Metview’s new Python interface

Workshop on developing Python frameworks for earth system sciences.
ECMWF, 2018

Iain Russell
Development Section, ECMWF

Thanks to
Sándor Kertész
Fernando Li
Stephan Siemen
Martin Janousek
What is Metview?

- Data access, examination, manipulation, visualisation
- UNIX, Open Source under Apache Licence 2.0
- Metview is a co-operation project with INPE (Brazil)

EUROPEAN CENTRE FOR MEDIUM-RANGE WEATHER FORECASTS

EUROPEAN CENTRE FOR MEDIUM-RANGE WEATHER FORECASTS
High-level data processing with Metview Macro

• We already had the Macro language…

```
forecast = retrieve (...) # GRIB from MARS
obs = retrieve (...) # BUFR from MARS
t2m_obs = obsfilter(data: obs,
                     parameter: 'airTemperatureAt2M',
                     output: 'geopoints')
plot(forecast - t2m_obs)
```
Why create a Python interface to Metview?

• Enable Metview to work seamlessly within the Python eco-system
  – Bring Metview’s data processing and interactive data inspection tools into Python sessions; interact with Python data structures
  – Use existing solutions where possible (e.g. for multi-dimensional data arrays, data models)

• Enable Metview to be a component of the Copernicus Climate Data Store Toolbox

• Python 3 from the start!
Current Status

- Beta release 0.9.0 (developed with B-Open)

- Available on github and PyPi
  - [https://github.com/ecmwf/metview-python](https://github.com/ecmwf/metview-python)
  - `pip install metview python3 -m metview selfcheck`

- Python layer only – this still requires the Metview binaries to be installed too
Macro / Python comparison

Macro

```python
# Metview Macro

t_fc24 = read('t_fc24.grib')
t_fc96 = read('t_fc96.grib')

diff = abs(t_fc96 - t_fc24)
pos = mcont(legend = 'on',
contour_level_selection_type = 'level_list',
contour_shade = 'on',
contour_shade_method = 'area_fill',
contour_shade_colour_direction = 'clockwise',
contour_max_level = 10,
contour_min_level = 0.5,
contour_level_list = [0.5, 1, 2, 4, 10],
contour_shade_max_level_colour = 'red',
contour_shade_min_level_colour = 'orange_yellow')
xseurope = mxsectview(line = [55, -6, 43, 16])
plot(xseurope, diff, pos)
```

Python

```python
import metview as mv

t_fc24 = mv.read('t_fc24.grib')
t_fc96 = mv.read('t_fc96.grib')
daiff = mv.abs(t_fc96 - t_fc24)
pos = mv.mcont(legend = 'on',
contour_level_selection_type = 'level_list',
contour_shade = 'on',
contour_shade_method = 'area_fill',
contour_shade_colour_direction = 'clockwise',
contour_max_level = 10,
contour_min_level = 0.5,
contour_level_list = [0.5, 1, 2, 4, 10],
contour_shade_max_level_colour = 'red',
contour_shade_min_level_colour = 'orange_yellow')
xseurope = mv.mxsectview(line = [55, -6, 43, 16])
mv.plot(xseurope, daiff, pos)
```
Macro / Python comparison

```python
# Metview Macro

1 t_fc24 = read('t_fc24')
2 t_fc96 = read('t_fc96')
3 diff = abs(t_fc24 - t_fc96)
4 pos = mcont(
5    legend
6    contour_lev
7    contour_sha
8    contour_sha
9    contour_sha
10   contour_sha
11   contour_sha
12   contour_sha
13   contour_sha
14   contour_sha
15   contour_sha
16   contour_sha
17   contour_sha
18   contour_sha
19   contour_sha
20   contour_sha
21   contour_sha
22 xs_europe = mxs
23 plot(xs_europe,
```

```python
# Python

```
Data types (1)

- All Metview Macro functions can be called from Python, e.g. `mv.covar(f1, f2)`
- Data types returned are either standard Python types (numbers, lists, strings, datetimes), numpy arrays …
- … or thin class wrappers around more complex objects such as fieldsets, geopoints or ODB
• Can extract numpy arrays from most Metview data types
• Example: compute and plot principal components of ensemble forecasts stored in GRIB

```python
[1]:
```import` metview as mv
`import` numpy as np
`from` scipy import linalg as LA

