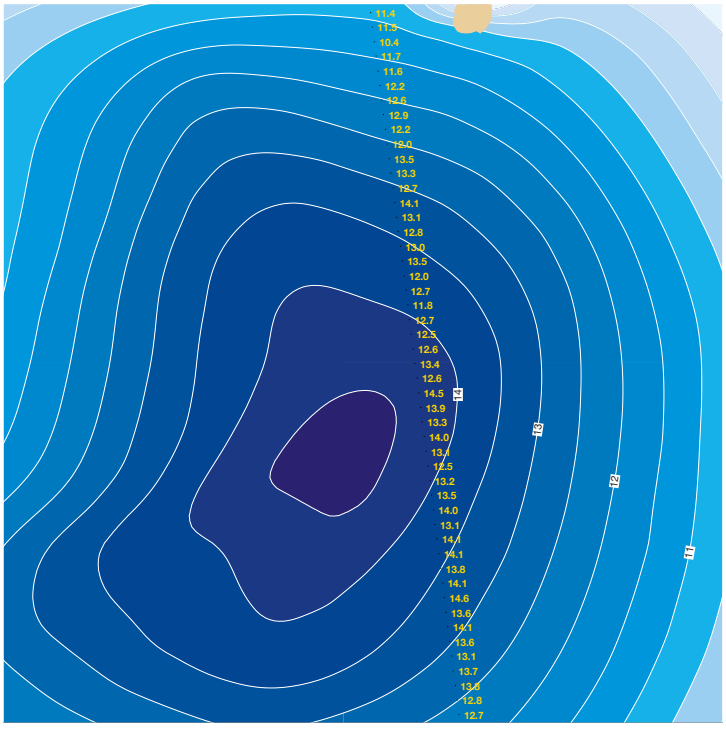
ECMWF Feature article

from Newsletter Number 162 – Winter 2019/20

COMPUTING

Metview's Python interface opens new possibilities



www.ecmwf.int/en/about/media-centre/media-resources

doi: 10.21957/hv3sp41ir5

This article appeared in the Computing section of ECMWF Newsletter No. 162 – Winter 2019/20, pp. 36–39

Metview's Python interface opens new possibilities

Iain Russell, Linus Magnusson, Martin Janousek, Sándor Kertész

Metview is ECMWF's interactive and batch processing software for accessing, manipulating and visualising meteorological data. Metview is used extensively both at ECMWF and in the Centre's Member and Co-operating States. A national meteorological service may for example use it to plot fields produced by the ECMWF model and a regional model, calculate some additional fields such as temperature advection, and plot vertical cross sections. The recent addition of a Python interface to Metview has expanded its range of uses and has made it accessible to more potential users. Already, applications in the areas of verification and diagnostics are using the new interface to great effect. The use of Python also opens up new ways of using Metview's functionality, such as through Jupyter notebooks for running data analyses or interacting with training material. For more information on Metview and how to install it, see Box A.

Metview

A

Since the release of version 1.0 in 1993, Metview has benefited from constant development work over the years to keep up with the demands of ever-increasing data volumes and more sophisticated ways of interacting with data. In addition to a graphical user interface (GUI), it introduced its own powerful scripting language called Macro. This high-level language allows users to develop concise scripts to process their data and to run those scripts in the operational environment as well as in the research environment. A big advantage of providing a high-level programming language is that Metview developers have been free to change the underlying libraries that Metview uses; for example, the libraries that Metview used for GRIB and BUFR decoding, as well as for data regridding, have been changed in recent years without users having to rewrite any of their code.

Metview is available on ECMWF machines through the *modules* system. Outside ECMWF, it is available in two parts: the binary layer and the Python module. The binary layer can be used as a standalone application through its graphical user interface or its Macro language. One way of installing it is to use the *conda* package manager. From a conda environment on Linux or macOS, the following command will install Metview's binaries: `conda install metview -c conda-forge`. Metview's web pages also provide links to repositories that provide RPM package manager binary installations, and the Ubuntu community maintains a Metview package for their users.

The Python module requires the binary layer to be installed. Available on PyPi, installation is performed simply through the command `pip install metview`. The source code of the Python module is available on github.

Metview's new Python interface

Python is a scripting language released in 1991. Its popularity has increased greatly in the last few years and it is often taught at universities. This means that Python code can be readily written and understood by many scientists. With the wide use of *NumPy* and several other scientific modules for calculations, the Python environment has become increasingly attractive to scientific programmers. Within this context, it was clear that developing a Python interface to Metview would allow it to be used in conjunction with many other scientific packages, and it would enable more users to start using it without having to learn a new programming language. These potential benefits prompted ECMWF to develop this interface, with the help of the software company B-Open.

