CliMetLab

Release 0.18.9

ECMWF

Oct 30, 2023

CONTENTS

Warning: This documentation is work in progress. It is not yet ready.

CliMetLab is a Python package aiming at simplifying access to climate and meteorological datasets, allowing users to focus on science instead of technical issues such as data access and data formats. It is mostly intended to be used in Jupyter notebooks, and be interoperable with all popular data analytic packages, such as NumPy, Pandas, Xarray, SciPy, Matplotlib, etc. as well as machine learning frameworks, such as TensorFlow, Keras or PyTorch. See *Overview* for more information.

CHAPTER

ONE

DOCUMENTATION

1.1 Overview

CliMetLab is a Python package which is intended to be used in Jupyter notebooks. Its main goal is to greatly reduce boilerplate code by providing high-level unified access to meteorological and climate datasets, allowing scientists to focus on their research instead of solving technical issues. Datasets are automatically downloaded, cached and transform into standard Python data structures such as NumPy, Pandas or Xarray, that can then be fed into scientific packages like SciPy and TensorFlow. *CliMetLab* also aims at simplifying plotting of 2D maps, by automatically selecting the most appropriate styles and projections for any given data.

The goal of *CliMetLab* is to simplify access to climate and meteorological datasets, by hiding the access methods and data formats. The snippet of code below would download the dataset *dataset-name*, cache it locally and decodes its content as a NumPy array:

```
import climetlab as clm

data = clm.load_dataset("dataset-name")
a = data.to_numpy()
```

To achieve this, *CliMetLab* introduces two concepts: *Data source* and *Dataset*. Data sources represent various access methods, such as reading files, downloading from a web site or using APIs.

CliMetLab provides the interface between the left side and the right side of the figure below:

CliMetLab also provides very high-level map plotting facilities. By default *CliMetLab* will automatically select the most appropriate way to plot a dataset, choosing the best projection, colours and other graphical attributes. Users can then control how maps are drawn by overriding the automatic choices with their own.

```
import climetlab as clm

data = clm.load_dataset("some-dataset")
cml.plot_map(data)
```

1.2 Installing

1.2.1 Pip install

To install CliMetLab, just run the following command:

```
pip install climetlab
```

The CliMetLab `pip` package has been tested successfully with the latest versions of its dependencies (build logs).

1.2.2 Conda install

No conda package has been created yet. `pip install climetlab` can be used in a conda environment.

Note: Mixing `pip` and `conda` could create some dependencies issues, we recommend installing as many dependencies as possible with conda, then install CliMetLab with `pip`, as recommended by the anaconda team.

1.2.3 Troubleshooting

Python 3.8 or above is required

CliMetLab requires Python 3.8 or above (mainly due the dependency *numpy*). Depending on your installation, you may need to substitute `pip` to `pip3` in the examples below. See the build logs to know on which version of Python CliMetLab is automatically tested.

No matching distribution found for ...

If the installation fails with the following error:

```
Collecting ecmwflibs>=x.x.x (from climetlab)
  Could not find a version that satisfies the requirement ecmwflibs>=0.0.90 (from␣
↪climetlab) (from versions: )
No matching distribution found for ecmwflibs>=x.x.x (from climetlab)
```

you will need to make sure that you are using the latest version of `pip` (>=21.0.0):

```
% pip install --upgrade pip
% pip install climetlab
```

WARNING: Retrying (Retry(total=4, connect=None, ...

If you are installing CliMetLab (or any other package) wihout internet access. You can get a similar error. This happens for instance on Kaggle when you are not logged it or when your account has not been verified.

```
WARNING: Retrying (Retry(total=4, connect=None, read=None, redirect=None, status=None))
after connection broken by 'NewConnectionError(': Failed to establish a new connection:
[Errno -3] Temporary failure in name resolution')':....
```

Module enum has no attribute 'IntFlag'

If the installation fails with the following error:

```
AttributeError: module 'enum' has no attribute 'IntFlag'
```

This means that there is an old version of of the `enum` package on your system that needs to be removed:

```
% pip uninstall -y enum34
% pip install climetlab
```

1.3 First Steps Tutorial

This is an easy to follow tutorial that gets you up to speed with *CliMetLab* to access scientific climate data. It assumes you have basic Python programming knowledge.

So, let's start with what *CliMetLab* does best:

1.3.1 Getting data

CliMetLab provides two main ways to access climate and meteorological data

1. Data Sources (*details*)
2. Datasets (*details*)

Data Sources

In *CliMetLab*, a *Data Source* refers to a local or remote storage server or data archive from where we can download or access files related to climate data.

To get started, let us first import *CliMetLab* at the top of our Python notebook:

```
import climetlab as cml
```

Now download test.grib (example GRIB file) to your project directory, or if you have `wget` command available, run following in your notebook:

```
!wget https://raw.githubusercontent.com/ecmwf/climetlab/main/docs/examples/test.grib
```

GRIB is a file format for storage and sharing of gridded meteorological data. You can think of gridded data as weather or some other data that is associated with specific locations using coordinates. For example, wind speed data for every longitude and latitude on a two dimensional grid.

The GRIB format (version 1 and 2) is endorsed by WMO. GRIB (GRIdded Binary) is a binary file format so you cannot look at it using a text editor. But you sure can use *CliMetLab* to explore it:

```
grib_data = cml.load_source("file", "test.grib")
```

Here we used `load_source` method from *CliMetLab* to load our GRIB file into `grib_data` variable. The first argument `"file"` specifies the type of our data source. Which currently is indeed a local file that we downloaded. The second argument is the path to that file.

Let's plot this data using the `plot_map` convenience method:

```
cml.plot_map(grib_data, title=True, path="test-grib-plot.svg")
```

The `title` and `path` arguments supplied to `plot_map` are optional. When `title` is set to `True`, the plot will include the title of the data from our data file. If you specify the `path` ending with a supported format (`.svg`, `.png`, `.pdf`), the plot will be saved at the specified location.

You can also use `load_source` to directly download and load files from remote server, in one step. For example, let's download and plot a NetCDF file:

```
netcdf_url = "https://raw.githubusercontent.com/ecmwf/climetlab/main/docs/examples/test.
↪nc"
netcdf_data = cml.load_source("url", netcdf_url)
cml.plot_map(netcdf_data)
```

If you don't know already, NetCDF is another commonly used data format for transporting scientific data.

So, we have downloaded and plotted scientific weather data, can we convert this data to work with our favorite Data Science library? `Pandas`, `Xarray` or `Numpy`? Don't worry, *CliMetLab* has got you covered.

If your data is gridded data, like in our examples above, you can simply call `to_xarray()` method to convert it to an `Xarray` object:

```
grib_xr = grib_data.to_xarray()
# as well as:
netcdf_xr = netcdf_data.to_xarray()
```

CliMetLab will infer the type of data by probing the downloaded file. If the file contains gridded data, such as meteorological fields, they will be accessible as an Xarray dataset, using the `to_xarray()` method. If the file contains point data, such as observations, they can be converted to a Pandas data frame via the `to_pandas()` method. Other data may only be available as NumPy arrays using the `to_numpy()` method.

Observations are usually data points corresponding to time for a certain location unlike gridded data that contains data points that is usually for a range of locations. For example, monthly average temperature of Lahore for the past ten years.

The following example downloads a `.csv` file from NOAA's *International Best Track Archive for Climate Stewardship* (IBTrACS) using the `url` data source. The file is downloaded into the local cache. We then convert it to a Pandas frame. The rows corresponding to the severe tropical cyclone Uma are extracted and plotted.

```
import climetlab as cml

data = cml.load_source(
    "url",
    (
        "https://www.ncei.noaa.gov/data/international-best-track-archive-for-climate-
↪stewardship-ibtracs"
```

(continues on next page)

(continued from previous page)

```
        "/v04r00/access/csv/ibtracs.SP.list.v04r00.csv"
    ),
)

pd = data.to_pandas()
uma = pd[pd.NAME == "UMA:VELI"]
cml.plot_map(uma, style="cyclone-track")
```

The Data Sources implement various methods to access and decode data. When data are downloaded from a remote site, they are *cached* on the local computer.

The first argument to `load_source` can take the following values:

Argument value	Description
`"file"`	A path to a local file name. *Read more*
`"url"`	A URL to a remote file. *Read more*
`"cds"`	A request to retrieve data from the Copernicus Climate Data Store (CDS). Requires an account. *Read more*
`"mars"`	A request to retrieve data from ECMWF's meteorological archive (MARS), using the ECMWF web API. Requires an account. *Read more*

To read more about Data Sources, head over to *Data Sources guide*.

Now let's dive into the second way that you can access climate scientific data using *CliMetLab*:

Datasets

Datasets are a higher-level concept compared to data sources.

Todo: metadata, hidden access to sources, control plotting, control conversion to pandas

---

The following Python code:

```
import climetlab as cml

data = cml.load_dataset("hurricane-database", bassin="atlantic")
print(data.home_page)
```

will print:

```
https://www.aoml.noaa.gov/hrd/hurdat/Data_Storm.html
```

then,

```
irma = data.to_pandas(name="irma", year=2017)
cml.plot_map(irma)
```

will plot:

Compare that with the *data source example*.

1.3.2 Simple plotting

CliMetLab will try to select the best way to plot data.

```
cml.plot_map(data)
```

Below are the parameters you can pass to the plot function:

Name	Value	Default	Description
title	str or bool	`False`	The title of the plot. Use `True` for automatic.
projection	str	`None`	The name of a map projection. Use `None` for automatic. See below for possible values.
style	str	`None`	The name of a plotting to apply. Use `None` for default. See below for possible values.
foreground	str	`None`	TODO. See below for possible values.
background	str	`None`	TODO. See below for possible values.
path	str	`None`	Save the plot in a file instead of displaying it. The file type is inferred from the path extension (`.png`, `.pdf`, `.svg`, ...)

You can find out what are the possible values for *projection*, *style*, *foreground* and *background* parameters using the code below:

```
import climetlab.plotting

# List of possible projections
for p in climetlab.plotting.projections():
    print(p)

# List of possible styles
for p in climetlab.plotting.styles():
    print(p)

# List of possible backgrounds/foregrounds
for p in climetlab.plotting.layers():
    print(p)
```

To get more information about a given projection, in a Jupyter Notebook:

```
from climetlab.plotting import projection

projection("global")
```

will output:

Name:	global
Collection:	projections
Path:	/opt/venv/lib/python3.7/site-packages/climetlab/data/projections/global.yaml
Definition:	

```yaml
magics:
  mmap:
    subpage_lower_left_latitude: -90.0
    subpage_lower_left_longitude: -180.0
    subpage_map_projection: cylindrical
    subpage_upper_right_latitude: 90.0
    subpage_upper_right_longitude: 180.0
```

1.3.3 Advanced plotting

Todo: Improve documentation.

There are two options to plot several datasets on the same map. If you do not need to specify data specific parameters (e.g. *style*), you can call `cml.plot_map()` with a list of data objects.

```
cml.plot_map((data1, data2), foreground=False)
```

or, if you want to specify a per-data custom *style*, you can use `cml.new_map()`:

```
p = cml.new_plot(projection="global")
p.plot_map(data1, style="style1")
p.plot_map(data2, style="style2")
p.show()
```

1.4 Examples

Here is a list of example notebooks to illustrate how to access data, create plots, and do machine learning using CliMetLab.

You can run this notebook in , in , in

[1]:
```
!pip3 install --quiet climetlab
```

[2]:
```
!test -f test.grib || wget https://github.com/ecmwf/climetlab/raw/main/docs/examples/
↪test.grib
```

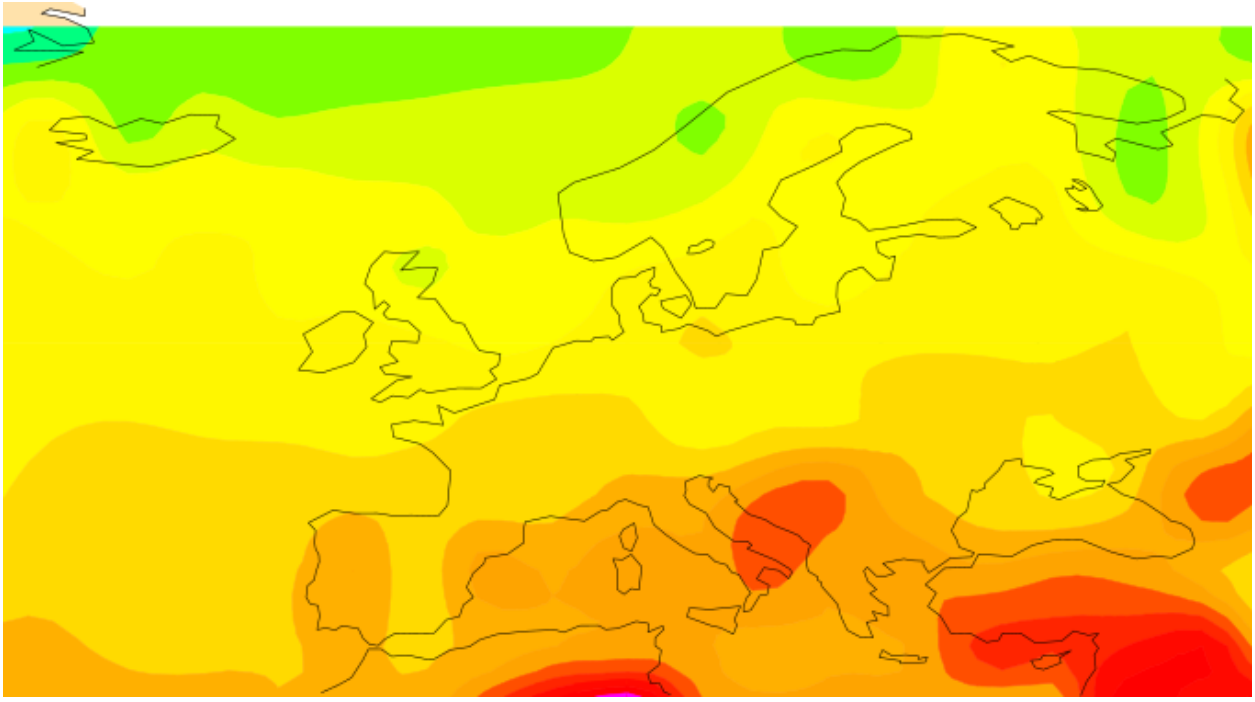
[3]:
```
!test -f test.nc || wget https://github.com/ecmwf/climetlab/raw/main/docs/examples/test.
↪nc
```

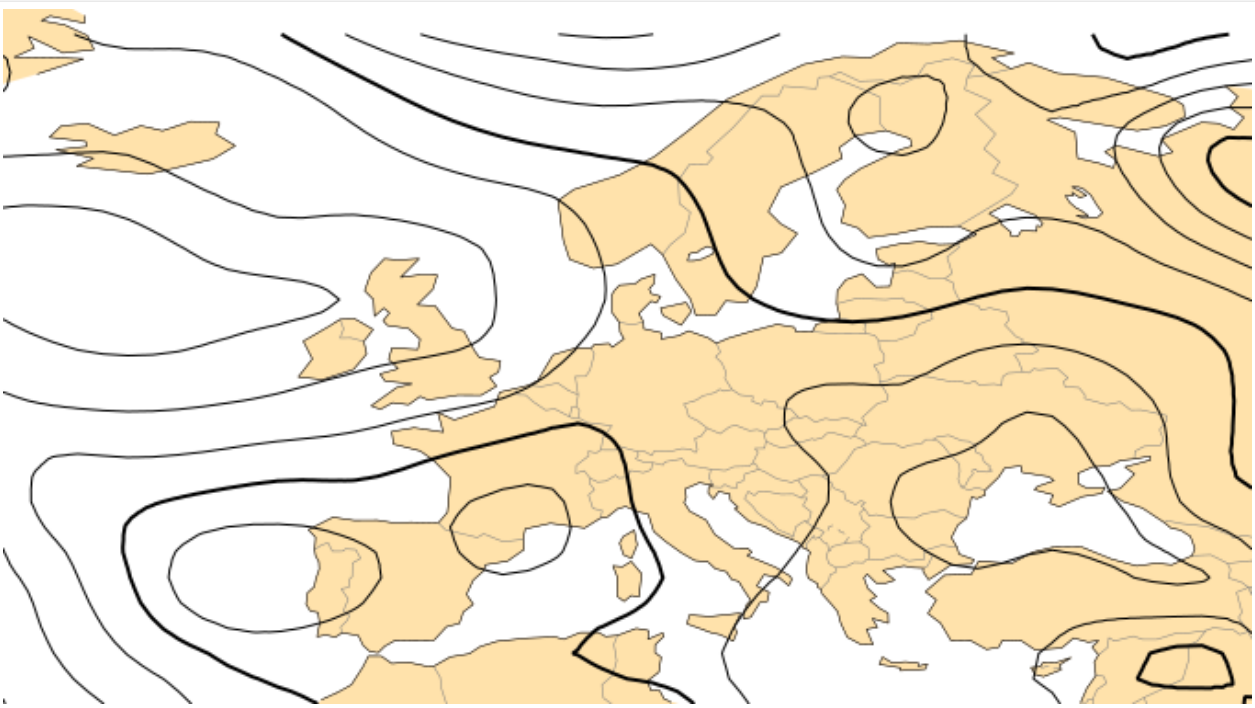
1.4.1 Reading data from a file

[4]:
```
import climetlab as cml
```

Plot GRIB data

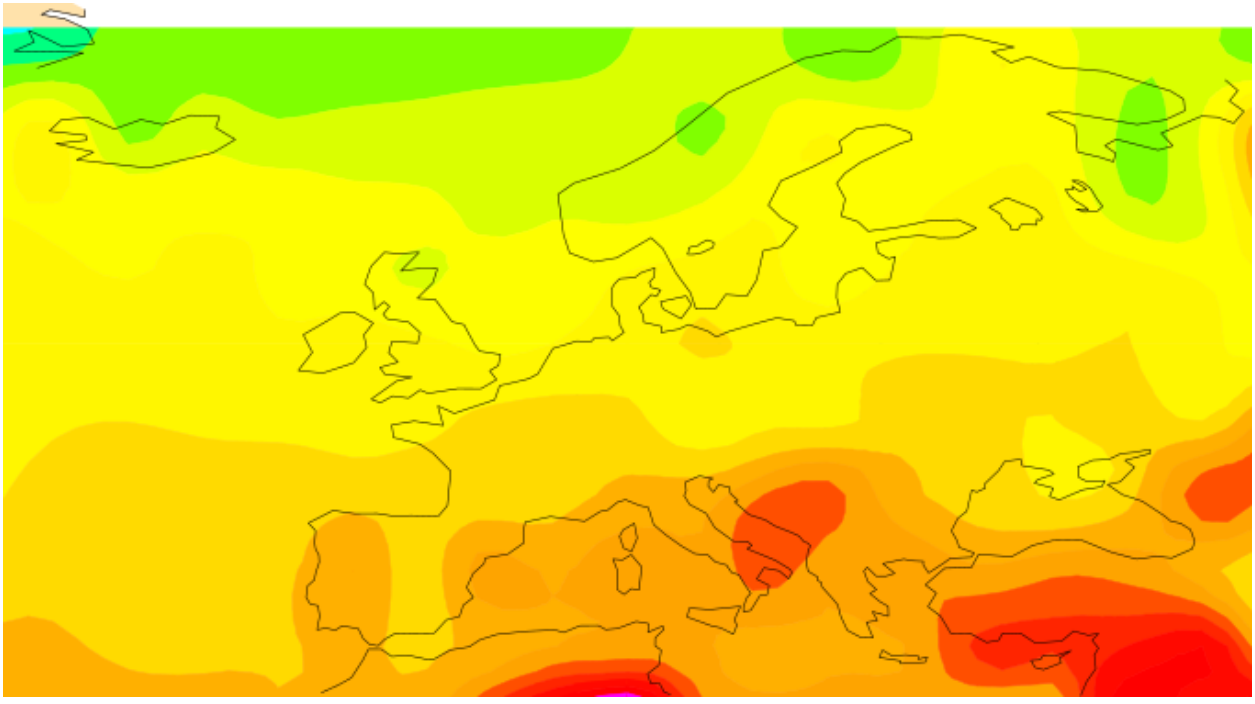
[5]:
```
source = cml.load_source("file", "test.grib")
for s in source:
    cml.plot_map(s)
```

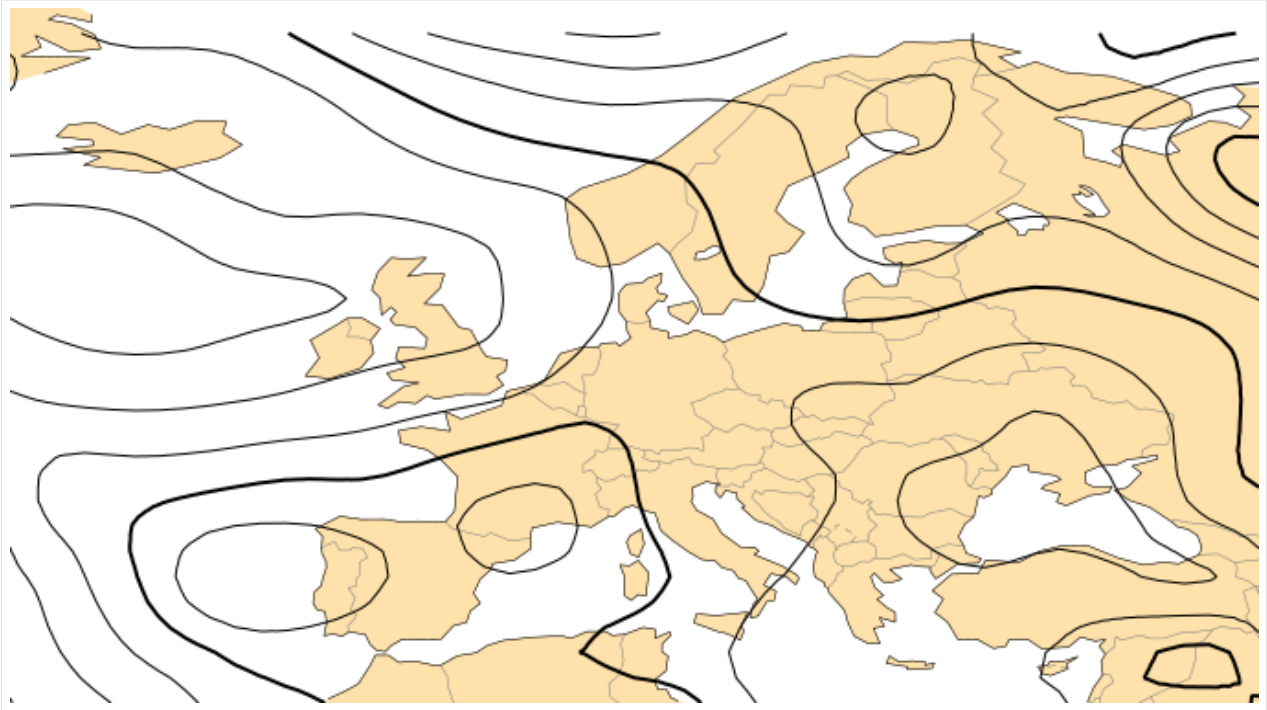




Plot NetCDF data

[6]:
```
source = cml.load_source("file", "test.nc")
for s in source:
    cml.plot_map(s)
```





You can run this notebook in , in , in

[1]:
```
!pip install --quiet climetlab
```

1.4.2 Downloading from an URL

[2]:
```
import climetlab as cml
```

[3]:
```
URL = "https://www.ncei.noaa.gov/data/international-best-track-archive-for-climate-
↪stewardship-ibtracs/v04r00/access/csv/ibtracs.SP.list.v04r00.csv"
```

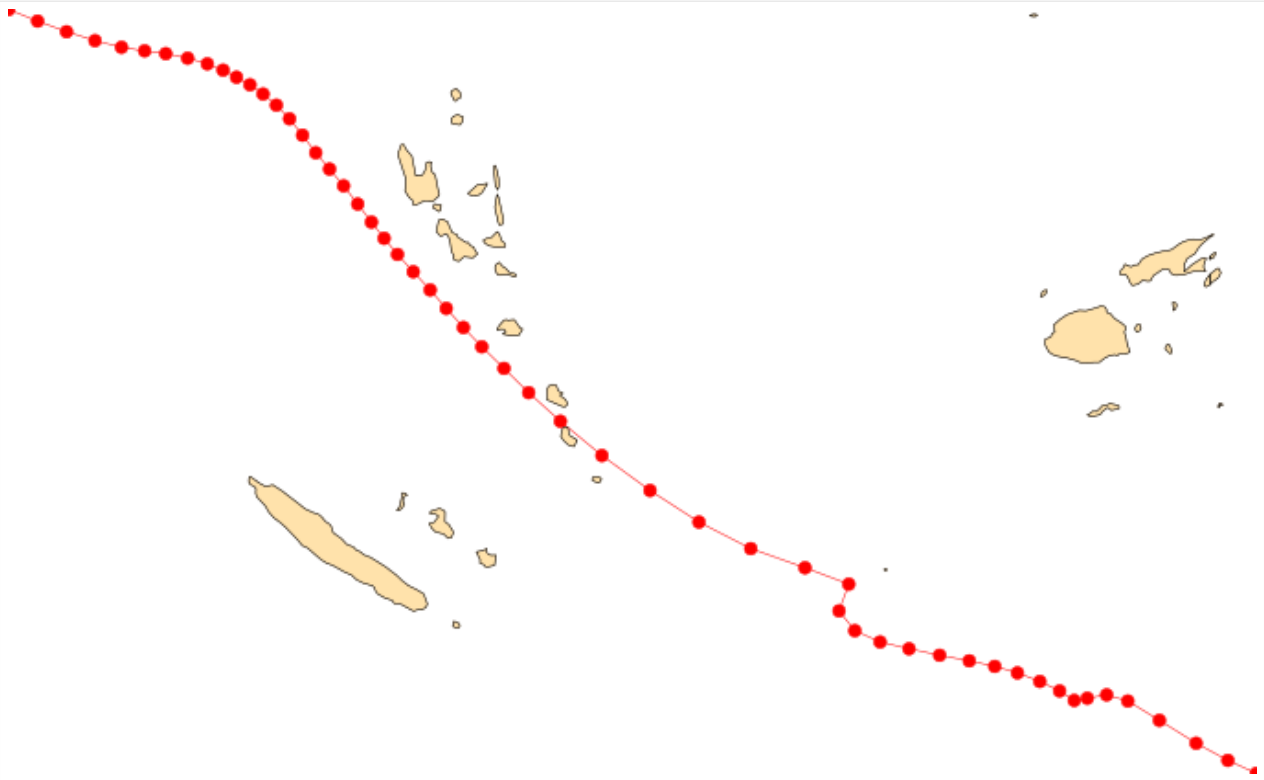
[4]:
```
data = cml.load_source("url", URL)
```

[5]:
```
pd = data.to_pandas()
```

```
/usr/local/lib/python3.9/site-packages/IPython/core/interactiveshell.py:3418:␣
↪DtypeWarning: Columns (1,2,8,9,14,25,161,162) have mixed types.Specify dtype option on␣
↪import or set low_memory=False.
  exec(code_obj, self.user_global_ns, self.user_ns)
```

[6]:
```
uma = pd[pd.NAME == "UMA:VELI"]
```

[7]:
```
cml.plot_map(uma, style="cyclone-track")
```



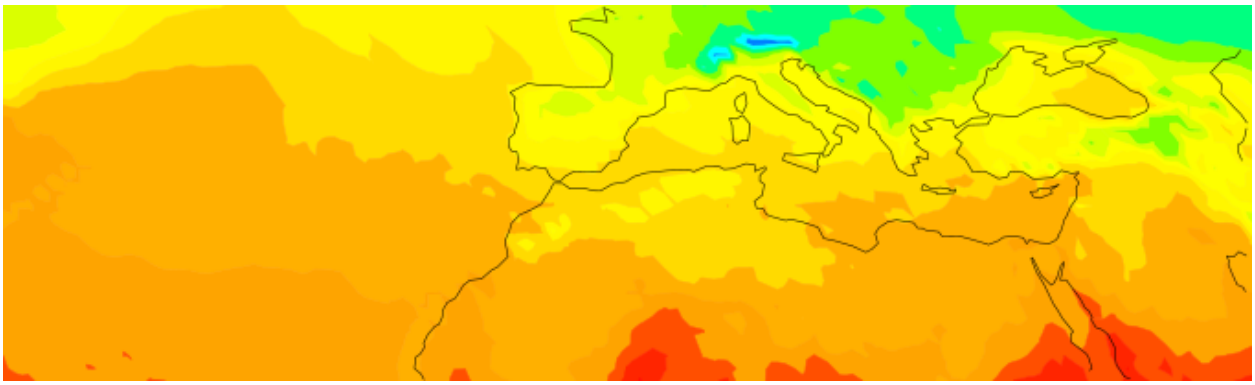
You can run this notebook in , in , in

[1]:
```
!pip install --quiet climetlab
```

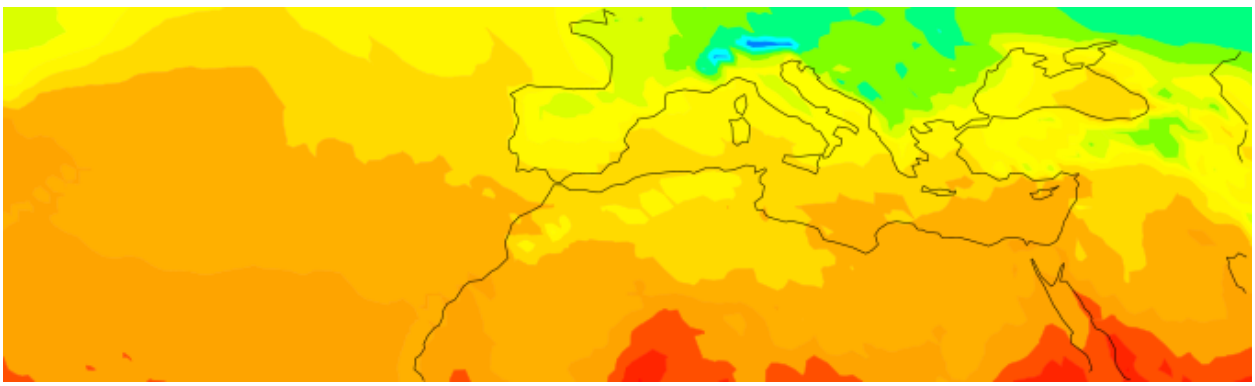
1.4.3 Retrieve ERA5 data from the CDS

[2]:
```
import climetlab as cml
```

[6]:
```
source = cml.load_source(
    "cds",
    "reanalysis-era5-single-levels",
    variable=["2t", "msl"],
    product_type="reanalysis",
    area=[50, -50, 20, 50],
    date="2012-12-12",
    time="12:00",
)
for s in source:
    cml.plot_map(s)
```



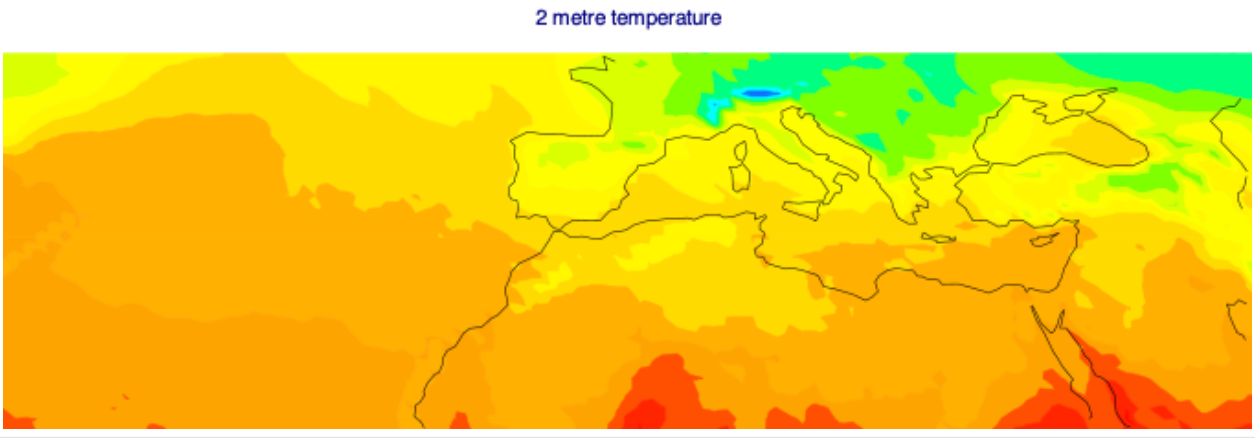
[8]:
```
source = cml.load_source(
    "cds",
    "reanalysis-era5-single-levels",
    variable=["2t", "msl"],
    product_type="reanalysis",
    area=[50, -50, 20, 50],
    date="2012-12-12",
    time="12:00",
    format="netcdf",
)
for s in source:
    cml.plot_map(s)
```



[10]:
```
source.to_xarray()
```

[10]:
```
<xarray.Dataset>
Dimensions:    (latitude: 121, longitude: 401, time: 1)
Coordinates:
  * longitude  (longitude) float32 -50.0 -49.75 -49.5 -49.25 ... 49.5 49.75 50.0
  * latitude   (latitude) float32 50.0 49.75 49.5 49.25 ... 20.5 20.25 20.0
  * time       (time) datetime64[ns] 2012-12-12T12:00:00
Data variables:
    t2m        (time, latitude, longitude) float32 ...
    msl        (time, latitude, longitude) float32 ...
Attributes:
    Conventions:  CF-1.6
    history:      2021-03-22 10:05:13 GMT by grib_to_netcdf-2.16.0: /opt/ecmw...
```

[12]:
```
cml.plot_map(source.to_xarray(), title=True)
```



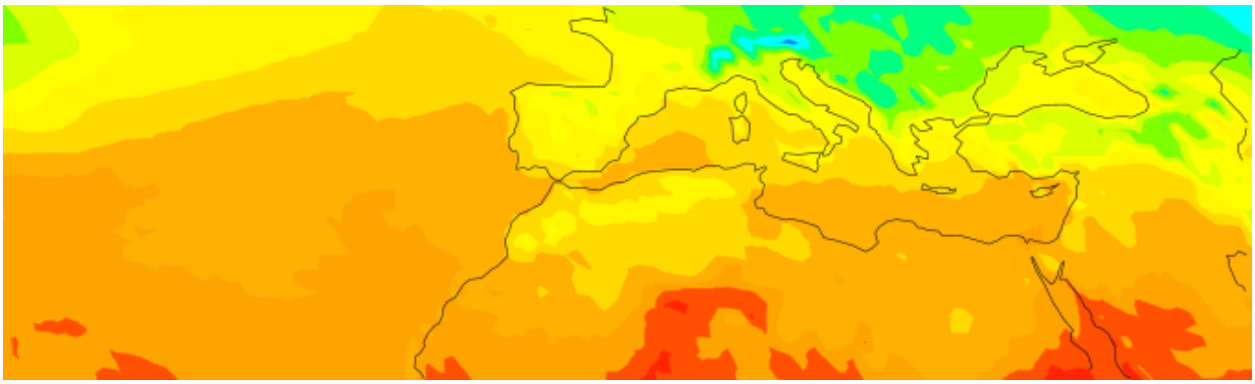
You can run this notebook in , in , in

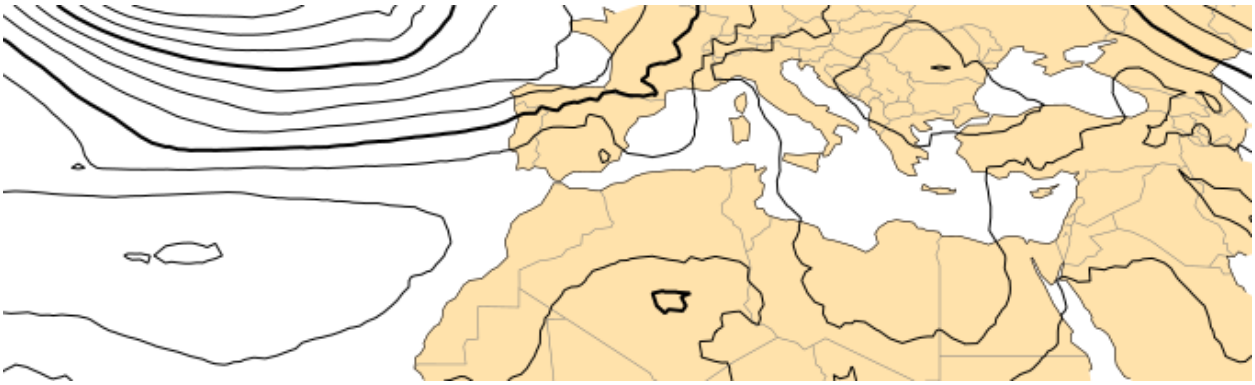
[1]:
```
!pip install --quiet climetlab
```

1.4.4 Retrieve data ECMWF MARS archive

[2]:
```
import climetlab as cml
```

[3]:
```
source = cml.load_source(
    "mars",
    param=["2t", "msl"],
    levtype="sfc",
    area=[50, -50, 20, 50],
    grid=[1, 1],
    date="2012-12-13",
)
for s in source:
    cml.plot_map(s)
```





[4]:
```
source.to_xarray()
```

[4]:
```
<xarray.Dataset>
Dimensions:     (latitude: 31, longitude: 101)
Coordinates:
    number      int64 ...
    time        datetime64[ns] ...
    step        timedelta64[ns] ...
    surface     int64 ...
  * latitude    (latitude) float64 50.0 49.0 48.0 47.0 ... 23.0 22.0 21.0 20.0
  * longitude   (longitude) float64 -50.0 -49.0 -48.0 -47.0 ... 48.0 49.0 50.0
    valid_time  datetime64[ns] ...
Data variables:
```

(continues on next page)

(continued from previous page)

```
    t2m         (latitude, longitude) float32 ...
    msl         (latitude, longitude) float32 ...
Attributes:
    GRIB_edition:            1
    GRIB_centre:             ecmf
    GRIB_centreDescription:  European Centre for Medium-Range Weather Forecasts
    GRIB_subCentre:          0
    Conventions:             CF-1.7
    institution:             European Centre for Medium-Range Weather Forecasts
    history:                 2021-03-24T13:23:13 GRIB to CDM+CF via cfgrib-0...
```

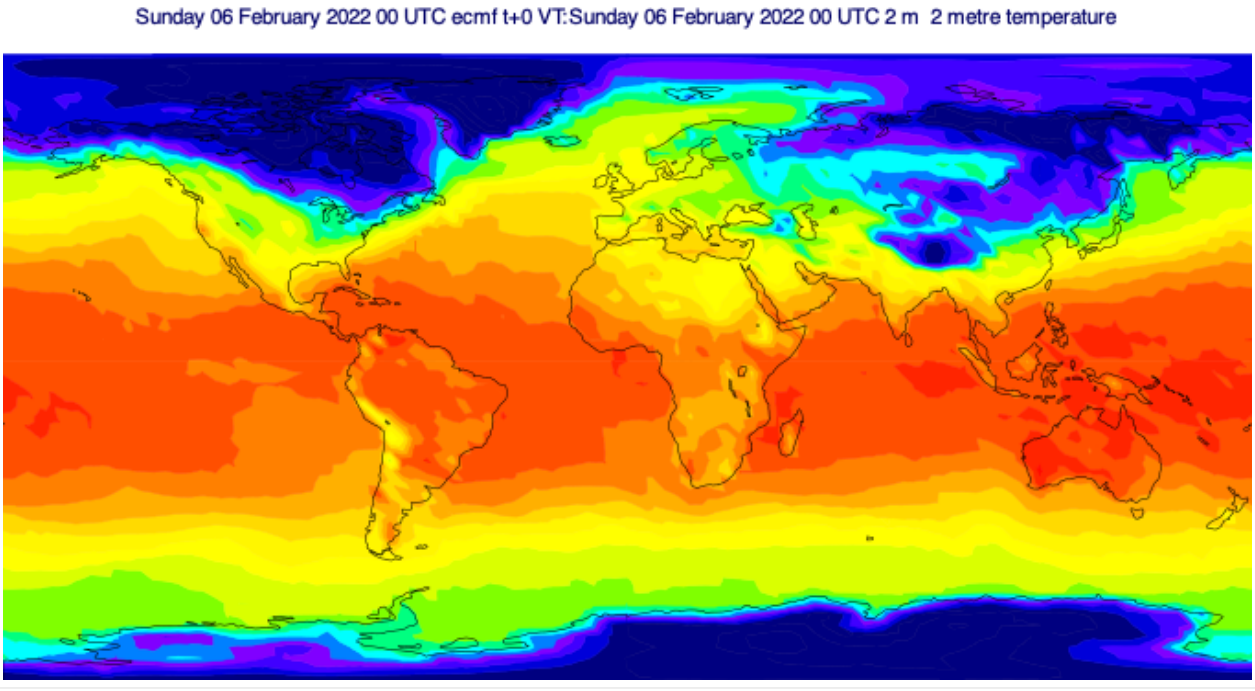
You can run this notebook in , in , in

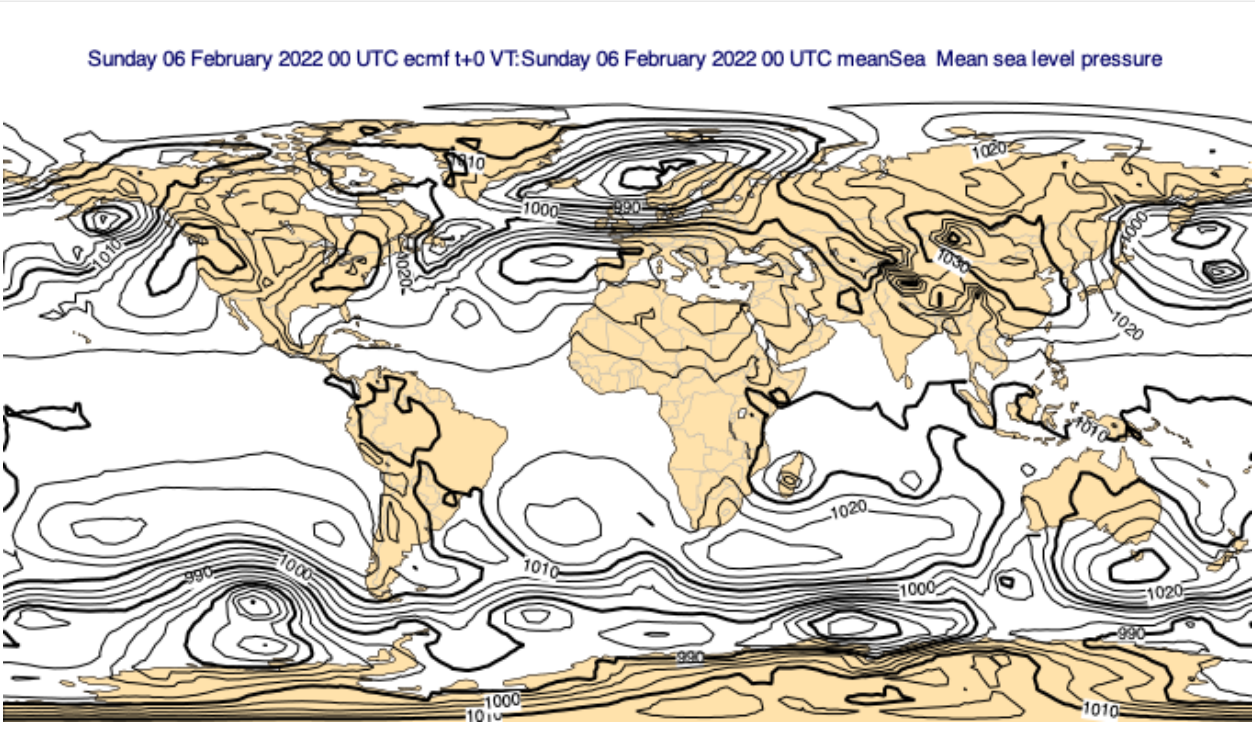
[20]:
```
!pip install --quiet climetlab
```

1.4.5 Access ECMWF Open Data feed

[21]:
```
import climetlab as cml
```

[22]:
```
source = cml.load_source(
    "ecmwf-open-data",
    param=["2t", "msl"],
)
for s in source:
    cml.plot_map(s, title=True)
```





[23]:
```
source.to_xarray()
```

[23]:
```
<xarray.Dataset>
Dimensions:             (latitude: 451, longitude: 900)
Coordinates:
    time                datetime64[ns] ...
    step                timedelta64[ns] ...
    heightAboveGround   float64 ...
  * latitude            (latitude) float64 90.0 89.6 89.2 ... -89.2 -89.6 -90.0
  * longitude           (longitude) float64 -180.0 -179.6 -179.2 ... 179.2 179.6
    valid_time          datetime64[ns] ...
    meanSea             float64 ...
Data variables:
    t2m                 (latitude, longitude) float32 ...
    msl                 (latitude, longitude) float32 ...
Attributes:
    GRIB_edition:            2
    GRIB_centre:             ecmf
    GRIB_centreDescription:  European Centre for Medium-Range Weather Forecasts
    GRIB_subCentre:          0
    Conventions:             CF-1.7
    institution:             European Centre for Medium-Range Weather Forecasts
    history:                 2022-02-06T12:42 GRIB to CDM+CF via cfgrib-0.9.1...
```

You can run this notebook in , in , in

[1]:
```
!pip install --quiet climetlab matplotlib
```

1.4.6 ERA5-based datasets

[2]:
```
import numpy as np
import matplotlib.pyplot as plt
```

[3]:
```
import climetlab as cml
```

[4]:
```
cml.plotting_options(width=400)
```

Surface temperature in France

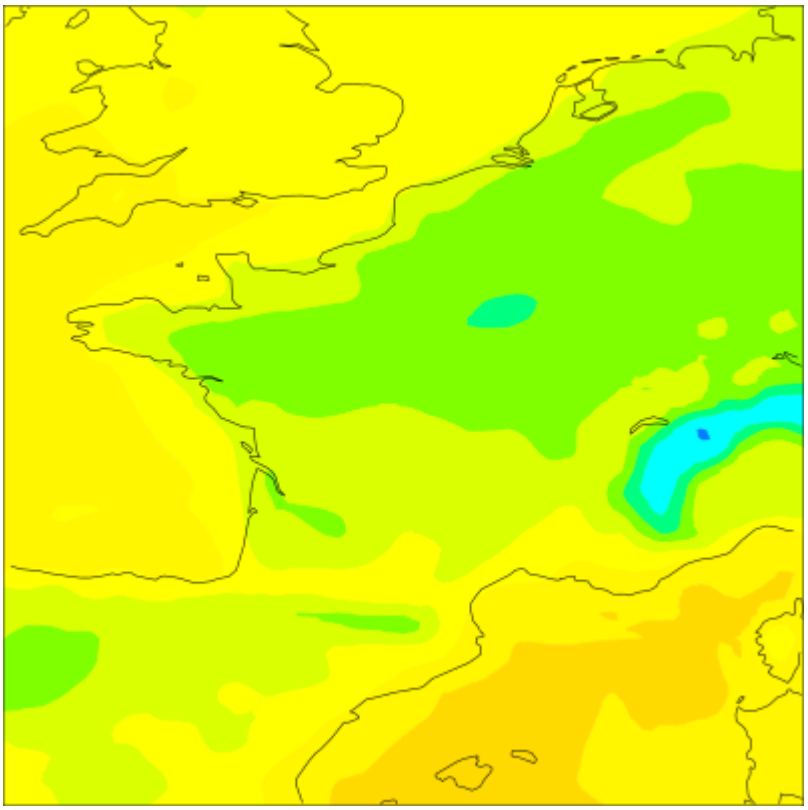
[5]:
```
ds = cml.load_dataset("era5-temperature", period=(1979, 1982), domain="France", time=12)
```

[6]:
```
%%time
len(ds)
```

```
CPU times: user 558 ms, sys: 94.9 ms, total: 653 ms
Wall time: 686 ms
```

[6]:
```
1461
```

[7]:
```
cml.plot_map(ds[-1])
```



[8]:
```
%%time
ds.to_numpy().shape
```

```
CPU times: user 603 ms, sys: 124 ms, total: 727 ms
Wall time: 739 ms
```

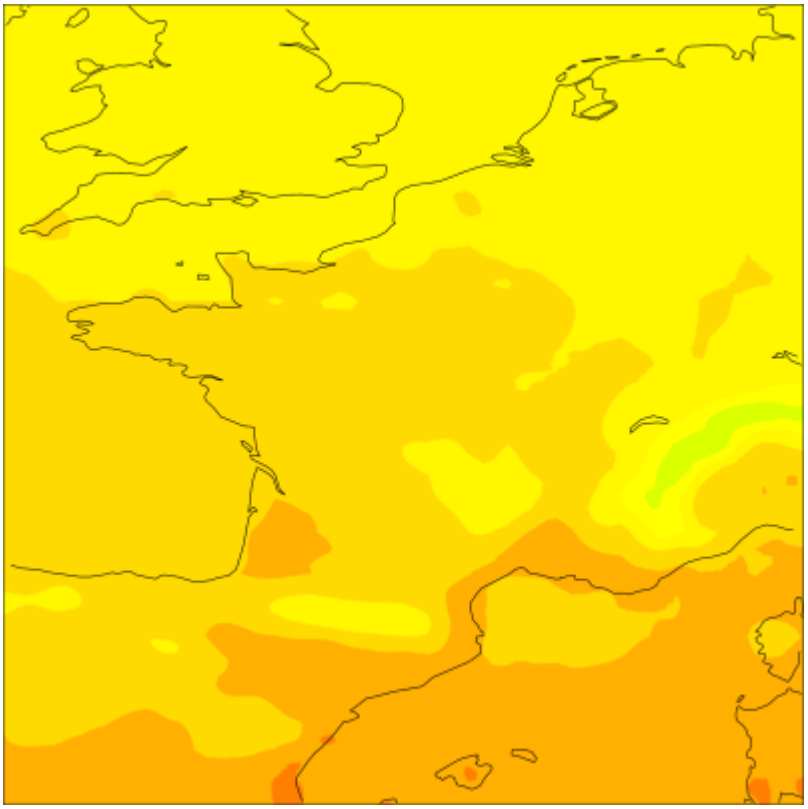
[8]:
```
(1461, 63, 63)
```

[9]:
```
average = np.mean(ds.to_numpy(), axis=0)
```

[10]:
```
average
```

[10]:
```
array([[[283.75848794, 283.5668571 , 283.28540925, ..., 283.07590083,
         283.10228607, 283.10129547],
        [283.4987798 , 283.60046529, 283.65712864, ..., 283.17566394,
         283.24349259, 283.31315138],
        [283.27489098, 282.96305539, 283.02593639, ..., 283.29626172,
         283.42080452, 283.51664   ],
        ...,
        [291.8236389 , 291.01727633, 290.47778203, ..., 292.8580171 ,
         292.87150182, 293.54986986],
        [292.33919705, 292.04851029, 291.94756675, ..., 292.50002974,
         292.48287807, 294.12919668],
        [292.80611291, 292.51362676, 292.36608895, ..., 292.37801758,
         292.20149706, 292.06164609]])
```

