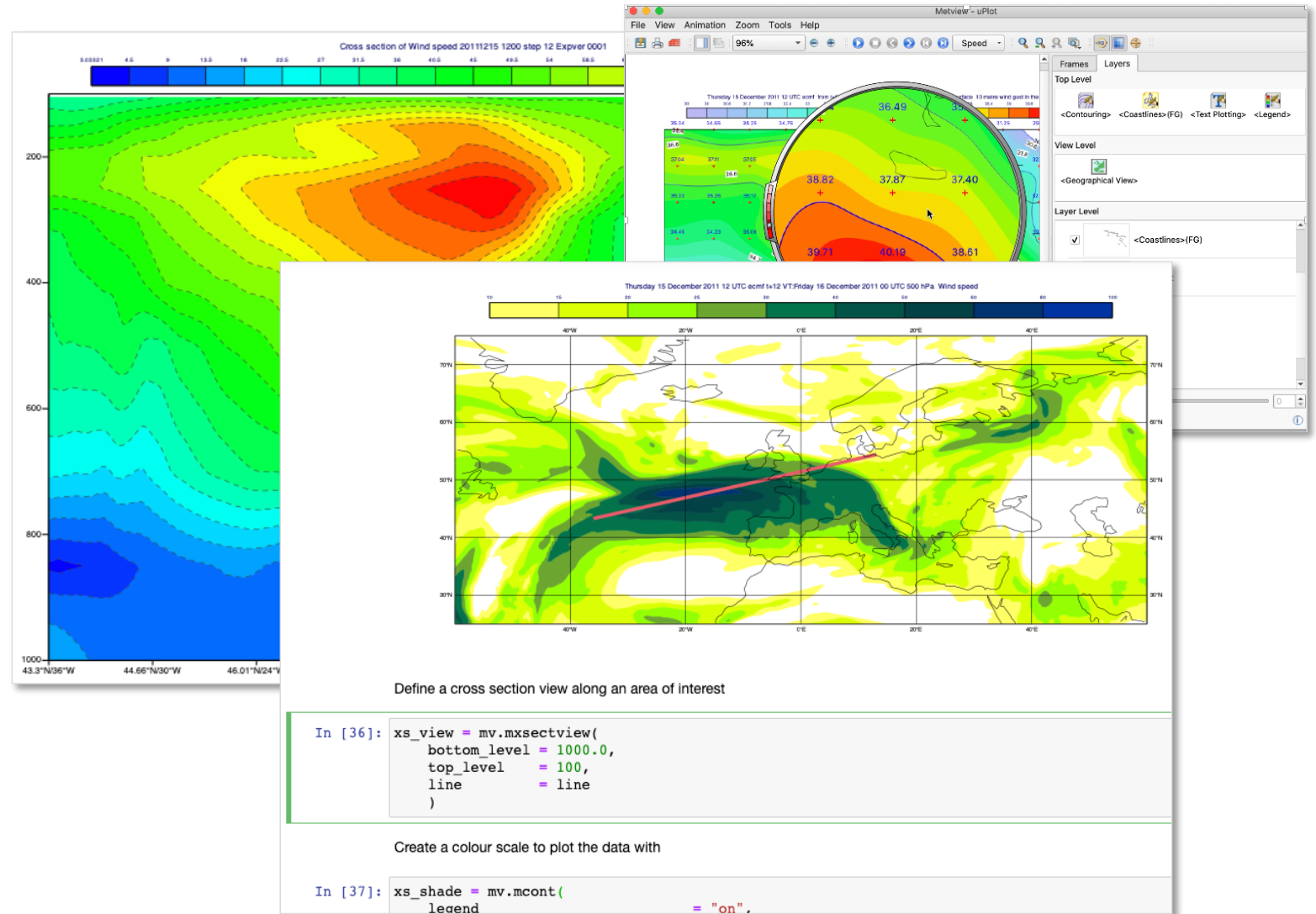


Interactive Analysis of ECMWF Data

Webinar - May 14, 2020

Iain Russell
Sándor Kertész

Development Section, ECMWF



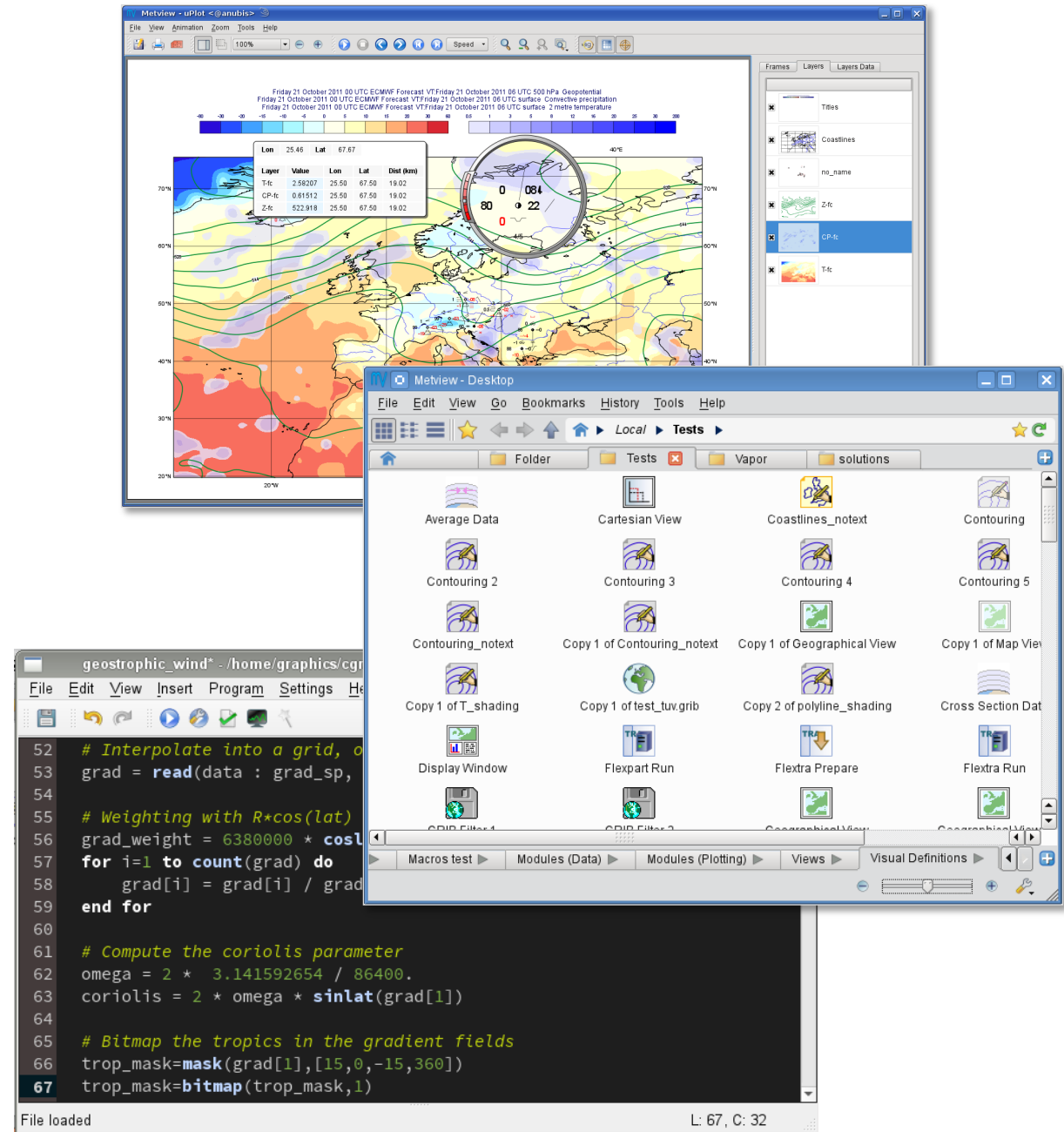
Outline

The webinar will be based on Metview's user interface, interactive plot window and Jupyter notebooks (using Metview's Python interface)

- Interactive data plotting and inspection
- Converting interactive work into Python code
- Finding extreme values in Python
- Extracting and plotting time series in Python
- Exploring vertically - profiles and cross sections
- Where to find out more

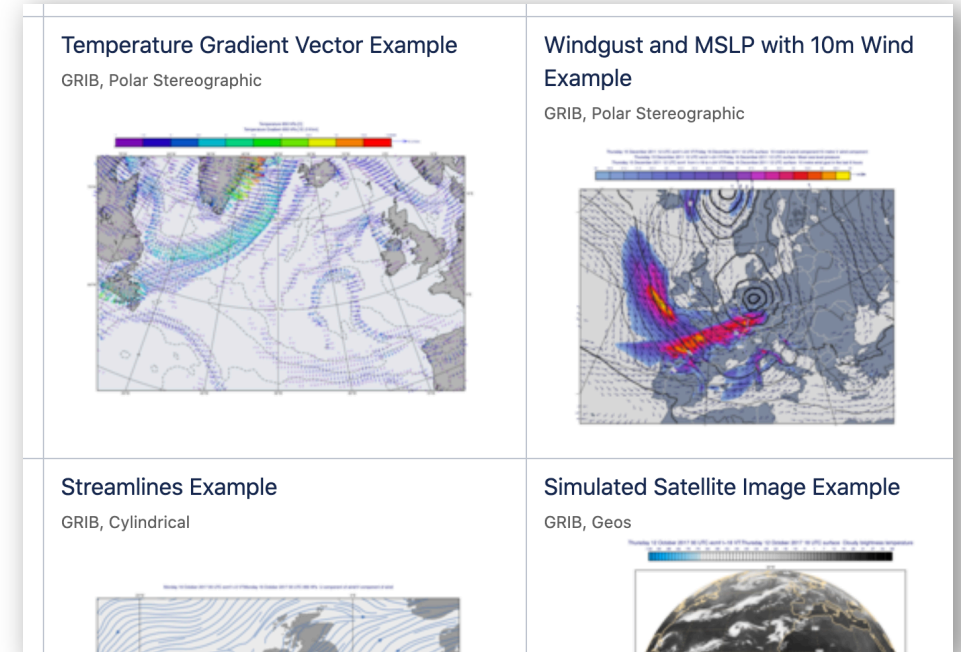
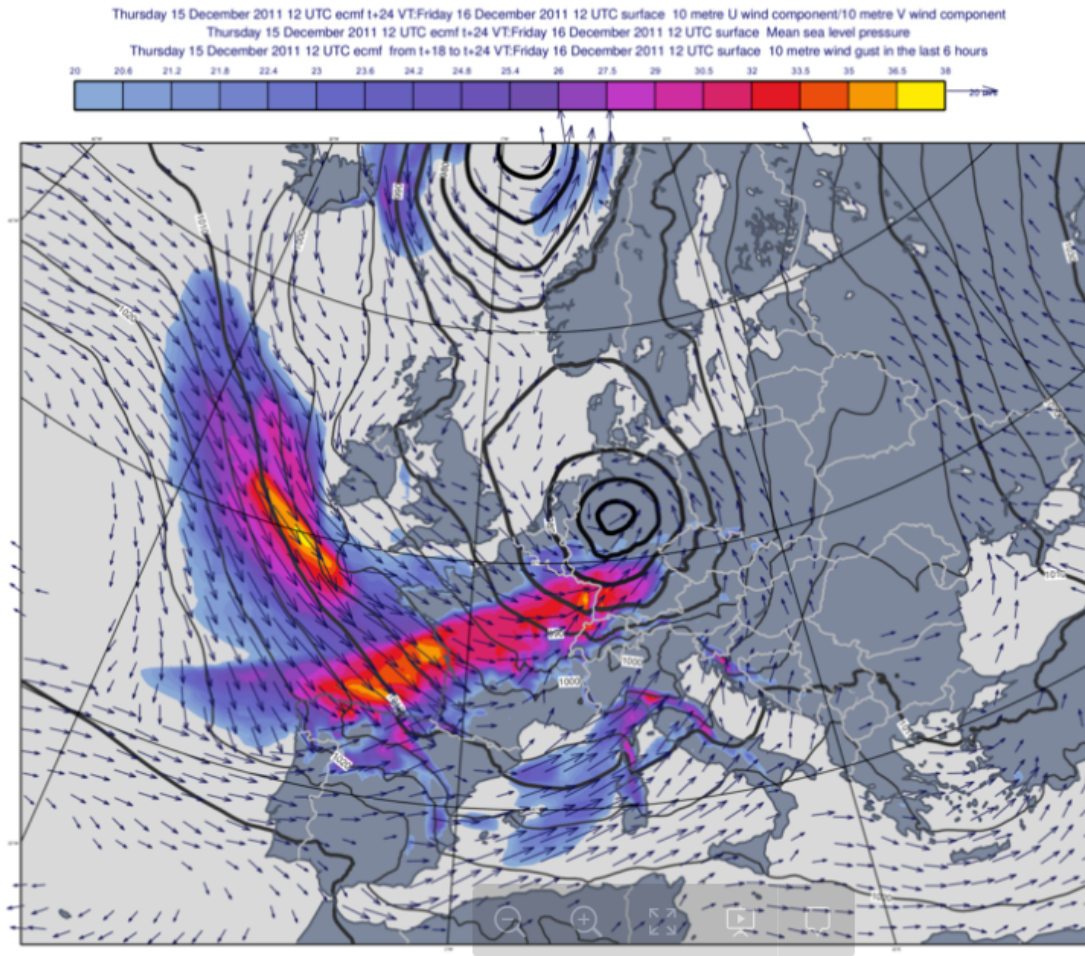
What is Metview?

