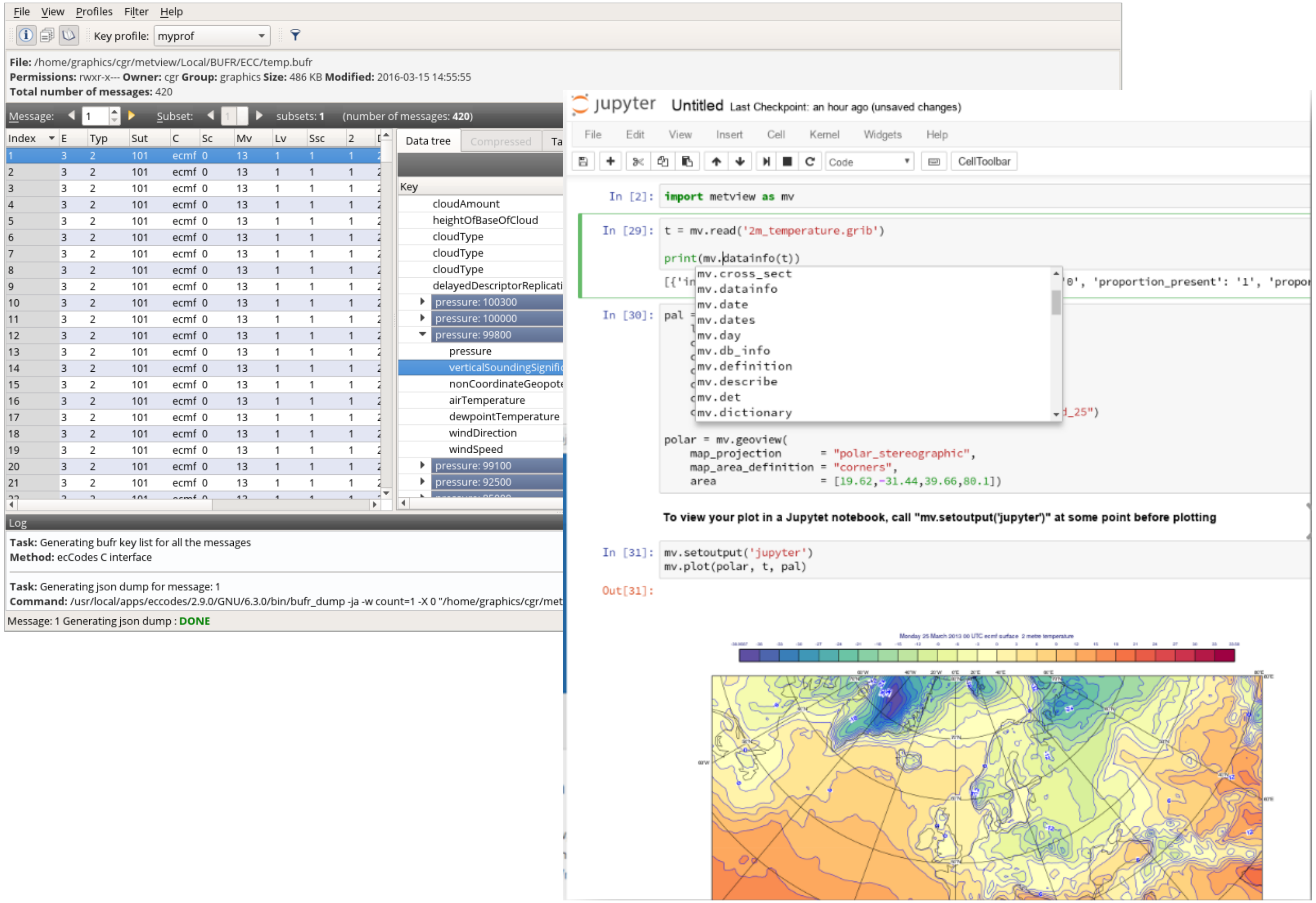
Post Processing of ECMWF Data

Webinar - May 14, 2019

Sándor Kertész
Iain Russell

Development Section, ECMWF



ECMWF

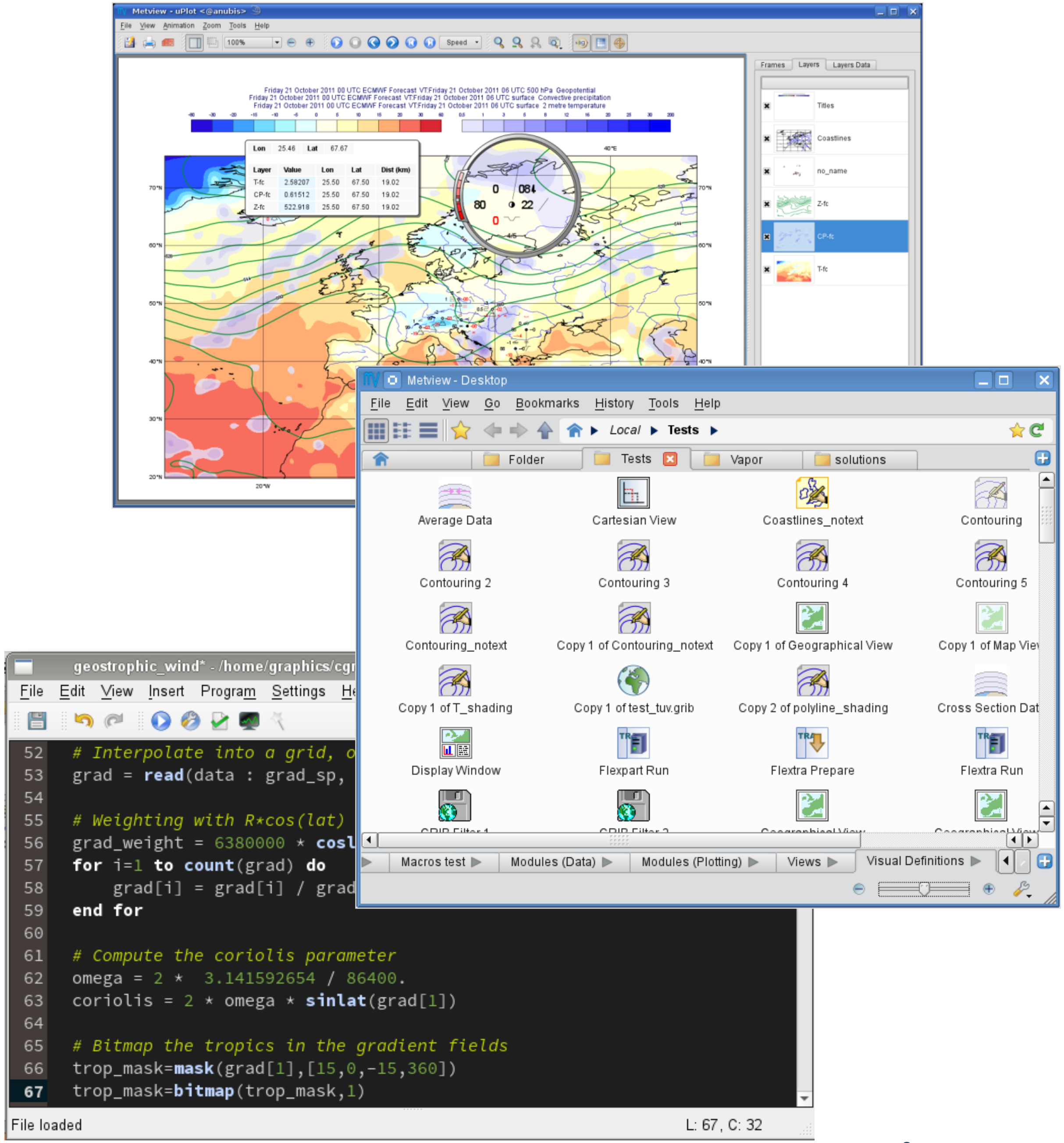
Outline

The webinar will be centred around Metview’s Python interface

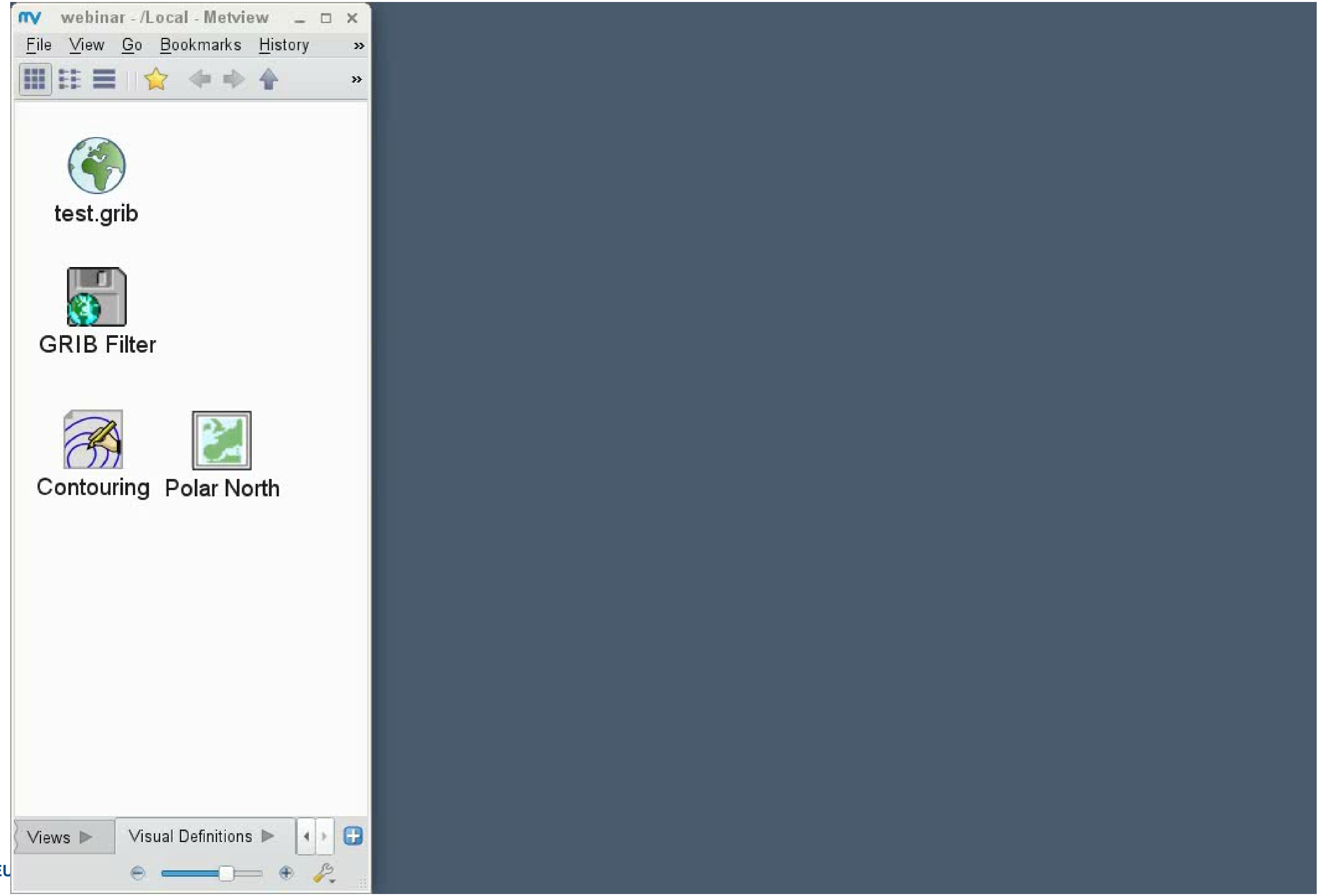
- Metview user interface: quick overview
- GRIB:
  - MARS, interpolation, filtering, value extraction
  - masking and bitmaps
  - computation types, profile and section generation
  - numpy, scipy, cfgrib, xarray
- NetCDF
- Observation handling: BUFR, ODB, Geopoints
  - pandas
- Climate Data Store
- Where to find out more

What is Metview?

- Workstation software, runs on UNIX, from laptops to supercomputers (including Mac OS X)
- Open source, Apache 2.0 license
- Visualisation
- [Data processing]
- Icon based user interface
- Powerful scripting languages (Macro and [Python (3)])
- Co-operation project between ECMWF and INPE (Brazil)



