DTU Library

Using Python for Wind Resource Assessment

Davis, Neil

Publication date:
2018

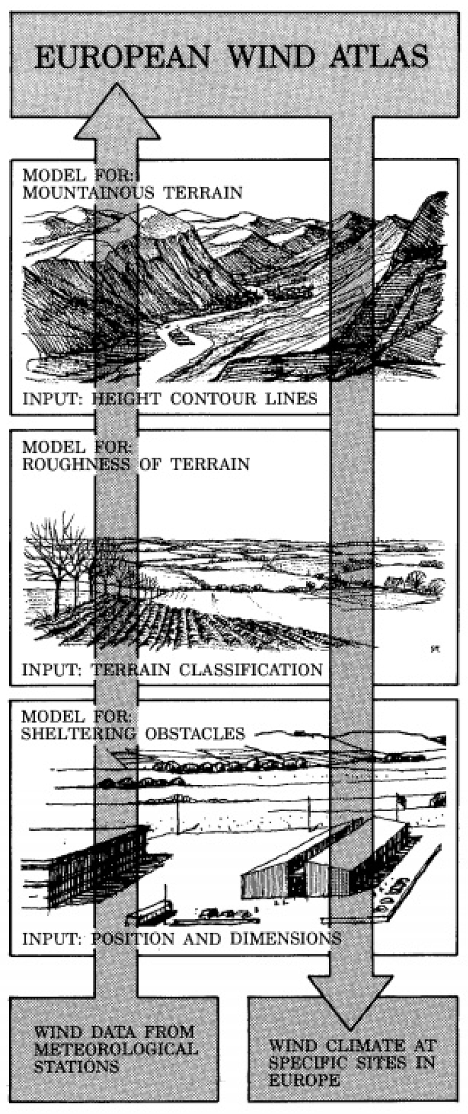
Document Version
Publisher's PDF, also known as Version of record

DTU Wind Energy
Department of Wind Energy

Using Python for Wind Resource Assessment

Neil Davis (DTU Wind Energy)

Wind Atlas Methodology



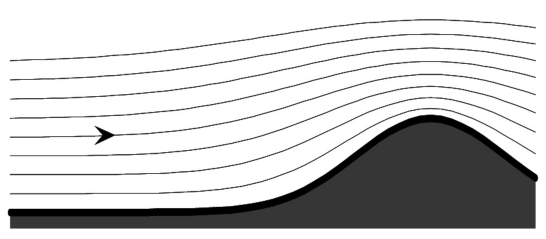


Figure 2. Wind speeds up as it goes over terrain due to the compression of the flow, and slows down on the lee of the hill.

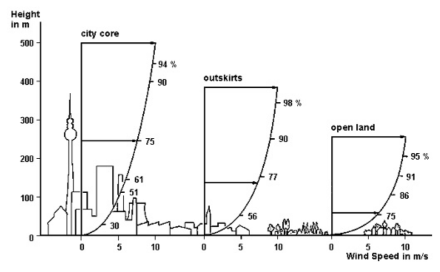


Figure 3. Wind near the surface is slower when it is going over rougher terrain, such as a city or forest.

Figure 1. Meteorological models are used to simulate the impact of surface features on the wind climate. (European Wind Atlas, 1989)

The WAsP Model simulates the surface affects on the wind speed distribution. This distrubution is found to be Weibull shaped, and surface affects can modify both the scale and shape parameters of the distribution. The WAsP model has been the leading software for wind resource assessment since its release in 1989.

WAsP Model

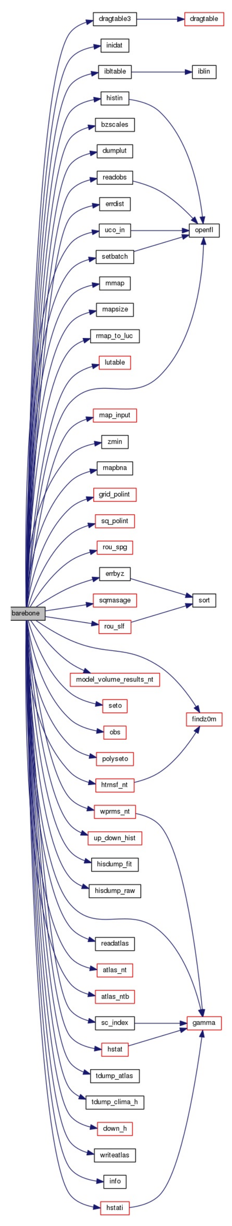


Figure 4. WAsP testbench, consisting of ~2500 lines of Fortran.