- Workstation software, runs on UNIX, from laptops to supercomputers (including macOS)
- Open source, Apache 2.0 license
- Visualisation
- Data processing
- Icon based user interface
- Powerful scripting languages - Macro and Python



The data we will use

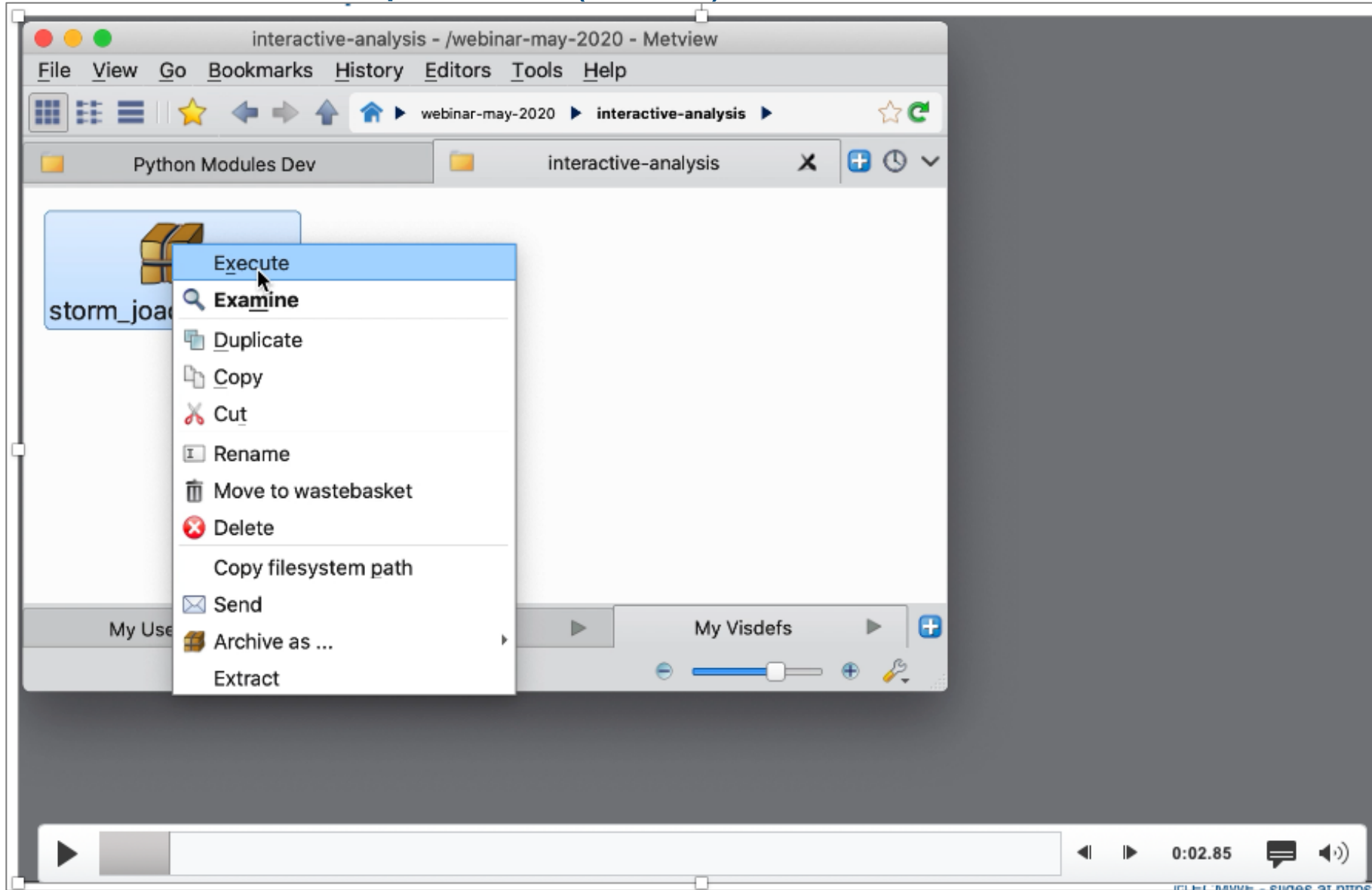
Storm Joachim, December 2011



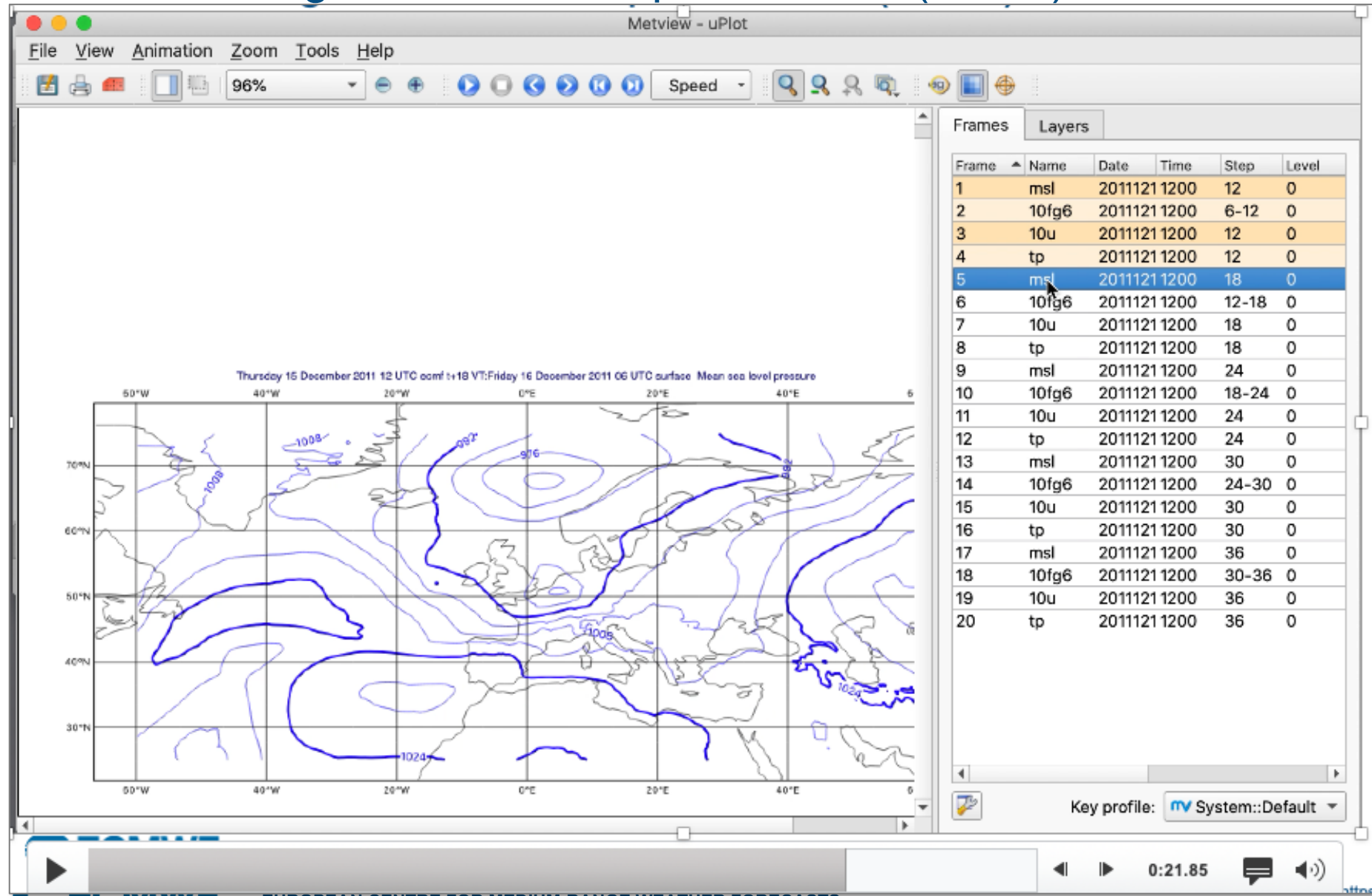
- Data from an example in the Metview Gallery
- All data is in **GRIB** format:
 - Deterministic surface forecast (wind gust)
 - Deterministic atmospheric forecast (wind U/V)
- Retrieved from the **MARS** archive and post-processed with Metview

We start with showing the interactive plot window

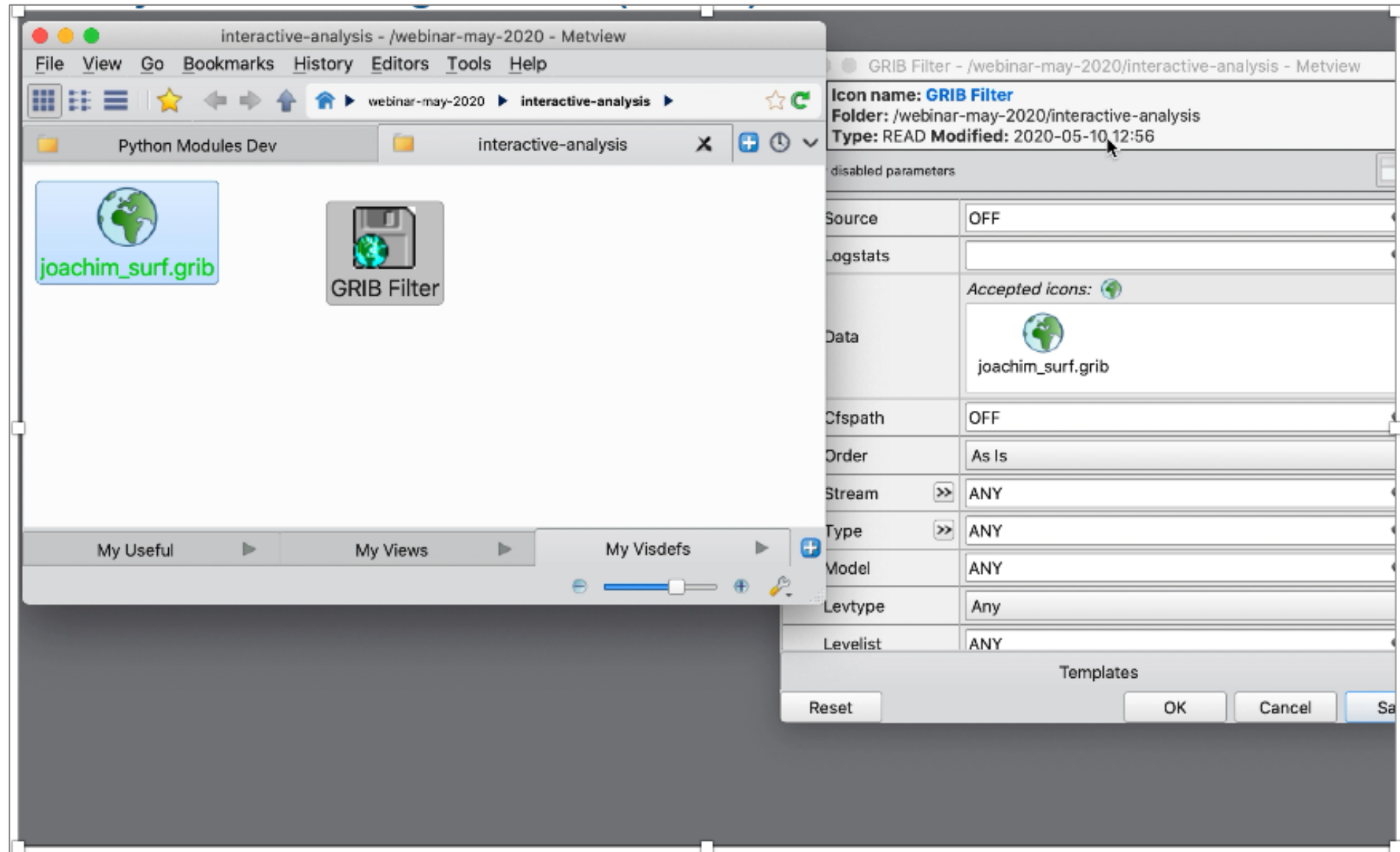
Check the example data (Video)



