caseDir/system/decomposeParDict	4.1
caseDir/system/blockMesh	4.1.1
caseDir/system/snappyHexMesh	4.1.2
caseDir/system/functionObjects/*	4.3

Table 1: All the files involved in running the ALM code in OpenFOAM.

4.1 Mesh

The choice of mesh is always crucial in CFD. When using the ALM, two main mesh topologies are used depending on the application. The O-grid topology, figure 2, is aligned with the rotor and provides good refinement qualities in the areas of interest, like the tip and root vortices. The ALM was initially developed using this topology. The mesh can be generated using ANSYS-ICEM or any other meshing software. However, the aspect ratio of the cells at this region might cause convergence issues and might drive the time step to very small values that will slow down the simulation. The octree topology, is generated using the OpenFOAM utility *snappyHexMesh*. This type of mesh is easier to build and maintains constant mesh quality over the computational domain of interest. However, regions where small structures are expected are not refined further. Here, the process of generating an octree mesh using OpenFOAM is elaborated. The O-grid mesh is not further discussed.

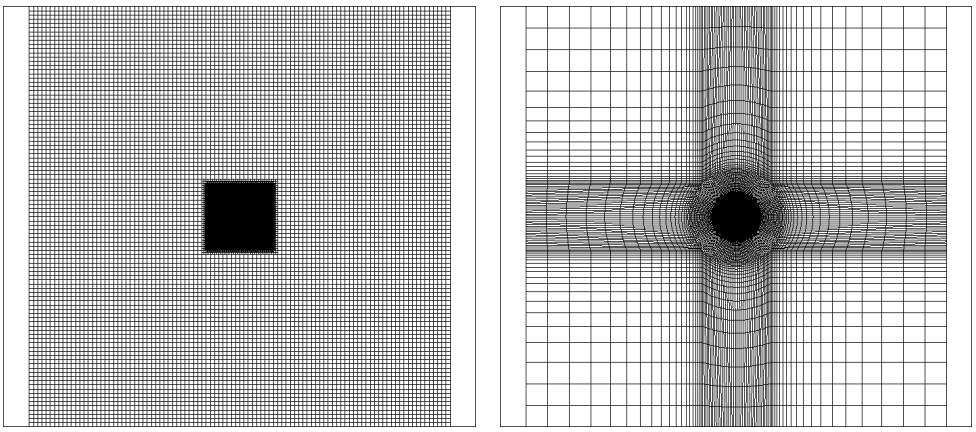

Figure 2: The octree topology(left) and the O-grid (right).

4.1.1 blockMesh

First, the block domain is generated using blockMesh. The usage of blockMesh is clearly explained here. The user needs to change the domain boundaries in **system/blockMeshDict** (ie x_{Min}, x_{Max}, y_{Min}, y_{Max}, z_{Min}, z_{Max}), as well as the number of cells in each direction (ie n_x,n_y,n_z) and run the utility *blockMesh*. These values are obtained by taking into consideration the shape and size of the cell. For example, if hexahedral cells are going to be used for the whole domain, the user has to make sure that

$$\frac{x_{Max} - x_{Min}}{n_x} = \frac{y_{Max} - y_{Min}}{n_y} = \frac{z_{Max} - z_{Min}}{n_z} \tag{2}$$

The final domain will be similar to figure 3.

4.1.2 snappyHexMesh

snappyHexMesh is a useful utility that in it's simplest implementation can refine a mesh created using *blockMesh*. Details about *snappyHexMesh* can be found here. For ALM applications, if no solid boundary is defined (for example, a tower that would require to cut a hole in the blockMesh) the user can deactivate two of the three features of *snappyHexMesh*, namely:

castellatedMesh	true;
snap	false;
addLayers	false;

Deactivating unecessary utilities from **snappyHexMeshDict**.

The user needs to change only the options in the castellated mesh dictionary, namely:

- The boundaries of the refined domains in the dictionary **geometry{}**.

