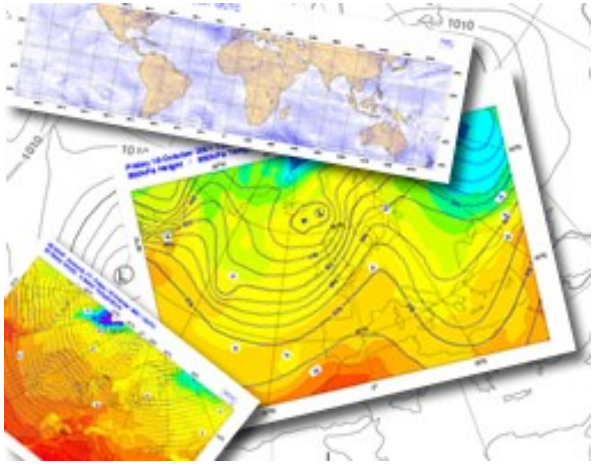
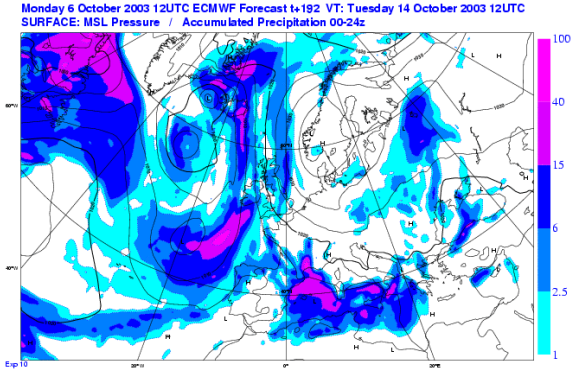
Magics Training course



Sylvie Lamy-Thépaut **Stephan Siemen**

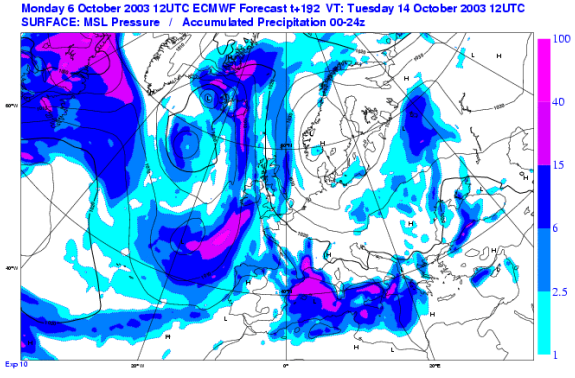
Meteorological Visualisation Section

Objectives of this course



- Giving you an overview of possibilities of the Magics graphics library
- Show you ways of using the documentation and find help
- Giving you good templates for your work
- Introduce the Magics team … us ☺

Magics

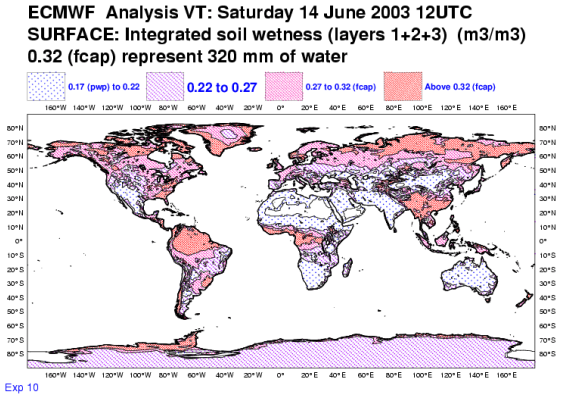


- Overview : What can it do?
- What is Magics++?
  - How to use it?
  - How the different APIs work?
    - ➔ Scripting language: Python
  - How to use Magics outputs?

Magics overview

- **Magics is meteorologically-oriented**
  - ➔ GRIB
  - ➔ BUFR
  - ➔ Specific Visualisation
- **Magics provides 3 simple APIs**
  - ➔ Fortran / C
  - ➔ Python
  - ➔ MagML / MagJSon
- **Magics provides a small set of actions**
  - ➔ Contouring, Symbol
- **And a large set of parameters for each action**
  - ➔ Large set of parameters
  - ➔ Small number of

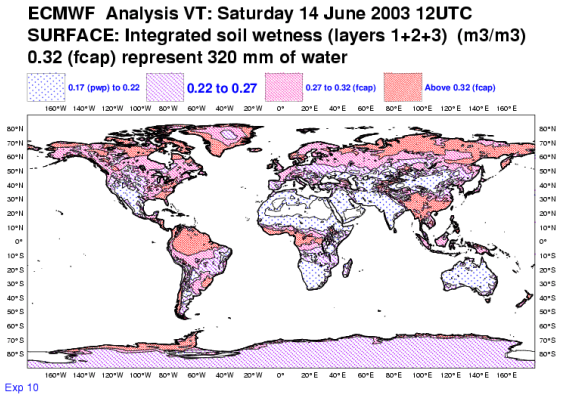
  Fortran callable subroutines



Visit to MetOps room

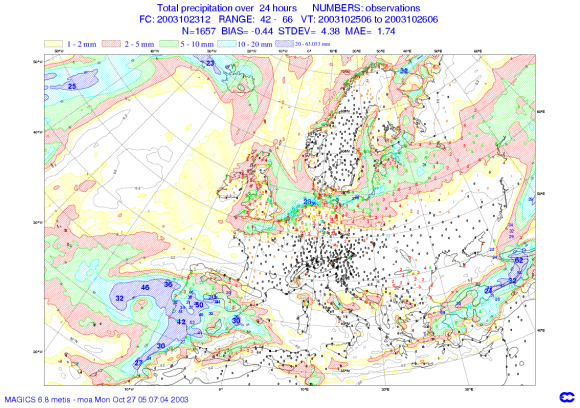
- We will now have a tour of the MetOps room
- The room contains a small sample of plots generated every day at ECMWF with Magics/Metview
- If you find types of maps interesting for your work, please free to ask us how you can try to generate the map(s) yourself

What can you do with Magics?



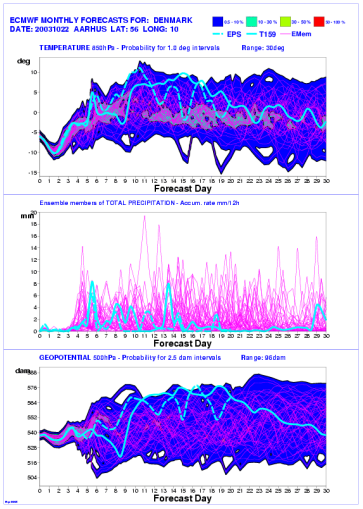
- Define your Geographical Area and Projection.
  - e.g. Stereographic Polar, Cylindrical
- Import data
  - GRIB, matrix (z, but also u/v)
  - NetCDF Data.
  - BUFR for Observations
- Plot some contours
  - High quality contouring

What can you do with Magics?



- Display wind fields
  - Arrows or flags
- Display Observations
  - Follow WMO representation
- Plot Symbols

What can you do with Magics?

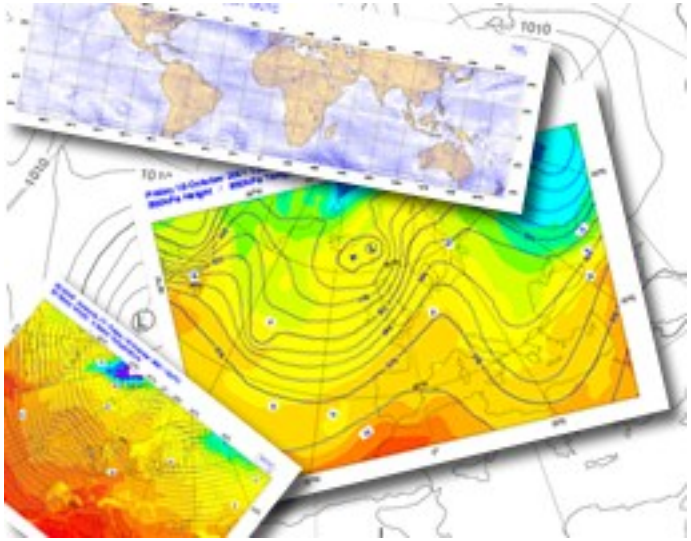


- Add Text and/or automatic title

  From the GRIB or BUFR Header

- Create a nice legend
- Display Axes and Graphs
- Display some satellite images
- Organise your plots with a nice layout

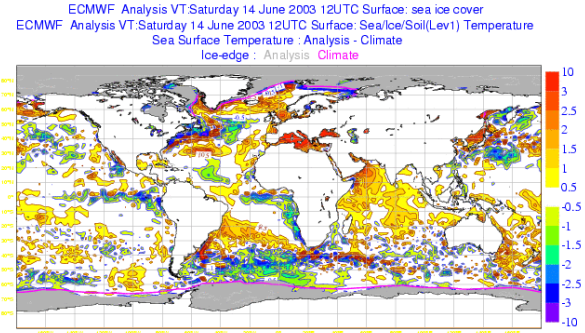
Introduction to Magics



Introduction to Magics

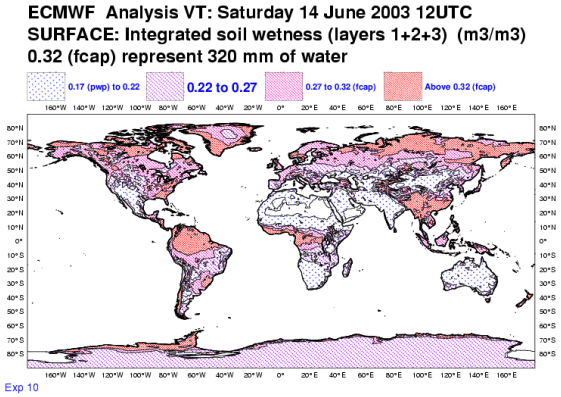
- This training course will give you the basis for your own future Magics programs.
- We mainly use the Python interface
- The Magics API contains too many parameters to be handled in detail within 2 days. That is why we concentrate on giving you templates for basic Magics programs. From them, you can build complex plots.
- The course will mostly be exercises in which you can learn how to write programs on your own with the help of the online documentation.
- Please do not hesitate to ask any questions!!!

MAGICS - History



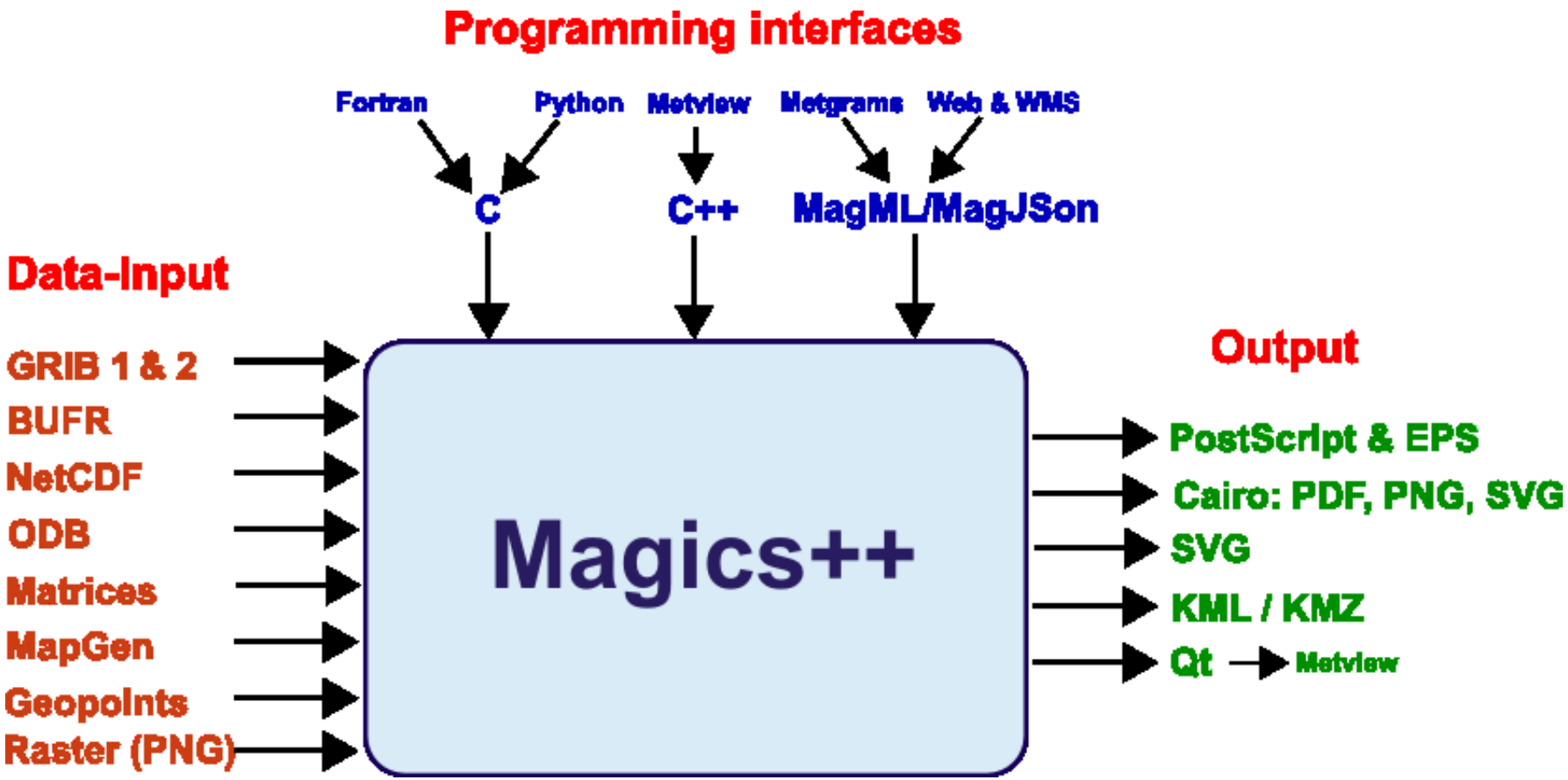
- 1983 - Development started
- Meteorological Applications Graphics Integrated Colour System
- MAGICS is a Fortran library
- MAGICS is installed in more than 30 member states and countries
- MAGICS is used by *Synergie* (Météo-France)
- MAGICS is the graphical kernel of Metview
- 2004 – (Re-)Development of Magics++ started
  - ♦ Graphical kernel of Metview 4 & ECMWF web products

Magics++ - Overview

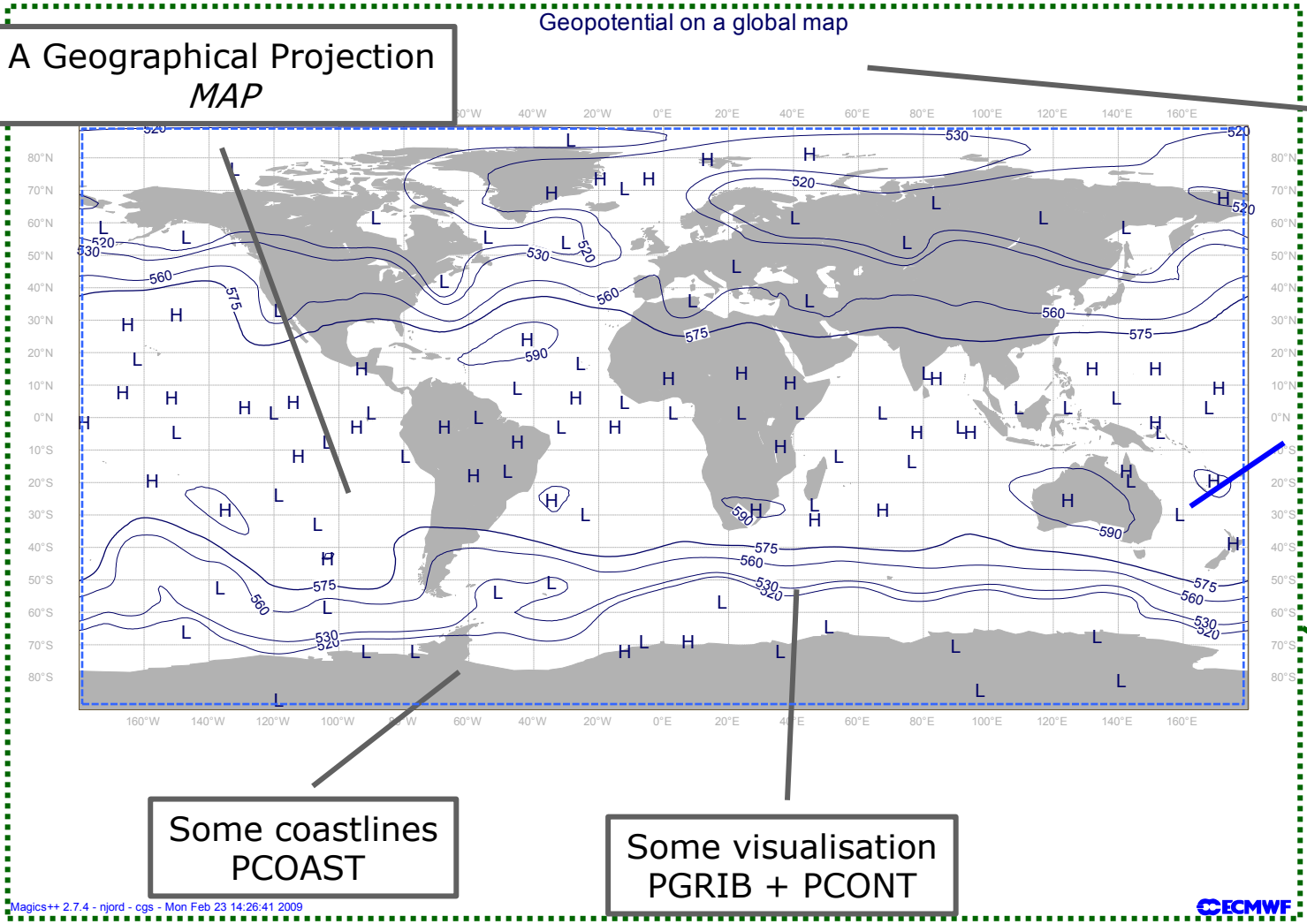


- Magics++ is a rewrite of MAGICS using C++ instead of Fortran
- Magics++ is (mostly) backwards compatible to MAGICS
- Magics++ makes it easier to add new features
- New APIs: C, Python, and MagML
- New output formats: PDF, EPS, PNG, SVG, KML
- New data input: GRIB 2, NetCDF, CSV/MapGen, ODB access
- New contouring

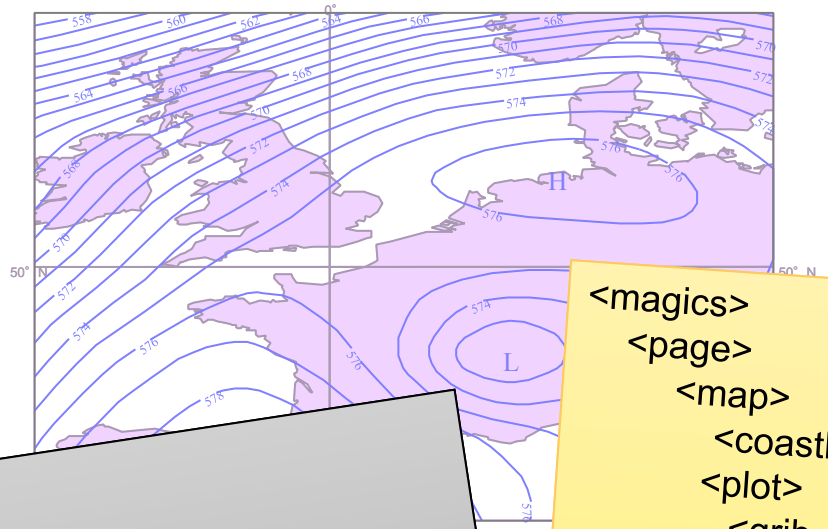
Magics++ architecture



What does a Magics plot consist of?