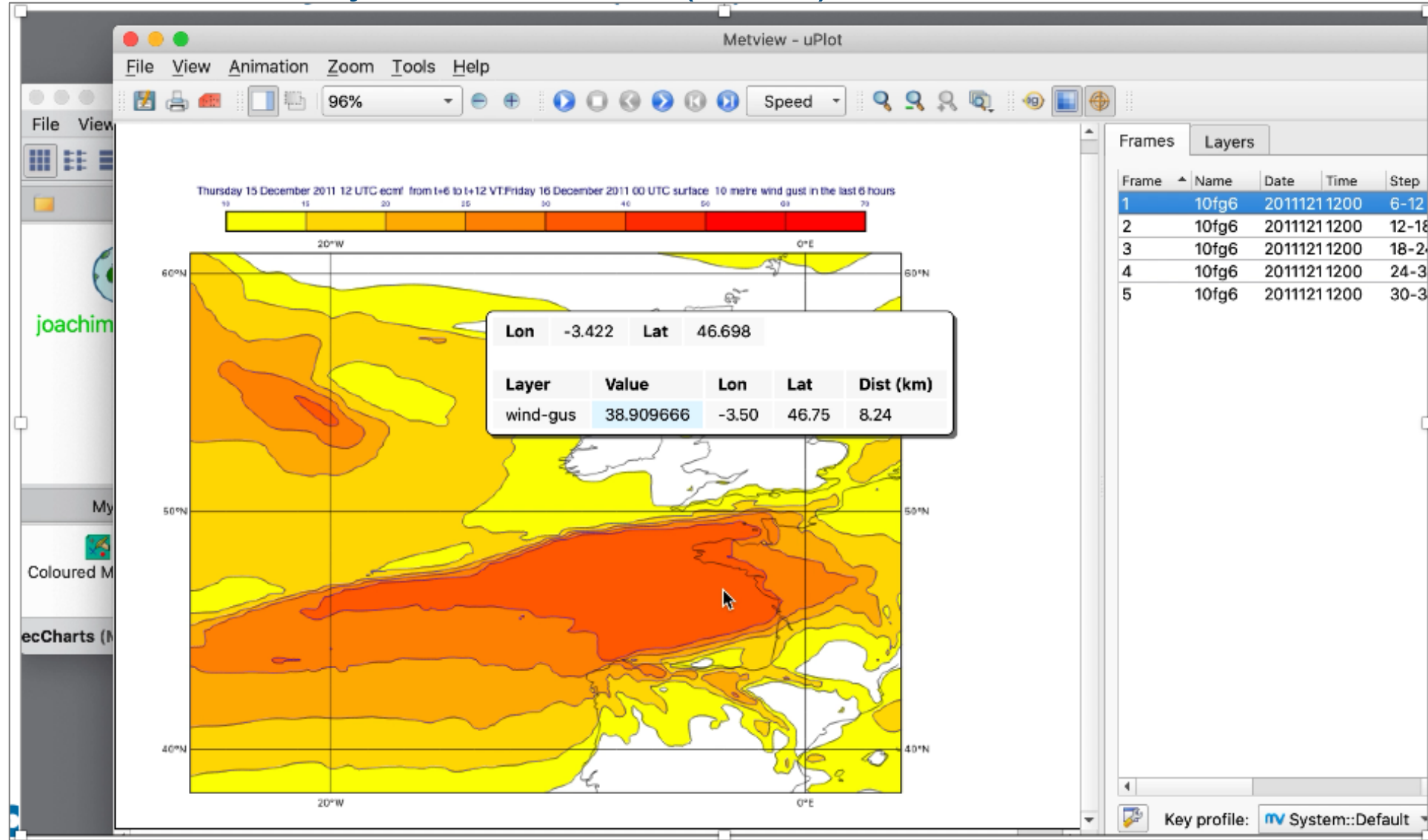
First look using the interactive plot window (Video)



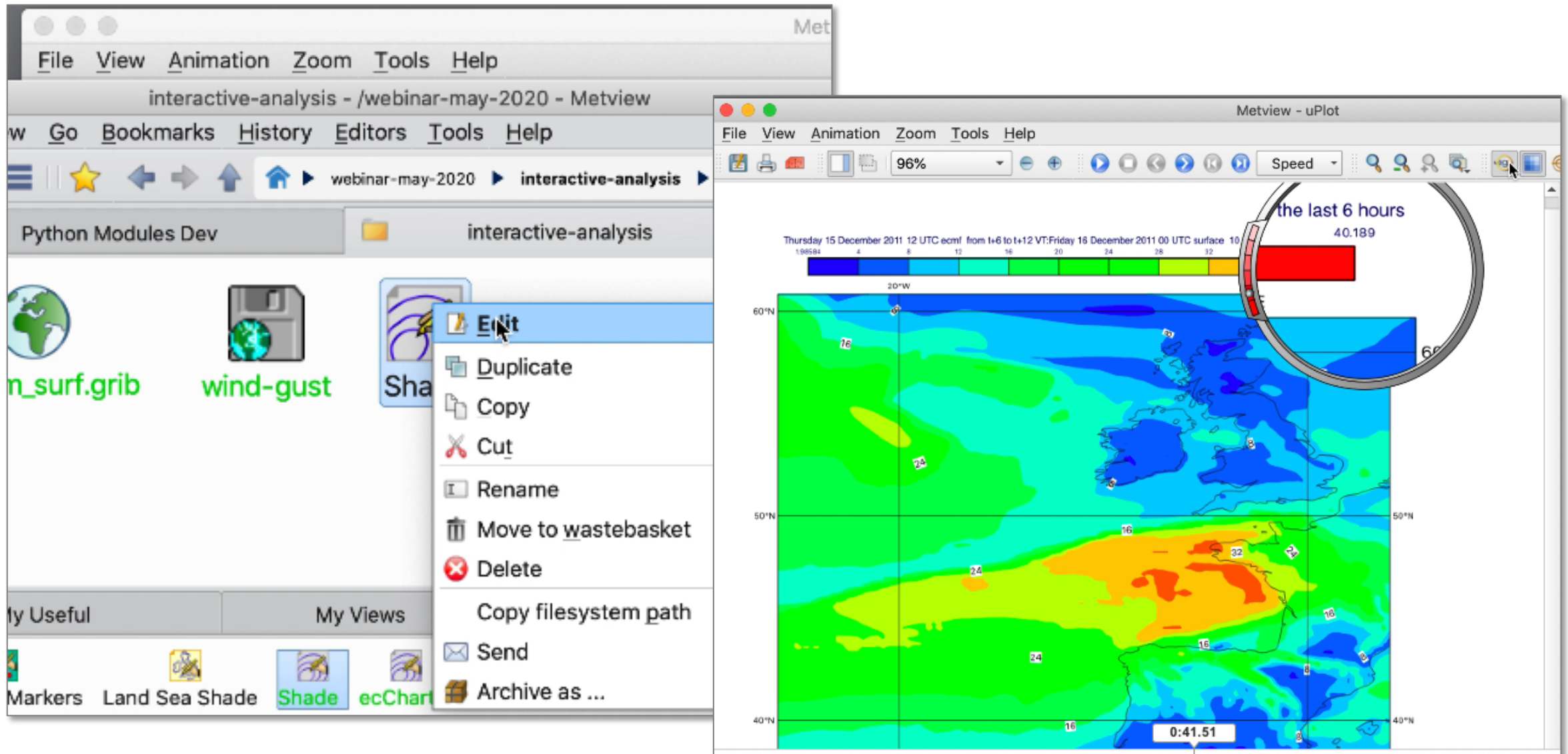
Filter just the wind gust data (Video)



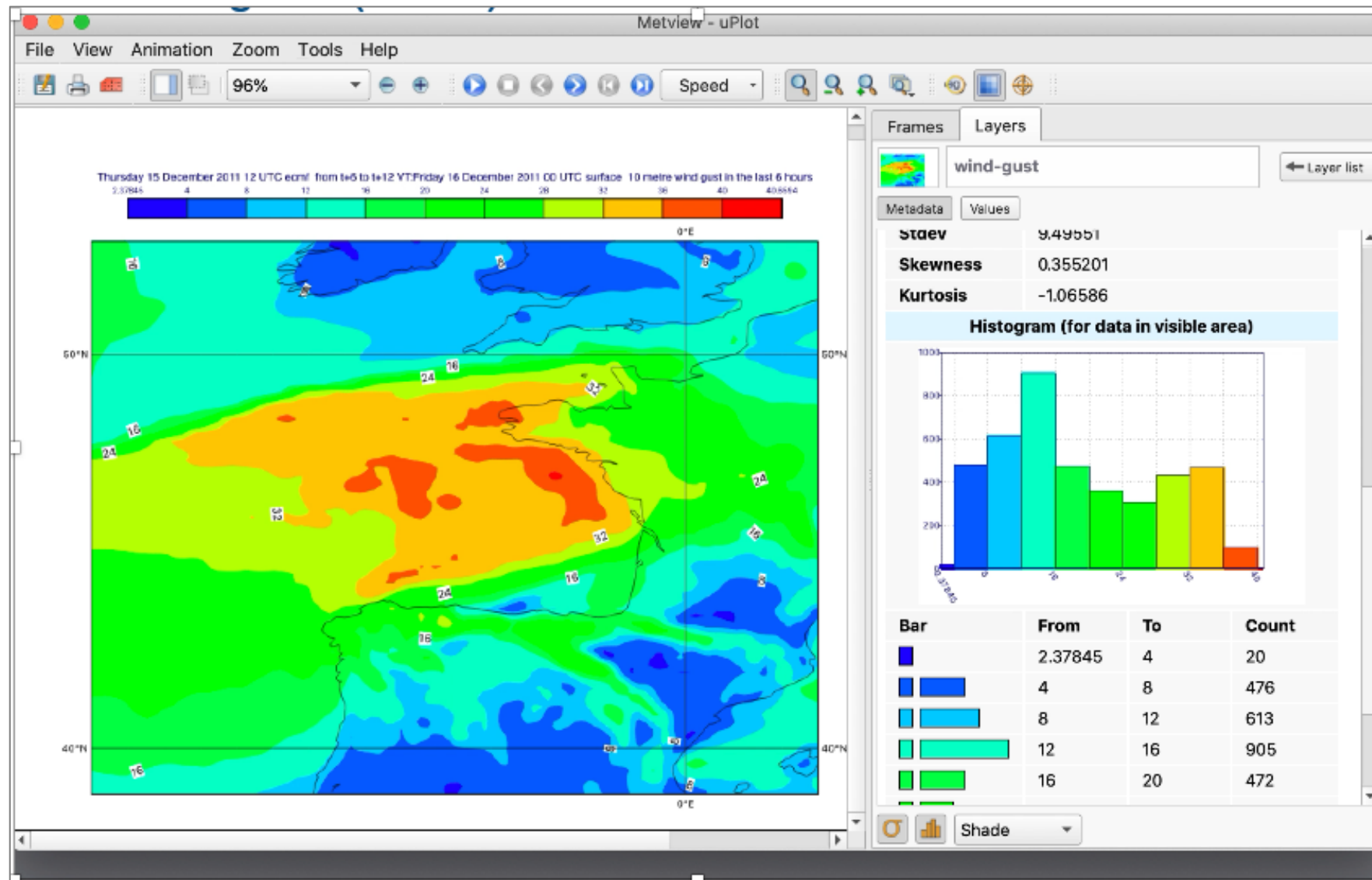
Use ecCharts style for first look (Video)



Apply auto-scale shading icon (Video)



Show histogram (Video)



Adjust contouring (Video)

The screenshot shows the configuration window for 'Shade 6' in Metview. The window title is 'Shade 6 - /System/MvTemporaryCache_11494 - Metview'. It contains several sections for configuring the contouring process.

Icon name: Shade 6
Folder: /System/MvTemporaryCache_11494
Type: MCONT Modified: 2020-05-10 15:20

Show disabled parameters: Legend, Contour, Contour Reference Level, Contour Level Selection Type, Contour Max Level, Contour Min Level, Contour Shade Max Level, Contour Shade Min Level, Contour Level Count.

Contouring Settings:

- Contour: On Off
- Contour Reference Level: 0,0
- Contour Level Selection Type: Count
- Contour Max Level: 1.0E+21
- Contour Min Level: 30
- Contour Shade Max Level: 1.0E+21
- Contour Shade Min Level: -1.0E+21
- Contour Level Count: 20