Metview's user interface

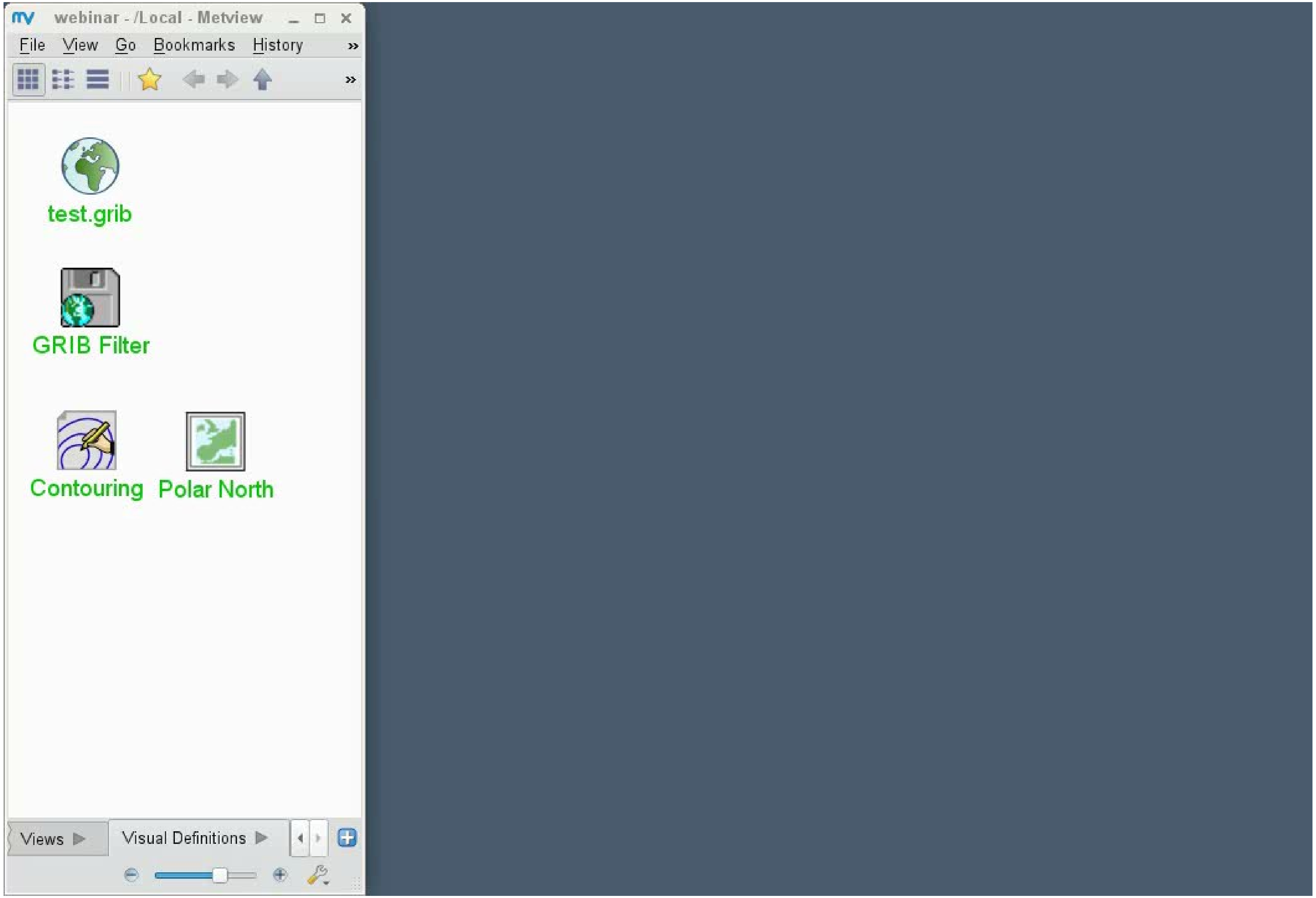


ECMWF

EU

Generating scripts

- All icons can be dropped into Metview’s code editor to generate code



Metview's script interface

- Everything that can be done interactively with icons can be done via scripting
- Scripting offers a lot of extra functionality especially for **data processing**

We will only use the script interface (Python) in the webinar

MARS access

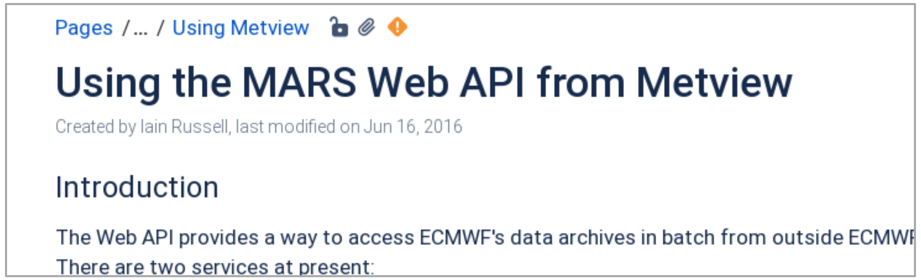
- ECMWF's Meteorological Archive (GRIB, BUFR, ODB)
- Integrated MARS client in Metview

At ECMWF
(e.g. ecgate)

direct access to MARS from Metview

Outside the
Centre

access through the **MARS WEB API**


Pages / ... / Using Metview

Using the MARS Web API from Metview

Created by Iain Russell, last modified on Jun 16, 2016

Introduction

The Web API provides a way to access ECMWF's data archives in batch from outside ECMWF
There are two services at present:

A typical MARS-GRIB workflow

Mars Retrieval

Grib Filter

In script it is called **read()**

Field arithmetic

Operations are performed on each gridpoint per field

Jupyter Notebook

```
import metview as mv
```

Retrieves data from MARS on a reduced Gaussian grid.

```
g = mv.retrieve(type='fc',
                param = ['t','z','u','v'],
                levType= 'pl', levelist = ['500', '1000'],
                step = [0, 6, 12, 18],
                grid = 'N48')
```

Filters u and v wind components on 500 hPa.

```
u = mv.read(data=g, param='u', levelist='500')
v = mv.read(data=g, param='v', levelist='500')
```

Computes the wind speed fields.

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

Saves the fieldset into a GRIB file.

```
mv.write('result.grib', sp)
```

Fieldsets

Metview's object to represent GRIB data

grib_get()

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

Fieldsets behave like **lists**. So we can get their **size**:

```
len(g)
```

```
32
```

Indexing and **slicing** works:

```
f = g[0]
f = g[0:10:2]
f[0] = f[0] - 273.16
```

We can work with them in **loops**:

```
f = mv.Fieldset()
for v in g:
    if mv.average(v) > 1000:
        f.append(v)

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

```
[['z', '500', '0'], ['z', '500', '6'], ['z', '500', '12'], ['z', '500', '1
8']]
```

Interpolation: re-gridding

Grib Filter

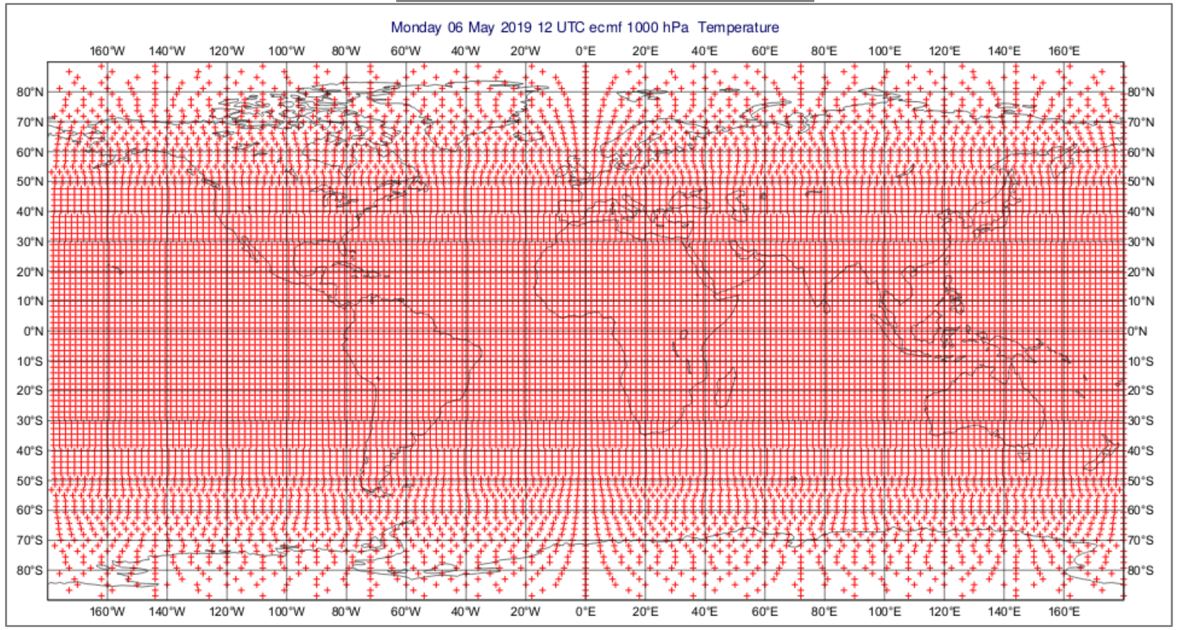
In script it is called **read()**

Interpolates the fieldset to a 5x5 degree regular latlon grid on a subarea.

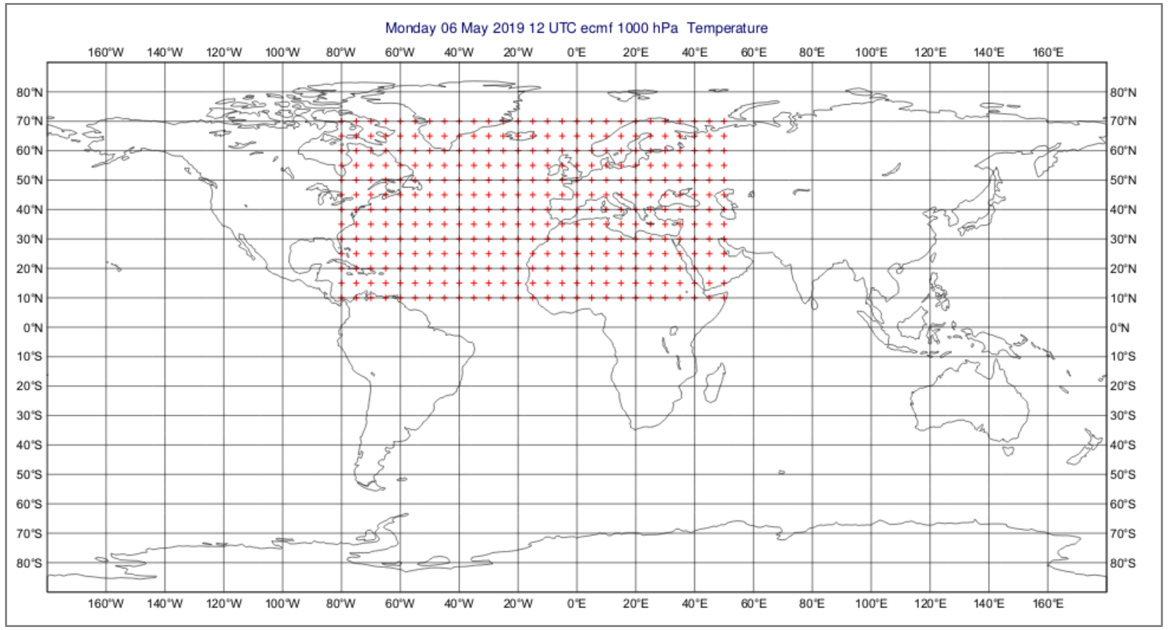
```
f = mv.read(data=g,
            grid=[5,5],
            area=[10,-80,70,50]) # S,W,N,E
```

Plots the original and new gridpoints.

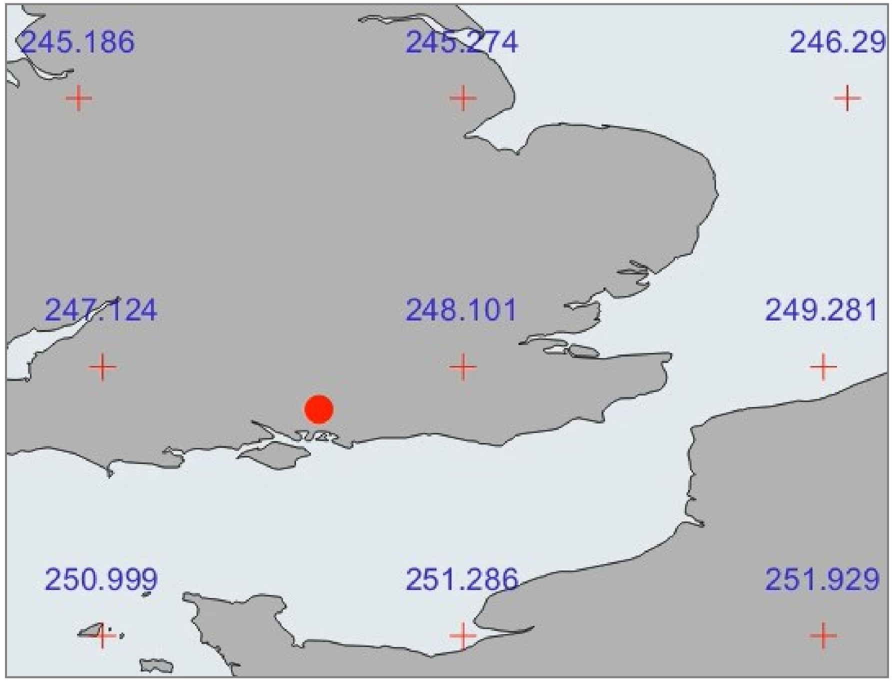
N48 grid



5x5 grid on subarea



Interpolation: extracting values at scattered locations



Interpolates fields to **scattered** locations using bilinear interpolation.

```
loc = [51, -1] #lat,lon
mv.interpolate(g[0], loc)
```

```
248.2566223473454
```

Getting the **nearest gridpoint** value.

```
mv.nearest_gridpoint(g[0], loc)
```

```
248.10105895996094
```

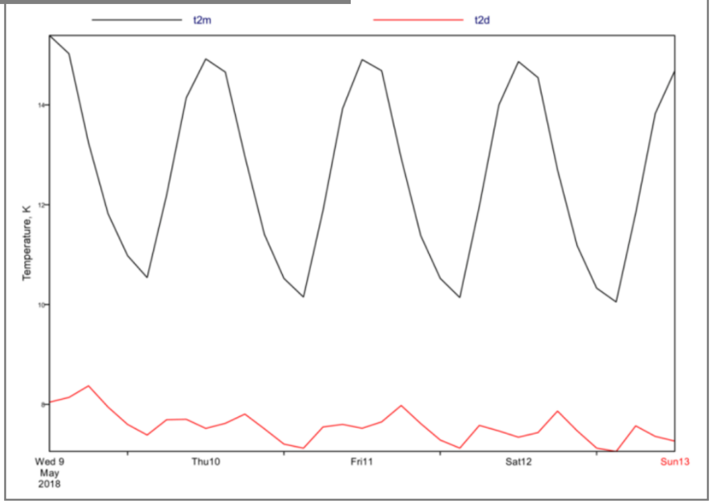
We can ask for all the **details** about the actual nearest gridpoint not just its value.

```
mv.nearest_gridpoint_info(g[0], loc)
```

```
[{'distance': 77.0006,
  'index': 1580.0,
  'latitude': 51.2944,
  'longitude': 0.0,
  'value': 248.101}]
```

Time series extraction

See “Time Series with GRIB” in Gallery for a more elaborate example



Extracts temperature on 1000 hPa.

```
t = mv.read(data=g, param='t', level=1000)
```

Gets **nearest gridpoint** from each field for the selected location as a list.

```
t_val = mv.nearest_gridpoint(t, [51, -1])
```

Gets **valid date** from the GRIB headers (it is computed from multiple keys) as a list of Python **datetime** objects.