The 3 simple APIs




```
call popen
call pcoast
call psetc('grib_input_file_name', 'grib.grb')
call pgrib
call psetc('contour_line_colour', 'blue')
call pcont
call pclose
```

```
<magics>
  <page>
    <map>
      <coastlines/>
      <plot>
        <grib grib_input_file_name = 'grib.grb/>
        <contour  contour_line_colour =  'blue/>'
      </plot>
    </map>
  </page
</magics>
```

```
grib = mgrib({'grib_input_file_name': 'grib.grb'})
contour = mcont({'contour_line_colour': 'blue'})
coast =  coast({})
plot(coast, grib, contour)
```

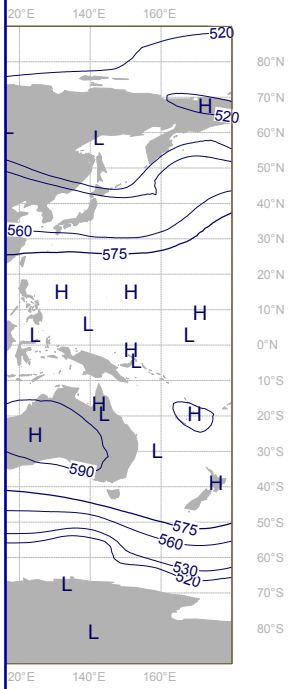
What is MagML/MagJSON?

- MagML/MagJSON is a description language based on XML/JSON
  - Tags, as in HTML <magics>, <page>, …
  - Hierarchical structure
  - Can be handled/processed by any XML tools/packages
    - ➔ JavaScript/AJAX, XSLT
- It is descriptive - only simple data processing possible
  - No ‘if-else’, loops or bindings to programming languages
  - It allows variables!

Main concepts of Magics

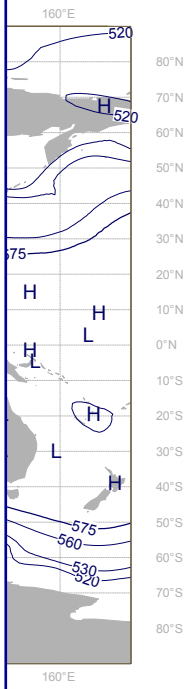
Fortran Interface

```
call popen
call psetc('map_coastline_land_shade', 'on')
call pcoast
call psetc('grib_input_file_name', 'z500.grb')
call pgrib
call psetc('contour_line_colour', 'navy')
call pcont
call pclose
```



Main concepts of Magics

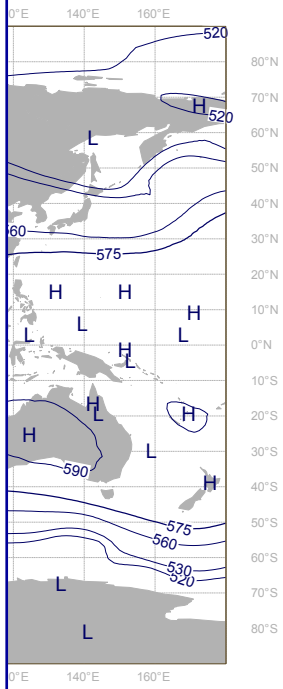
```
<magics>
  <page>
    <map>
      <coastlines map_coastline_land_shade='on'/>
      <plot>
          <grib grib_input_file_name='z500.grb'/>
          <contour contour_line_colour='navy'/>
      </plot>
    </map>
  </page>
</magics>
```



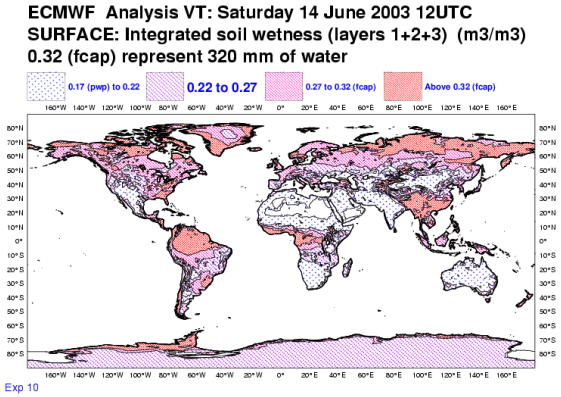
Main concepts of Magics

PYTHON Interface

```
coast = mcoast(
        map_coastlines_shade_on='on' )
grib = mgrib(
        grib_input_file_name='z500.grb')
contour = mcont(
        contour_line_colour='navy')
plot(coast, grib, contour)
```

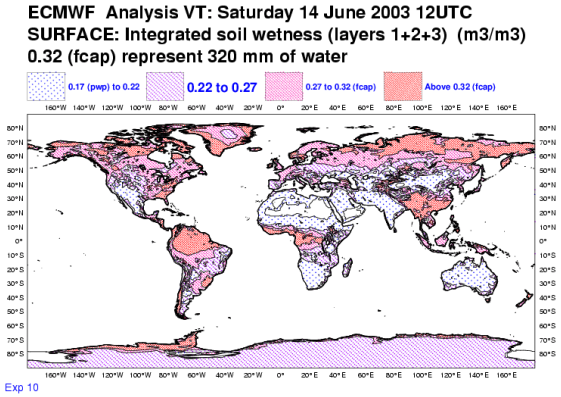


Metview-like python interface

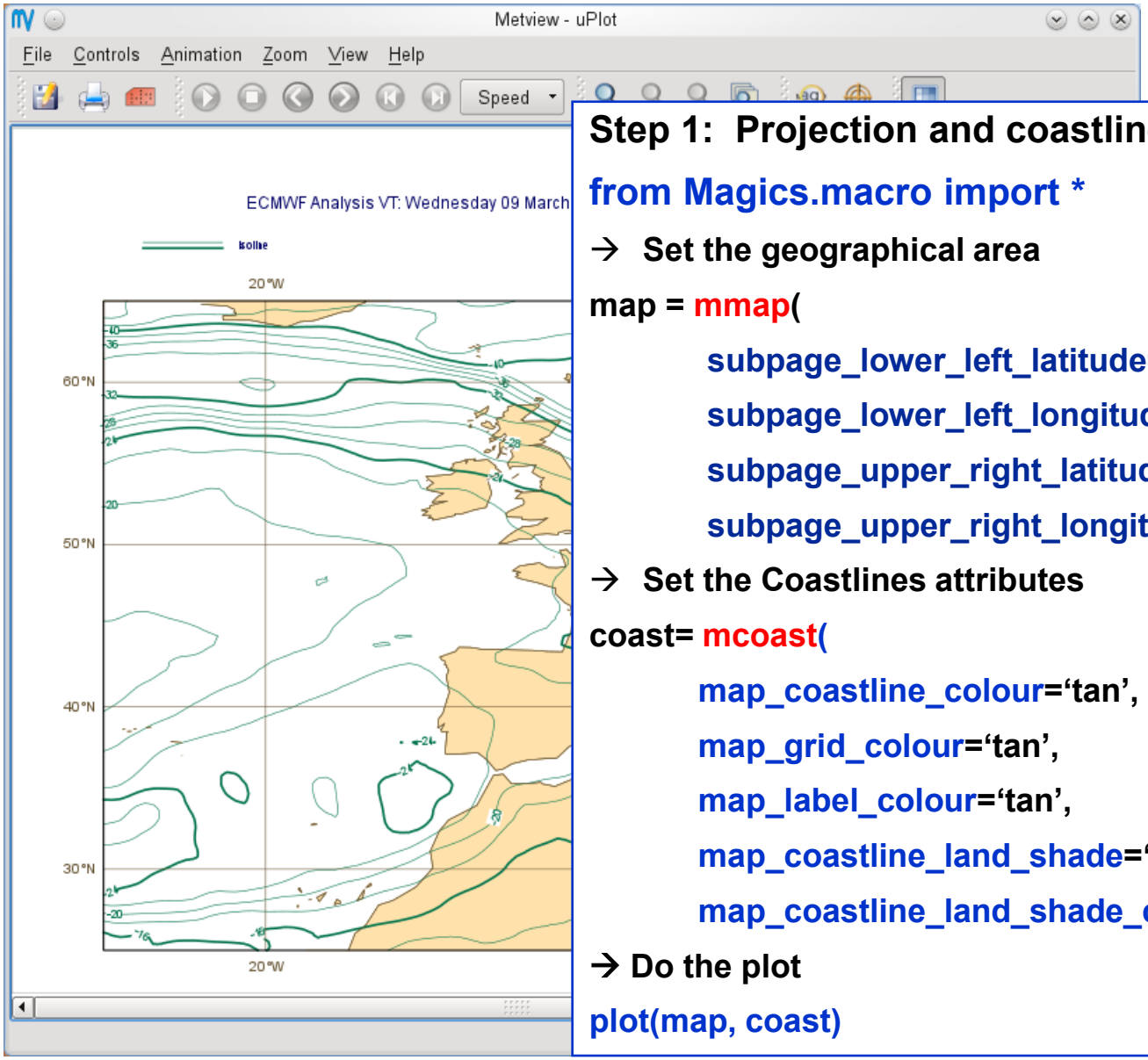


- **New higher level python module**
  - ➔ Very similar to the Metview Macro
  - ➔ Can be easily embedded in a more complex python script
- **Comes with a basic set of Objects…**
  - ➔ To setup the projection
    - ♦ **mmap**
  - ➔ To load data
    - ♦ **mgrib, mnetcdf …**
  - ➔ To perform visual actions
    - ♦ **mcoast, mcont, mwind, …**
  - ➔ To create a new page
    - ♦ **page**

Metview-like python interface



- Each object has a large set of parameters …
- An Python object has the same parameters as its equivalent C/Fortran action routine (same name, same default)
  - mcoast→pcoast , mgrib→pgrib, mcont→pcont
  - The ‘m’ prefix is for the compatibility with Metview Macro.
- The parameters are only set for the relevant object.
- The method *plot* will perform the plot, and call the action routines sequentially.



Step 1: Projection and coastlines

```
from Magics.macro import *
```

→ Set the geographical area

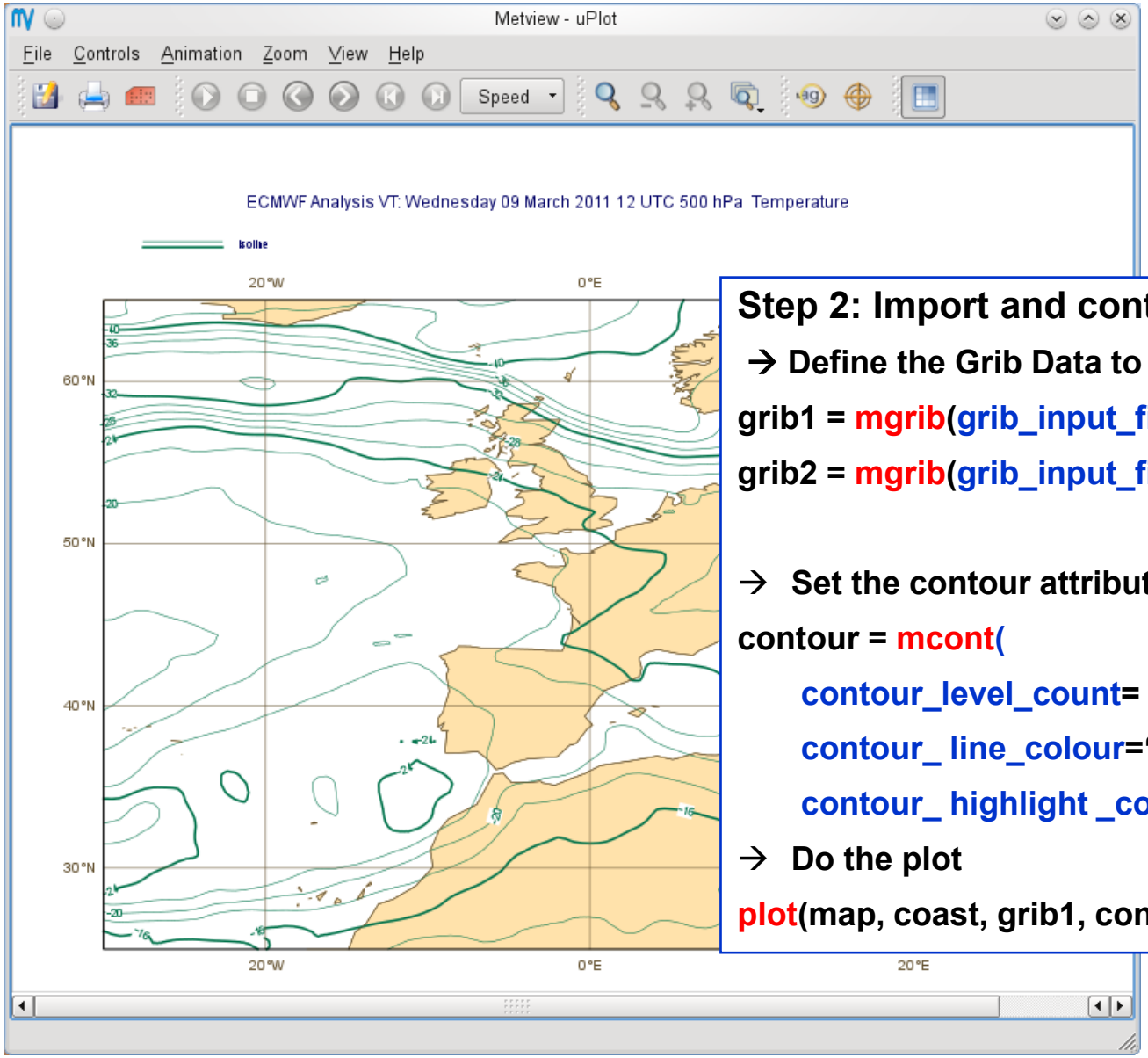
```
map = mmap(
        subpage_lower_left_latitude=25.,
        subpage_lower_left_longitude=-30.,
        subpage_upper_right_latitude=65.,
        subpage_upper_right_longitude=25.)
```

→ Set the Coastlines attributes

```
coast= mcoast(
        map_coastline_colour='tan',
        map_grid_colour='tan',
        map_label_colour='tan',
        map_coastline_land_shade='on',
        map_coastline_land_shade_colour='cream')
```

→ Do the plot

```
plot(map, coast)
```



Step 2: Import and contour the GRIB

→ Define the Grib Data to plot

```
grib1 = mgrib(grib_input_file_name='t500.grb')
grib2 = mgrib(grib_input_file_name='z500.grb')
```

→ Set the contour attributes

```
contour = mcont(
      contour_level_count= '20',
      contour_ line_colour='evergreen',
      contour_ highlight _colour='evergreen')
```

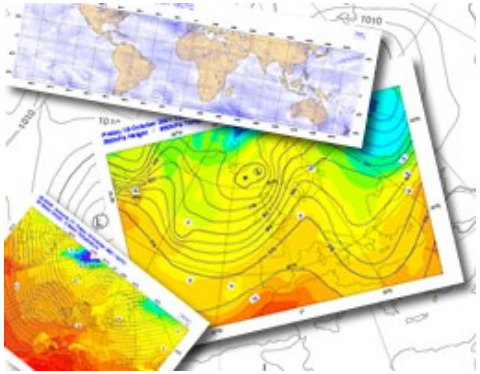
→ Do the plot

```
plot(map, coast, grib1, contour, grib2, contour)
```

How to interpret your Magics++ Python script?

- Setup Magics once per shell: use magics++
- Create your Python script
  - Use an editor (*vi*, xemacs, *nedit, kwrite*) to write your Magics object
- Interpret your script
  - python *mymagics.py*
- View your result - Use a postscript viewer : *gv*, *display*
- Modify your script if necessary …

How to use the tutorial?



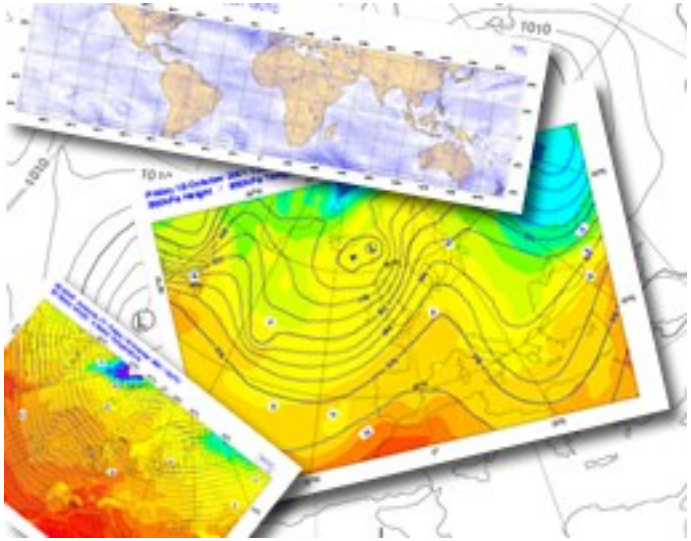
How to use the tutorial?

- The tutorial contains exercises, clues and solutions.
- For each new concept, you will find a link to the complete Magics documentation.