Contour Shade Method: Area Fill

Contour Shade Max Level Colour: Red

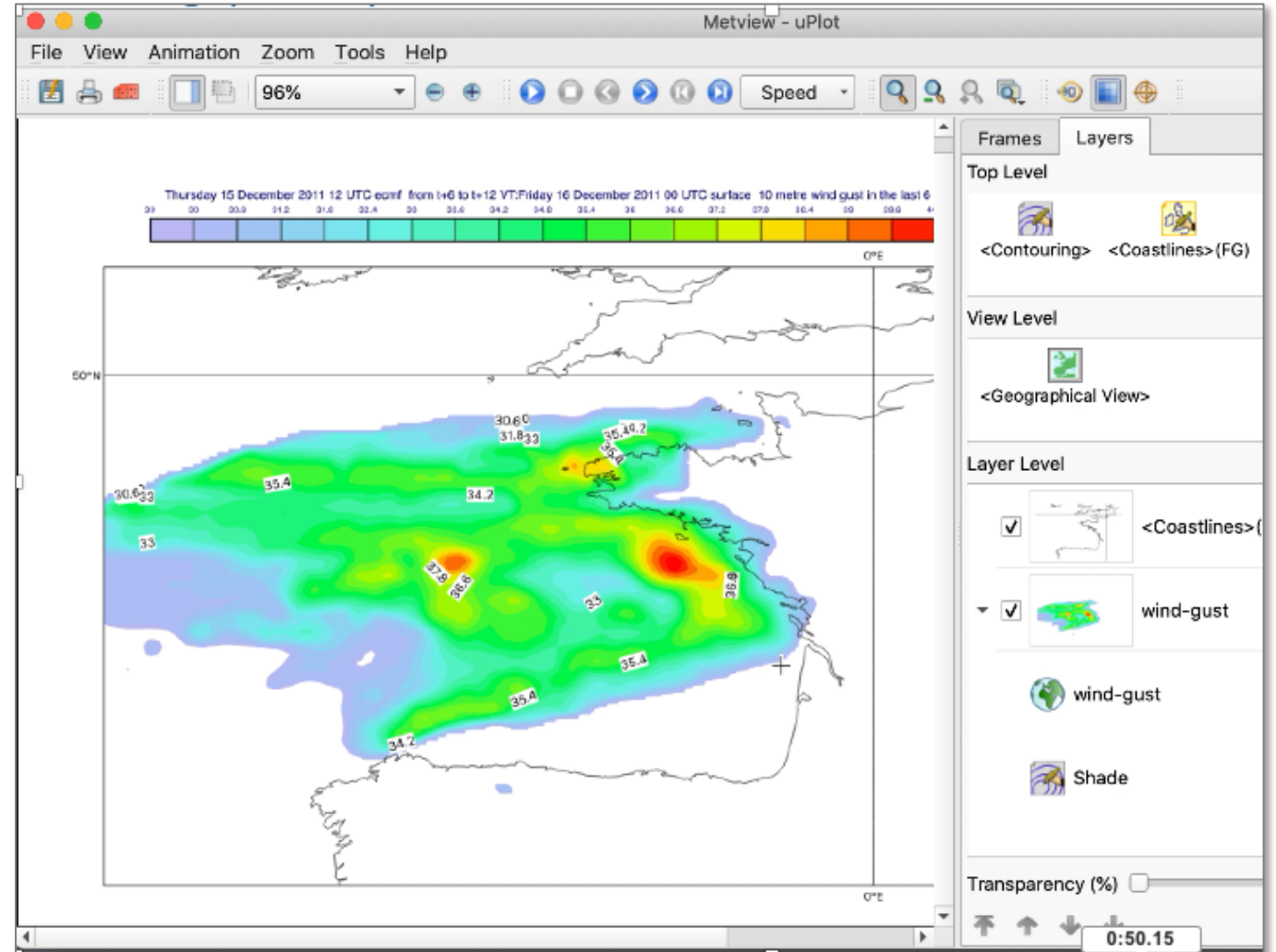
Contour Shade Min Level Colour: RGB(0.7511,0.7511,0.943)

Contour Shade Colour Direction: Clockwise

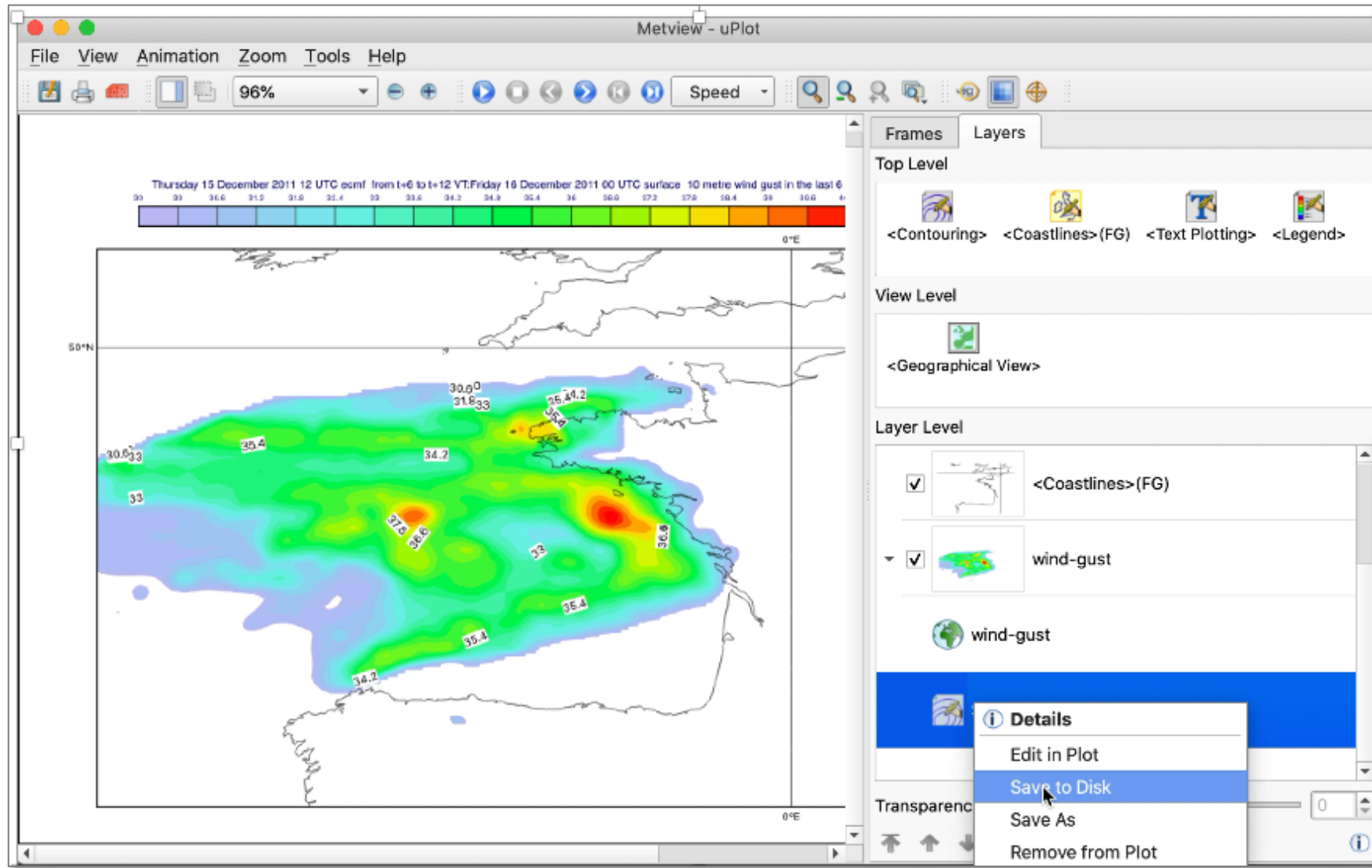
Contour Legend Text:

Color Selection: Wheel, Grid, Red: 19, Green: 19, Blue: 24, Opacity: [slider], HTML: #c, Macro: 'R'

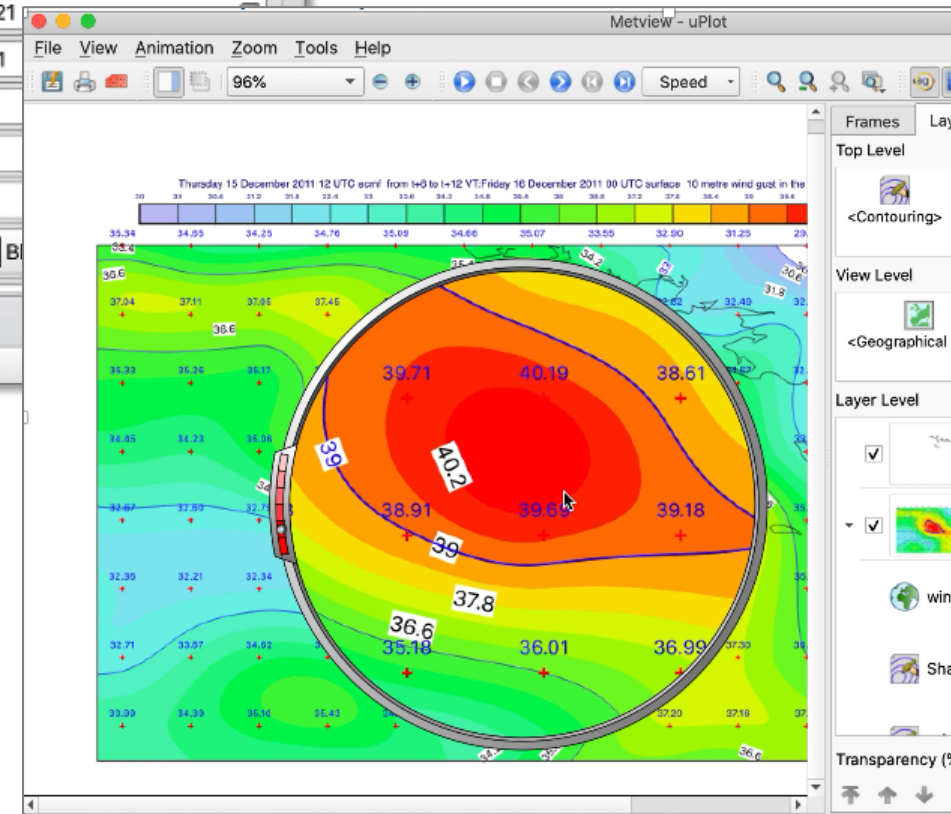
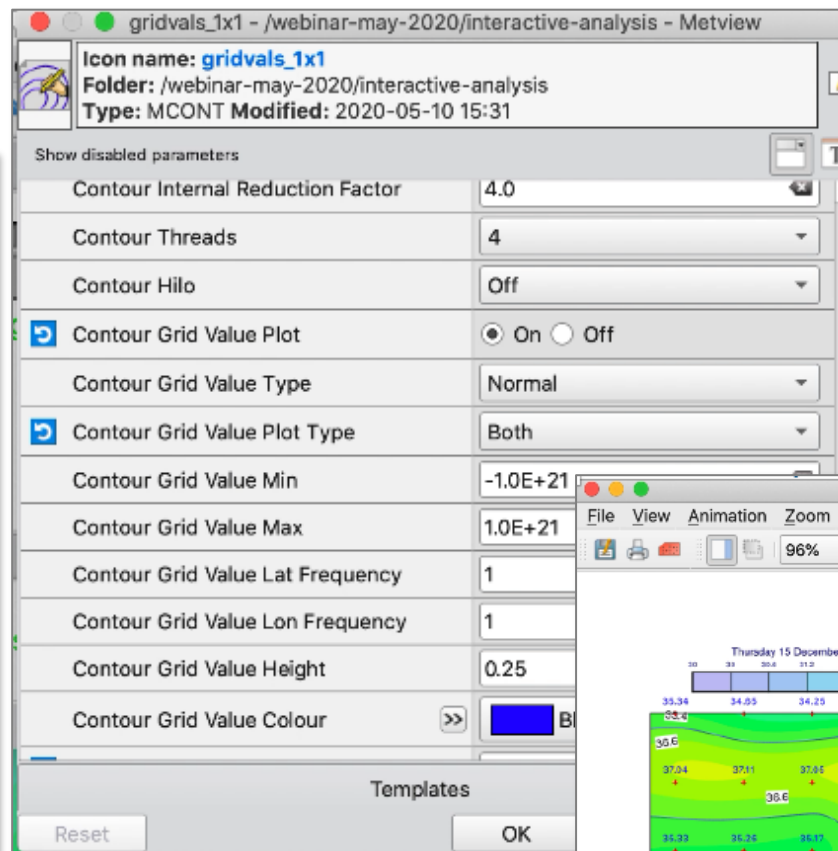
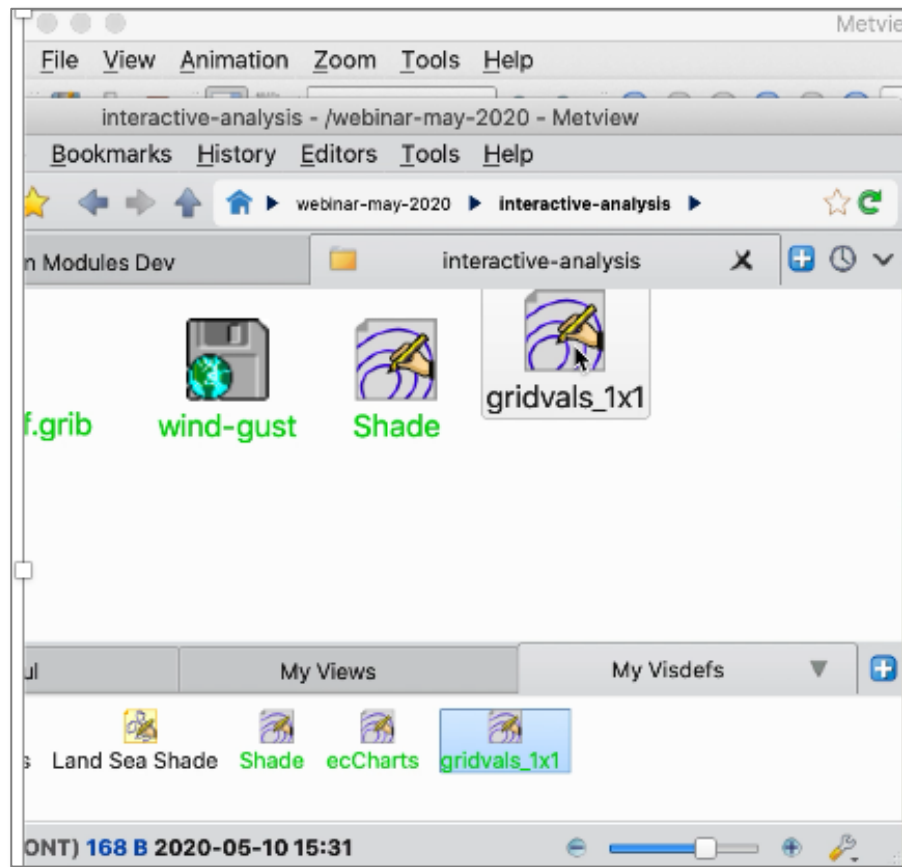
Buttons: Reset, OK, Cancel, Save



Save edited contouring (Video)



Plot grid point values (Video)



Generate Python script from icons (Video)

The image shows a screenshot of the Metview interactive analysis interface. On the left, a window titled 'interactive-analysis - /webinar-may-2020 - Metview' displays a file browser with icons for 'joachim_surf.grib', 'wind_gust', 'Shade', 'gridvals_1x1', and 'Python Script.py'. The 'Shade' icon is highlighted with a blue border. Below the file browser, there are tabs for 'My Useful', 'My Views', and 'My Visdefs', and a status bar showing 'Shade (MCONT) 325 B 2020-05-10 15:24'.