```
d_val = mv.valid_date(t)
```

Prints the dates and values together.

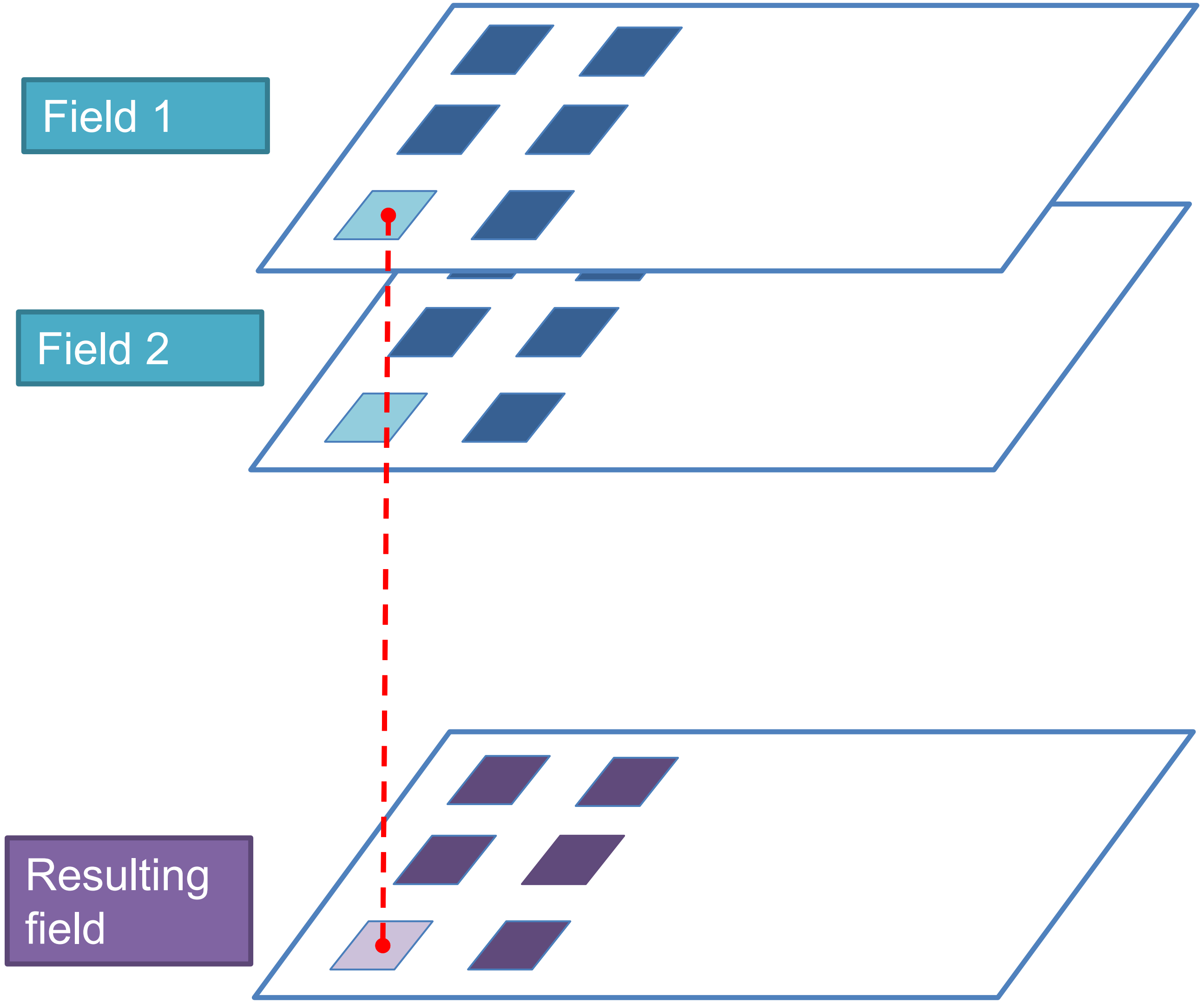
```
for dd,vv in zip(d_val, t_val):
    print('{} h --> {} K'.format(dd,vv))
```

```
2019-05-02 12:00:00 h --> 285.30157470703125 K
2019-05-02 18:00:00 h --> 283.7098083496094 K
2019-05-03 00:00:00 h --> 281.2944030761719 K
2019-05-03 06:00:00 h --> 280.19874572753906 K
```

Point-wise aggregation

- The computations are performed on each gridpoint individually throughout the fields
- The result is always one field
- Functions in this group:
  - **sum()**
  - **mean()**
  - **stdev()**
  - **min()**
  - **max()**

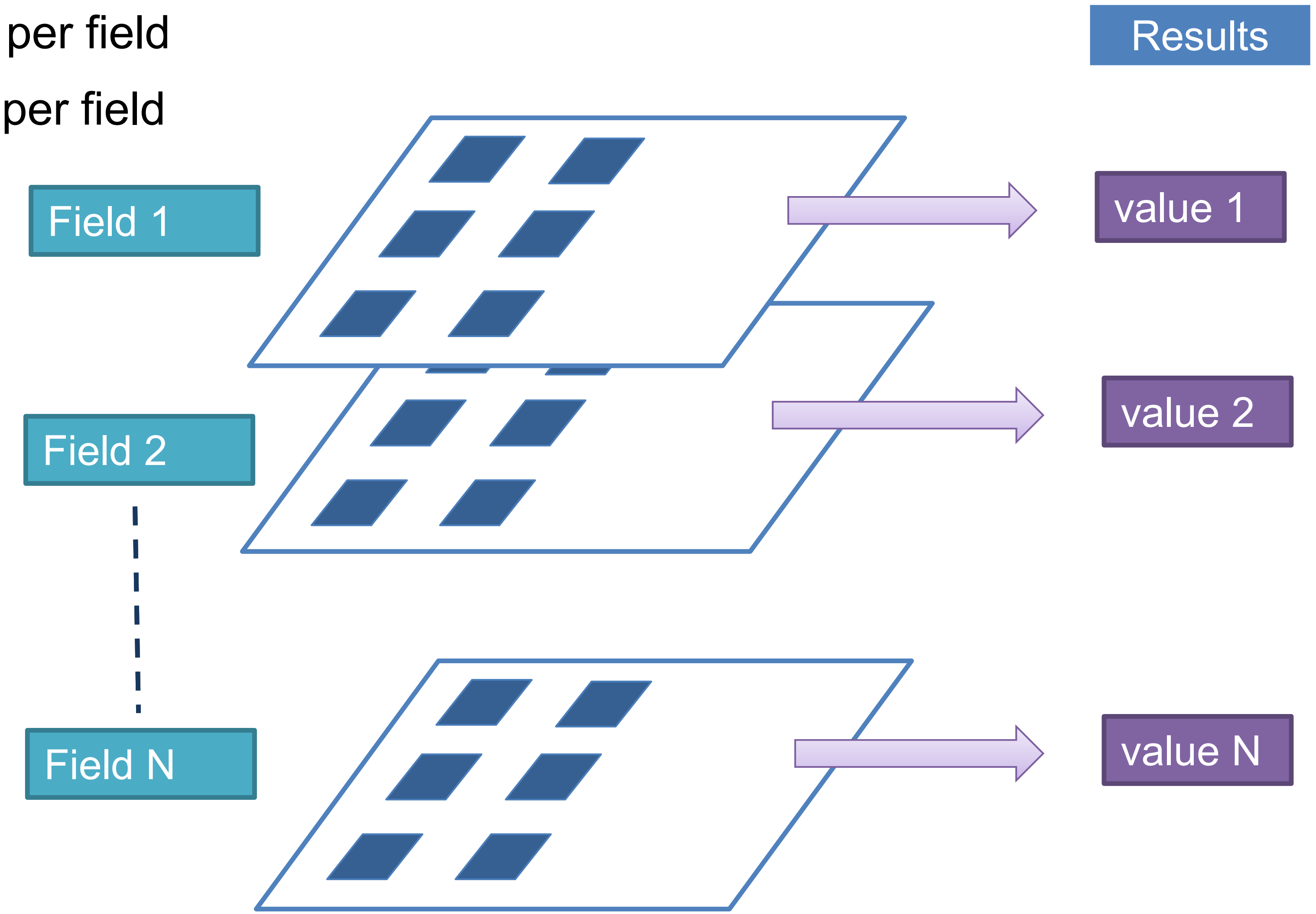
  and many more …



Field-wise aggregation

- The computations are performed per field
- The result is always one number per field
- Functions in this group:
  - **accumulate()**
  - **average()**
  - **integrate()**
  - **covar_a()**
  - **stdev_a()**

  and many more …



Weighting by grid cell area

- In the grids we typically work with the grid cell area changes from the Equator towards the Poles
- To correctly carry out horizontal computations we should apply a proper weighting
- Many functions perform this weighting, but some do not
- E.g. **averaging over an area**. Metview offers two functions for it:

average()

computes the **mathematical** average of all the values in a given field

$$average = \frac{1}{N}\sum_{i}^{N} f_i$$

integrate()

computes a **weighted average** to take into account the grid cell sizes

$$average = \frac{\sum_i f_i A_i}{\sum_i A_i} = \frac{\sum_i f_i cos\phi_i \Delta\lambda_i}{\sum_i cos\phi_i \Delta\lambda_i}$$

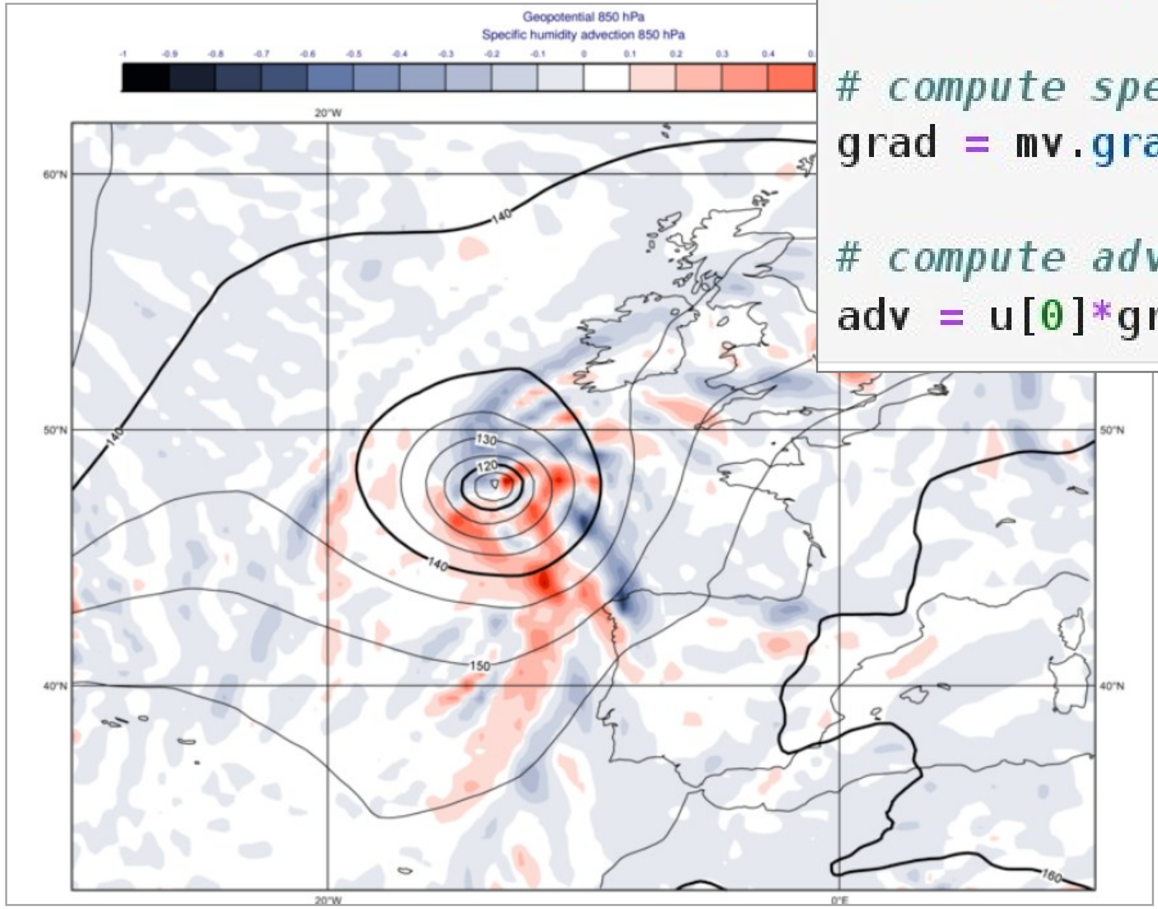
Horizontal derivatives

See “Humidity advection” example from the Gallery

Second order finite difference scheme

- **first and second derivatives**
- **gradient()**
- **vorticity()**
- **divergence()**
- **laplacian()**

…




```
# extract fields
q = mv.read(data = f, param = "q")
u = mv.read(data = f, param = "u")
v = mv.read(data = f, param = "v")