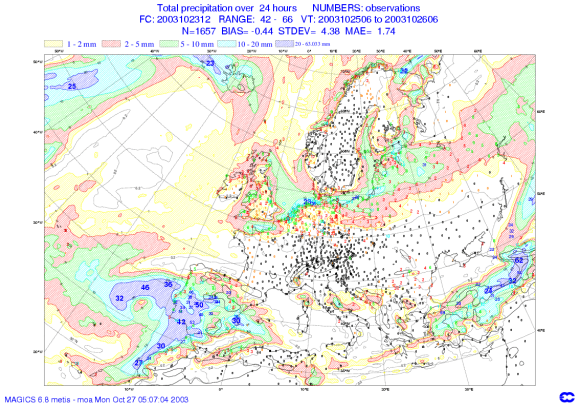
Python

- Interpreter
- We use 2.x versions with NumPy
- Be careful:
  - Indent sensitive (avoid tabs)
  - 1 / 2 = 0 !!! → 1.0 / 2.0 = 0.5

More on Coastlines

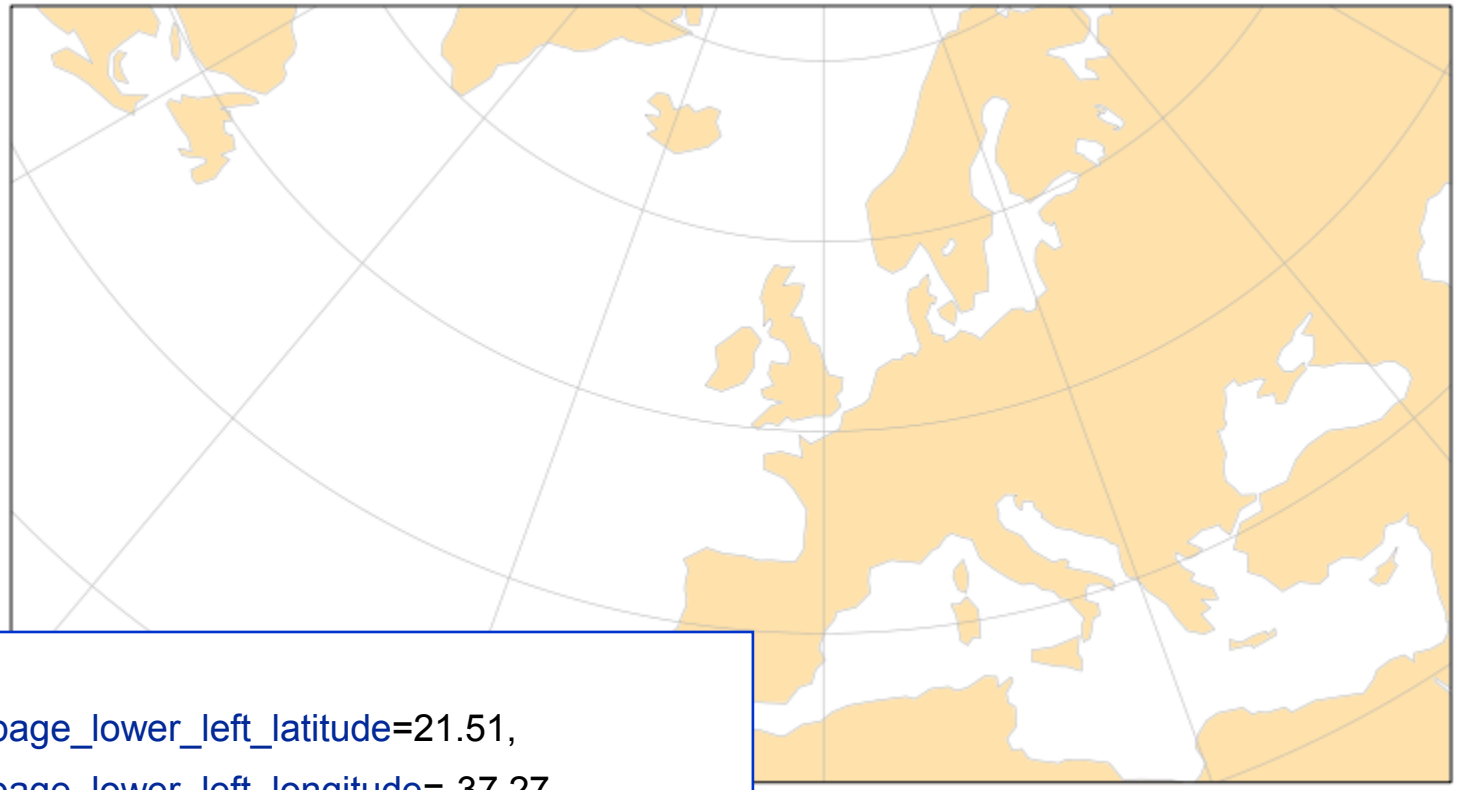


A Bit More on Coastlines



- Selection of the projection
  - Polar stereographic, Mercator, cylindrical
- Selection of the Geographical Area
- Definition of the coastlines and grid attributes
- Land and/or Sea Shading

Setting projection and area

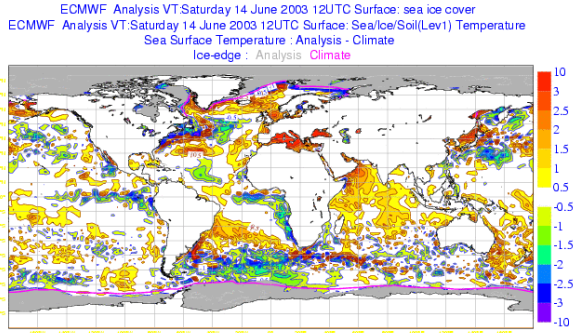


```
mmap(
        subpage_lower_left_latitude=21.51,
        subpage_lower_left_longitude=-37.27,
        subpage_upper_right_latitude=51.28,
        subpage_upper_right_longitude=65.,
        subpage_map_projection='polar_stereographic')
```

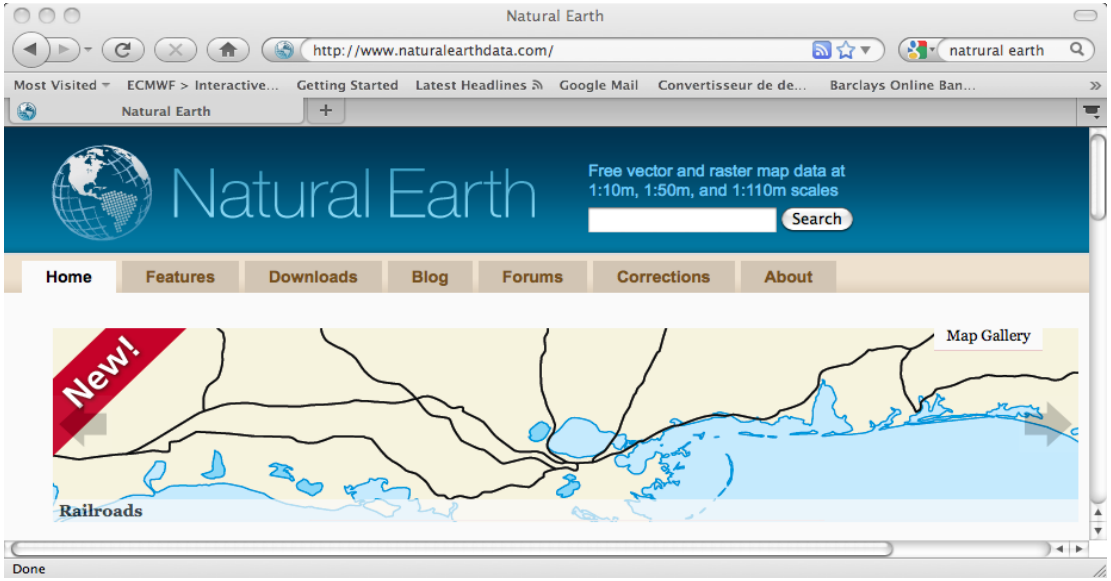
Setting land shading

```
coast = mcoast(
            map_coastline_colour='grey',
            map_coastline_land_shade='on',
            map_coastline_land_shade_colour='cream',
            map_coastline_sea_shade='on',
            map_coastline_sea_shade_colour='white')
```

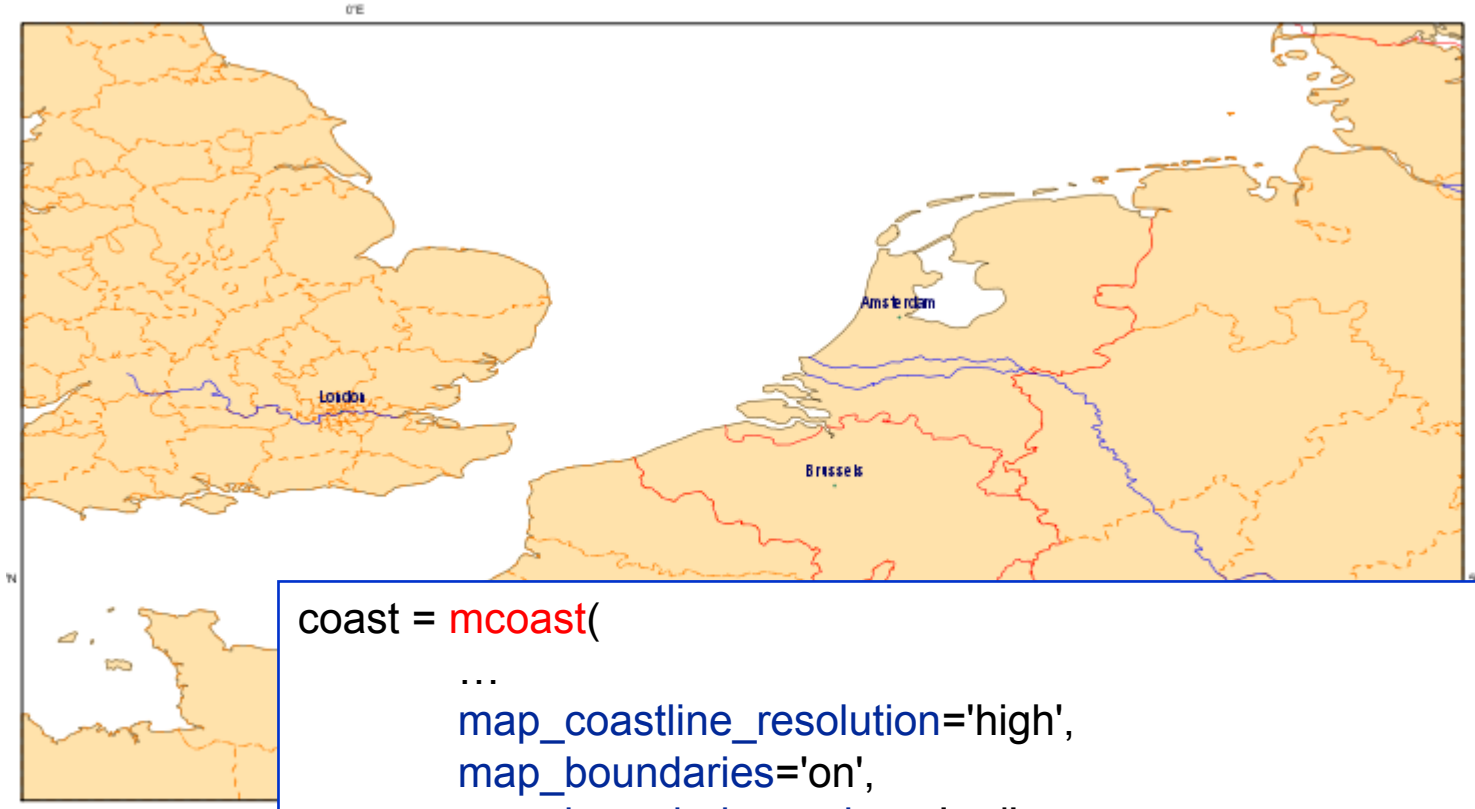
Coastlines and boundaries ...



- Boundaries can now be plotted with Magics
- Our dataset comes from Natural Earth (Washington Post)



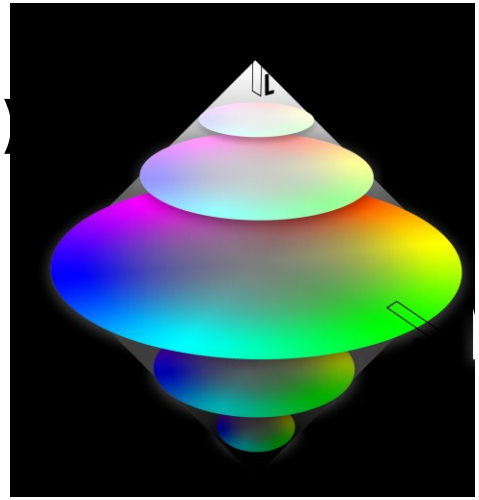
Coastlines and Boundaries



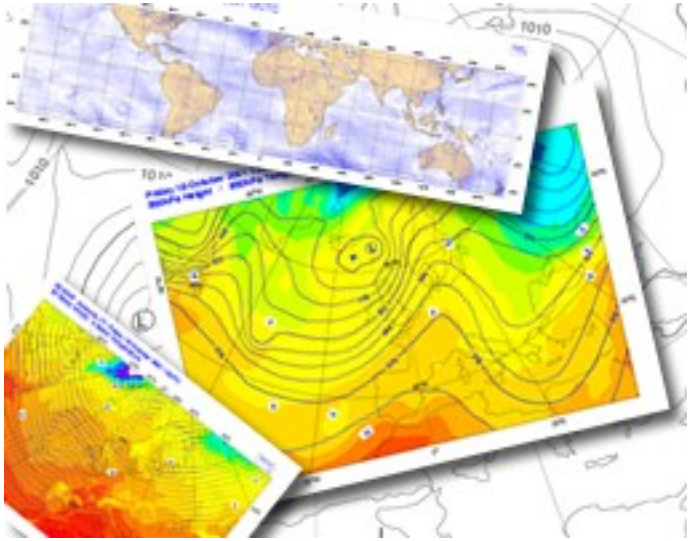

```
coast = mcoast(
            …
            map_coastline_resolution='high',
            map_boundaries='on',
            map_boundaries_colour='red',
            map_administrative_boundaries='on',
            map_administrative_boundaries_countries_list=['FRA', 'DEU', 'GBR'],
            map_administrative_boundaries_colour='orange',
            map_cities='on',
            map_rivers='on', )
```

How to set colours

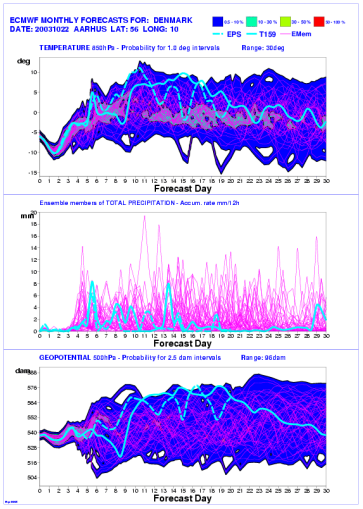
- Three ways of setting colours
- By name: “black”, “green”, “tangerine”
- RGB: “RGB(0.2 , 0.6 , 0.1)” (Red, Green & Blue components)
  - Also in hex notation: “#ff0000” (red)
  - Transparency expressed as rgba: “RGB(0.2 ,0.6,0.1,0.5)”
- HSL: “HSL(270.0 , 0.6 , 0.1)”
  (Hue, Saturation & Lightness)



Data Input



A bit more on data Input

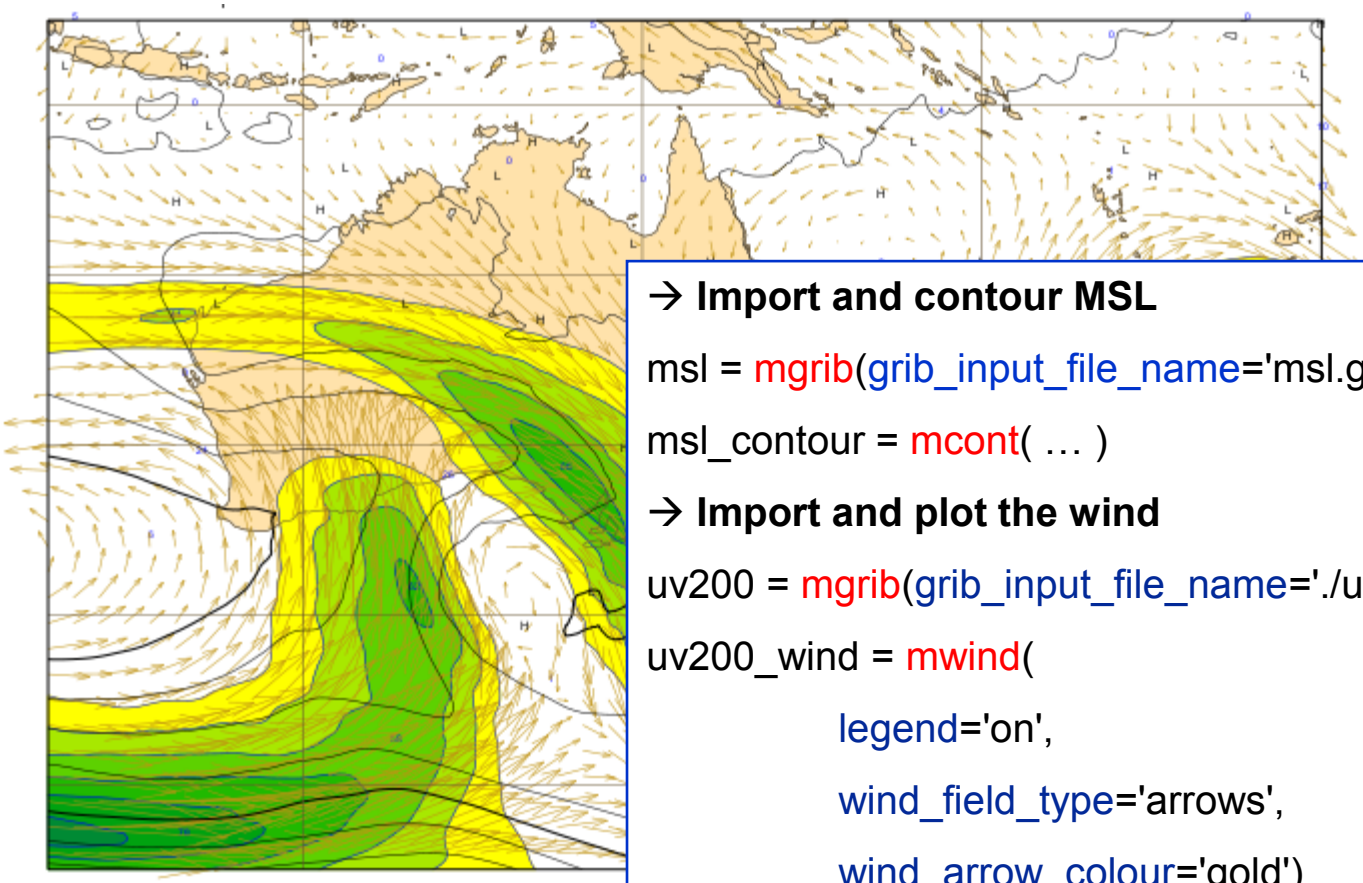


- Fields
  - GRIB 1 & 2, netCDF, arrays
  - Regular grid, Gaussian grid
  - Stretched and rotated grid (work in progress)
- Observations
  - BUFR
  - Observational DataBase (ODB)

How to deal with GRIB data : PGRIB

- Decode GRIB code
- Scale meteorological fields
- Setup stretched / rotated grids
- Set Magics contouring / wind / image parameters

Importing data example



→ **Import and contour MSL**

```
msl = mgrib(grib_input_file_name='msl.grb', grib_id='z500')
msl_contour = mcont( … )
```

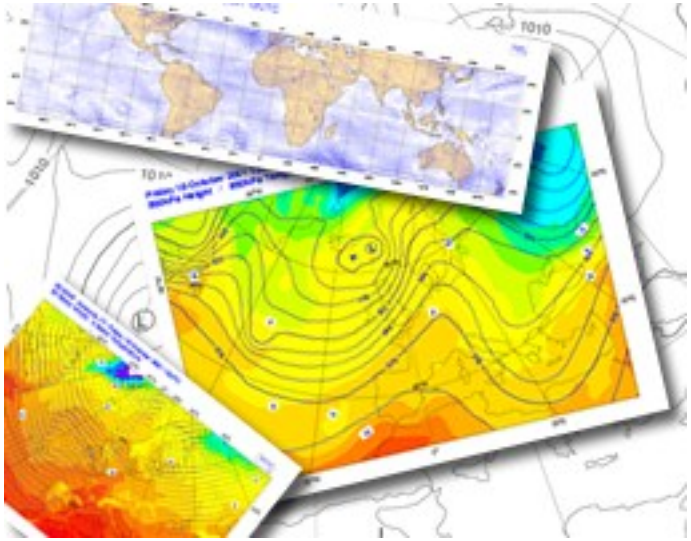
→ **Import and plot the wind**

```
uv200 = mgrib(grib_input_file_name='./uv200.grb', grib_id='uv200')
uv200_wind = mwind(
                legend='on',
                wind_field_type='arrows',
                wind_arrow_colour='gold')
```

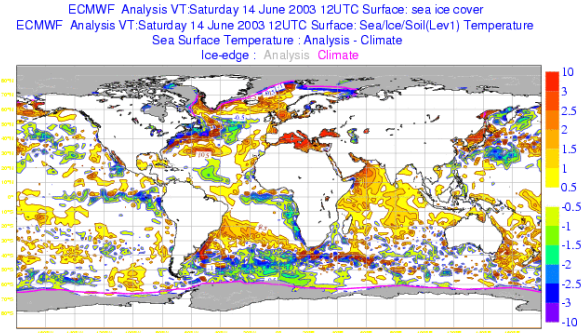
→ **Display these layers**

```
plot(…, uv200, uv200_wind, msl, msl_contour, …)
```