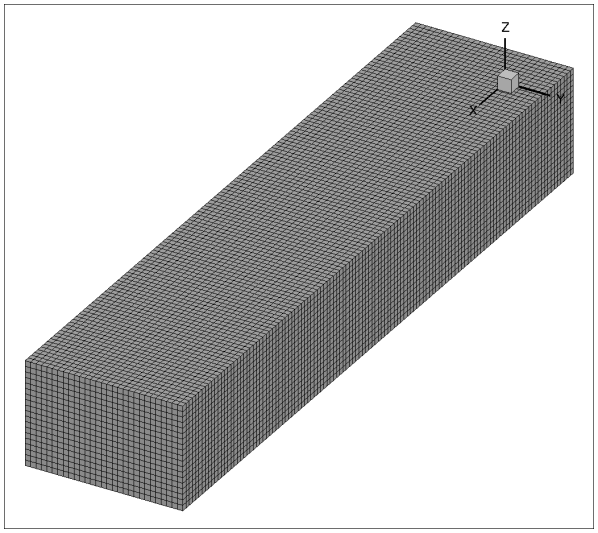

Figure 3: Domain created using the blockMesh utility.

- The levels of refinement within those boundaries in the dictionary **castellatedMeshControls.refinementRegions{}**.
- The variable **locationInMesh** defined in the **castellatedMeshControls{}** dictionary has to be a point vector (x, y, z) that is within the mesh boundaries.
- In the case of a ground boundary that needs refinement, the user will need to define the ground surface in the **geometry{}** dictionary and refine the surface in the **refinementRegions{}** dictionary.

A cross section of the resulting mesh is seen in figure 4.

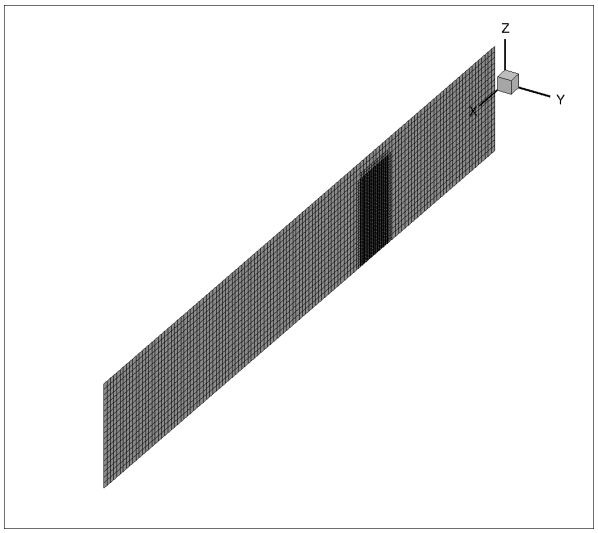

Figure 4: Domain created using the snappyHexMesh utility.

4.2 CFD Model

Here, the contents of **system/controlDict** are elaborated. **system/fvSchemes**, and **system/fvSolution** are not explained, but typical values and schemes are used for the discretization and solution of the governing equations and the user can find details about them in any CFD manual or test cases of similar flows. Unless there are issues with the simulation, the majority of the content of those two files is not changed by the user.

In **system/controlDict**, the end time of the simulation is defined in **endTime**, the time step **deltaT**, the writing properties, that define every how much time or iterations the solution will be backed up in a file. **purgeWrite** defines the maximum number of solutions that exist in the storage. This might be important in very large simulations, where memory restrictions might allow storing only 2 solutions, at all times. The **writeFormat** is usually binary, and **writeCompression** off. The **timePersicion** defines how many decimals are stored in the written folders. **runTimeModifiable true;** means that even when the code is running, changes in any of the input files will be read by the solver. Finally, the **functions{}** dictionary contains all the run-time processing functions.

constant/turbulenceProperties contains the details about the turbulence model and can be copied and (if needed) altered by an already existing tutorial case. **constant/transportProperties** contains the viscosity of the fluid, and must be changed when tidal turbine simulations are performed, from air to water.

4.3 Function objects

An essential part of a CFD simulation is run-time data collection. In OpenFOAM this is achieved using function objects. Here, a few basic function objects for the ALM are presented, but they can be enhanced or removed depending on the simulation.