# compute specific humidity gradient
grad = mv.gradient(q[0])

# compute advection
adv = u[0]*grad[0] + v[0]*grad[1]
```

Vertical computations

Model levels ↔ Pressure levels

- computes geopotential on model levels: **mvl_geopotential_to_ml()**:
  - Geopotential is not archived on model levels in MARS!
- computes pressure on model levels: **unipressure()**
- interpolates model levels to pressure levels **mvl_ml2hpa()**

Vertical integration

univertint(): performs vertical integration using the following formula

The function computes:

$$\int_{bottom}^{top} f \frac{dp}{g}$$

where

- f is the fieldset
- p is the pressure
- g is the acceleration of gravity (9.81 m/s^2).

Masking and bitmaps – the concept

Masking

Logical operation on a field turning values into 0s and 1s

t = t > 273.16

In the resulting field all points with values > 273.16 will be 1s, while all other points will be 0s

Bitmaps

The set of points with **missing value** in a field are called a **bitmap**

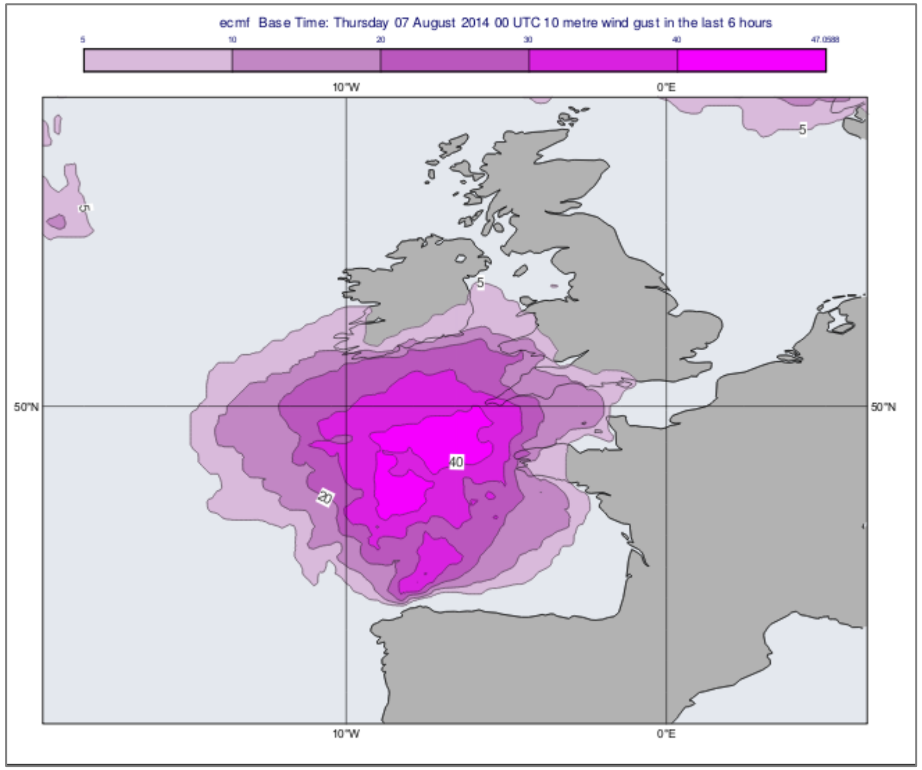
The bitmap in a field is excluded from computations and visualisation

E.g. wave forecast fields, SST

They are typically used together

Masking: computing ENS probability

We want to compute the probability that the windgust is > 20 m/s from a 51-member ENS forecast



Reads windgust ensemble forecast data.

```
wg = mv.read(source = 'fc_ens.grib', step='78')
len(wg)
```

```
51
```

Creates **mask** for values > 20 m/s.

```
wg_mask = wg > 20
len(wg_mask)
```

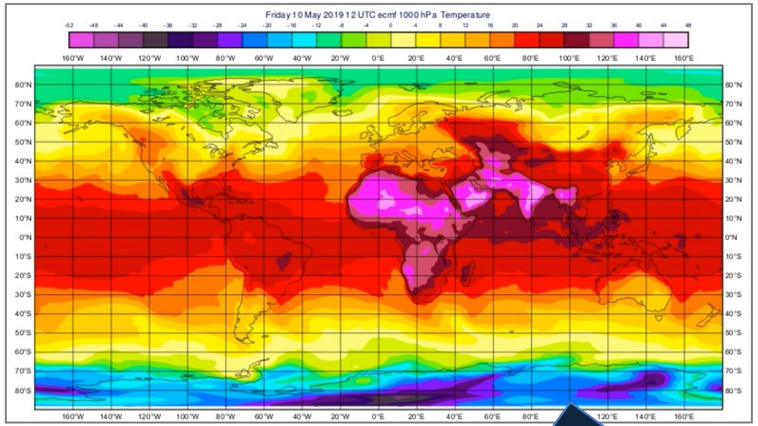
```
51
```

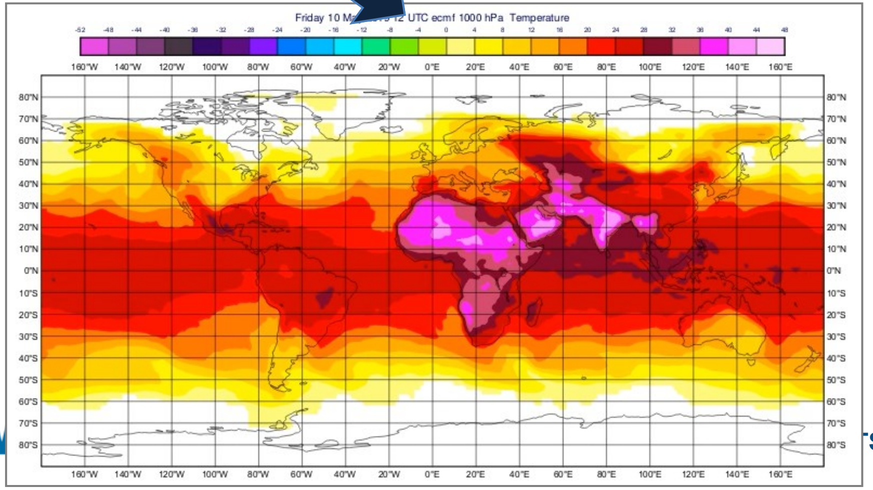
Computes **probability**.

```
prob = mv.mean(wg_mask) * 100
```

Masking and bitmaps

We want to create a fieldset with valid values only where T > 0C





Reads temperature at 1000 hPa.

```
t = mv.read(data=g, param='t', levelist='1000')
```

Creates **mask** for positive values.

```
t_pos_mask = t > 273.16
```

Turns **zeros** into **missing values** in mask.

```
t_pos_bm = mv.bitmap(t_pos_mask, 0)
```

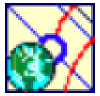
Applies **bitmap** to the temperature field.

```
t_pos = mv.bitmap(t, t_pos_bm)
```

Thermodynamic profiles

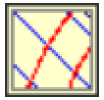
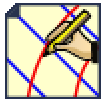
See example "Parcel path" from the Gallery

Thermodynamic profile extraction

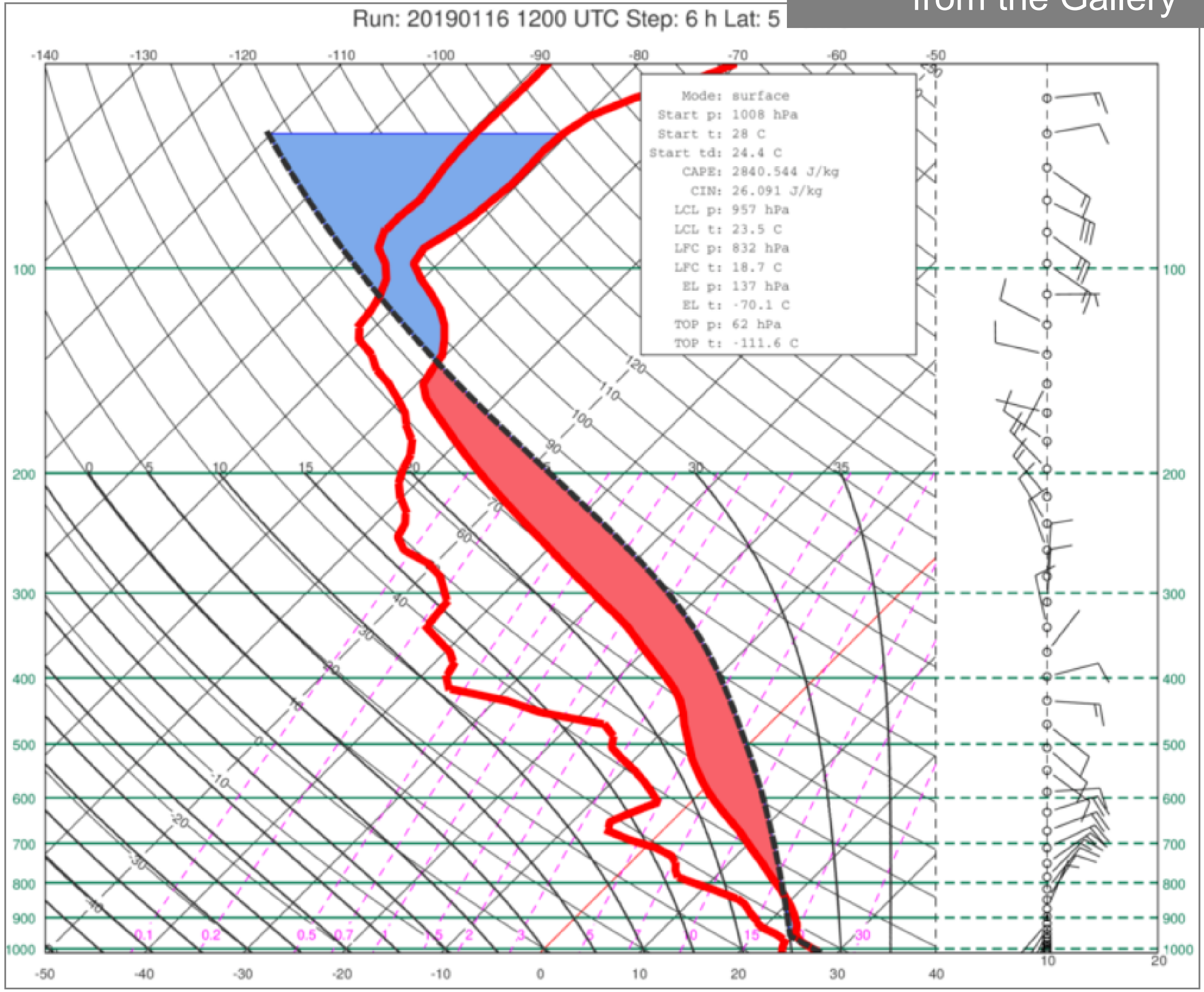


Thermo Data

store extracted profiles in **NetCDF** for further processing and visualisation on **tephigram**, **skew-t** or **emagram**

Thermo View, Plotting and Grid



Vertical cross sections

Example "Cross Section" from the Gallery

Vertical cross sections along a straight line for scalar and vector fields

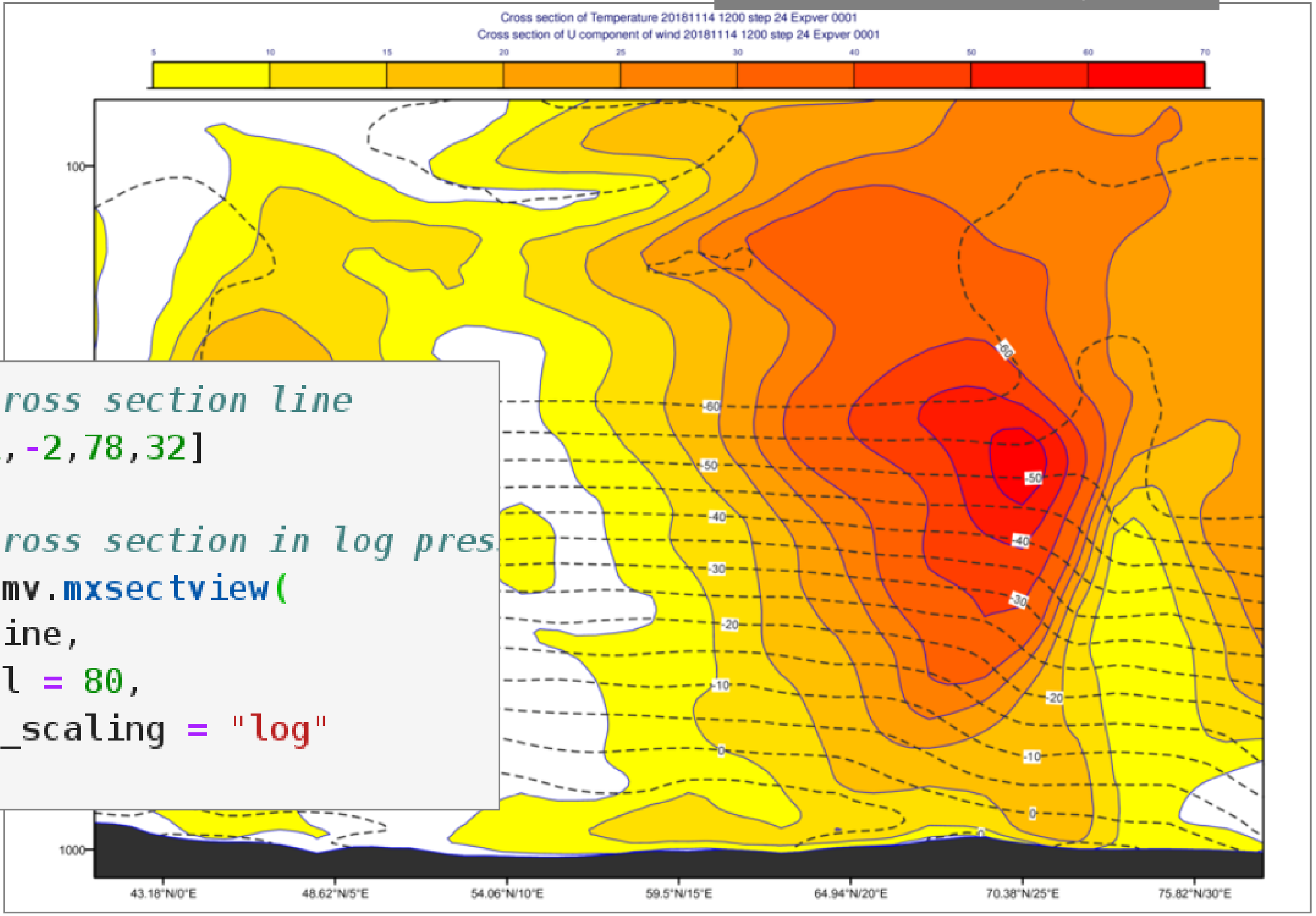
Cross Section Data

Computes cross section data and store it as **NetCDF** for further processing and visualisation

Cross Section View

```
# define cross section line
line = [41,-2,78,32]