Contouring and shading



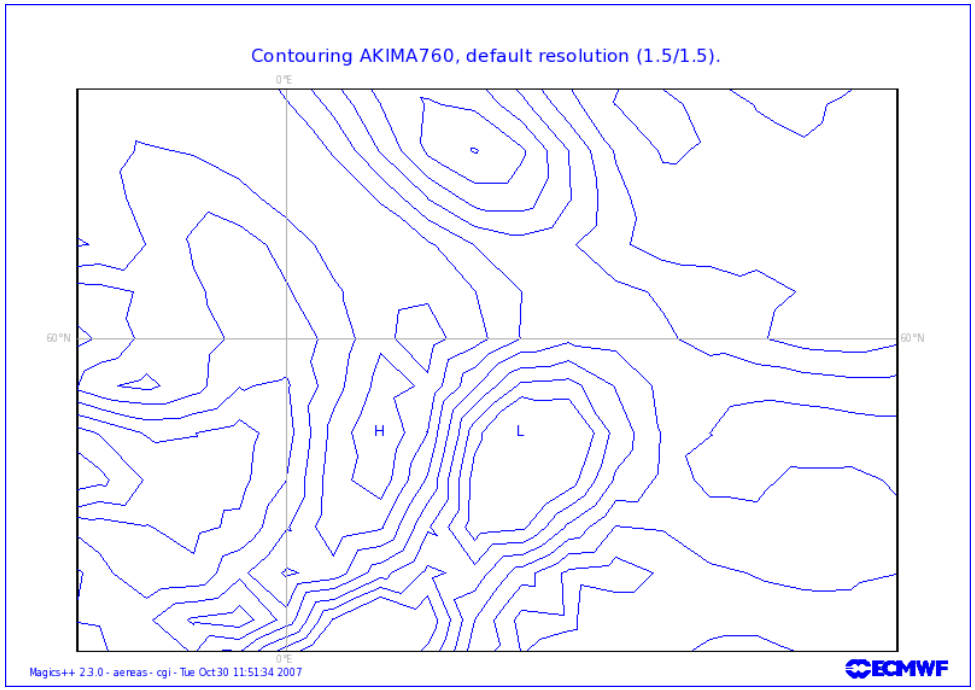
More on Contouring

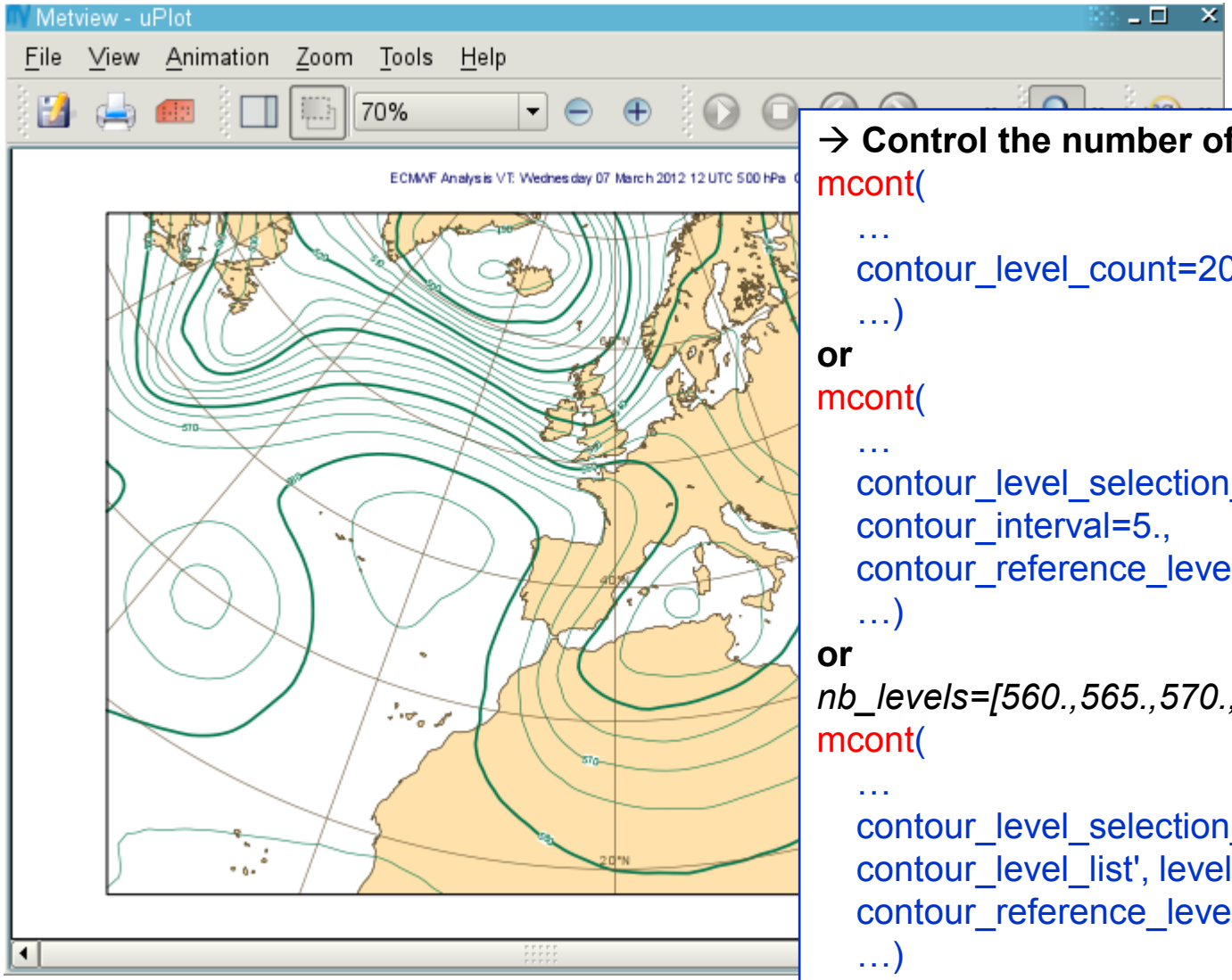


- Method
  - Akima + Linear (variable resolutions)
- Action Routine PCONT
  - Fortran: CALL PCONT
  - Python: mycont = mcont()
- Contour parameters
- Contour Shading

Contouring based Akima

- Algorithms developed by Hiroshi Akima - documented in the ACM Transactions on Mathematical Software
- INPE/CPTEC (Brazil) has successfully implemented a C++ version
- Accuracy is configurable by the user, although Magics++ will always choose sensible automatic values by default





→ **Control the number of isolines**

```
mcont(
   …
   contour_level_count=20,
   …)
```

or

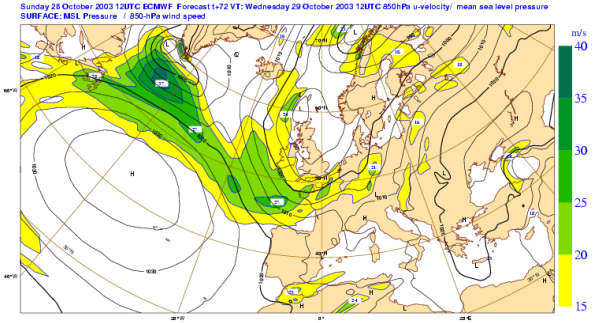
```
mcont(
   …
   contour_level_selection_type='interval',
   contour_interval=5.,
   contour_reference_level=560.,
   …)
```

or

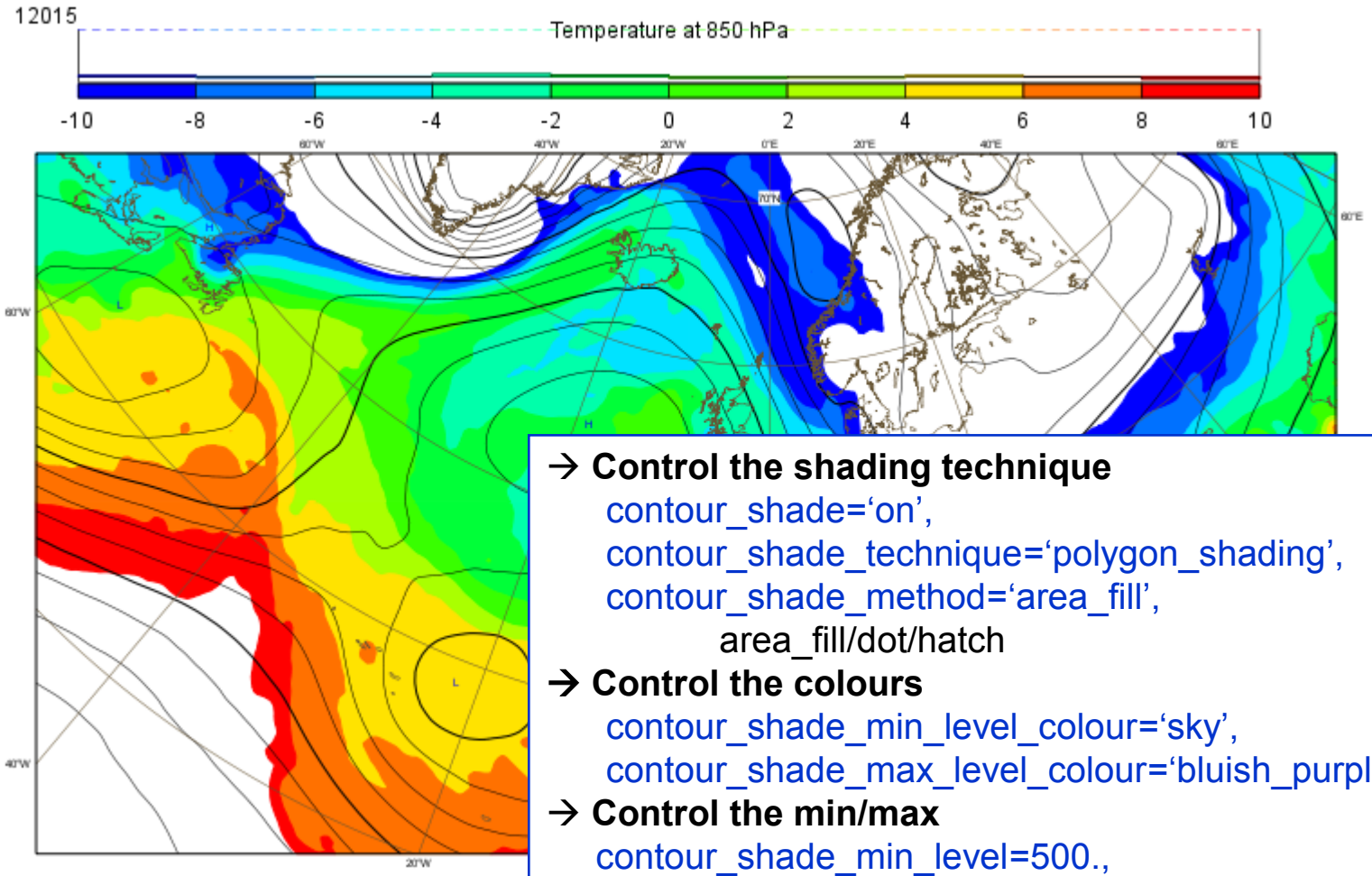
nb_levels=[560.,565.,570.,580.]

```
mcont(
   …
   contour_level_selection_type='level_list',
   contour_level_list', levels=nb_levels,
   contour_reference_level=560.,
   …)
```

More on contouring: shading



- **Shading techniques**
  - **Techniques:**
    - ➔ POLYGON
      - **DOT,**
      - **AREA_FILL,**
      - **HATCH**
    - ➔ CELL_SHADING
    - ➔ MARKER_SHADING
    - ➔ GRID_SHADING



→ **Control the shading technique**

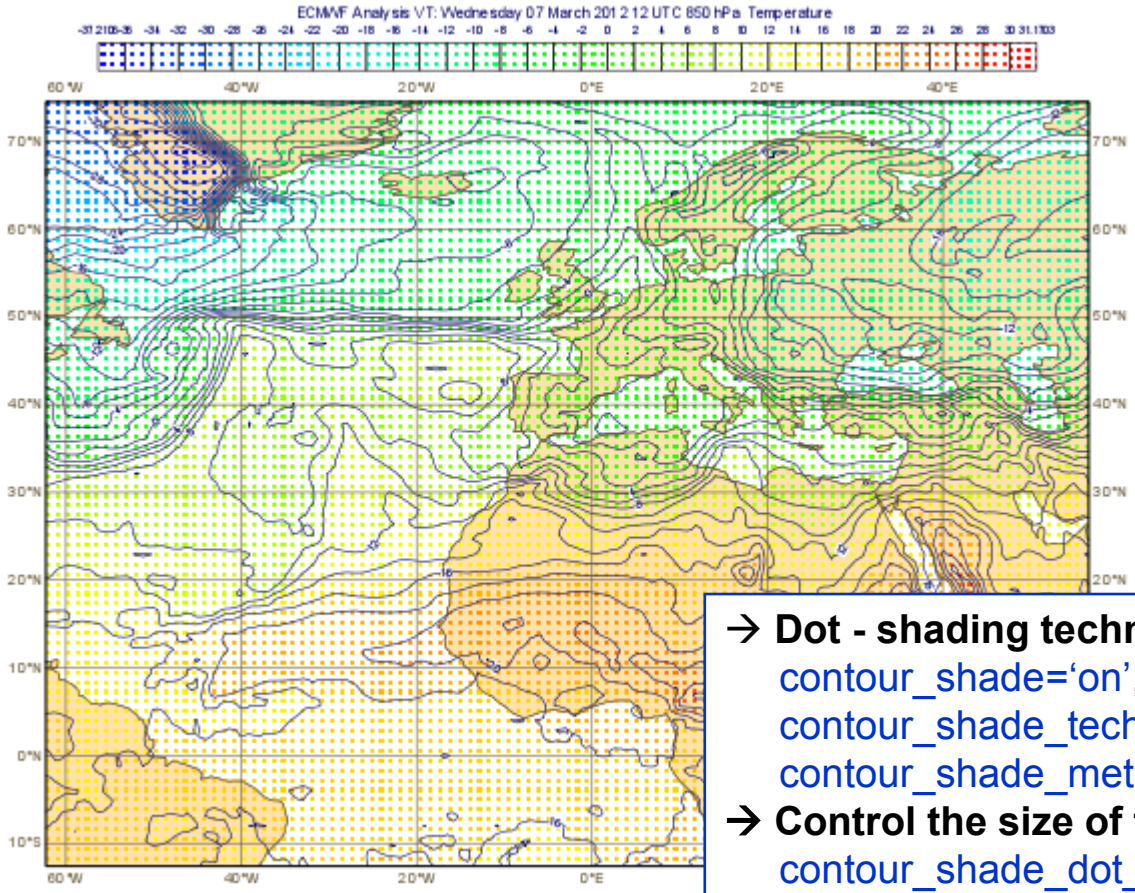
contour_shade='on',
contour_shade_technique='polygon_shading',
contour_shade_method='area_fill',
area_fill/dot/hatch

→ **Control the colours**

contour_shade_min_level_colour='sky',
contour_shade_max_level_colour='bluish_purple',

→ **Control the min/max**

contour_shade_min_level=500.,
contour_shade_max_level=560.,



→ **Dot - shading technique**

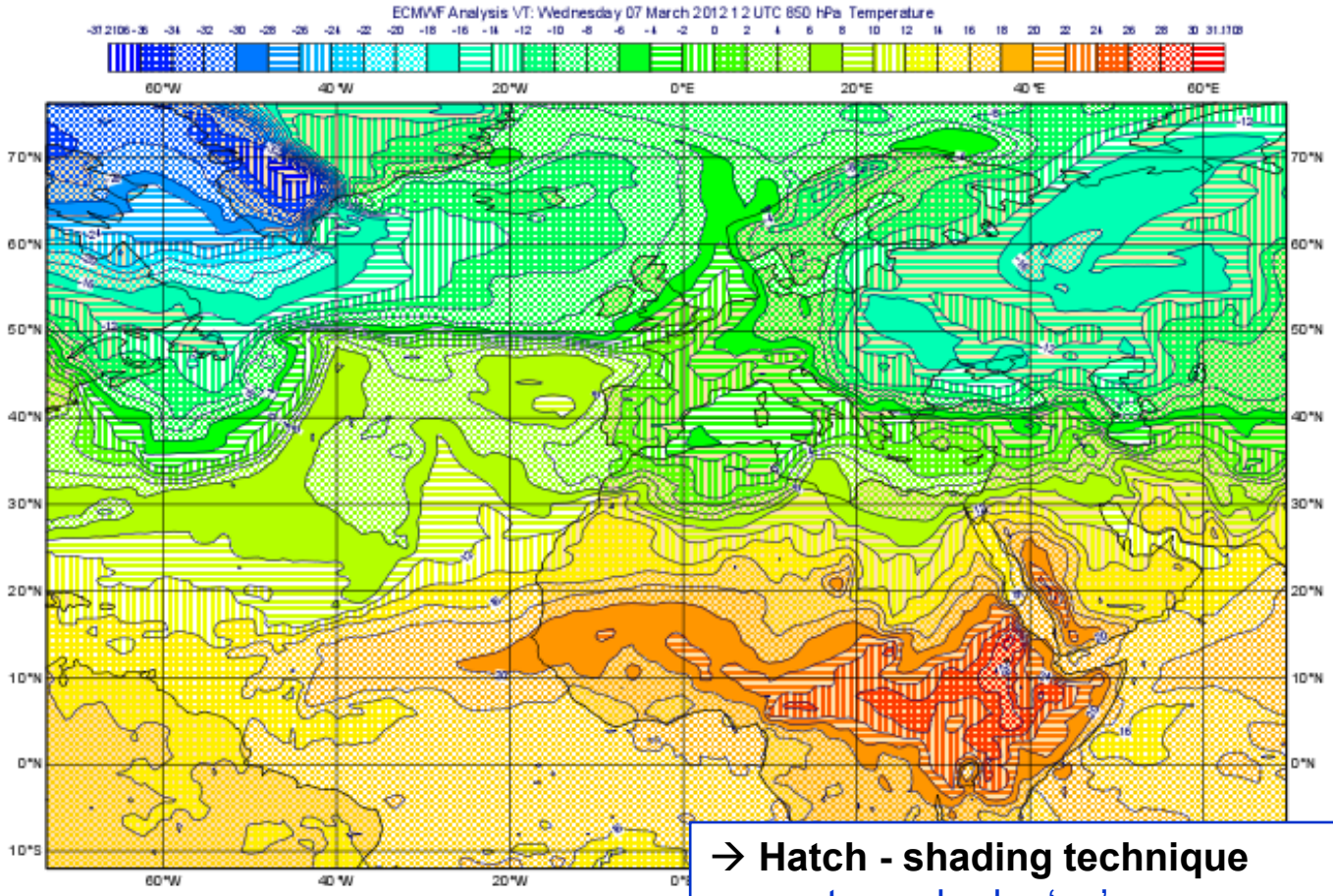
contour_shade='on',
contour_shade_technique='polygon_shading',
contour_shade_method='dot',

→ **Control the size of the dots**

contour_shade_dot_size=0.1,
contour_shade_min_level_density=25,
contour_shade_max_level_density=25,