On the right, a code editor window titled 'Python Script.py - /Users/fain/metview/' shows the following Python code:

```
2
3 # Importing : /webinar-may-2
4
5 joachim = mv.read("joachim_s
6
7 # Importing : /webinar-may-2
8
9 wind_gust = mv.read(
10     param = "10fg6",
11     data = joachim
12 )
13
14
15
```

Below the code editor, a smaller window titled 'Python Script.py - /Users/fain/metview/webinar-may-2020/interactive-analysis/Python Script.py' shows the generated Python code for the 'Shade' icon:

```
14 # Importing : /webinar-may-2020/interactive-analysis/Shade
15
16 shade = mv.mcont(
17     legend = "on",
18     contour = "off",
19     contour_min_level = 30,
20     contour_level_count = 20,
21     contour_shade = "on",
22     contour_shade_method = "area_fill",
23     contour_shade_max_level_colour = "red",
24     contour_shade_min_level_colour = "RGB(0.7511,0.7511,0.943)",
25     contour_shade_colour_direction = "clockwise"
26 )
27
```

Our work continues in a Jupyter notebook

Step 1: Getting the data

read()

to give us a Fieldset object
containing 5 fields



```
import metview as mv
```

Read the data from GRIB file into a Metview Fieldset:

```
wg = mv.read("joachim_wind_gust.grib")  
print(wg)  
print(len(wg))
```

```
<metview.bindings.Fieldset object at 0x106294d10>  
5
```

The data is a **Fieldset** with 5 fields. Let's inspect the contents with a few GRIB keys:

```
mv.grib_get(wg, ['shortName', 'dataDate', 'dataTime',  
                'stepRange', 'validityDate', 'validityTime'])
```

```
[['10fg6', '20111215', '1200', '6-12', '20111216', '0'],  
 ['10fg6', '20111215', '1200', '12-18', '20111216', '600'],  
 ['10fg6', '20111215', '1200', '18-24', '20111216', '1200'],  
 ['10fg6', '20111215', '1200', '24-30', '20111216', '1800'],  
 ['10fg6', '20111215', '1200', '30-36', '20111217', '0']]
```

grib_get()

to access **ecCodes** keys
from the GRIB header



Step 2: Find the minimum and maximum values

`min/maxvalue()`

Gives min/max values over
all the fields



Iterate over the fields to
see max per field



First let's check the minimum and maximum values over all the fields:

```
print(mv.minvalue(wg), mv.maxvalue(wg))
```

```
0.9916973114013672 39.74169731140137
```

Now, the maximum values for each field - iterate over the fieldset:

```
all_maxes = [mv.maxvalue(f) for f in wg]  
all_maxes
```

```
[39.74169731140137,  
37.20536518096924,  
37.363698959350586,  
37.525485038757324,  
37.46825695037842]
```

Step 3: Find the locations of the max values, produce time series

Look at just the first field in the Fieldset



```
wg0 = wg[0]  
max0 = all_maxes[0]
```

Find the locations where the value equals the maximum:

Get list of lat/lon locations where the max value occurs



```
max_location = mv.find(wg0, max0)  
max_location
```

```
[[47.0, -3.5]]
```

Obtain a time series of values at this location (one value from each field):

Over all fields: get the gridpoint value at this location and also the validity date/time



```
vals_for_point = mv.nearest_gridpoint(wg, max_location[0])  
times = mv.valid_date(wg)  
for tv in zip(times, vals_for_point):  
    print(tv)
```

```
(datetime.datetime(2011, 12, 16, 0, 0), 39.74169731140137)  
(datetime.datetime(2011, 12, 16, 6, 0), 36.70536518096924)  
(datetime.datetime(2011, 12, 16, 12, 0), 25.863698959350586)  
(datetime.datetime(2011, 12, 16, 18, 0), 32.525485038757324)  
(datetime.datetime(2011, 12, 17, 0, 0), 27.468256950378418)
```

Step 5: Set up a time series plot



Axis



Cartesian View

4/8/15/16€



Input Visualiser



Graph Plotting

```
haxis = mv.maxis(  
    axis_type = "date",  
    axis_date_type = "hours",  
    axis_hours_label = "on",  
    axis_hours_label_height = 0.4,  
    axis_years_label_height = 0.4,  
    axis_months_label_height = 0.4,  
    axis_days_label_height = 0.4  
)  
  
ts_view = mv.cartesianview(  
    x_automatic = "on",  
    x_axis_type = "date",  
    y_automatic = "on",  
    horizontal_axis = haxis  
)  
  
curve_wg = mv.input_visualiser(  
    input_x_type = "date",  
    input_date_x_values = times,  
    input_y_values = vals_for_point)  
  
visdef = mv.mgraph()
```

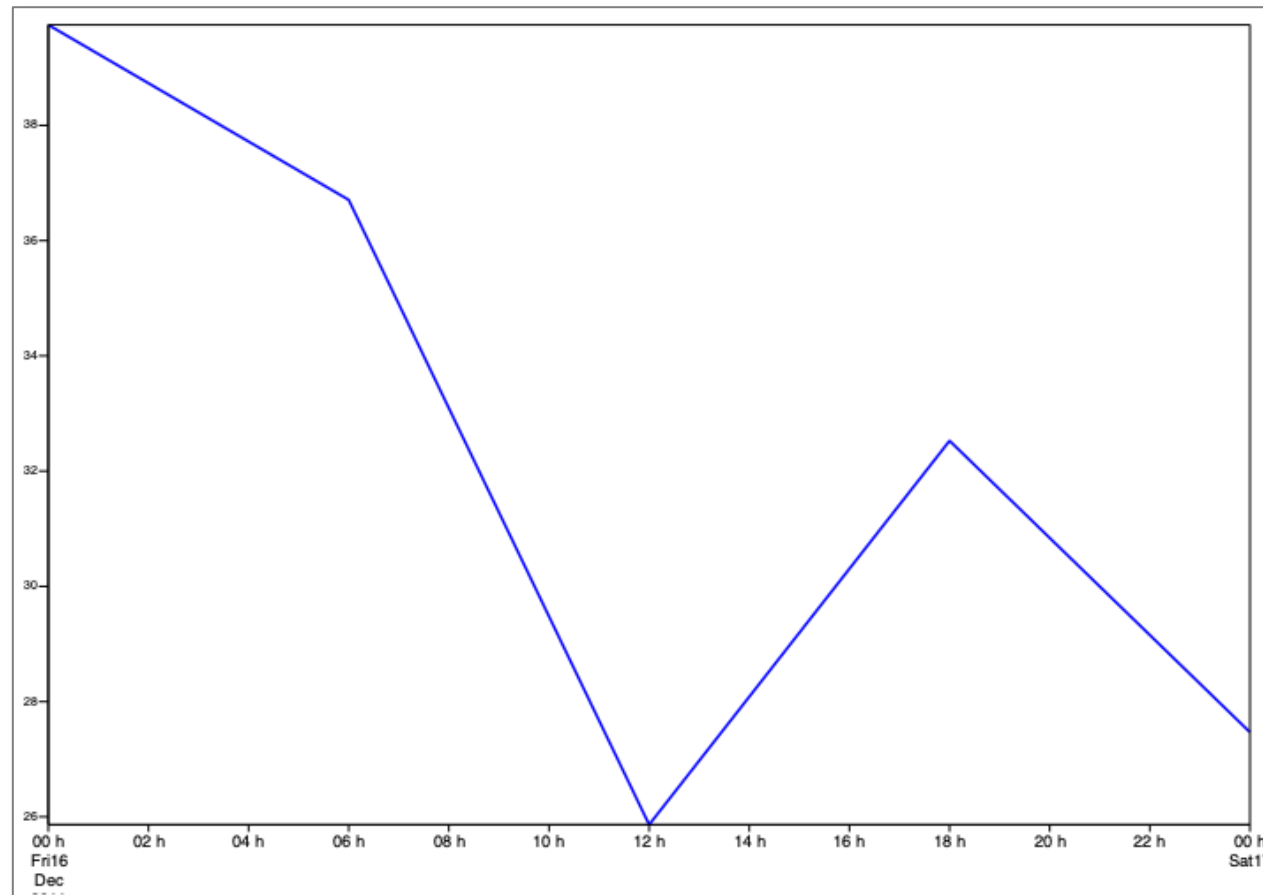
Step 6: Generate the plot

Tells Metview to generate a plot inside the **Jupyter** notebook



```
mv.setoutput('jupyter')
```

```
mv.plot(ts_view, curve_wg, visdef)
```



Let's look further up the atmosphere

Step 1: Getting the data



MARS Retrieval

uv - /webinar-may-2020 - Metview

Icon name: uv
Folder: /webinar-may-2020
Type: RETRIEVE Modified: 2020-05-10 11:25

Show disabled parameters

| | | | |
|-----------|----|--|---|
| Type | >> | FC | ✕ |
| Stream | >> | DA | ✕ |
| Expver | | 1 | ✕ |
| Repres | | Spherical Harmonics | ▼ |
| Reportype | | OFF | ✕ |
| Levtype | | Pressure Levels | ▼ |
| Levelist | | 1000/925/850/700/500/400/300/250/200/150/100 | ✕ |
| Param | >> | U/V | ✕ |
| Date | | 20111215 | ✕ |
| Defdate | | OFF | ▼ |

Templates ▶

Reset OK Cancel Save

Connects to a local MARS server if available

Otherwise uses the MARS Web API to get MARS data over an internet connection (registration required)

Step 1: Getting the data (in Python)



MARS Retrieval



```
uv = mv.retrieve(  
    type      = "fc",  
    levellist = [1000,925,850,700,500,400,300,250,200,150,100],  
    param     = ["u","v"],  
    date      = 20111215,  
    step      = 12,  
    area      = [25,-60,75,60],  
    grid      = [0.25,0.25]  
)
```

```
mv.grib_get(uv, ['shortName', 'level'])
```

```
[[ 'u', '1000'],  
 [ 'v', '1000'],  
 [ 'u', '925'],  
 [ 'v', '925'],  
 [ 'u', '850'],  
 [ 'v', '850'],  
 [ 'u', '700'],  
 [ 'v', '700'],  
 [ 'u', '500'],  
 [ 'v', '500'],  
 [ 'u', '400'],  
 [ 'v', '400'],  
 [ 'u', '300'],  
 [ 'v', '300'],  
 [ 'u', '250'],  
 [ 'v', '250']
```


Step 2: Compute the wind speed

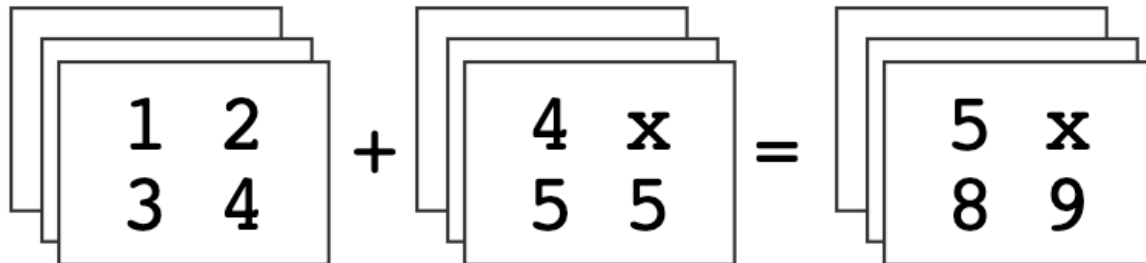


Grib Filter

In script it is called `read()`



Fieldset operations operate on each grid point in each field



Extract the U and V components into different Fieldsets

```
u = mv.read(data = uv, param = "u")  
v = mv.read(data = uv, param = "v")
```

Compute the wind speed directly on the Fieldsets

```
spd = mv.sqrt(u*u + v*v)
```

Change the paramId and extract 500hPa level for plotting

```
spd = mv.grib_set_long(spd, ['paramId', 10])  
spd500 = mv.read(data = spd, levelist = 500)
```