[11]:
```
cml.plot_map(average, metadata=ds[0])
```



[12]:
```
x = ds.to_xarray()
```

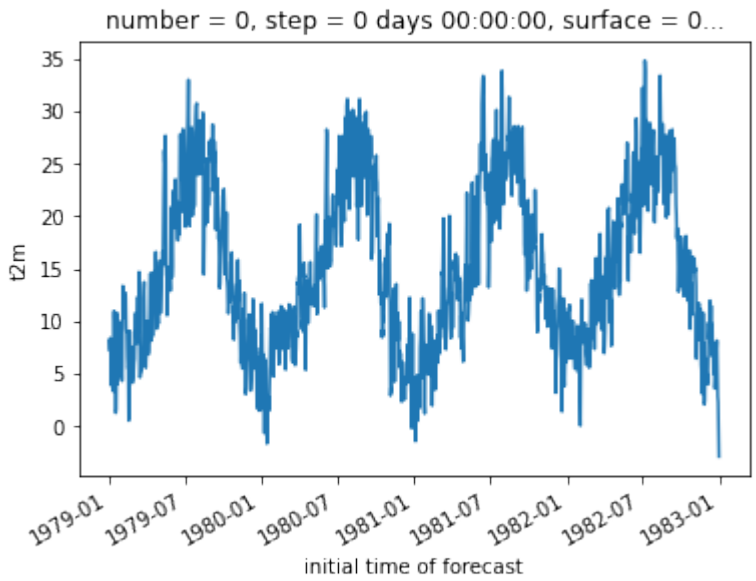
[13]:
```
x
```

[13]:
```
<xarray.Dataset>
Dimensions:     (latitude: 63, longitude: 63, time: 1461)
Coordinates:
  * time        (time) datetime64[ns] 1979-01-01T12:00:00 ... 1982-12-31T12:0...
    number      int64 0
    step        timedelta64[ns] 00:00:00
    surface     int64 0
  * latitude    (latitude) float64 54.5 54.25 54.0 53.75 ... 39.5 39.25 39.0
  * longitude   (longitude) float64 -6.0 -5.75 -5.5 -5.25 ... 8.75 9.0 9.25 9.5
    valid_time  (time) datetime64[ns] 1979-01-01T12:00:00 ... 1982-12-31T12:0...
Data variables:
    t2m         (time, latitude, longitude) float32 269.13837 ... 285.14648
```

[14]:
```
point = x.t2m.isel(latitude=50, longitude=5) - 273.15
```

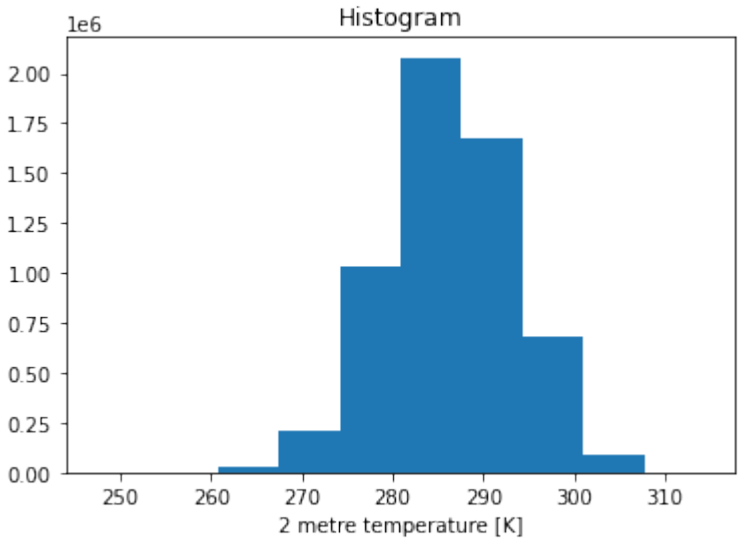
[15]:
```
point.plot()
```

[15]:
```
[<matplotlib.lines.Line2D at 0x10f788130>]
```

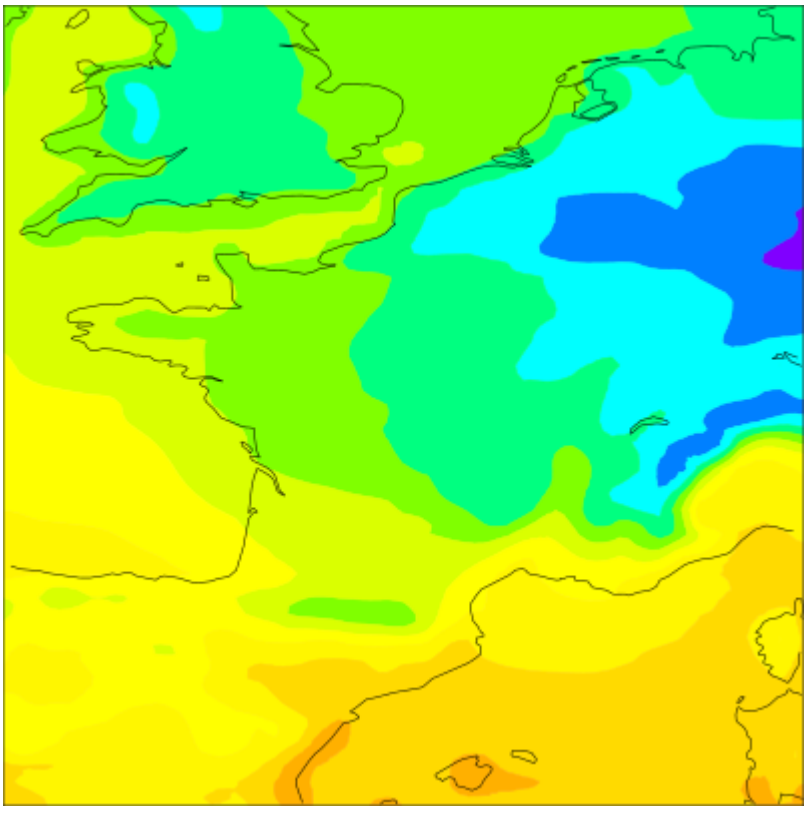


[16]:
```
x.t2m.plot()
```

[16]:
```
(array([6.200000e+01, 1.715000e+03, 2.437000e+04, 2.066870e+05,
        1.035682e+06, 2.078559e+06, 1.673685e+06, 6.841460e+05,
        9.120900e+04, 2.594000e+03]),
 array([247.29189, 254.00276, 260.71362, 267.4245 , 274.13538, 280.84625,
        287.55713, 294.268  , 300.97888, 307.68976, 314.40063],
       dtype=float32),
 <BarContainer object of 10 artists>)
```



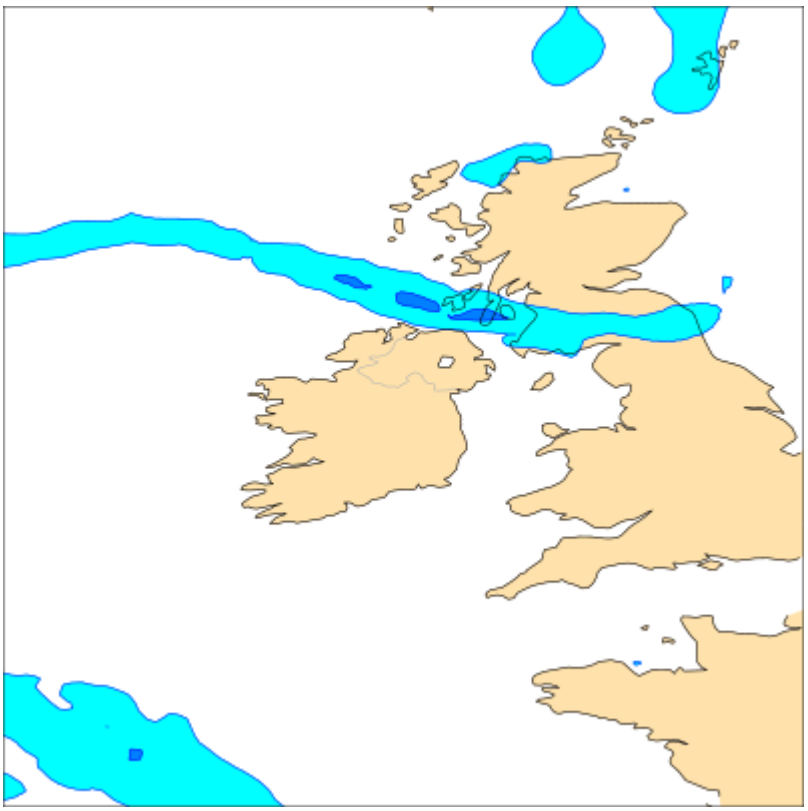
[17]:
```
cml.plot_map(x)
```



Precipitations from Ireland

[18]:
```
ds = cml.load_dataset(
    "era5-precipitations", period=(1979, 1982), domain="Ireland", time=12
)
```

[19]:
```
cml.plot_map(ds[0])
```



You can run this notebook in , in , in

[1]:
```
!pip install --quiet climetlab tensorflow
```

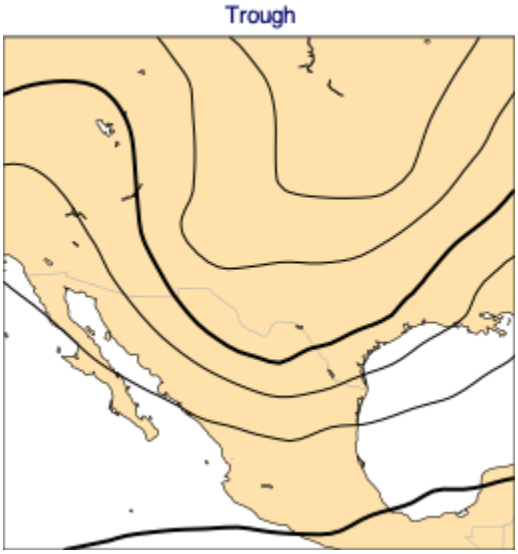
1.4.7 Machine learning example

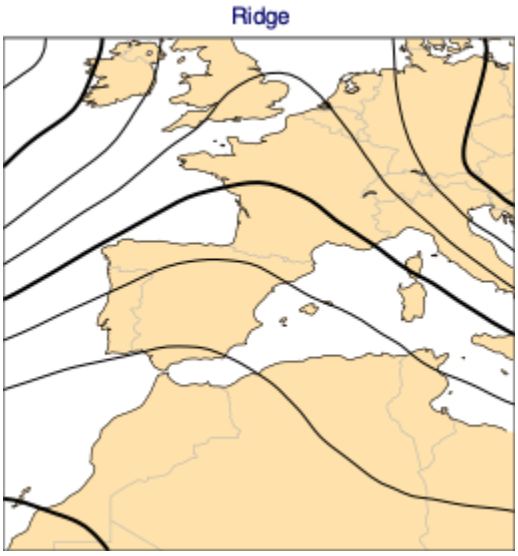
[2]:
```
from tensorflow.keras.layers import Input, Dense, Flatten
from tensorflow.keras.models import Sequential
```

[3]:
```
import climetlab as cml
```

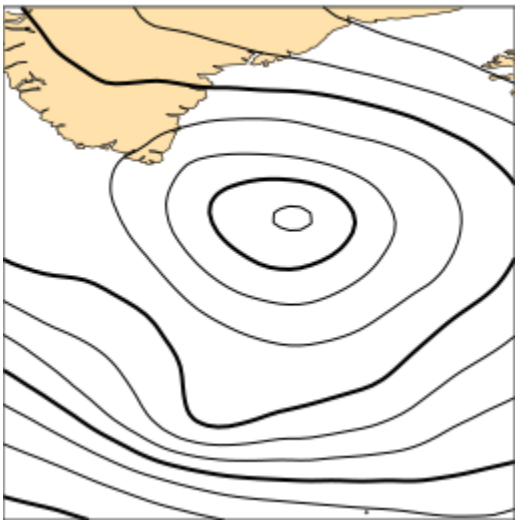
Load the high-low data set and plot all the fields and their label

[4]:
```
highlow = cml.load_dataset("high-low")
for field, label in highlow.fields():
    cml.plot_map(field, width=256, title=highlow.title(label))
```

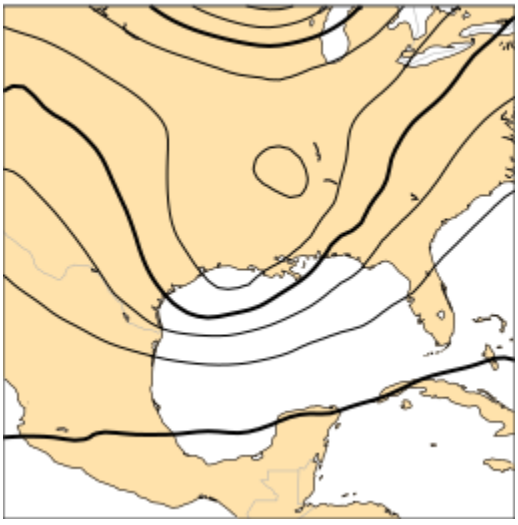




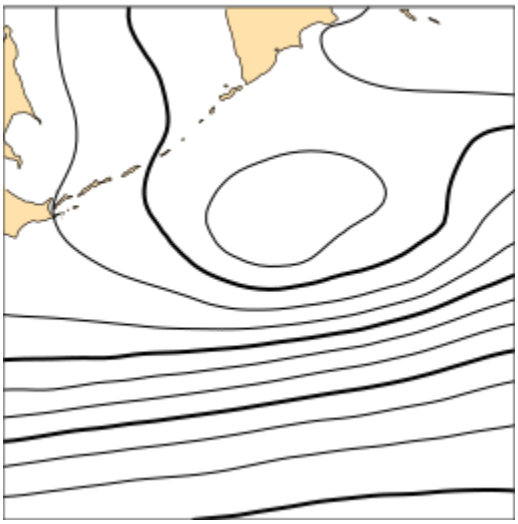
Low



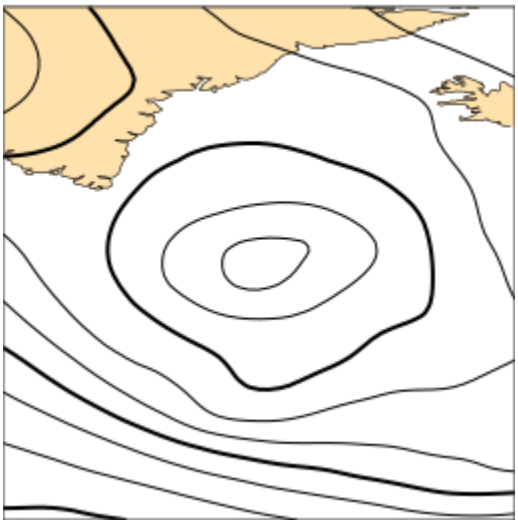
Trough



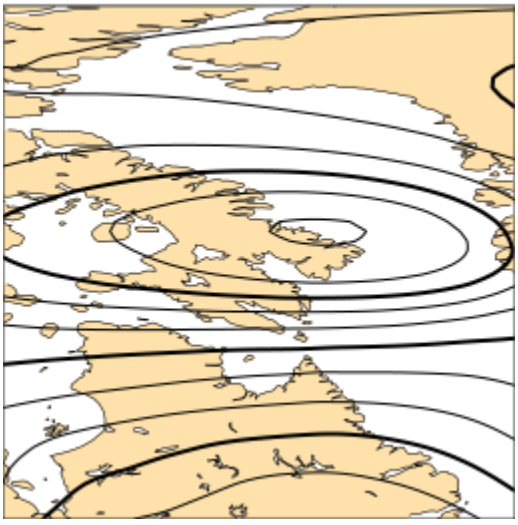
Trough



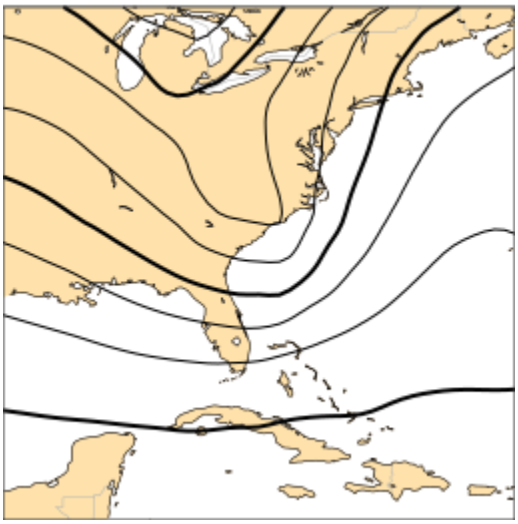
Low



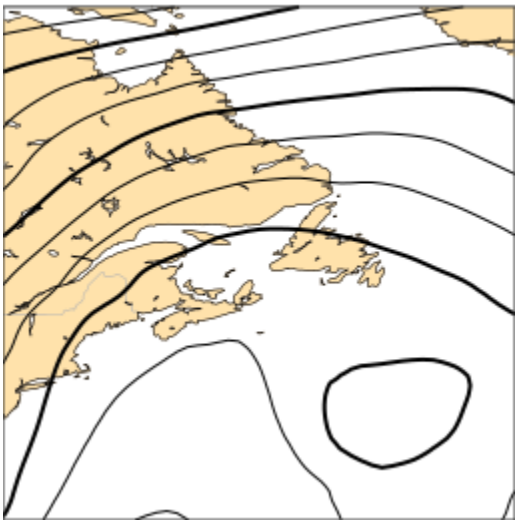
Low



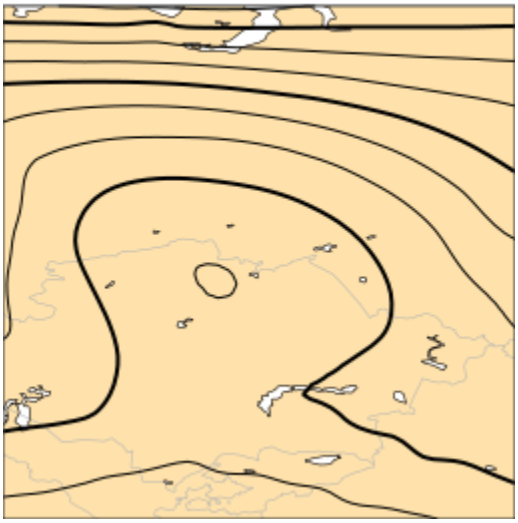
Trough



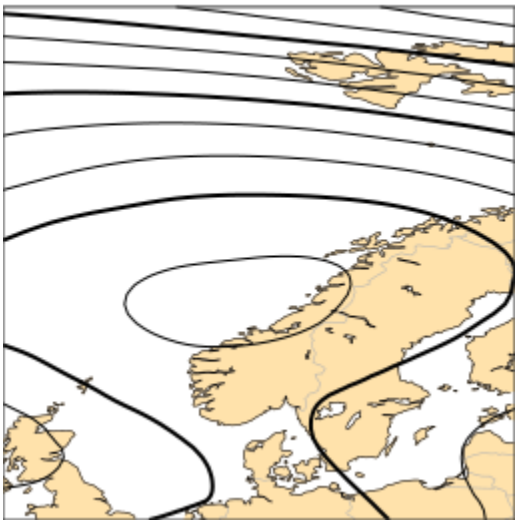
Ridge



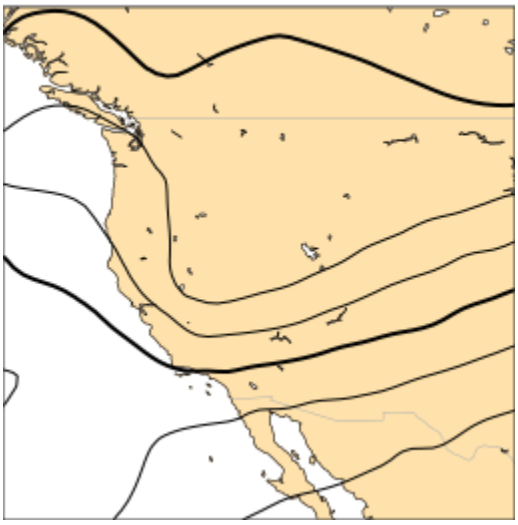
High



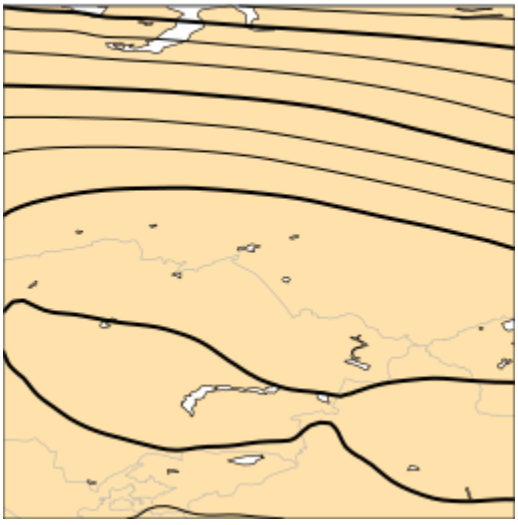
High



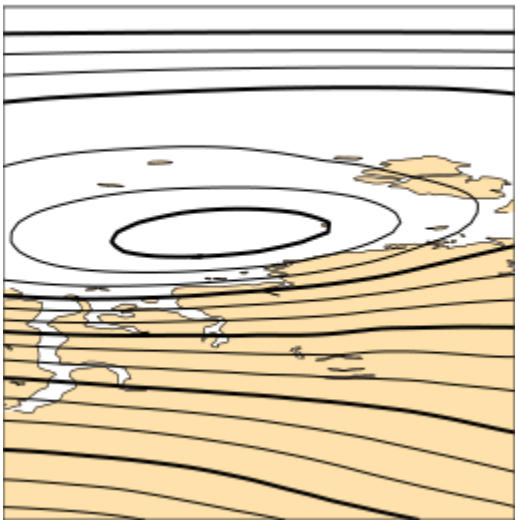
Trough



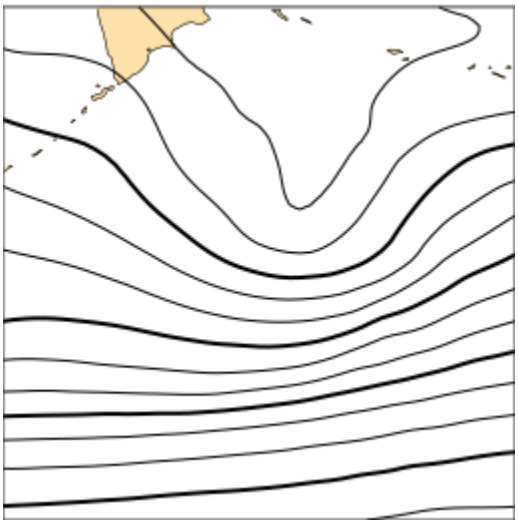
High



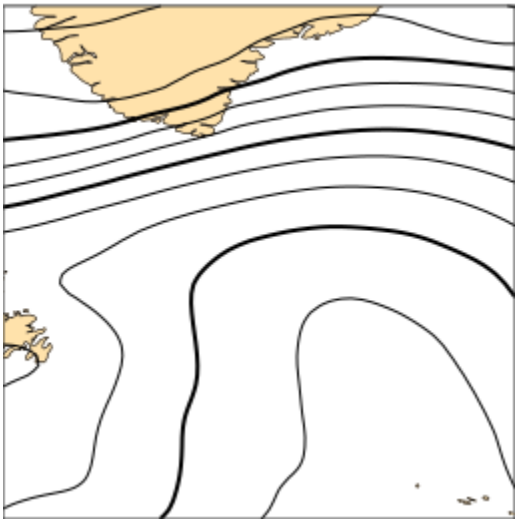
Low



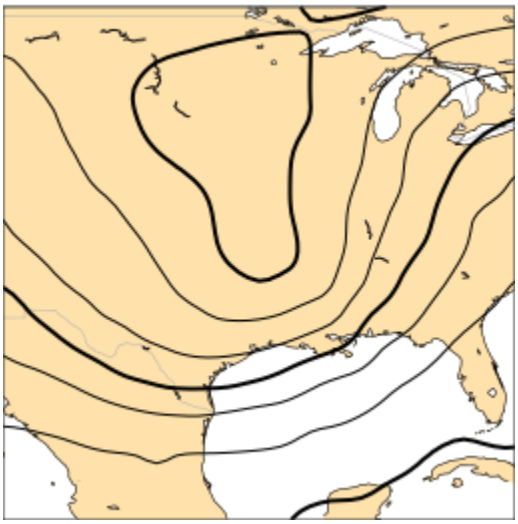
Trough



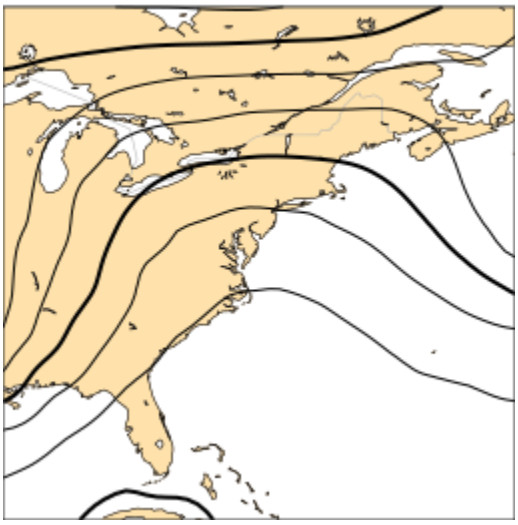
Ridge



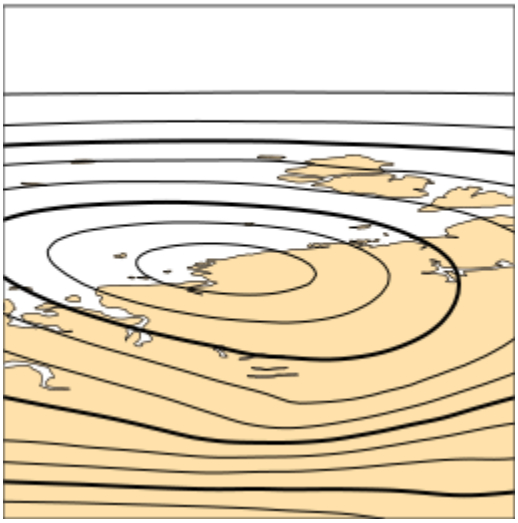
Trough



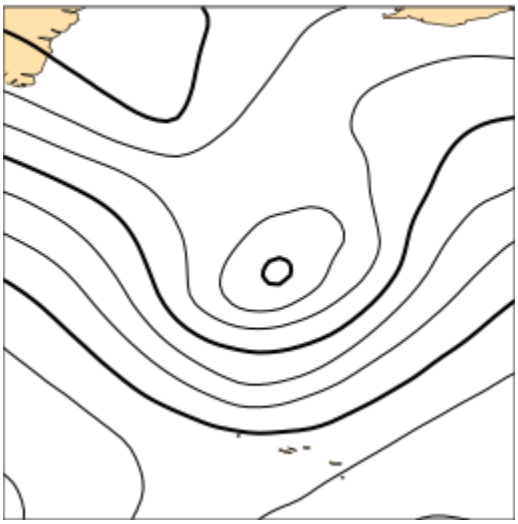
Ridge



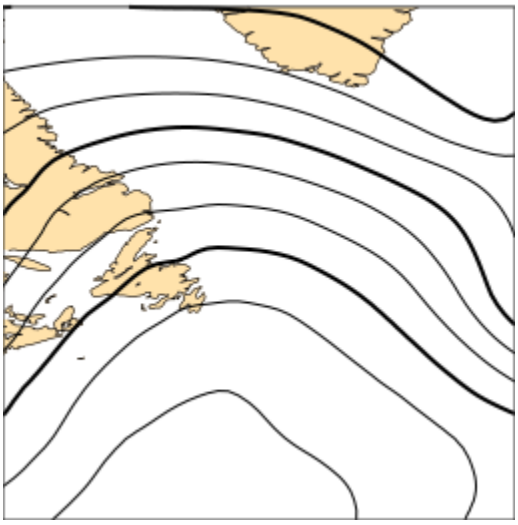
Low



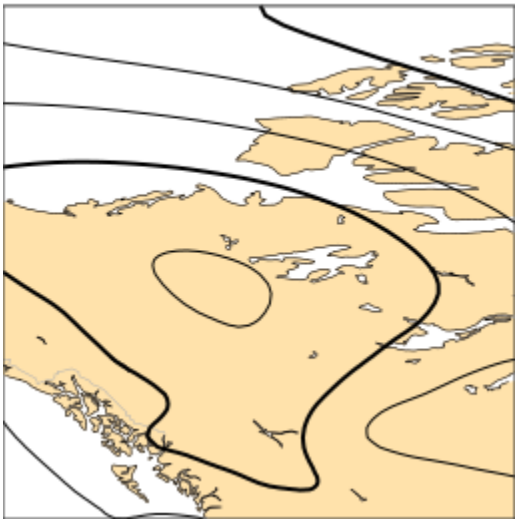
Trough



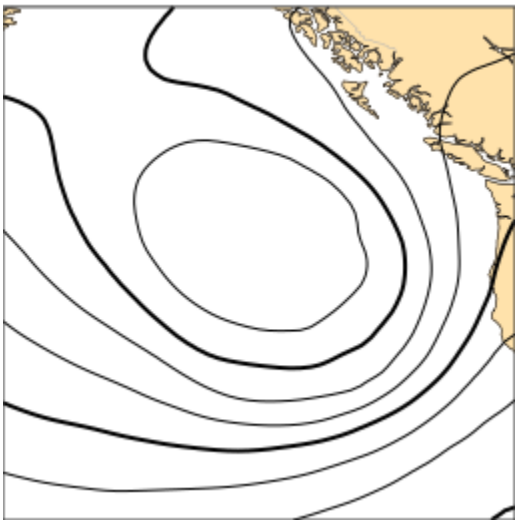
Ridge



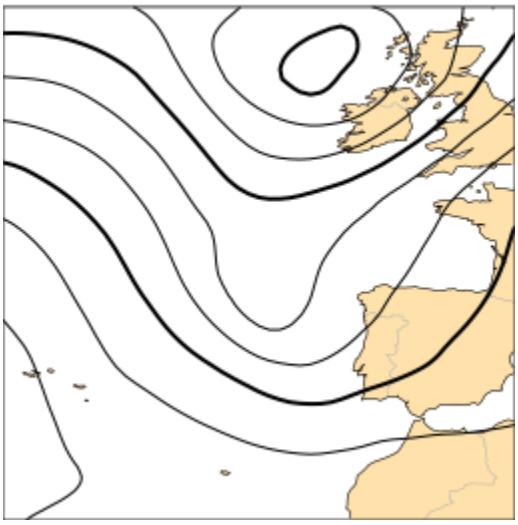
High



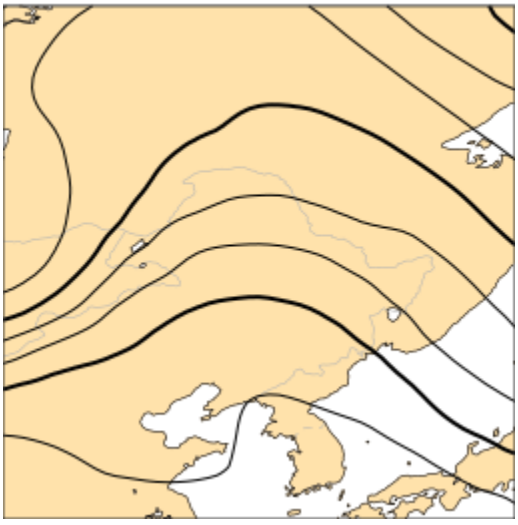
Low



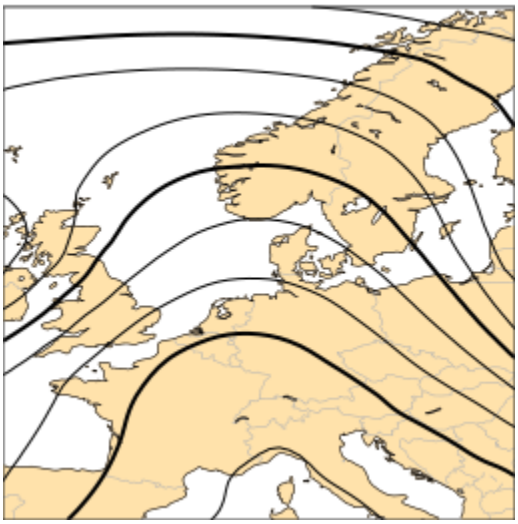
Trough



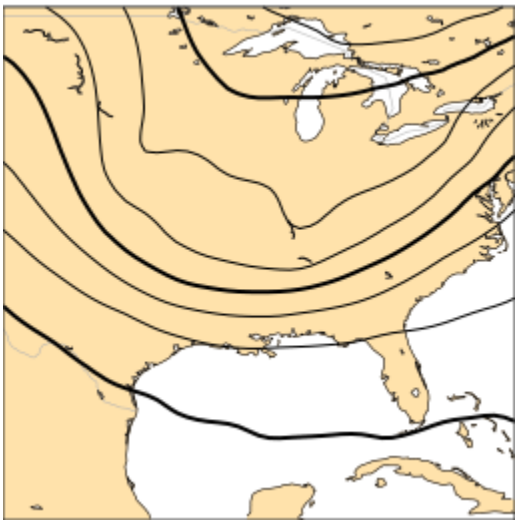
Ridge



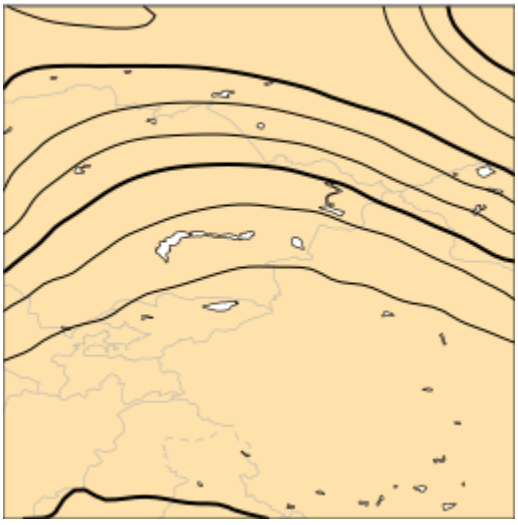
Ridge



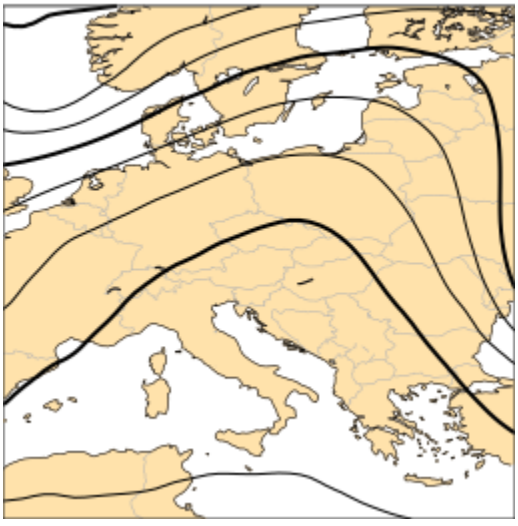
Trough



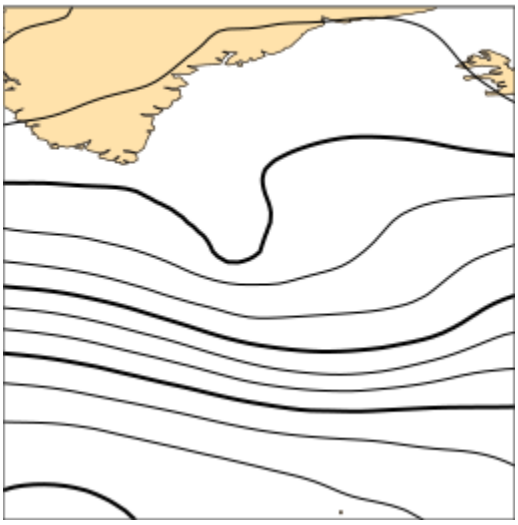
High



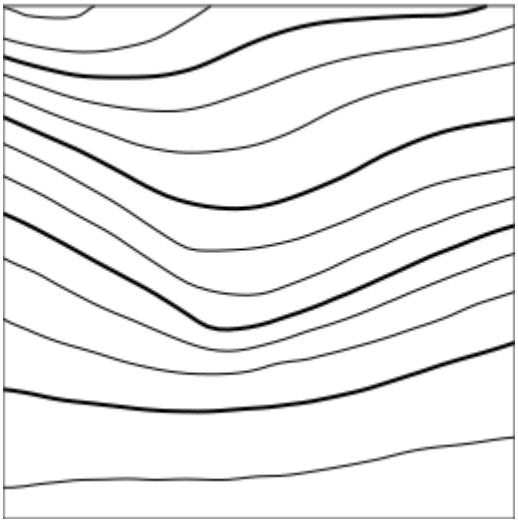
Ridge



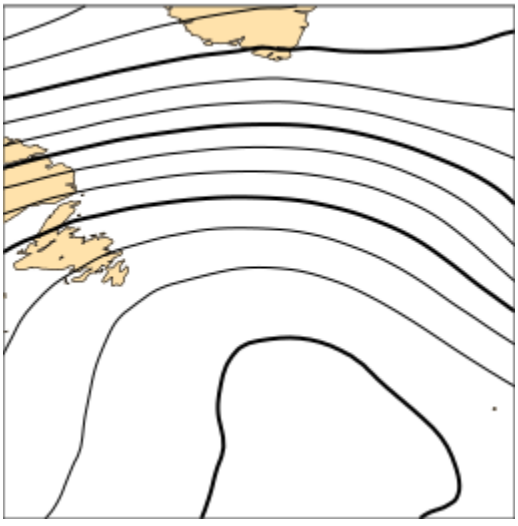
Trough



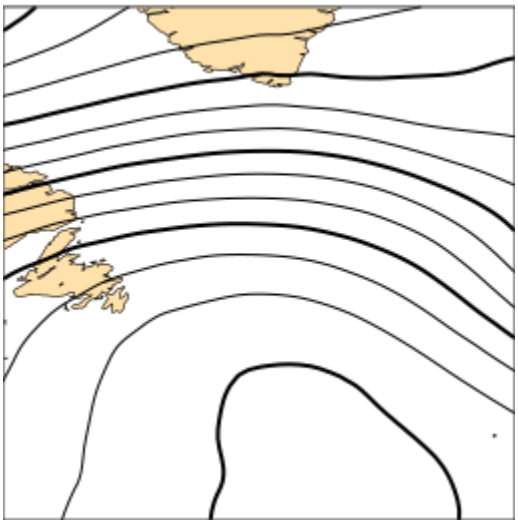
Trough



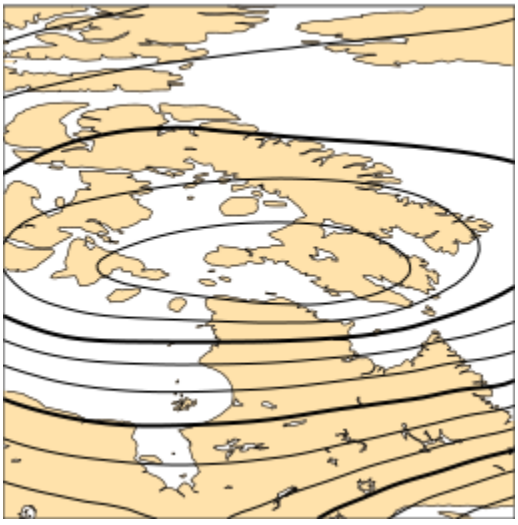
Ridge



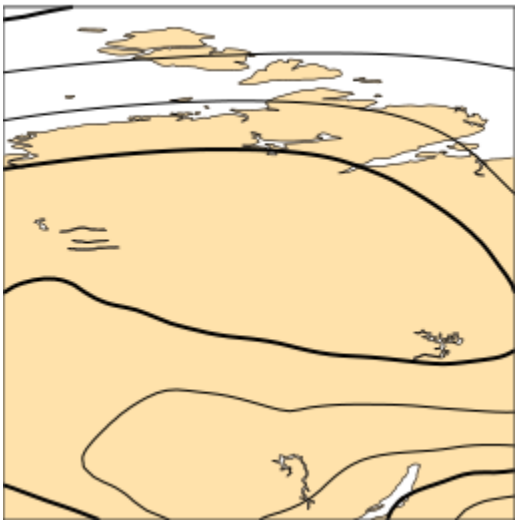
Ridge

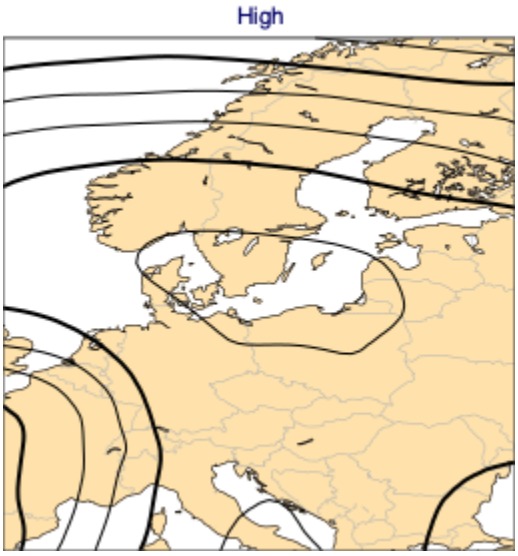


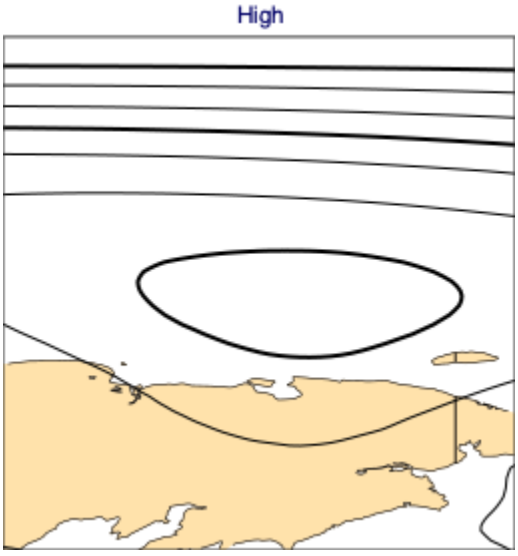
Low



High







Get the train and test sets

[5]:
```
(x_train, y_train, f_train), (x_test, y_test, f_test) = highlow.load_data(
    test_size=0.3, fields=True
)
```

Build the model

[6]:
```
model = Sequential()
model.add(Input(shape=x_train[0].shape))
model.add(Flatten())
model.add(Dense(64, activation="sigmoid"))
model.add(Dense(4, activation="softmax"))
```

[7]:
```
model.compile(optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"])
print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 441)               0
_________________________________________________________________
dense (Dense)                (None, 64)                28288
_________________________________________________________________
dense_1 (Dense)              (None, 4)                 260
=================================================================
Total params: 28,548
Trainable params: 28,548
Non-trainable params: 0
_________________________________________________________________
None
```

Train the model

[8]:
```
h = model.fit(x_train, y_train, epochs=100, verbose=0)
```

Evaluate the model on the test set

[9]:
```
model.evaluate(x_test, y_test)
```

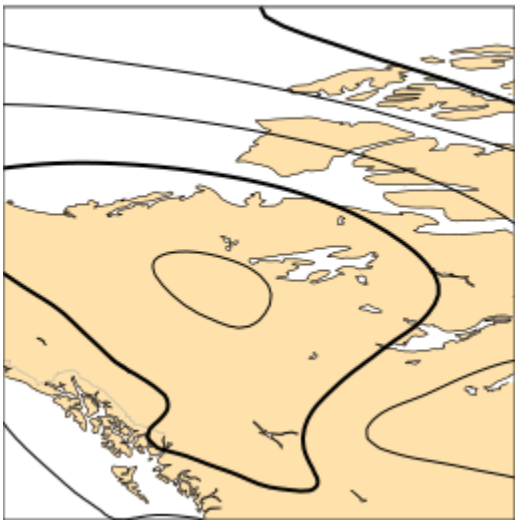
```
1/1 [==============================] - 0s 1ms/step - loss: 0.2691 - accuracy: 0.9167
```

[9]:
```
[0.26906612515449524, 0.9166666865348816]
```

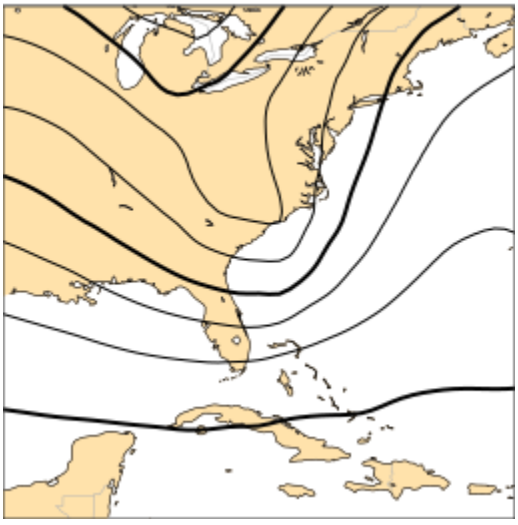
Plot the predictions

[10]:
```
predicted = model.predict(x_test)

for p, f in zip(predicted, f_test):
    cml.plot_map(f, width=256, title=highlow.title(p))
```

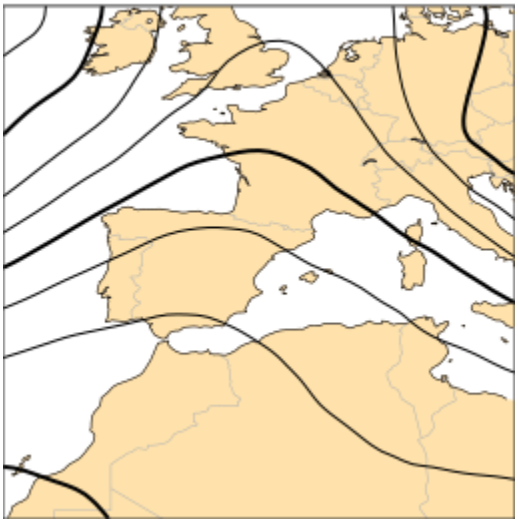
High (88%)



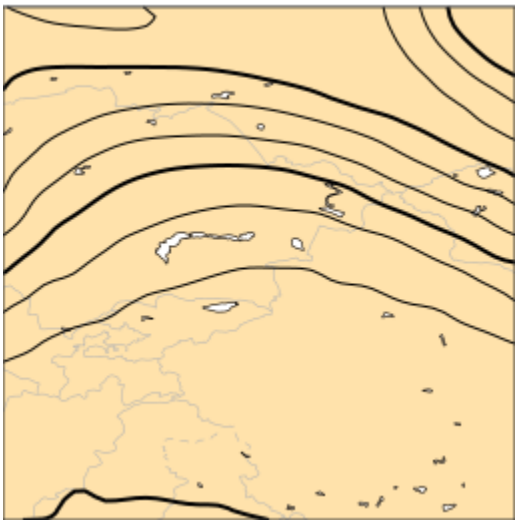
Trough (75%)



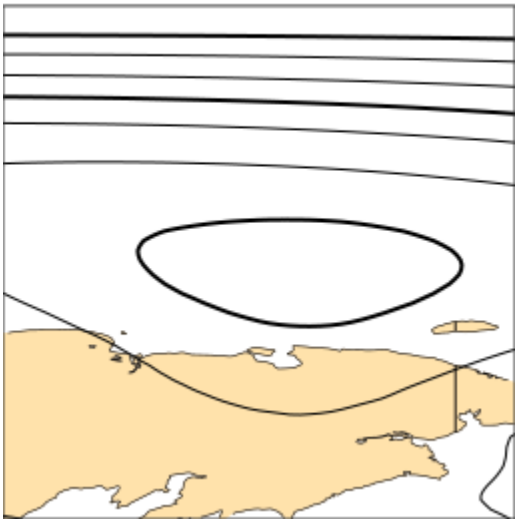
Ridge (90%)



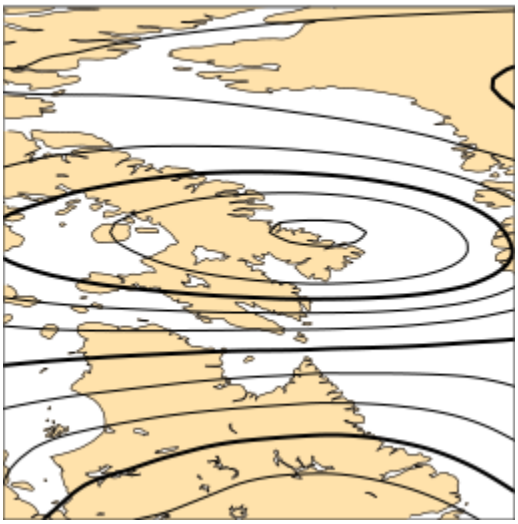
Ridge (67%)



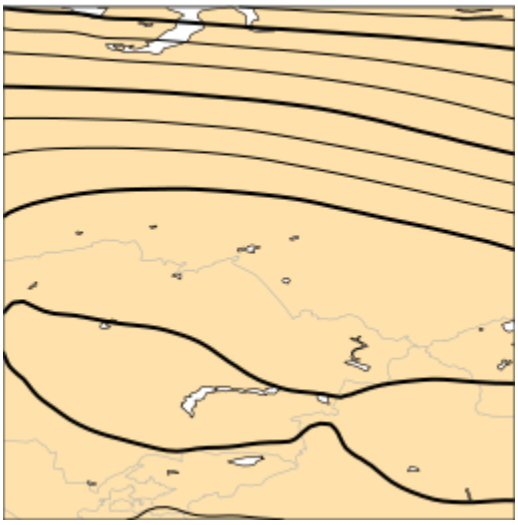
High (89%)