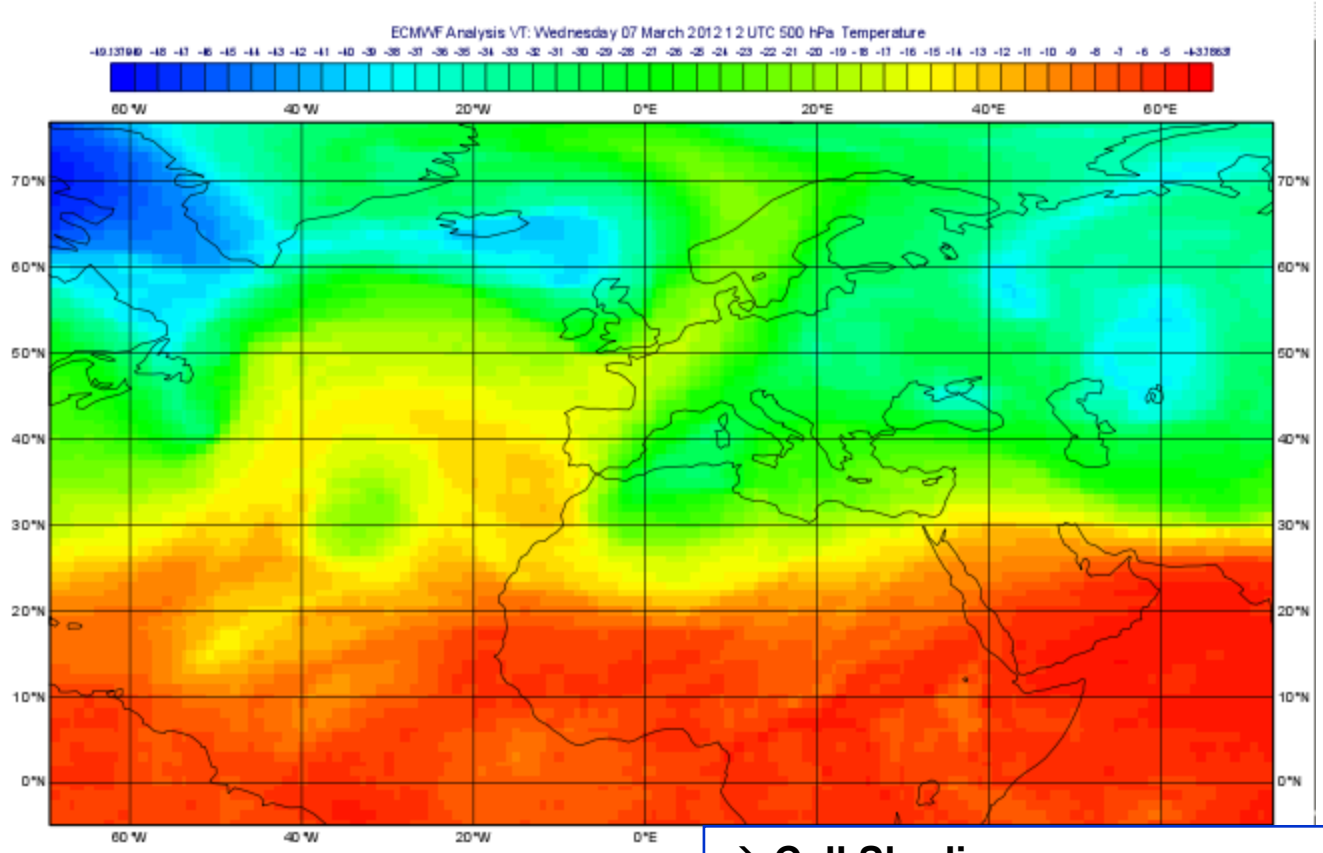
→ **Add a legend**

legend='on',
legend_display_type='continuous',



→ **Hatch - shading technique**

contour_shade='on',
contour_shade_technique='polygon_shading',
contour_shade_method='hatch',

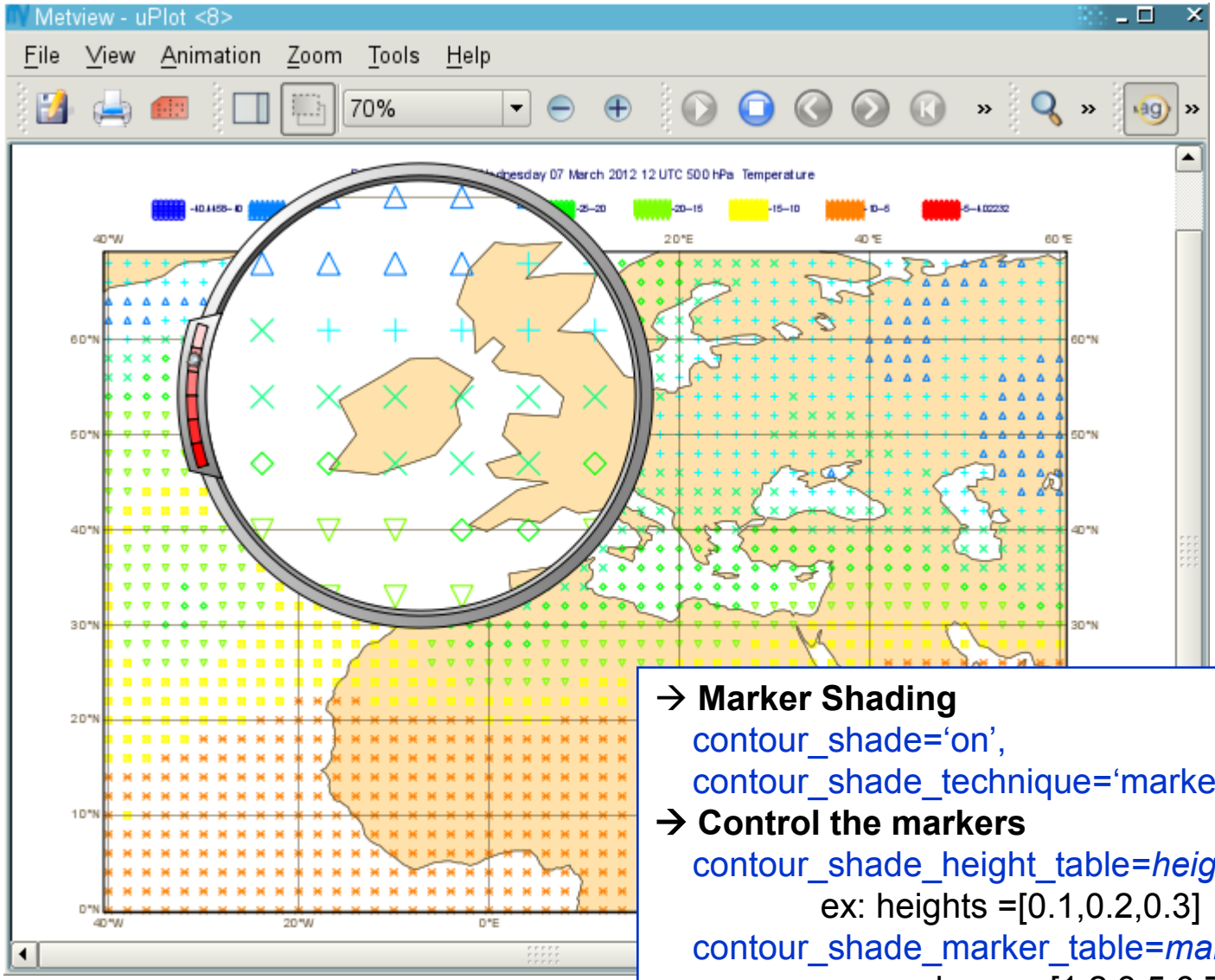


→ **Cell Shading**

contour_shade='on',
contour_shade_technique='cell_shading',

→ **Control the resolution**

contour_shade_cell_resolution=5.,



→ **Marker Shading**

contour_shade='on',

contour_shade_technique='marker',

→ **Control the markers**

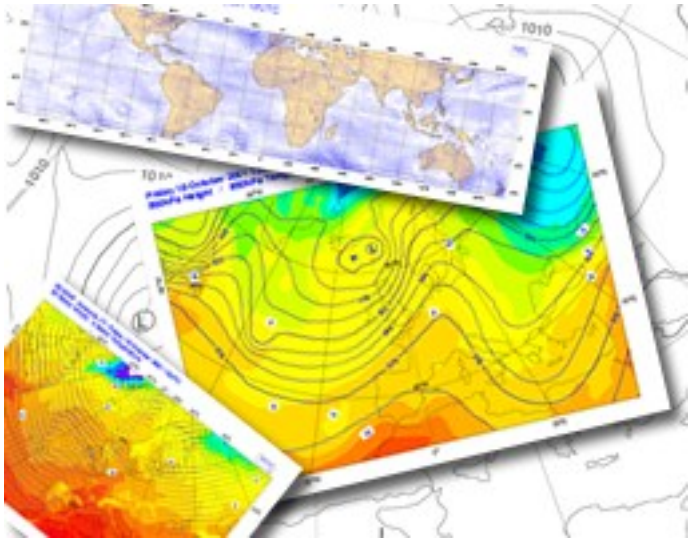
contour_shade_height_table=*heights*,

ex: heights =[0.1,0.2,0.3]

contour_shade_marker_table=*markers*

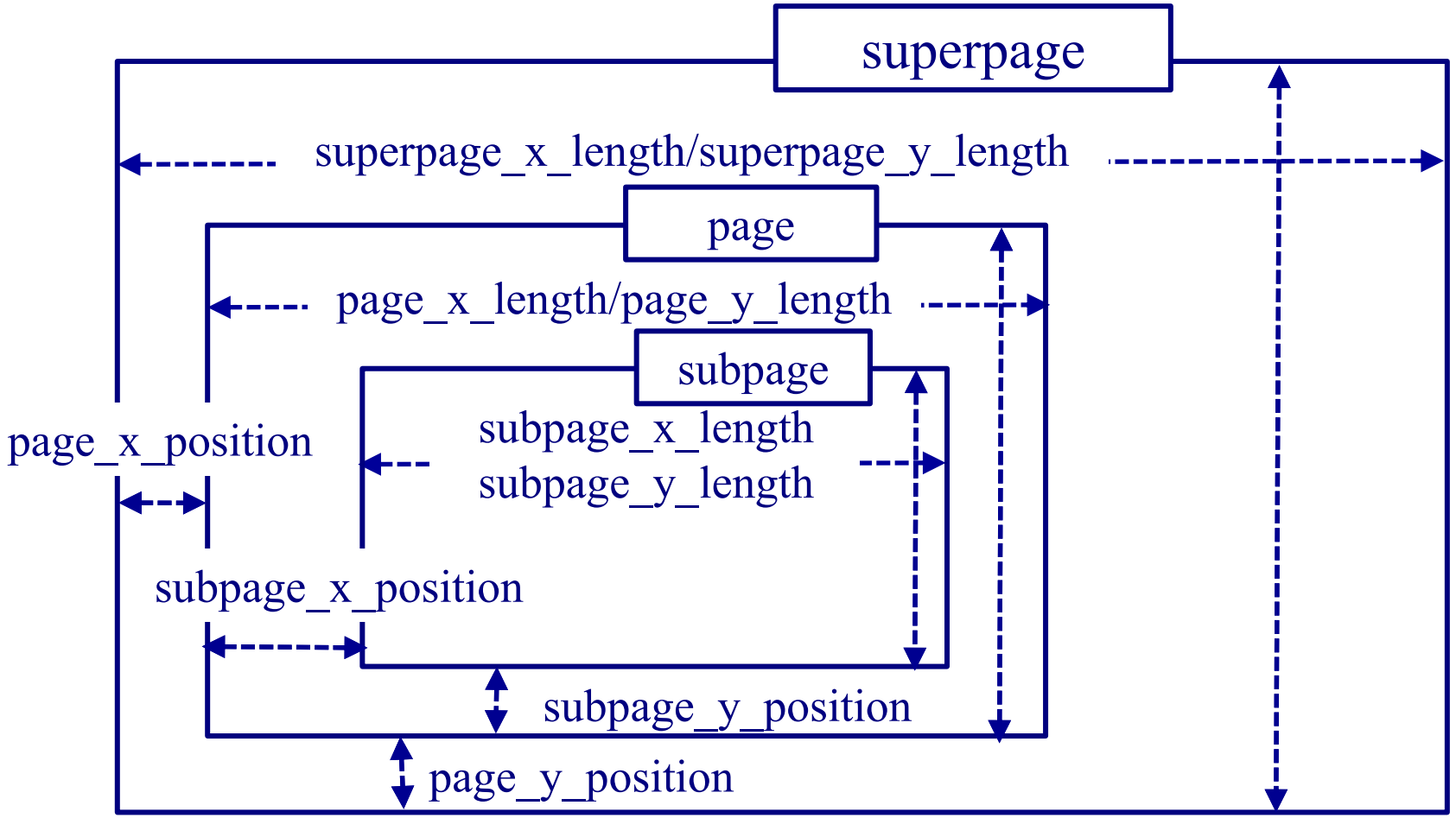
ex: markers = [1,2,3,5,6,7,8,9,10]

Layout – Text - Legend



Layout

- 3 main concepts : superpage/page/subpage



Layout → things to remember

- **The layout and projection settings are done during the call to the first action-routine.**
- **The instantiation of a page object will create a new page.**

```
page(page_x_length =14., page_y_length=10.,
     page_x_position=15., page_y_position=10.5,
     subpage_x_position=0.5,  subpage_y_position=0.5,
     subpage_x_length=12.,  subpage_y_length=8.)
```

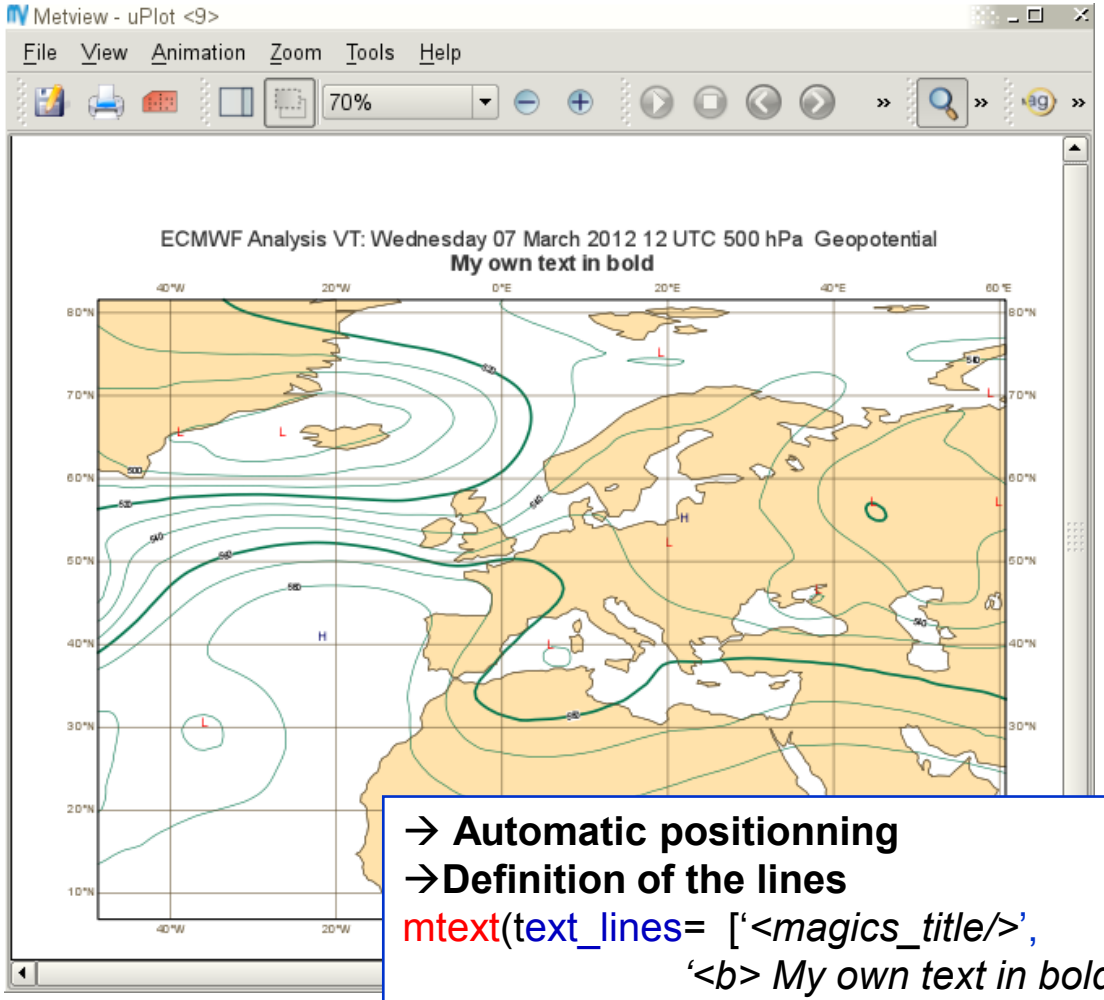
- **The dimensions are in cm.**
- **The subpage is the area where the plot will be rendered: There is always a projection attached to it : geographical or Cartesian.**

Text setting and formatting

- The position of a the text can be
  - automatic (attach to the top of the subpage)
  - positional (text_x_position, text_y_position text_x_length, text_y_length)
- The text is passed as an array of strings.

Text setting and formatting

- Basic html formatting can be used
  - My text : bold
  - My text : colour and font-size
  - _{My text}: subscribed
  - ^{My text}: superscripted
- Some tags allow to extract and use metadata from the grib headers (using the grib_api keys, or magics_specific keys)
  - <grib_info key='param'/> (name, base_date valid_date)
  - <magics_title/> will create an automatic title.