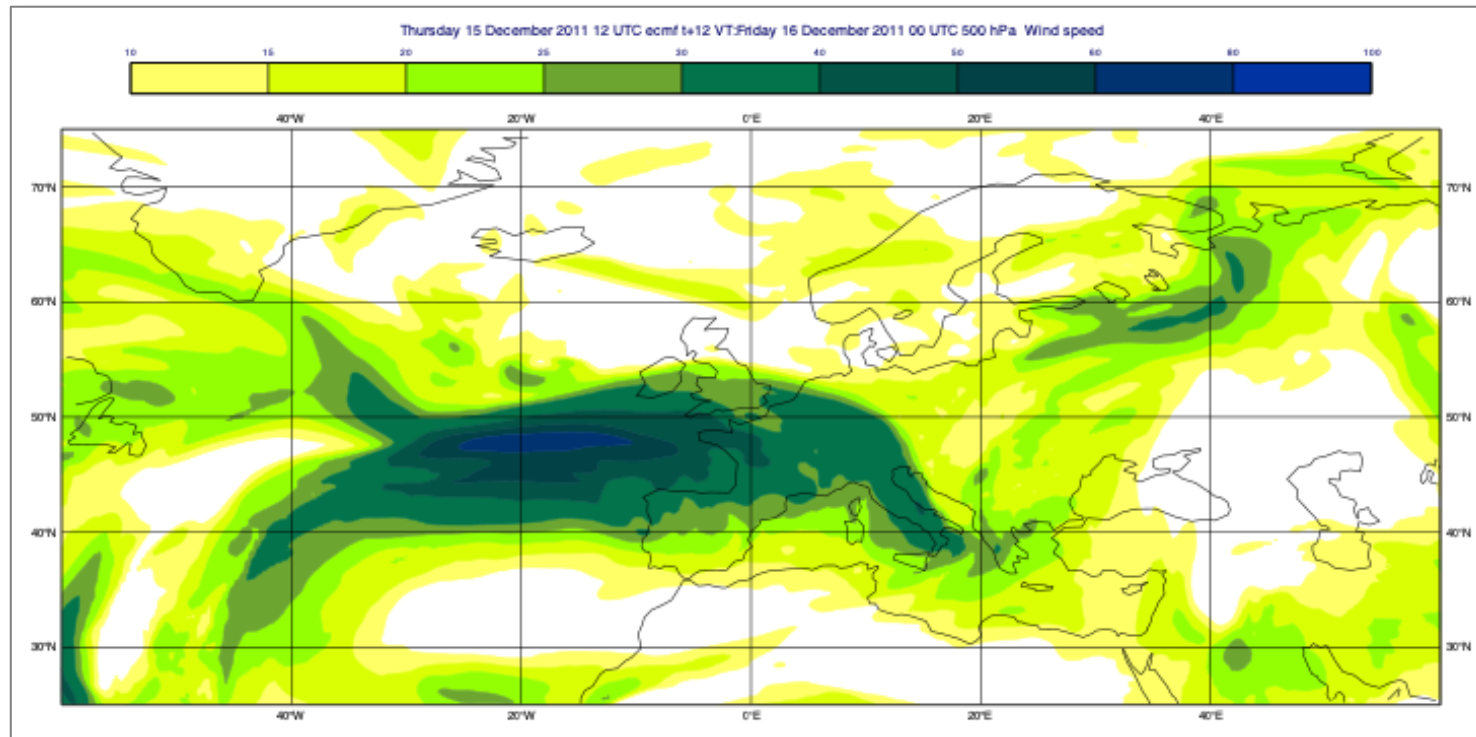
Step 3: Plot the wind speed on a map



Geographical
view



```
view = mv.geoview(  
    map_area_definition = "corners",  
    area                 = [25,-60,75,60]  
)  
mv.plot(view, spd500, mv.mcont(contour_automatic_setting='ecmwf', legend='on'))
```



Step 4: Define a vertical cross section view

Easiest way is to use the user interface to define the cross section view details, then drop the icon into the Python editor to generate the code



Cross Section View

The screenshot displays the Metview software interface. The top window, titled 'xs_joachim - /webinar-may-2020 - Metview', shows the configuration for a vertical cross section view. The 'Icon name' is 'xs_joachim', the 'Folder' is '/webinar-may-2020', and the 'Type' is 'MXSECTIONVIEW'. The 'Modified' date is '2020-05-13 15:24'. Below this, there are three main parameters:

- Bottom Level:** 1000.0
- Top Level:** 100
- Line:** 43.3/-36.0/54.4/13.1

The bottom window, titled 'Metview - Geography Tool', shows a map of the North Atlantic region with a vertical cross section line drawn across it. The map displays contour lines and a color scale for the 'U component' of the wind. The color scale ranges from 0.0662998 to 63.660559. The map axes are labeled with latitude (30°N to 70°N) and longitude (60°W to 60°E). The cross section line is defined by the coordinates 43.3/-36.0/54.4/13.1. The window also includes a 'Save' button and 'OK'/'Cancel' buttons.

Step 5: Plot the line on the map

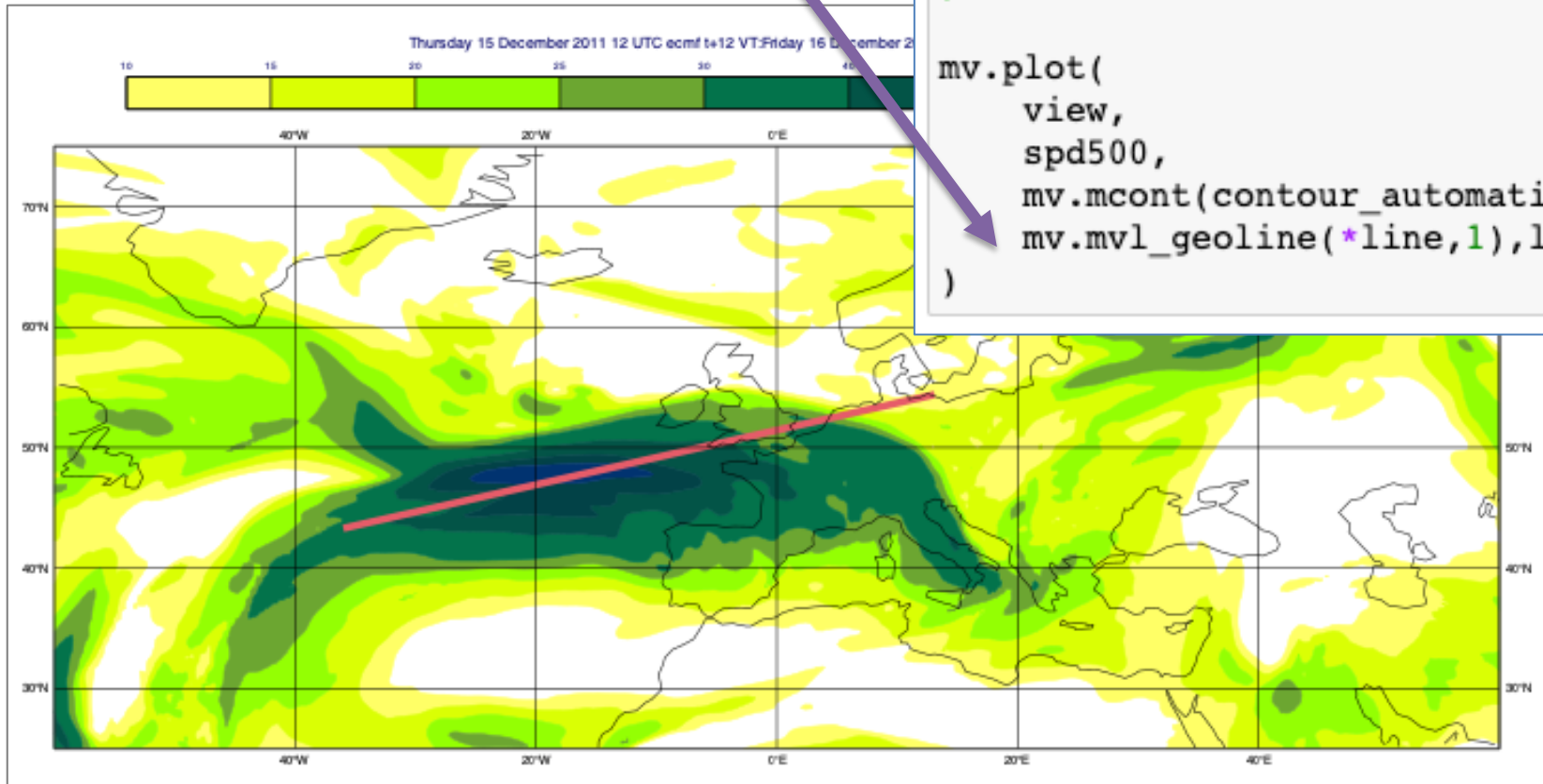
Overlay the line using
`mvl_geoline()` and
`mgraph()`

Define a line along an area of interest and plot it onto the map

```
line = [43.3, -36.0, 54.4, 13.1] # S, W, N, E
```

```
line_graph = mv.mgraph(  
    graph_type      = "curve",  
    graph_line_colour = "pink",  
    graph_line_thickness = 5  
)
```

```
mv.plot(  
    view,  
    spd500,  
    mv.mcont(contour_automatic_setting='ecmwf', legend='on'),  
    mv.mvl_geoline(*line,1),line_graph  
)
```



Step 6: Define the cross section view and contouring



Cross Section View



Contouring



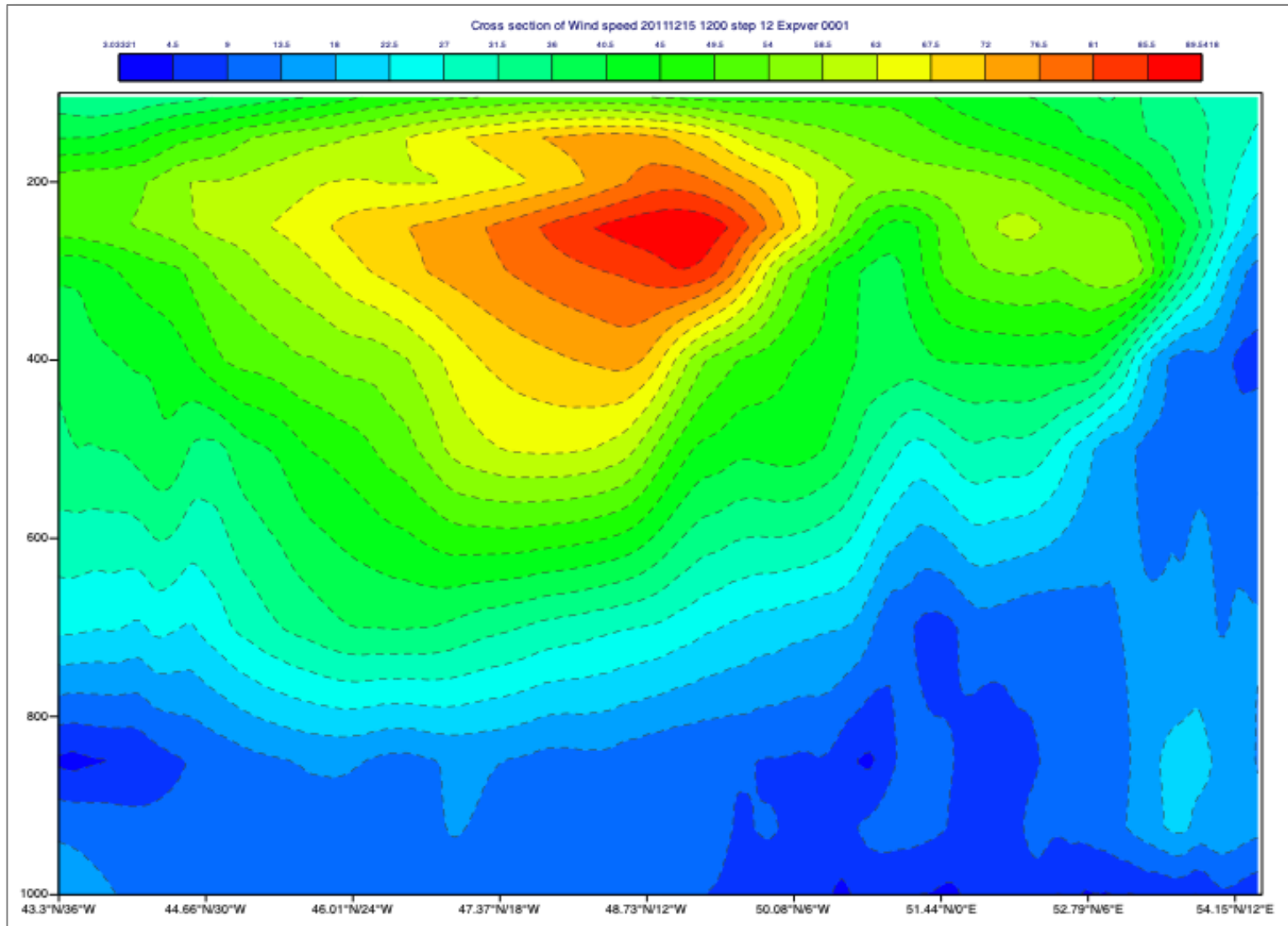
Define a cross section view along an area of interest

```
xs_view = mv.mxsectview(  
    bottom_level = 1000.0,  
    top_level    = 100,  
    line        = line  
)
```

Create a colour scale to plot the data with

```
xs_shade = mv.mcont(  
    legend                = "on",  
    contour_line_style    = "dash",  
    contour_line_colour    = "charcoal",  
    contour_highlight     = "off",  
    contour_level_count   = 20,  
    contour_label         = "off",  
    contour_shade         = "on",  
    contour_shade_method  = "area_fill",  
    contour_shade_max_level_colour = "red",  
    contour_shade_min_level_colour = "blue",  
    contour_shade_colour_direction = "clockwise"  
)
```