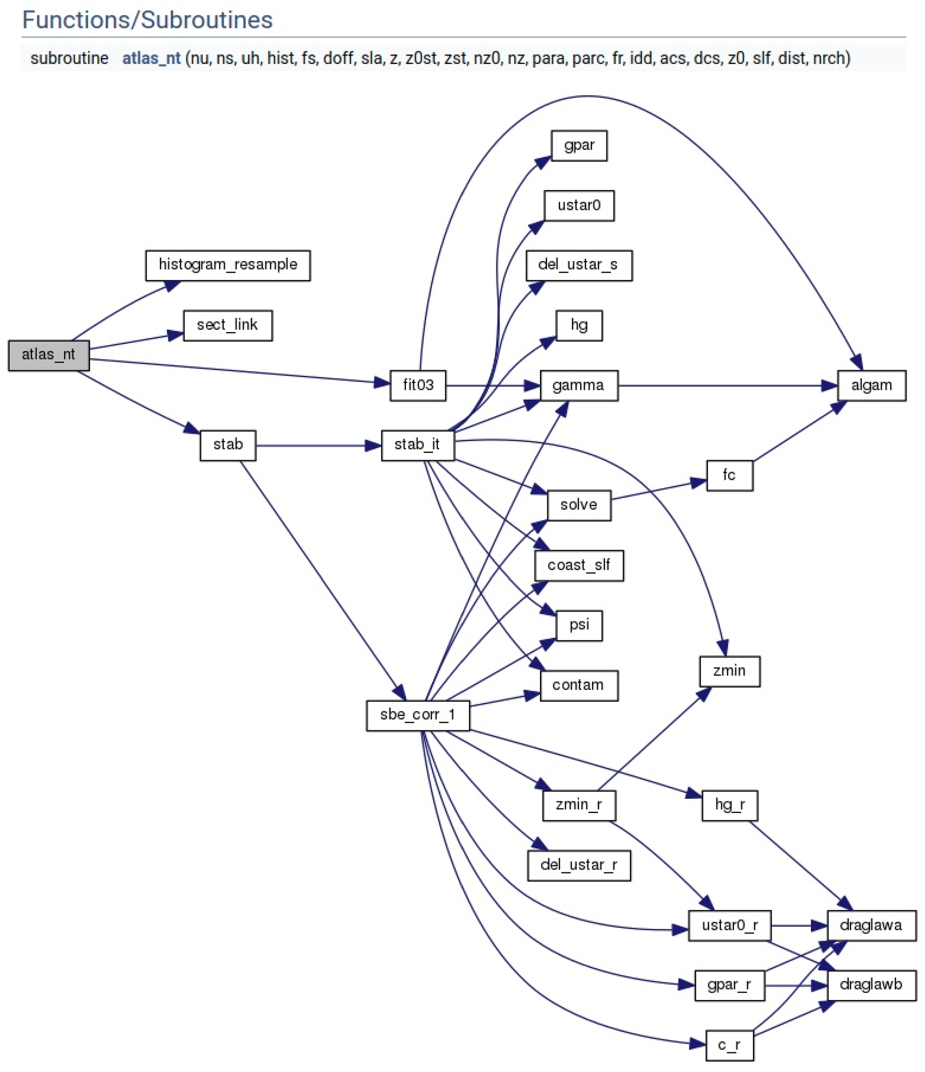


Figure 5. Atlas_NT, handles the conversion of mast data to a Generalized wind. Signified as the left arrow on Figure 1. Note that this call has 22 arguments, and a quite extensive call tree.