# define cross section in log pres
xs_view = mv.mxsectview(
  line = line,
  top_level = 80,
  vertical_scaling = "log"
)
```



Average vertical cross sections

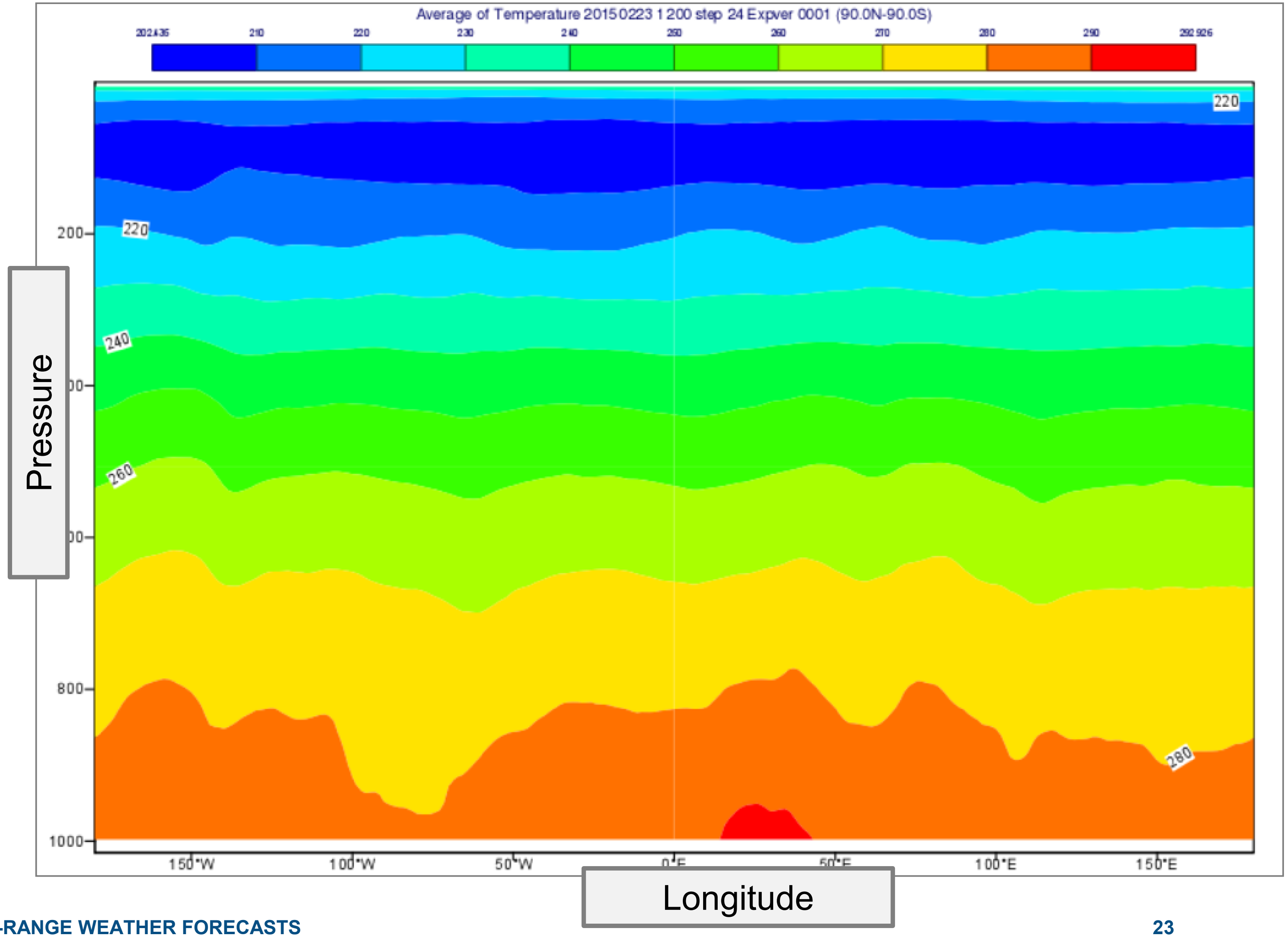
Zonal and meridional average cross section for an area or line

Average Data

Computes average data and store it as **NetCDF** for further processing and visualisation

Average View

Meridional average



Hovmoeller diagrams

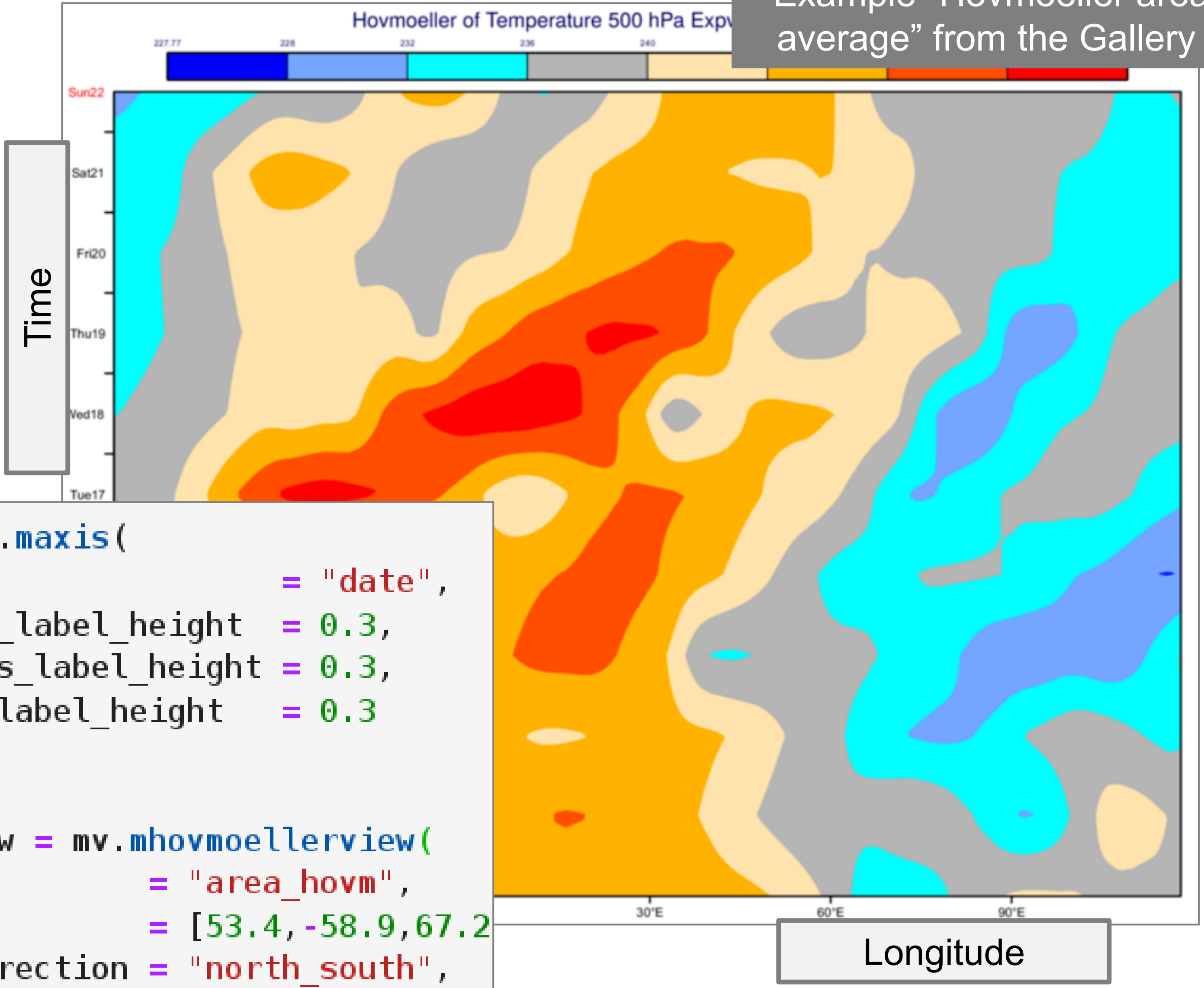
Hovmoeller diagrams for areas, lines and points

Hovmoeller Data

Computes data and store it as **NetCDF** for further processing and visualisation

Hovmoeller View

Example "Hovmoeller area average" from the Gallery




```
time_axis = mv.maxis(
    axis_type                  = "date",
    axis_years_label_height    = 0.3,
    axis_months_label_height   = 0.3,
    axis_days_label_height     = 0.3
)

hovmoeller_view = mv.mhovmoellerview(
    type                 = "area_hovm",
    area                 = [53.4,-58.9,67.2
    average_direction    = "north_south",
    time_axis            = time_axis
    )
```

Connecting fieldsets to the Python ecosystem

To perform computations not available in Metview

values()

gets a whole field as a **numpy** array (or 2D array for several fields)

NumPy

set_values()

saves values back into a field

Gets **values** of fieldset t into a 2D **numpy array**.

```
v = mv.values(t)
v.shape
```

```
(4, 13280)
```

Computes the kurtosis for each grid point with scipy.

```
v = stats.kurtosis(v, axis=0)
v
```

```
array([-1.74082924, -1.74538847, -1.75074808, ..., -1.5352
       -0.80854514, -0.88767356])
```

Saves the results into a new field.

```
r = mv.set_values(t[0], v)
```

Metview with Numpy and Scipy

Jupyter Notebook example from the Gallery

Principal component analysis of ensemble forecast fields

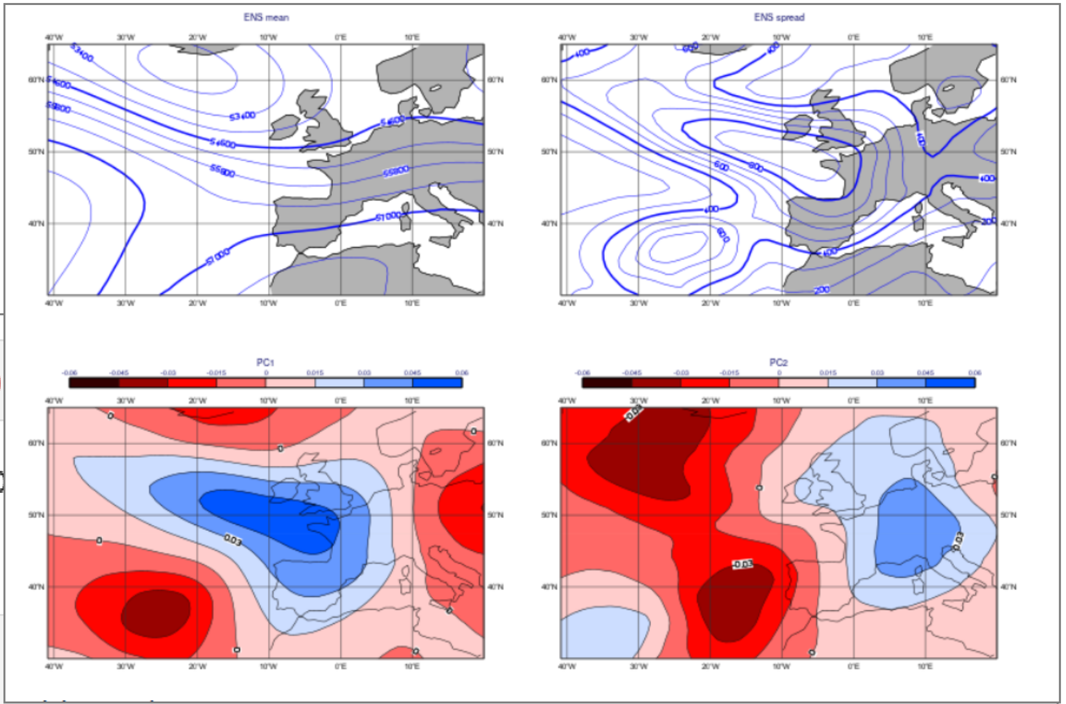
NumPy

In this example we will perform a principal component (PCA) analysis on ensemble forecast fields stored in GRIB format. We will use a combination of Metview, numpy and scipy to achieve this.

```
fs = mv.read("./z500_ens.grib
```

We will compute the principal compo

into a numpy array.




```
v = fs.values()
print(v.shape)
```

(51, 3266)

For the PCA we center the data, create the covariance matrix and compute the eigenvalues and eigenvectors of it.

```
v -= np.mean(v, axis = 0)
cov = np.cov(v, rowvar = False)
evals , evecs = LA.eigh(cov)
```

The resulting *evecs* array stores the eigenvectors as columns. The eigenvectors are guaranteed

Metview with XArray

Computing ensemble mean and spread with xarray and plotting the results with Metview

multidimensional labelled arrays (close to NetCDF data model)

Jupyter Notebook example from the Gallery

```
fs = mv.read(source="./wgust_ens.grib")
```

We load our fieldset into an xarray dataset.

```
xr.set_options(keep_attrs=True)
ds = fs.to_dataset()
```

The computation of the ensemble mean and spread for each timestep can be done by aggregating along the *number* (i.e. the ensemble) dimension of the dataset.

```
ds_mean = ds.mean(dim='number')
ds_spread = ds.std(dim='number')
ds_spread
```

```
<xarray.Dataset>
Dimensions:     (latitude: 41, longitude: 51, step: 3)
```

to_dataset()

exposes fieldset as an **xarray** dataset

(feature implemented via **cfgrib:** an ECMWF/B-Open development)

Having produced these datasets we will plot them with Metview so we convert back ou into fieldsets (i.e. into GRIB).

