AMR-Wind Automation Tutorial

Sydney Richardson

August 23, 2023

1 Introduction/TLDR

This base file is setup in the format so that you can declare all necessary variables to run AMR-Wind in one file, and by running the "base_file_transform" code you can generate the needed 3 input files (Spinup, Precursor, and Turbines) to run AMR-Wind. To use the transformation code, you MUST change the output_folder variable to the location of your base file alongside changing the precursor_file_name to match that of your base file name. Both these variables are at the very top of the transformation code. Then simply run the code (I used VScode). The code is divided into three sections: Spinup, Precursor, and Turbines with each section being defined by a semicolon. Please DO NOT delete these semicolon comments as they provide the premise for transforming the base file. If you wish to run the "Error Checker" code it will ensure the variables the user has defined match with the type declared in the commented brackets. Please DO NOT delete these brackets if you wish to error check your code. Please see the bottom of this file if there is anymore AMR-Wind variables you would like to add to the code. They have been formatted to simply be able to copy and paste the currently commented out variable into your desired section of the base file.

2 Variables

A common complaint about using AMR-Wind was the need to declare different variables with the same value, specifically the wind flow velocity. This automation code has been updated so that you do not need to declare velocity 3 times, instead only once and it will automatically generate the other variables in the output of the transformation code. There is also no need to redeclare any variables that are repeated between the Spinup, Precursor, and Turbines sections. Anything declared in the Spinup section will be declared in the Precursor and Turbines output. Likewise, anything declared in the Spinup section will also be declared in the Turbines output. If any variables need to be updated between section, one may simply redeclare the variable in the respective new section. For example, if the variable time.stop_time has been declared as 7201.0 in the Spinup section but needs to be 7801.0 in the Precursor section, you may declare the same variable in the Precursor section and the transformation code will automatically update these variables for both the Precursor and Turbines outputs. The output files will be named "spinup_data", "precursor_data", and "turbines_data".

3 File Format

The file format has been designed in a very specific way to work with the transformation code. Please DO NOT delete the section divides "; Spinup", "; Precursor", and "; Turbines" as this is what tells the transformation code which variables belong in which output file. Also, please DO NOT delete the commented brackets with the variable type (e.g. # [integer]) if you wish to use the error checker as the error checker uses these bracketed comments to double check you have declared the correct kind of value for all the variables. You can add as many variables as you please and if there is any new variables that have not been included in the code you can also add those! If you do not include a comment with bracket notation this will not be error checked but it will be correctly integrated into the output files. There is some additional variables at the very bottom of the base file example if you wish to browse more variables without having to consult the AMR-Wind user manual (https://exawind.github.io/amr-wind/user/user.html).

4 Error Checker

The error checker works similarly to the transformation code. You will need to update the file_path variable at the top of the script to be the path to that of your base file, then just run the code and it will tell you if it finds any errors. The errors this code searches for is variable type, if a variable matches from a list of options (for example a list of valid strings), and some physical checks such as gravity or density. Here are examples of how the error checker works:

```
# Error Check Example:

# The code you submit to error checker:
incflo.use_godunov    = 1                   # [boolean]
incflo.godunov_type   = wene_z              # [plm; ppm; ppm_nolim; weno_js; weno_z]
TKE.source_terms      = KsgsM84Src          # [string]
TKE.interpolation     = PiecewiseConstant   # [string]
incflo.gravity        = 0.  0. -9.81        # Gravity (3D), [list of 3 real numbers] (m/s2)
incflo.density        = 10.0                # Reference density, [real number] (kg/m3)
incflo.verbose        = 2.5                 # incflo_level, [integer]

# What the code should look like:
incflo.use_godunov    = 1                   # [boolean]
incflo.godunov_type   = weno_z              # [plm; ppm; ppm_nolim; weno_js; weno_z]
TKE.source_terms      = KsgsM84Src          # [string]
TKE.interpolation     = PiecewiseConstant   # [string]
incflo.gravity        = 0.  0. -9.81        # Gravity (3D), [list of 3 real numbers] (m/s2)
incflo.density        = 1.225               # Reference density, [real number] (kg/m3)
incflo.verbose        = 0                   # incflo_level, [integer]

# The is what the error checker will output:
Errors found:
Error: 'incflo.godunov_type' should be one of ['plm', 'ppm', 'ppm_nolim', 'weno_js',
    'weno_z'], but found 'wene_z'
Error: 'Are you sure you don't want incflo.density to be 1.225? It is currently 10.0'
Error: 'incflo.verbose' should be an integer, but found '2.5'
```

As you can see in the example, the error checker will check if a variable is declared as one of the given options in the case of incflo.godunov_type, valid physics in the case of incflo.density, and valid variable type as in the case of incflo.verbose.

5 References

- AMR-Wind User Manual: https://exawind.github.io/amr-wind/user/user.html
- Alex Rybchuk's amr-wind-tutorial: https://github.com/rybchuk/amr-wind-tutorial