For both RANS and LES simulations, it is recommended to use the **solverInfo** function object that prints out the resiudals of the transport equations a data file, and the **probes** function object that samples the field in a specific location at every time-step. Those two function objects provide an easy way to check the convergence of the simulation by looking at the time-developement of those printed values.

To facilitate the definition of the probes in **probes**, some geometrical properties of the rotor can be defined in the file **setUp**, and are in turn read by **probes** and automatically create probes at some wake regions. Of course, users can create their own probes.

For LES simulations it is essential to average the instantaneous fields in time, so the **fieldAverage** function object that prints both the mean velocity and the Reynolds stresses is needed. It is also good to print the subgrid scale turbulent kinetic energy using **turbulenceFields**.

4.4 Actuator Line Model

In **constant/turbineArrayProperties**, the farm properties are defined. Often only one rotor is simulated, but it is possible to define multiple turbines with different geometries. This file is analyzed line-by-line, as it is the heart of the actuator line method and this is the only documentation available for it.

turbineType	$\langle word \rangle$;	The name of the turbine in **constant/turbineProperties/***.
baseLocation	$\langle vector \rangle$;	Coordinates of the base see fig 5.
fluidDensity	$\langle scalar \rangle$;	Density in kg/m^3.
numBladePoints	$\langle int \rangle$;	Number of collocation points.
pointDistType	$\langle word \rangle$;	Distribution of collocation points: “uniform”,“cosine”, “halfCosine”.
rotationDir	$\langle word \rangle$;	Direction of rotation: “ccw” or “cw”.
Azimuth	$\langle scalar \rangle$;	Azimuth angle at $t = 0$, in degrees.
RotSpeed	$\langle scalar \rangle$;	Rotor speed in RPM.
NacYaw	$\langle scalar \rangle$;	Yaw angle in degrees.
pitchZero	$\langle scalar \rangle$;	Pitch angle in degrees.

Table 2: General settings in **constant/turbineArrayProperties**.

samplingMethod	⟨*word*⟩;	Available methods: "Schluntz3Points","Schluntz3PointsConstZ" "lineAverage".
numSamplePoints	⟨*int*⟩;	Number of collocation points.
sphereRadiusScalar	⟨*scalar* ≥ 1 ⟩;	Defines how much larger the diameter of the cell searching. sphere will be wrt the rotor diameter.
// Case 1: Constant sampling distance		
sampleDistParam	⟨*scalar* ≥ 0⟩;	The constant sampling distance in m.
sampleDistScalar	1e30;	Is not used when the sampling distance is constant.
// Case 2: Sampling distance as a function of the smear radius.		
sampleDistParam	-1;	If negative, then the sampling distance will be a linear function of the smearing radius.
sampleDistScalar	⟨*scalar*⟩;	At each collocation point sampleRadius = sampleDistScalar*smearRadius.

Table 3: Sampling settings in **constant/turbineArrayProperties**.

tipRootLossCorrType	⟨*word*⟩;	Available methods: "None", "Glauert", "Shen", "Wimshurst", "CaoWillden".
tipSpeedRatioShen	⟨*scalar*⟩;	TSR needed for "Wimshurst" and "Shen".
c1Faxi	⟨*scalar*⟩;	Tip correction factors for "Wimshurst".
c2Faxi	⟨*scalar*⟩;	Tip correction factors for "Wimshurst".
c3Faxi	⟨*scalar*⟩;	Tip correction factors for "Wimshurst".
c1Ftani	⟨*scalar*⟩;	Tip correction factors for "Wimshurst".
c2Ftani	⟨*scalar*⟩;	Tip correction factors for "Wimshurst".
c3Ftani	⟨*scalar*⟩;	Tip correction factors for "Wimshurst".
airfoilInterpolationMethod	⟨*word*⟩;	Available methods; "linearInterpolation" & "IDInterpolation".
rotorUindc	⟨*scalar*⟩;	Induced velocity at the rotor plane. ie $U_{inf}/3$.
rotorUturb	⟨*scalar*⟩;	
dynamicGustModel	"None";	No dynamic gust models have been implemented yet.