The goal of this project was to create a Python module that would provide this interface. The module would give programmers the power of Metview's high-level meteorological data access, manipulation and plotting functions while fully interacting with the rest of the Python ecosystem. To achieve this, some simple principles were applied: the module should expose all the functionality of Metview's Macro language, but handle native and scientific Python data types where appropriate. Through the use of the *cffi* package, the Python bindings (special code that bridges two programming languages) are able to load a shared C++ library that is installed with Metview and query for a list of available functions. A small

doi: 10.21957/hv3sp41ir5

translation layer handles the conversion of data types between the C++ and Python code. This system makes it possible for new functions to be added to Metview's C++ code without the need for an update to the Python bindings. It also handles the fact that Python's indexing starts at zero, whereas Metview internally uses 1. Version 1.0.0 of the Python interface was released in December 2018 after much in-house testing of alpha and beta versions.

All of Metview's functions are available in Python through the *metview* namespace, often abbreviated to mv. Figure 1 shows the code required to retrieve 2-metre temperature at different forecast steps and then create a set of time-stamped value averages. The result of the retrieval is a *Fieldset* object, an iterable container with many overridden operators. For example, `t2m_fc - 273.15` would return a new Fieldset with temperatures adjusted to degrees Celsius. Note that most functions can be called in an object-oriented way and a functional way, for example:

```
t2m_fc.valid_date()
```

could also have been written as

```
mv.valid_date(t2m_fc).
```

```
import metview as mv

t2m_fc = mv.retrieve(
    type    = 'fc',
    levtype = 'sfc',
    param   = '2t',
    date    = -5,
    step    = [6,12,18,24],
    grid    = 'o1280')

z = zip(t2m_fc.valid_date(), t2m_fc.average())
for i in z:
    print(i)
```

```
(datetime.datetime(2019, 10, 18, 18, 0), 288.0034532855043)
(datetime.datetime(2019, 10, 19, 0, 0), 287.5664412794946)
(datetime.datetime(2019, 10, 19, 6, 0), 287.9902335998964)
(datetime.datetime(2019, 10, 19, 12, 0), 288.3080870352618)
```

Figure 1 Python code using Metview to retrieve and average a time series of forecast fields.

Providing Metview's functions through a Python module is just one part of making the Python interface work within Python's scientific ecosystem: interoperability with other Python modules requires that data structures can be passed between them. Python has an increasingly well-established set of scientific modules, based around the NumPy module, to efficiently handle data arrays. Metview functions that take vector arguments or return vector results in the Macro language take or return NumPy arrays in Python.

A *Pandas dataframe* is a Python object that contains rows and columns of data, with many methods for analysis and manipulation. Metview's *Geopoints*, *Odb* and *Table* classes can export their data to a Pandas dataframe with the `to_dataframe()` method. Figure 2 shows some code that filters temperature observation data from a BUFR file, computes the differences between those observations and the model forecast data (read from GRIB), and exports the result to a Pandas dataframe for further analysis.

An *xarray* object presents data as a labelled multi-dimensional hypercube. For meteorological data, the dimensions of this hypercube could be latitude, longitude, step and level, for example. In order to map GRIB data into the model used by xarray, a new Python module called *cfgrib* was developed with B-Open. To ensure compatibility with ECMWF data, cfgrib leverages the ecCodes Python module to handle GRIB data. This effort has been very successful, and xarray now officially supports cfgrib as a GRIB engine. Not all GRIB data can be described in such a way. In particular, the geography of non-regular grids, such as reduced Gaussian grids, has no real analogue in xarray, but such grids can still be considered 1-dimensional data arrays for the purposes of computation.

doi: 10.21957/hv3sp41ir5

```
import metview as mv

model_t2m = mv.read('t2m_forecast.grib')
obs_bufr = mv.read('obs.bufr')

obs_t2m = mv.obsfilter(
    data   = obs_bufr,
    parameter = 'airTemperatureAt2M',
    output = 'geopoints')

diff = model_t2m - obs_t2m
df = diff.to_dataframe()
print(df.describe())
```

```
          latitude    longitude   level        value
count  1363.000000  1363.000000  1363.0  1363.000000
mean     46.445407    22.170946     0.0    -0.235432
std       8.546075    14.236381     0.0     1.215969
min      30.110000   -22.730000     0.0    -5.261390
25%      40.435000    13.425000     0.0    -0.905671
50%      45.680000    23.110000     0.0    -0.370365
75%      51.550000    32.985000     0.0     0.310031
max      70.930000    45.950000     0.0     6.031244
```

Figure 2 Python code using Metview to compute the differences between observations and a forecast field. The result is converted into a Pandas dataframe for further analysis.