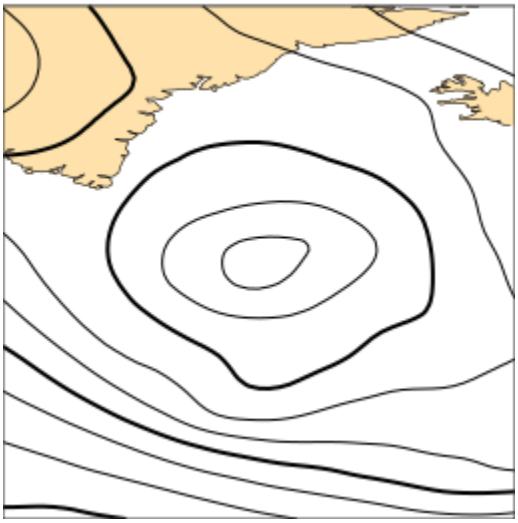
Low (94%)



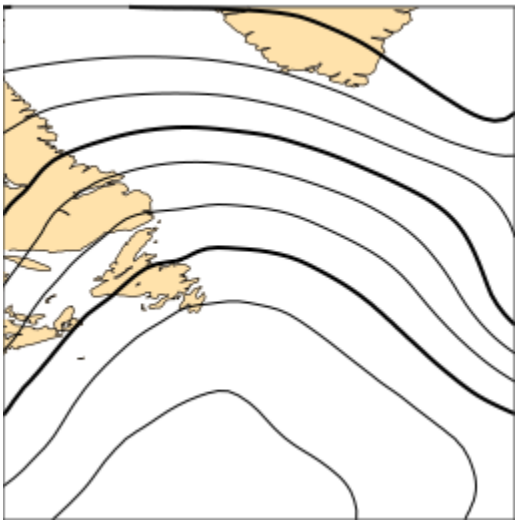
High (83%)



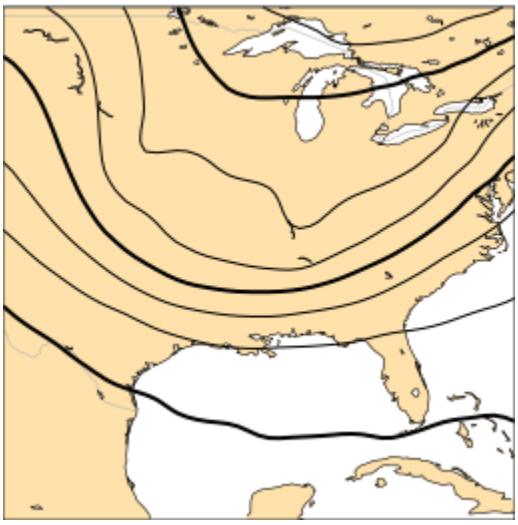
Low (94%)



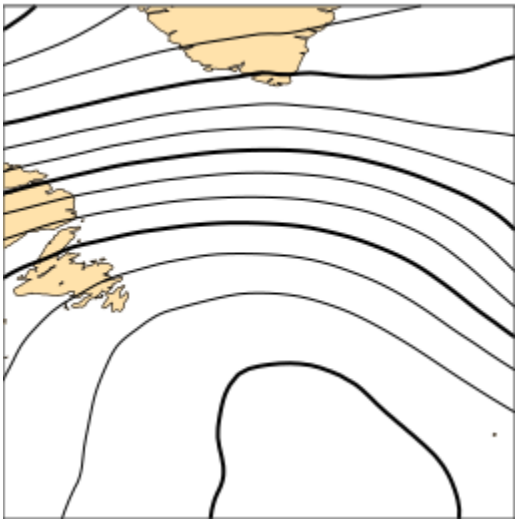
Ridge (70%)



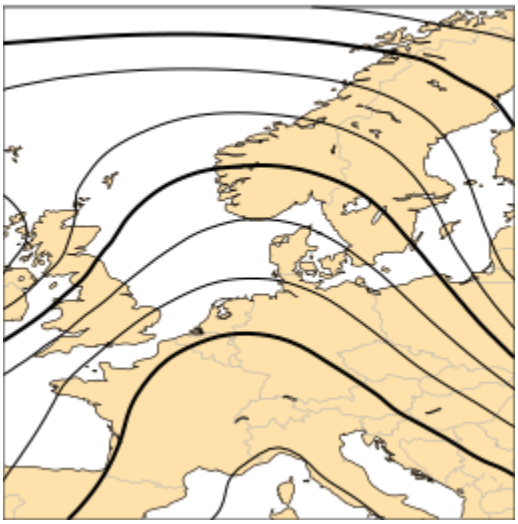
Trough (98%)



Ridge (80%)



Ridge (87%)



You can run this notebook in , in , in

[1]:
```
!pip install --quiet climetlab
```

1.4.8 BUFR data

[2]:
```
import climetlab as cml
```

[3]:
```
source = cml.load_dataset("sample-bufr-data")
```

[4]:
```
source
```

[4]:
```
YAML[/Users/baudouin/git/climetlab/climetlab/datasets/sample-bufr-data.yaml]
```

See https://github.com/ecmwf/pdbufr

[5]:
```
pd = source.to_pandas(
    columns=(
        "stationNumber",
        "latitude",
        "longitude",
        "data_datetime",
        "pressure",
        "airTemperature",
    ),
    filters={},
)
```

[6]:
```
pd.head()
```

[6]:
```
   stationNumber  latitude  longitude  pressure  airTemperature  \
0            907     58.47     -78.08  100300.0           258.3
1            907     58.47     -78.08  100000.0           259.7
2            907     58.47     -78.08   99800.0           261.1
3            907     58.47     -78.08   99100.0           261.7
4            907     58.47     -78.08   92500.0           258.1

        data_datetime
0 2008-12-08 12:00:00
1 2008-12-08 12:00:00
2 2008-12-08 12:00:00
3 2008-12-08 12:00:00
4 2008-12-08 12:00:00
```

[7]:
```
pd.tail()
```

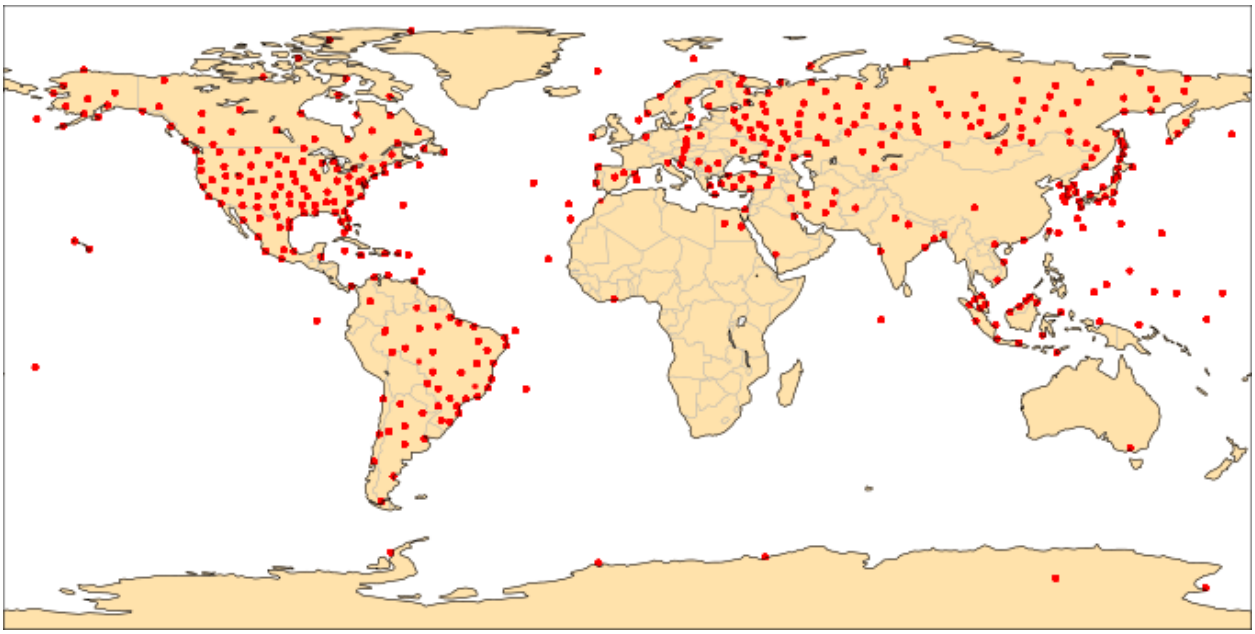
[7]:
```
       stationNumber  latitude  longitude  pressure  airTemperature  \
26191            968     25.03     121.52   10000.0           197.9
26192            968     25.03     121.52    9520.0           196.3
26193            968     25.03     121.52    7000.0           201.5
26194            968     25.03     121.52    5000.0           209.1
26195            968     25.03     121.52    3000.0           217.3
```

(continues on next page)

(continued from previous page)

```
            data_datetime
26191 2008-12-08 12:00:00
26192 2008-12-08 12:00:00
26193 2008-12-08 12:00:00
26194 2008-12-08 12:00:00
26195 2008-12-08 12:00:00
```

[8]:
```
cml.plot_map(pd, projection="global")
```



You can run this notebook in , in , in

[1]:
```
!pip install --quiet climetlab[interactive]
```

1.4.9 ECMWF Observation bespoke format (ODB)

[2]:
```
import climetlab as cml
```

[3]:
```
source = cml.load_source(
    "mars",
    type="ofb",
    obsgroup="conv",
    time="12",
    format="odb",
    reportype=16001,
    date="2020-05-18",
)
```

[4]:
```
pd = source.to_pandas()
```

[5]:
```
pd
```

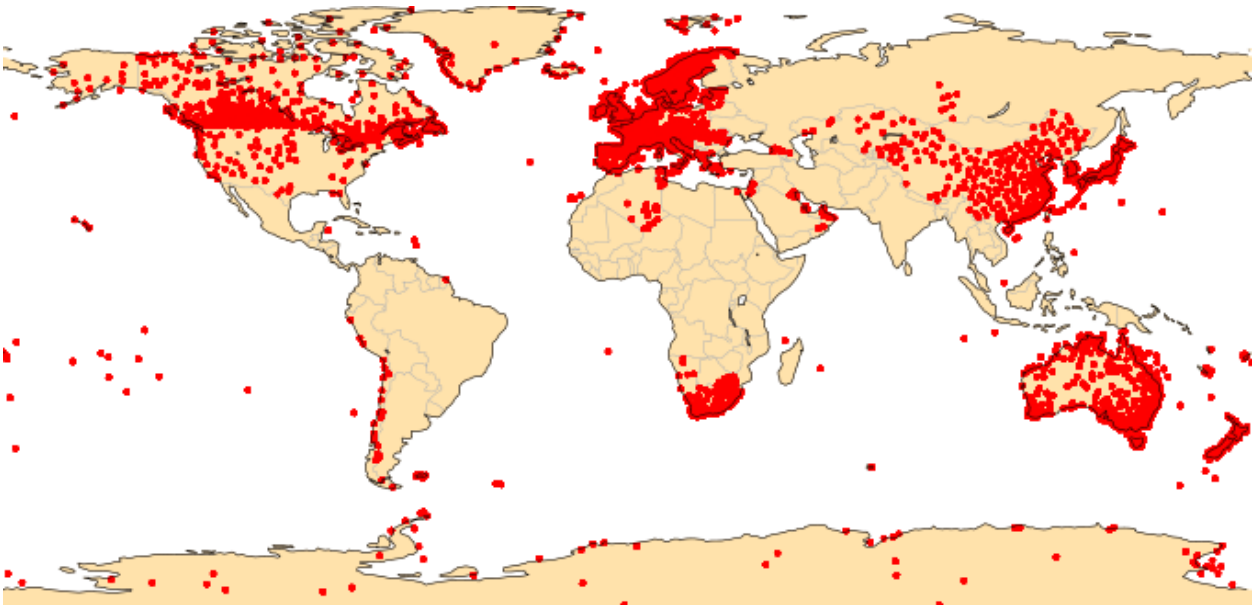
[5]:
```
        type    expver  class  stream    andate  antime  reportype  \
0        263      0001      1    1025  20200518  120000      16001
1        263      0001      1    1025  20200518  120000      16001
2        263      0001      1    1025  20200518  120000      16001
3        263      0001      1    1025  20200518  120000      16001
4        263      0001      1    1025  20200518  120000      16001
...      ...       ...    ...     ...       ...     ...        ...
410015   263      0001      1    1025  20200518  120000      16001
410016   263      0001      1    1025  20200518  120000      16001
410017   263      0001      1    1025  20200518  120000      16001
410018   263      0001      1    1025  20200518  120000      16001
410019   263      0001      1    1025  20200518  120000      16001

        numtsl@desc  timeslot@timeslot_index  seqno@hdr  ...  \
0                17                       15       4443  ...
1                17                       15       4443  ...
2                17                       15       4443  ...
3                17                       15       4443  ...
4                17                       15       4443  ...
...             ...                      ...        ...  ...
410015           17                       15    9133718  ...
410016           17                       15    9133718  ...
410017           17                       15    9133718  ...
410018           17                       15    9133718  ...
410019           17                       15    9133718  ...

        datum_rdbflag@body  biascorr@body  biascorr_fg@body  qc_pge@body  \
0                        0            NaN               NaN          NaN
1                        0            NaN               NaN          NaN
2                        0            0.0               0.0          NaN
3                        0            0.0               0.0          NaN
4                        0            0.0               0.0          NaN
...                    ...            ...               ...          ...
410015                   0            0.0               0.0          NaN
410016                   0            0.0               0.0          NaN
410017                   0            0.0               0.0          NaN
410018                   0            0.0               0.0          NaN
410019                   0            0.0               0.0          NaN

        an_depar@body  fg_depar@body obs_error@errstat  \
0          -10.560810     -14.283884         41.155659
1           36.250599      34.712360         41.155659
2           -9.160848     -12.384340         33.596455
3                 NaN            NaN               NaN
4            2.482570       2.683633          1.408449
...               ...            ...               ...
410015            NaN            NaN               NaN
410016            NaN            NaN               NaN
410017            NaN            NaN               NaN
410018            NaN            NaN               NaN
410019            NaN            NaN               NaN
```

(continues on next page)

(continued from previous page)

```
       final_obs_error@errstat  fg_error@errstat  eda_spread@errstat
0                    41.155659         25.144444           13.044278
1                    41.155659         25.144444           13.044278
2                    33.596455          0.260278            0.011694
3                          NaN               NaN                 NaN
4                     1.408449          0.502528            0.395115
...                        ...               ...                 ...
410015                     NaN               NaN                 NaN
410016                     NaN               NaN                 NaN
410017                     NaN               NaN                 NaN
410018                     NaN               NaN                 NaN
410019                     NaN               NaN                 NaN

[410020 rows x 51 columns]
```

[6]:
```
cml.plot_map(pd)
```



You can run this notebook in , in , in

[1]:
```
# !pip install --quiet climetlab
```

1.4.10 Meteonet

This is an retrieve an plot various data types from https://github.com/meteofrance/meteonet

[2]:
```
import climetlab as cml
```

Radar images

[3]:
```
ds = cml.load_dataset("meteonet-samples-radar")
```

[4]:
```
ds
```

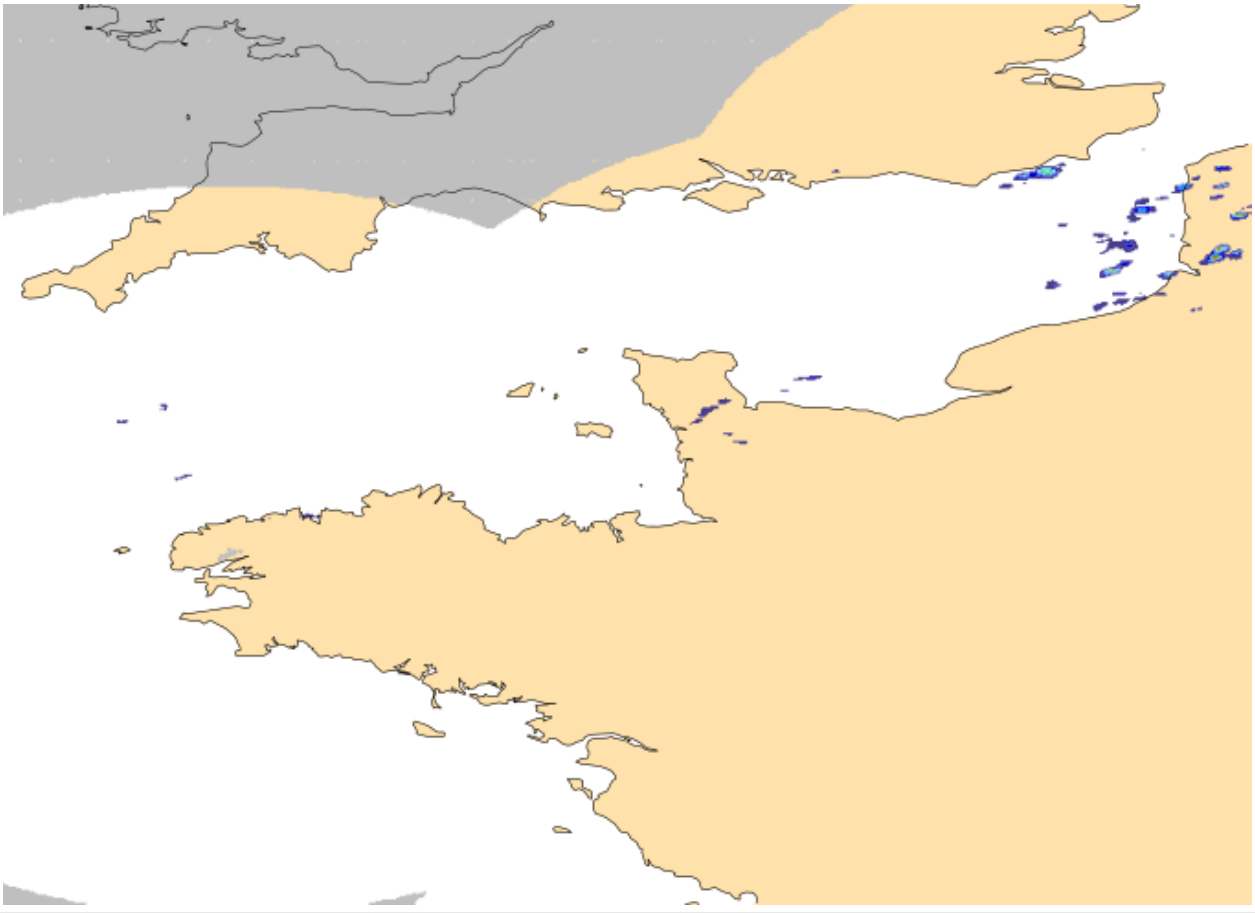
[4]:
```
<climetlab.datasets.meteonet_samples.radar.MeteonetRadar at 0x104319b50>
```

[5]:
```
ds.licence
```

[5]:
```
'https://meteonet.umr-cnrm.fr/dataset/LICENCE.md'
```

[6]:
```
cml.plot_map(ds)
```

```
/Users/baudouin/git/climetlab/climetlab/normalize.py:17: UserWarning: Deprecated␣
↪decorator @normalize_arg. Use @normalise on each argument instead.
  warnings.warn(
```



[7]:
```
ds.to_xarray()
```

[7]:
```
<xarray.Dataset>
Dimensions:   (time: 45, y: 565, x: 784)
Coordinates:
  * x         (x) int64 0 1 2 3 4 5 6 7 8 ... 776 777 778 779 780 781 782 783
  * y         (y) int64 0 1 2 3 4 5 6 7 8 ... 557 558 559 560 561 562 563 564
```

(continues on next page)

(continued from previous page)

```
    lon       (y, x) float64 -5.832 -5.822 -5.812 -5.802 ... 1.978 1.988 1.998
    lat       (y, x) float64 51.89 51.89 51.89 51.89 ... 46.25 46.25 46.25 46.25
  * time      (time) datetime64[ns] 2016-08-21T00:10:00 ... 2016-08-31T00:30:00
Data variables:
    rainfall  (time, y, x) int16 -1 -1 -1 -1 -1 -1 -1 -1 -1 ... 0 0 0 0 0 0 0 0
```

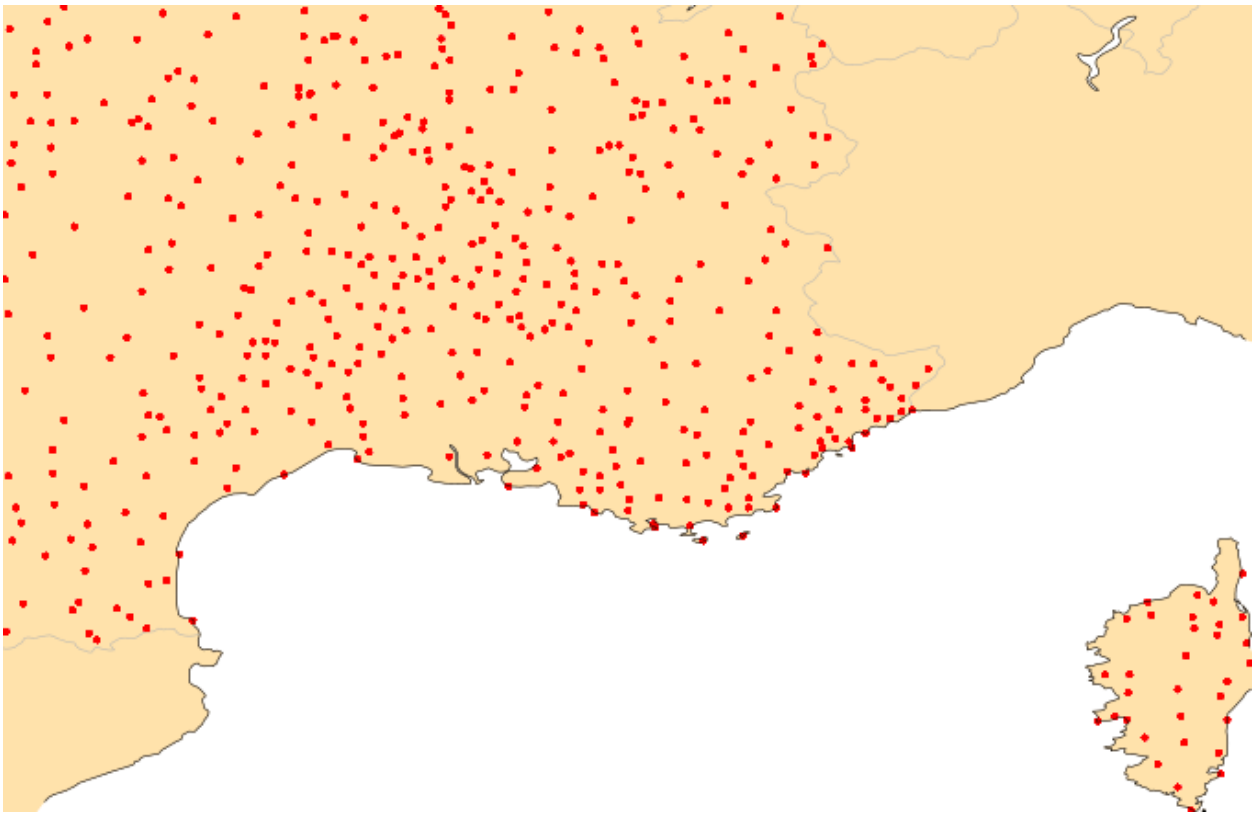
Ground station observations

[8]:
```
ds = cml.load_dataset("meteonet-samples-ground-stations", domain="SE")
```

[9]:
```
ds.to_pandas()
```

[9]:
```
        number_sta       lat      lon  height_sta                date     dd  \
0          1027003  45.83000  5.11000       196.0 2016-01-01 00:00:00    NaN
1          1033002  46.09000  5.81000       350.0 2016-01-01 00:00:00    0.0
2          1034004  45.77000  5.69000       330.0 2016-01-01 00:00:00    0.0
3          1072001  46.20000  5.29000       260.0 2016-01-01 00:00:00    NaN
4          1089001  45.98000  5.33000       252.0 2016-01-01 00:00:00    0.0
...            ...       ...      ...         ...                 ...    ...
111618    84085004  43.94000  5.23000       488.0 2016-01-01 23:54:00  100.0
111619    84086001  43.81000  5.15000       672.0 2016-01-01 23:54:00  140.0
111620    84087001  44.14000  4.86000        55.0 2016-01-01 23:54:00  130.0
111621    84107002  44.04067  5.49283       836.0 2016-01-01 23:54:00  120.0
111622    84150001  44.34000  4.91000       141.0 2016-01-01 23:54:00  110.0

         ff  precip     hu      td       t       psl
0       NaN     NaN   98.0  278.75  279.05       NaN
1       0.0     0.0   99.0  278.25  278.35       NaN
2       0.0     0.0  100.0  279.15  279.15       NaN
3       NaN     0.0    NaN     NaN  276.55       NaN
4       0.0     0.0   95.0  278.85  279.55  102720.0
...     ...     ...    ...     ...     ...       ...
111618  2.0     0.0   94.0  280.05  280.95       NaN
111619  7.6     0.0   93.0  279.65  280.75       NaN
111620  5.0     0.0   76.0  281.25  285.35  101760.0
111621  3.8     0.2   99.0  278.55  278.65       NaN
111622  3.1     0.0   82.0  281.85  284.85       NaN

[111623 rows x 12 columns]
```

[10]:
```
cml.plot_map(ds)
```



Weather models

[11]:
```
ds = cml.load_dataset("meteonet-samples-weather-models", model="arome")
```

[12]:
```
ds.to_xarray()
```

[12]:
```
<xarray.Dataset>
Dimensions:            (time: 1, step: 25, heightAboveGround: 1, latitude: 227,␣
↪longitude: 315)
Coordinates:
  * time               (time) datetime64[ns] 2018-05-01
  * step               (step) timedelta64[ns] 00:00:00 ... 1 days 00:00:00
  * heightAboveGround  (heightAboveGround) int64 2
  * latitude           (latitude) float64 51.9 51.87 51.85 ... 46.3 46.27 46.25
  * longitude          (longitude) float64 -5.842 -5.817 -5.792 ... 1.983 2.008
    valid_time         (time, step) datetime64[ns] dask.array<chunksize=(1, 25), meta=np.
↪ndarray>
Data variables:
    t2m                (time, step, heightAboveGround, latitude, longitude) float32 dask.
↪array<chunksize=(1, 25, 1, 227, 315), meta=np.ndarray>
    d2m                (time, step, heightAboveGround, latitude, longitude) float32 dask.
↪array<chunksize=(1, 25, 1, 227, 315), meta=np.ndarray>
    r                  (time, step, heightAboveGround, latitude, longitude) float32 dask.
↪array<chunksize=(1, 25, 1, 227, 315), meta=np.ndarray>
Attributes:
```

(continues on next page)

(continued from previous page)

```
    GRIB_edition:            1
    GRIB_centre:             lfpw
    GRIB_centreDescription:  French Weather Service - Toulouse
    GRIB_subCentre:          0
    Conventions:             CF-1.7
    institution:             French Weather Service - Toulouse
    history:                 2021-11-13T18:08:50 GRIB to CDM+CF via cfgrib-0...
```

[13]:
```
ds = cml.load_dataset(
    "meteonet-samples-weather-models", model="arpege", variable="P_sea_level"
)
```

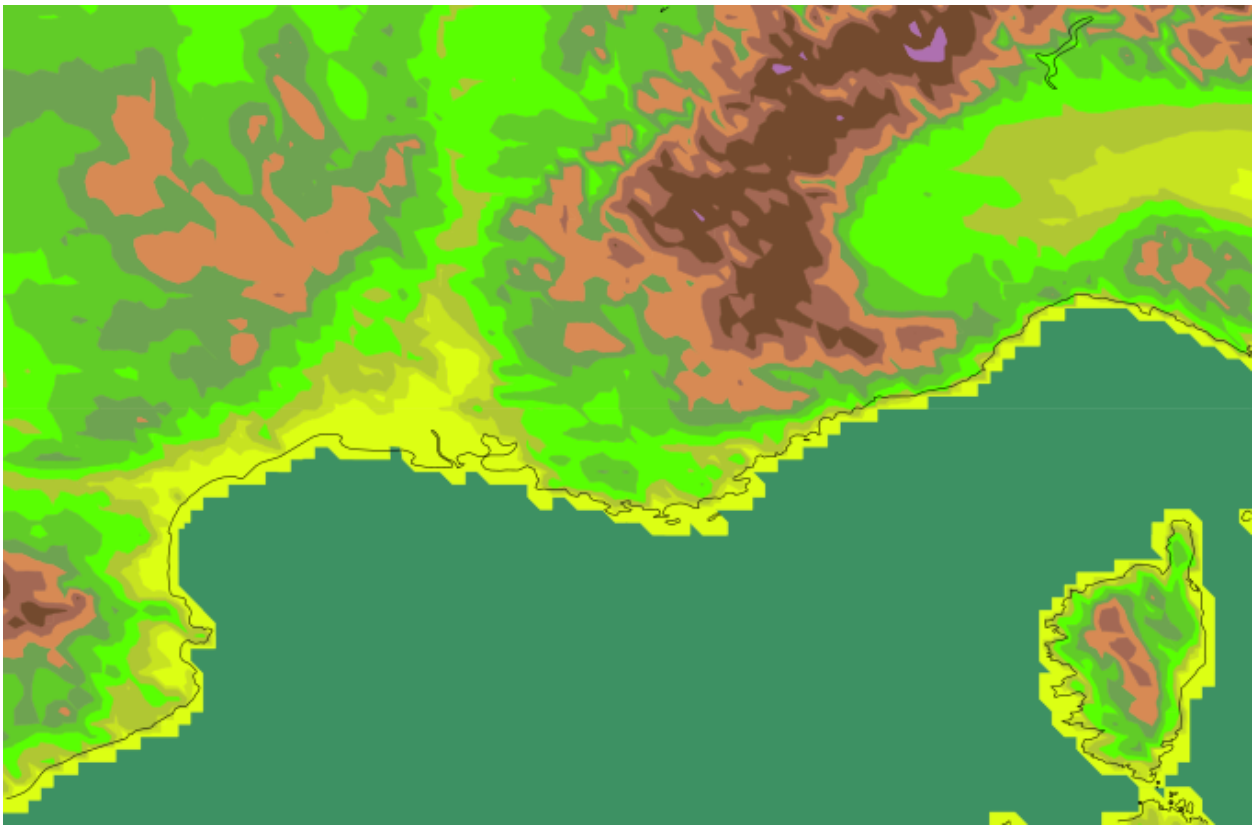
[14]:
```
cml.plot_map(ds[0])
```

Masks

[15]:
```
ds = cml.load_dataset("meteonet-samples-masks", domain="SE")
```

[16]:
```
cml.plot_map(ds[0])
```

[17]:
```
cml.plot_map(ds[1])
```



You can run this notebook in , in , in

[1]:
```
!pip install --quiet climetlab matplotlib
```

1.4.11 WeatherBench

This is an attempt to reproduce this research: https://arxiv.org/abs/2002.00469. There is a notebook available at: https://binder.pangeo.io/v2/gh/pangeo-data/WeatherBench/master?filepath=quickstart.ipynb

[2]:
```
import matplotlib.pyplot as plt
```

[3]:
```
import climetlab as cml
```

[ ]:
```
ds = cml.load_dataset("weather-bench")
```

[5]:
```
ds
```

[5]:
```
<climetlab.datasets.weather_bench.WeatherBench at 0x7f621c0638b0>
```

[6]:
```
print(ds.citation)
```

```
@article{rasp2020weatherbench,
  title={WeatherBench: A benchmark dataset for data-driven weather forecasting},
  author={Rasp, Stephan and Dueben, Peter D and Scher, Sebastian and Weyn,
  Jonathan A and Mouatadid, Soukayna and Thuerey, Nils},
```

(continues on next page)

(continued from previous page)

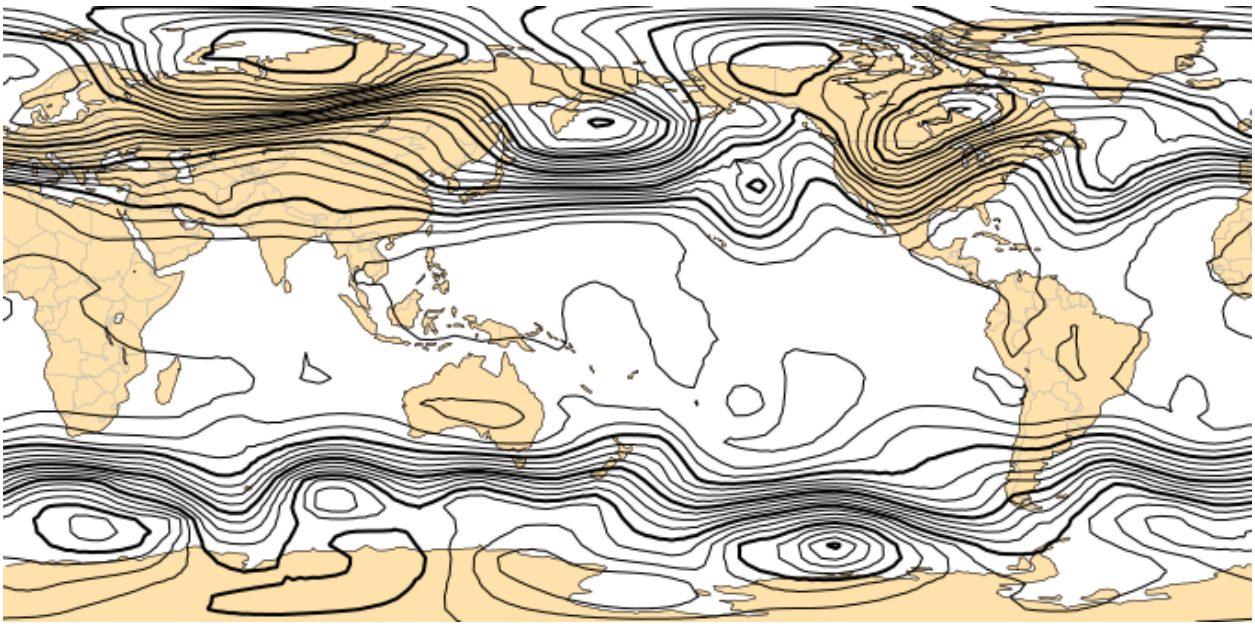
```
  journal={arXiv preprint arXiv:2002.00469},
  year={2020}
}
```

[7]:
```
z500 = ds.to_xarray()
```

[8]:
```
z500
```

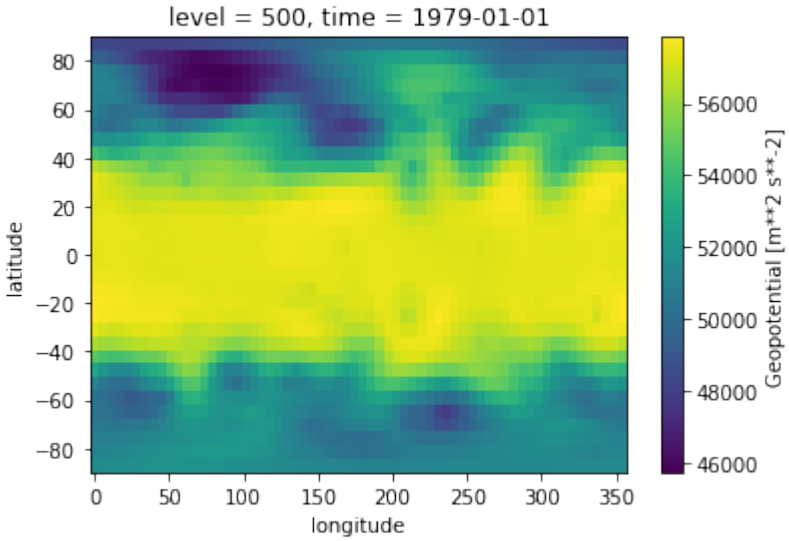
[8]:
```
<xarray.Dataset>
Dimensions:  (lat: 32, lon: 64, time: 350640)
Coordinates:
    level    int32 500
  * lon      (lon) float64 0.0 5.625 11.25 16.88 ... 337.5 343.1 348.8 354.4
  * lat      (lat) float64 -87.19 -81.56 -75.94 -70.31 ... 75.94 81.56 87.19
  * time     (time) datetime64[ns] 1979-01-01 ... 2018-12-31T23:00:00
Data variables:
    z        (time, lat, lon) float32 dask.array<chunksize=(8760, 32, 64), meta=np.
↪ndarray>
Attributes:
    Conventions:  CF-1.6
    history:      2019-11-10 20:33:23 GMT by grib_to_netcdf-2.14.0: /opt/ecmw...
```

[9]:
```
cml.plot_map(z500)
```

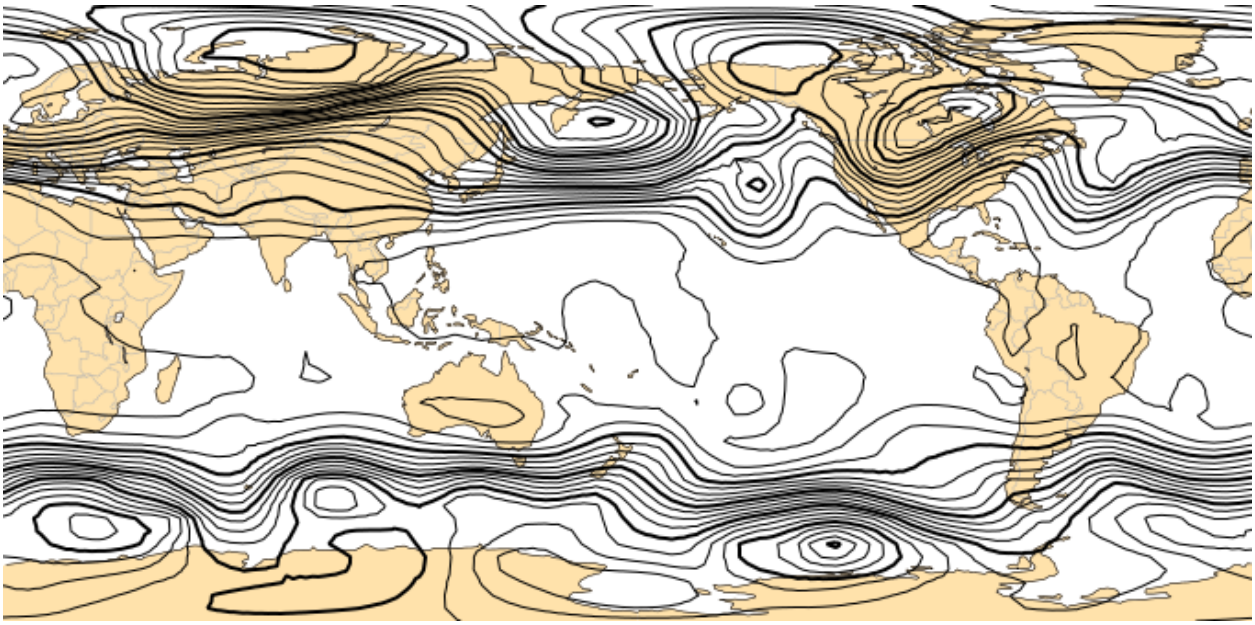


[10]:
```
z500.z.isel(time=0).plot()
```

[10]:
```
<matplotlib.collections.QuadMesh at 0x7f62146b2cd0>
```



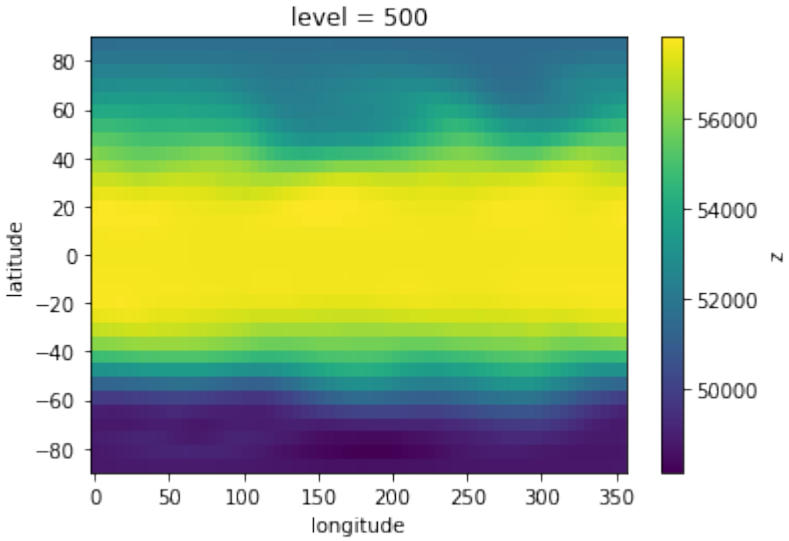
[11]:
```
cml.plot_map(z500.z.isel(time=0))
```



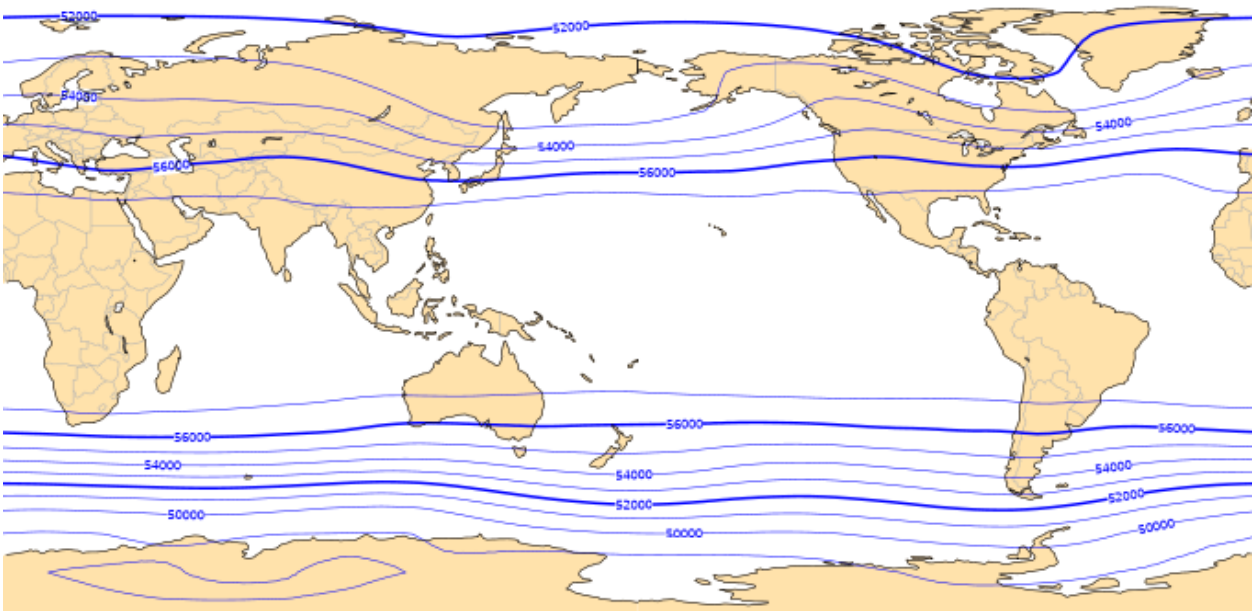
[12]:
```
climatology = z500.sel(time=slice("2016", "2016")).mean("time").load()
```

[13]:
```
climatology.z.plot()
```

[13]:
```
<matplotlib.collections.QuadMesh at 0x7f62145059a0>
```



[14]:
```
cml.plot_map(climatology.z)
```



[15]:
```
climatology.z
```

[15]:
```
<xarray.DataArray 'z' (lat: 32, lon: 64)>
array([[48765.18 , 48774.066, 48782.83 , ..., 48745.145, 48750.582,
        48757.645],
       [48755.164, 48798.348, 48845.02 , ..., 48690.23 , 48700.72 ,
        48721.727],
       [48890.742, 48941.703, 48997.42 , ..., 48815.234, 48822.87 ,
        48850.105],
       ...,
       [52378.613, 52398.484, 52420.074, ..., 52361.254, 52363.426,
        52366.344],
```

(continues on next page)

(continued from previous page)

```
       [51937.207, 51943.78 , 51950.414, ..., 51906.508, 51919.223,
        51928.72 ],
       [51565.38 , 51571.426, 51578.15 , ..., 51543.97 , 51551.11 ,
        51558.305]], dtype=float32)
Coordinates:
    level    int32 500
  * lon      (lon) float64 0.0 5.625 11.25 16.88 ... 337.5 343.1 348.8 354.4
  * lat      (lat) float64 -87.19 -81.56 -75.94 -70.31 ... 75.94 81.56 87.19
```

You can run this notebook in , in , in

[55]:
```
!pip install --quiet climetlab matplotlib
```

```
DEPRECATION: Configuring installation scheme with distutils config files is deprecated␣
↪and will no longer work in the near future. If you are using a Homebrew or Linuxbrew␣
↪Python, please see discussion at https://github.com/Homebrew/homebrew-core/issues/76621
DEPRECATION: Configuring installation scheme with distutils config files is deprecated␣
↪and will no longer work in the near future. If you are using a Homebrew or Linuxbrew␣
↪Python, please see discussion at https://github.com/Homebrew/homebrew-core/issues/76621
```

1.4.12 NOAA's hurricane database

[56]:
```
import climetlab as cml
```

[57]:
```
atlantic = cml.load_dataset("hurricane-database", bassin="atlantic")
```

Get the Pandas frame

[58]:
```
df = atlantic.to_pandas()
```

[59]:
```
df.bassin.unique()
```

[59]:
```
array(['AL'], dtype=object)
```

Huricanes making landfall at category 5

[60]:
```
df[(df.category == 5) & (df.type == "L")]
```

[60]:
```
      bassin  number  year     name                time type status   lat  \
17942     AL       3  1935  unnamed 1935-09-03 02:00:00    L     HU  24.8
24300     AL      10  1955    janet 1955-09-27 17:00:00    L     HU  17.4
24303     AL      10  1955    janet 1955-09-28 05:00:00    L     HU  18.4
28888     AL       9  1969  camille 1969-08-18 04:00:00    L     HU  30.3
36448     AL       8  1988  gilbert 1988-09-14 15:00:00    L     HU  20.7
37967     AL       4  1992   andrew 1992-08-23 21:00:00    L     HU  25.4
37971     AL       4  1992   andrew 1992-08-24 08:40:00    L     HU  25.5
37972     AL       4  1992   andrew 1992-08-24 09:05:00    L     HU  25.5
45134     AL       4  2007     dean 2007-08-21 08:30:00    L     HU  18.7
45182     AL       6  2007    felix 2007-09-04 12:00:00    L     HU  14.3
```

(continues on next page)

(continued from previous page)

```
49925     AL      11  2017     irma 2017-09-06 05:45:00    L     HU  17.7
49927     AL      11  2017     irma 2017-09-06 11:15:00    L     HU  18.1
49929     AL      11  2017     irma 2017-09-06 16:30:00    L     HU  18.5
49941     AL      11  2017     irma 2017-09-09 03:00:00    L     HU  22.3
50139     AL      15  2017    maria 2017-09-19 01:15:00    L     HU  15.4
50835     AL      14  2018  michael 2018-10-10 17:30:00    L     HU  30.0

        lon  knots  category  pressure
17942 -80.8  160.0         5     892.0
24300 -83.9  140.0         5       NaN
24303 -87.8  150.0         5     914.0
28888 -89.4  150.0         5     900.0
36448 -87.0  140.0         5     900.0
37967 -76.6  140.0         5     923.0
37971 -80.2  145.0         5     926.0
37972 -80.3  145.0         5     922.0
45134 -87.7  150.0         5     905.0
45182 -83.2  140.0         5     934.0
49925 -61.8  155.0         5     914.0
49927 -63.1  155.0         5     914.0
49929 -64.4  155.0         5     915.0
49941 -77.9  145.0         5     924.0
50139 -61.3  145.0         5     922.0
50835 -85.5  140.0         5     919.0
```

Get the track for Katrina

[61]:
```
katrina = df[(df.name == "katrina") & (df.year == 2005)]
```

[62]:
```
katrina
```

[62]:
```
      bassin  number  year     name                time type status   lat  \
44065     AL      12  2005  katrina 2005-08-23 18:00:00          TD  23.1
44066     AL      12  2005  katrina 2005-08-24 00:00:00          TD  23.4
44067     AL      12  2005  katrina 2005-08-24 06:00:00          TD  23.8
44068     AL      12  2005  katrina 2005-08-24 12:00:00          TS  24.5
44069     AL      12  2005  katrina 2005-08-24 18:00:00          TS  25.4
44070     AL      12  2005  katrina 2005-08-25 00:00:00          TS  26.0
44071     AL      12  2005  katrina 2005-08-25 06:00:00          TS  26.1
44072     AL      12  2005  katrina 2005-08-25 12:00:00          TS  26.2
44073     AL      12  2005  katrina 2005-08-25 18:00:00          TS  26.2
44074     AL      12  2005  katrina 2005-08-25 22:30:00    L     HU  26.0
44075     AL      12  2005  katrina 2005-08-26 00:00:00          HU  25.9
44076     AL      12  2005  katrina 2005-08-26 06:00:00          HU  25.4
44077     AL      12  2005  katrina 2005-08-26 12:00:00          HU  25.1
44078     AL      12  2005  katrina 2005-08-26 18:00:00          HU  24.9
44079     AL      12  2005  katrina 2005-08-27 00:00:00          HU  24.6
44080     AL      12  2005  katrina 2005-08-27 06:00:00          HU  24.4
44081     AL      12  2005  katrina 2005-08-27 12:00:00          HU  24.4
44082     AL      12  2005  katrina 2005-08-27 18:00:00          HU  24.5
44083     AL      12  2005  katrina 2005-08-28 00:00:00          HU  24.8
```

(continues on next page)

(continued from previous page)

```
44084     AL      12  2005  katrina 2005-08-28 06:00:00          HU  25.2
44085     AL      12  2005  katrina 2005-08-28 12:00:00          HU  25.7
44086     AL      12  2005  katrina 2005-08-28 18:00:00          HU  26.3
44087     AL      12  2005  katrina 2005-08-29 00:00:00          HU  27.2
44088     AL      12  2005  katrina 2005-08-29 06:00:00          HU  28.2
44089     AL      12  2005  katrina 2005-08-29 11:10:00    L     HU  29.3
44090     AL      12  2005  katrina 2005-08-29 12:00:00          HU  29.5
44091     AL      12  2005  katrina 2005-08-29 14:45:00    L     HU  30.2
44092     AL      12  2005  katrina 2005-08-29 18:00:00          HU  31.1
44093     AL      12  2005  katrina 2005-08-30 00:00:00          TS  32.6
44094     AL      12  2005  katrina 2005-08-30 06:00:00          TS  34.1
44095     AL      12  2005  katrina 2005-08-30 12:00:00          TD  35.6
44096     AL      12  2005  katrina 2005-08-30 18:00:00          TD  37.0
44097     AL      12  2005  katrina 2005-08-31 00:00:00          EX  38.6
44098     AL      12  2005  katrina 2005-08-31 06:00:00          EX  40.1

        lon  knots  category  pressure
44065 -75.1   30.0         1    1008.0
44066 -75.7   30.0         1    1007.0
44067 -76.2   30.0         1    1007.0
44068 -76.5   35.0         1    1006.0
44069 -76.9   40.0         1    1003.0
44070 -77.7   45.0         1    1000.0
44071 -78.4   50.0         1     997.0
44072 -79.0   55.0         1     994.0
44073 -79.6   60.0         1     988.0
44074 -80.1   70.0         1     984.0
44075 -80.3   70.0         1     983.0
44076 -81.3   65.0         1     987.0
44077 -82.0   75.0         1     979.0
44078 -82.6   85.0         2     968.0
44079 -83.3   90.0         2     959.0
44080 -84.0   95.0         2     950.0
44081 -84.7  100.0         3     942.0
44082 -85.3  100.0         3     948.0
44083 -85.9  100.0         3     941.0
44084 -86.7  125.0         4     930.0
44085 -87.7  145.0         5     909.0
44086 -88.6  150.0         5     902.0
44087 -89.2  140.0         5     905.0
44088 -89.6  125.0         4     913.0
44089 -89.6  110.0         3     920.0
44090 -89.6  110.0         3     923.0
44091 -89.6  105.0         3     928.0
44092 -89.6   80.0         1     948.0
44093 -89.1   50.0         1     961.0
44094 -88.6   40.0         1     978.0
44095 -88.0   30.0         1     985.0
44096 -87.0   30.0         1     990.0
44097 -85.3   30.0         1     994.0
44098 -82.9   25.0         1     996.0
```

Plot the track

[63]:
```
cml.plot_map(katrina, margins=0.25)
```

Retrieve matching fields from ERA5

[64]:
```
hurricane = katrina[katrina.status == "HU"]
hurricane
```

[64]:
```
      bassin  number  year     name                 time type status   lat  \
44074     AL      12  2005  katrina 2005-08-25 22:30:00    L     HU  26.0
44075     AL      12  2005  katrina 2005-08-26 00:00:00          HU  25.9
44076     AL      12  2005  katrina 2005-08-26 06:00:00          HU  25.4
44077     AL      12  2005  katrina 2005-08-26 12:00:00          HU  25.1
44078     AL      12  2005  katrina 2005-08-26 18:00:00          HU  24.9
44079     AL      12  2005  katrina 2005-08-27 00:00:00          HU  24.6
44080     AL      12  2005  katrina 2005-08-27 06:00:00          HU  24.4
44081     AL      12  2005  katrina 2005-08-27 12:00:00          HU  24.4
44082     AL      12  2005  katrina 2005-08-27 18:00:00          HU  24.5
44083     AL      12  2005  katrina 2005-08-28 00:00:00          HU  24.8
44084     AL      12  2005  katrina 2005-08-28 06:00:00          HU  25.2
44085     AL      12  2005  katrina 2005-08-28 12:00:00          HU  25.7
44086     AL      12  2005  katrina 2005-08-28 18:00:00          HU  26.3
44087     AL      12  2005  katrina 2005-08-29 00:00:00          HU  27.2
44088     AL      12  2005  katrina 2005-08-29 06:00:00          HU  28.2
44089     AL      12  2005  katrina 2005-08-29 11:10:00    L     HU  29.3
44090     AL      12  2005  katrina 2005-08-29 12:00:00          HU  29.5
44091     AL      12  2005  katrina 2005-08-29 14:45:00    L     HU  30.2
44092     AL      12  2005  katrina 2005-08-29 18:00:00          HU  31.1

        lon  knots  category  pressure
44074 -80.1   70.0         1     984.0
44075 -80.3   70.0         1     983.0
44076 -81.3   65.0         1     987.0
44077 -82.0   75.0         1     979.0
44078 -82.6   85.0         2     968.0
44079 -83.3   90.0         2     959.0
44080 -84.0   95.0         2     950.0
44081 -84.7  100.0         3     942.0
44082 -85.3  100.0         3     948.0
44083 -85.9  100.0         3     941.0
44084 -86.7  125.0         4     930.0
44085 -87.7  145.0         5     909.0
44086 -88.6  150.0         5     902.0
44087 -89.2  140.0         5     905.0
44088 -89.6  125.0         4     913.0
44089 -89.6  110.0         3     920.0
44090 -89.6  110.0         3     923.0
44091 -89.6  105.0         3     928.0
44092 -89.6   80.0         1     948.0
```

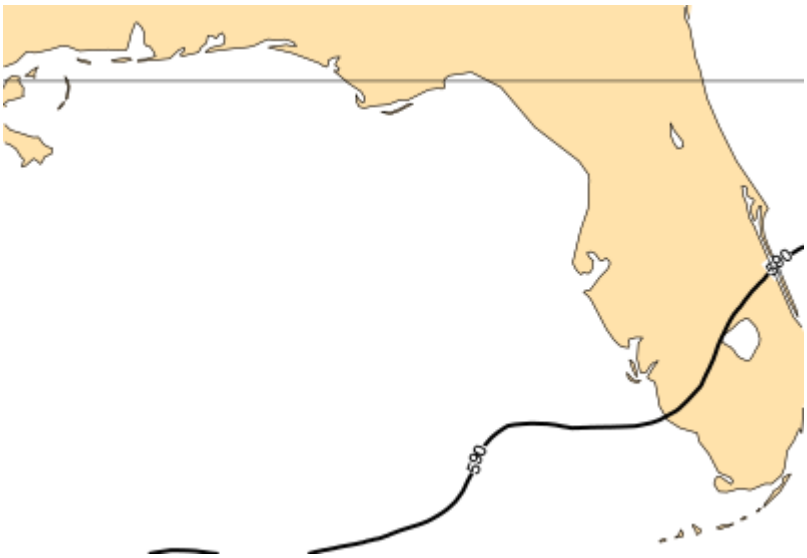
We can use the Pandas frame to specify the `date` and `area` of the request to the CDS.