Table 4: Look-up settings in **constant/turbineArrayProperties**.

smearingModel	⟨*word*⟩;	Available models: "Global", "localChord", "liftCoefficient", "localChordCapped", "powerFunction", "cosineFunction".
epsilon	⟨*scalar*⟩;	Read by the methods: "Global", "localChordCapped". "powerFunction", "cosineFunction"
localChordFraction	⟨*scalar*⟩;	Read by all the methods, apart from "Global".
epsilonFunctionPow	⟨*scalar*⟩;	Read by "powerFunction" &"cosineFunction".

Table 5: Smearing settings in **constant/turbineArrayProperties**.

NumBl	⟨*int*⟩;	Number of blades, ie 3.
TipRad	⟨*scalar*⟩;	The radius of the rotor in m. See fig 5.
HubRad	⟨*scalar*⟩;	The radius of the hub in m. See fig 5.
TowerHt	⟨*scalar*⟩;	The height of the tower in m. See fig 5.
OverHang	⟨*scalar*⟩;	Overhang of the rotor in m. See fig 5.
Airfoils	List ⟨*word*⟩;	A list of the airfoils along the blade that corresponds to the filenames in **constant/airfoilProperties/***.
BladeData	List ⟨*List*⟨*scalar*⟩⟩;	Defining the blade stations using a list of 4-variable vectors of the form: (radius(m) Re TI airfoilID).
rotorDesign	List ⟨*vector*⟩;	Defining the rotor design using a list of vectors of the form: (radius(m) chord(m) twist(deg) airfoilID).
The rest of the options will not be analyzed but usually remain constant for simple rotor geometries.		

Table 6: The variables in **constant/turbineProperties/*** that need to be changed.

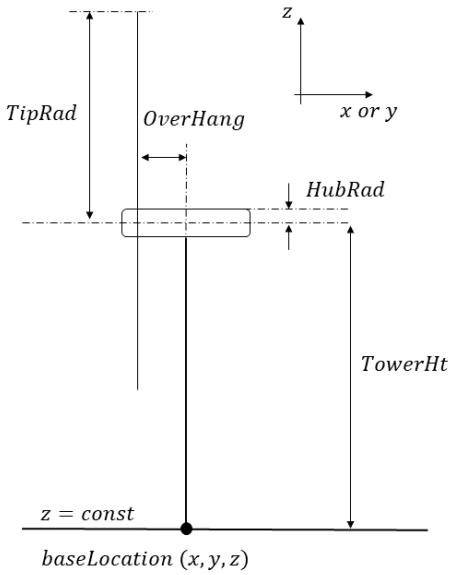

Figure 5: Some of the dimensions defined. **Always make sure that the z direction is aligned with the tower.**

5 Post processing

The raw data coming from running the ALM code, need to be averaged, processed and analyzed in order to obtain meaningful conclusions from the simulation. The basic post processing needed after every ALM simulation has been developed in Python, and presented in an interactive notebook. This section describes how to use python notebooks on ARCUS, and the basic input and outputs of the notebook provided.

5.1 Using Jupyter Notebooks

To use JuPyter Notebooks

- In your **desktop computer** use the command *jupyter notebook –NotebookApp.iopub_data_rate_limit=1.0e10*.
- In **ARC**, follow this link.

5.2 Inputs of the post-processor

The post processor reads the output of function objects in **postProcessing/***, as well as the output of the actuator line code in **turbineOutput/***. The user can specify which outputs they are interested in by switching with 0/1 the variables of table 7.

convergenceProbes = 1	# Time series of instantaneous velocities.
standardDeviationProbes = 1	# Time series of instantaneous mean velocity and Reynolds stresses.
solverInfo = 1	# Plot of residuals.
turbineOutput = 1	# Calculates C_P, C_T, spanwise forces & AoA distributions
	# and the time series of P & T.

Table 7: Switches of the post processor.