File `z500_ens.grib` contains 500 hPa geopotential ECMWF data into a `Fieldset` which is Metview's own class to handle data.

```python
[2]:
`fs = mv.read("./z500_ens.grib")`

We will compute the principal components (PC) using Metview:

```python
[3]:
`v = fs.values()
`print`(v.shape)

(51, 3266)

For the PCA we center the data, create the covariance:

```python
[4]:
`v -= np.mean(v, axis = 0)
`cov = np.cov(v, rowvar = `False`
`evals, evecs = LA.eigh(cov)

Finally, we plot each field with a custom title. We compute the ensemble mean and spread on the fly with fieldset functions from Metview:

```python
[12]:
`mv.plot(dw[0], fs.mean(), mv.mtext(text_line_1 = "ENS mean"),
`dw[1], fs.stdev(), mv.mtext(text_line_1 = "ENS spread"),
`dw[2], g[0], cont_pc, mv.mtext(text_line_1 = "PC1"),
`dw[3], g[1], cont_pc, mv.mtext(text_line_1 = "PC2")
```
Data types (3)

• Can also export Metview Geopoints (and BUFR via the filter), ODB and Table data types to **pandas** Dataframes (table-like format, common in scientific data processing)

• Example:
  - run a filter on a BUFR file containing tropical cyclone tracks
  - Convert to pandas dataframe

```
import metview as mv
import pandas as pd

f = mv.read("tropical_cyclone.bufr")
res = mv.bufr_filter(
    data = f,
    output = "CSV",
    message_index = 1,
    custom_condition_count = 1,
    custom_key_1 = "ensembleMemberNumber",
    custom_value_1 = 2,
    parameter_count = 1,
    parameter_1 = "pressureReducedToMeanSeaLevel",
    extract_mode = "all"
)

df=res.to_dataframe()

print(df)
```

<table>
<thead>
<tr>
<th>date</th>
<th>latitude</th>
<th>level</th>
<th>longitude</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2015-11-18</td>
<td>5.4</td>
<td>0.0</td>
<td>156.9</td>
<td>100000.0</td>
</tr>
<tr>
<td>2015-11-18</td>
<td>6.3</td>
<td>0.0</td>
<td>155.8</td>
<td>100000.0</td>
</tr>
<tr>
<td>2015-11-18</td>
<td>6.8</td>
<td>0.0</td>
<td>154.6</td>
<td>100300.0</td>
</tr>
<tr>
<td>2015-11-18</td>
<td>7.7</td>
<td>0.0</td>
<td>153.8</td>
<td>100100.0</td>
</tr>
<tr>
<td>2015-11-18</td>
<td>8.2</td>
<td>0.0</td>
<td>152.1</td>
<td>100300.0</td>
</tr>
<tr>
<td>2015-11-18</td>
<td>8.8</td>
<td>0.0</td>
<td>151.3</td>
<td>100000.0</td>
</tr>
<tr>
<td>2015-11-18</td>
<td>9.4</td>
<td>0.0</td>
<td>150.7</td>
<td>100300.0</td>
</tr>
<tr>
<td>2015-11-18</td>
<td>9.9</td>
<td>0.0</td>
<td>149.9</td>
<td>100100.0</td>
</tr>
<tr>
<td>2015-11-18</td>
<td>10.2</td>
<td>0.0</td>
<td>148.7</td>
<td>100300.0</td>
</tr>
</tbody>
</table>
• Can also export Metview Fieldsets to \texttt{xarray} Datasets
  - Provides data in the Common Data Model used by netCDF and the CDS
• Uses the \texttt{cfgrib} package developed by B-Open, available on github and PyPi
• Can also pass some \texttt{xarray} datasets into Metview functions
Running Metview from an IDE

- From command line or IDE, e.g. Jupyter, PyCharm – can provide code completion and debugging facilities
- We can improve the amount of information we supply to IDEs
Running Python scripts from a Metview session (1)
Running Python scripts from a Metview session (2)
Ways to run a Metview Python script (5)