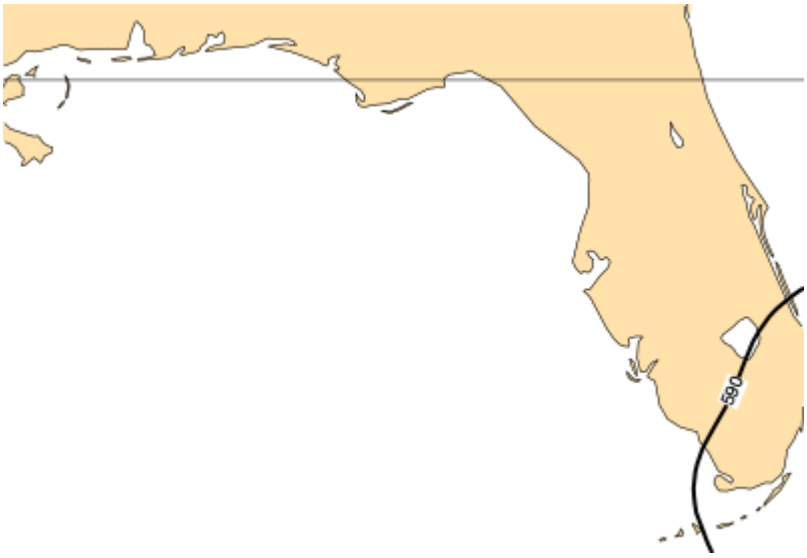
[65]:
```
source = cml.load_source(
    "cds",
    "reanalysis-era5-pressure-levels",
    variable="z",
    level=500,
```

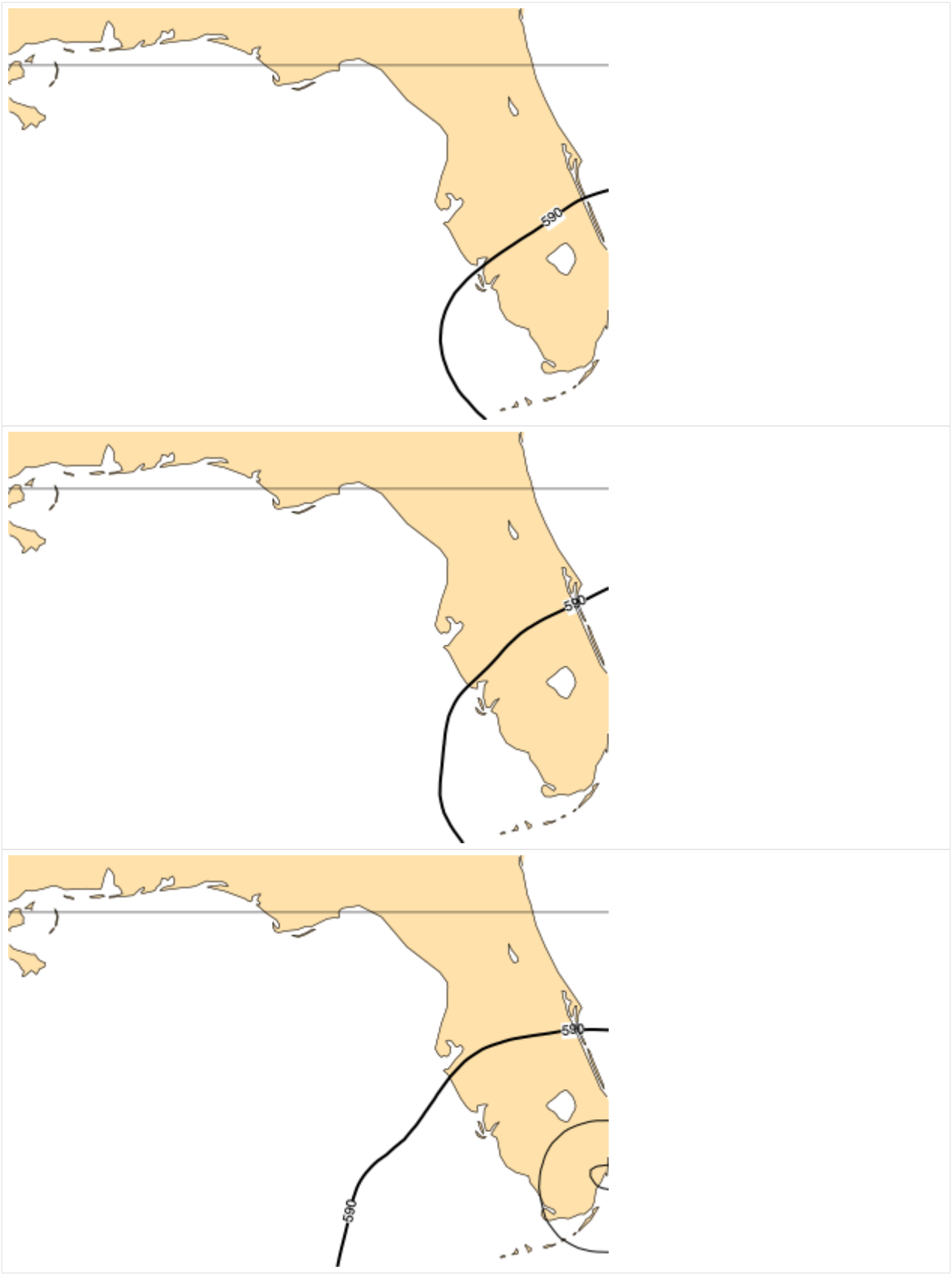
(continues on next page)

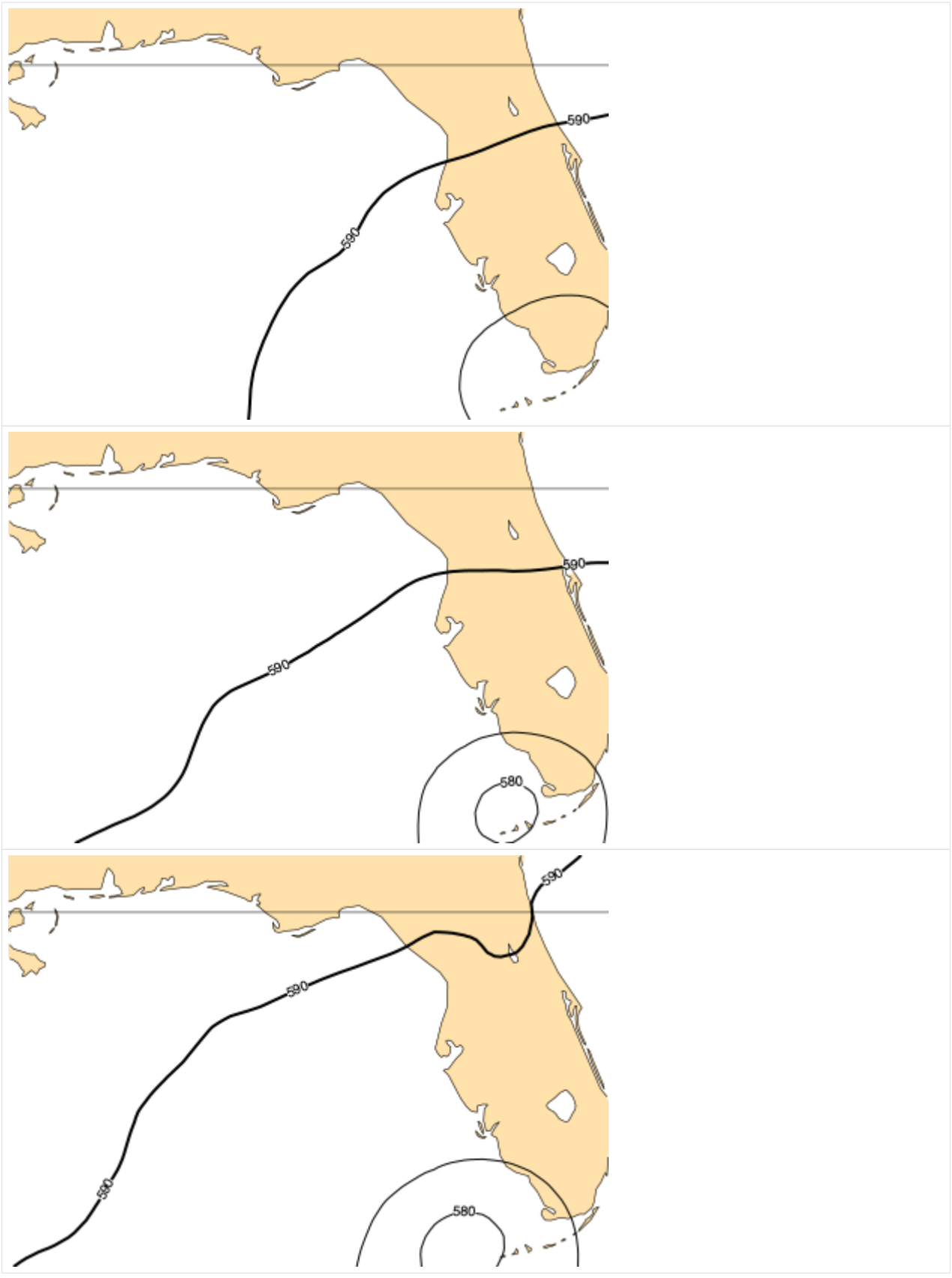
(continued from previous page)

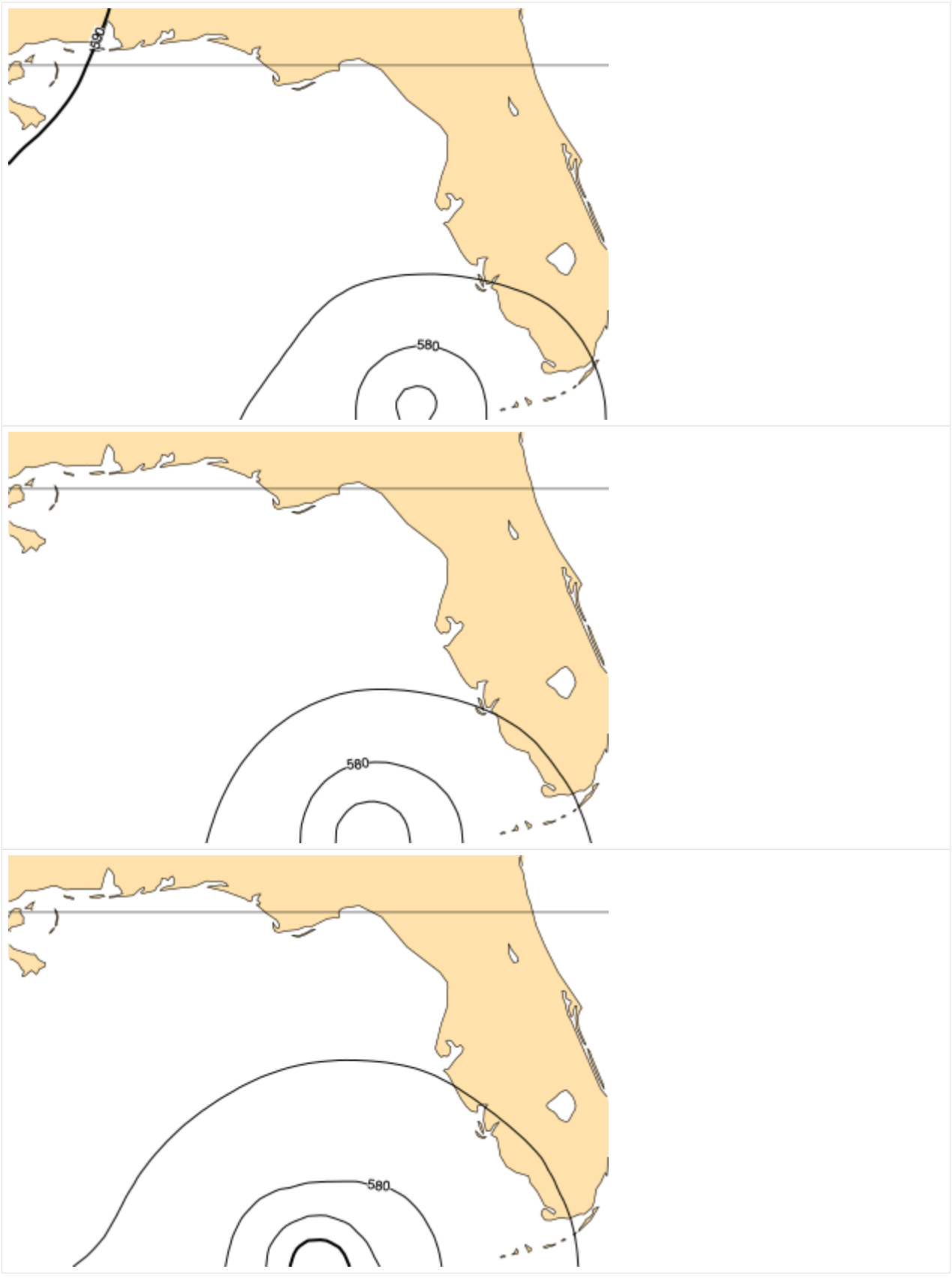
```
    product_type="reanalysis",
    time=[0, 6, 12, 18],
    date=hurricane,
    area=hurricane,
)

for s in source:
    cml.plot_map(s, width=400, grid=True)
```

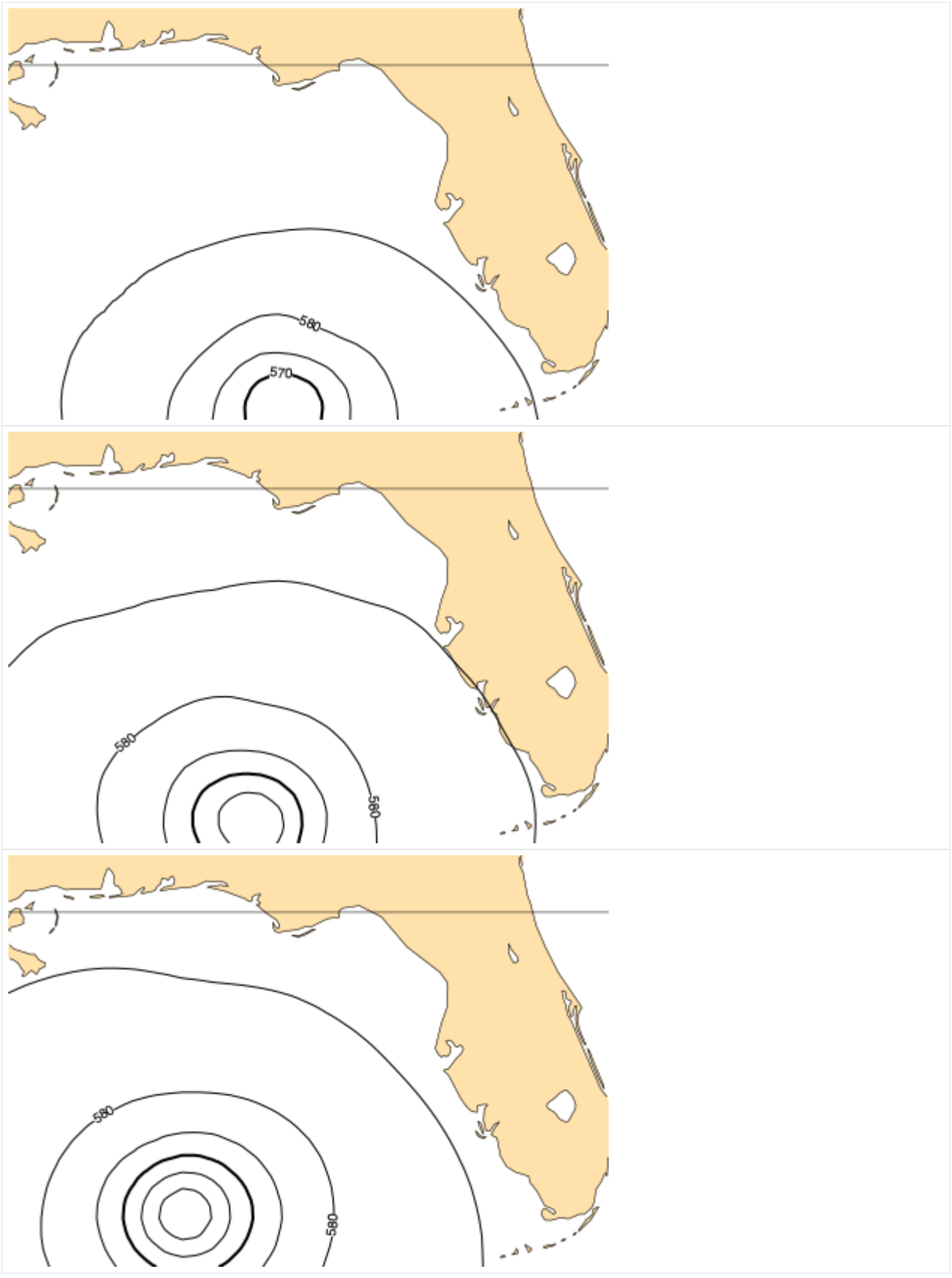


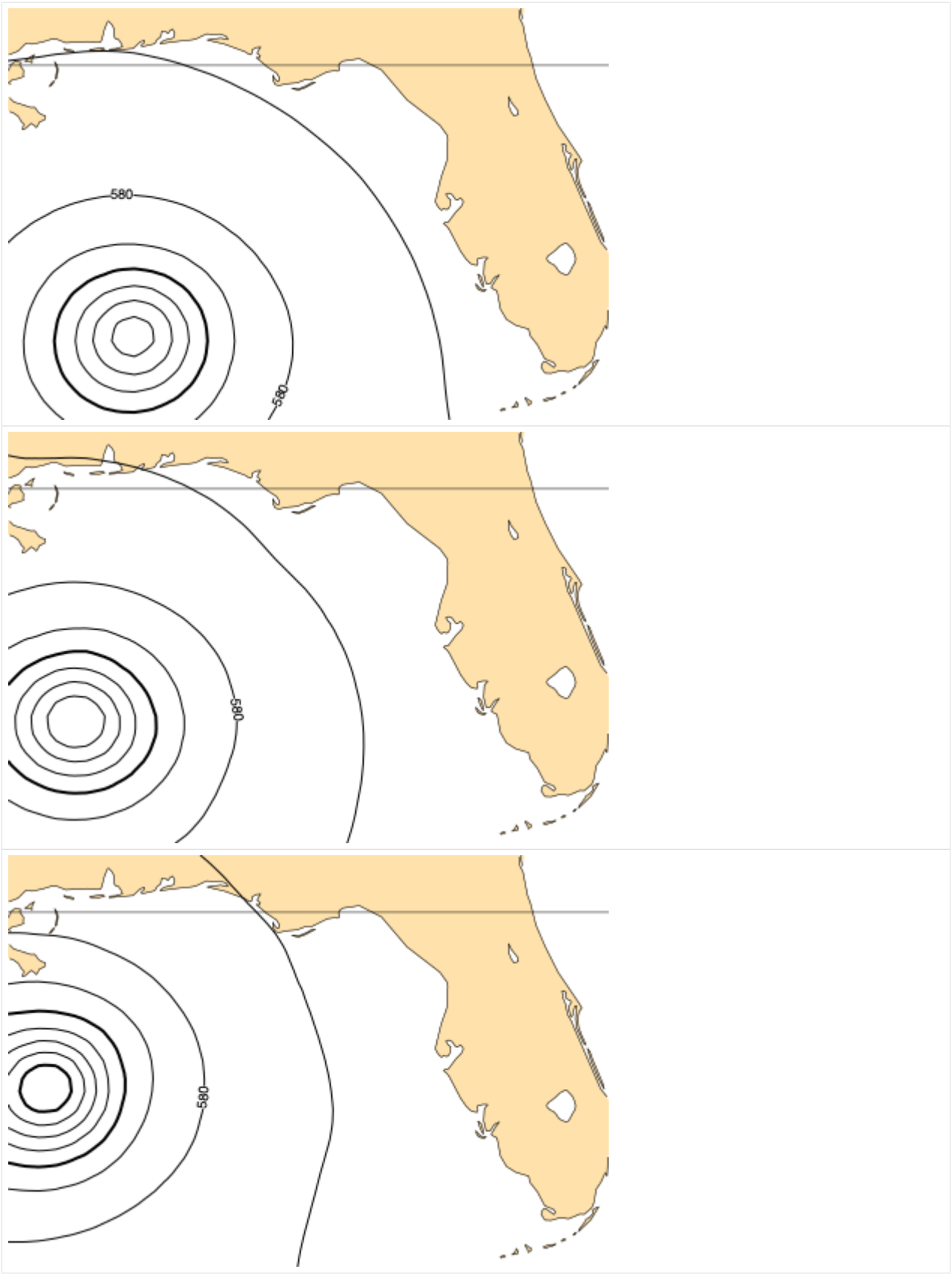


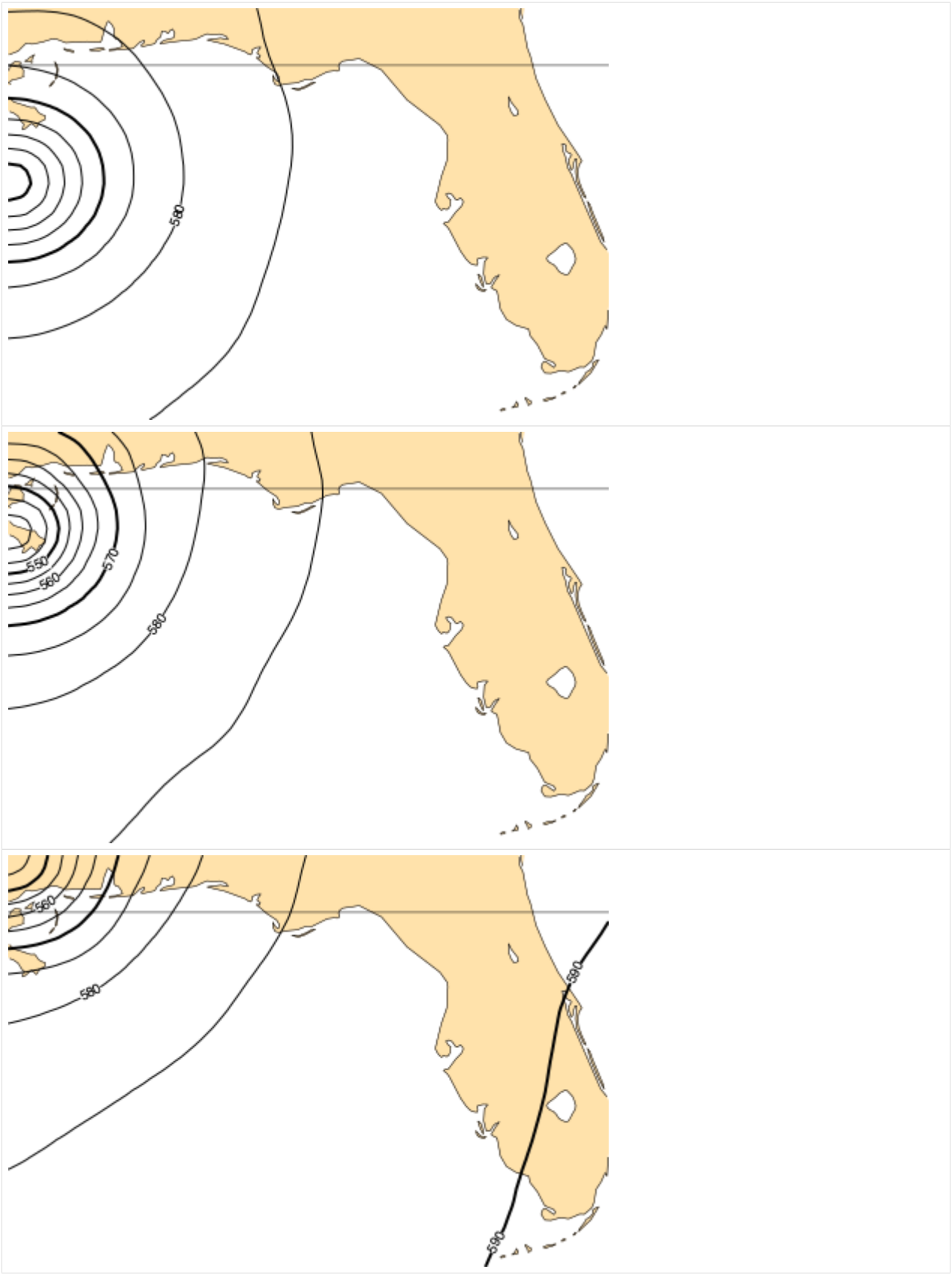

590
590
590
590


590
590
590
590
580
590
590
590
580


590
580
580
580


580
570
580
560
580
580


580
580
580
580
580
580


580
550
560
570
580
560
580
590
590

Pacific bassin

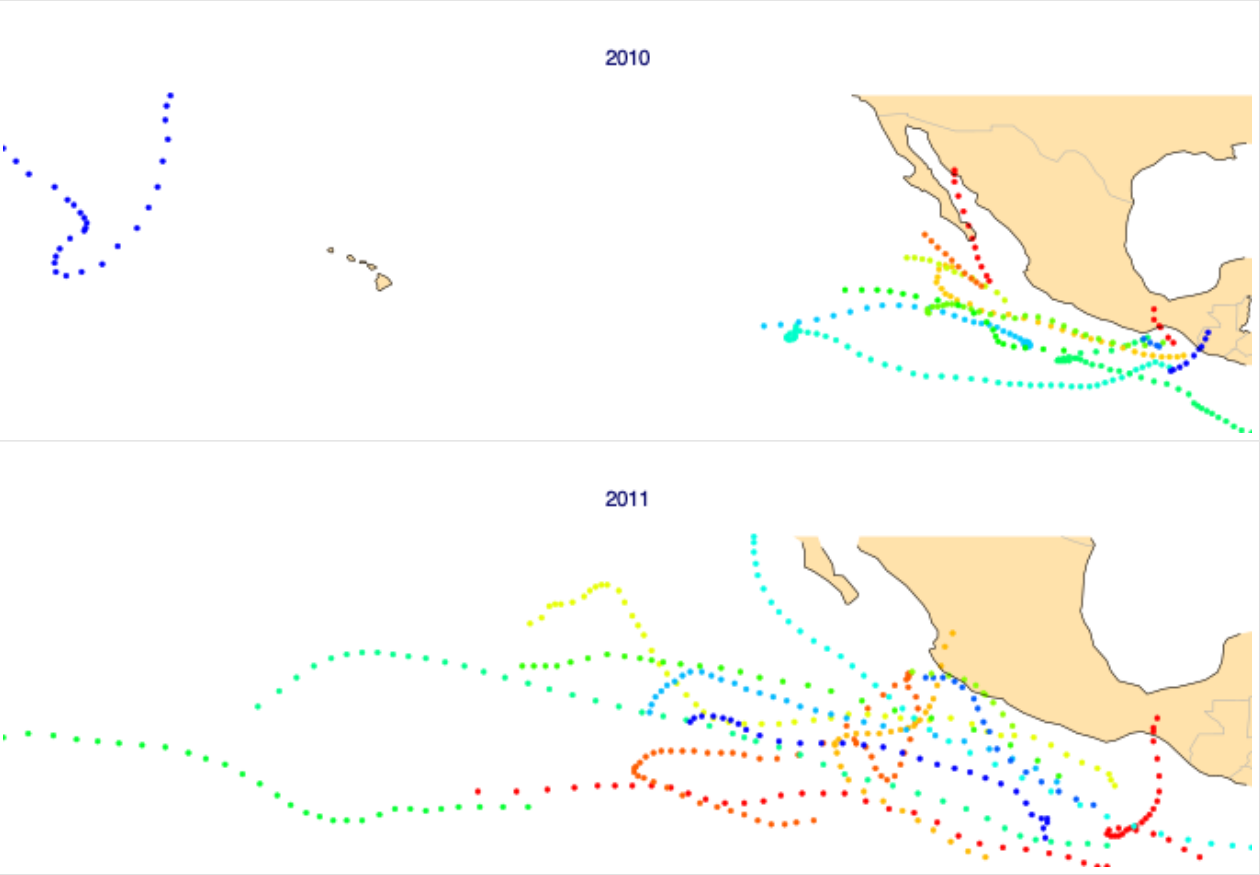
[66]:
```
pacific = cml.load_dataset("hurricane-database", bassin="pacific")
```

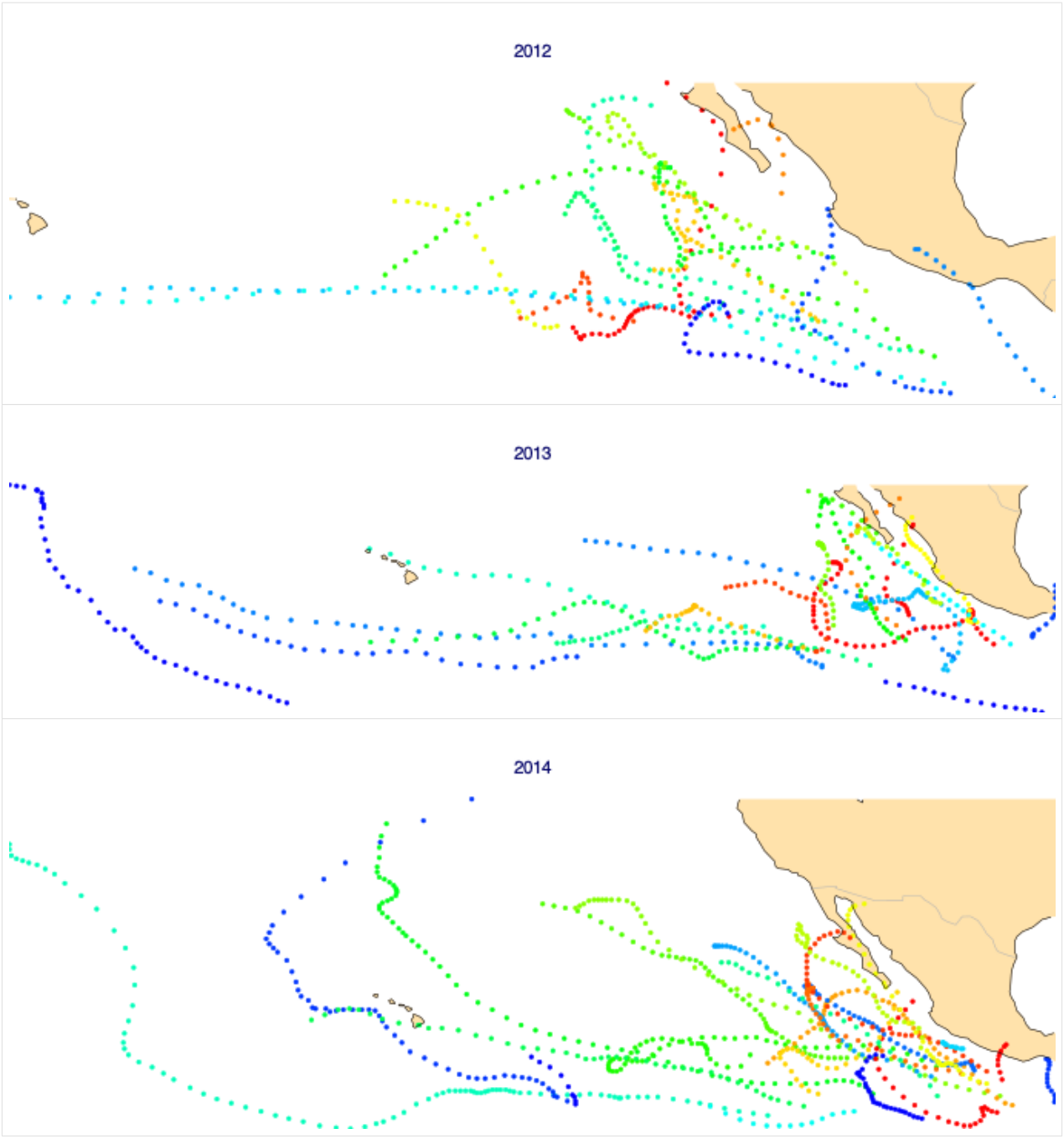
Get the Pandas frame

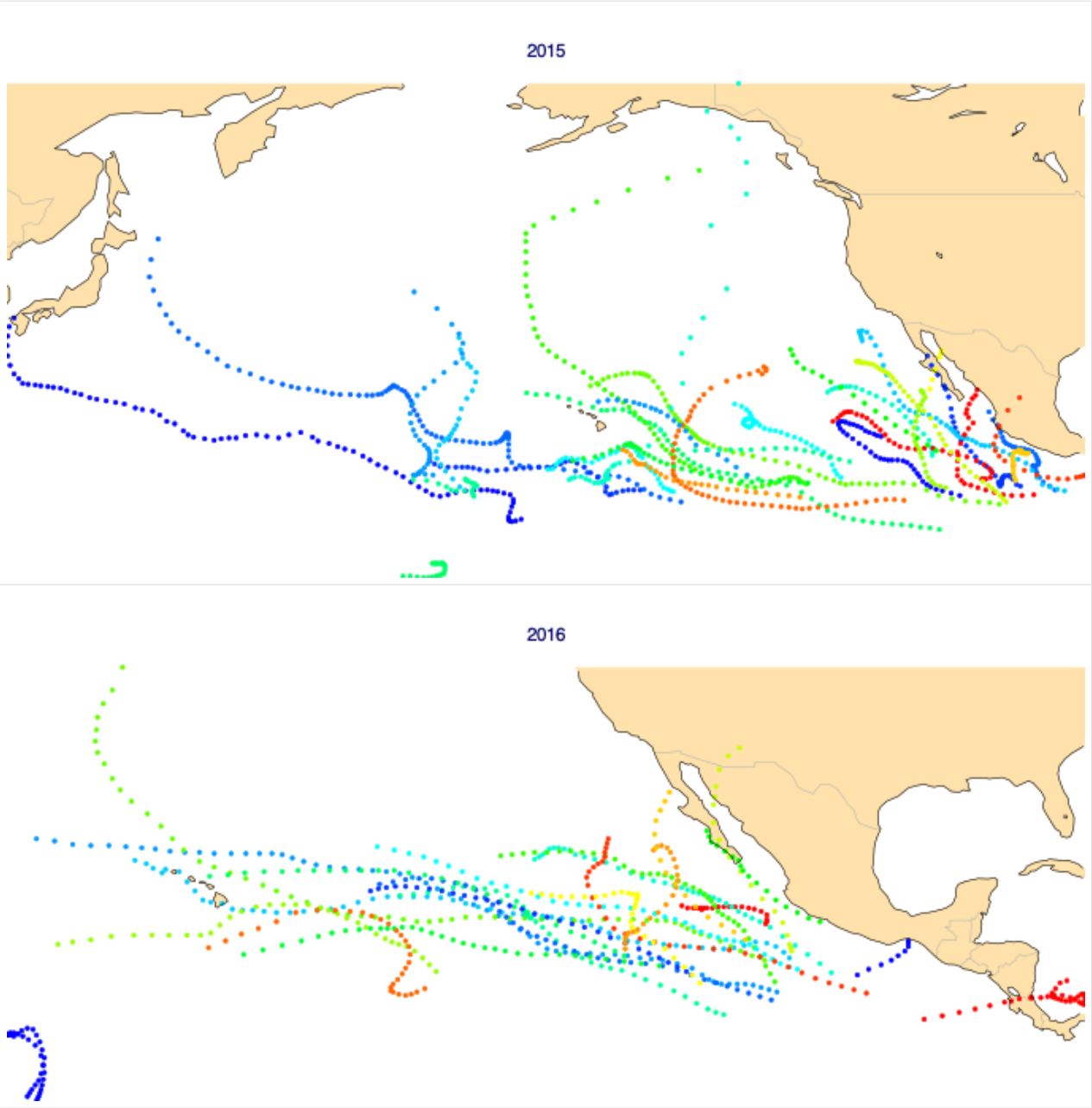
[67]:
```
df = pacific.to_pandas()
```

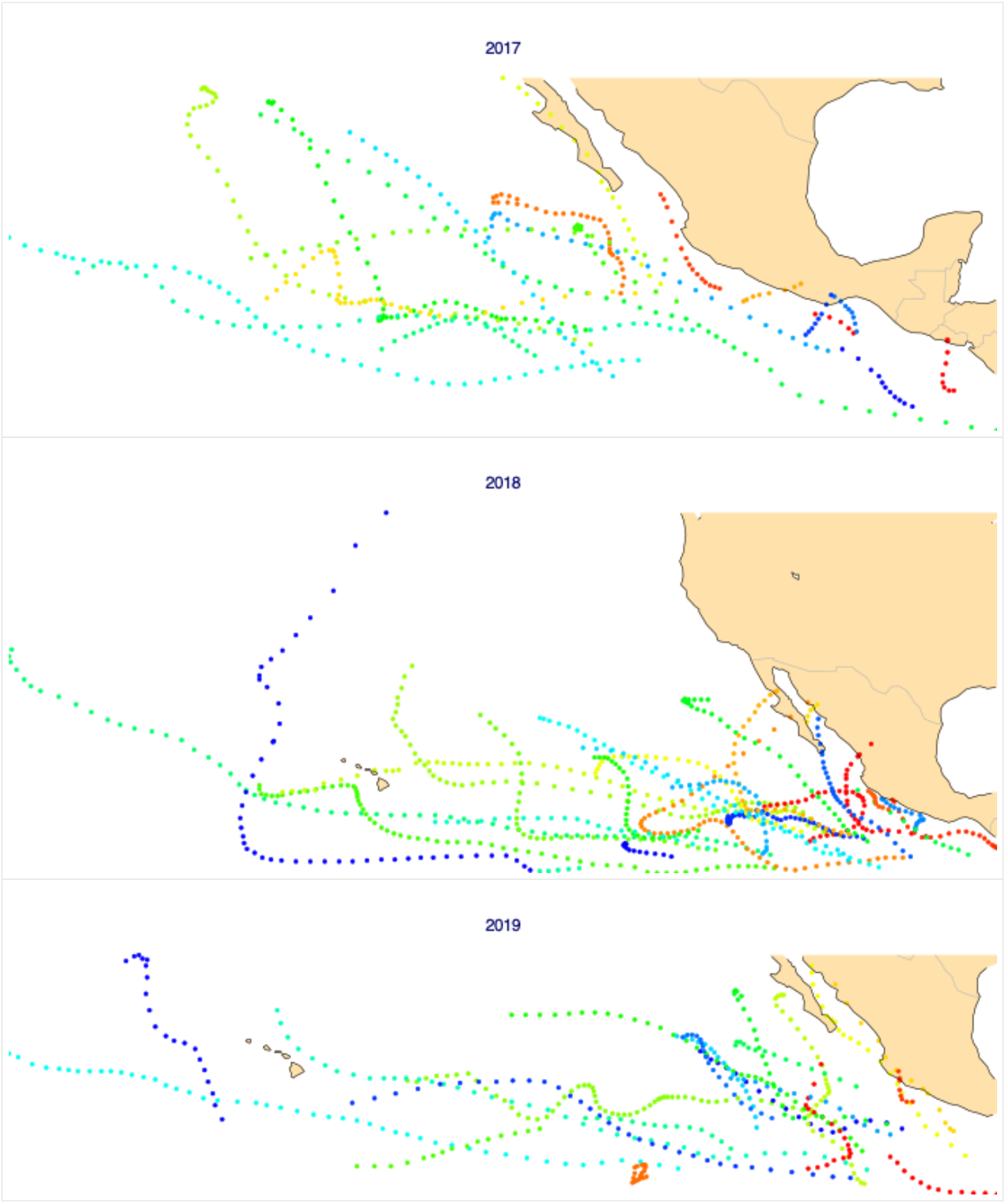
Plot a few years

[68]:
```
for year in range(2010, 2020):
    cml.plot_map(
        df[df.year == year], column="number", title=year, style="rainbow-markers"
    )
```




2012
2013
2014


2015
2016

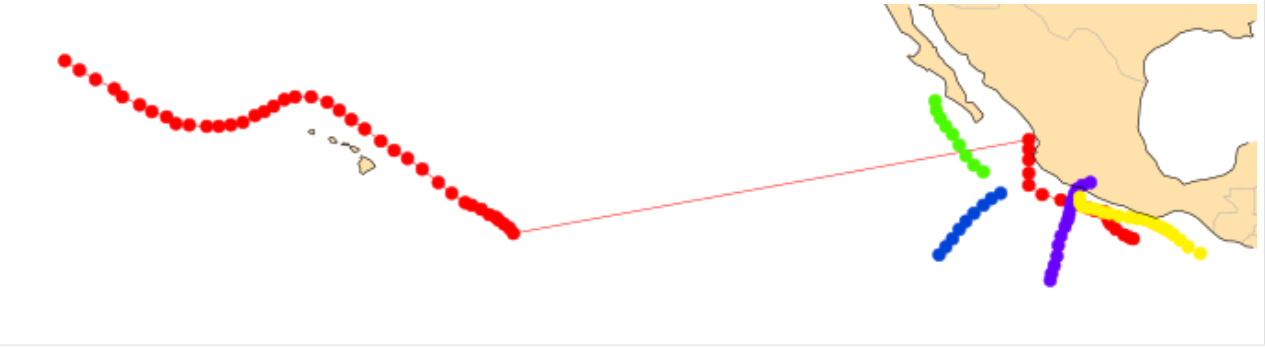

2017
2018
2019

Plot each track independently

[69]:
```
import matplotlib.pyplot as plt
```

[70]:
```
cmap = plt.get_cmap("prism")
```

[71]:
```
p = cml.new_plot(margins="5%")
i = 0
for year in range(1950, 1955):
    cp = df[(df.year == year) & (df.number == 1)]
    if len(cp):
        p.plot_map(cp, style={"+symbol_colour": cmap((i * 20) % cmap.N)})
        i += 1
p.show()
```



[ ]:
```

```

You can run this notebook in , in , in

[13]:
```
!pip install --quiet climetlab
```

[14]:
```
import climetlab as cml
```

[15]:
```
r = {
    "class": "e2",
    "date": "1662-10-01/to/1663-12-31",
    "dataset": "icoads",
    "expver": "1608",
    "groupid": "17",
    "reportype": "16008",
    "format": "ascii",
    "stream": "oper",
    "time": "all",
    "type": "ofb",
}


source = cml.load_source("mars", **r)
```