Step 7: Generate the plot



```
mv.plot(xs_view, spd, xs_shade)
```

Note: we can also obtain the raw cross section data as NetCDF:

```
xs_data = mv.mcross_sect(  
    data = spd,  
    line = line  
)  
print(xs_data)
```

```
<metview.bindings.NetCDF object at 0x10e5b8ed0>
```

Vertical profiles are very similar – nice UI to define



Vertical Profile View

The screenshot displays the Metview software interface. The main window is titled 'Vertical Profile View - /webinar-may-2020 - Metview'. It shows the following configuration details:

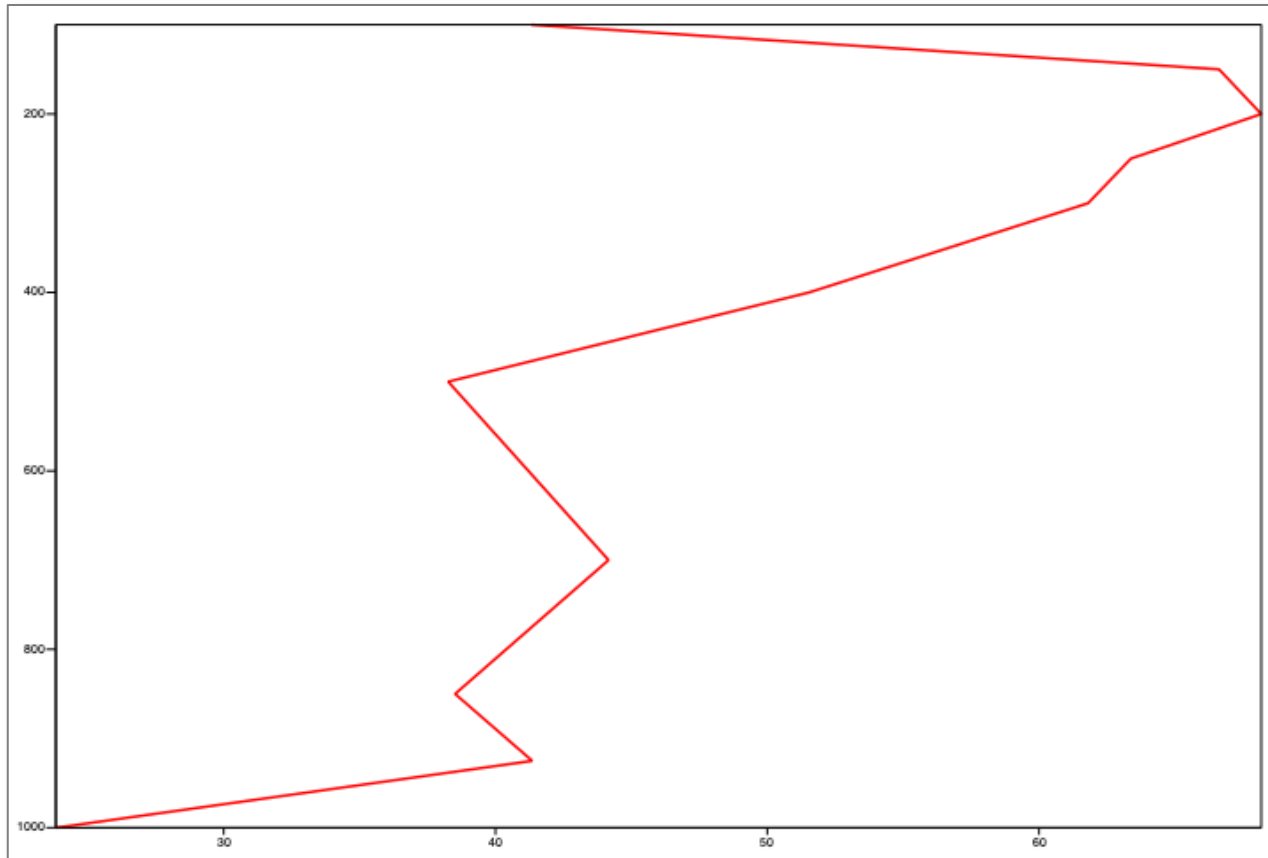
- Icon name: Vertical Profile View
- Folder: /webinar-may-2020
- Type: MVPROFILEVIEW Modified: 2020-05-13 16:11

Below these details is a section for 'Show disabled parameters'. The main configuration area includes:

- Input Mode: Point
- Point: 47.0/-3.5
- Bottom Level: 1000
- Top Level: 100

At the bottom of the configuration window are 'OK', 'Cancel', and 'Save' buttons. In the foreground, the 'Metview - Geography Tool' window is open, showing a map of Europe and the surrounding region. The map has a coordinate grid with latitude lines from 30°N to 60°N and longitude lines from 20°W to 0°E. A blue circle on the map indicates the location of the profile, centered at approximately 47.0°N, 3.5°W. Below the map, the coordinates are displayed as 'S/W/N/E: 47.0/-3.5'. The 'Geography Tool' window also has 'OK' and 'Cancel' buttons.

Translate into Python and generate the plot



Define a vertical profile view and plotting attributes

```
vp_view = mv.mvertprofview(  
    point          = [47.0, -3.5],  
    bottom_level   = 1000,  
    top_level      = 100  
)  
  
graph_plotting = mv.mgraph(  
    graph_line_colour = "red",  
    graph_line_thickness = 3  
)
```

```
mv.plot(vp_view, spd, graph_plotting)
```


And finally...

Resources

- All material from this webinar will be made available here:
- <https://confluence.ecmwf.int/metview/Webinars>

Using Fieldset operations to preserve only the extreme values

Alternative way to obtain the largest values - use **Fieldset** operations:

In [18]:

```
# compute field of 1s and 0s according to test
largest_mask = wg0 > (max0*0.85)

# convert 0s into missing values
largest_mask = mv.bitmap(largest_mask, 0)

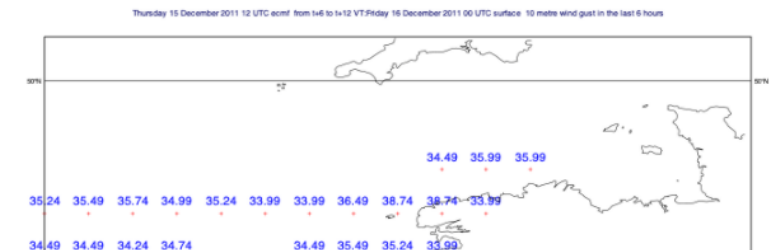
# copy the pattern of missing values to wg0
masked_wg0 = mv.bitmap(wg0, largest_mask)
```

The result has **missing values** where the original values were below our threshold. In terms of actual data storage, if we were to write much smaller than the original GRIB file because GRIB is very efficient at (not) storing missing values. Let's plot the result with grid p

In [19]:

```
gridvals_1x1 = mv.mcont(
    contour = "off",
    contour_grid_value_plot = "on",
    contour_grid_value_plot_type = "both",
    contour_grid_value_format = "(F4.2)",
    contour_grid_value_height = 0.45,
    grib_scaling_of_retrieved_fields = "off"
)

mv.plot(view, masked_wg0, gridvals_1x1)
```



Where to find out more

Icon reference

ECMWF Spaces

User Guide

- Using Metview
- The Macro Language
- Metview's Python Interface
- Icon Reference**
 - Annotation View
 - Average Data
 - Average View
 - Axis Plotting
 - Binning
 - Bufr Picker
 - Cartesian View
 - Clean File
 - Coastlines
 - Common View Paramete
 - Contouring
 - Cross Section Data
 - Cross Section View
 - Display Window
 - Download from URL
 - ECCHARTS
 - ECFS

Space tools

Data access icons

| | | | | | | | |
|-------------------|------------|------|------------------|-----------------|----------------|---------------|----------|
| | | | | | | | |
| Download from URL | ECCHARTS | ECFS | FLEXPART Prepare | FLEXTRA Prepare | MARS Retrieval | Met3D Prepare | Stations |
| | | | | | | | |
| VAPOR Prepare | WMS Client | | | | | | |

Data filter icons

| | | | | | |
|-------------|------------|-------------|------------|--------------------|--------------|
| | | | | | |
| Bufr Picker | Clean File | GRIB Filter | ODB Filter | Observation Filter | Table Reader |

Data processing icons

| | | | | | | | |
|------------------|--------------------|------------------|-----------------------|-------------------|--------------|------------------------------|-------------------|
| | | | | | | | |
| Average Data | Cross Section Data | FLEXPART Release | FLEXPART Run | FLEXTRA Run | Formula | Grib To Geopoints | Geopoints To Grib |
| | | | | | | | |
| Geopoints To KML | Hovmoeller Data | Percentile | Potential Temperature | Relative Humidity | Reprojection | Rotational or Divergent Wind | RTTOV Run |

Where to find out more

Function documentation

The screenshot shows the ECMWF documentation interface. On the left is a navigation sidebar with categories like 'User Guide', 'The Macro Language', and 'List of Operators and Functions'. The main content area displays the documentation for the 'geostrophic_wind_pl' function, including its signature, description, mathematical formulas for wind components, and a list of parameters. The top navigation bar includes 'Spaces', 'Calendars', and 'Create' buttons, along with a search bar and user profile icons.

fieldset `geostrophic_wind_pl` (z: fieldset)

Computes the geostrophic wind from geopotential fields defined on pressure levels. For a given z geopotential field the computation of the geostrophic wind components is based on the following formulas:

$$u_g = -\frac{1}{f} \frac{1}{R} \frac{\partial z}{\partial \phi}$$
$$v_g = \frac{1}{f} \frac{1}{R \cos \phi} \frac{\partial z}{\partial \lambda}$$

where

- R is the radius of the Earth
- ϕ is the latitude
- λ is the longitude
- $f=2\Omega \sin \phi$ is the Coriolis parameter, where Ω is the Earth's angular velocity.

The derivatives are computed with a second order finite-difference approximation. The resulting fieldset contains two fields for each input field: the u and v geostrophic wind components. In each output field the points close to the poles and the Equator are bitmapped (they contain missing values). Please note that this function is only implemented for regular latitude-longitude grids.

geopoints `gfind` (fieldset,number)
geopoints `gfind` (fieldset,number,number)

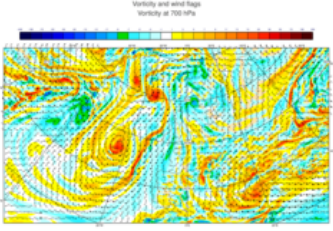
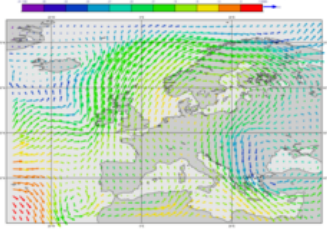
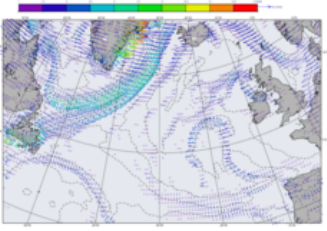
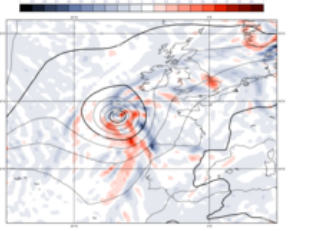
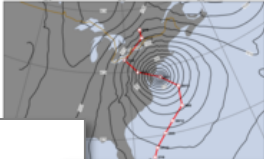
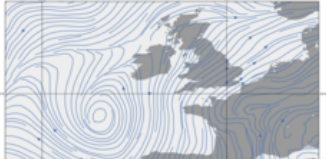
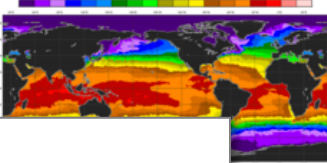
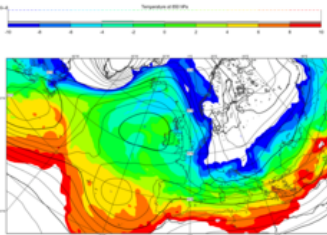
A filtering function that returns a geopoints holding the grid points whose value is equal to the *value* of the first number. Missing values in the input field are not returned. If a second number is given as the third argument it is a tolerance *threshold* and the geopoints will hold the grid points for which :

$$\text{abs}(\text{data-value}) \leq \text{threshold}$$

Where to find out more

ECMWF Spaces

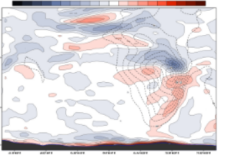
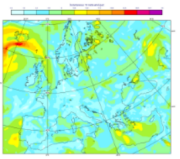
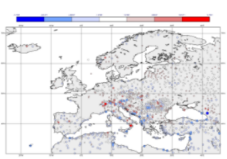
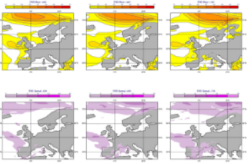
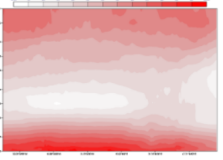
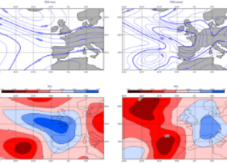
- Change History
- User Guide
- FAQ
- Training
- Gallery
 - Python Jupyter Notebooks
 - OpenIFS Workshop 2016
 - 2m Temperature Plot Exam
 - Aircraft Observation Exam
 - Bar Plotting Example
 - Boundaries, Cities and Rive
 - BUFR Synop Example
 - Contour Shading and Posit
 - Cross Section Example
 - Cross Section with Orograp

| | | | |
|--|---|--|---|
| <p>Vorticity and Wind Example GRIB, Polar Stereographic</p>  | <p>Wind Coloured By Temperature Example GRIB, Cylindrical</p>  | <p>Temperature Gradient Vector Example GRIB, Polar Stereographic</p>  | <p>Humidity advection Example GRIB, Cylindrical</p>  |
| <p>Storm track Example GRIB, CSV, Polar Stereographic</p>  | <p>Streamlines Example GRIB, Cylindrical</p>  | <p>SST on Extended Cylindrical Map Example GRIB, Cylindrical</p>  | <p>Histogram Legend Example GRIB, Polar Stereographic</p>  |