```python
21
22
23 coloured_symbols = mv.msymb(
24 legend = "on",
25 symbol_type = "marker",
26 symbol_table_mode = "advanced",
27 symbol_outline = "on",
28 symbol_outline_colour = "charcoal",
29 symbol_advanced_table_max_level_colour = "red",
30 symbol_advanced_table_min_level_colour = "blue",
31 symbol_advanced_table_colour_direction = "clockwise",
32 symbol_advanced_table_height_list = 0.4
33 )
34
35 mv.plot(view, t2m, coloured_symbols)
```
Implementation details

• We use the **cffi** package to bridge C++/Python
  – Links to a shared library of Metview functions
  – Some of these functions call other Metview services (e.g. Cross Section, uPlot)
  – Metview now needs to be context-aware because Macro uses 1-based indexing, Python uses 0-based indexing

• Metview binaries vs Python layer independence
  – Try to keep the interface functions as generic as possible
  – The Python layer queries the binaries for the list of available functions
  – New data types in Metview will require a little code in the Python layer

• Faster import
  – we noticed that importing some modules was quite slow (e.g. IPython for detecting the Jupyter environment), so we only import when actually needed
Feedback

- Feedback has so far been very positive
- We have some enthusiastic users: “It combines all the power of Metview with all the power of Python!” (internal user to Iain, Oct 2018)
- As is often the case, we only hear from users if they encounter a problem, but log files suggest quite a lot of activity

```python
### read in the MSLP analysis for calculation of surface pressure ###
mslpan = mv.read("/path/to/data/msl_elda_bg_\"+datein+\"_+timein\".grb")

### read in the 2m temperature ###
t2m_an = mv.read("/path/to/data/t2m_elda_bg_\"+datein+\"_+timein\".grb")

## loop through the EDA members ##
for iens in range(0,1): #26
    
    ### q ###
    if(typein == "obs"): valaq = mv.values(data_q,'obsvalue'+str(iens)) ; valaq[valaq < 0] = 0
    if(typein == "bg"): valaq = mv.values(data_q,'obsvalue'+str(iens))
    temp = np.column_stack((latg,long))
    temp = np.column_stack((temp,levelq))
    dfq = pd.DataFrame(data=temp, columns=['lat', 'lon', 'level'])
    dfq['q'] = valaq
    dfq['date'] = dateq
    dfq['time'] = timeq
    dfq = dfq.loc[([dfq['level'] > 7000])]

    ### u ###
```

EUROPEAN CENTRE FOR MEDIUM-RANGE WEATHER FORECASTS
Reaching out – forecast verification toolbox (1)

- One of the major verification packages at ECMWF has been using Python for > 10 years
- Provides a simple interface to describe which statistics of what forecasts are to be computed
  - data format details, etc are hidden
- The package has been successful, with some shortcomings:
  - interfacing to low-level data decoding packages and consequently implementing own geographical and meteorological algorithms
  - lack of flexibility for newly emerged verification and diagnostic techniques and requirements
Reaching out – forecast verification toolbox (2)

• The solution: a new verification toolbox built on Metview’s Python layer
  – replace the data interfacing and manipulation layer with Metview
  – take the opportunity to improve other layers
  – involve developers of other verification packages to broaden the scope of the toolbox
  – repack the user interface layer to fully use the toolbox (to support the existing users)
Future

• Advertise beta version more widely to get more feedback
• Release version 1.0.0 – end of 2018 / early 2019?

• Provide tools for automatic translation from Macro to Python
• Improve information available for IDEs (e.g. function descriptions)

• Investigate conda for packaging Metview’s binaries and the Python layer together

• Plenty more we want to do!
For more information…

• Email us:
  – Developers: metview@ecmwf.int
  – Support: software.support@ecmwf.int

• Visit our web pages:
  – http://confluence.ecmwf.int/metview

• Download (Metview source, binaries)
• Documentation and tutorials available
• Metview articles in ECMWF newsletters
• e-Learning material
• Download Metview’s Python interface:
  – pip install metview
  – https://github.com/ecmwf/metview-python

Questions?