The files that the post processor is capable of processing are the following:

- **turbineOutput/*/{power,thrustRotor,alphaC,*Force,radiusC}** output from the ALM.
- **postProcessing/convergenceProbes** a result of using **FOprobe**.
- **postProcessing/standardDeviationProbes** a result of using (**FOfieldAverage** & **FOprobe**).
- **postProcessing/solverInfo** a result of using (**FOsolverInfo**) function object.

Additionally, some more information need to be specified

pwd = $\langle word \rangle$	# Working directory
nbLastPeriods = $\langle int \rangle$	# how many periods to average over to calculate Pmean and Tmean for C_P,C_T
probe_ids = $[\langle int \rangle, \langle int \rangle, ...]$	# Probe IDs to plot
period = $\langle scalar \rangle$	# rotating period in s
rho = $\langle scalar \rangle$	# density in kg/m^3
U = $\langle scalar \rangle$	# inflow velocity (used for cp,ct) in m/s
D = $\langle scalar \rangle$	# diameter in m

Table 8: The input needed by the post processor.

5.3 Outputs of the post-processor

The available outputs from the post processor are

- Time series of (1) Residuals (2) Power and thrust (3) Probed instantaneous velocities (4) Probed mean velocities (5) Probed Reynolds stresses. Only when those are sufficiently converged, it is meaningful to begin to interpret the results.

- Power and thrust coefficient table.
- Spanwise blade force and angle of attack distribution.
- Power spectral density of specific probes.

An example of the output is located in the repository. To create an .html file of the postprocessing the following command is used:
jupyter nbconvert notebookName.ipynb −−to html −−output outputName.html

6 Setting up a case step-by-step

In this section, the set-up of a case will be described in detail using a step-by-step guide.

- Copy an existing wind turbine case from the github repository.
- If the mesh is an octree the mesh boundaries need to be first defined. Alter the x_{min}, x_{max},y_{min}, y_{max}, z_{min}, z_{max} as well as n_x, n_y, n_z values in to the **system/blockMeshDict** desired values.
- Change any of the turbine properties in **constant/.** folder, as elaborated in section 4.4.
- In **system/controlDict**, choose a valid time-step, respecting the CFL condition as well as the actuator line time-step requirements. Also, the timeStartAveraging is user defined, as this will depend on the case-specific flow-through time. Here, you can also activate and de-activate function objects by commenting them out (with //).
- Usually no changes have to be made in **system/fvSolutions** and **system/fvSchemes** unless there are numerical considerations or convergence issues.
- Depending on the modelling or not of the support structure, the file **system/fvOptions** can be modified. The option **active on/off;** should work as a switch of the fvOptions function.
- Make the mesh using the command *sbatch run_preprocess*. Make sure to carefully adjust the time needed to run, name of job and if you want to be notified by e-mail you can add your e-mail in the script header.
- It is good practice to visualize the mesh in Tecplot before proceeding to the simulation.
- If all looks good adjust the number of processors, time, e-mail and name of job in **run_solve** and submit the run to the queue using *sbatch run_solve*. It is good practice to first have an idea of the total time needed to run the simulation. This can be calculated by running the code in the devel partition for 10 minutes and extrapolating the information from the **log.pimpleFoam** file. A bash script in the github repository named **howMuchLonger**, that takes as an input the log.pimpleFoam file, ie *howMuchLonger log.pimpleFoam** will give an estimate of how much time the simulation needs to reach endTime based on how much time the simulation has run for.
- ... wait for your simulation to finish ...
- Once finished, run the post processing to ensure everything is converged, and the power and thrust coefficients make sense.

Good luck!

References

[1] M. Drela. Xfoil: An analysis and design system for low reynolds number airfoils. In T. J. Mueller, editor, *Low Reynolds Number Aerodynamics*, pages 1–12, Berlin, Heidelberg, 1989. Springer Berlin Heidelberg.

[2] W. Z. Shen. Numerical modeling of wind turbine wakes, journal of fluids engineering. *Journal of Fluids Engineering*, 124, 2002.