```
fs_mean = mv.dataset_to_fieldset(ds_mean, no_warn = True)
fs_spread = mv.dataset_to_fieldset(ds_spread, no_warn = True)
```

dataset_to_fieldset()

converts an **xarray dataset back** to fieldset (experimental)

NetCDF

Array oriented binary format

metadata access

functions and operators work on he current variable

value access

as a numpy array

```
nc = mv.read('wgust_era5.nc')
```

```
mv.variables(nc)
```

```
['longitude', 'latitude', 'time', 'i10fg']
```

```
mv.setcurrent(nc, 'i10fg')
```

```
'i10fg'
```

```
mv.attributes(nc)
```

```
{'_FillValue': -32767.0,
 'add_offset': 19.23302114169299,
 'long_name': 'Instantaneous 10 metre wind gust',
 'missing_value': -32767.0,
 'scale_factor': 0.0005813805163047624,
 'units': 'm s**-1'}
```

```
for name, val in zip(mv.dimension_names(nc), mv.dimensions(nc)):
    print(name, val)
```

```
time 1.0
latitude 721.0
longitude 1440.0
```

```
mv.values(nc)
```

```
array([5.0484993 , 5.0484993 , 5.0484993 , ..., 9.23909007, 9.23909
       9.23909007])
```

Geopoints

- Geopoints is Metview's format to handle spatially irregular data (e.g. observations)
- It is a column-based ASCII format (CSV)
- Can be directly plotted (symbol, vector, number)
- Rich API

```
geopoints distance ( geopoints,number,number )
geopoints distance ( geopoints,list )
```

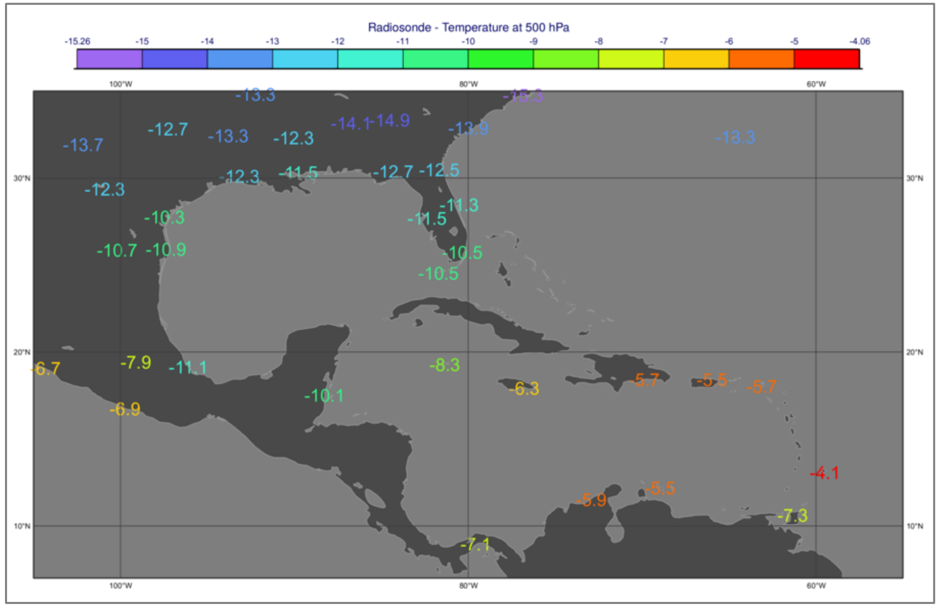
Returns geopoints with the value of each point being the distance in meters from the given geo
may be specified by supplying either two numbers (latitude and longitude respectively) or a 2-ele
and longitude in that order. The location should be specified in degrees.

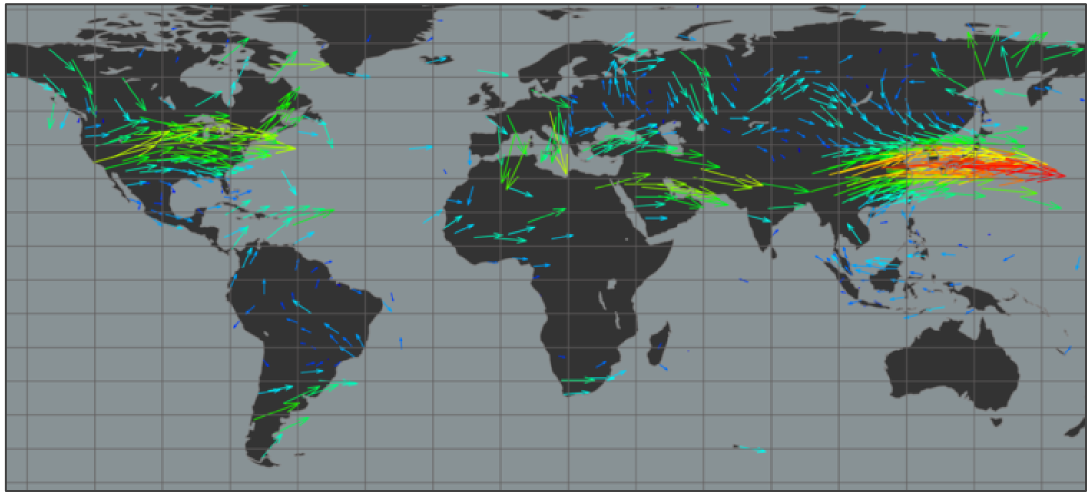
```
geopoints filter ( geopoints,geopoints )
```

A filter function to extract a subset of its geopoints input using a second geopoints as criteria.
have the same number of values. The resulting output geopoints contains the values of the first
the second geopoints is non-zero. It is usefully employed in conjunction with the comparison op

```
freeze = filter(temperature,temperature < 273.15)
```

- Usage is demonstrated with BUFR





BUFR

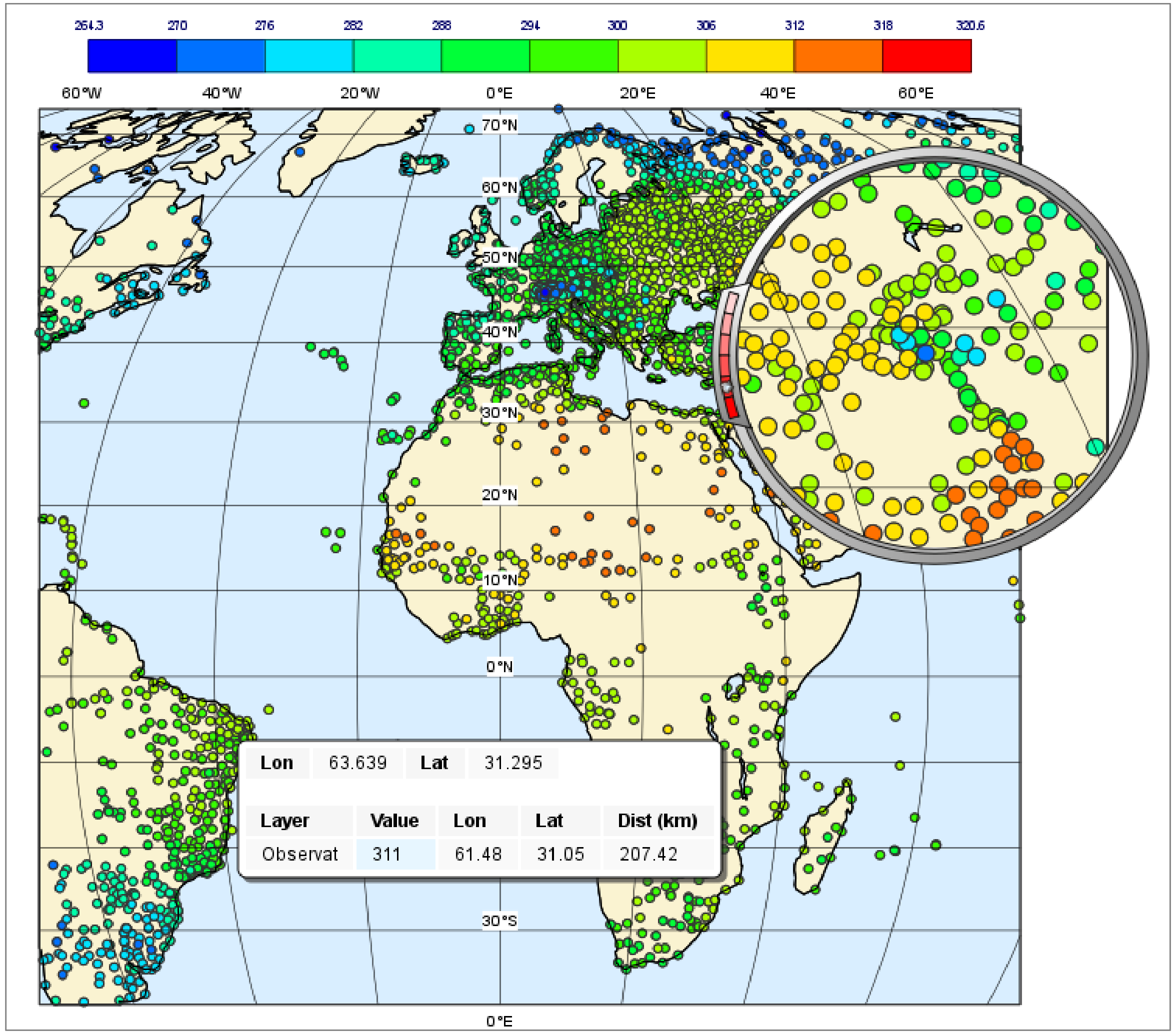
WMO's binary format (observations)

BUFR data can be fairly complex

In Metview BUFR data is filtered into Geopoints or CSV to plot and post-process it

Observation Filter

BUFR Picker



Computing forecast - observation difference

T2m forecast field

T2m BUFR observation filtered into geopoints

subtracting geopoints from fieldset

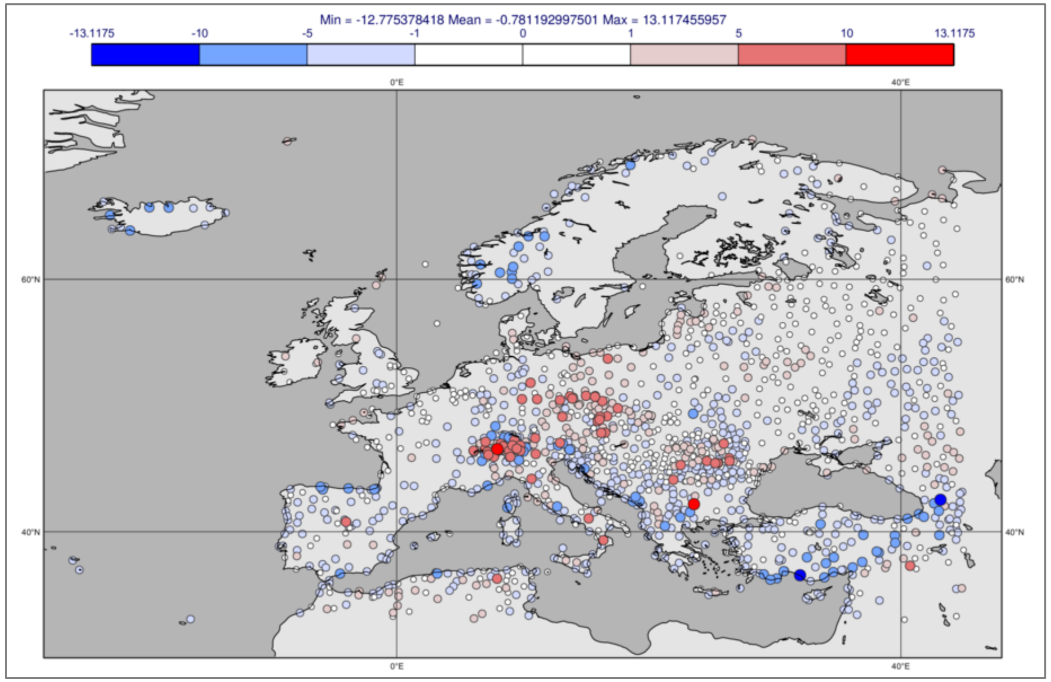
involves interpolation of field values to the Geopoints locations

```
t2m_fc48 = mv.read('t2m_fc48.grib')
synop = mv.read('t2m_obs.bufr')

# filter just the 2m temperature from the obs da
synop_t2m = mv.obsfilter(
    output    = "geopoints",
    parameter = "airTemperatureAt2M",
    data      = synop)

# compute the difference
diff = t2m_fc48 - synop_t2m
```

"Model-Obs Difference" example from the Gallery



BUFR filtering and Pandas

pandas

$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$

table data and time series analysis

to_dataframe()

Converts Geopoints into a **Pandas dataframe**

```
b = mv.read('temp.bufr')
```

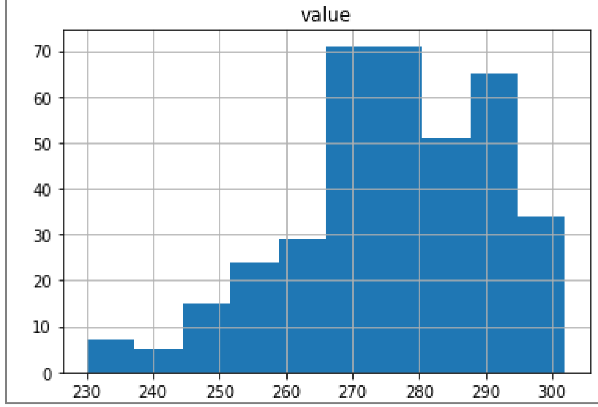
```
gpt = mv.obsfilter(data=b,
                   output='geopoints',
                   parameter='airTemperature',
                   level='descriptor_value',
                   level_descriptor='7004', first_level = 92500)
```

```
df = gpt.to_dataframe()
df.value.describe()
```

```
count    372.000000
mean     276.073118
std       14.973698
min      230.100000
25%      267.300000
50%      277.100000
75%      288.700000
max      302.000000
Name: value, dtype: float64
```

```
df.hist(column='value')
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f116cc8d
]],
      dtype=object)
```



ODB

Developed at ECMWF to handle observations in data assimilation

Set of data columns that can be accessed via an ODB/SQL query

ODB Filter

Performs an ODB/SQL query.
The result is another ODB.

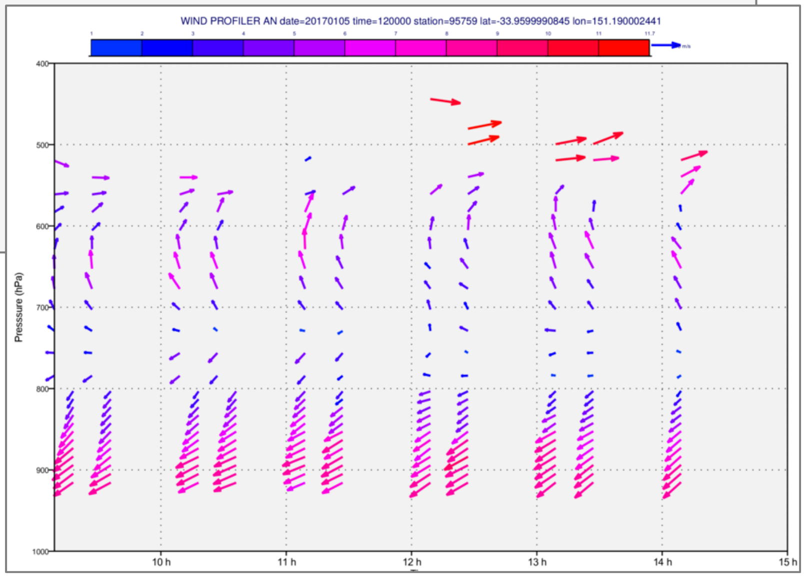
Example “ODB Wind Profiler” from the Gallery

```
# define station id
statid = "95759"

# read db
db = mv.read("wprof.odb")

# define query for u wind component
q_u = """select obsvalue as val,
      vertco_reference_1 as p,
      date@hdr as date,
      time@hdr as time
      where varno=3 and statid='{}'""".format(statid)

# filter u
f_u = mv.odb_filter(
    odb_query   =   q_u,
    odb_data    =   db
    )
```



ODB filtering and Pandas

pandas

to_dataframe()

converts ODB into a **Pandas dataframe**

Reads ODB data.