Metview also uses cfgrib: a Fieldset object has a `to_dataset()` method, which uses cfgrib to generate an xarray dataset from its data. One advantage of this is the convenience with which it is possible to work on specific dimensions of a data cube. Figure 3 shows an example where Metview reads a GRIB file that contains multiple time steps and vertical levels. Then xarray is used to compute the ensemble mean for each level and time step, and the result is passed back to Metview for plotting, with an implicit conversion back to GRIB in the process.

```
import metview as mv

fs = mv.read('wgust_ens.grib')

ds = fs.to_dataset()
ds_mean = ds.mean(dim='number')

mv.plot(ds_mean)
```

Figure 3 Using xarray to aggregate on a dimension of the hypercube.

New ways of interacting with Metview

A popular way of interactively presenting and running Python scripts is through a *Jupyter Notebook*, an open source web application for creating and sharing documents containing text, code and graphics. This allows code to be edited and run from within a web browser that is connected to a Python-based server running either on the same machine or remotely. The output, be it text or graphics, can be saved as part of the notebook, allowing others to view it, even if they do not run the code themselves. If the Jupyter server is running locally, Metview's `plot()` command will invoke an interactive plot window. If the command `mv.setoutput('jupyter')` is called, then subsequent plots will appear inline in the notebook, as shown in Figure 4. Jupyter opens new opportunities to interact with Metview and makes it possible to create rich tutorials in the form of notebooks that users can download and run themselves.

doi: 10.21957/hv3sp41ir5

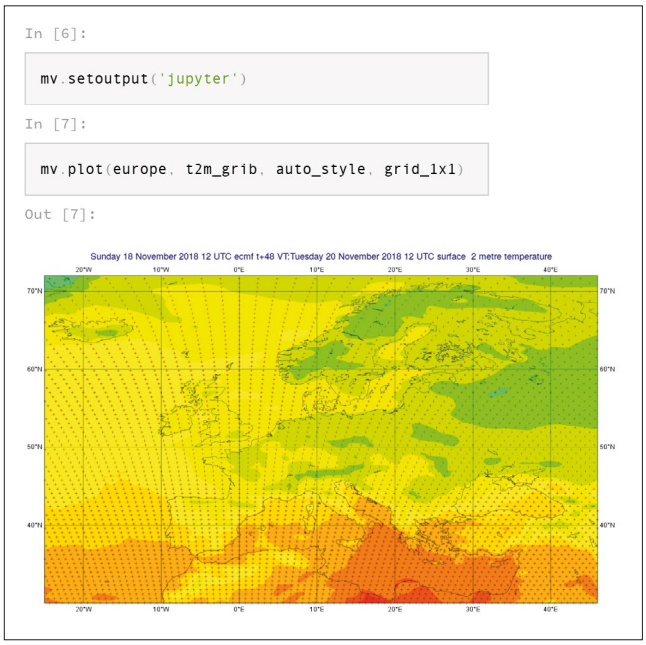


Figure 4 Metview's plots can appear inline in a Jupyter notebook.

Use in verification

The availability of Metview's classes and functions in Python has prompted a significant overhaul of a major package in ECMWF's operational verification software. Metview now takes care of all meteorological data decoding, filtering and post-processing, including geography-aware calculations, extending the applicability of the verification software to a wider range of data formats, grids and parameters. Using Metview has significantly reduced the code base of this application and opened up the path to a more modular and fast-to-develop software architecture.

Use in evaluation and diagnostics

For model evaluation and diagnostics, Metview's Python interface is a clear step forward. In many cases, diagnostic work results in a time series of values (or another type of array). Here the combination in Python of using Metview for the calculations on the GRIB files and the Pandas module for making the time-series analysis has proven to be very powerful. One example application is a simple cyclone tracker intended for advanced diagnostics, for which an example of output is shown in Figure 5a. The program is a Python script that uses the Metview module to first retrieve data from the MARS archive. Next, a Python function performs the cyclone tracking by using several Metview functions on the fieldset containing the mean-sea-level-pressure. It returns a new position of the cyclone, which is used together with the `distance()`, `mask()` and `integrate()` functions in Metview to obtain various diagnostic quantities. These are collected in a Pandas dataframe, which can then for example be used for plotting with the matplotlib library. An example is shown in Figure 5b, where various diagnostics for a tropical cyclone are plotted for two high-resolution forecasts (HRES) and two ensemble control forecasts (ENS CF).