→ **Automatic positionning**

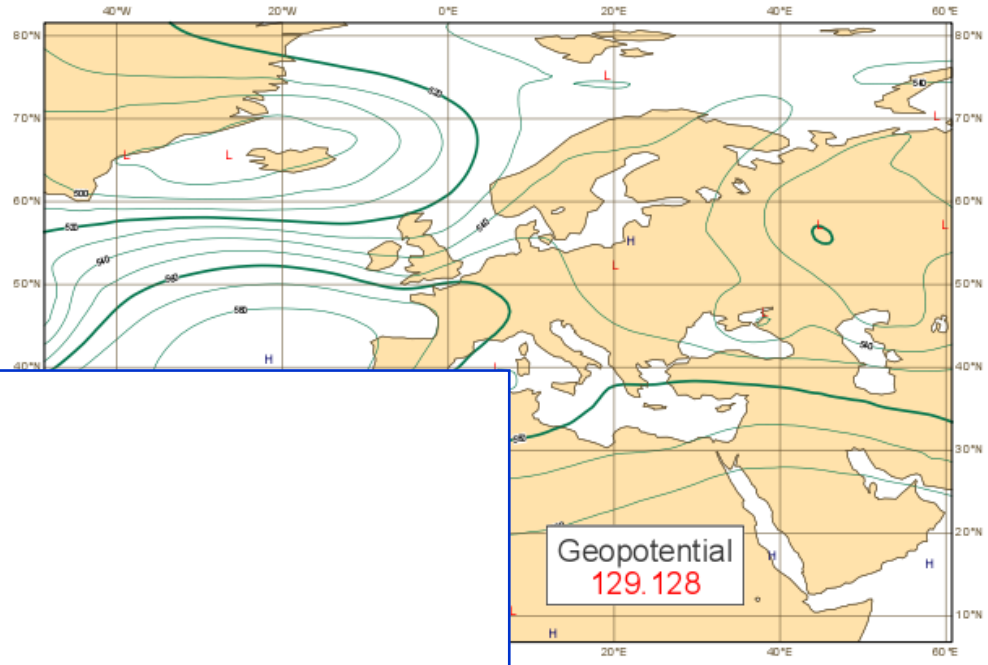
→**Definition of the lines**

mtext(text_lines= [‘*<magics_title/>*’,

‘ My own text in bold ’])

automatic title build from the GRIB header

user text with basic HTML formatting



→**User-defined position**

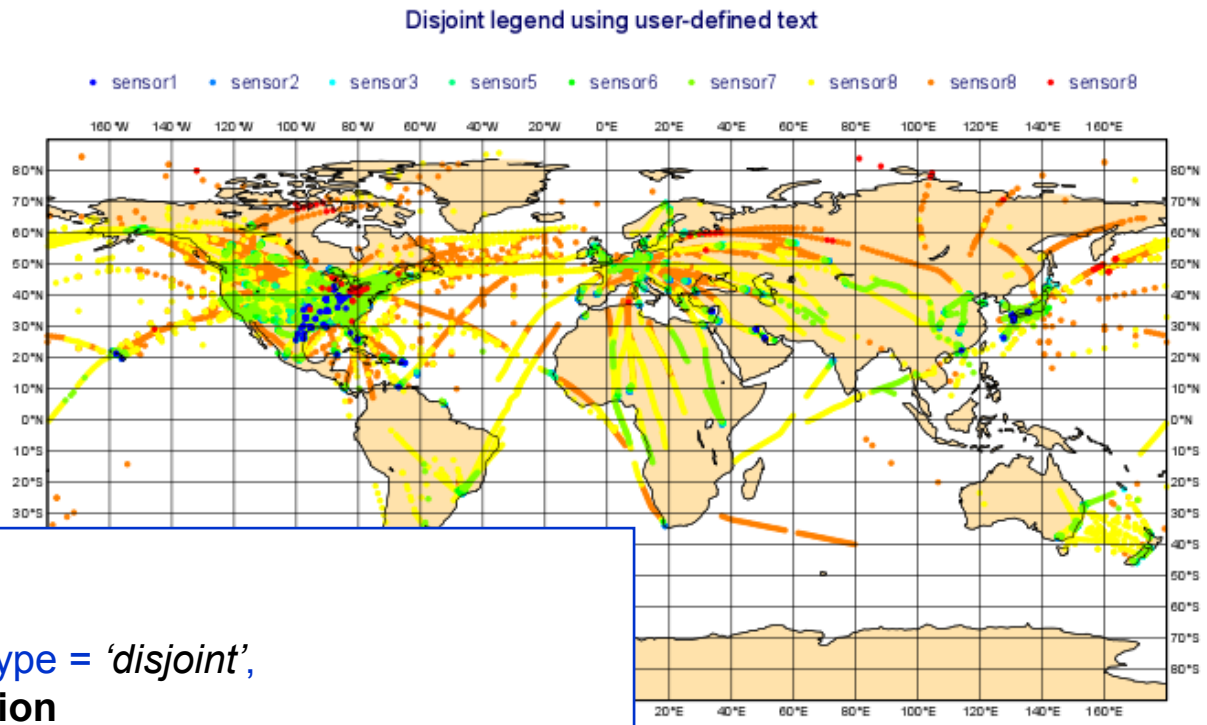
mtext(text_mode= *'positional'*,
text_box_x_position= *15.*,
text_box_y_position= *2.*,
text_box_x_length= *5.*,
text_box_y_length= *2.*,
dimensions in cm from the bottom-left corner
text_box_blanking= *'on'*,
text_border= *'on'*,
text_border_colour= *'charcoal'*,

→**Definition of the lines**

text_lines= [*'<grib_info key="name"/>'*,
extract information from the GRIB header using GribAPI keys
* '<grib_info key="param"/>'*']
adding basic HTML formatting

Legend

- **As a text, the legend (legend_text_mode) can be :**
  - **automatic**
    - ➔ attach to the top of the subpage, below the automatic title
  - **positional**
    - ➔ legend_x_position/legend_y_position /legend_x_length/legend_y_length
- **The legend (legend_display_type) can be**
  - **disjoint**
  - **continuous**
  - **histogram**



→Legend definition

mlegend(legend = *'on'*,

legend_display_type = *'disjoint'*,

→User-defined text definition

legend_text_composition= *'user_text_only'*)

legend_user_lines= *['sensor1',…,sensor8']*

legend_text_font_size=*0.4*,

legend_text_colour= *'navy'*,

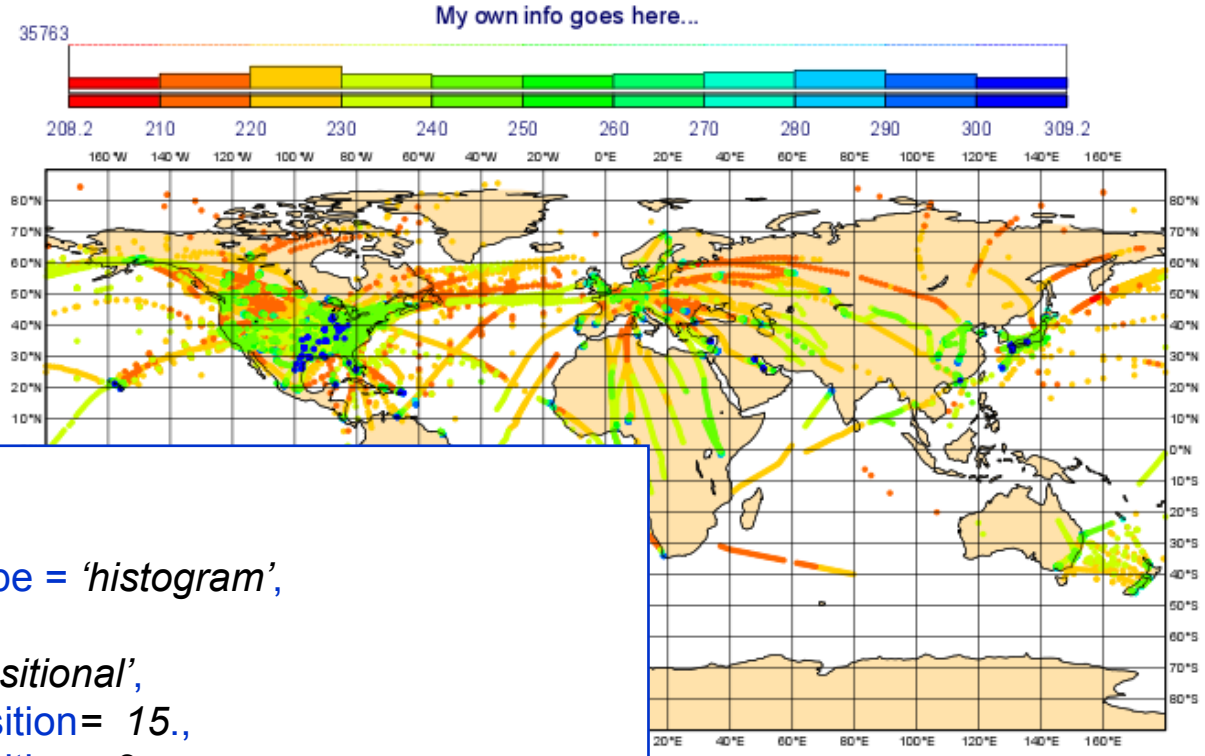
→Definition of the title

legend_title'= *'on'*,

legend_title_text' =

'Disjoint legend …')

using basic HTML formatting



→**Legend definition**

mlegend(legend =*'on'*,

legend_display_type = *'histogram'*,

→**User-defined position**

legend_mode= *'positional'*,

legend_box_x_position= *15.*,

legend_box_y_position= *2.*,

legend_box_x_length= *5.*,

legend_box_y_length= *2.*,

dimensions in cm from the bottom-left corner

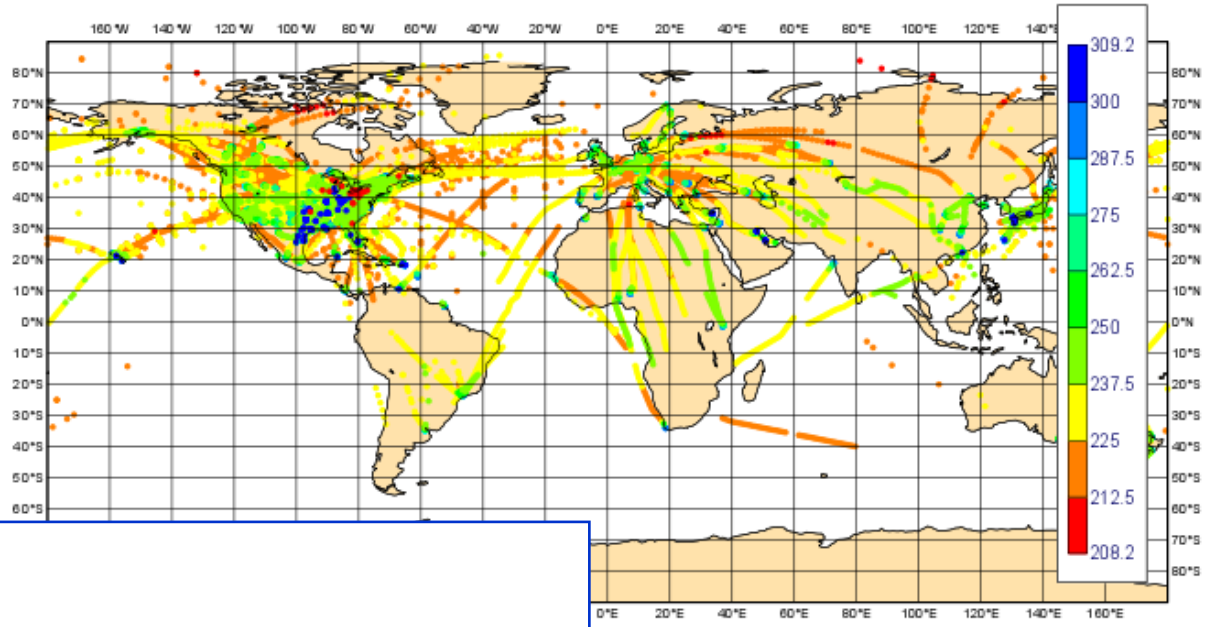
→**Definition of the title**

legend_title= *'on'*,

legend_title_text' =

'My own info goes here…')

using basic HTML formatting



→**Legend definition**

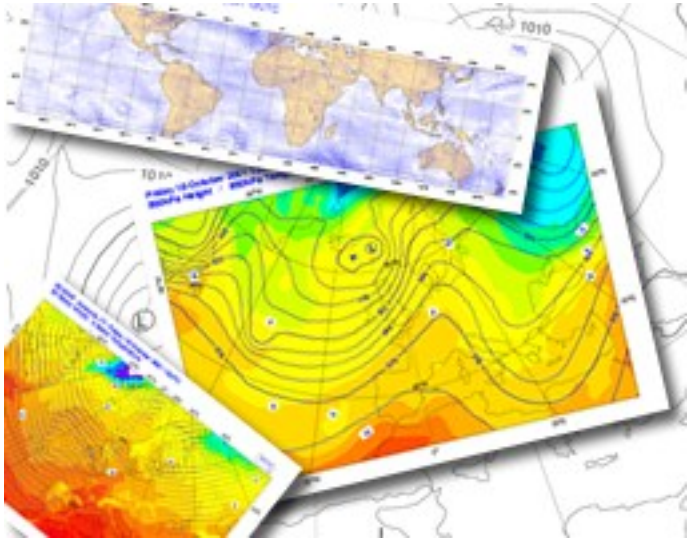
mlegend(legend = *'on'*,
legend_display_type = *'continuous'*,

→**User-defined position (vertical)**

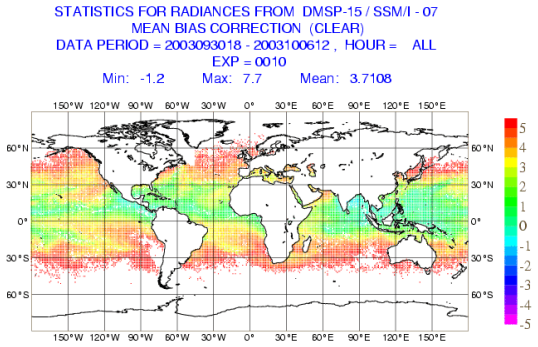
legend_mode= *'positional'*,
legend_box_x_position= *25.*,
legend_box_y_position= *1.5*,
legend_box_x_length= *2.*,
legend_box_y_length= *13.*)

Dimensions in cm from the bottom-left corner
The number of rows and columns can be adjusted.

Magics outputs



Working with Magics output

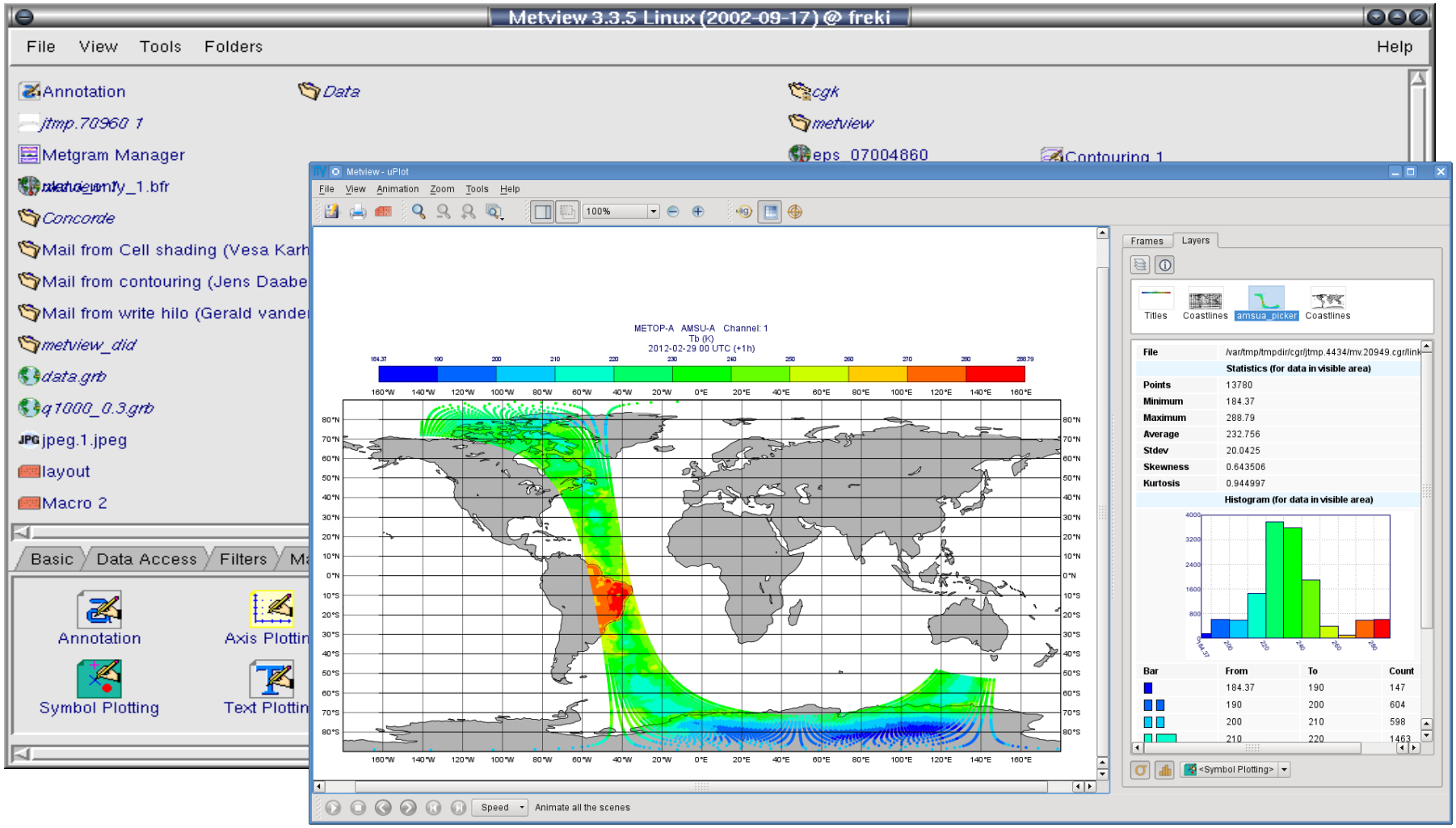


- Magics has many output formats:

  (depending on the installation)

  - Vector formats → PostScript (for printers), EPS, PDF, SVG
  - Raster formats → PNG
  - Animated formats → Qt for Metview, KML for Google Earth
- PostScript is the default and most reliable output format
- PNG, as a raster format, has not as good quality as vector formats
- SVG is a web vector format which is becoming increasing popular (HTML5)
- Some formats only support transparent colours

Magics Qt output → Metview



Viewing and processing Magics output

- The output files can be viewed with:

  PostScript : Ghostview (*gv*), *okular*

  PNG : *display*, web browser

- Files (especially PostScript) can be easily compressed and uncompressed by *gzip* and *gunzip*. This can save a lot of disk space and speeds up network transfers (email)!!!
- Output files can be processed by many standard tools, such as *xv*, *display* and any web browser.
- Especially powerful is a free package called *ImageMagick*. It contains many tools to process images.

http://www.imagemagick.org

Magics++ - defining output

- **File naming:**

  Fortran: CALL PSETC ('OUTPUT_NAME', 'example_name')

  Python: output(output_name='example_name')

  will result in names as *example_name.ps* **or** *example_name.1.png***. The file extension is added automatically by the driver.**

- **Formats which have one page per file (like PNG) get also a number:**

  wind.1.svg, wind.2.svg

Magics++ - multiple output (Python)

- **Single file format:**

  `output(output_format='png')`

- **Multiple formats at the same time:**

  `output(output_formats=['png','ps'])`

→ Very efficient to let Magics do it – data processing only performed once!