```
db = mv.read('amsua.odb')
```

Defines **ODB/SQL query** and runs it.

```
q = """select lat as lat,
        lon as lon,
        obsvalue as value
    where vertco_reference_1=5"""

f = mv.odb_filter(odb_query = q,
                  odb_data = db)
```

Converts resulting ODB to **Pandas dataframe**.

```
df = f.to_dataframe()
df.describe()
```

	lat@hdr	lon@hdr	value
count	16419.000000	16419.000000	16419.000000
mean	2.406042	-8.115537	250.466231
std	40.980465	108.687330	7.957377
min	-76.627098	-179.999405	225.009995
25%	-31.948250	-101.569847	244.460007
50%	0.498400	-19.469299	252.639999
75%	33.283550	86.641251	256.940002
max	87.500198	179.991806	266.730011

Climate Data Store (CDS)

Large number of datasets including ECMWF data

Publicly available

GRIB + NetCDF

IMPLEMENTED BY ECMWF

Climate Change Service

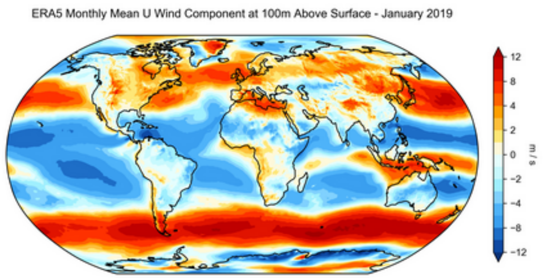
This is a new service --

Home Search Datasets Applications Toolbox FAQ

ERA5 monthly averaged data on single levels from 1979 to present

Overview **Download data** **Documentation**

ERA5 is the fifth generation ECMWF reanalysis for the global climate and weather for the past 4 to 7 decades. Currently data is available from 1979. When complete, ERA5 will contain a detailed record from 1950 onwards. ERA5 replaces the ERA-Interim reanalysis.



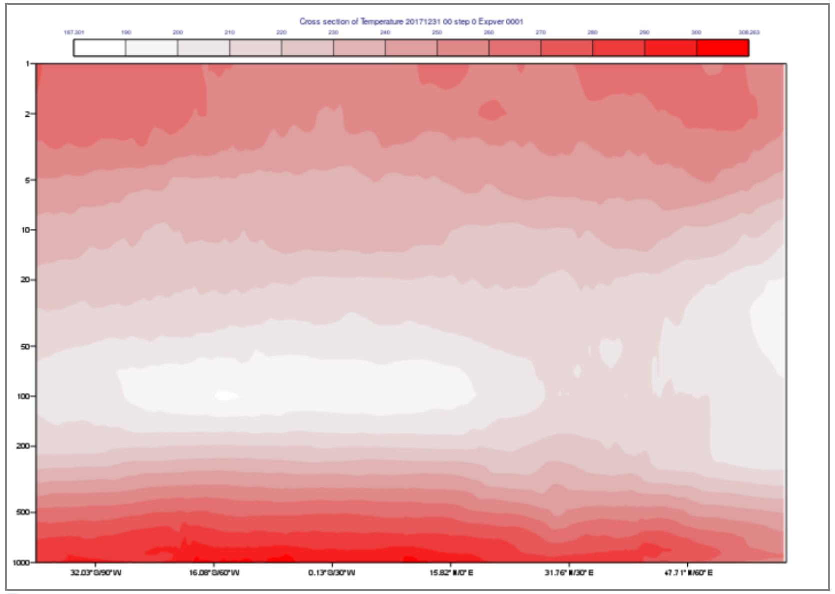
Reanalysis combines observations into globally complete fields using the laws of physics with the method of data assimilation (4D-Var in the case of ERA5). ERA5 provides hourly estimates for a large number of atmospheric, ocean-wave and land-surface quantities. An uncertainty estimate is sampled by an underlying 10-member ensemble at three-hourly intervals. Ensemble mean and spread have been pre-computed for convenience. Such uncertainty estimates are closely related to the information content of the available observing system which has evolved dramatically over time. They also indicate flow-dependent sensitive areas.

The native resolution of the ERA5 atmosphere and land reanalysis is 31km on a reduced Gaussian grid (Tl639) and 63km (TL319) for the ensemble members. Ocean-wave products are produced at 0.36 degrees and 1 degree for the ensemble. The atmospheric component consists of 137 levels in the vertical from the surface up to 1 Pa (about 80km). This spans the troposphere, stratosphere and mesosphere. There are both analysis fields and short forecast fields that link the assimilation windows used in 4D-Var. A detailed description can be found in the online ERA5 documentation. The full data set resides in the MARS tape archive.

Climate Data Store (CDS)

accessed through a Python API = cdsapi

works well with Metview's Python interface



Jupyter Notebook example from the Gallery

```python
import metview as mv
import cdsapi
```

Retrieve ERA5 temperature data in GRIB format using the CDS API (access needs to be set up first).

In [ ]:

```python
c = cdsapi.Client()

c.retrieve("reanalysis-era5-pressure-levels",
  {
    "variable": "temperature",
    "pressure_level":['1','2','3','5','7','10','20','30','50','70','100','150
                      '200','250','300','400','500','600','700','800','850',
                      '900','925','950','1000'
                     ],
    "product_type": "reanalysis",
    "date": "20171231",
    "time": "00:00",
    "format": "grib"
   },
  "temp.grib")
```

Downloads ERA5 GRIB and generates cross section

Where to find out more

Icon reference

ECMWF Spaces Log in

- 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

ECMWF

Where to find out more

Function documentation

ECMWF Spaces Calendars Create ··· Search

- Change History
- User Guide
  - Using Metview
  - The Macro Language
    - Macro syntax
    - Macro Data Types
    - List of Operators an
      - Information Functi
      - The nil Operand
      - Number Functions
      - String Functions
      - Date Functions
      - List Functions
      - Vector Functions
      - **Fieldset Functions**
      - Geopoints Functio
      - Geopointset Funct
      - NetCDF Functions
      - ODB Functions
      - Table Functions
      - Observations Func
      - Definition Functior
      - File I/O Functions
      - Timing Functions

Space tools

```
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=2Ωsinφ` 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.

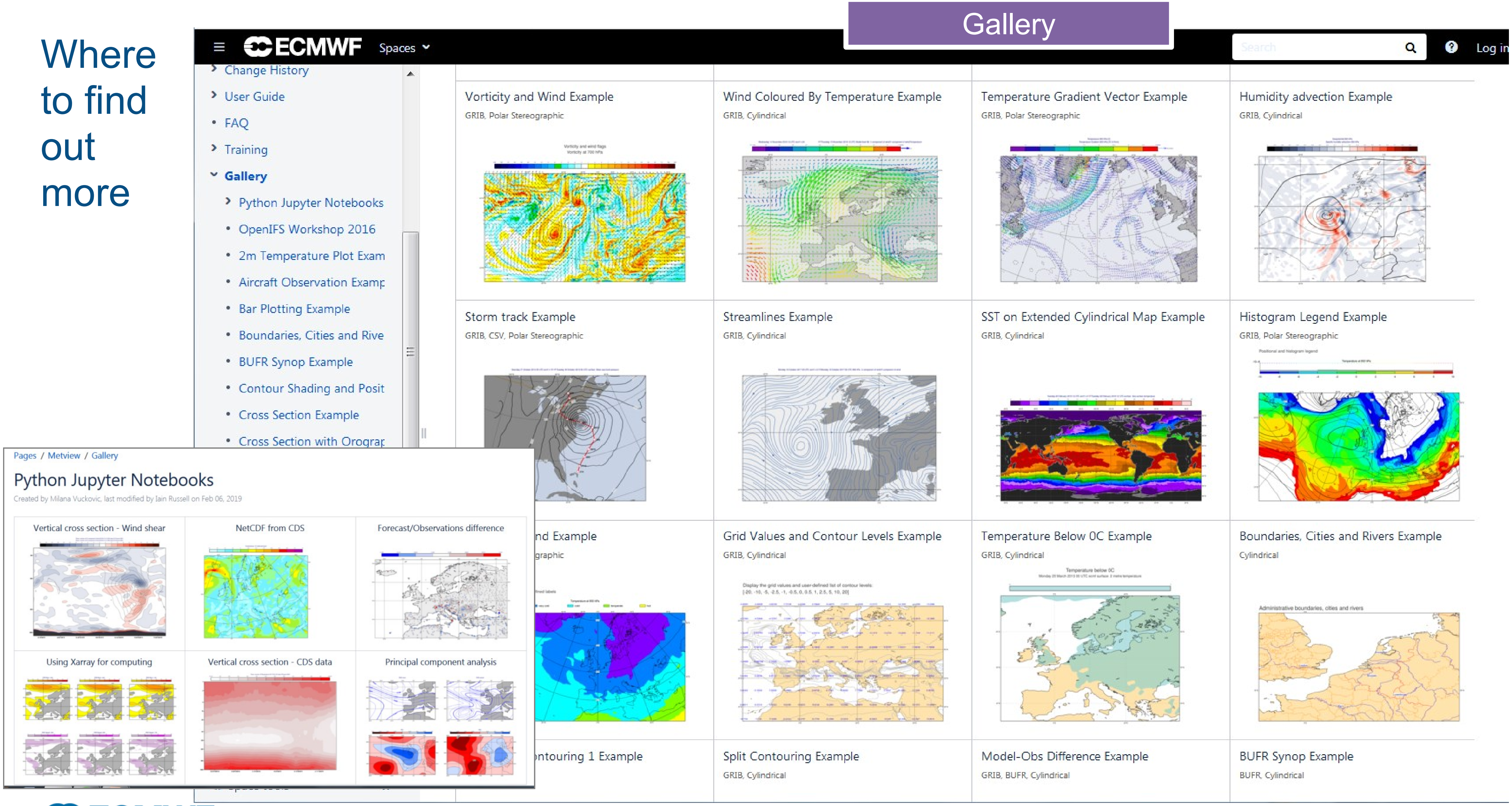
```
geopoints gfind ( fieldset,number )
geopoints gfind ( fieldset,number,number )
```

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 :

```
abs(data-value) <= threshold
```