[16]:
```
pd = source.to_pandas()
```

[17]:
```
pd
```

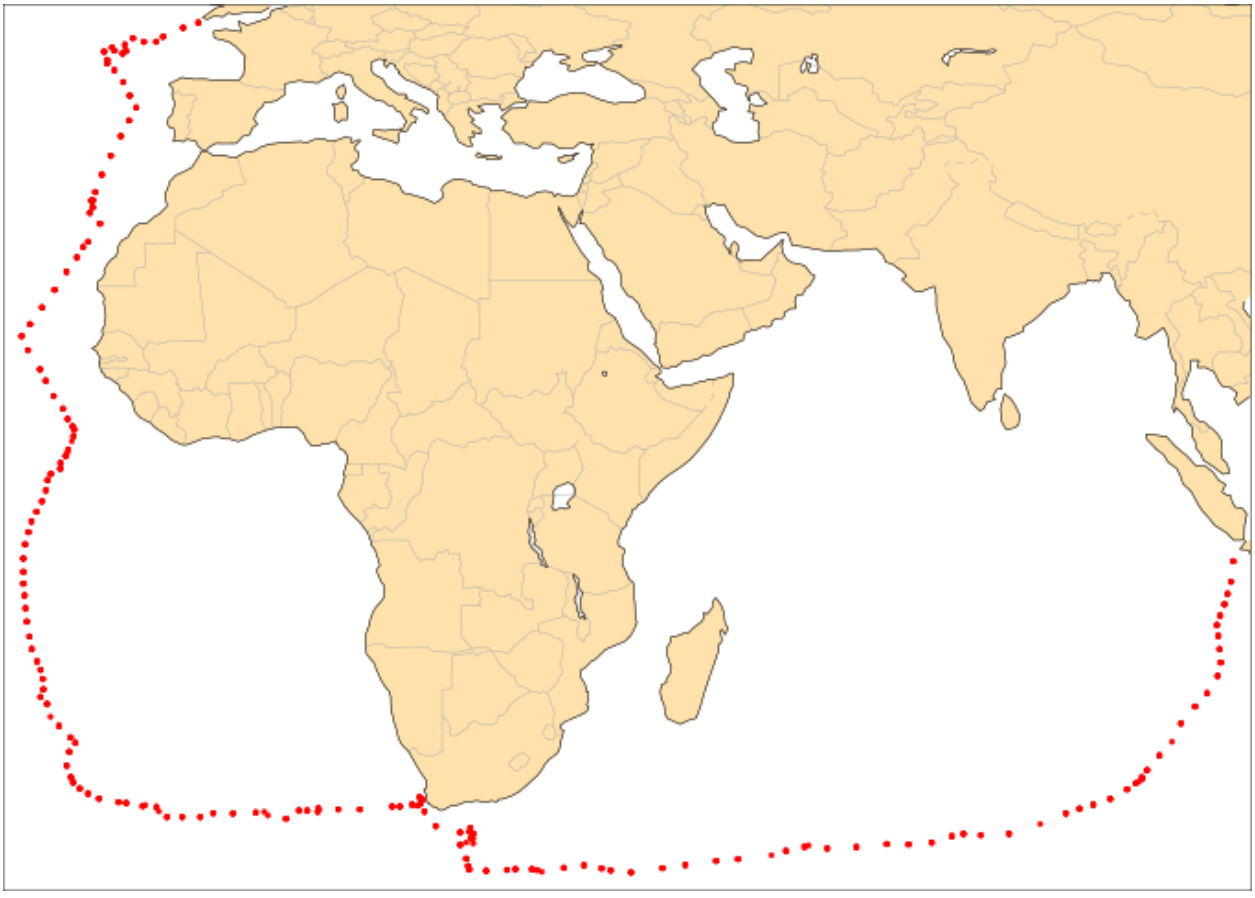
[17]:
```
     expver@desc  type@desc  class@desc  stream@desc  andate@desc  \
0    '1608'             263          22         1025     16621015
1    '1608'             263          22         1025     16621015
2    '1608'             263          22         1025     16621015
3    '1608'             263          22         1025     16621015
4    '1608'             263          22         1025     16621016
..          ...         ...         ...          ...          ...
673  '1608'             263          22         1025     16630425
674  '1608'             263          22         1025     16630426
675  '1608'             263          22         1025     16630426
676  '1608'             263          22         1025     16630426
677  '1608'             263          22         1025     16630426

     antime@desc  seqno@hdr  reportype@hdr  bufrtype@hdr  subtype@hdr  ...  \
0         120000          1          16008             1           11  ...
1         120000          1          16008             1           11  ...
2         120000          1          16008             1           11  ...
3         120000          1          16008             1           11  ...
4         120000          2          16008             1           11  ...
..           ...        ...            ...           ...          ...  ...
673        60000         25          16008             1           11  ...
674        60000         26          16008             1           11  ...
675        60000         26          16008             1           11  ...
676        60000         26          16008             1           11  ...
677        60000         26          16008             1           11  ...

     entryno@body  obsvalue@body  varno@body  vertco_type@body  \
0               1     158.000000         111                 1
1               2       2.600000         112                 1
2               3      -0.973977          41                 1
3               4       2.410678          42                 1
4               1     158.000000         111                 1
..            ...            ...         ...               ...
673             4       2.155498          42                 1
674             1     124.000000         111                 1
675             2       6.700000         112                 1
676             3      -5.554552          41                 1
677             4       3.746593          42                 1

    vertco_reference_1@body  datum_status@body  bias_volatility@body  \
0                       0.0                  1                   NaN
1                       0.0                  1                   NaN
2                       0.0                  1                   NaN
3                       0.0                  1                   NaN
4                       0.0                  1                   NaN
..                      ...                ...                   ...
673                     0.0                  1                   NaN
674                     0.0                  1                   NaN
675                     0.0                  1                   NaN
676                     0.0                  1                   NaN
677                     0.0                  1                   NaN
```

(continues on next page)

(continued from previous page)

```
     an_depar@body  fg_depar@body  ppcode@conv_body
0              NaN            NaN                 0
1              NaN            NaN                 0
2              NaN            NaN                 0
3              NaN            NaN                 0
4              NaN            NaN                 0
..             ...            ...               ...
673            NaN            NaN                 0
674            NaN            NaN                 0
675            NaN            NaN                 0
676            NaN            NaN                 0
677            NaN            NaN                 0

[678 rows x 44 columns]
```

[12]:
```
cml.plot_map(pd, margins=2)
```



You can run this notebook in , in , in

[1]:
```
!pip install --quiet climetlab climetlab-demo-source climetlab-demo-dataset
```

1.4.13 External plugins

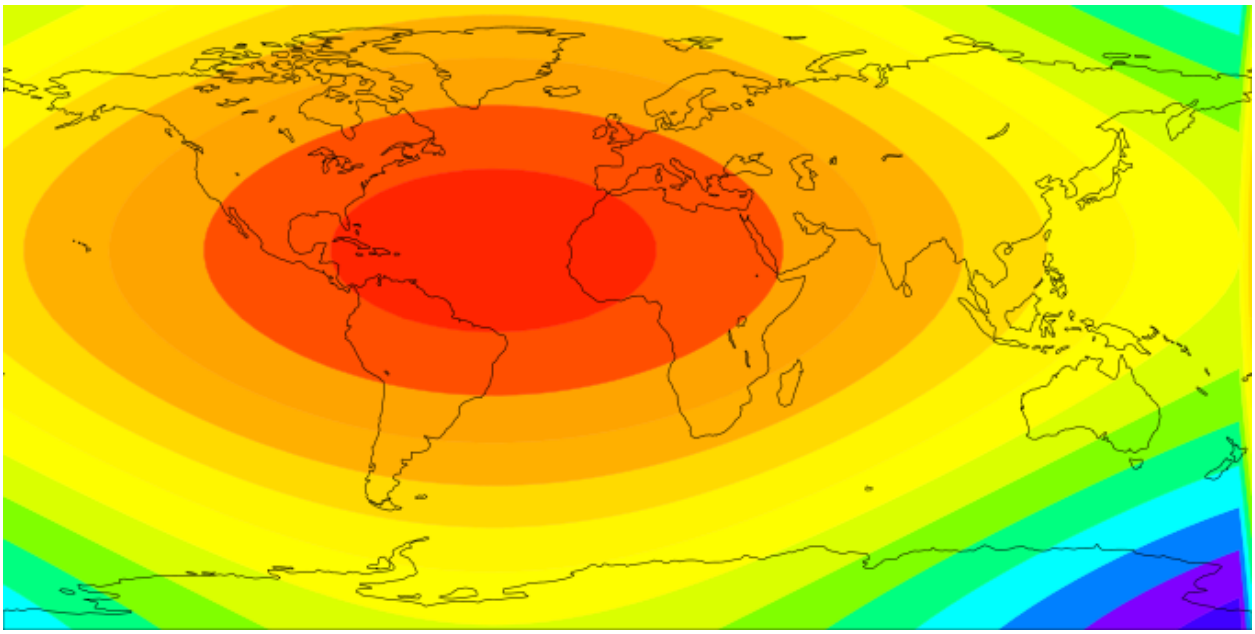
[2]:
```
import climetlab as cml
```

Get the demo dataset:

[3]:
```
ds = cml.load_dataset("demo-dataset")
```

Plot it:

[4]:
```
cml.plot_map(ds)
```



External data source

[5]:
```
!test -f test.db || wget https://github.com/ecmwf/climetlab/raw/main/docs/examples/test.
↪db
```

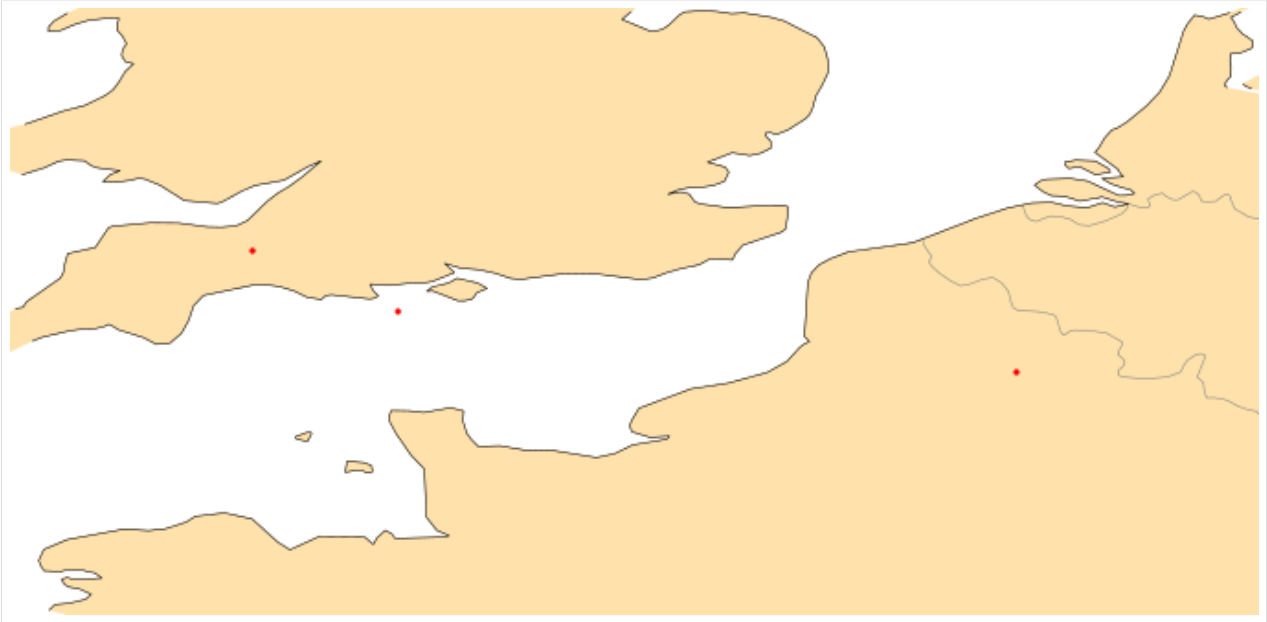
[6]:
```
s = cml.load_source(
    "demo-source", "sqlite:///test.db", "select * from data;", parse_dates=["time"]
)
```

[7]:
```
df = s.to_pandas()
```

[8]:
```
df
```

[8]:
```
    lat  lon       time  value
0  50.0  3.3 2001-01-01    4.9
1  51.0 -3.0 2001-01-02    7.3
2  50.5 -1.8 2001-01-03    5.5
```

[9]:
```
cml.plot_map(df, margins=2)
```



You can run this notebook in , in , in

[1]:
```
!pip install --quiet climetlab
```

[2]:
```
%config Application.log_level="INFO"
```

1.4.14 More plotting examples

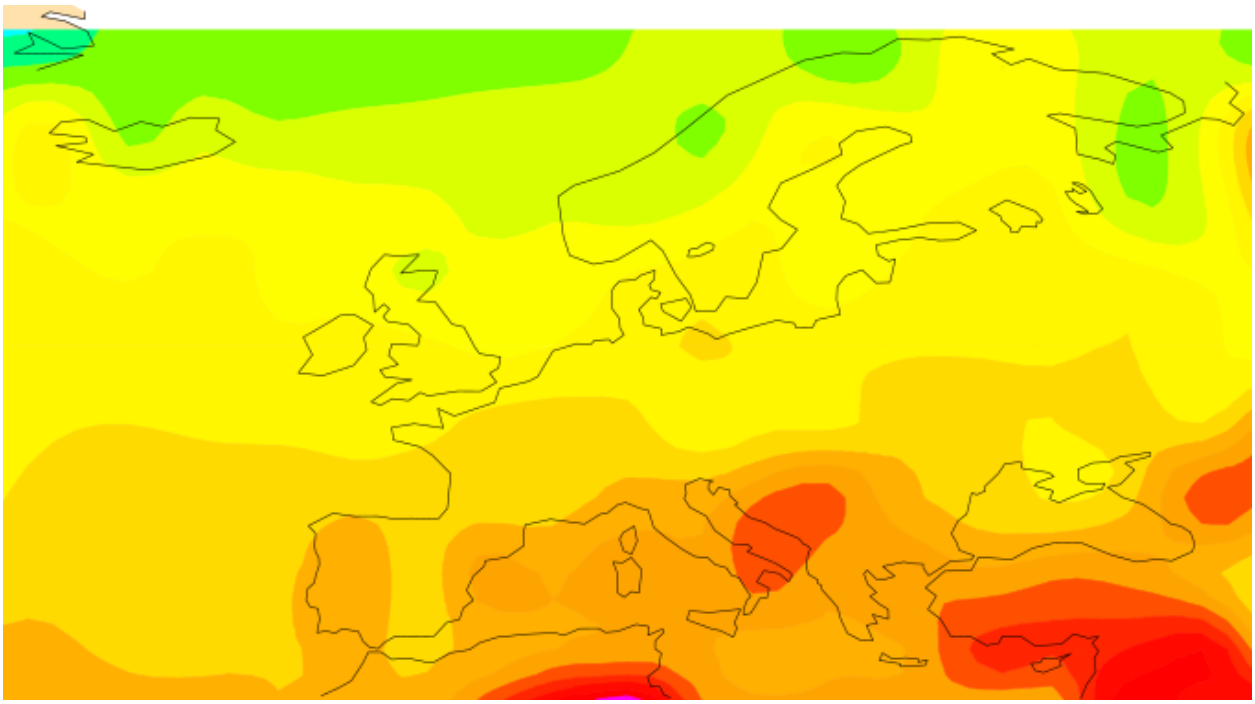
[3]:
```
import climetlab as cml
```

Get some GRIB data

[4]:
```
ds = cml.load_dataset("sample-grib-data")
```

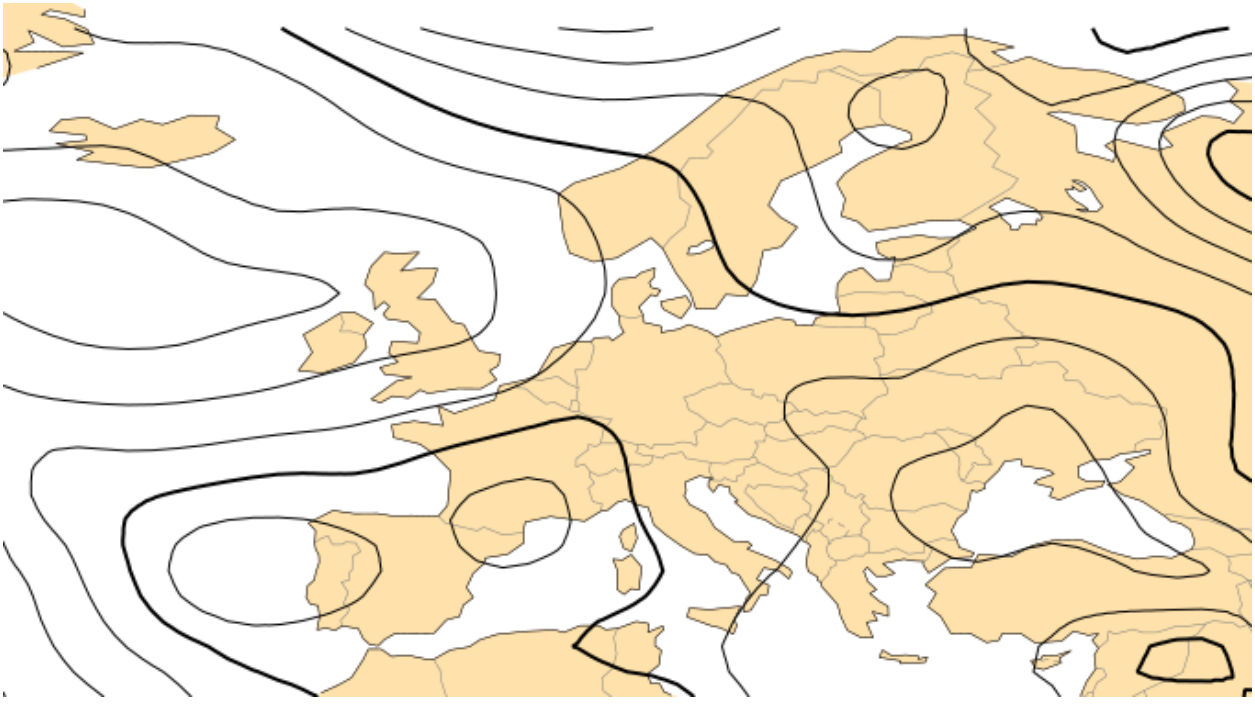
Plot the first field

[5]:
```
cml.plot_map(ds[0])
```



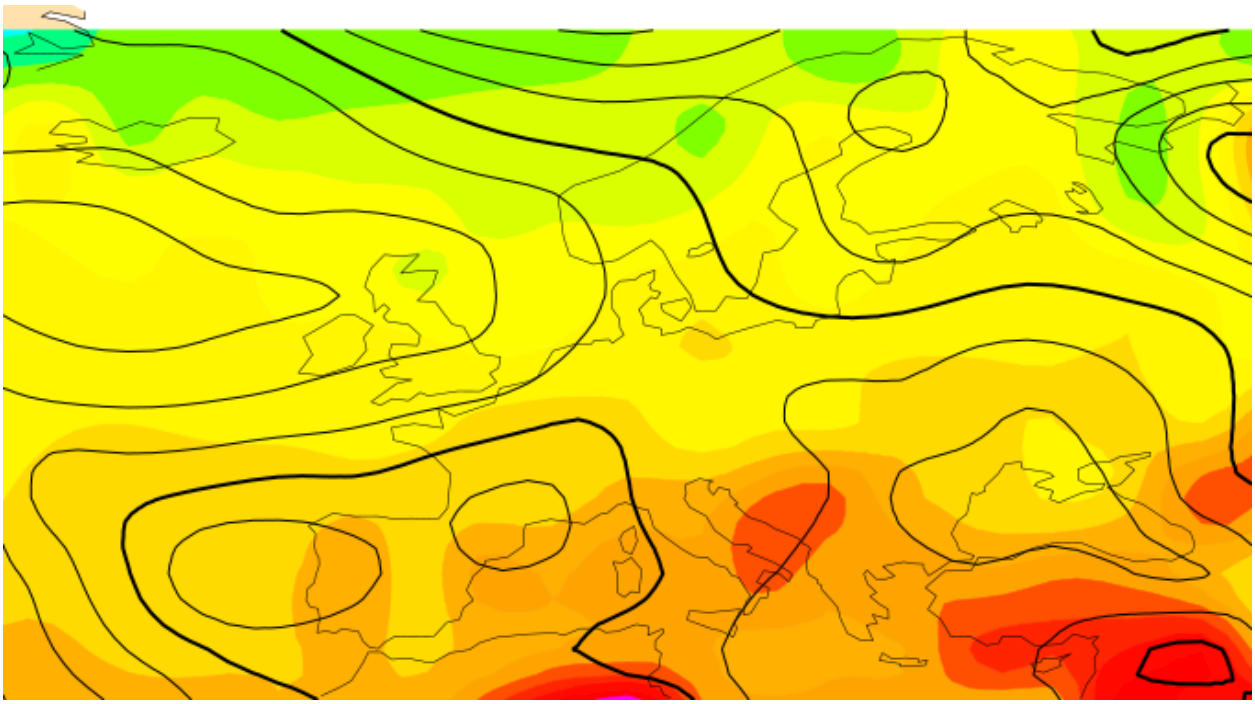
Plot the second field

[6]:
```
cml.plot_map(ds[1])
```



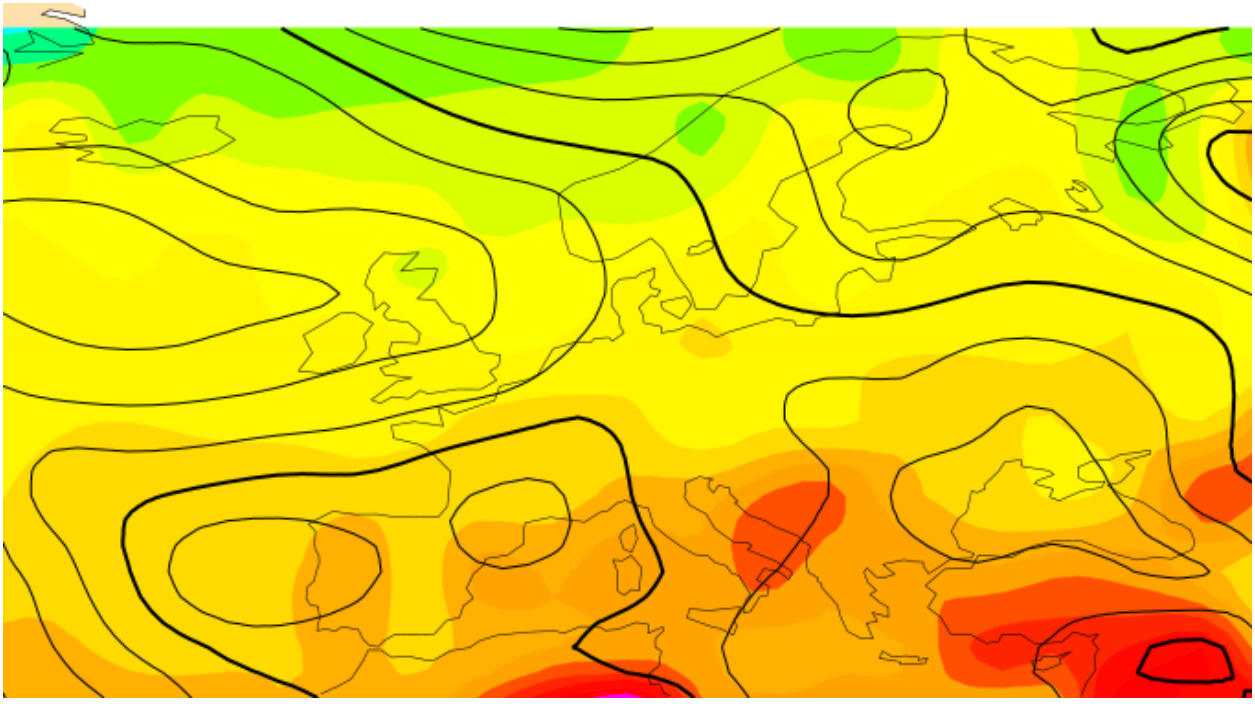
Plot both fields on the same map

[7]:
```
cml.plot_map((ds[0], ds[1]))
```



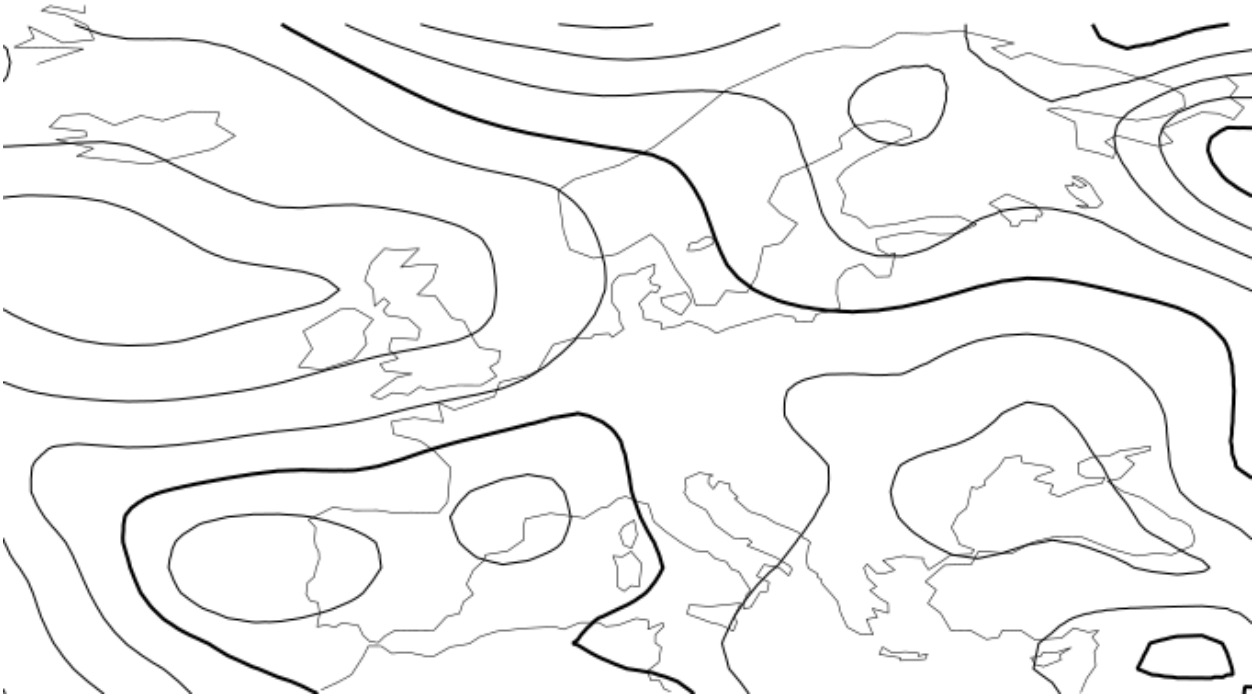
Alternative method:

[8]:
```
p = cml.new_plot()
p.plot_map(ds[0])
p.plot_map(ds[1])
p.show()
```



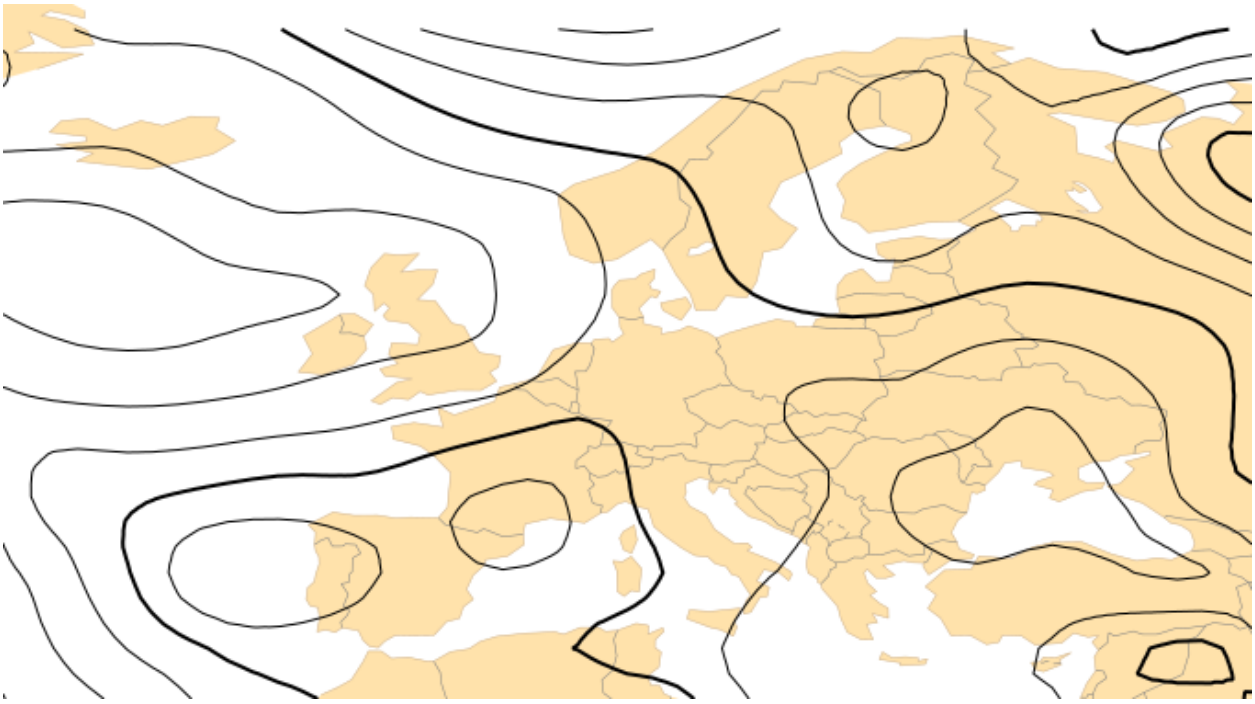
Switch off background map

[9]:
```
cml.plot_map(ds[1], background=False)
```



Switch off both foreground and background

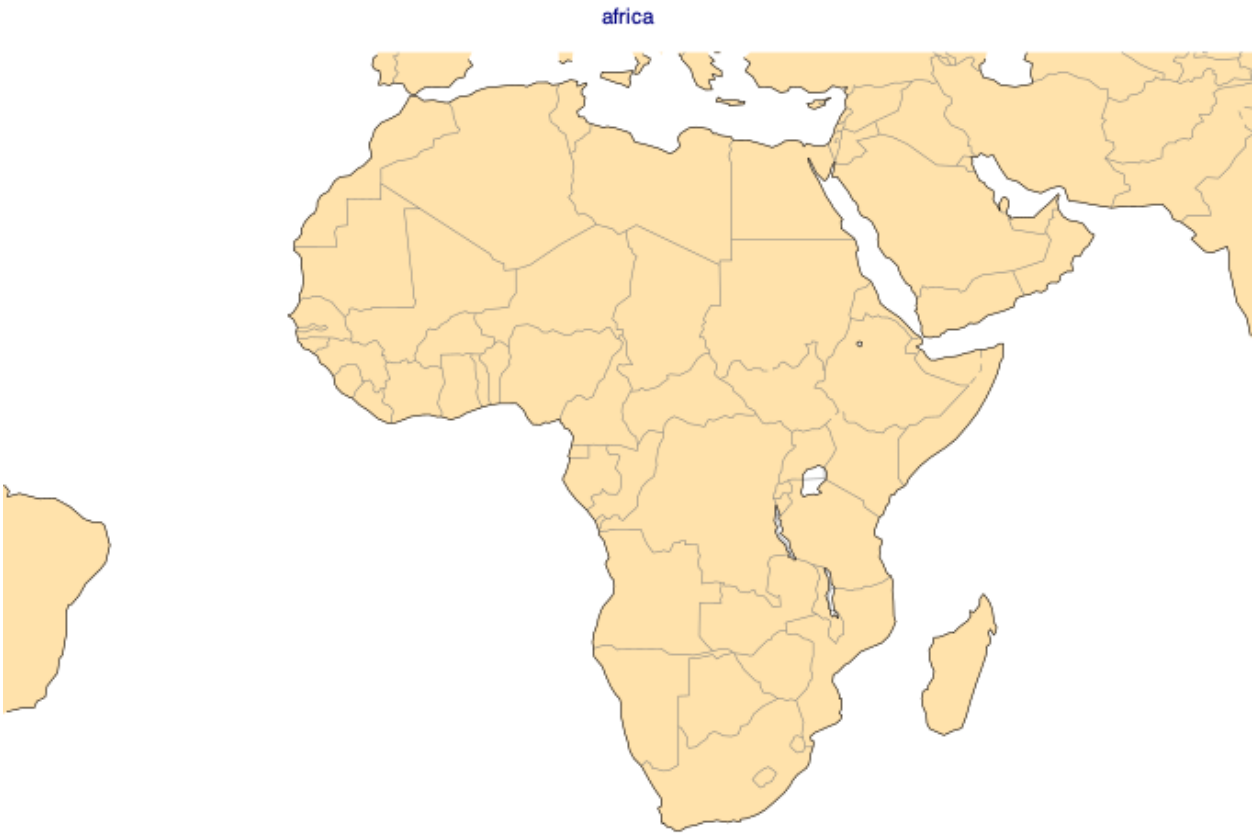
[10]:
```
cml.plot_map(
    ds[1],
    foreground=False,
)
```

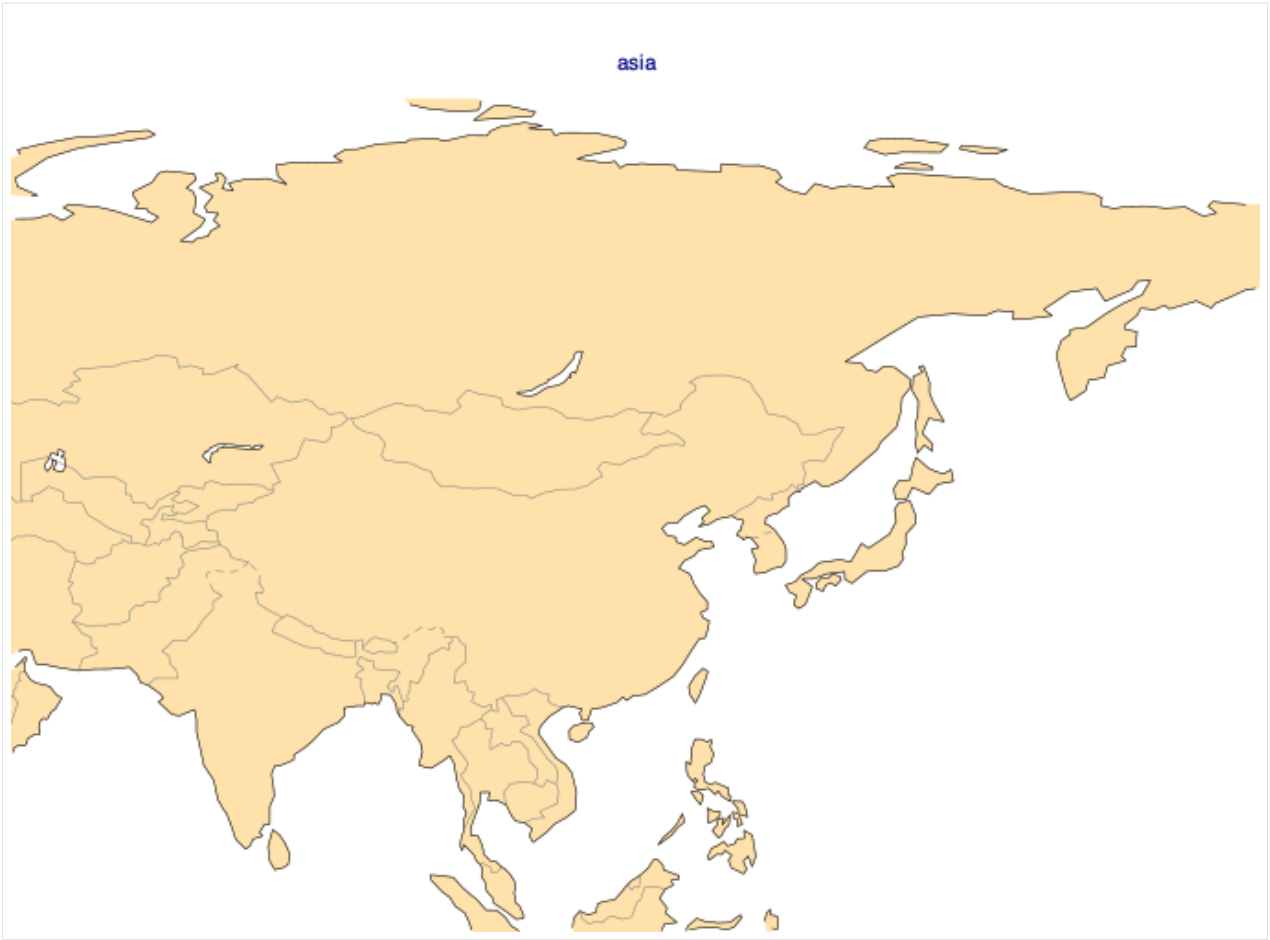


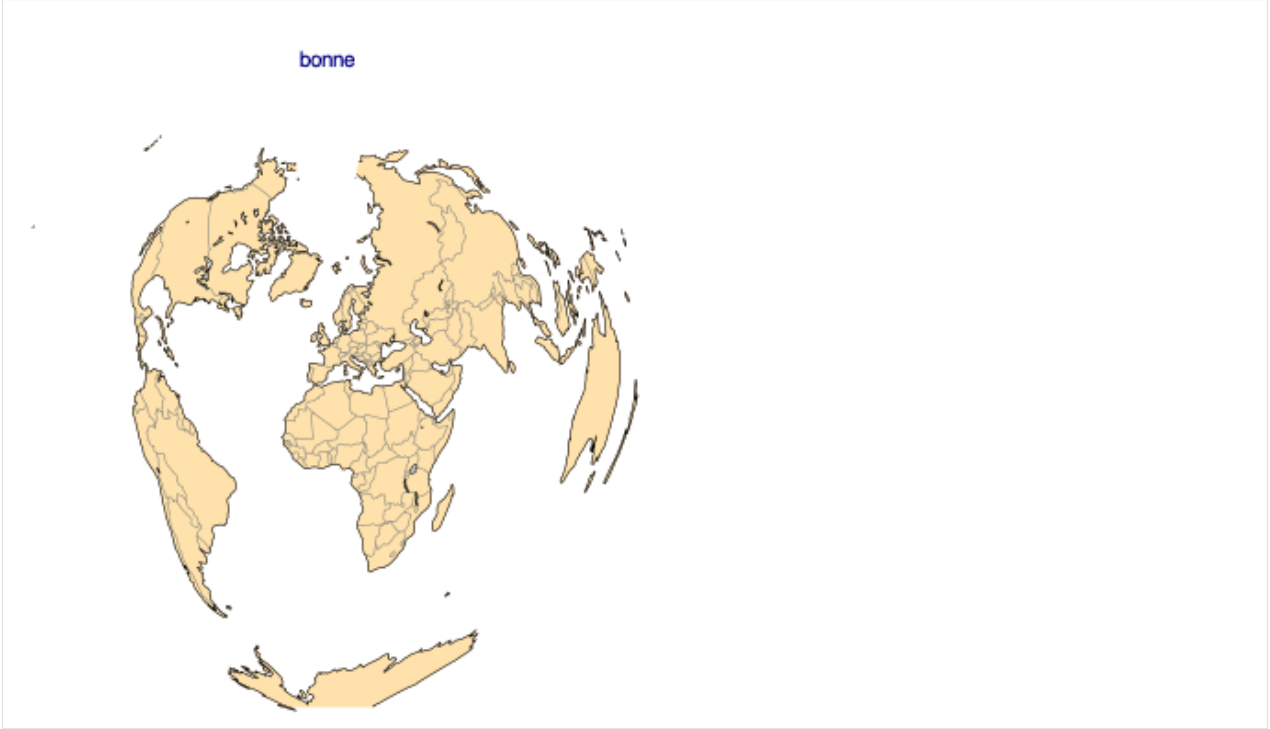
Projections

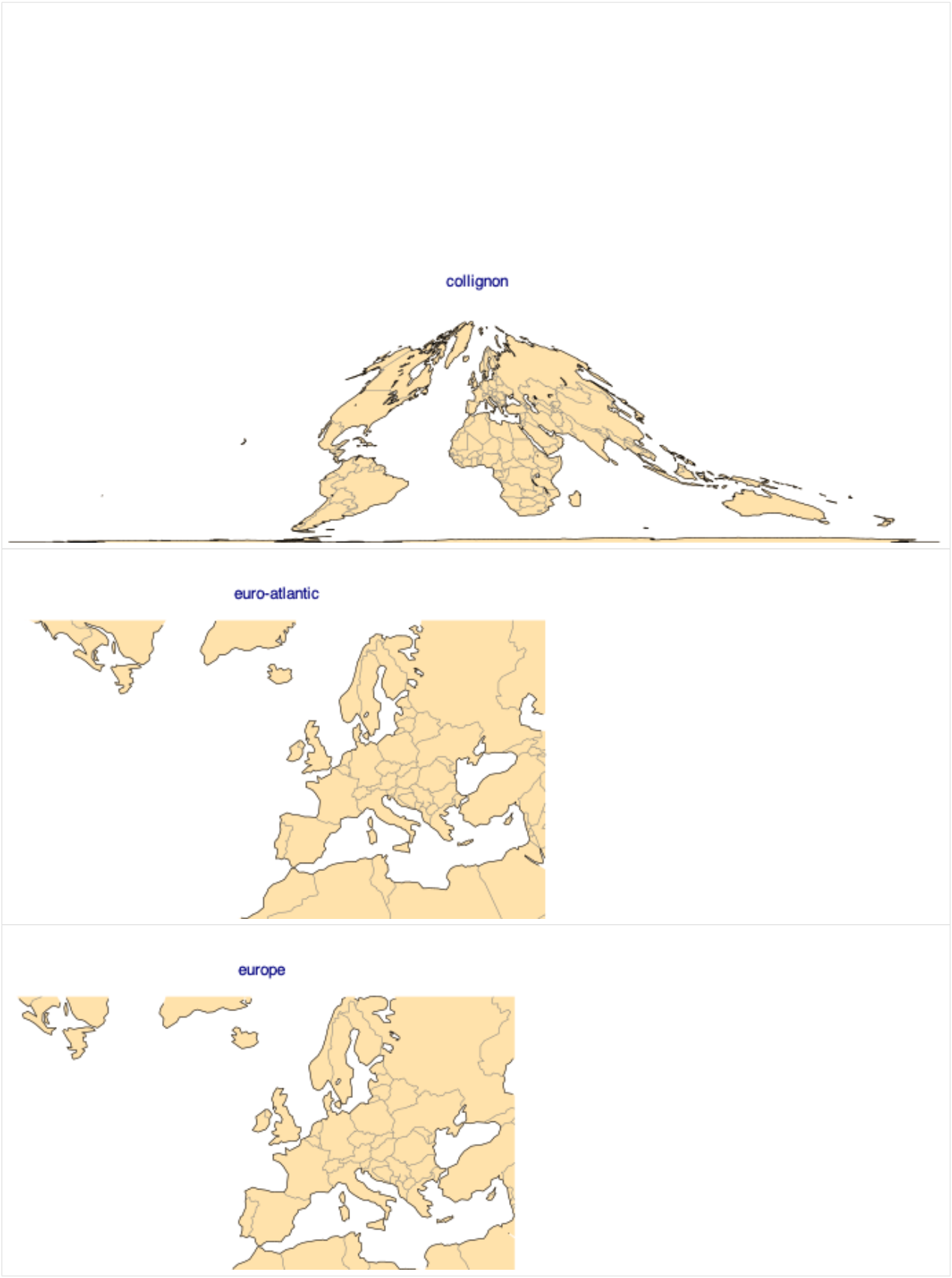
[11]:
```
from climetlab.plotting import projections
```

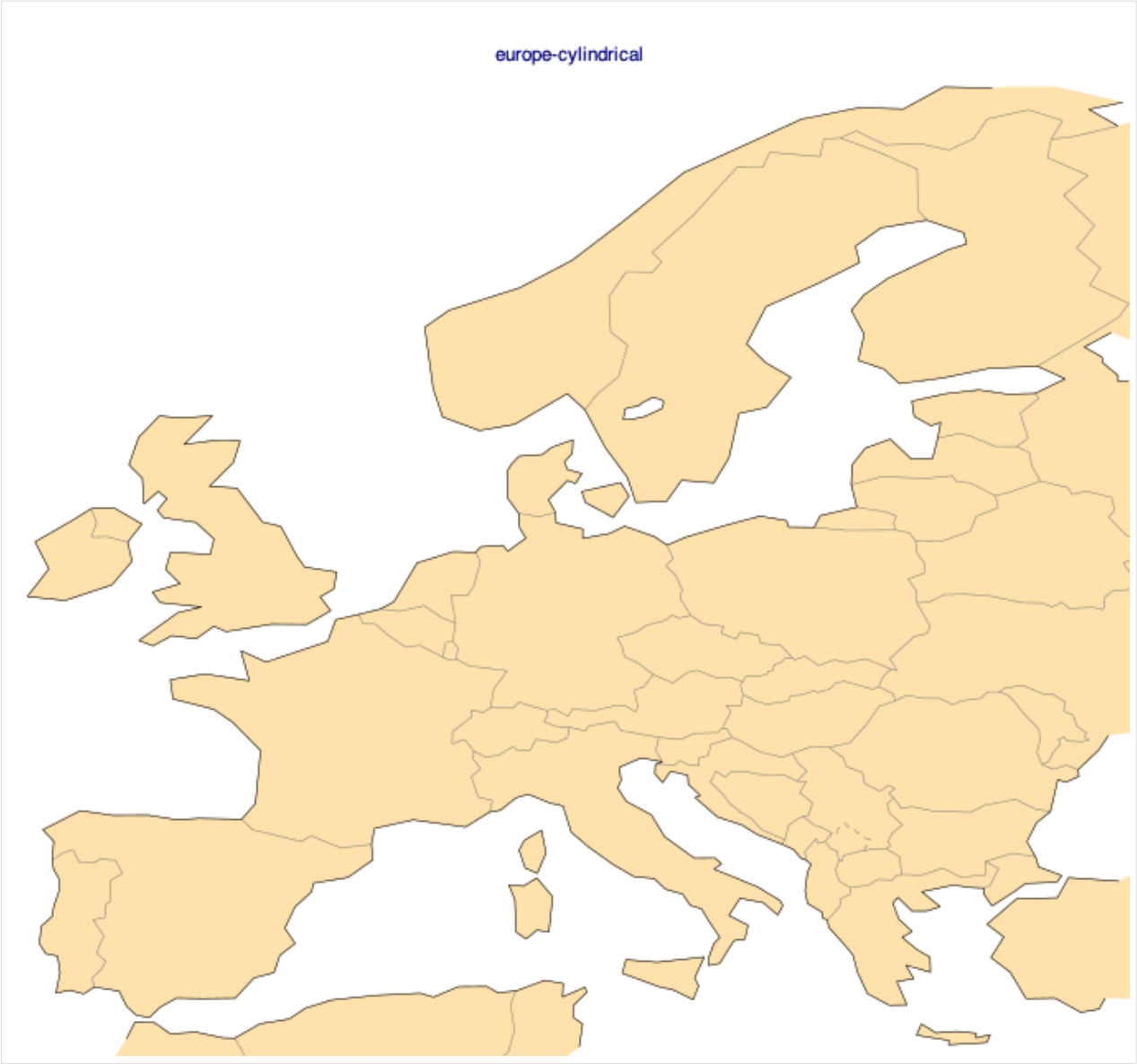
[12]:
```
for projection in projections():
    cml.plot_map(projection=projection, title=projection)
```