Learning about Metview's Python interface

Although the bulk of the Macro documentation has not yet been revised to include Python code, a large amount of work has gone into updating the Metview Gallery so that every example now has a Python version of the code. Also available from the Gallery are a set of Jupyter notebooks with more detailed examples. A number of webinars were given in the last year, and their content is also available from the Metview web pages. Metview's built-in code editor can also help: if running Metview's graphical user interface, many characteristics of computations and plots can be configured interactively and then committed to code by dropping the edited icons into the Code Editor. The editor also provides help with Metview's Python functions by opening the reference web pages at the correct place.

doi: 10.21957/hv3sp41ir5

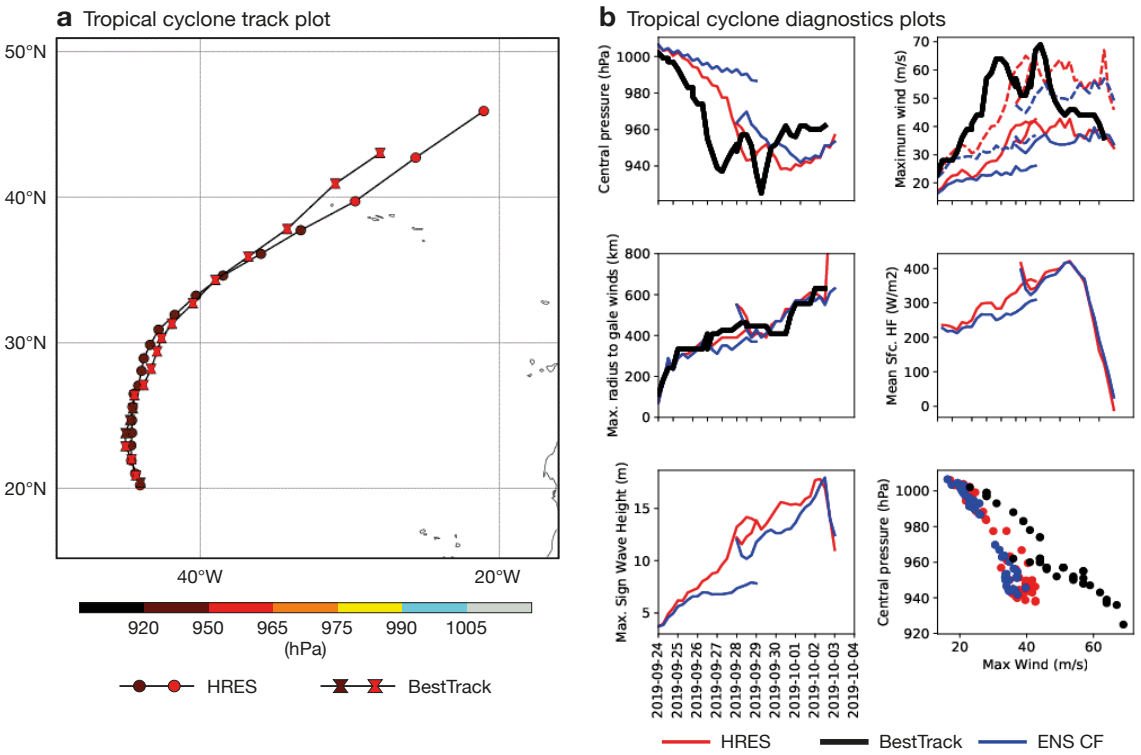


Figure 5 Charts computed and plotted in Python with Metview, showing (a) the track of Hurricane Lorenzo in the ECMWF HRES forecast from 28 September 2019 (circles) and the observed track from the BestTrack database (hourglass). The colours of the symbols indicate the central pressure of the cyclone (in hPa). Panel (b) shows diagnostics of central pressure (top-left), maximum wind speed (top-right, solid) and maximum model wind gusts (top-right, dashed), maximum radius to gale winds (middle-left), surface heat flux in the model (middle-right), maximum significant wave height (bottom-left) and wind/presssure relation (bottom-right). The forecasts shown are HRES (red) and ENS control (blue) from 24 and 28 September. The black lines/dots show BestTrack data.

The future

Since Python is a more accessible language than C++, many users will more easily be able to contribute to Metview. Adding new functions to Metview's Python module would be straightforward, but they would not be accessible from the graphical user interface. Work is planned to add the ability to develop Python-based modules that will run alongside the existing C++ modules as part of Metview's service-oriented architecture. This would enable these modules to be run both from the graphical user interface and from Python. In the meantime, Metview's Python interface code is available on github, and we welcome contributions.

Metview's Python interface is a great start in terms of providing high-level meteorological data handling, manipulation and plotting functions to Python, but it is also considered a fundamental building block for future Python work, which could provide higher-level abstractions and further interfacing with established Python modules.

For more information on Metview, visit: *https://confluence.ecmwf.int/metview*.

Metview's Python interface can be found on github: *https://github.com/ecmwf/metview-python*.

doi: 10.21957/hv3sp41ir5