WAsP Model GUI

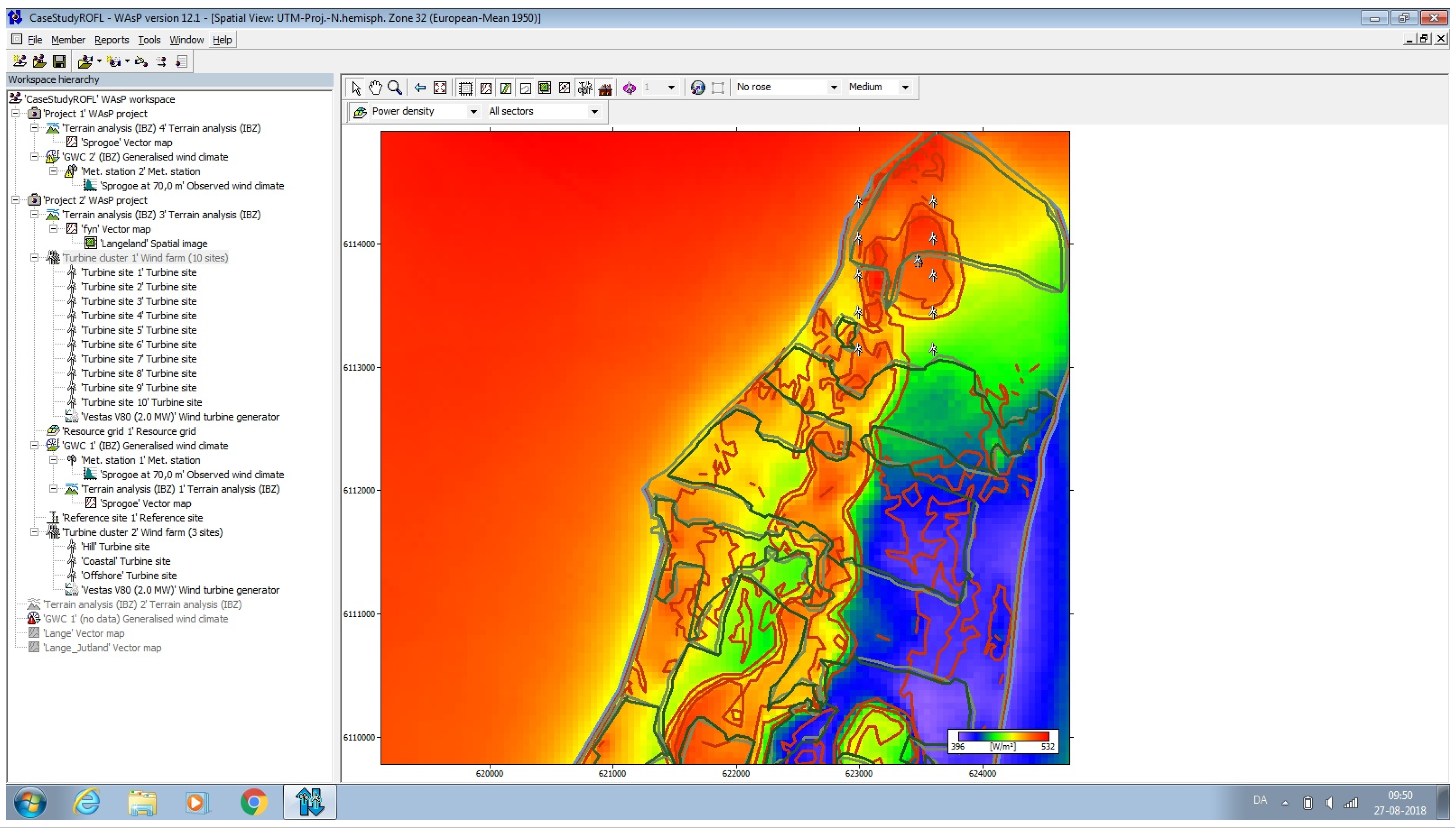


Figure 6. WAsP 12.1 GUI on Windows. This shows the WAsP interface with a Resource estimate (colors) for a Danish island.