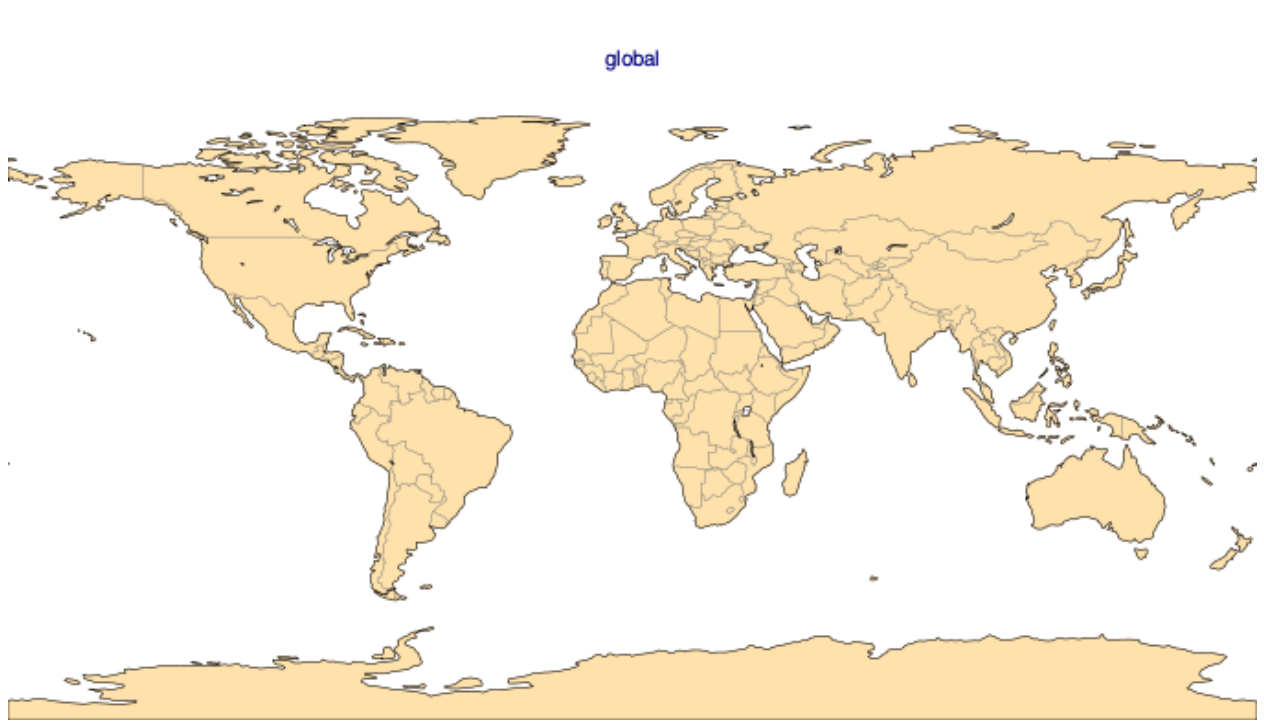



asia

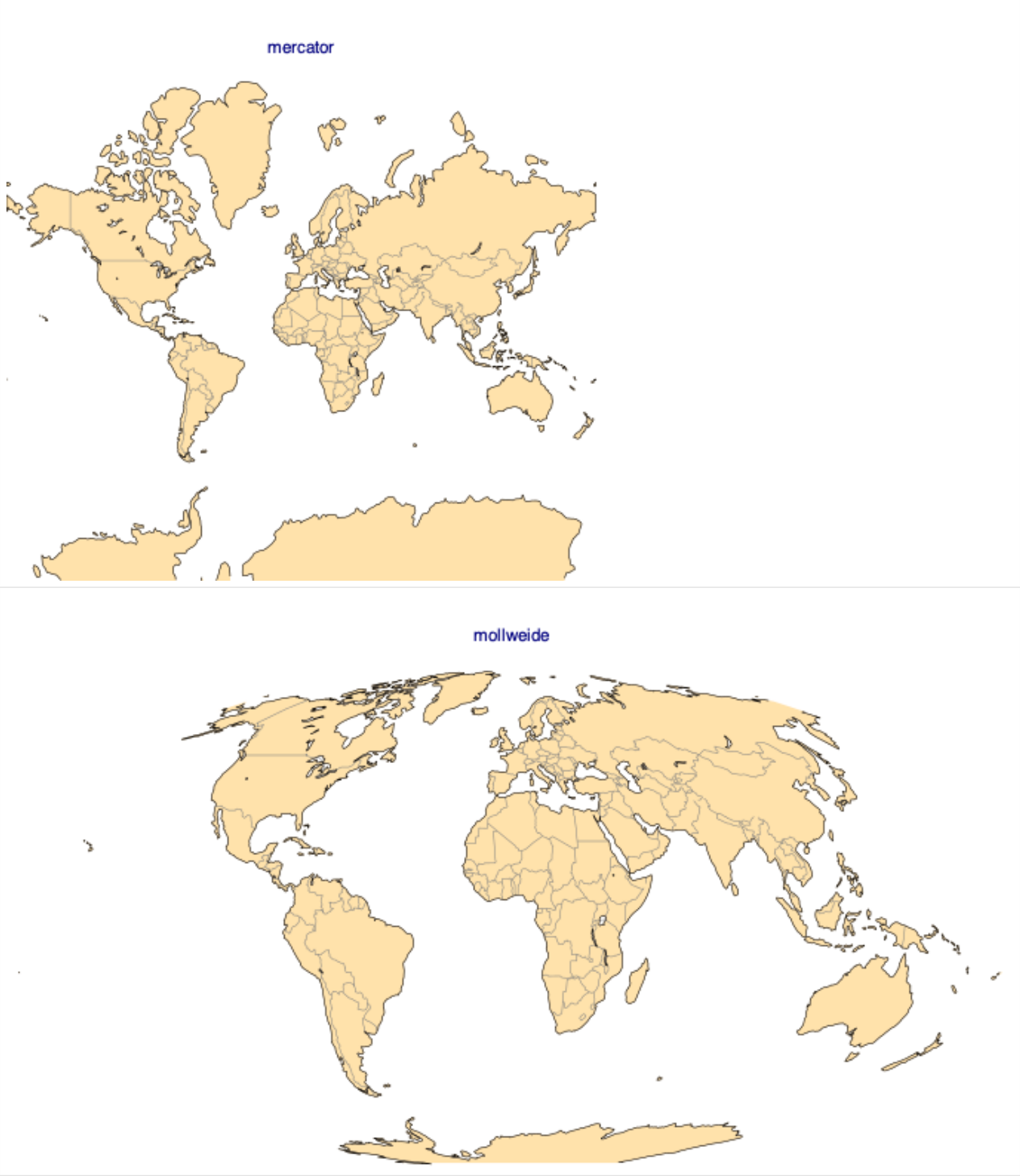

bonne


collignon
euro-atlantic
europe


europe-cylindrical


global


goode

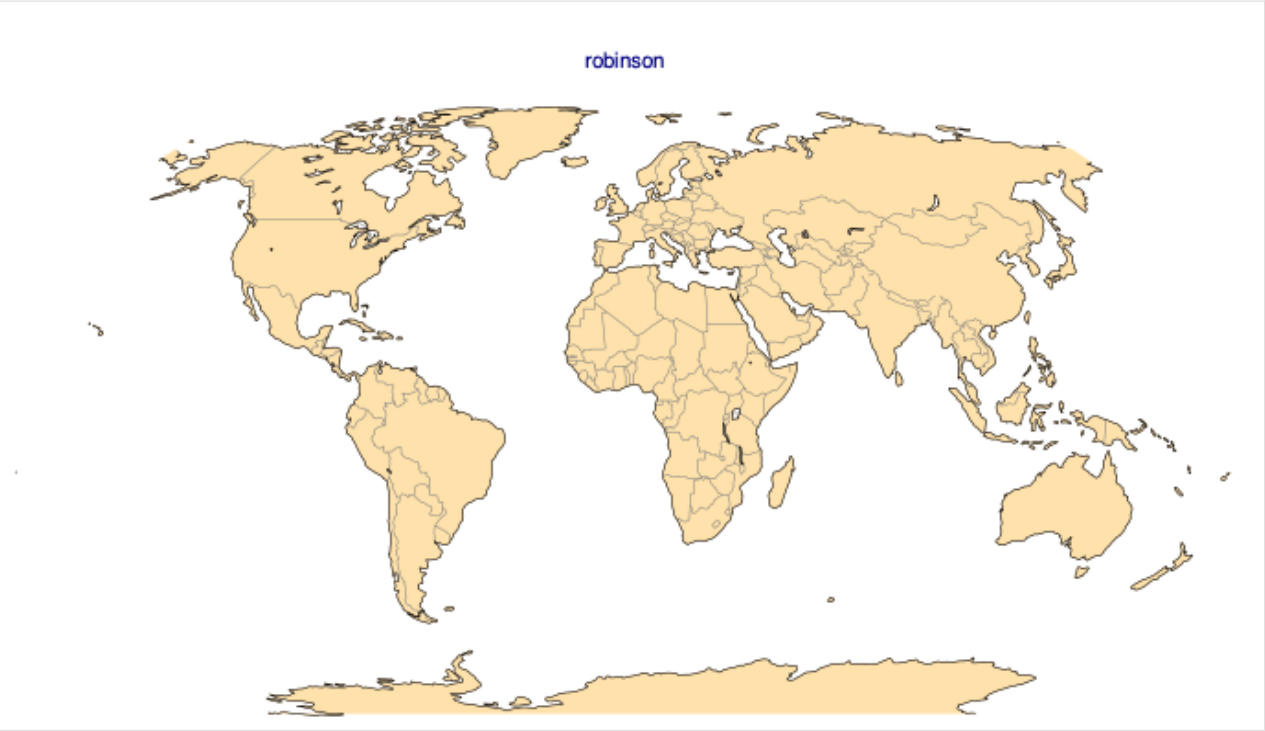

mercator
mollweide


north-america
north-america1

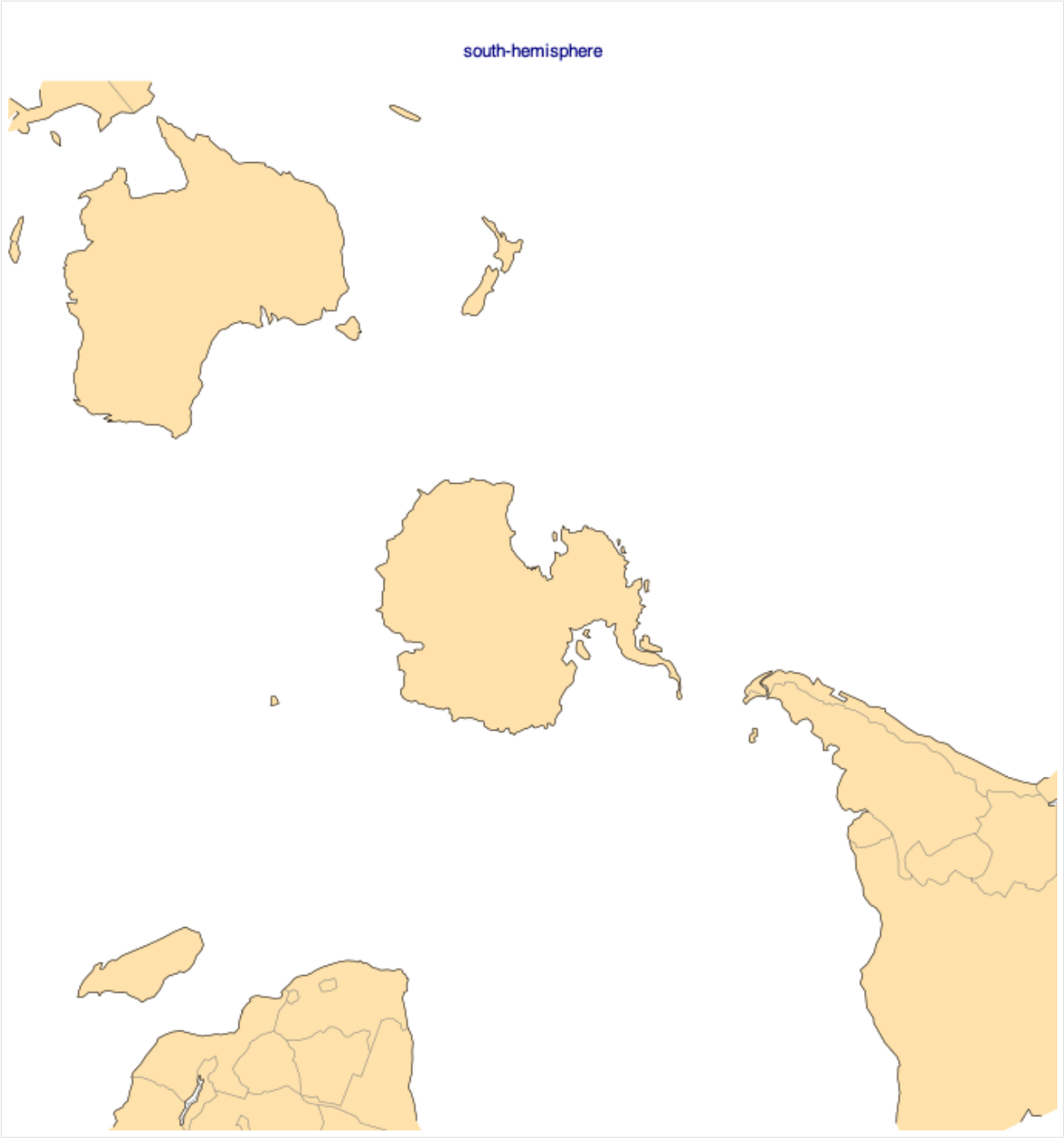

north-atlantic


north-hemisphere


polar-north


robinson


south-america
south-atlantic


south-hemisphere


south-pacific
tropics-east


tropics-west


web-mercator

Low-level graphical attributes

[13]:
```
cml.plotting_options(width=500)
```

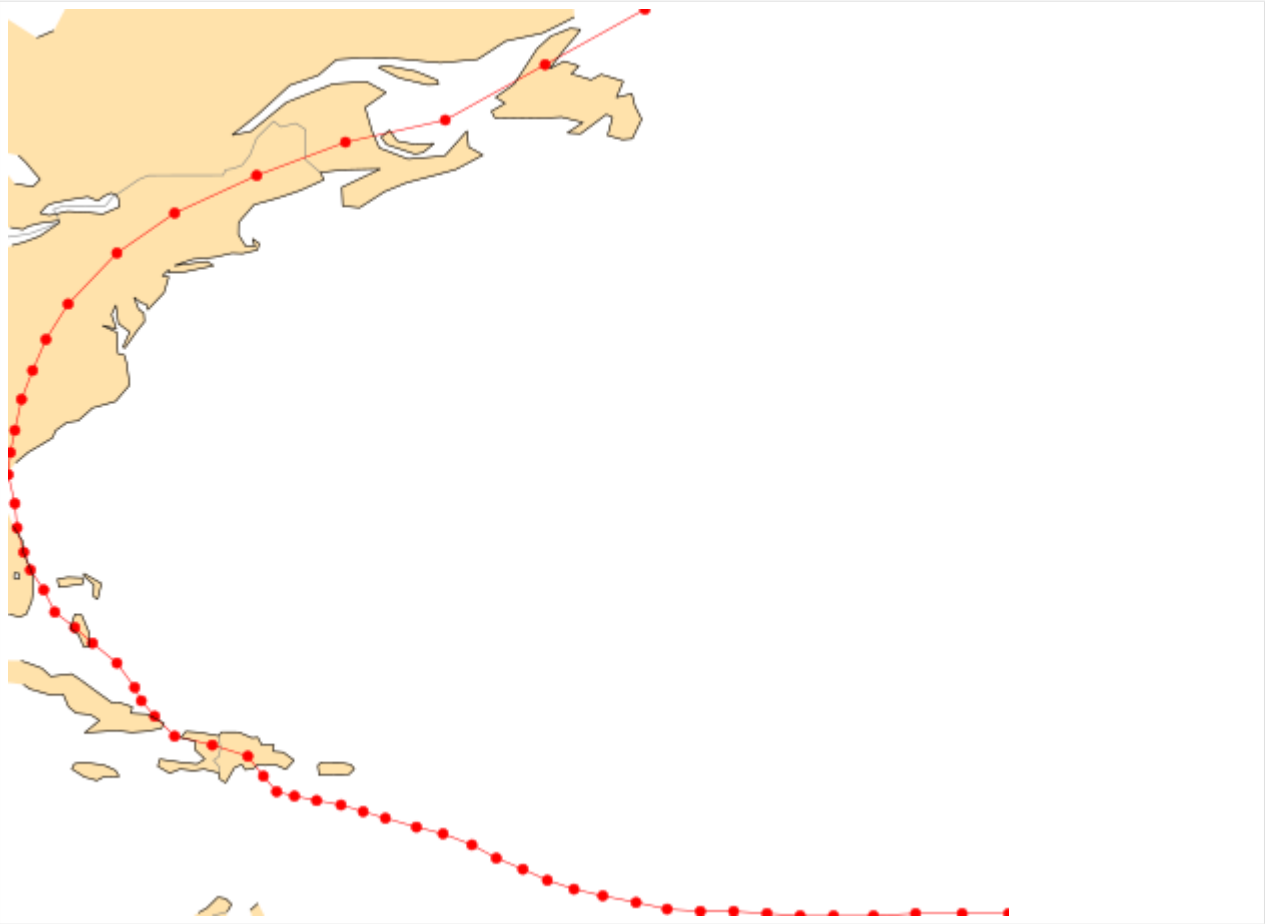
[14]:
```
atlantic = cml.load_dataset("hurricane-database", bassin="atlantic")
```

[15]:
```
df = atlantic.to_pandas()
```

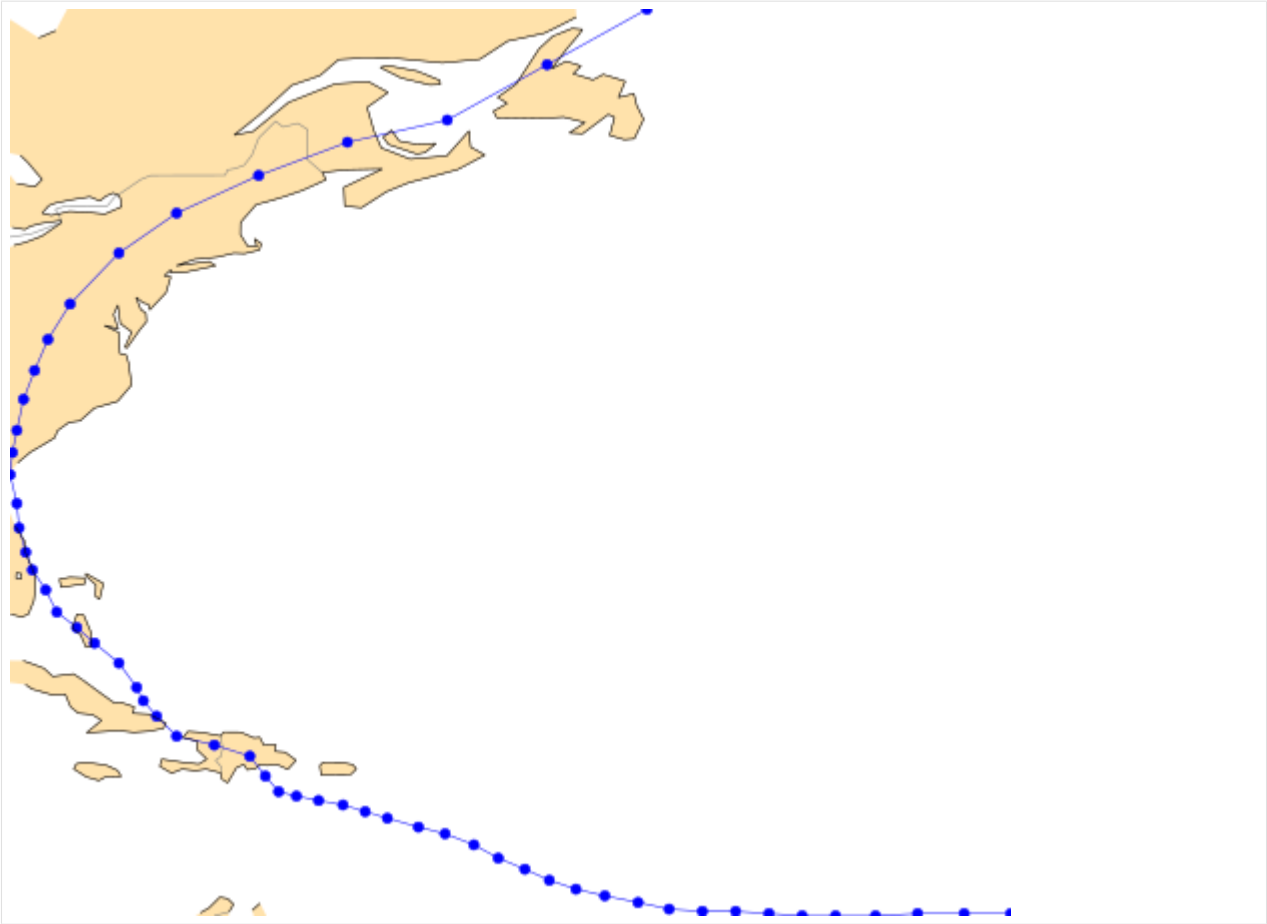
[16]:
```
david = df[(df.name == "david")]
```

[17]:
```
cml.plot_map(
    david, style=dict(symbol_colour="blue", symbol_type="marker", symbol_marker_index=3)
)
```

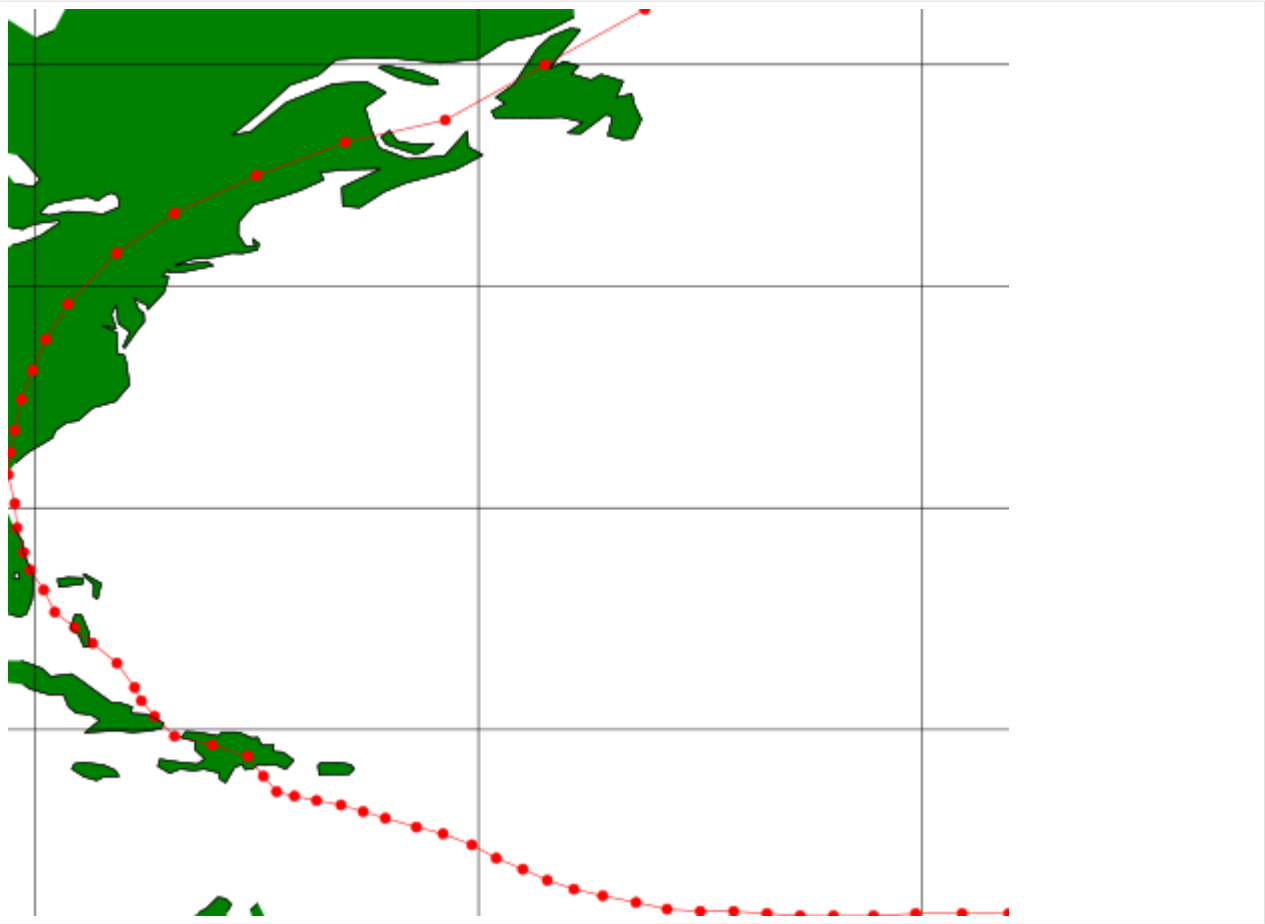
[18]:
```
cml.plot_map(david)
```



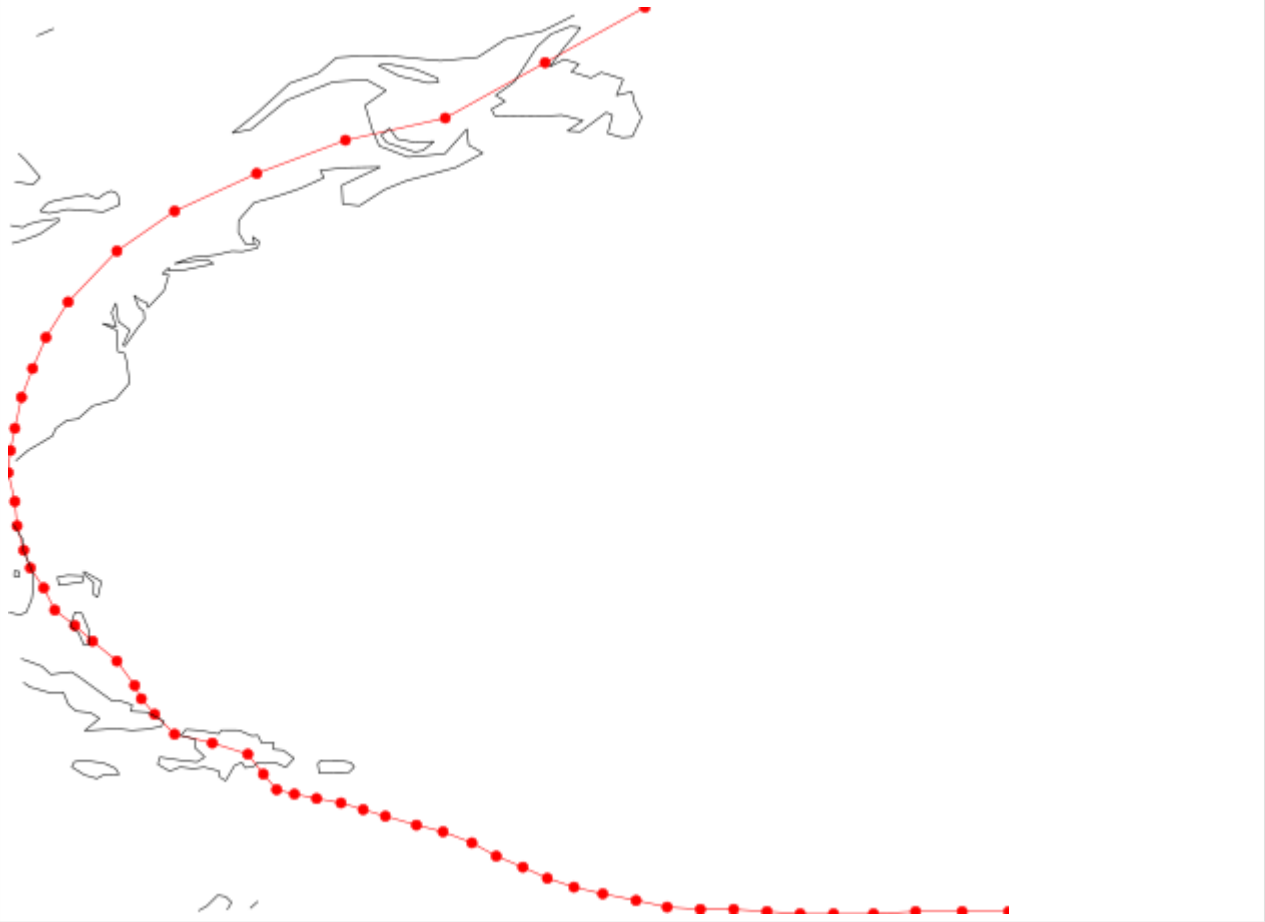
[19]:
```
cml.plot_map(david, style={"+symbol_colour": "blue"})
```



[20]:
```
cml.plot_map(
    david,
    background={
        "map_coastline_land_shade": True,
        "map_coastline_land_shade_colour": "green",
    },
)
```



[21]:
```
cml.plot_map(david, background=False)
```



You can run this notebook in , in , in

[1]:
```
!pip install --quiet climetlab
```

[2]:
```
import climetlab as cml
```

[3]:
```
source = cml.load_source(
    "cds",
    "insitu-observations-gruan-reference-network",
    variable="air_temperature",
    year="2017",
    month="01",
    day=[
        "01",
        "02",
        "03",
        "04",
        "05",
        "06",
        "07",
        "08",
        "09",
        "10",
```

(continues on next page)

(continued from previous page)

```
        "11",
        "12",
        "13",
        "14",
        "15",
        "16",
        "17",
        "18",
        "19",
        "20",
        "21",
        "22",
        "23",
        "24",
        "25",
        "26",
        "27",
        "28",
        "29",
        "30",
        "31",
    ],
    format="csv-lev.zip",
)
```

[4]:
```
df = source.to_pandas()
```

[5]:
```
df
```

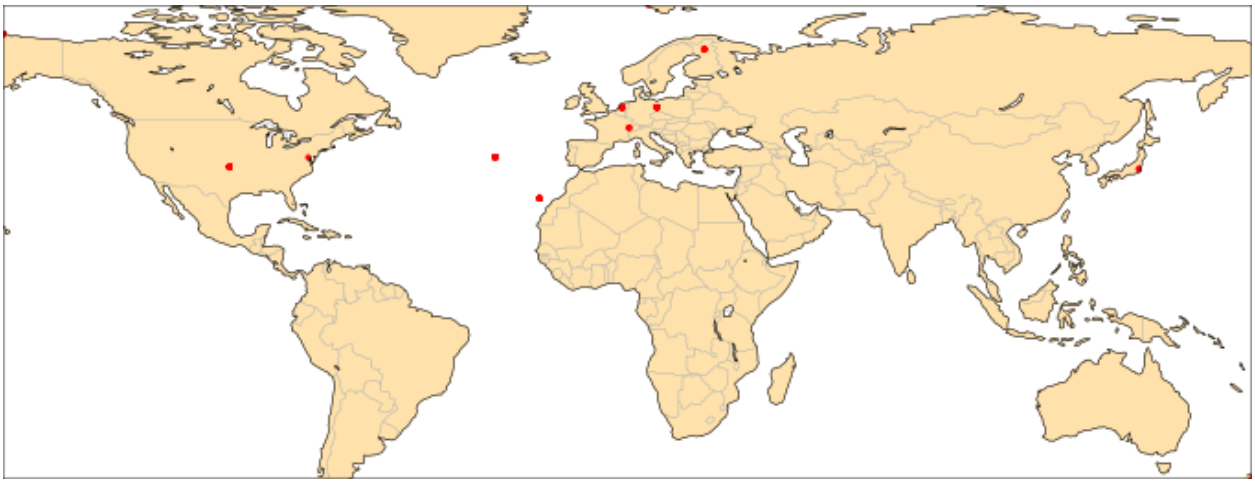
[5]:
```
        station_name          report_timestamp  time_since_launch  report_id  \
0                LIN 2017-01-01 00:00:00+00:00            0.00000     242982
1                LIN 2017-01-01 00:00:00+00:00            1.00000     242982
2                LIN 2017-01-01 00:00:00+00:00            2.00000     242982
3                LIN 2017-01-01 00:00:00+00:00            3.00012     242982
4                LIN 2017-01-01 00:00:00+00:00            4.00012     242982
...              ...                       ...                ...        ...
1645953          LIN 2017-01-31 18:00:00+00:00         5002.16000     246127
1645954          LIN 2017-01-31 18:00:00+00:00         5003.16000     246127
1645955          LIN 2017-01-31 18:00:00+00:00         5004.16000     246127
1645956          LIN 2017-01-31 18:00:00+00:00         5005.16000     246127
1645957          LIN 2017-01-31 18:00:00+00:00         5006.16000     246127

         longitude  latitude  height_of_station_above_sea_level  air_pressure  \
0          14.1203   52.2094                              103.8     101096.00
1          14.1203   52.2094                              103.8     101019.00
2          14.1204   52.2094                              103.8     100946.00
3          14.1204   52.2095                              103.8     100879.00
4          14.1204   52.2095                              103.8     100810.00
...            ...       ...                                ...          ...
1645953    14.3240   51.9910                              103.8       2354.49
1645954    14.3247   51.9909                              103.8       2352.38
1645955    14.3253   51.9909                              103.8       2350.24
```

(continues on next page)

(continued from previous page)

```
1645956    14.3260   51.9908                                103.8       2347.39
1645957    14.3267   51.9907                                103.8       2344.79

         air_pressure_total_uncertainty  air_temperature
0                                36.8895          274.357
1                                36.8886          274.321
2                                36.8878          274.282
3                                36.8870          274.255
4                                36.8861          274.257
...                                  ...              ...
1645953                          37.2003          198.804
1645954                          37.1999          198.776
1645955                          37.1994          198.747
1645956                          37.1988          198.710
1645957                          37.1982          198.670

[1645958 rows x 10 columns]
```

[6]:
```
cml.plot_map(df[(df.time_since_launch == 0)])
```



You can run this notebook in , in , in

[ ]:
```
!pip install --quiet climetlab[interactive]
```

1.4.15 Interactive maps

[4]:
```
import climetlab as cml
```

[5]:
```
atlantic = cml.load_dataset("hurricane-database", bassin="atlantic")
df = atlantic.to_pandas()
katrina = df[(df.name == "katrina") & (df.year == 2005)]
cml.interactive_map(katrina)
```

[5]:
```
<IPython.core.display.HTML object>
```

[ ]:

You can run this notebook in , in , in

[ ]:
```
!pip install --quiet climetlab
```

1.4.16 Creating a shared dataset of GRIBs

[1]:
```
import climetlab as cml
```

Download data to the climetlab cache

[ ]:
```
for month in range(1, 13):  # This takes a few minutes.
    cml.load_source(
        "mars",
        param=["2t"],
        levtype="sfc",
        area=[50, -50, 20, 50],
        grid=[1, 1],
        date=f"2012-{month}",
    )
```

[ ]:
```
cml.load_source(
    "mars",
    param="msl",
    levtype="sfc",
    area=[50, -50, 20, 50],
    grid=[1, 1],
    date="2012-12-01",
);
```

Export the data to a shared directory

This is optional, you could keep working on the data from the cache if you are the only user of the data and you do not mind redownloading it later. Other people should not use your cache: - When using climetlab the cache will eventually fills up and the data may be deleted automatically, - You will need to deal with permissions issues. - It will make it difficult to share the data with other people.

Let us export the data to a shared directory `shared-data/temperature-for-analysis`

[4]:
```
# Some housekeeping
!rm -rf shared-data/temperature-for-analysis
!mkdir -p shared-data/temperature-for-analysis
```

[5]:
```
# export all data from my cache which is from mars and not older that 1 day
!climetlab export_cache shared-data/temperature-for-analysis --newer 1h --match mars
```

```
Copying cache entries matching 'mars' and newer than '2023-03-11 13:29:29' to shared-
↪data/temperature-for-analysis.
100%|| 13/13 [00:00<00:00, 367.98it/s]
Copied 13 cache entries to shared-data/temperature-for-analysis.
```

Create indexes to speed up data access when using it. (Optional)

[ ]:
```
!climetlab index_directory shared-data/temperature-for-analysis
```

[ ]:
```
!climetlab availability shared-data/temperature-for-analysis
```

Using the data

[18]:
```
DATA = "shared-data/temperature-for-analysis"
```

[19]:
```
source = cml.load_source("indexed-directory", DATA)
```

[20]:
```
source.availability
```

[20]:
```
class=od, domain=g, expver=0001, levtype=sfc, md5_grid_
↪section=ce1bd075c48ae7a5bf34f4e47166e942, step=0, stream=oper, time=1200, type=an
   date=20120101/to/20121231, param=2t
   date=20121201, param=msl
```

This is a good time to check the data, is all the data here? Are they missing dates? Parameters?

The data is ready to be used as numpy, tensorflow or xarray object.

[11]:
```
source.sel(param="msl").to_numpy().mean()
```

[11]:
```
101725.47522756307
```

[22]:
```
cml.load_source("indexed-directory", DATA, param="msl").to_numpy().mean()
```

[22]:
```
101725.47522756307
```

[23]:
```
temp = source.sel(param="2t").order_by("date")
temp.to_tfdataset()
```

[23]:
```
<PrefetchDataset element_spec=TensorSpec(shape=<unknown>, dtype=tf.float32, name=None)>
```

[24]:
```
temp.to_xarray()
```

[24]:
```
<xarray.Dataset>
Dimensions:     (number: 1, time: 366, step: 1, surface: 1, latitude: 31,
                 longitude: 101)
Coordinates:
  * number      (number) int64 0
  * time        (time) datetime64[ns] 2012-01-01T12:00:00 ... 2012-12-31T12:0...
  * step        (step) timedelta64[ns] 00:00:00
  * surface     (surface) float64 0.0
```

(continues on next page)

(continued from previous page)

```
  * latitude    (latitude) float64 50.0 49.0 48.0 47.0 ... 23.0 22.0 21.0 20.0
  * longitude   (longitude) float64 -50.0 -49.0 -48.0 -47.0 ... 48.0 49.0 50.0
    valid_time  (time, step) datetime64[ns] ...
Data variables:
    t2m         (number, time, step, surface, latitude, longitude) float32 ...
Attributes:
    GRIB_edition:            1
    GRIB_centre:             ecmf
    GRIB_centreDescription:  European Centre for Medium-Range Weather Forecasts
    GRIB_subCentre:          0
    Conventions:             CF-1.7
    institution:             European Centre for Medium-Range Weather Forecasts
    history:                 2023-03-11T14:35 GRIB to CDM+CF via cfgrib-0.9.1...
```

[25]:
```
# Note that this is wrong (not implemented yet)
temp.availability
```

[25]:
```
class=od, domain=g, expver=0001, levtype=sfc, md5_grid_
↪section=ce1bd075c48ae7a5bf34f4e47166e942, step=0, stream=oper, time=1200, type=an
   date=20120101/to/20121231, param=2t
   date=20121201, param=msl
```

[ ]:

[ ]:

You can run this notebook in , in , in

1.4.17 Save data to GRIB

Here is an example of generating custom GRIB using climetlab.

To generate a GRIB, CliMetLab providesthe function `cml.new_grib_output()`, requiring the following arguments:

- A numpy array containing the data to save.
- An object containing a template for the GRIB metadata, such as a CliMetLab source
- (Optional) A dictionary of metadata to override the template metadata.

Using this function is similar to `open(path)` in python, as shown below:

A template for the GRIB metadata

[1]:
```
import climetlab as cml

s = cml.load_source(
    "cds",
    "reanalysis-era5-single-levels",
    variable=["2t", "msl"],
    product_type="reanalysis",
    area=[50, -50, 20, 50],
```

(continues on next page)

(continued from previous page)

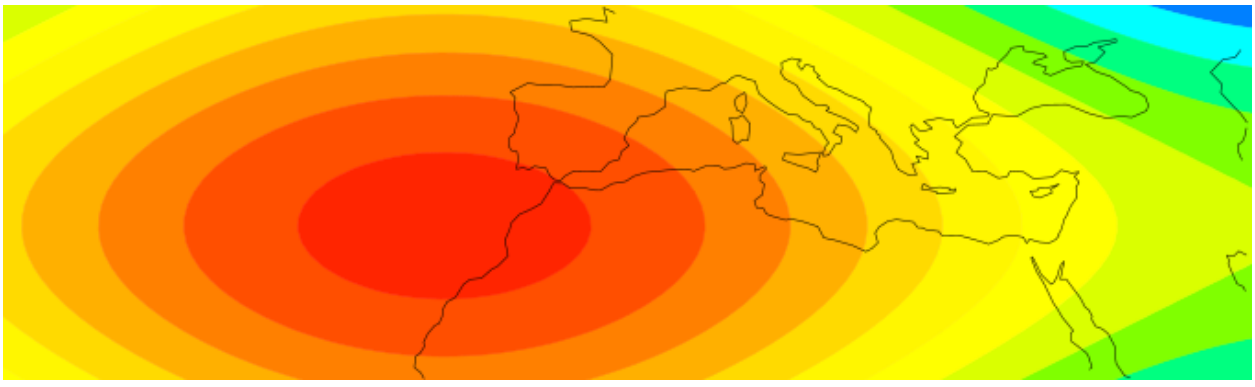
```
    date="2012-12-12",
    time="12",
)

template = s[0]  # temperature template
```

An numpy array with the data:

[2]:
```
import numpy as np
import math


def generate_numpy_field(shape, ref=273.15, offset1=0, offset2=0):
    f = np.zeros(shape=shape, dtype=np.float64)
    for i in range(f.shape[0]):
        for j in range(f.shape[1]):
            f[i, j] = (
                ref
                + (math.sin((i + offset1) / 45.0) + math.sin((j + offset2) / 90.0)) * 15
            )
    return f


arr = generate_numpy_field(template.shape)
```

Let us plot the generated field for inspection. Notice how `cml.plot_map()` also uses the template to find the metadata required to plot the field as a temperature field.

[3]:
```
cml.plot_map(arr, metadata=template)
```



Write to GRIB

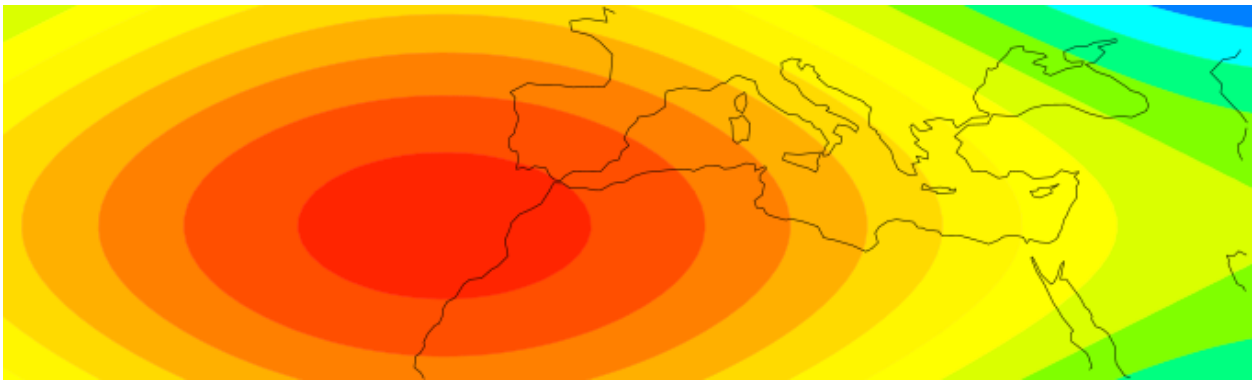
Method 1

Compare to the usual python code:

```
f = open(path)
f.write(data)
f.close()
```

[4]:
```
output = cml.new_grib_output("test1.grib", template=template)
output.write(arr)
output.close()
```

[5]:
```
cml.plot_map(cml.load_source("file", "test1.grib"))
```



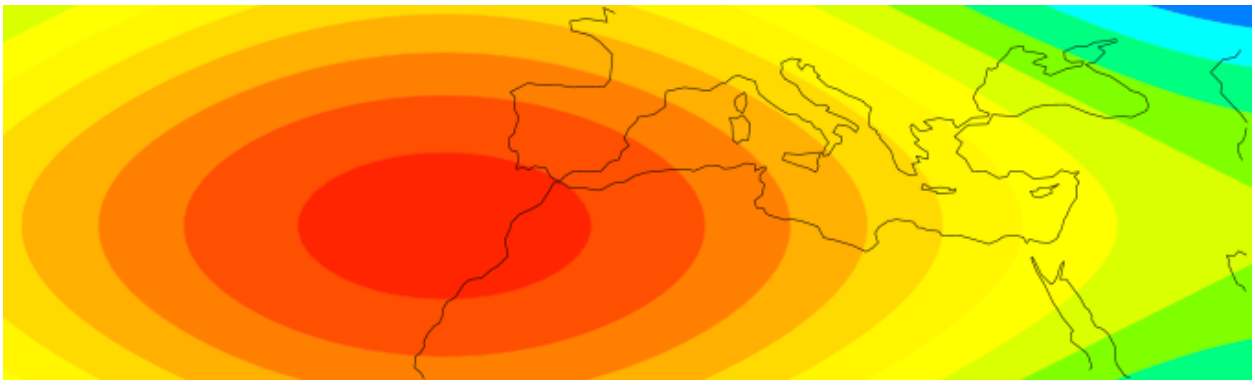
Write to GRIB

Method 2

The template can also be added in the `.write()` method instead of the `cml.new_grib_output()` function:

[6]:
```
output = cml.new_grib_output("test2.grib")
output.write(arr, template=template)
output.close()
```

[7]:
```
cml.plot_map(cml.load_source("file", "test2.grib"))
```



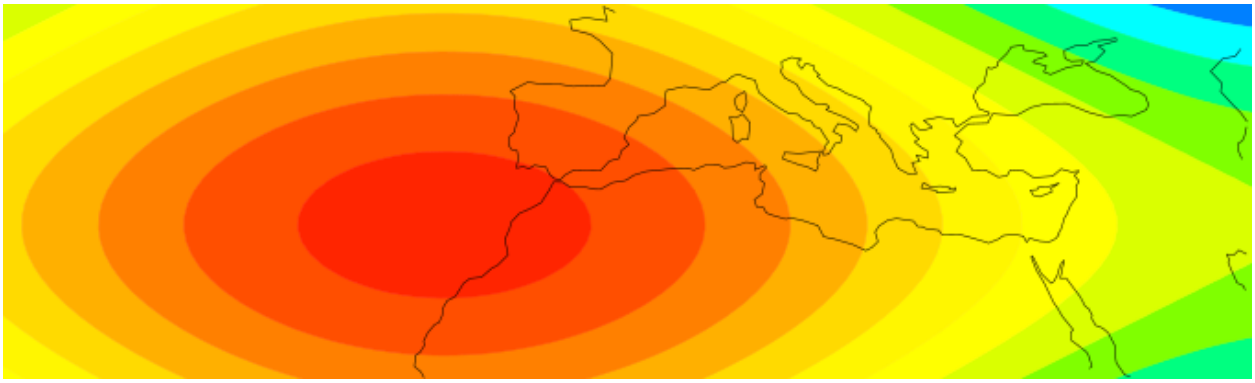
Method 3

Compare to the usual python code:

```
with open(path) as f:
    f.write(data)
```

[8]:
```
with cml.new_grib_output("test3.grib", template=template) as output:
    output.write(arr)
```

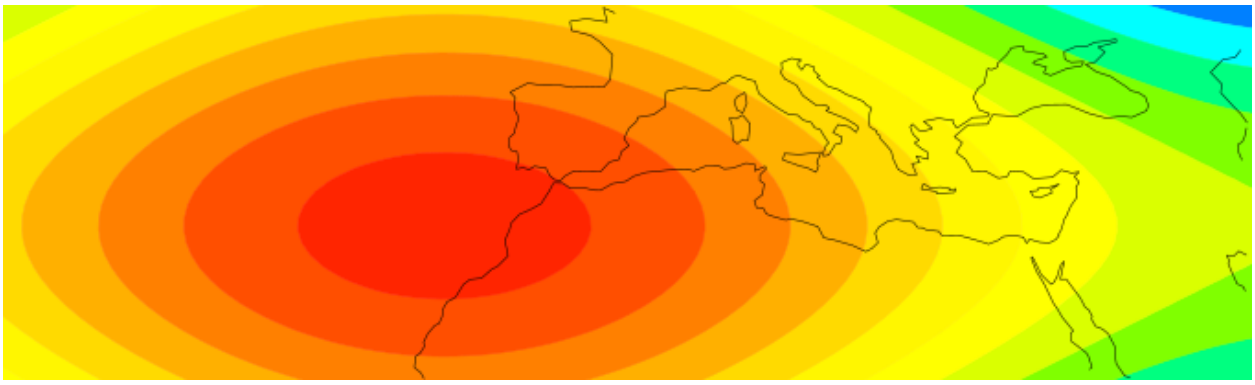
[9]:
```
cml.plot_map(cml.load_source("file", "test3.grib"))
```

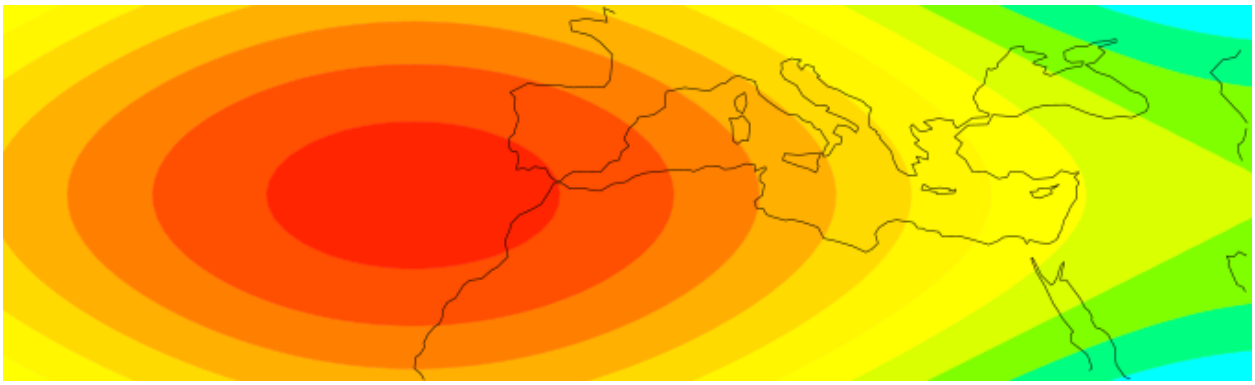


Once a GRIB output has been open, several fields can be written to it, as follows:

[10]:
```
with cml.new_grib_output("test4.grib", template=template) as output:
    for i in range(24):
        f = generate_numpy_field(template.shape, offset1=i, offset2=i)
        output.write(f, metadata={"time": i})
```

[11]:
```
test4 = cml.load_source("file", "test4.grib")

assert len(test4) == 24

cml.plot_map(test4[0])
cml.plot_map(test4[10])
cml.plot_map(test4[-1])
```





You can run this notebook in , in , in

[1]:
```
!pip install --quiet climetlab
```

1.4.18 Dataset availability

(Work in progress)

[2]:
```
from climetlab.utils.availability import Availability
from climetlab.decorators import availability
```

[3]:
```
C0 = [
    {"level": "500", "param": "Z", "step": "24"},
    {"level": "500", "param": "Z", "step": "36"},
    {"level": "500", "param": "Z", "step": "48"},
    {"level": "500", "param": "T", "step": "24"},
    {"level": "500", "param": "T", "step": "36"},
    {"level": "500", "param": "T", "step": "48"},
    {"level": "850", "param": "T", "step": "36"},
    {"level": "850", "param": "T", "step": "48"},
    {"level": "1000", "param": "Z", "step": "24"},
    {"level": "1000", "param": "Z", "step": "48"},
]
```

[4]:
```
a0 = Availability(C0)
```

[5]:
```
a0
```

[5]:
```
<climetlab.utils.availability.Availability at 0x114a0ad60>
```

[6]:
```
C1 = [
    {"level": "500", "param": "Z", "step": "24"},
    {"level": "500", "param": "Z", "step": "36"},
    {"level": "500", "param": "Z", "step": "48"},
    {"level": "500", "param": "T", "step": "24"},
    {"level": "500", "param": "T", "step": "36"},
    {"level": "500", "param": "T", "step": "48"},
    {"level": "850", "param": "T", "step": "36"},
    {"level": "850", "param": "T", "step": "48"},
    {"level": "1000", "param": "Z", "step": "24"},
    {"level": "1000", "param": "Z", "step": "48"},
    {"level": "850", "param": "Z", "step": "36"},
    {"level": "850", "param": "Z", "step": "48"},
]
```

[7]:
```
a1 = Availability(C1)
```

[8]:
```
a1
```

[8]:
```
<climetlab.utils.availability.Availability at 0x10341e430>
```

[9]:
```
a1.select(param="Z")
```

[9]:
```
<climetlab.utils.availability.Availability at 0x114a0a790>
```

[10]:
```
a1.select(param="T")
```

[10]:
```
<climetlab.utils.availability.Availability at 0x114a2d0d0>
```

[11]:
```
for r in a1.select(param="T").iterate():
    print(r)
```

```
{'param': ('T',), 'level': ('500',), 'step': ('24', '36', '48')}
{'param': ('T',), 'level': ('850',), 'step': ('36', '48')}
```

[12]:
```
for r in a1.select(param="T").iterate(True):
    print(r)
```

```
{'param': 'T', 'level': '500', 'step': '24'}
{'param': 'T', 'level': '500', 'step': '36'}
{'param': 'T', 'level': '500', 'step': '48'}
{'param': 'T', 'level': '850', 'step': '36'}
{'param': 'T', 'level': '850', 'step': '48'}
```

[13]:
```
a1.select(step="36")
```

[13]:
```
<climetlab.utils.availability.Availability at 0x114a2daf0>
```

[14]:
```
a1.select(step="36", param="T")
```

[14]:
```
<climetlab.utils.availability.Availability at 0x114a2db20>
```

[15]:
```
a1.count(step="36", param="T")
```

[15]:
```
2
```

[16]:
```
a1.count()
```

[16]:
```
12
```

[17]:
```
a1.select(step="22")
```

[17]:
```
<climetlab.utils.availability.Availability at 0x114a2d7c0>
```

[18]:
```
a1.select(param="Z", step="99").count()
```

[18]:
```
0
```

[19]:
```
a3 = Availability(
    [
        {"date": ["1990-01-01/1990-01-02"], "param": ["Z", "T"]},
        {"date": ["1990-01-02/1990-01-05"], "param": ["Z"]},
        {"date": ["1990-01-04/1990-01-15"], "param": ["Z", "T"]},
    ],
    intervals=["date"],
)
```

[20]:
```
a3
```

[20]:
```
<climetlab.utils.availability.Availability at 0x114a2d460>
```

[21]:
```
a3.count()
```

[21]:
```
29
```

[22]:
```
a3.select(date="1990-01-02/1990-01-04")
```

[22]:
```
<climetlab.utils.availability.Availability at 0x114a2d610>
```

[23]:
```
for r in a3.select(date="1990-01-02/1990-01-04").iterate(True):
    print(r)
```

```
{'date': datetime.date(1990, 1, 2), 'param': 'T'}
{'date': datetime.date(1990, 1, 4), 'param': 'T'}
{'date': datetime.date(1990, 1, 2), 'param': 'Z'}
{'date': datetime.date(1990, 1, 3), 'param': 'Z'}
{'date': datetime.date(1990, 1, 4), 'param': 'Z'}
```

[24]:
```
a3.missing(param="T", date="1990-01-01/1990-01-15")
```

[24]:
```
<climetlab.utils.availability.Availability at 0x114a2dc10>
```

[ ]:
```

```

User Guide (TODO)

1.5 Howtos

1.5.1 How to install CliMetLab?

```
pip install climetlab
```

See the *installing instructions* for more details.