Magics++ - multiple output (Fortran)

- **Single file format:**

```
CALL PSETC ('OUTPUT_FORMAT', ‘PNG')
```

- **Multiple formats at the same time:**

```
CHARACTER*10 FORMATS
DIMENSION FORMATS(2)
DATA FORMATS /'PS', ‘PNG'/
…
CALL PSET1C ('OUTPUT_FORMATS', FORMATS, 2)
```

Magics++ - multiple output (MagML)

- **Single file format:**

```
<drivers>
        <PNG output_fullname = “mytest.png”>
</drivers>
```

- **Multiple formats at the same time:**

```
<drivers>
        <png output_fullname = “mytest.png” />
        <kml output_fullname = “othertest.kmz” />
</drivers>
```

Defining output size

- **For most drivers size is set through**

  superpage_x_length & superpage_y_length

  (default: A4 landscape)

- **For some raster drivers (i.e. PNG)**

  output_width (default: 800 pixels)

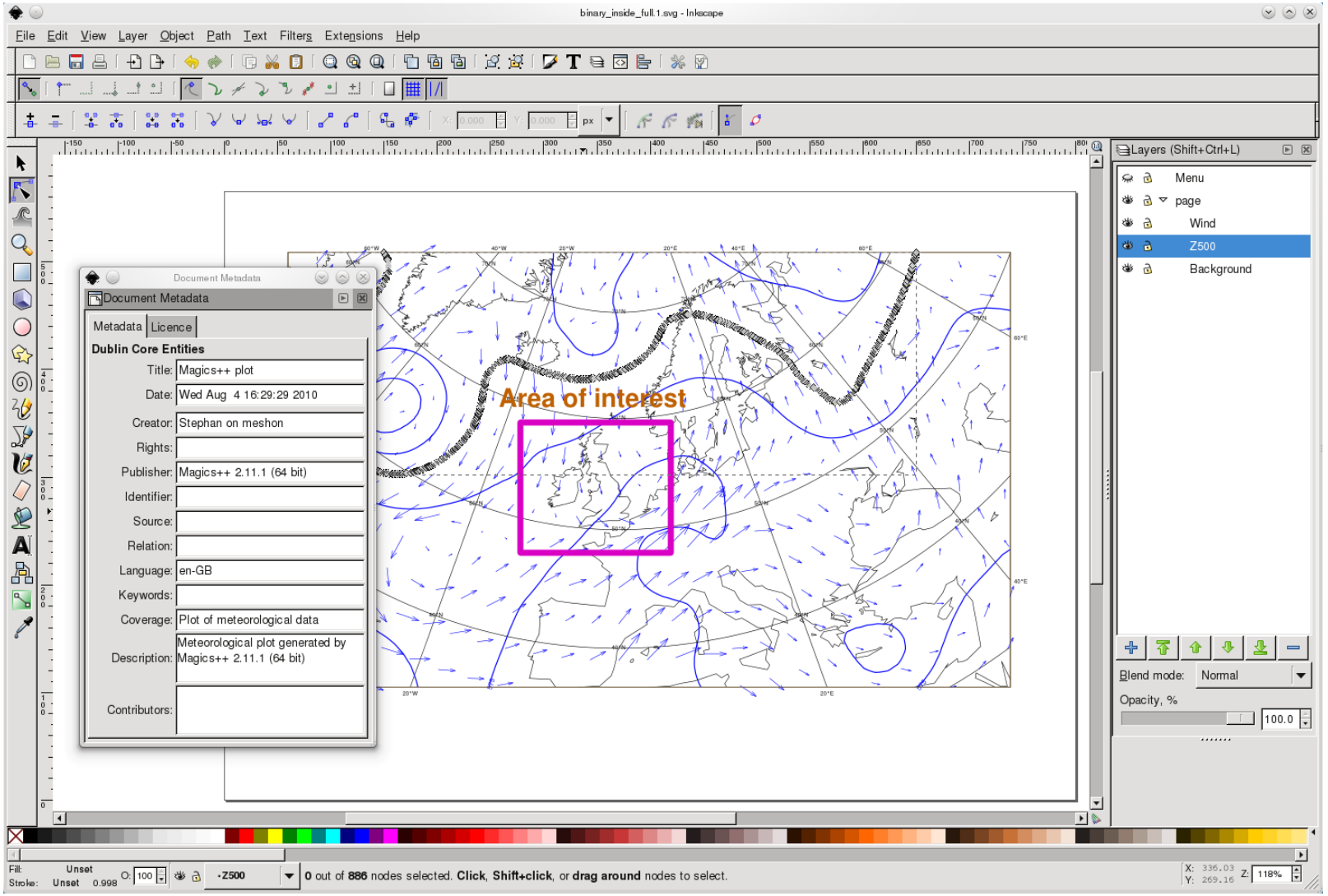
  (where the ratio of superpage gives height)

- **KML is very special!**
  - **Does not have an output size**
  - **Only works when Cylindrical projections is set!**

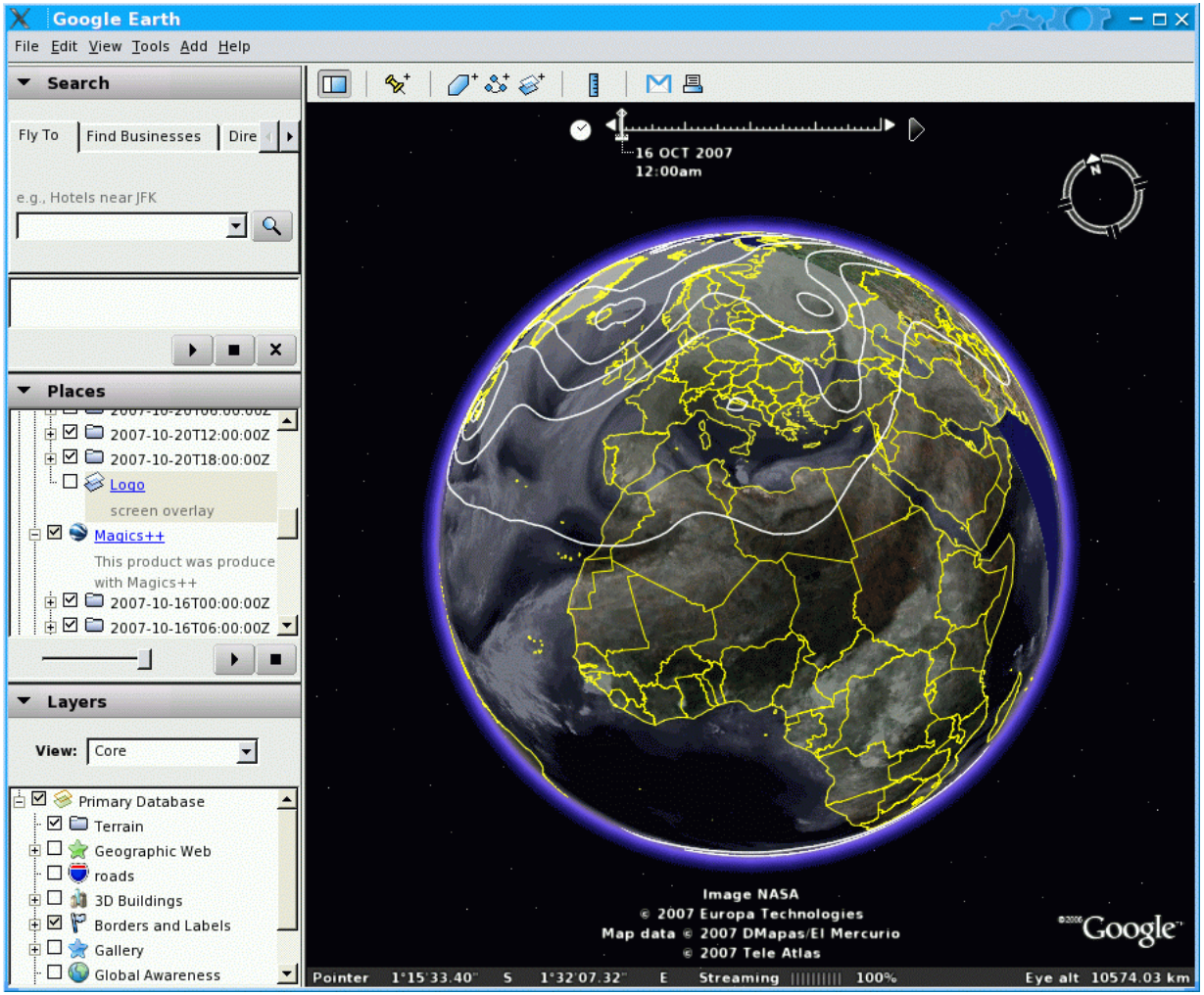
Magics++ - new formats

- EPS/PDF
  - Addition to PostScript output to support inclusion in documents (Word, Latex) and on the web
- SVG
  - Vector format for web/printing (HTML5)
  - Human readable and supports interactivity
- KML
  - For use in Google Earth and Google Maps
  - Still in the beginning (no legend, no wind flags/arrows)

Magics++ : SVG output

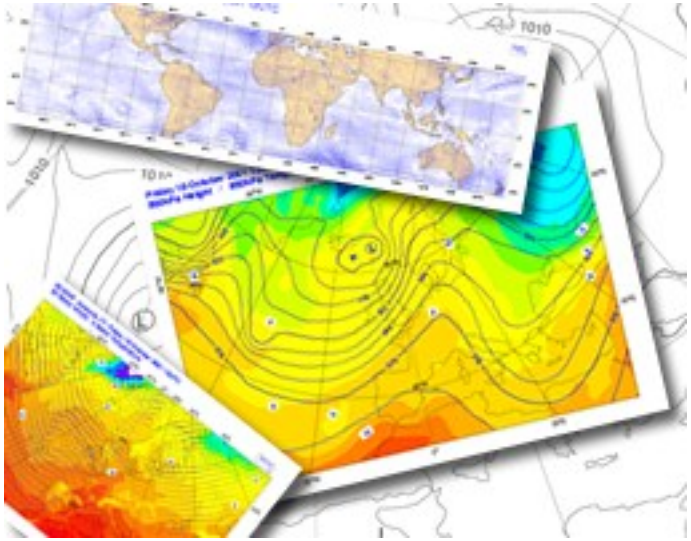


Magics++ : KML output

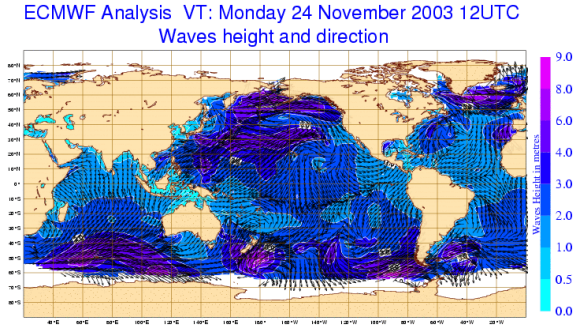


(Works not for all types of data/projections)

Wind Plotting



More on Wind Plotting

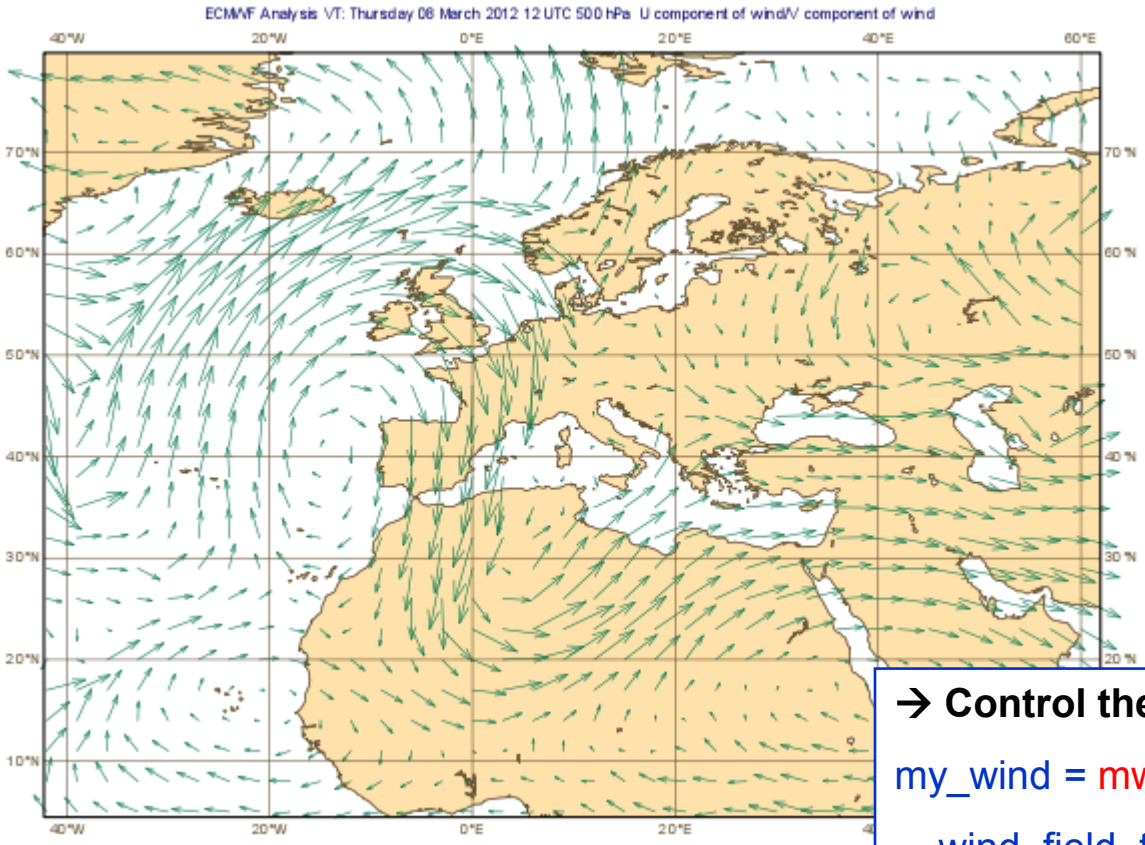


- **Wind field :**
  - ♦ U/V velocity components
  - ♦ Speed and direction
  - → Set mode through ‘GRIB_WIND_MODE’
- **Wind plotting**
  - ♦ Wind arrows
  - ♦ WMO standard wind flags
  - → Set type through ‘WIND_FIELD_TYPE’

More on Wind plotting

- The projection of wind fields is done automatically
- There is no interpolation.
- A thinning process may be applied
- User has control over:
  - Colour
  - Thickness
  - Minimum/maximum speed
  - Calm indicator

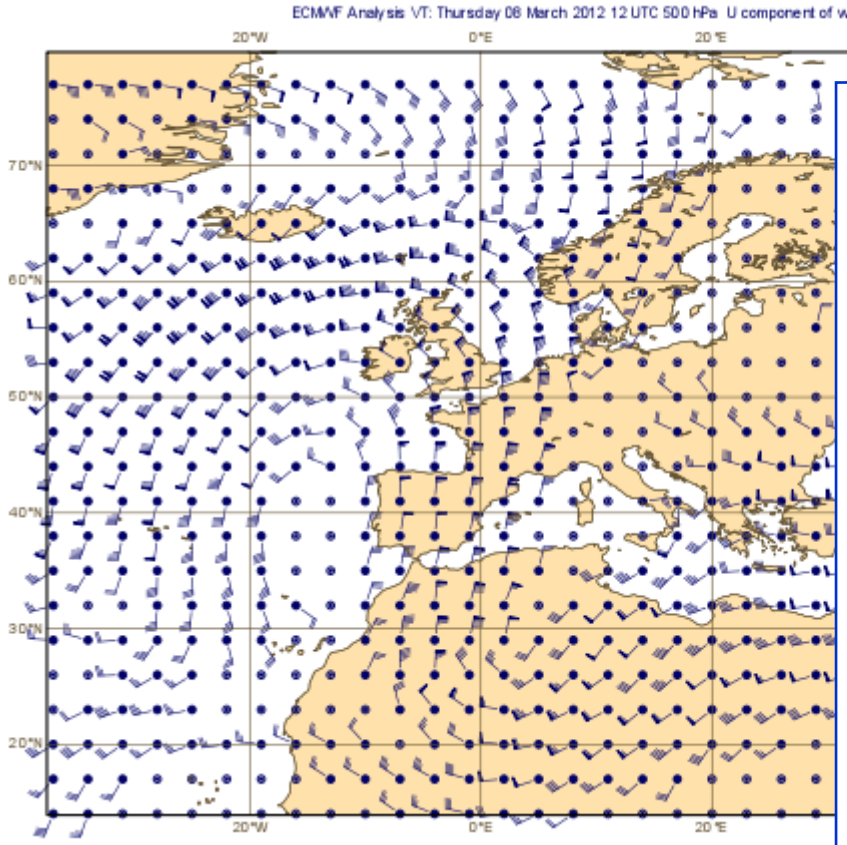
Wind arrows example



→ **Control the wind plotting**

```
my_wind = mwind(
    wind_field_type='arrows',
                **arrows**/flags
    wind_arrow_colour='evergreen')
```

Wind flag example



→ **Control the wind flags**

```
my_wind = mwind(
    wind_field_type='flags',
    wind_flag_colour='navy',
    wind_flag_thickness=1.5,
    wind_flag_origin_marker='dot',
```

→ **Control some global attributes**

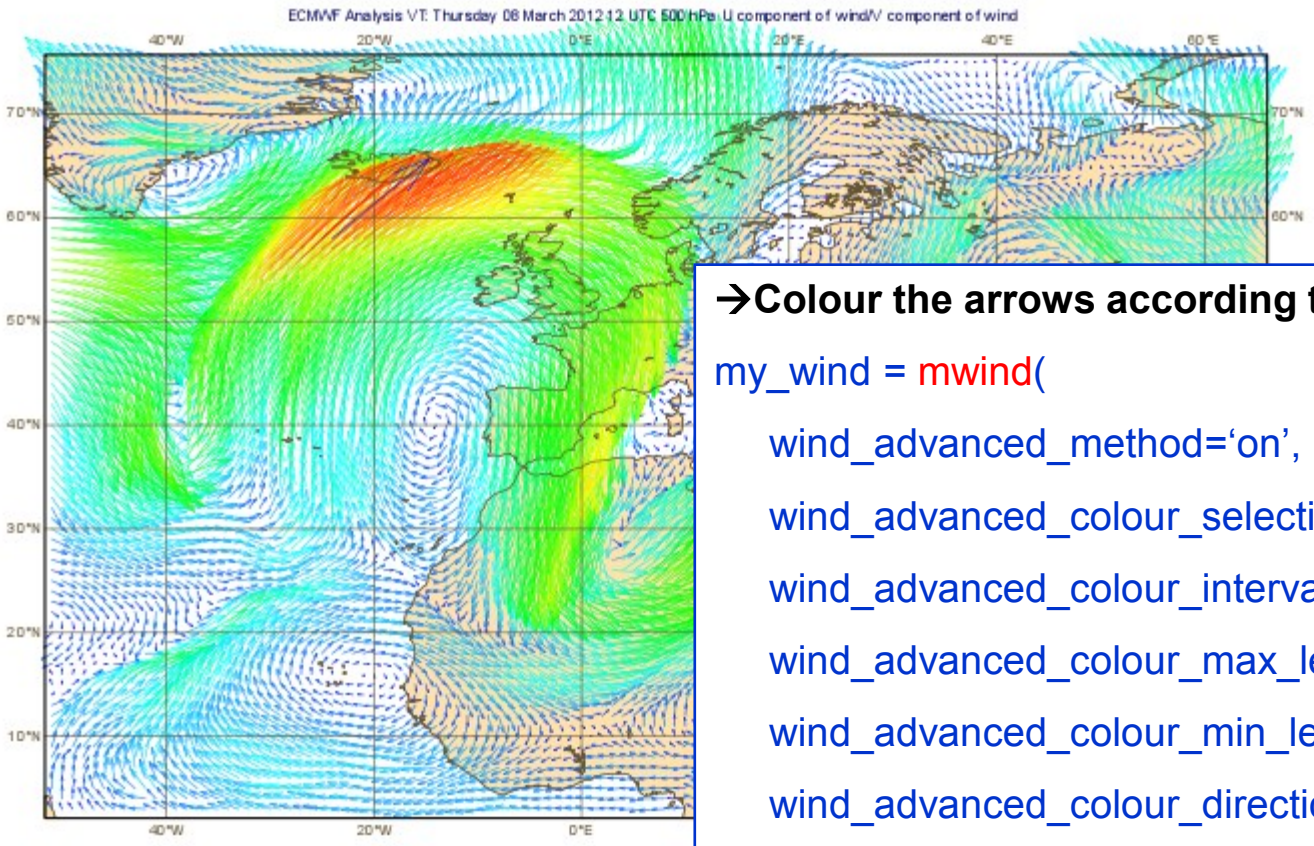
```
    wind_thinning_factor=3.,
```

Control calm wind and min/max (m/s)

→ **Values are given in m/s BUT displayed in knots (according to WMO standard)**

```
    wind_flag_calm_below=1.5,
    wind_flag_min_speed=1.,
    wind_flag_max_speed=10.)
```

Advanced wind plotting



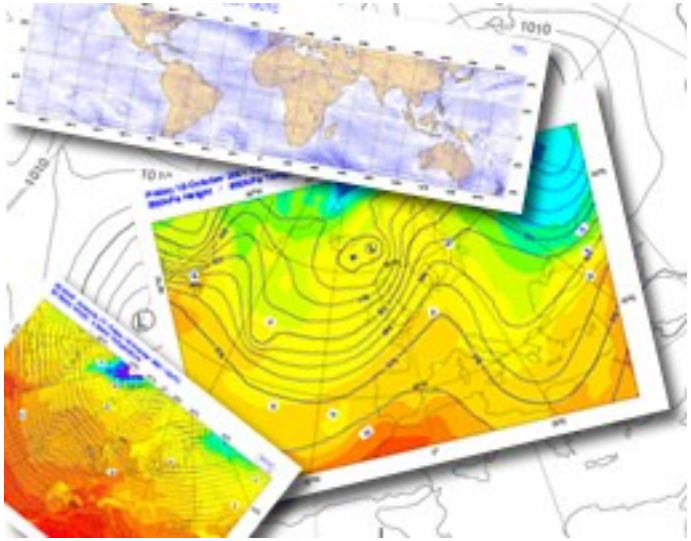
→Colour the arrows according to the wind speed

```
my_wind = mwind(
    wind_advanced_method='on',
    wind_advanced_colour_selection_type='interval',
    wind_advanced_colour_interval=2.,
    wind_advanced_colour_max_level_colour='red',
    wind_advanced_colour_min_level_colour='blue',
    wind_advanced_colour_direction='clockwise',
```