Most users of WAsP do not call the Fortran itself, but interface with the model through an advanced Windows GUI program. This program provides GIS capabilities to add topographic information along with knobs to tweak the different model parameters, and set the required fields. In addition to the GUI provided by DTU, the WAsP model has been included in several other wind resource planning tools.

The WAsP GUI code has more than 1200 monthly users, who appreciate its simple design and ease of use. However, it was found that researchers at DTU were less likely to use the GUI tool for several reasons.

1. Researchers were increasingly using Linux and Mac envrionments.
2. Researchers wanted to execute large numbers of simulations using scripts.
3. Computers were now much faster and able to run on multiple cores.
4. Researchers used Python for analysis and wanted that for WAsP.

Due to these reasons, we investigated different ways of combining WAsP with Python to enable a cross platform solution for researchers

In addition, it was hoped that by moving to a Python framework, the model could be verified and validated more easily through the use of unit tests and comprehensive evalation against measurments on a routine basis.

Pungi Intitive

Create WAsP simulation

To replicate a WAsP simulation, you will need a .tab file at the measurement site, and a .map file that covers both the measurement site and the prediction site.

With those inputs you can create a WAsP prediction at the site by performing the following steps:

1. Read Tabfile using `BinWindClimate.from_file`
2. Read in Elevation and Roughness maps to `VectorMap.from_combo`
3. Create `TopographyMap`
4. Run `TopographyMap.to_point` at Tabfile location to create `TopographyPoint`
5. Run `generalize` to create `GenWindClimate`
6. Run `TopographyMap.to_point` at prediction location to create `TopographyPoint`
7. Run `downscale` to create prediction at prediction location

Below is an example of this process.

```
import pywasp_flow
conf = pywasp_flow.wasp.Config()

# User set data
tab_file = "data/tabfiles/risoe_76.tab"
map_file = "data/mapfiles/risoe.map"

gen_hgts = [10., 20., 40., 80., 100.]
gen_z0s = [0.000, 0.030, 0.100, 0.400, 1.0]

# Read in data
tab = pywasp_flow.io.BinWindClimate.from_file(tab_file)
elev_map = pywasp_flow.io.VectorMap.from_file(map_file, map_type=0, reproj={"from_epsg": 32632}
rou_map = pywasp_flow.io.VectorMap.from_file(map_file, map_type=1, reproj={"from_epsg": 32632})

# Generalize
topo_map = pywasp_flow.io.TopographyMap(elev_map, rou_map)
topo_tabpt = topo_map.to_point(694132, 6176364, 12, tab.height, pywasp_flow.wasp.params)
lib = pywasp_flow.wasp.generalize(tab, topo_tabpt, gen_z0s, gen_hgts, conf)

# Predict
topo_predpt = topo_map.to_point(694132, 6176364, 12, 80., pywasp_flow.wasp.params)
pywasp_flow.wasp.downscale(lib, topo_predpt, conf)
```

To create an interface to WAsP that would be useful to todays researchers, we turined to F2Py to wrap the Fortran code, and developed custom classes for the different steps in the wind resource assessment workflow. This enabled us to create a simple 7 step process for running a resource assessment project, provided that you had the correct input files.

The class based structure we developed allowed us to hide the long subroutine calls behind easier to use interfaces for the user. In the code to the right, the generalize function calls the atlas_nt function from above, but here only requires 5 arguments.

In addition to simplifying the interface, the Python development has allowed us to easily run WAsP on our HPC system, allowing for us to run simulations with millions of variations across many processes.