1.5.2 How to access data?

There are two ways to access data using CliMetLab:

- *Using a Dataset*: CliMetLab provides a few demo datasets. In order to access other datasets with `cml.load_dataset()`, the relevant plugin must be installed.
- *Using a data Source*: A data Source allows loading various kinds of data format and location through `cml.load_source()`. Data sources should be used when there is no dataset plugin for the data you are interested in.

1.5.3 How do I access data from this BUFR/GRIB/... file?

Use the *file* source.

1.5.4 How do I access data from this file on this unsupported format?

Write your own python code to open the file.

Todo: Add plugins and doc for readers.

1.5.5 How to help others to use my data ?

Creating a CliMetLab plugin can be a solution to share some code along with the dataset that you are publishing/using. See the *plugin documentation*.

1.5.6 How to set up my CliMetLab cache directory ?

See *Caching*.

1.5.7 How to share my cache directory with another user ?

It is not recommended to share your cache with others. What you are looking for may be a mirror. And this feature is not implemented yet.

A more robust approach is to put the data to a shared location, and define *a CliMetLab dataset* plugin to use it.

1.5.8 Can CliMetLab help using dask ?

See *Using dask*.

1.6 Datasets

A **Dataset** is an object created using `cml.load_dataset(name, arg1, arg2=..., ...)` with the appropriate **name** and **arguments**, which provides access to a clearly well-defined **dataset**: these data have been defined and curated by somebody providing code along with the data. It also provides **metadata** and **additional functionalities**.

- The **name** is a string that uniquely identifies the dataset.
- The **argument(s)** `arg1` and keyword argument(s) `arg2` can be used to specify a subset of the dataset.
- The **data** can be accessed using methods such as `to_xarray()` or `to_pandas()` or other.
- Relevant **metadata** are attached directly to the dataset to provides additional information such as *an URL, a citation, licence, etc.*
- **Additional functionalities**: When working on data, we are often writing code to transform, preprocess, adapt the data to our needs. While it may be very nice to understand deeply to deep magic under the hood, this process could be very time-consuming. Once somebody else did the hard work of preparing the data for a given purpose and designing relevant functions to process it, their code can be integrated and made available to others through a CliMetLab dataset plugin. The Dataset object is an instance of a Python class in which the plugin maintainers/users can share additional code.

Note: *Dataset* objects differ from data *Source* objects, as Datasets refer to a given set of data (such as "the 2m temperature on Europe in 2015", while Sources are more generic such as "url").

CliMetLab has build-in datasets (as examples) and most of the datasets are available as plugins (non-exhaustive *list of CliMetLab plugins*).

1.6.1 How to load a dataset?

The relevant plugin must be installed first, using pip.

```
pip install climetlab-demo-dataset
```

This ensures that the dataset can be loaded with `climetlab.load_dataset()`.

```
>>> import climetlab as cml
>>> ds = cml.load_dataset("demo-dataset")
```

The first argument is the name of the dataset. It is used to find the relevant plugin and class to use. Other arguments are defined by the plugin maintainer and are documented in the plugin documentation (see *List of CliMetLab plugins*).

The Dataset object provides methods to access and use its data such as `to_xarray()` or `to_pandas()` or `to_numpy()` (there are other *methods that can be used to access data* from a Dataset).

```
>>> ds.to_xarray() # for gridded data
>>> ds.to_pandas() # for non-gridded data
>>> ds.to_numpy() # When the data is a n-dimensional array.
>>> ds.to_tfrecord() # Experimental
```

Note: The name of the python package for a CliMetLab plugin usually starts with "climetlab-".

Note: The plugin name does not necessarily match the dataset name, and one plugin can provide several datasets. As an example, the plugin `climetlab-s2s-ai-challenge` provides the datasets `s2s-ai-challenge-training-input` and `s2s-ai-challenge-training-output`:

```
>>> !pip install climetlab-s2s-ai-challenge
>>> climetlab.load_dataset("s2s-ai-challenge-training-input", ...)
>>> climetlab.load_dataset("s2s-ai-challenge-training-output", ...)
```

1.6.2 Xarray for gridded data

Gridded data typically are field data such as temperature or wind from climate or weather models or satellite images.

```
>>> import climetlab as cml
>>> ds = cml.load_dataset("demo-dataset")
>>> ds.to_xarray()
<xarray.Dataset>
Dimensions:    (latitude: 181, longitude: 360)
Coordinates:
  * longitude  (longitude) float64 -180.0 -179.0 -178.0 ... 177.0 178.0 179.0
  * latitude   (latitude) float64 90.0 89.0 88.0 87.0 ... -88.0 -89.0 -90.0
Data variables:
    t2m        (latitude, longitude) float64 273.1 273.3 273.5 ... 250.7 250.6
```

1.6.3 Pandas for non-gridded data

None-gridded data typically is tabular non-structured data such as observations. It often includes a column for the latitude and longitude of the data.

```
>>> import climetlab as cml
>>> ds = cml.load_dataset("dataset-name", **options)
>>> ds.to_pandas()
```

1.6.4 Metadata attached to a dataset

Metadata attached to a dataset include the following.

home_page: A link to the home page related to the dataset.

licence: A link to the licence of the dataset.

documentation: A link to the documentation related to the dataset.

citation: A citation related to the dataset.

terms_of_use: A text or link to the terms of use of the data.

```
>>> import climetlab as cml
>>> ds = cml.load_dataset("demo-dataset")
>>> ds.home_page
'https://github.com/ecmwf/climetlab-demo-dataset'
>>> ds.documentation
'Generates a dummy temperature field'
```

1.6.5 Best practices

Note: When sharing a python notebook, it is a good practice to add `!pip install climetlab-...` at the top of the notebook. If the package is not installed, CliMetLab fails with a NameError exception.

```
>>> # if the package climetlab-demo-dataset is not installed
>>> import climetlab as cml
>>> ds = cml.load_dataset("demo-dataset")
NameError: Cannot find dataset 'demo-dataset' (values are: ...),
```

```
>>> !pip install climetlab_demo_dataset --quiet
>>> import climetlab as cml
>>> ds = cml.load_dataset("demo-dataset")
>>>
```

There is no need to import the plugin package to enable loading the dataset:

```
>>> import climetlab_demo_dataset  # Not needed
```

1.7 Data sources

1.7.1 What is a data Source?

A data **Source** is an object created using `cml.load_source(name, *args, **kwargs)` with the appropriate name and arguments, which provides access to the data.

A Source also provides metadata and additional functionalities:

- The source **name** is a string that uniquely identifies the source type.
- The **arguments** (args and kwargs) are used to specify the data location to access the data. They can include additional parameters to access the data.
- The **additional functionalities** include python code related to caching, plotting and interacting with other data.

```
>>> import climetlab as cml
>>> source = cml.load_source(name, *args, **kwargs)
```

The Source object provides methods to access and use its data such as `to_xarray()` or `to_pandas()` or `to_numpy()` (there are other *methods that can be used to access data* from a data Source).

```
>>> source.to_xarray() # for gridded data
>>> source.to_pandas() # for non-gridded data
>>> source.to_numpy() # When the data is a n-dimensional array.
>>> source.to_tfrecord() # Experimental
```

Note: *Source* objects differ from data *Dataset* objects, as Datasets refer to a given set of data (such as "the 2m temperature on Europe in 2015", while Sources are more generic such as "url").

1.7.2 Which data Sources are available?

CliMetLab has built-in sources and additional sources can be made available *as plugins*.

Built-in data sources:

- *file* source: Load data from a file.
- *url* source: Load data from a URL.
- *url-pattern* source: Load data from list of URL created from a pattern.
- *cds* source: Load data from the Copernicus Data Store (CDS).
- *mars* source: Load data from the Meteorological Archival and Retrieval System at ECMWF (MARS).
- *multi (advanced usage)* source: Aggregate multiple sources.
- *zenodo* source (experimental): Load data from Zenodo.
- *indexed_urls* source (experimental): Load data from GRIB urls with partial download.

1.7.3 file

The simplest data source is the "file" source that accesses a local file.

```
>>> import climetlab as cml
>>> data = cml.load_source("file", "path/to/file")
>>> data.to_xarray() # for gridded data fields
>>> data.to_pandas() # for non-gridded data
```

CliMetLab will inspect the content of the file to check for any of the supported data formats listed below:

- **Fields:**
  - NetCDF
  - GRIB (see *GRIB support*)
- **Observations:**
  - CSV (comma-separated values)
  - BUFR (https://en.wikipedia.org/wiki/BUFR)
  - ODB (a bespoke binary format for observations)
- **Archive formats:** When given an archive format such as `.zip`, `.tar`, `.tar.gz`, etc, *CliMetLab* will attempt to open it and (recursively) extract any usable file.

```
>>> import climetlab as cml
>>> data = cml.load_source("url",
                           "https://www.example.com/data.tgz",
                           unpack=False)
```

Todo: Support for additionnal formats could be implemented as plugins.

GRIB file example

```
>>> import climetlab as cml
>>> data = cml.load_source("file", "examples/test.grib")
>>> data.to_xarray()
<xarray.Dataset>
Dimensions:     (number: 1, time: 1, step: 1, surface: 1, latitude: 11,␣
↪longitude: 19)
Coordinates:
  * number      (number) int64 0
  * time        (time) datetime64[ns] 2020-05-13T12:00:00
  * step        (step) timedelta64[ns] 00:00:00
  * surface     (surface) float64 0.0
  * latitude    (latitude) float64 73.0 69.0 65.0 61.0 ... 45.0 41.0 37.0 33.0
  * longitude   (longitude) float64 -27.0 -23.0 -19.0 -15.0 ... 37.0 41.0 45.0
    valid_time  (time, step) datetime64[ns] ...
Data variables:
    t2m         (number, time, step, surface, latitude, longitude) float32 ...
    msl         (number, time, step, surface, latitude, longitude) float32 ...
```

(continues on next page)

(continued from previous page)

```
Attributes:
    GRIB_edition:            1
    GRIB_centre:             ecmf
    GRIB_centreDescription:  European Centre for Medium-Range Weather Forecasts
    GRIB_subCentre:          0
    Conventions:             CF-1.7
    institution:             European Centre for Medium-Range Weather Forecasts
    history:                 2022-02-08T10:50 GRIB to CDM+CF via cfgrib-0.9.1..
↪.
```

NetCDF file example

```
>>> import climetlab as cml
>>> data = cml.load_source("file", "examples/test.nc")
>>> data.to_xarray()
<xarray.Dataset>
Dimensions:     (number: 1, time: 1, step: 1, surface: 1, latitude: 11,␣
↪longitude: 19)
Coordinates:
  * number      (number) int64 0
  * time        (time) datetime64[ns] 2020-05-13T12:00:00
  * step        (step) timedelta64[ns] 00:00:00
  * surface     (surface) float64 0.0
  * latitude    (latitude) float64 73.0 69.0 65.0 61.0 ... 45.0 41.0 37.0 33.0
  * longitude   (longitude) float64 -27.0 -23.0 -19.0 -15.0 ... 37.0 41.0 45.0
    valid_time  (time, step) datetime64[ns] ...
Data variables:
    t2m         (number, time, step, surface, latitude, longitude) float32 ...
    msl         (number, time, step, surface, latitude, longitude) float32 ...
Attributes:
    GRIB_edition:            1
    GRIB_centre:             ecmf
    GRIB_centreDescription:  European Centre for Medium-Range Weather Forecasts
    GRIB_subCentre:          0
    Conventions:             CF-1.7
    institution:             European Centre for Medium-Range Weather Forecasts
    history:                 2022-02-08T10:50 GRIB to CDM+CF via cfgrib-0.9.1..
↪.
```

Other format file example

If the format is supported, see the *Examples* notebooks for a working example.

If the format is not supported, additional code can be included in CliMetLab to support it.

Todo: Support for additionnal formats could be implemented as plugins.

1.7.4 url

The "url" data source is very similar to the "file" source.

This sources downloads the data from the specified address and stores it in the *cache*, then it operates similarly to the *"file" source* above. The supported data formats are the same as for the *"file" source*.

```
>>> import climetlab as cml
>>> data = cml.load_source("url", "https://www.example.com/data.csv")
```

When given an archive format such as `.zip`, `.tar`, `.tar.gz`, etc, *CliMetLab* will attempt to open it and extract any usable file. If you want to keep the downloaded file as is, pass `unpack=False` to the method.

```
>>> import climetlab as cml
>>> data = cml.load_source("url",
                           "https://www.example.com/data.tgz",
                           unpack=False)
```

1.7.5 url-pattern

The "url-pattern" data source will build urls from the pattern specified, using the other arguments to fill the pattern. Each argument can be a list to iterate and create the cartesian product of all lists. Then each url is downloaded and stored it in the *cache*. The supported download the data from the address data formats are the same as for the *file* and *url* data sources above.

```
import climetlab as cml

data = cml.load_source("url-pattern",
     "https://www.example.com/data-{foo}-{bar}-{qux}.csv",
     foo = [1,2,3],
     bar = ["a", "b"],
     qux = "unique"
     )
```

The code above will download and process the data from the six following urls:

1. https://www.example.com/data-1-a-unique.csv
2. https://www.example.com/data-2-a-unique.csv
3. https://www.example.com/data-3-a-unique.csv
4. https://www.example.com/data-1-b-unique.csv
5. https://www.example.com/data-2-b-unique.csv
6. https://www.example.com/data-3-b-unique.csv

If the urls are pointing to archive format, the data will be unpacked by `url-pattern` according to the **unpack** argument, similarly to what the source `url` does (see above the *url* source).

Once the data have been properly downloaded [and unpacked] and cached. It can can be accessed using `to_xarray()` or `to_pandas()`.

To provide a unique xarray.Dataset (or pandas.DataFrame), the different datasets are merged. The default merger strategy for field data is to use `xarray.open_mfdataset` from *xarray*. This can be changed by providing a custom merger to the `url-pattern` source. See *merging sources*

1.7.6 cds

The "cds" data source accesses the Copernicus Climate Data Store (CDS), using the cdsapi package. A typical *cdsapi* request has the following format:

```
import cdsapi

client = cdsapi.Client()

client.retrieve("dataset-name",
                {"parameter1": "value1",
                 "parameter2": "value2",
                 ...})
```

to perform the same operation with *CliMetLab*, use the following code:

```
import climetlab as cml

data = cml.load_source("cds",
                       "dataset-name",
                       {"parameter1": "value1",
                        "parameter2": "value2",
                        ...})
```

Data downloaded from the CDS is stored in the the *cache*.

To access data from the CDS, you will need to register and retrieve an access token. The process is described here.

For more information, see the CDS knowledge base.

1.7.7 mars

The "mars" source allows handling data from the Meteorological Archival and Retrieval System (MARS).

To figure out which data you need, or discover relevant data available on MARS, see the publicly accessible MARS catalog (or this access restricted catalog). Notice that various datasets of interests are also publicly available. To access data from the MARS, you will need to register and retrieve an access token. For a more extensive documentation about MARS, please refer to the MARS documentation (or its access from the internet through its web API).

```
from ecmwfapi import ECMWFDataServer

server = ECMWFDataServer()

client.retrieve(
    { "parameter1": "value1",
      "parameter2": "value2",
      ...
    })
```

to perform the same operation with *CliMetLab*, use the following code:

```
import climetlab as cml

data = cml.load_source("mars",
    { "parameter1": "value1",
```

(continues on next page)

(continued from previous page)

```
    "parameter2": "value2",
    ...
})
```

Data downloaded from MARS is stored in the the *cache*.

Examples

See the *Examples* notebooks for a working example.

1.7.8 zenodo

Zenodo is a general-purpose open repository developed and operated by CERN. It allows researchers to deposit research papers, datasets, etc. For each submission, a persistent digital object identifier (DOI) is minted, which makes the stored items easily citeable.

The `"zenodo"` source provides access data from zenodo.org, including downloading, caching, etc.

```
>>> ds = load_source("zenodo", record_id=...)
```

Example

```
>>> import climetlab as cml
>>> def only_csv(path):
        return path.endswith(".csv")
>>> source = cml.load_source("zenodo", record_id=5020468, filter=only_csv)
>>> source.to_pandas()
```

Note: Support for zenodo access is experimental.

1.7.9 indexed_urls

```
>>> ds = load_source( "indexed-urls", index, request)
```

Experimental. See *GRIB support*.

1.7.10 multi (advanced usage)

```
>>> ds = load_source( "multi", source1, source2, ...)
```

Todo: add documentation on multi-source.

1.8 Data Manipulation

Todo: This section is a draft.

1.8.1 Methods provided by CliMetLab data objects

Methods provided by CliMetLab data objects (such as a Dataset, a data Source or a Reader): Depending on the data, some of these methods are or are not available.

A CliMetLab data object provides methods to access and use its data.

```
>>> source.to_xarray() # for gridded data
>>> source.to_pandas() # for non-gridded data
>>> source.to_numpy() # When the data is a n-dimensional array.
>>> source.to_tfrecord() # Experimental
```

Todo: Explain fields.to_xarray() and obs.to_pandas().

Explain data[0]

Add here more details about the .to_… methods.

1.8.2 Iterating

When a CliMetLab data *source* or dataset provides a list of fields, it can be iterated over to access each field (in a given order see *below*).

Let us get a source of fields from the Climate Data Store (CDS) and iterate through the list, each element is a field.

```
>>> import climetlab as cml
>>> ds = cml.load_source(
        "cds",
        "reanalysis-era5-single-levels",
        param=["2t", "msl"],
        product_type="reanalysis",
        grid='5/5',
        date=["2012-12-12", "2012-12-13"],
        time=[600, 1200, 1800],
    )

>>> len(ds)
10

>>> for f in ds: print(f)
GribField(2t,None,20121212,600,0,0)
GribField(msl,None,20121212,600,0,0)
GribField(2t,None,20121212,1200,0,0)
GribField(msl,None,20121212,1200,0,0)
GribField(2t,None,20121212,1800,0,0)
```

(continues on next page)

(continued from previous page)

```
GribField(msl,None,20121212,1800,0,0)
GribField(2t,None,20121213,600,0,0)
GribField(msl,None,20121213,600,0,0)
GribField(2t,None,20121213,1200,0,0)
GribField(msl,None,20121213,1200,0,0)
GribField(2t,None,20121213,1800,0,0)
GribField(msl,None,20121213,1800,0,0)
```

1.8.3 Selection with `[...]`

When a CliMetLab data *source* or dataset provides a list of fields, it can be *iterated* over to access each field (in a given order see *below*).

A subset of the list can be created using the standard python list interface relying on brackets and slices.

```
>>> import climetlab as cml
>>> ds = cml.load_source(
        "cds",
        "reanalysis-era5-single-levels",
        param=["2t", "msl"],
        product_type="reanalysis",
        grid='5/5',
        date=["2012-12-12", "2012-12-13"],
        time=[600, 1200, 1800],
    )

>>> len(ds)
10

>>> print(f[0])
GribField(2t,None,20121212,600,0,0)

>>> for f in ds[0:3]: print(f)
GribField(2t,None,20121212,600,0,0)
GribField(msl,None,20121212,600,0,0)
GribField(2t,None,20121212,1200,0,0)

>>> for f in ds[0:5:2]: print(f)
GribField(2t,None,20121212,600,0,0)
GribField(2t,None,20121212,1200,0,0)
GribField(2t,None,20121212,1800,0,0)
```

1.8.4 Selection with `.sel()`

When a CliMetLab data *source* or dataset provides a list of fields, it can be *iterated* over to access each field (in a given order see *below*).

The method `.sel()` allows filtering this list to **select a subset** of the list of fields.

For instance, the following examples shows how to select various subsets of fields from a list of fields. After selection the required list of fields, the selected data from this subset is available with the methods `.to_numpy()`, `.to_pytorch()`, `.to_xarray()`, etc...

This list of fields can be filtered to extract on the fields corresponding to the 2m-temperature parameter with `.sel(param="2t")`:

```
>>> import climetlab as cml
>>> ds = cml.load_source(
        "cds",
        "reanalysis-era5-single-levels",
        param=["2t", "msl"],
        product_type="reanalysis",
        grid='5/5',
        date=["2012-12-12", "2012-12-13"],
        time=[600, 1200, 1800],
    )

>>> len(ds)
10

>>> subset = ds.sel(param="2t")
>>> len(subset)
6
>>> for f in subset:
GribField(2t,None,20121212,600,0,0)
GribField(2t,None,20121212,1200,0,0)
GribField(2t,None,20121212,1800,0,0)
GribField(2t,None,20121213,600,0,0)
GribField(2t,None,20121213,1200,0,0)
GribField(2t,None,20121213,1800,0,0)
```

This list of fields can be filtered to extract on the fields corresponding to 12h time with `.sel(time=1200)`:

```
>>> import climetlab as cml
>>> ds = cml.load_source(
        "cds",
        "reanalysis-era5-single-levels",
        param=["2t", "msl"],
        product_type="reanalysis",
        grid='5/5',
        date=["2012-12-12", "2012-12-13"],
        time=[600, 1200, 1800],
    )

>>> len(ds)
10
>>> subset = ds.sel(time=1200)
```

(continues on next page)

(continued from previous page)

```
>>> len(subset)
4
>>> for f in subset:
GribField(2t,None,20121212,1200,0,0)
GribField(msl,None,20121212,1200,0,0)
GribField(2t,None,20121213,1200,0,0)
GribField(msl,None,20121213,1200,0,0)
```

Or both filters can be applied simultaneously with `.sel(param="2t", time=1200)`.

```
>>> import climetlab as cml
>>> ds = cml.load_source(
        "cds",
        "reanalysis-era5-single-levels",
        param=["2t", "msl"],
        product_type="reanalysis",
        grid='5/5',
        date=["2012-12-12", "2012-12-13"],
        time=[600, 1200, 1800],
    )

>>> len(ds)
10
>>> subset = ds.sel(param="2t", time=1200)
>>> len(subset)
2
>>> for f in subset:
GribField(2t,None,20121212,1200,0,0)
GribField(2t,None,20121213,1200,0,0)
```

Filtering on multiple values is also possible by providing a list of values `.sel(param="2t", time=[600, 1200])`.

```
>>> import climetlab as cml
>>> ds = cml.load_source(
        "cds",
        "reanalysis-era5-single-levels",
        param=["2t", "msl"],
        product_type="reanalysis",
        grid='5/5',
        date=["2012-12-12", "2012-12-13"],
        time=[600, 1200, 1800],
    )

>>> len(ds)
10
>>> subset = ds.sel(param="2t", time=[600, 1200])
>>> len(subset)
4
>>> for f in subset:
GribField(2t,None,20121212,600,0,0)
GribField(2t,None,20121212,1200,0,0)
GribField(2t,None,20121213,600,0,0)
GribField(2t,None,20121213,1200,0,0)
```

1.8.5 Ordering with `.order_by()`

1.9 Merging Data sources

Warning: The merger functionality is experimental, the API may change.

Todo: add documentation on merging. merge=concat(). merge=merge().

```
import climetlab as cml
import xarray as xr

class MyMerger():
    def __init__(self, *args, **kwargs):
        pass
    def merge(self, paths, **kwargs):
        return xr.open_mfdataset(paths)

data = cml.load_source("url-pattern",
     "https://www.example.com/data-{foo}-{bar}-{qux}.csv",
     foo = [1,2,3],
     bar = ["a", "b"],
     qux = "unique"
     merger = MyMerger()
     )
```

1.10 Machine learning tools

Warning: This part of CliMetLab is still a work in progress. Documentation and code behaviour will change.

Todo: TODO: Develop and document machine learning related tools

1.10.1 to_tfdataset()

To use a CliMetLab dataset with tensorflow, use the `_to_tfdataset()` method.

```
>>> import climetlab as cml
>>> ds = cml.load_dataset(x)
>>> x = ds.to_tfdataset(options)
>>> model.fit(x, ....)
```

The discussion is still open to decide whether `to_dataset()` returns:

- `tf.keras.utils.experimental.DatasetCreator`

- `tf.data.Dataset`
- `tf.keras.utils.Sequence`
- A custom CliMetLab class

1.10.2 PyTorch support

Todo: A long-term goal of CliMetLab is to provide a easy way to use a dataset with Pytorch. A merge request within this respect would be welcome. The CliMetLab API (exposed to the end-user) should be mostly identical for Tensorflow or Pytorch.

1.11 Plotting

See also : https://confluence.ecmwf.int/display/MAGP/Reference+guide

Todo: Introduce advance plotting options

```
import climetlab as cml

# cml.settings.set("plotting-options", {'dump_yaml': True})

dataset = cml.load_dataset("example-dataset")
data = dataset[0]

cml.plot_map(data, foreground=False)

cml.plot_map(data, foreground=True)


cml.plot_map(data, foreground="example-foreground")

cml.plot_map(
    data,
    foreground=dict(
        map_grid=False,
        map_label=False,
        map_grid_frame=True,
        map_grid_frame_thickness=5,
        map_boundaries=True,
    ),
)

# Partial update of the current `foreground`
# How to do is still to be decided

# Option 1
cml.plot_map(
```

(continues on next page)

(continued from previous page)

```
    data,
    foreground={
        "+map_rivers": True,
        "+map_cities": True,
        "+map_label": True,
        "-map_boundaries": None,
    },
)

# Option 2
cml.plot_map(
    data,
    foreground={
        "set": {"map_rivers": True, "map_cities": True, "map_label": True},
        "clear": ["map_boundaries"],
    },
)
# Option 3
cml.plot_map(
    data,
    foreground={
        "+": {"map_rivers": True, "map_cities": True, "map_label": True},
        "-": ["map_boundaries"],
    },
)
# Option 4
cml.plot_map(
    data,
    update_foreground={
        "map_rivers": True,
        "map_cities": True,
        "map_label": True,
        "map_boundaries": None,
    },
)

# Option 5
cml.plot_map(
    data,
    update={
        "foreground": {
            "map_rivers": True,
            "map_cities": True,
            "map_label": True,
            "map_boundaries": None,
        },
    },
)
```

```
import climetlab as cml

cml.plotting_options(width=400, foreground=False)
```

1.11.1 Projections

file_content Traceback (most recent call last):

File "/home/docs/checkouts/readthedocs.org/user_builds/climetlab/checkouts/0.18.9/climetlab/sphinxext/module_output.py
module.execute(*args)

File "/home/docs/checkouts/readthedocs.org/user_builds/climetlab/checkouts/0.18.9/climetlab/sphinxext/file_content.py", l
print(".. code-block::", LANGUAGES[ext])

KeyError: '.YAML'

See mmap.

1.11.2 Styles

TODO

See mcont and msymb.

1.11.3 Layers

See mcoast.

TODO

1.12 Caching

Warning: This part of CliMetLab is still a work in progress. Documentation and code behaviour will change.

1.12.1 Purpose

CliMetLab caches most of the remote data access on a local cache. Running again `cml.load_dataset` or `cml.load_source` will use the cached data instead of downloading it again. When the cache is full, cached data is deleted according it cache policy (i.e. oldest data is deleted first). CliMetLab cache configuration is managed through the CliMetLab *Settings*.

Warning: The CliMetLab cache is intended to be used by a single user. Sharing cache with **multiple users is not recommended**. Downloading a local copy of data on a shared disk to have multiple users working is a different use case and should be supported through using mirrors. Feedback and feature requests are welcome.

1.12.2 Cache location

The cache location is defined by the `cache-directory` setting. Its default value depends on your system:

- `/tmp/climetlab-$USER` for Linux,
- `C:\\Users\\$USER\\AppData\\Local\\Temp\\climetlab-$USER` for Windows
- `/tmp/.../climetlab-$USER` for MacOS

The cache location can be read and modified either with shell command or within python.

Note: It is recommended to restart your Jupyter kernels after changing the cache location.

From a shell with the `climetlab` command:

```
# Find the current cache directory
$ climetlab settings cache-directory
/tmp/climetlab-$USER

# Change the value of the setting
$ climetlab settings cache-directory /big-disk/climetlab-cache

# Cache directory has been modified
$ climetlab settings cache-directory
/big-disk/climetlab-cache
```

From a python notebook or python script:

```
>>> import climetlab as cml
>>> cml.settings.get("cache-directory") # Find the current cache directory
/tmp/climetlab-$USER
>>> # Change the value of the setting
>>> cml.settings.set("cache-directory", "/big-disk/climetlab-cache")

# Python kernel restarted

>>> import climetlab as cml
>>> cml.settings.get("cache-directory") # Cache directory has been modified
/big-disk/climetlab-cache
```

More generally, the CliMetLab settings can be read, modified, reset to their default values using the `climetlab` command or from python, see the *Settings documentation*.

1.12.3 Cache limits

Maximum-cache-size The `maximum-cache-size` setting ensures that CliMetLab does not use to much disk space. Its value sets the maximum disk space used by CliMetLab cache. When CliMetLab cache disk usage goes above this limit, CliMetLab triggers its cache cleaning mechanism before downloading additional data. The value of cache-maximum-size is absolute (such as "10G", "10M", "1K").

Maximum-cache-disk-usage The `maximum-cache-disk-usage` setting ensures that CliMetLab leaves does not fill your disk. Its values sets the maximum disk usage of the filesystem containing the cache directory. When the disk space goes below this limit, CliMetLab triggers its cache cleaning mechanism before downloading additional data. The value of maximum-cache-disk-usage is relative (such as "90%" or "100%").

Warning: If your disk is filled by another application, CliMetLab will happily delete its cached data to make room for the other application as soon as it has a chance.

Note: When tweaking the cache settings, it is recommended to set the `maximum-cache-size` to a value below the user disk quota (if appliable) and `maximum-cache-disk-usage` to `None`.

1.12.4 Caching settings default values

Name	Default	Description
cache-directory	'/tmp/climetlab-${USER}-docs'	Directory of where the downloaded files are cached, with `${USER}` is the user id. See *Caching* for more information.
maximum-cache-disk-usage	'90%'	Disk usage threshold after which CliMetLab expires older cached entries (% of the full disk capacity). See *Caching* for more information.
maximum-cache-size	None	Maximum disk space used by the CliMetLab cache (ex: 100G or 2T).

Other CliMetLab settings can be found *here*.

1.13 Settings

CliMetLab is maintaining a set of global settings which control its behaviour.

The settings are saved in `~/.climetlab/settings.yaml`. They can be accessed from Python or using the `climetlab` command line.

1.13.1 Accessing settings

CliMetLab settings can be accessed using the python API:

```
import climetlab as cml

# Access one of the settings
cache_path = cml.settings.get("cache-directory")
print(cache_path)

# If this is the last line of a Notebook cell, this
```

(continues on next page)

(continued from previous page)

```
# will display a table with all the current settings
cml.settings
```

Or through the command line interface.

```
$ climetlab settings cache-directory
```

Or using the `climetlab` interactive prompt:

```
$ climetlab
(climetlab) settings cache-directory
```

1.13.2 Changing settings

Note: It is recommended to restart your Jupyter kernels after changing or resetting settings.

CliMetLab settings can be modified using the python API:

```
import climetlab as cml

# Change the location of the cache:
cml.settings.set("cache-directory", "/big-disk/climetlab-cache")

# Set some default plotting options (e.g. all maps will
# be 400 pixels wide by default):
cml.settings.set("plotting-options", width=400)
```

Or through the command line interface (CLI). Note that changing settings containing dictionary values is not possible with the CLI.

```
$ climetlab settings cache-directory /big-disk/climetlab-cache
```

Or using the `climetlab` interactive prompt:

```
$ climetlab
(climetlab) settings cache-directory /big-disk/climetlab-cache
```

1.13.3 Resetting settings

Note: It is recommended to restart your Jupyter kernels after changing or resetting settings.

CliMetLab settings can be reset using the python API:

```
import climetlab as cml

# Reset a named setting to its default value
cml.settings.reset("cache-directory")
```

(continues on next page)

(continued from previous page)

```
# Reset all settings to their default values
cml.settings.reset()
```

Or through the command line interface (CLI):

```
$ climetlab settings_reset cache-directory
$ climetlab settings_reset --all
```

Or using the `climetlab` interactive prompt:

```
$ climetlab
(climetlab) settings_reset
To wipe the cache completely, please use the --all flag. Use --help for more information.
(climetlab) settings_reset --all
(climetlab) Ctrl^D
```

1.13.4 Default values

Name	Default	Description
cache-directory	'/tmp/climetlab-docs'	Directory of where the downloaded files are cached, with `${USER}` is the user id. See *Caching* for more information.
check-out-of-date-urls	True	Perform a HTTP request to check if the remote version of a cache file has changed
dask-directories	['/home/docs/.climetlab/dask']	List of directories where to search for dask cluster definitions. See dask for more information.
datasets-catalogs-urls	['https://github.com/ecmwf-lab/climetlab-datasets/raw/main/datasets']	List of url where to search for catalogues of datasets definitions. See *Datasets* for more information.
datasets-directories	['/home/docs/.climetlab/datasets']	List of directories where to search for datasets definitions. See *Datasets* for more information.
download-out-of-date-urls	False	Re-download URLs when the remote version of a cached file as been changed
layers-directories	['/home/docs/.climetlab/layers']	List of directories where to search for layers definitions. See *Layers* for more information.
maximum-cache-disk-usage	'90%'	Disk usage threshold after which CliMetLab expires older cached entries (% of the full disk capacity). See *Caching* for more information.
maximum-cache-size	None	Maximum disk space used by the CliMetLab cache (ex: 100G or 2T).
number-of-download-threads	5	Number of threads used to download data.
plotting-options	{}	Dictionary of default plotting options. See *Plotting* for more information.
projections-directories	['/home/docs/.climetlab/projections']	List of directories where to search for projections definitions. See *Projections* for more information.

1.14 Using dask

1.14.1 Climetlab dask tools

You do not need CliMetLab to use *dask*. Dask is an independent python package often use with. to *xarray* and *pandas*. As CliMetLab seeks a strong integration to *dask*, some high-level functions may be useful to ease the interaction with dask.

Todo: This whole part about dask is EXPERIMENTAL, it will change in the future, may be removed or (very likely) moved to another package.

Start an local cluster and client

```
from climetlab.utils.dask import start
start('local')
# or $ climetlab dask start local
# or $ climetlab dask local --start
```

Start an ssh cluster and client

```
from climetlab.utils.dask import start
start('ssh')
```

Start a SLURM dask cluster and client

```
from climetlab.utils.dask import start
start('slurm')
```

Start a slurm dask cluster on HPC

This is assumes that your HPC admin set up the hpc-name-config-1.yaml file on the appropriate location.

```
from climetlab.utils.dask import start
start('hpc-name-config-1')
```

Access the dask dashboard

Todo: Not implemented yet.

Access the dask logs

Todo: Not implemented yet.

Stop the dask cluster

The dask cluster and client will usually stop automatically when the python process ends. Nevertheless, it is possible to stop dask if it has been started from climetlab.

```
from climetlab.utils.dask import stop
stop()
```

1.14.2 Advanced dask usages

Note: In this section a "dask deployement" refers to a client and a cluster. It does not refers to a Cloud deployement using Kubernetes, etc.

Create a custom dask deployement specifications

Create the yaml file $HOME/.climetlab/dask/hpc-name-config-1.yaml. Then use it with:

```
from climetlab.utils.dask import start
start('hpc-name-config-1')
```

Todo: This is EXPERIMENTAL.

Note: For HPC system admin: Adding yaml files in /opt/climetlab/dask/*.yaml will give global access to all users.

Reuse the dask client

```
from climetlab.utils.dask import start
client = start('local').client
```

Scale the dask cluster

Todo: Define what "scale" mean in this context.

```
from climetlab.utils.dask import start
deploy = start('slurm')
deploy.scale(..)
```

1.15 List of CliMetLab plugins

Note: This list is **not exhaustive**. Some plugins are not listed here because we are not aware of them or because they are for internal use only, or not ready to be shared.

If you would like a plugin to be added in this list, please submit a pull request to CliMetLab or open an issue on github.

1.15.1 Dataset plugins

Installing a *dataset* plugin, allows using an additional dataset in `cml.load_dataset`.

- climetlab-s2s-ai-challenge: Providing data for the Sub-seasonal to Seasonal (S2S) Artificial Intelligence Challenge (https://s2s-ai-challenge.github.io)

  Datasets provided: `s2s-ai-challenge-*`

- climetlab-demo-dataset: Demo plugin to illustrate to dataset plugin mechanism.

  Dataset provided: `demo-dataset`

- climetlab-tropical-cyclone-dataset: Tropical cyclones. In progress.

  Datasets provided: `tc-*`

- climetlab-maelstrom-yr: Alpha. Gridded weather data for the Nordics, designed for ML postprocessing. Part of the MAELSTROM project.

  Dataset provided: `maelstom-yr`

- climetlab-maelstrom-nogwd: Alpha. Dataset for learning non-orographic gravity wave parametrization. Part of the MAELSTROM project.

  Dataset provided: `maelstom-nogwd`

- climetlab-maelstrom-radiation: Alpha. Dataset for learning radiative heating parametrization. Part of the MAELSTROM project.

  Datasets provided: `maelstom-radiation`, `maelstom-radiation-tf`

- climetlab-maelstrom-ens10: Alpha. Dataset for testing ensemble postprocessing techniques. Part of the MAELSTROM project.

  Datasets provided: `maelstrom-ens5mini`, `maelstrom-ens10`

- climetlab-maelstrom-downscaling: Alpha. Dataset for testing downscaling techniques. Part of the MAELSTROM project.

  Dataset provided: `maelstrom-downscaling`

- climetlab-maelstrom-power-production: Alpha. Dataset for predicting wind farm power production from weather data. Part of the MAELSTROM project.

  Datasets provided: `maelstrom-power-production`, `maelstrom-weather-model-level`, `maelstrom-weather-pressure-level`, `maelstrom-weather-surface-level`, `maelstrom-constants-a-b`

Drafts Dataset Plugins

- climetlab-cems-flood: Glofas data. In progress.

  Dataset provided: `glofas`

- climetlab-eumetnet-postprocessing-benchmark:

  Eumetnet postprocessing benchmark dataset.

- *climetlab-meteonet*: Meteonet dataset developed by Météo-France.

—

1.15.2 Source plugins

Installing a *source* plugin, allows using an additional source in `cml.load_source`.

- google-drive

  Access public files in Google Drive with `cml.load_source("google-drive", file_id="...")`

- climetlab-demo-source

  Demo plugin to illustrate to source plugin mechanism.

Drafts Source Plugins

- stvl

  Access data the STVL database with `cml.load_source("stvl", ...)`

1.16 Command line tool

Installing climetlab also make available the `climetlab` utility command line interface.

1.16.1 Usage

```
$ climetlab <command> [options]
```

```
$ climetlab
(climetlab) <command> [options]
```

1.16.2 Interactive prompt

`climetlab`

Running the `climetlab` command with no argument starts the interactive prompt. Autocompletion is enabled on the interactive prompt. To exit the interactive prompt use Control+D.

```
$ climetlab
(climetlab) <command> [options]
(climetlab)
```

1.16.3 availability

Usage

```
climetlab availability [-h] [--stdout] [--yaml] [--keys KEYS [KEYS ...]] source
```

Create json availability file.

Positional arguments

```
source                File or directory for describing a dataset or source␣
↪of data
                      with GRIB data.
```

options:

-h, --help	show this help message and exit
--stdout	Output to stdout (no file).
--yaml	Output yaml format.

–keys KEYS [KEYS …] Keys to use in availability default is (param,time,date,step,levelist).

1.16.4 benchmark (experimental)

Usage

```
climetlab benchmark [-h] [--indexedurl] [--dataloading] [--nargs [NARGS ...]]␣
↪[--all]
```

Run predefined benchmarks, for CliMetLab development purposes.

options:

-h, --help	show this help message and exit
--indexedurl	Benchmark on using indexed URL (byte-range) and various servers.
--dataloading	Test loading some data.

–nargs [NARGS …] –all Run all benchmarks.

1.16.5 cache

Usage

```
climetlab cache [-h] [--json] [--all] [--path] [--sort KEY] [--reverse] [--
↪match STRING]
           [--newer DATE] [--older DATE] [--accessed] [--larger SIZE] [--
↪smaller SIZE]
```

Cache command to inspect the CliMetLab cache. The selection arguments are the same as for the `climetlab decache` deletion command.

Examples

```
climetlab cache --all
```

options:

-h, --help show this help message and exit

--json produce a JSON output

–all –path print the path of cache directory and exit –sort KEY sort output according to increasing values of KEY. –reverse reverse the order of the sort, from larger to smaller –match STRING TODO –newer DATE TODO –older DATE TODO –accessed use the date of last access instead of the creation date –larger SIZE consider only cache entries that are larger than SIZE bytes –smaller SIZE consider only cache entries that are smaller than SIZE bytes

SIZE can be expressed using suffixes such a K, M, G, etc. For example `--larger 1G` will match all cache entries larger than 1 GiB.

DATE can be expressed as absolute time like `2021-10-10T22:59:00`` or relative such as `1h` (one hour ago) or `2d` (two days ago).

The `--older` and `--newer` consider the *creation date* of cache entries, unless `--accessed` is specified. In this case the time of *last access* is used.

Example, to remove large files not accessed for one week:

```
decache --accessed --older 1w --larger 1G
```

1.16.6 check (experimental)

Usage

```
climetlab check [-h]
```

Experimental script to help debugging.

options:

-h, --help show this help message and exit

1.16.7 completion

Usage

```
climetlab completion [-h] [SHELL]
```

Enable autocompletion for the “climetlab” shell command.

Supported shells are : zsh bash.

Usage: climetlab completion

Positional arguments

```
SHELL       Shell to use for autocompletion. Must be zsh or bash.
```

options:

-h, --help show this help message and exit

1.16.8 create

Usage

```
climetlab create [-h] [--target TARGET] [--init] [--load] [--statistics] [--
↪config CONFIG]
            [--parts PARTS [PARTS ...]] [--no-write] [--cache-dir CACHE_DIR]
            [--format FORMAT] [--no-check] [--force] [--timeout TIMEOUT]
```

options:

-h, --help show this help message and exit

--target TARGET Where to store the final data. Currently only a path to a new ZARR is supported.

--init Initialise zarr.

--load Load data into zarr.

--statistics Compute statistics.

--config CONFIG Use with –init. A yaml file that describes which data to use as input and how to organise them in the target.

–parts PARTS [PARTS . . .] Use with –load. Part(s) of the data to process. –no-write Only compute statistics, do not write them. –cache-dir CACHE_DIR Use with –load. Location of cache directory for temporary

data.

--format FORMAT The format of the target storage into which to load the data (default is inferred from target path extension) only .zarr is currently supported.

--no-check Skip checks.

--force Overwrite if already exists.

--timeout TIMEOUT Stop with error (SIGALARM) after TIMEOUT seconds.

1.16.9 decache

Usage

```
climetlab decache [-h] [--all] [--match STRING] [--newer DATE] [--older DATE]␣
↪[--accessed]
             [--larger SIZE] [--smaller SIZE]
```

Cache deletion command (decache) to clean the cache from selected files. The selection arguments are the same as for the `climetlab cache` query command.

options:

-h, --help show this help message and exit

–all –match STRING TODO –newer DATE TODO –older DATE TODO –accessed use the date of last access instead of the creation date –larger SIZE consider only cache entries that are larger than SIZE bytes –smaller SIZE consider only cache entries that are smaller than SIZE bytes

SIZE can be expressed using suffixes such a K, M, G, etc. For example `--larger 1G` will match all cache entries larger than 1 GiB.

DATE can be expressed as absolute time like `2021-10-10T22:59:00`` or relative such as `1h` (one hour ago) or `2d` (two days ago).

The `--older` and `--newer` consider the *creation date* of cache entries, unless `--accessed` is specified. In this case the time of *last access* is used.

Example, to remove large files not accessed for one week:

```
decache --accessed --older 1w --larger 1G
```

1.16.10 dump_index

Usage

```
climetlab dump_index [-h] filename
```

Positional arguments

```
filename    Database filename.
```

options:

-h, --help show this help message and exit

1.16.11 export_cache

Usage

```
climetlab export_cache [-h] [--match STRING] [--newer DATE] [--older DATE] [--
↪accessed]
                       [--larger SIZE] [--smaller SIZE] [--permissions [PERMISSIONS␣
↪...]]
                       DIRECTORY
```

Copy part of the cache content to a directory.

Positional arguments

```
DIRECTORY                             Output directory where to copy the cache␣
↪content.
```

options:

-h, --help	show this help message and exit
--match STRING	TODO
--newer DATE	TODO
--older DATE	TODO
--accessed	use the date of last access instead of the creation date
--larger SIZE	consider only cache entries that are larger than SIZE bytes
--smaller SIZE	consider only cache entries that are smaller than SIZE bytes

–permissions [PERMISSIONS ...] By default, the exported data is public and writable by its owner. –permissions read-only : Make the data public and read only. –permissions disabled : Do not tweak the files or directories permissions.

SIZE can be expressed using suffixes such a K, M, G, etc. For example `--larger 1G` will match all cache entries larger than 1 GiB.

DATE can be expressed as absolute time like `2021-10-10T22:59:00`` or relative such as `1h` (one hour ago) or `2d` (two days ago).

The `--older` and `--newer` consider the *creation date* of cache entries, unless `--accessed` is specified. In this case the time of *last access* is used.

Example, to remove large files not accessed for one week:

```
decache --accessed --older 1w --larger 1G
```

1.16.12 grib_info

Usage

```
climetlab grib_info [-h] [--param PARAM] [--json] [--stdin]
```

Display information about grib parameters.

options:

-h, --help	show this help message and exit
--param PARAM	Comma separated list of parameters.
--json	Long json output format.
--stdin	Using stdin as input (for bash piping).

1.16.13 index_directory

Usage

```
climetlab index_directory [-h] [--no-follow-links] [--relative-paths] [--
↪output OUTPUT]
                      directory
```

Index a directory containing GRIB files.

Positional arguments

```
directory          Directory containing the GRIB files to index.
```

options:

-h, --help	show this help message and exit
--no-follow-links	Do not follow symlinks.
--relative-paths	Use relative paths. Default is to use relative paths, except when a custom location is provided for the index location. (when the argument –output is used, default for –relative-path is False)
--output OUTPUT	Custom location of the database file, will write absolute filenames in the database.

1.16.14 index_url

Usage

```
climetlab index_url [-h] URL
```

Create json index files for remote Grib url.

Positional arguments

```
URL         url to index
```

options:

-h, --help show this help message and exit

1.16.15 index_urls

Usage

```
climetlab index_urls [-h] [--baseurl BASEURL] URLS [URLS ...]
```

Create json index files for remote Grib urls. If the option –baseurl is provided, the given url are relative to the BASEURL. This allows creating an index for multiple gribs.

Positional arguments

```
URLS                List of urls to index.
```

options:

-h, --help show this help message and exit

--baseurl BASEURL Base url to use as a prefix to happen on each URLS to build actual urls.

1.16.16 libraries

Usage

```
climetlab libraries [-h] [--json]
```

options:

-h, --help show this help message and exit

--json produce a JSON output

1.16.17 plugins

Usage

```
climetlab plugins [-h] [--json]
```

List the available plugins

options:

-h, --help	show this help message and exit
--json	produce a JSON output

1.16.18 settings

Usage

```
climetlab settings [-h] [--json] [SETTING ...]
```

Display or change CliMetLab settings. See https://climetlab.readthedocs.io/guide/settings.html for more details.

Examples

```
climetlab settings cache-directory /big-disk/climetlab-cache
```

Positional arguments

```
SETTING
```

options:

-h, --help	show this help message and exit
--json	produce a JSON output

1.16.19 settings_reset

Usage

```
climetlab settings_reset [-h] [--all] [SETTING ...]
```

Display or change CliMetLab settings. See https://climetlab.readthedocs.io/guide/settings.html for more details.

Examples

```
climetlab settings cache-directory /big-disk/climetlab-cache
```

Positional arguments

```
SETTING
```

options:

-h, --help	show this help message and exit
--all	All settings

1.16.20 test_data

Usage

```
climetlab test_data [-h] [DIRECTORY]
```

Create a directory with data used to test climetlab.