→Control some global attributes

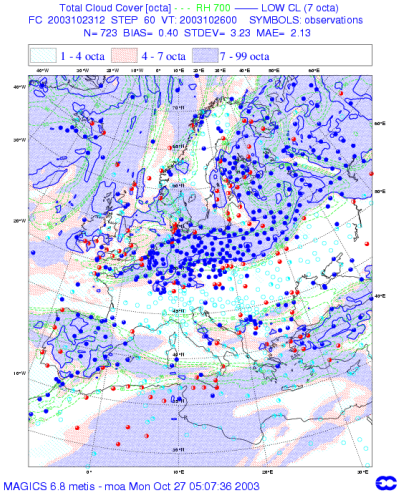
```
    wind_arrow_unit_velocity=25.,
    wind_thinning_factor=1.)
```

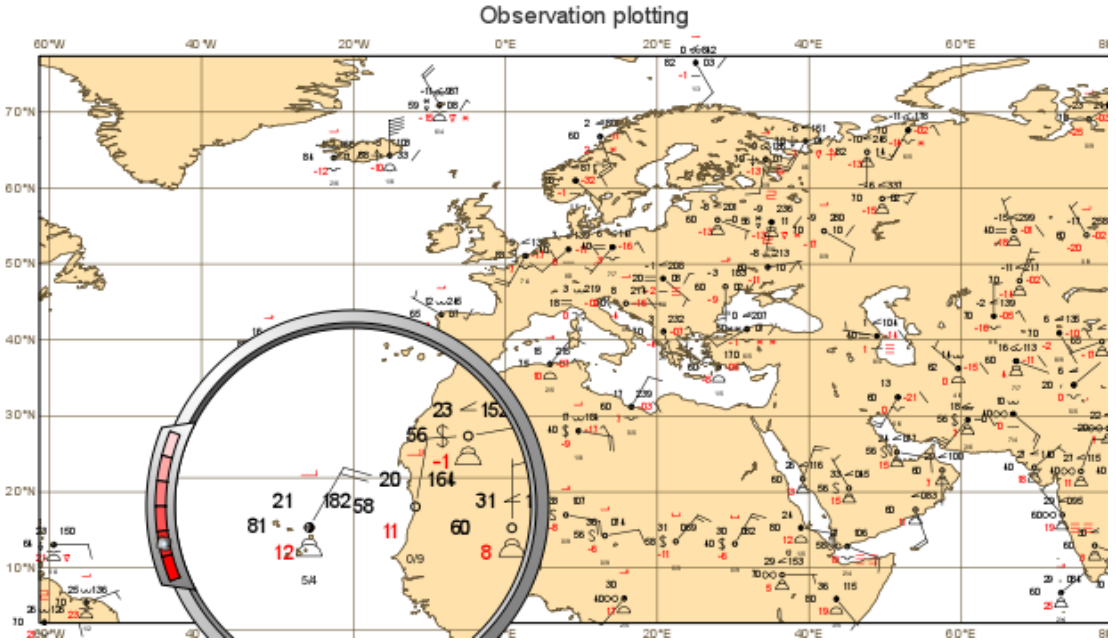
Observations and symbol plotting



More on Observations

- Magics++ supports basic observation plotting
  - Synop
- WMO plotting standards.
- WMO BUFR code

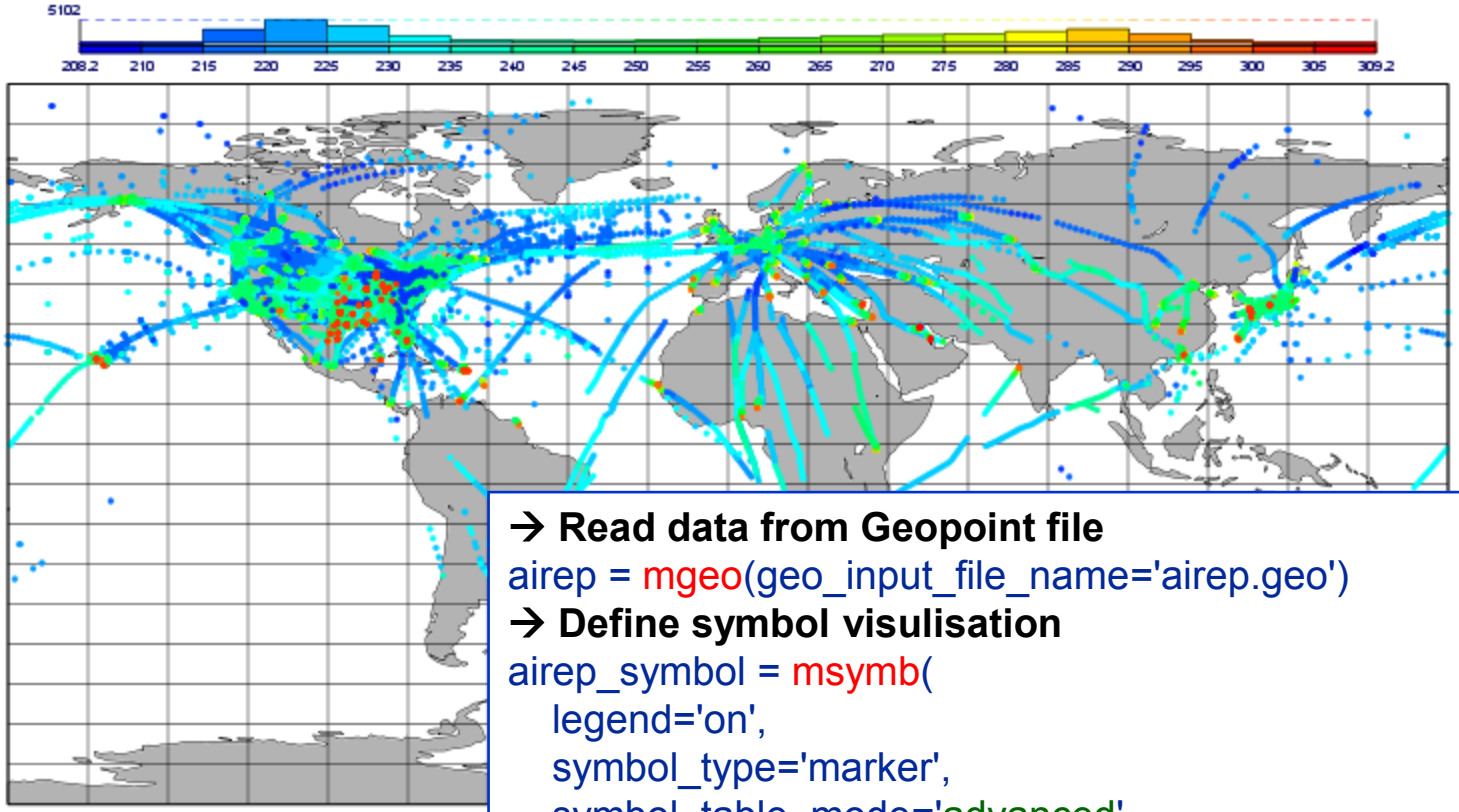




Symbol plotting

- Magics has many options to plot symbols representing data, such as observations
- Magics++ improves the handling of large amounts of data coming from satellites stored in ODB
- Valid inputs:
  - **Arrays of values**
    - → x_values/y_values
  - **Geopoints files**
    - → Ascii files (Metview)
  - **ODB**

Monitoring of airep data



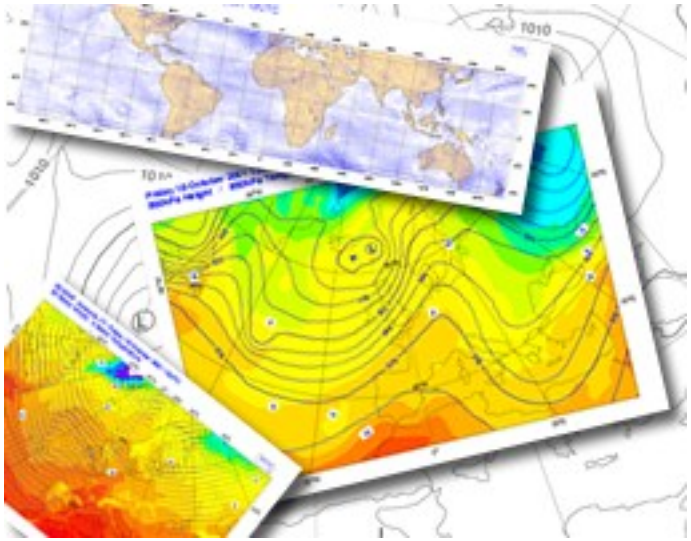
→ **Read data from Geopoint file**

```
airep = mgeo(geo_input_file_name='airep.geo')
```

→ **Define symbol visulisation**

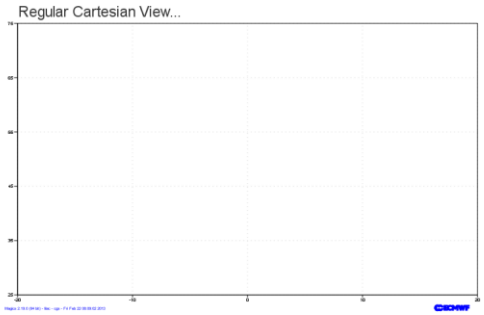
```
airep_symbol = msymb(
    legend='on',
    symbol_type='marker',
    symbol_table_mode='advanced',
    symbol_advanced_table_selection_type='interval',
    symbol_advanced_table_interval=5.,
    symbol_advanced_table_min_level_colour='blue',
    symbol_advanced_table_max_level_colour='red',
    symbol_advanced_table_colour_direction='clockwise',
    symbol_marker_index=15)
```

Cartesian projection
and
Graph Plotting



Cartesian View

- **A Cartesian view is defined by 2 axis i.e. 2 calls to *paxis*.**




```
mmap(subpage_map_projection = 'cartesian',
        subpage_x_axis_type='regular',
        subpage_x_min = -20,  subpage_x_max=20,
        subbpage_y_axis_type='regular',
        subpage_y_min = -20,  subpage_y_max=20)
```

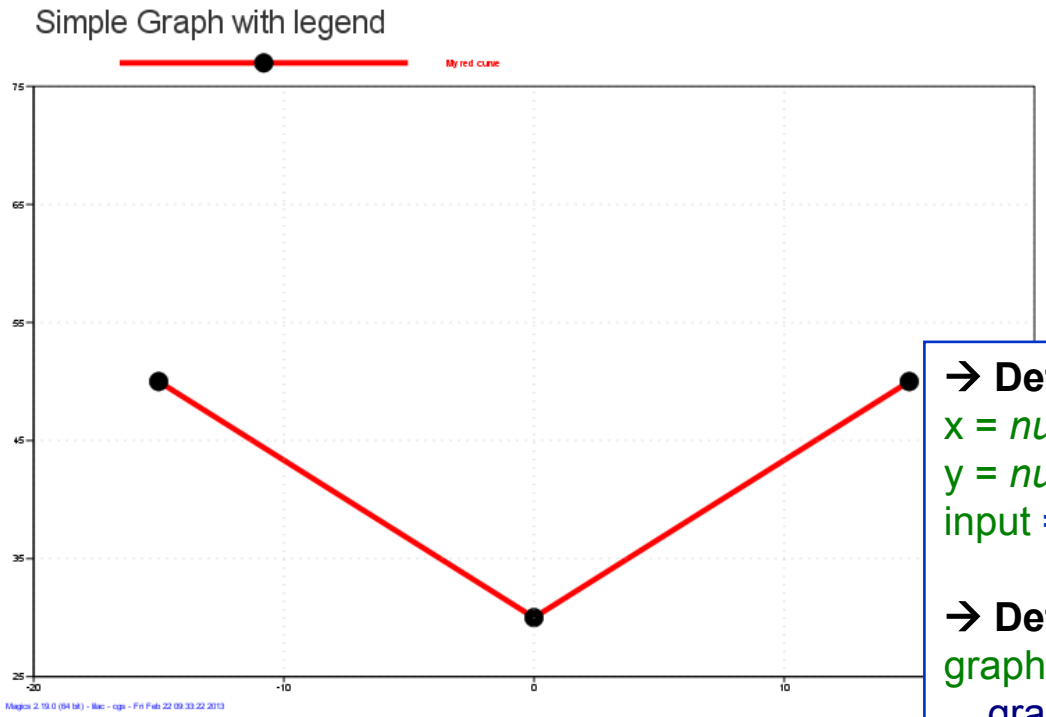
- **The setting should follow these basic rules**

→First step : define the horizontal axis

```
horizontal = maxis(axis_orientation='horizontal')
```

→Second step : define the vertical axis

```
vertical = maxis(axis_orientation='vertical')
```

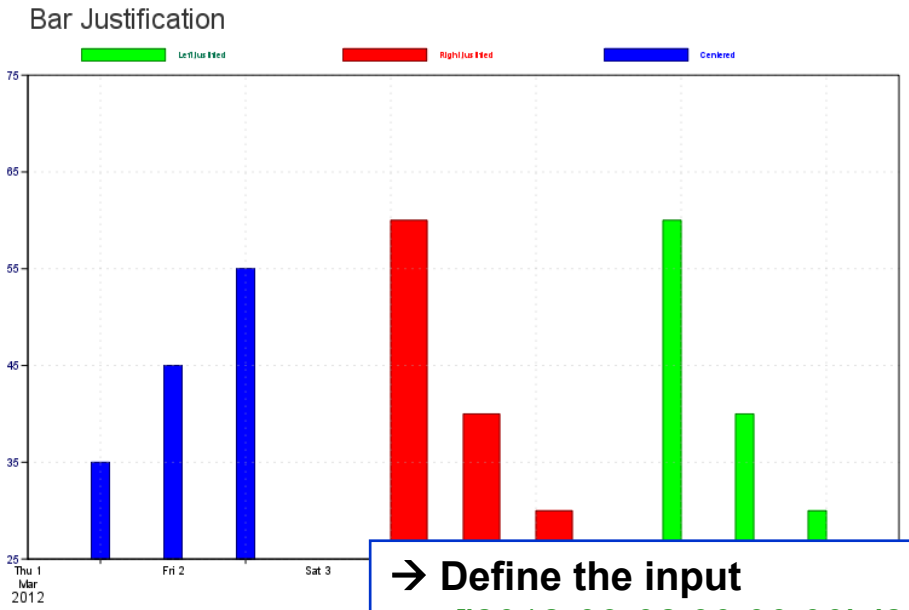


→ **Define the input**

```
x = numpy.array([-15., 0., 15.])
y = numpy.array([50., 30., 50.])
input = minput(input_x_values = x,
               input_y_values = y,
```

→ **Define the graph**

```
graph = mgraph(
    graph_line_colour='red',
    graph_line_thickness=8,
    graph_symbol='on',
    legend='on',
    legend_user_text="My red curve",
    graph_symbol_marker_index=15,
    graph_symbol_colour='black',
    graph_symbol_height=1.)
```



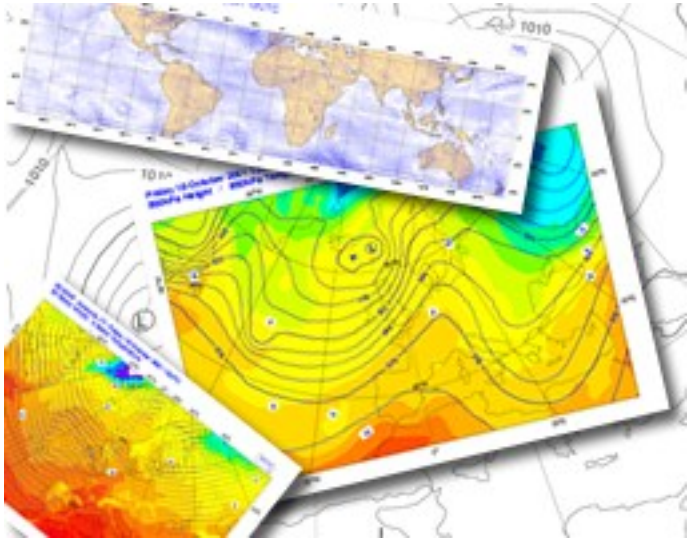
→ **Define the input**

```
x = ['2012-03-02 00:00:00', '2012-03-02 12:00:00', '2012-03-03 00:00:00']
y = numpy.array([35., 45., 55.])
input = minput(  input_x_values= x,
                 input_y2_values= y,
                  input_y_values= [0.] * len(y),
```

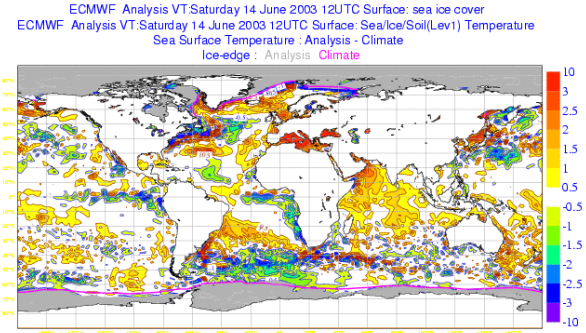
→ **Define the graph**

```
centre = mgraph(
    graph_type='bar',
    graph_bar_justification='centre',
    graph_bar_colour='blue',
    graph_bar_width=3 * 3600.,
    legend='on',
    legend_user_text="Centered")
```

Wrap-up



Where to find information?



- On the Magics web site, you will find:
  - Gallery of examples
  - Tutorial
  - Reference manual
  - Magics change history
  - Information about input and output formats
  - Links to Newsletter article about Magics

https://software.ecmwf.int/magics

Contacting the Magics developers

- We encourage users to contact us to give feedback.
- If you send emails to magics@ecmwf.int to ask for help please provide us with this information:
  - A clear subject line (e.g. “Logo looks upside down”)
  - A short description of your system (e.g. ‘uname -a’ and compilers used)
  - A time frame which a problem needs to be fixed
  - Please compress large files (gzip / bzip)
- Currently we aim for two major releases a year and you might have to wait for the next release to get an update

Getting Magics++

- **Download free (under Apache license) from**

  https://software.ecmwf.int/magics/Releases

- **Installation Guide at**

  https://software.ecmwf.int/magics/Installation+Guide

  ./configure ; make ; make check ; make install

How to compile and execute your Magics++ program at home?

- Magics++ provides a script called magics-config to help you setting up your environment and compile Magics programs.
- The script is installed in $prefix/bin .
- Setup Magics once per shell: magics-config --print-setup
- Compile and link using
  - gfortran test.f -o ftest `magics-config --f90libs`
  - gcc test.c -o ctest `magics-config --cxxincludes --clibs`

  or even

  - magics-config --compile=myprogram.f (only for pure Magics programs!)

How to compile and execute your Magics++ program at ECMWF ?

- Setup Magics once per shell: use magics++
- Create your Fortran program
  - Use an editor (*vi*, xemacs, *nedit, kwrite*) to write your Magics calls
- Compile and link
  - ➔ **Linux:** *pgf90 –o myMagics myMagics.f $MAGPLUSLIB_SHARED*
- Run your program
- View your result - Use a PostScript viewer : *gv*, display
- Modify your program if necessary …

Any questions / suggestions?

Any questions and suggestions are welcome!

Contact us via email:

magics@ecmwf.int

Visit our webpage:

https://software.ecmwf.int/magics