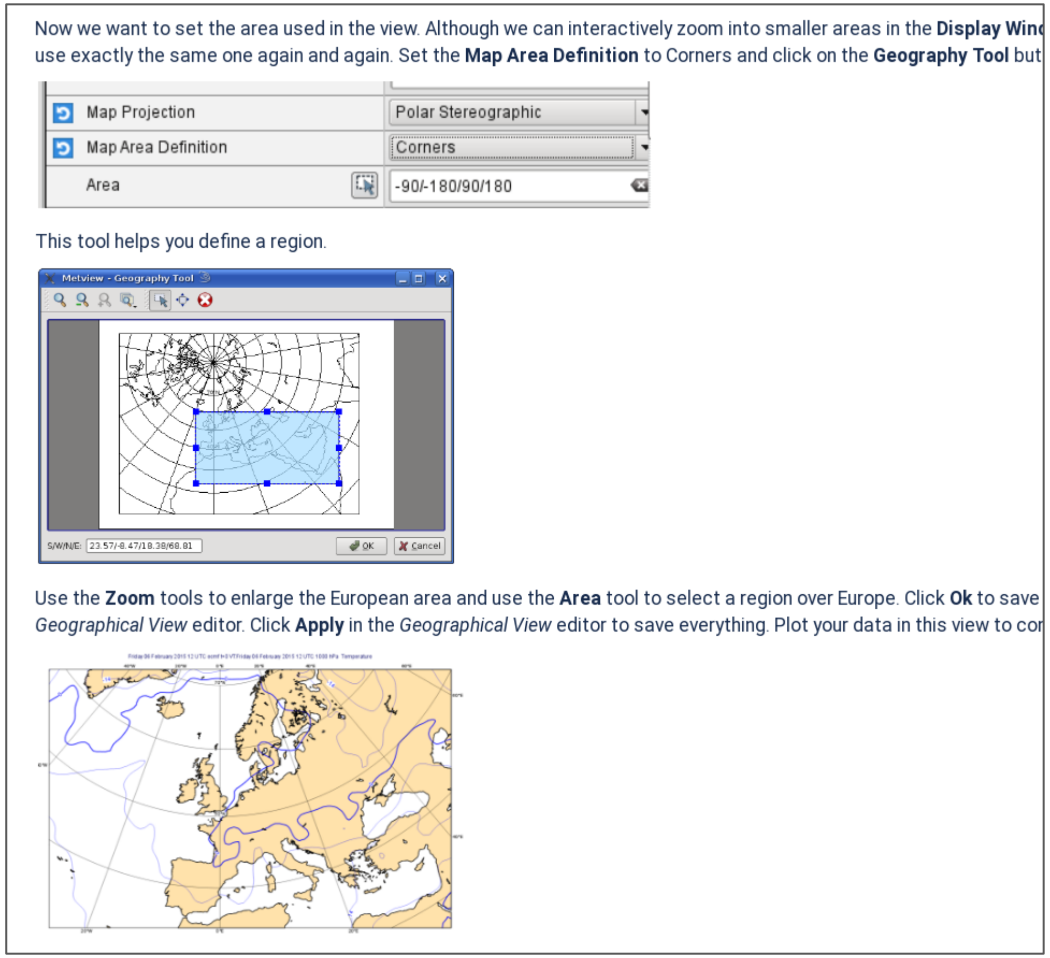
ECMWF

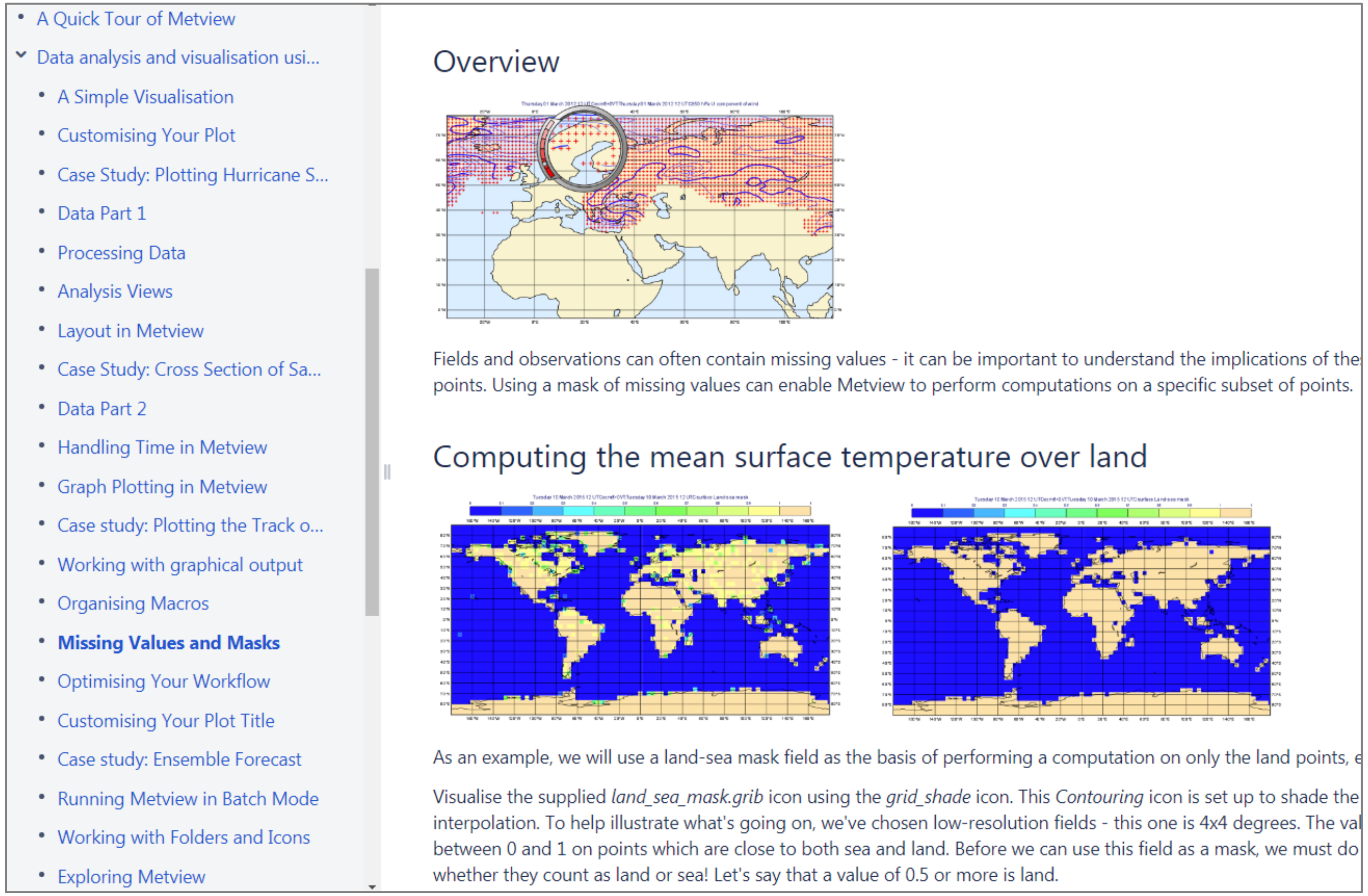
Where to find out more



Where to find out more

Lots of material online including tutorials





Metview Availability – on ECMWF systems

- Versioned using the 'module' system

Interactive session

```
module swap metview/new
metview
```

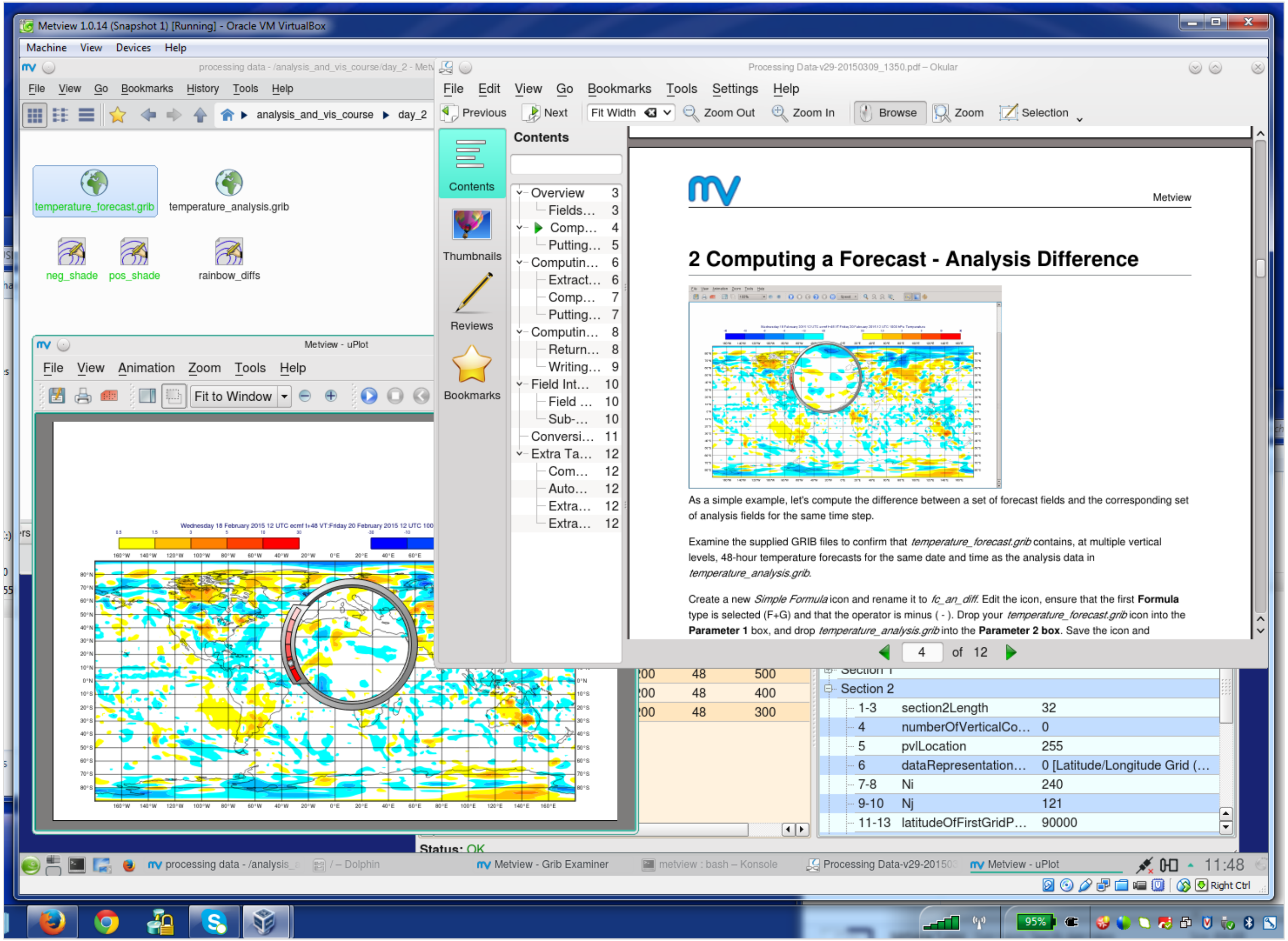
Batch, Jupyter notebook

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

Metview availability – outside ECMWF

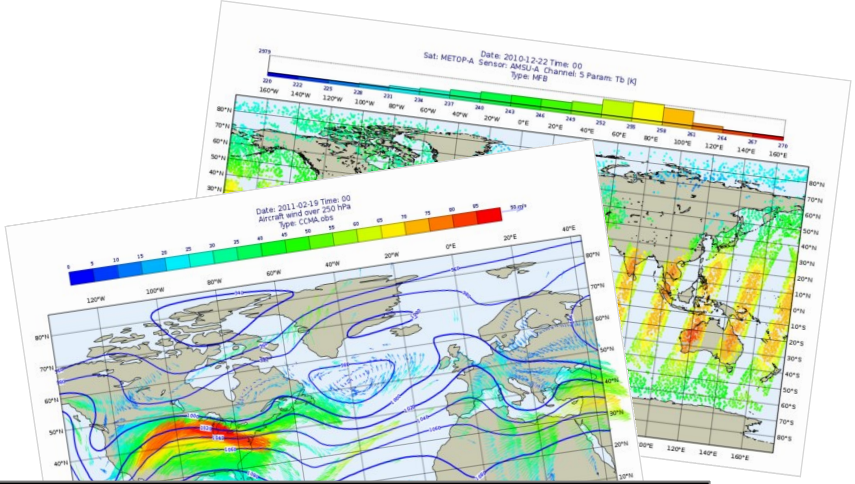
- Install from binaries
- Conda (via conda-forge)
- Build from source
- Build from bundle
- The Metview Python interface has to be installed separately:

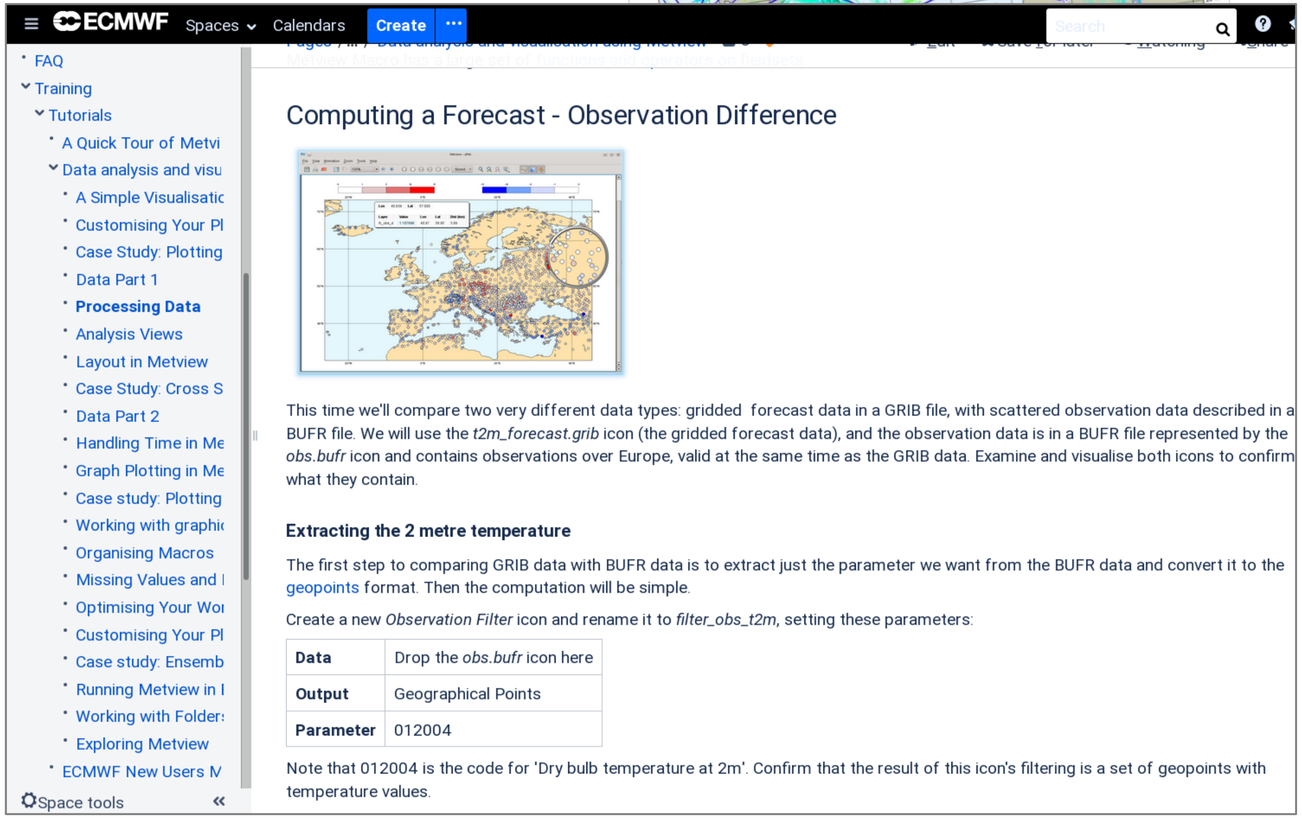
```
pip install metview
```



For more information…

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





ECMWF Spaces Calendars Create

- FAQ
- Training
  - Tutorials
    - A Quick Tour of Metvi
    - Data analysis and visu
      - A Simple Visualisatic
      - Customising Your Pl
      - Case Study: Plotting
      - Data Part 1
      - **Processing Data**
      - Analysis Views
      - Layout in Metview
      - Case Study: Cross S
      - Data Part 2
      - Handling Time in Me
      - Graph Plotting in Me
      - Case study: Plotting
      - Working with graphi
      - Organising Macros
      - Missing Values and I
      - Optimising Your Wo
      - Customising Your Pl
      - Case study: Ensemb
      - Running Metview in I
      - Working with Folder
      - Exploring Metview
    - ECMWF New Users M

Space tools

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 *obs.bufr* 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.

Questions?