Pages / Metview / Gallery

Python Jupyter Notebooks

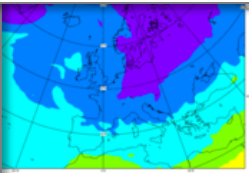
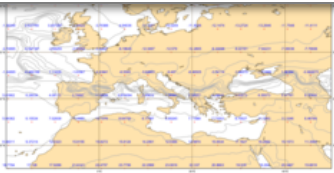
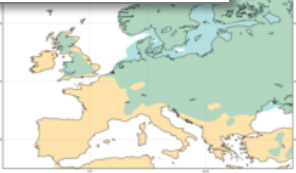

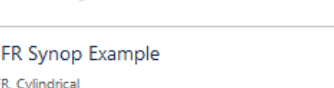
Created by Milana Vuckovic, last modified by Iain Russell on Feb 06, 2019

| | | |
|---|--|--|
| <p>Vertical cross section - Wind shear</p>  | <p>NetCDF from CDS</p>  | <p>Forecast/Observations difference</p>  |
| <p>Using Xarray for computing</p>  | <p>Vertical cross section - CDS data</p>  | <p>Principal component analysis</p>  |

Try the example notebooks on Binder!

Click the link below to start a Binder session to try the examples online now:

[launch binder](#)

| | | | |
|---|---|---|--|
| <p>Contouring 1 Example</p>  | <p>Split Contouring Example GRIB, Cylindrical</p>  | <p>Model-Obs Difference Example GRIB, BUFR, Cylindrical</p>  | <p>Boundaries, Cities and Rivers Example Cylindrical</p>  |
| <p>BUFR Synop Example BUFR, Cylindrical</p>  | | | |

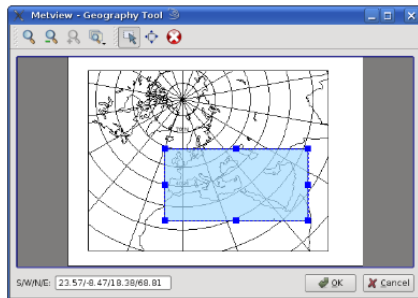
Where to find out more

Lots of material online including tutorials

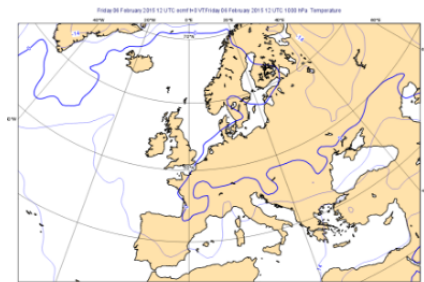
Now we want to set the area used in the view. Although we can interactively zoom into smaller areas in the **Display Window**, we can use exactly the same one again and again. Set the **Map Area Definition** to **Corners** and click on the **Geography Tool** button.



This tool helps you define a region.

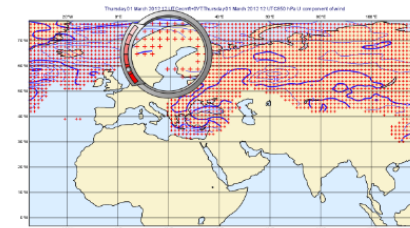


Use the **Zoom** tools to enlarge the European area and use the **Area** tool to select a region over Europe. Click **Ok** to save the **Geographical View** editor. Click **Apply** in the **Geographical View** editor to save everything. Plot your data in this view to compare.



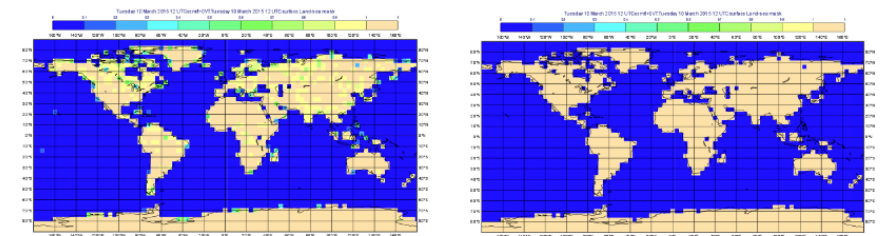
- A Quick Tour of Metview
- Data analysis and visualisation using Metview
- A Simple Visualisation
- Customising Your Plot
- Case Study: Plotting Hurricane S...
- Data Part 1
- Processing Data
- Analysis Views
- Layout in Metview
- Case Study: Cross Section of Sa...
- Data Part 2
- Handling Time in Metview
- Graph Plotting in Metview
- Case study: Plotting the Track o...
- Working with graphical output
- Organising Macros
- **Missing Values and Masks**
- Optimising Your Workflow
- Customising Your Plot Title
- Case study: Ensemble Forecast
- Running Metview in Batch Mode
- Working with Folders and Icons
- Exploring Metview

Overview



Fields and observations can often contain missing values - it can be important to understand the implications of the points. Using a mask of missing values can enable Metview to perform computations on a specific subset of points.

Computing the mean surface temperature over land



As an example, we will use a land-sea mask field as the basis of performing a computation on only the land points, e.g. the mean surface temperature. Visualise the supplied `land_sea_mask.grib` icon using the `grid_shade` icon. This `Contouring` icon is set up to shade the interpolation. To help illustrate what's going on, we've chosen low-resolution fields - this one is 4x4 degrees. The values between 0 and 1 on points which are close to both sea and land. Before we can use this field as a mask, we must do something to indicate whether they count as land or sea! Let's say that a value of 0.5 or more is land.

Metview availability – on ECMWF systems

- Versioned using the 'module' system

Interactive session

```
module load python3
module swap metview/new
metview
```

Batch, Jupyter notebook

```
module load python3
module swap metview/new
jupyter notebook <path>
```

Metview availability – outside ECMWF

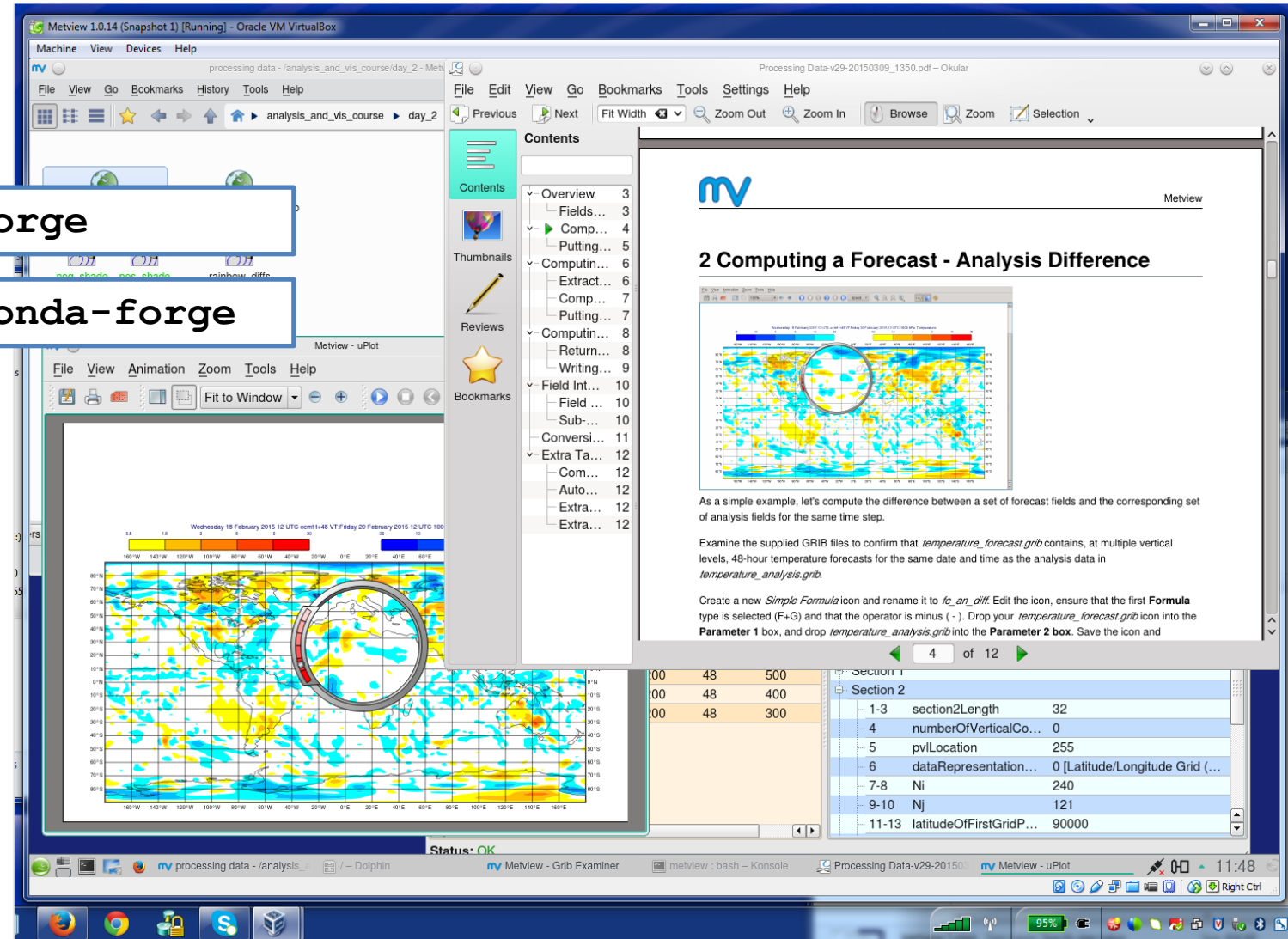
- Install from binaries
- Conda (via conda-forge)

```
conda install metview -c conda-forge
```

```
conda install metview-batch -c conda-forge
```

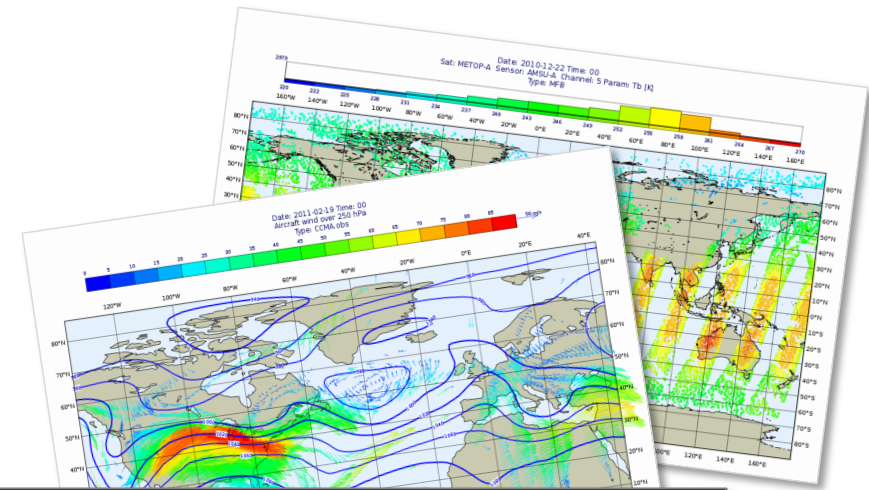
- Build from source
- Build from bundle
- The Metview Python interface has to be installed separately:

```
pip install metview
```



For more information...

- Ask for help:
 - Software.Support@ecmwf.int
- Visit our web pages:
 - <http://confluence.ecmwf.int/metview>



Questions?

Computing a Forecast - Observation Difference

This time we'll compare two very different data types: gridded forecast data in a GRIB file, with scattered observation data described in a BUFR file. We will use the `t2m_forecast.grib` icon (the gridded forecast data), and the observation data is in a BUFR file represented by the `obs.bufr` icon and contains observations over Europe, valid at the same time as the GRIB data. Examine and visualise both icons to confirm what they contain.

Extracting the 2 metre temperature

The first step to comparing GRIB data with BUFR data is to extract just the parameter we want from the BUFR data and convert it to the `geopoints` format. Then the computation will be simple.

Create a new *Observation Filter* icon and rename it to `filter_obs_t2m`, setting these parameters:

| | |
|------------------|--|
| Data | Drop the <code>obs.bufr</code> icon here |
| Output | Geographical Points |
| Parameter | 012004 |

Note that 012004 is the code for 'Dry bulb temperature at 2m'. Confirm that the result of this icon's filtering is a set of geopoints with temperature values.