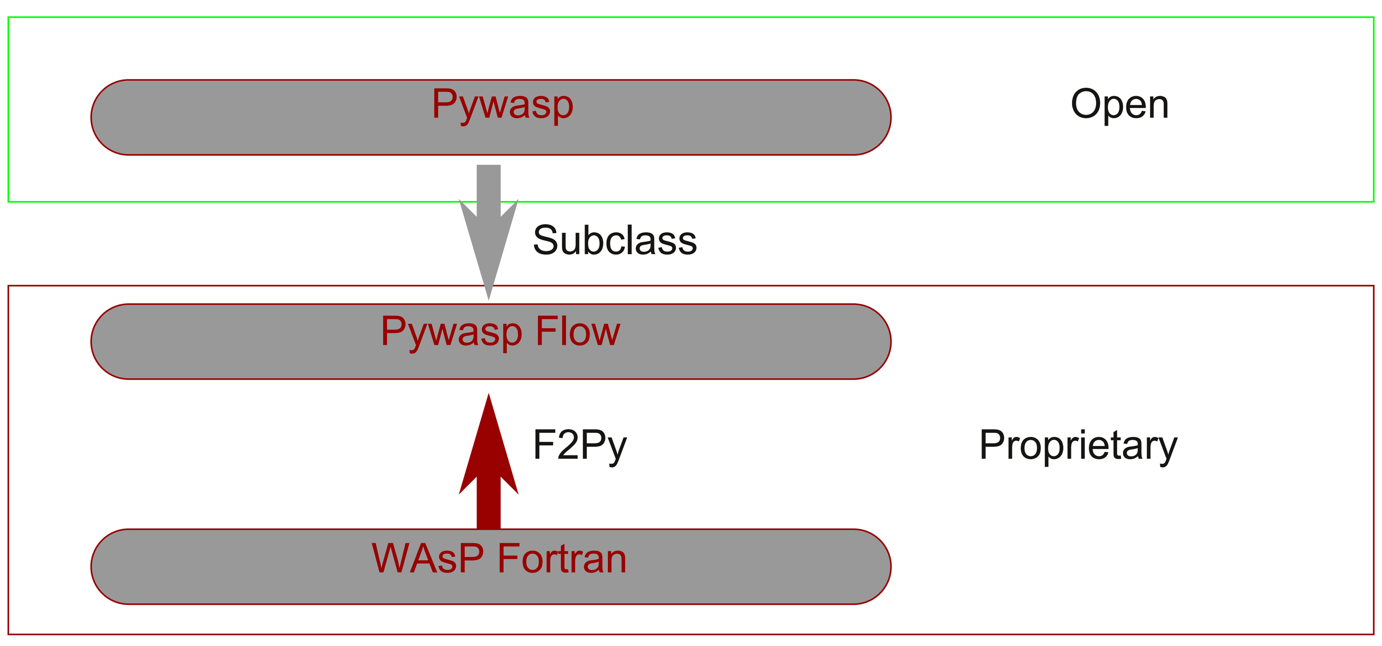
F2Py vs CTYPES for Research Code

F2Py	CTYPES
No derived type support	Can support any structure
Highly Automatic	Requires custom interface coding
Brittle to Python and Numpy versions	Brittle to changes in the Fotran API
Custom .SO files	Uses standard ISO C interface to Fortran
Automatic support for Numpy arrays	Unclear how to format Numpy arrays
Can be part of a setup.py build	Needs separate build for Fortran

In our case we found that F2py is an invaluable tool for wrapping Fortran. The killer features for us are the ease of use, in particular the ability to wrap routines without having to code any Python and the ability to build the Fortran as part of the setup.py process.

There are however some limiations to the use of F2py that we are experiencing, and would like to help rectify. First is the lack of derived type support, which is a feature that is becoming comon in our codebase. Second is the dependency on specific Python and Numpy versions. For Windows, there is a desire for a single .dll file that can be used for both the GUI and the Python implementation.

Package Structure



Lessons Learned

Spend upfront time on design to ensure that you understand your model code, and the specific pieces you wish to make available to Python.

Develop a classbased structure to enable easier changes to your Fortran API, and allow your backend structure to change

Don't be afraid to start with Fortran like data-structures, we have successfully moved to more Pythonic structures on the Python side hiden by the class variables.

Develop any new I/O formats first, and ensure they are in an open package. This is why we made the pywasp package, since we want others to use our formats even if they aren't paying for the use of the model.