Positional arguments

```
DIRECTORY   Shell to use for autocompletion. Must be zsh or bash.
```

options:

-h, --help	show this help message and exit

1.16.21 versions

Usage

```
climetlab versions [-h] [--json] [--all] [PACKAGE ...]
```

List the versions of important Python packages.

Positional arguments

```
PACKAGE     optional list of Python packages
```

options:

-h, --help	show this help message and exit
--json	produce a JSON output

–all

Plugin Developer Guide

1.17 Overview

"CliMetLab provides a common place to share code used in the Weather and Climate community to pre-process data, plot it, and include additional tools, especially for machine learning purposes."

Warning: CliMetLab is still a work in progress: Backward compatibility is not ensured as long a version 1.0 is not released (see *todo list*). Nevertheless, the API and functionalities provided for the dataset and source plugins are **mostly** stable and already usable.

The **plugins developer guide** part of the CliMetLab documentation describes how to create plugins (or YAML files) to add data and functionalities to CliMetLab, to make it available to the end-users.

1.17.1 Sharing code using plugins

In order to avoid rewriting the same code over and over, consider distributing it, the design of CliMetLab allows this through plugins developed by data providers and data users and other stakeholders.

CliMetLab has several types of plugins.

- *Dataset plugin*
- *Sources plugin*
- Reader plugin (draft)
- Helper plugin (draft)

- Machine learning plugin (not yet implemented)

Depending on the functionalities provided by your code, it can be integrated in CliMetLab differently either as a dataset or a source or a reader or a helper plugin (please refer to the table below.) If you are distributing or referring to a dataset, the right plugin type for you is likely to be a *dataset plugin*.

For more details, here is also a general description of the *CliMetLab plugin mechanism*.

Plugin type	Use case	End-User API
Dataset	Sharing code to access a curated dataset optionally with additional functionalities.	`climetlab.load_dataset()`
Source	Sharing code to access a new type of location where there are data.	`climetlab.load_source()`
Reader (DRAFT)	Sharing code to read data on a given format, using specific conventions, or requiring conversions. Readers will be available to the code written for the sources.	`climetlab.load_source()`
Helper (DRAFT)	Sharing code related to plotting data, enhancing data with additional functionalities.	`climetlab.plot_map()`
Machine Learning (TODO)	Sharing weather and climate specific code related to machine learning.	`climetlab.Dataset`, `climetlab.Source`

1.17.2 How else can I contribute?

See the *todo list*.

1.18 How to create a dataset plugin (pip)

From the end-user's perspective, a **Dataset** is a object created using `cml.load_dataset(name, *args)` with the appropriate name and arguments, which provides data.

From the plugin's developer perspective, a **Dataset** is a Python class that inherits from the CliMetLab class `climetlab.Dataset`. This class contains the Python code providing specific helper functions and curated access to the data. Dataset can also be defined from *yaml files* if they have no specific Python code and rely on (yet to defined) standard conventions.

CliMetLab has build-in example datasets for demo purposes. And more examples can be found in the non-exhaustive *list of CliMetLab plugins*.

Note: Naming convention

- A plugin package name (pip) should preferably start with `climetlab-` and use dashes "-".
- The Python package to import should start with `climetlab_` and must use underscores "_".
- A CliMetLab dataset defined by a plugin should start with the plugin name and must use dashes "-".

1.18.1 Blueprint

Automatic creation script

While creating the package manually from the documentation and from the example above is possible, there is also a semi-automated way to generate a pip package from a template. The generated package has a predefined dataset and is ready to be shared on Github and distributed with pip.

```
$ pip install climetlab-plugin-tools
$ climetlab plugin_create_dataset
# Answer the questions
# Only the first question (plugin name) and the latest (licence) are required.
# Other have sensible default values.
```

Here is a verbose output of running the plugin creation script.

Note: "Unknown command plugin_create_dataset" This error happens if you have not installed the package *climetlab-plugin-tools*.

```
$ climetlab plugin_create_dataset
Unknown command plugin_create_dataset. Type help for the list of known command␣
↪names.
```

Dataset names

The plugin mechanism relies on using *entry_points*. The three lines highlighted below are registering the class *climetlab_dataset_plugin.rain_observations:RainObservations* with *entry_points* to a CliMetLab dataset called `dataset-plugin-rain-observations`. The python plugin mechanism is exhaustively described in the Python reference documentation and here are more details about *how on CliMetLab uses it*.

```
setuptools.setup(
 name="climetlab-dataset-plugin",
 version="0.0.1",
 description="Example climetlab external dataset plugin",

 entry_points={"climetlab.datasets":
  ["dataset-plugin-rain-observations=climetlab_dataset_plugin.rain_observations:
↪RainObservations"]
 },

)
```

Once *entry_point* has registered the class, the end-user can use this external plugin to access it

```
import climetlab as cml
cml.load_dataset("dataset-plugin-rain-observations")
```

Automatic testing of the plugin

In the folder `tests` are set up automatic tests of the plugin using pytest. If the repository is hosted on github, the tests triggers automatically when pushing to the repository. Additionally, code quality is enabled using black, isort and flake.

All tests could be disabled or adapted in the `.github/workflows/` folder.

Notebooks as documentation

The folder `notebooks` in each plugin can be used to store usage example or demo on how to use the data, such as this notebook, Notebook are automatically tested if the repository is on github.

Links on the README file are pointing to binder, colab, etc. to run the automatically created notebook.

Manually creating the Python package

Here is a minimal example of pip package defining a dataset plugin : https://github.com/ecmwf/climetlab-demo-dataset

1.18.2 Adapting plugin code

Renaming a dataset

The dataset name can be changed by changing the `setup.py` file.

```
- ["dataset-plugin-rain-observations=climetlab_dataset_plugin.rain_observations:
↪RainObservations"]
+ ["dataset-plugin-new-name         =climetlab_dataset_plugin.rain_observations:
↪RainObservations"]
```

A good practice is to change keep the class name in sync with the dataset name.

Adding a dataset to a plugin

New datasets can be added to the plugin, as long as the corresponding class is created:

```
- ["dataset-plugin-rain-observations=climetlab_dataset_plugin.rain_observations:
↪RainObservations"]
+ ["dataset-plugin-rain-observations=climetlab_dataset_plugin.rain_observations:
↪RainObservations",
+  "dataset-plugin-rain-forecast    =climetlab_dataset_plugin.rain_observations:
↪RainForecast"]
```

CliMetLab hooks

Todo: Document .source attribute, to_xarray(), to_pandas(), to_etc() Point to decorator

1.19 How to create a dataset plugin (YAML)

YAML file definitions can be used for simple datasets which rely on existing built-in *data source*, and cannot be as flexible to end-users. The following example shows how to use a source when the data consists of a single file downloadable from a URL.

```
---
dataset:
  source: url
  args:
    url: http://get.ecmwf.int/test-data/metview/gallery/temp.bufr

  metadata:
    documentation: Sample BUFR file containing TEMP messages
```

Todo:

Document the YAML file way to create a dataset. Choose a good way to implement the workflow.

- Create a dataset YAML file.
- Distribute it.

1.20 How to create a source plugin

A *Data source* is a Python class that accesses data from a given location. CliMetLab has build-in sources (the most common being the "url" source) and a plugin can add more sources capabilities. A Source provides access to data, the code performing the actual reading can either be located in the Source itself or delegated to a Reader class. .. See details in Source class.

Note: **Naming convention**: A plugin package name should preferably starts with `climetlab-` and use "-". The Python package to import should starts with `climetlab_` and use "_".

1.20.1 Adding a new source as a pip plugin

The plugin mechanism for data sources relies on `entry_points`. In the `setup.py` file of the package, we should have the `entry_points` integration as follow:

```
setuptools.setup(
    entry_points={"climetlab.sources": [
        "source-name = package_name:ClassName"
        ]
    },
)
```

The package name and the class name should match the class defined in the code of the plugin:

- **source-name**: is the string that will be used in `cml.load_source("source-name", ...)` in oder to trigger the source plugin code.
- **package_name**: is the python package, as it would be used in `import package_name`
- **ClassName**: is the source class which inherits from `climetlab.Source`, it must be importable from the package_name as `from package_name import ClassName`.

1.20.2 Example

As an example, the code located at https://github.com/ecmwf/climetlab-demo-source creates build a pip package named `climetlab-demo-source`.

This data source plugin allows accessing data from a SQL database using CliMetLab.

Once the plugin is installed (with *pip* `), tabular data can be read from as follow:

```
>>> import climetlab as cml
>>> s = cml.load_source(
        "climetlab-demo-source",
        "sqlite:///test.db",
        "select * from data;",
        parse_dates=["time"],
    )
    df = s.to_pandas()
```

The integration is performed by `entry_points` called from `setup.py`.

```
setuptools.setup(
    entry_points={"climetlab.sources": [
        "demo-source = climetlab_demo_source:DemoSource"
        ]
    },
)
```

1.21 Normalize decorator

This section discusses *the purpose* of the *@normalize* decorator, shows *how to use it* and provides the *reference documentation*.

1.21.1 Purpose (discussion)

When sharing code, a large amount of code is usually dedicated to processing the arguments of the functions or methods to check their value and normalize it to a standard format.

The Python language offers the ability to accept a large range of input type on a unique function through *duck typing*. This leads to better integration of the different objects at stake, for instance using an object such as `xarray.Dataset` or a `pandas.DataFrame` or `pandas.Serie` as input to provide a list of dates.

CliMetLab offers predefined shortcuts to implement this. The short API aims to address 80% of the use cases using the `@normalize` decorator. The longer form aims to tackle specific needs.

Compare the following codes snippets:

Boilerplate code

Tedious and error-prone Python code is needed to check and normalize the values of the function arguments given by the user.

```
def __init__(self, date, option):
    if date is None:
      date = DEFAULT_DATE_LIST
    if isinstance(date, tuple):
       date = list(date)
    if not isinstance(date, list):
       date = [date]
    for d in date:
        check_date_is_sunday(d)
    if not option in VALID_OPTIONS:
        raise ValueError(f"option={option} invalid")
    (...)
    (more checks and transformations)
    (...)
    do_stuff(date, option)
```

Using CliMetLab short form API

The decorator *@normalize* provides generic default behaviour to handle domain-specific arguments (for dates, meteorological and climate parameters, bounding boxes, etc.)

```
from climetlab.decorators import normalize
@normalize("date","date(%Y%m%d)")
@normalize("option",["foo", "bar"])
def __init__(self, date, option):
    do_suff(date, option)
```

1.21.2 How to use

- How to ensure that the value in the function belongs to a list?

```
from climetlab.decorators import normalize


@normalize("param", ["tp", "gh"])
def f(self, param):
    print(param)
```

- How to ensure that the value in the function is a date with this format "YYYY-MM-DD"?

```
from climetlab.decorators import normalize


@normalize("option", "date(%Y-%m-%d)")
def f(option):
    return option


assert f(option="2022-12-31") == "2022-12-31"
assert f(option="20221231") == "2022-12-31"
assert f(option=20221231) == "2022-12-31"
```

- How to ensure that the value in the function is a list?

  Add the keyword argument `multiple=True`. Not available for `bounding-box`.

- How to ensure that the value in the function is a list of int?

```
from climetlab.decorators import normalize


@normalize("option", "int", multiple=True)
def f(option):
    return option


# Alternative shorter version
@normalize("option", "int-list")
def g(option):
    return option


assert f(option="2022") == [2022]
assert g(option="2022") == [2022]
assert f(option=[48, 72.0, "96"]) == [48, 72, 96]
assert g(option=[48, 72.0, "96"]) == [48, 72, 96]
```

- How to ensure that the value in the function is not a list?

  Add the keyword argument `multiple=False`.

- How to accept list or non-list as input?

  Add the keyword argument `multiple=None`. Not available for `bounding-box`.

- How to add alias/shortcuts/special values to be replaced by actual predefined values?

  Use the keyword argument `alias` and provide a dictionary.

```
from climetlab.decorators import normalize


@normalize("param", ["tp", "gh"])
def f(param):
    return param


assert f(param="tp") == "tp"
# f(param="t2m") # fails
```

```
from climetlab.decorators import normalize

DATES = dict(
    april=["20210401", "20210402", "20210403"],
    june=["20210610", "20210611"],
)


@normalize("x", "date-list(%Y%m%d)", aliases=DATES)
def f(x):
    return x


assert f("2021-06-10") == ["20210610"]
assert f("june") == ["20210610", "20210611"]
assert f("1999-01-01") == ["19990101"]
```

1.21.3 Reference

Warning: This API is experimental, things may change.

`@normalize(name, values, aliases={}, multiple=None, **kwargs)`

The `@normalize` decorator transforms the arguments provided when calling the decorated function, modifies it if needed, and provides a normalised value to the function. It ensures that the value of the argument is what is expected to be processed by the function.

values If *values* is a list, the list provides allowed values for the parameter. If *values* is a string, it is expected to be a shortcut similar to *"type(options)"* where *type* is one of the following: `"date"`, `"date-list"`, `"bounding-box"`. These shorts cut aims at providing an easy way to define many options in a more concise manner.

Example: `"date-list(%Y%m%d)"`

type Type of value expected by the function. The type should be one of the following: `"str"`, `"int"`, `"float"`, `"date"`, `"date-list"`, `"str-list"`, `"int-list"`, `"float-list"`.

format The keyword argument *format* is available for *type* ='date' and 'date-list'. It provides the expected format according to *datetime.strftime*. Example: format='%Y%m%d'

convention Experimental. To be documented.

aliases Replace a value with another using a dictionary of aliases.

multiple The keyword argument *multiple* is not available for `bounding-box`.

True: Ensure a list value. Turn input into a list if needed.

False: Ensure a non-list value. Turn a list input as non-list if the list has only one element. Fails with ValueError if the list has more than one element.

None: Accept list and non-list values without transformations.

1.22 Availability decorator

Todo: Not implemented yet

1.22.1 Purpose

When sharing code, a large amount of code is usually dedicated to processing the arguments of the functions or methods where users are requesting data. This boilerplate code needs to check their values and ensure that data is actually available. In case of failure, it would be helpful to generate an appropriate error message to the user, in order to help them understand why the call to the function was not successful.

CliMetLab offers predefined shortcuts to implement this. The short API aims to address the most common use cases using the `@availability` decorator.

Example:

```
@availability("availability.json")
def func(date, option):
    do_stuff(date, option)
```

1.22.2 Usage

1.22.3 Reference

1.23 Alias argument decorator

This section discusses the purpose of the *@alias_argument* decorator, shows how to use it and provides reference documentation.

1.23.1 Purpose

This decorator allows a function to accept a range of parameters names, to make the interface easier to use for the end-user.

```
from climetlab.decorators import alias_argument


@alias_argument(param="parameter")
def func(param, other):
    return "param=" + param


func(param="tp", other=1)
# -> param=tp

func(parameter="tp", other=1)
# -> param=tp
```

The decorator also accept a list of alias values.

```
from climetlab.decorators import alias_argument


@alias_argument(param=["parameter", "variable"])
def func(param, other=1):
    return "param=" + param


func(param="tp")
# -> param=tp

func(parameter="tp")
# -> param=tp

func(variable="tp")
# -> param=tp
```

1.24 GRIB support

CliMetLab provides built-in functionalities regarging GRIB format handling.

1.24.1 Reading GRIB files

- CliMetLab can read *gridded* GRIB files (or urls) and provide data as `xarray.Dataset` objects. This can be performed using `cml.load_source("file", "myfile.grib")`.
- In addition to reading GRIB from local or remote sources, CliMetLab can also use a precomputed index, avoiding the need to parse the GRIB file to know its content. Using this index allows partial read of local files, and merging of mutliple GRIB sources.

This can be performed using `cml.load_source("indexed-directory", "my/dir")`. To allow fast access to the data in the directory, CliMetLab relies on an index. Note that the index must have been created on this directory, CliMetLab will create one (see GRIB indexing below).

1.24.2 Writing GRIB files

There are two ways to write GRIB files:

- To save data from MARS, CDS or other, when GRIB is already the native format of the data,

use the `source.save(filename)` method. This method is implemented only on a sources relying on GRIB.

- CliMetLab also supports writing custom GRIB files, with **modified values or custom attributes**

through the function `` `cml.new_grib_output()`. See usage example in the example notebook (*Examples*).

1.24.3 Building indexes

CliMetLab can create GRIB index files through its command line interface.

How to build a index for a directory containing GRIB files ?

```
climetlab index_directory my/dir
```

This will create a CliMetLab index file in *my/local/dir*, allowing other to access the data with `cml.load_source("indexed-directory", "my/dir")`.

How to build a index for one given URL containing a GRIB file ?

```
climetlab index_url "https://get.ecmwf.int/repository/test-data/climetlab/test-data/
↪input/indexed-urls/large_grib_1.grb" > large_grib_1.index
```

Then upload the file *large_grib_1.index* and make sure it is available at: "https://get.ecmwf.int/repository/test-data/climetlab/test-data/input/indexed-urls/large_grib_1.index"

This allows accessing the data with

```
cml.load_source("indexed-url",
                "https://get.ecmwf.int/repository/test-data/climetlab/test-data/input/
↪indexed-urls/large_grib_1.grb"
)
```

How to build indexes for a set of URLs containing GRIB files ?

Or, if you do the same for another URL "https://get.ecmwf.int/repository/test-data/climetlab/test-data/input/indexed-urls/large_grib_2.grb".

```
climetlab index_url "https://get.ecmwf.int/repository/test-data/climetlab/test-data/
↪input/indexed-urls/large_grib_1.grb" > large_grib_1.index
climetlab index_url "https://get.ecmwf.int/repository/test-data/climetlab/test-data/
↪input/indexed-urls/large_grib_2.grb" > large_grib_2.index
```

Then upload the files *large_grib_1.index* and *large_grib_2.index* and make sure they are available on the server at: "https://get.ecmwf.int/repository/test-data/climetlab/test-data/input/indexed-urls/large_grib_1.index" "https://get.ecmwf.int/repository/test-data/climetlab/test-data/input/indexed-urls/large_grib_2.index"

This allows accessing the data with

```
cml.load_source("indexed-urls",
                "https://get.ecmwf.int/repository/test-data/climetlab/test-data/input/
↪indexed-urls/large_grib_{n}.grb",
                {"n": [1, 2]},
)
```

How to build a index for a set of URLs containing GRIB files ?

Todo: Not implemented yet.

```
climetlab index_urls --base-url "https://get.ecmwf.int/repository/test-data/climetlab/
↪test-data/input/indexed-urls" large_grib_1.grb large_grib_2.grb > global_index.index
```

Then upload the file *global_index.index* and make sure it is available at: "https://get.ecmwf.int/repository/test-data/climetlab/test-data/input/indexed-urls/global_index.index"

This allows others to access the data with :

1.24.4 How to export files from the CliMetLab cache to another directory ?

When using CliMetLab to access MARS, CDS or other source, data is cached into the CliMetLab cache directory (the cache folder is `climetlab settings cache-directory`).

To prevent the cache from growing forever, old data in the cache directory are deleted automatically by CliMetLab when new data is downloaded. CliMetLab can create a shareable directory with some of the data from the cache through its command line interface.

```
climetlab export_cache DIRECTORY --help
```

Todo: Update this when mirror implementation changes.

Warning: This part of CliMetLab is still a work in progress. Documentation and code behaviour will change.

1.25 Plotting

Todo: Explain how to contribute layers, projections and styles

CliMetLab Developer Guide

1.26 Overview

The is the **CliMetLab** developer guide part of the CliMetLab documentation.

The CliMetLab documentation is split as follow:

- *Getting started*: General introduction with the main idea described there.
- *User guide*: This is the part you should read if you are using CliMetLab and plugins developed by others.
- *Plugin Developer guide*: describes how to create plugins (or YAML files) to add data and functionalities to CliMetLab, to make it available to the users above. In order to avoid rewriting the same code over and over, consider distributing it, the design of CliMetLab allows to do this with plugins.
- **CliMetLab Developers guide**: Please refer to this part either if you are willing to develop further CliMetLab or if you want to achieve something that is not possible with the current plugin framework.

1.27 Help wanted

CliMetLab is still experimental and its development status is still Alpha. Here are the parts that would need work and/or benefit from a community contribution.

Backward compatibility is not ensured as long a version 1.0 is not released. Nevertheless the functionalities are expected to be usable and stable, please submit a issue if needed (pull request welcome).

1.27.1 How can I to contribute ?

- Submit bug reports, propose enhancements, on github.
- You can also contribute to the core code by forking and submitting a pull request.
- TODO: fill this todo list.

1.28 Architecture

Message dispatch

With:

1.29 Plotting

1.29.1 mcoast

This action controls the plotting of coastlines, rivers, cities and country boundaries, as well as the latitude/longitude grid lines.

Name	Type	Default
map_coastline_general_style Use a predefined style depending on the general theme	str	""
map_coastline Plot coastlines on map (ON/OFF)	bool	True
map_grid Plot grid lines on map (On/OFF)	bool	True
map_label Plot label on map grid lines (On/OFF)	bool	True
map_coastline_resolution Select one of the pre-defined resolutions: automatic, low, medium, and high. When set to AUTOMATIC, a resolution appropriate to the scale of the map is chosen in order to balance quality with speed.	"automatic", "low", "medium", "high"	"automatic"
map_coastline_land_shade Sets if land areas are shaded	bool	False
map_coastline_land_shade_colour Colour of Shading of land areas	str	"green"

continues on next page

Table 1 – continued from previous page

Name	Type	Default
map_coastline_sea_shade Shade the sea areas	bool	False
map_coastline_sea_shade_colour Colour of Shading of sea areas	str	“blue”
map_boundaries Add the political boundaries	bool	False
map_cities Add the cities (capitals)	bool	False
map_rivers Display rivers (on/off)	bool	False
map_rivers_style Line style for rivers	“solid”, “dash”, “dot”, “chain_dash”, “chain_dot”	“solid”
map_rivers_colour Colour of the rivers	str	“blue”
map_rivers_thickness Line thickness of rivers	int	1
map_user_layer Display user shape file layer	bool	False
map_user_layer_name Path + name of the shape file to use	str	“”
map_user_layer_projection Projection used in the shape file	str	“”

continues on next page

Table 1 – continued from previous page

Name	Type	Default
map_user_layer_style Line style for User Layer	“solid”, “dash”, “dot”, “chain_dash”, “chain_dot”	“solid”
map_user_layer_colour Colour of the User Layer	str	“blue”
map_user_layer_thickness Line thickness of User Layer	int	1
map_coastline_colour Colour of coastlines	str	“black”
map_coastline_style Line style of coastlines	“solid”, “dash”, “dot”, “chain_dash”, “chain_dot”	“solid”
map_coastline_thickness Line thickness of coastlines	int	1
map_grid_latitude_reference Reference Latitude from which all latitude lines are drawn	float	0.0
map_grid_latitude_increment Interval between latitude grid lines	float	10.0
map_grid_longitude_reference Reference Longitude from which all longitude lines are drawn	float	0.0
map_grid_longitude_increment Interval between longitude grid lines	float	20.0

continues on next page

Table 1 – continued from previous page

Name	Type	Default
map_grid_line_style Line style of map grid lines	“solid”, “dash”, “dot”, “chain_dash”, “chain_dot”	“solid”
map_grid_thickness Thickness of map grid lines	int	1
map_grid_colour Colour of map grid lines	str	“black”
map_grid_frame Add a frame around the projection	bool	False
map_grid_frame_line_style Line style of map grid lines	“solid”, “dash”, “dot”, “chain_dash”, “chain_dot”	“solid”
map_grid_frame_thickness Thickness of map grid lines	int	1
map_grid_frame_colour Colour of map grid lines	str	“black”
map_label_font Font of grid labels	str	“sansserif”
map_label_font_style Font of grid labels	str	“normal”
map_label_colour Colour of map labels	str	“black”

continues on next page

Table 1 – continued from previous page

Name	Type	Default
map_label_height Height og grid labels	float	0.25
map_label_blanking Blanking of the grid labels	bool	True
map_label_latitude_frequency Evry Nth latitue grid is labelled	float	1.0
map_label_longitude_frequency Evry Nth longitude grid is labelled	float	1.0
map_label_left Enable the labels on the left of the map	bool	True
map_label_right Enable the labels on the right of the map	bool	True
map_label_top Enable the labels on the top of the map	bool	True
map_label_bottom Enable the labels on the bottom of the map	bool	True

1.29.2 mcont

This action controls the plotting of isolines, contour bands and grid points. It is used to plot gridded data, such as fields.

Name	Type	Default
legend Turn legend on or off	bool	False

continues on next page

Table 2 – continued from previous page

Name	Type	Default
contour Turn contouring on or off	bool	True
contour_method Contouring method	“automatic”, “linear”, “akima760”, “akima474”	“automatic”
contour_interpolation_floor Any value below this floor will be forced to the floor value. avoid the bubbles artificially created by the interpolation method	float	-2147483647.0
contour_interpolation_ceiling any value above this ceiling will be forced to the ceiling value. avoid the bubbles artificially created by the interpolation method	float	2147483647.0
contour_automatic_setting Turn the automatic setting of contouring attributes	False, “style_name”, “ecmwf”	False
contour_style_name Use of a predeined setting	str	“”
contour_metadata_only Only get the metadata	bool	False
contour_hilo Plot local maxima/minima	1, 0, “hi”, “lo”	False
contour_grid_value_plot Plot Grid point values	bool	False
contour_akima_x_resolution X resolution of Akima interpolation.	float	1.5

continues on next page

Table 2 – continued from previous page

Name	Type	Default
contour_akima_y_resolution Y resolution of Akima interpolation.	float	1.5
contour_grid_value_min The minimum value for which grid point values are to be plotted	float	-1e+21
contour_grid_value_max The maximum value for which grid point values are to be plotted	float	1e+21
contour_grid_value_lat_frequency The grid point values in every Nth latitude row are plotted	int	1
contour_grid_value_lon_frequency The grid point values in every Nth longitude column are plotted	int	1
contour_grid_value_height Height of grid point values	float	0.25
contour_grid_value_colour Colour of grid point values	str	“blue”
contour_grid_value_format Format of grid point values	str	“(automatic)”
contour_grid_value_marker_height Height of grid point markers	float	0.25
contour_grid_value_marker_colour Colour of grid point markers	str	“red”
contour_grid_value_marker_qual Quality of the grid point marker	“high”, “medium”, “low”	“low”

continues on next page

Table 2 – continued from previous page

Name	Type	Default
contour_grid_value_marker_index Table number of marker index. See Appendix for Plotting Attributes	int	3
contour_grid_value_position Position of the value	“right”, “left”, “bottom”, “top”	“top”
contour_shade_max_level_colour Highest shading band colour	str	“blue”
contour_shade_min_level_colour Lowest shading band colour	str	“red”
contour_shade_colour_direction Direction of colour sequencing for shading	“clockwise”, “anti_clockwise”	“anti_clockwise”
contour_shade_cell_resolution Number of cells per cm for CELL shading	float	10.0
contour_shade_cell_method NMethod of determining the colour of a cell	“nearest”, “interpolate”	“nearest”
contour_shade_cell_resolution_method if adaptive, magics will switch to grid_shading when the data resolution is greater that the requested resolution	“classic”, “adaptive”	“classic”
contour_max_level Highest level for contours to be drawn	float	1e+21
contour_min_level Lowest level for contours to be drawn	float	-1e+21
contour_shade_max_level Highest level for contours to be shaded	float	1e+21

continues on next page

Table 2 – continued from previous page

Name	Type	Default
contour_shade_min_level Lowest level for contours to be shaded	float	-1e+21
contour_level_count Count or number of levels to be plotted. Magics will try to find “nice levels”, this means that the number of levels could be slightly different from the asked number of levels	int	10
contour_level_tolerance Tolerance: Do not use nice levels if the number of levels is really to different [count +/- tolerance]	int	2
contour_reference_level Contour level from which contour interval is calculated	float	0.0
contour_shade_dot_size Size of dot in shading pattern	float	0.02
contour_shade_max_level_density Dots/square centimetre in highest shading band	float	50.0
contour_shade_min_level_density Dots/square centimetre in lowest shading band	float	1.0
contour_gradients_colour_list Colour used at the stops : the gradeint will be calculated between 2 consecutive ones.	List[str]	[]
contour_gradients_waypoint_method waypoints at the left, right, middle of the interval.	“both”, “ignore”, “left”, “right”	“both”
contour_gradients_technique Technique to apply to compute the gradients rgb/hcl/hsl	“rgb”, “hcl”, “hsl”	“rgb”

continues on next page

Table 2 – continued from previous page

Name	Type	Default
contour_gradients_technique_direction Technique to apply to compute the gradients clockwise/anticlockwise	“clockwise”, “anti_clockwise”, “shortest”, “longest”	“clockwise”
contour_gradients_step_list Number of steps to compute for each interval	List[int]	[]
contour_shade_method Method used for shading	“area_fill”, “solid”, “dot”, “hatch”	“dot”
contour_grid_shading_position Middle : the point is in the midlle of the cell, bottom_left : the point is in the bottom left corner	“middle”, “bottom_left”	“middle”
contour_shade_hatch_index The hatching pattern(s) to use. 0 Provides an automatic sequence of patterns, other values set a constant pattern across all contour bands.	int	0
contour_shade_hatch_thickness Thickness of hatch lines	int	1
contour_shade_hatch_density Number of hatch lines per cm.	float	18.0
contour_hilo_height Height of local maxima/minima text or numbers	float	0.4
contour_hi_colour Colour of local maxima text or number	str	“blue”
contour_lo_colour Colour of local minima text or number	str	“blue”

continues on next page

Table 2 – continued from previous page

Name	Type	Default
contour_hilo_format Format of HILO numbers (MAGICS Format/(AUTOMATIC))	str	"(automatic)"
contour_hilo_marker_height Height of HighLow marker symbol	float	0.1
contour_hilo_marker_index Index of marker symbol	int	3
contour_hilo_marker_colour Colour of grid point markers	str	"red"
contour_hi_text Text to represent local maxima	str	"H"
contour_lo_text Text to represent local minima	str	"L"
contour_hilo_blanking Blank around highs and lows	bool	False
contour_hilo_type Type of high/low (TEXT/NUMBER/BOTH)	"text", "number", "both"	"text"
contour_hilo_window_size Size of the window used to calculate the Hi/Lo	int	3
contour_hilo_max_value Local HiLo above specified value are not drawn	float	1e+21
contour_hilo_min_value Local HiLo below specified value are not drawn	float	-1e+21

continues on next page

Table 2 – continued from previous page

Name	Type	Default
contour_hi_max_value Local HI above specified value are not drawn	float	1e+21
contour_hi_min_value Local HI below specified value are not drawn	float	-1e+21
contour_lo_max_value Local Lo above specified value are not drawn	float	1e+21
contour_lo_min_value Local Lo below specified value are not drawn	float	-1e+21
contour_hilo_marker Plot hilo marker (ON/OFF)	bool	False
contour_interval Interval in data units between two contour lines	float	8.0
contour_highlight_style Style of highlighting (SOLID/ DASH/ DOT/ CHAIN_DASH/ CHAIN_DOT)	“solid”, “dash”, “dot”, “chain_dash”, “chain_dot”	“solid”
contour_highlight_colour Colour of highlight line	str	“blue”
contour_highlight_thickness Thickness of highlight line	int	3
contour_highlight_frequency Frequency of highlight line	int	4
contour_label_type Type of label (text/number/both	“text”, “number”, “both”	“number”

continues on next page

Table 2 – continued from previous page

Name	Type	Default
contour_label_text Text for labels	str	“”
contour_label_height Height of contour labels	float	0.3
contour_label_format Format of contour labels (MAGICS Format/(AUTOMATIC))	str	“(automatic)”
contour_label_blanking Label Blanking	bool	True
contour_label_font Name of the font	str	“sansserif”
contour_label_font_style Style of the font normal/bold/italic	“normal”, “bold”, “italic”	“normal”
contour_label_colour Colour of contour labels	str	“contour_line_colour”
contour_label_frequency Every Nth contour line is labelled	int	2
contour_shade_technique Technique used for shading (POLYGON_SHADING/ CELL_SHADING/ MARKER)	“polygon_shading”, “grid_shading”, “cell_shading”, “dump_shading”, “marker”	“polygon_shading”
contour_shade_colour_method Method of generating the colours of the bands in contour shading (list/calculate/advanced)	“calculate”, “list”, “gradients”, “palette”	“calculate”

continues on next page

Table 2 – continued from previous page

Name	Type	Default
contour_level_list List of contour levels to be plotted	List[float]	[]
contour_shade_colour_list List of colours to be used in contour shading.	List[str]	[]
contour_shade_colour_table Colour table to be used with marker shading technique	List[str]	[]
contour_shade_height_table Height table to be used with marker shading technique	List[float]	[]
contour_shade_marker_table_type index: using contour_shade_marker_table and defining the markers by index, name: using contour_shade_marker_name_table and defining the symbols by their names	“index”, “name”	“index”
contour_shade_marker_table Marker table to be used with marker shading technique	List[int]	[]
contour_shade_marker_name_table Marker name table to be used with mareker shading technique	List[str]	[]
contour_line_style Style of contour line	“solid”, “dash”, “dot”, “chain_dash”, “chain_dot”	“solid”
contour_line_thickness Thickness of contour line	int	1
contour_line_colour_rainbow if On, rainbow colouring method will be used.	bool	False

continues on next page

Table 2 – continued from previous page

Name	Type	Default
contour_line_colour Colour of contour line	str	"blue"
contour_line_colour_rainbow_method Method of generating the colours for isoline	"calculate", "list"	"calculate"
contour_line_colour_rainbow_max_level_colour Colour to be used for the max level	str	"blue"
contour_line_colour_rainbow_min_level_colour Colour to be used for the mainlevel	str	"red"
contour_line_colour_rainbow_direction Direction of colour sequencing for colouring	"clockwise", "anti_clockwise"	"anti_clockwise"
contour_line_colour_rainbow_colour_list List of colours to be used in rainbow isolines	List[str]	[]
contour_line_colour_rainbow_colour_list_policy What to do if the list of colours is smaller that the list of contour: lastone/cycle	"lastone", "cycle"	"lastone"
contour_line_thickness_rainbow_list List of thickness to used when rainbow method is on	List[int]	[]
contour_line_thickness_rainbow_list_policy What to do if the list of thickness is smaller that the list of contour: lastone/cycle	"lastone", "cycle"	"lastone"
contour_line_style_rainbow_list List of line style to used when rainbow method is on	List[str]	[]
contour_line_style_rainbow_list_policy What to do if the list of line styles is smaller that the list of contour: lastone/cycle	"lastone", "cycle"	"lastone"

continues on next page

Table 2 – continued from previous page

Name	Type	Default
contour_highlight Plot contour highlights (ON/OFF)	bool	True
contour_level_selection_type count: calculate a reasonable contour interval taking into account the min/max and the requested number of isolines. interval: regularly spaced intervals using the reference_level as base. level_list: uses the given list of levels.	“count”, “interval”, “level_list”	“count”
contour_label Plot labels on contour lines	bool	True
contour_shade Turn shading on	bool	False
contour_legend_only Inform the contour object do generate only the legend and not the plot!	bool	False
contour_shade_palette_name Colour used at the stops : the gradient will be calculated between 2 consecutive ones.	str	“”
contour_shade_palette_policy What to do if the list of colours is smaller that the list of levels: lastone/cycle	“lastone”, “cycle”	“lastone”
contour_grid_value_type For Gaussian fields, plot normal (regular) values or reduced grid values. (NORMAL/REDUCED/akima). If akima, the akima grid values will be plotted	“normal”, “reduced”, “akima”	“normal”
contour_grid_value_plot_type (VALUE/MARKER/BOTH)	“value”, “marker”, “both”	“value”

1.29.3 mmap

Name	Type	Default
subpage_x_axis_type	“regular”, “date”, “geoline”, “logarithmic”	“regular”
subpage_y_axis_type	“regular”, “date”, “geoline”, “logarithmic”	“regular”
x_min	float	0.0
subpage_x_automatic	bool	False
subpage_y_automatic	bool	False
x_max	float	100.0
y_min	float	0.0
y_max	float	100.0
thermo_annotation_width Percentage of the width used to display the annotation on the right side .	float	25.0
subpage_x_position Y-Coordinate of lower left hand corner of subpage in cm. -1 is the default: 7.5% of the parent page	float	-1.0

continues on next page

Table 3 – continued from previous page

Name	Type	Default
subpage_y_position X-Coordinate of lower left hand corner of subpage in cm. -1 is the default: 5% of the parent page	float	-1.0
subpage_x_length Length of subpage in horizontal direction in cm. -1 is the default: 85% of the parent page	float	-1.0
subpage_y_length Length of subpage in vertical direction in cm. -1 is the default: 85% of the parent page	float	-1.0
subpage_map_library_area if On, pickup a predefined geographical area	bool	False
subpage_map_area_name Name of the predefined area	str	False

continues on next page

Table 3 – continued from previous page

Name	Type	Default
subpage_map_projection Projection to set in the drawing area	"cylindrical", "polar_stereographic", "polar_north", "polar_south", "geos", "meteosat", "meteosat_57E", "goes_east", "lambert", "EPSG:3857", "EPSG:900913", "EPSG:32661", "EPSG:32761", "EPSG:4326", "goode", "collignon", "mollweide", "robinson", "bonne", "google", "efas", "tpers", "automatic", "lambert_north_atlantic", "mercator", "cartesian", "taylor", "tephigram", "skewt", "emagram"	"cylindrical"
subpage_clipping Apply a clipping to the subpage to avoid any symbol, flag or arrow to go outside of the plotting area	bool	False
subpage_background_colour Colour of the subpage background	str	"none"
subpage_frame Plot frame around subpage (ON/OFF)	bool	True
subpage_frame_colour Colour of subpage frame (Full choice of colours)	str	"charcoal"

continues on next page

Table 3 – continued from previous page

Name	Type	Default
subpage_frame_line_style Style of subpage frame (SOLID/DASH/DOT/CHAIN_DASH/CHAIN_DOT)	“solid”, “dash”, “dot”, “chain_dash”, “chain_dot”	“solid”
subpage_frame_thickness Thickness of subpage frame	int	2
subpage_vertical_axis_width width of the vertical axis in cm	float	1.0
subpage_horizontal_axis_height height of the horizontal axis in cm	float	0.5
subpage_align_horizontal Used in automatic layout to setup the horizontal alignment of the drawing area in the subpage	“left”, “right”	“left”
subpage_align_vertical Used in automatic layout to setup the vertical alignment of the drawing area in the subpage	“bottom”, “top”	“bottom”
subpage_lower_left_latitude Latitude of lower left corner of map.	float	-90.0
subpage_lower_left_longitude Longitude of lower left corner of map	float	-180.0
subpage_upper_right_latitude Latitude of upper right corner of map	float	90.0
subpage_upper_right_longitude Longitude of upper right corner of map	float	180.0

continues on next page

Table 3 – continued from previous page

Name	Type	Default
subpage_map_area_definition_polar Method of defining a polar stereographic map	“full”, “corners”, “centre”	“corners”
subpage_map_hemisphere Hemisphere required for polar stereographic map(NORTH/SOUTH)	str	“north”
subpage_map_vertical_longitude Vertical longitude of polar stereographic or Aitoff map	float	0.0
subpage_map_centre_latitude Latitude of centre of polar stereographic map defined by ‘CENTRE’ or centre latitude of Lambert/satellite subarea projections	float	90.0
subpage_map_centre_longitude Longitude of centre of polar stereographic map defined by ‘CENTRE’ or centre longitude of Lambert/satellite subarea projections	float	0.0
subpage_map_scale Scale of polar stereographic or Aitoff map	float	50000000.0
subpage_map_area_definition method used to define the geographical area.	“corners”, “full”	“full”
subpage_map_true_scale_north Developement in progress	float	6.0
subpage_map_true_scale_south Developement in progress	float	-60.0
subpage_map_projection_height height (in meters) above the surface	float	42164000.0
subpage_map_projection_tilt angle (in degrees) away from nadir	float	0.0

continues on next page

Table 3 – continued from previous page

Name	Type	Default
subpage_map_projection_azimuth bearing (in degrees) from due north	float	20.0
subpage_map_projection_view_latitude latitude (in degrees) of the view position	float	20.0
subpage_map_projection_view_longitude longitude (in degrees) of the view position	float	-60.0
subpage_map_geos_sweep the sweep angle axis of the viewing instrument	float	0.0
taylor_standard_deviation_min Min of the Standard deviation axis.	float	0.0
taylor_standard_deviation_max Max of the Standard deviation axis.	float	1.0

1.29.4 msymb

This action controls the plotting of meteorological and marker symbols. It is used to plot point data, such as observations.

Name	Type	Default
symbol_advanced_table_selection_type Technique to use to calculate the shading band levels.	“count”, “interval”, “list”	“count”
symbol_advanced_table_min_value Min value to plot	float	-1e+21
symbol_advanced_table_max_value Max value to plot	float	1e+21

continues on next page

Table 4 – continued from previous page

Name	Type	Default
symbol_advanced_table_level_count Count or number of levels to be plotted. Magics will try to find "nice levels", this means that the number of levels could be slightly different from the requested number of levels	int	10
symbol_advanced_table_level_tolerance Tolerance: Do not use "nice levels" if the number of levels is really to different [count +/- tolerance]	int	2
symbol_advanced_table_interval Interval in data units between different bands of shading	float	8.0
symbol_advanced_table_reference_level Level from which the level interval is calculated	float	0.0
symbol_advanced_table_level_list List of shading band levels to be plotted	List[float]	[]
symbol_advanced_table_colour_method Method of generating the colours of the bands in polygon shading	"calculate", "list"	"calculate"
symbol_advanced_table_max_level_colour Highest shading band colour	str	"blue"
symbol_advanced_table_min_level_colour Lowest shading band colour	str	"red"
symbol_advanced_table_colour_direction Direction of colour sequencing for plotting (CLOCKWISE/ ANTI_CLOCKWISE)	"clockwise", "anti-clockwise"	"anti_clockwise"
symbol_advanced_table_colour_list List of colours to be used in symbol plotting	List[str]	[]

continues on next page

Table 4 – continued from previous page

Name	Type	Default
symbol_advanced_table_colour_list_policy What to do if the list of colours is smaller than the list of intervals: lastone/cycle	“lastone”, “cycle”	“lastone”
symbol_advanced_table_marker_list List of markers to be used in symbol plotting	List[int]	[]
symbol_advanced_table_marker_name_list List of markers to be used in symbol plotting symbol	List[str]	[]
symbol_advanced_table_marker_list_policy What to do if the list of markers is smaller than the list of intervals: lastone/cycle	“lastone”, “cycle”	“lastone”
symbol_advanced_table_height_method Method of generating the height	“calculate”, “list”	“list”
symbol_advanced_table_height_max_value Maximum height to use	float	0.2
symbol_advanced_table_height_min_value Mininimum height to use	float	0.1
symbol_advanced_table_height_list List of heights to be used	List[float]	[]
symbol_advanced_table_height_list_policy What to do if the list of heights is smaller than the list of intervals: lastone/cycle	“lastone”, “cycle”	“lastone”
symbol_advanced_table_text_list Text to display	List[str]	[]
symbol_advanced_table_text_list_policy What to do if the list of text is smaller that the list of intervals lastone: reuse the last one, cycle: return to the fisrt one	“lastone”, “cycle”	“cycle”

continues on next page

Table 4 – continued from previous page

Name	Type	Default
symbol_advanced_table_text_font Font to use for text plotting.	str	"sansserif"
symbol_advanced_table_text_font_size Font size	float	0.25
symbol_advanced_table_text_font_style Font Style	str	"normal"
symbol_advanced_table_text_font_colour Symbol Colour	str	"automatic"
symbol_advanced_table_text_display_type How to display text none:do not display it centre : display it instead of the symbol, right : attached it to the right of the symbol, top : attached it to the top of the symbol, bottom: attached it to the bottom of the symbol,	"centre", "none", "right", "left", "top", "bottom"	"none"
symbol_advanced_table_outlayer_method outlayer method	"none", "simple"	"none"
legend_user_text if set, the text to be shown for the symbol group in the legend	str	""
symbol_colour Colour of symbols.	str	"blue"
symbol_height Height of symbols.	float	0.2
symbol_marker_mode Method to select a marker : by name, by index, by image : in that case, Magics will use an external image as marker.	str	"index"
symbol_marker_index Marker indice: An integer between 1 and 28	int	1

continues on next page

Table 4 – continued from previous page

Name	Type	Default
symbol_marker_name Symbol name. Choose in a list of available markers dot/circle/ww_00 …	str	“dot”
symbol_image_path Path to the image	str	“”
symbol_image_format Format of the image file. If set to AUTOMATIC, the file extension will be used to determine the file type.	“automatic”, “png”, “svg”	“automatic”
symbol_image_width width of the image	float	-1.0
symbol_image_height height of the image	float	-1.0
symbol_text_list list of texts to plot	List[str]	[]
symbol_text_position Position of the text	“right”, “left”, “bottom”, “top”	“right”
symbol_text_font Font to use	str	“sansserif”
symbol_text_font_size Font size	float	0.25
symbol_text_font_style Font style	str	“normal”
symbol_text_font_colour Font colour.	str	“automatic”

continues on next page

Table 4 – continued from previous page

Name	Type	Default
symbol_legend_height If set, the height will be used to plot the symbols in the legend	float	-1.0
legend Turn legend on or off (ON/OFF) : New Parameter!	bool	False
symbol_scaling_method Turn legend on or off (ON/OFF) : New Parameter!	bool	False
symbol_scaling_level_0_height Turn legend on or off (ON/OFF) : New Parameter!	float	0.1
symbol_scaling_factor Turn legend on or off (ON/OFF) : New Parameter!	float	4.0
symbol_type Defines the type of symbol plotting required	“number”, “text”, “marker”, “wind”	“number”
symbol_table_mode Specifies if plotting is to be in advanced, table (on) or individual mode (off). Note: The simple table mode is not recommended anymore, try to use the advanced mode instead, this should give you easier control of the plot.	0, “advanced”, 1	“OFF”
symbol_format Format used to plot values (MAGICS Format/(AUTOMATIC))	str	“(automatic)”
symbol_text_blanking blanking of the text	bool	False
symbol_outline Add an outline to each symbol	bool	False
symbol_outline_colour Colour of the outline	str	“black”

continues on next page

Table 4 – continued from previous page

Name	Type	Default
symbol_outline_thickness thickness of the outline	int	1
symbol_outline_style Line Style of outline	“solid”, “dash”, “dot”, “chain_dash”, “chain_dot”	“solid”
symbol_connect_line Connect all the symbols with a line	bool	False
symbol_connect_automatic_line_colour if on, will use the colour of the symbol	bool	True
symbol_connect_line_colour Colour of the connecting line	str	“black”
symbol_connect_line_thickness thickness of the connecting line	int	1
symbol_connect_line_style Line Style of connecting line	“solid”, “dash”, “dot”, “chain_dash”, “chain_dot”	“solid”
symbol_min_table Table of minimum values. The table is used in conjunction with SYMBOL_MAX_TABLE	List[float]	[]
symbol_max_table Table of maximum values. The table is used in conjunction with SYMBOL_MIN_TABLE	List[float]	[]
symbol_marker_table Table of MARKER indices. The table is to be used in conjunction with SYMBOL_MIN_TABLE and SYMBOL_MAX_TABLE	List[int]	[]

continues on next page

Table 4 – continued from previous page

Name	Type	Default
symbol_name_table Table of Symbol names. The table is to be used in conjunction with SYMBOL_MIN_TABLE and SYMBOL_MAX_TABLE	List[str]	[]
symbol_colour_table Table of SYMBOL colours. T The table is to be used in conjunction with SYMBOL_MIN_TABLE and SYMBOL_MAX_TABLE	List[str]	[]
symbol_height_table Table of SYMBOL heights. The table is to be used in conjunction with SYMBOL_MIN_TABLE and SYMBOL_MAX_TABLE	List[float]	[]

1.29.5 mtable

Name	Type	Default
table_filename Path to the table data	str	“”
table_delimiter Used delimiter	str	“,”
table_combine_delimiters Consecutive delimiters will be considered as one	bool	False
table_header_row Which row (first is 1) is the header line on?	int	1
table_data_row_offset How many rows after the header row does the data start? 1 if no header row.	int	1
table_meta_data_rows List of row indexes containing meta-data of the form P1=V1 P2=V2.	List[int]	[]
table_x_type Type used for X variable_index: number or date	“number”, “date”	“number”
table_y_type Type used for Y variable_index: number or date	“number”, “date”	“number”
table_variable_identifier_type are we refering to the columns by names or index (index/name)	str	“index”
table_x_variable X variable_index or name	str	1
table_y_variable Y variable_index or name	str	2

1.30 Gallery

1.30.1 Layers

default-background

default-foreground

land-sea

1.30.2 Styles

cyclone-track

default-style-fields

default-style-observations

land-sea-mask

no-style

orography

rainbow-markers

1.30.3 Projections

africa

asia

bonne

collignon

euro-atlantic

europe

europe-cylindrical

global

goode

mercator

mollweide

north-america

north-america1

north-atlantic

north-hemisphere

polar-north

robinson

south-america

south-atlantic

south-hemisphere

south-pacific

tropics-east

tropics-west

web-mercator

1.31 Climetlab Plugin mechanism

This document discuss how plugins are integrated into CliMetLab. There are two ways to add a plugin into CliMetLab:

- A Python package using the standard Python plugin mechanism based on `entry_points`. This is the generic CliMetLab plugin mechanism.
- A YAML file can be also be used to create plugins, when the plugin is simple enough and used only generic predefined code. (currently only for *dataset plugins*).

1.31.1 Plugin as python packages using `entry_points`

During the installation of the pip package, the plugin registers itself thanks to the entry points in its setup.py file, making CliMetLab aware of the new capabilities. Then, the user can take advantage of the shared code though the enhanced `climetlab.load_dataset()`, `climetlab.load_source()` and `climetlab.plot_map()`, etc.

For `pip` packages using `setuptools`, creating a plugin consists in adding an entry in `setup.py`:

```
setuptools.setup(
    name = 'climetlab-package-name',
    ...
    entry_points={"climetlab.<plugintype>":
            ["foo = climetlab_package_name:FooClass",
             "bar = climetlab_package_name:BarClass"]
    },
)
```

In this package called **climetlab-package-name**, the class `climetlab_package_name.FooClass` provides Python code related to `"foo"`. Additional code related to `"bar"` is located in the class `climetlab_package_name.BarClass`. The **<plugintype>** is one of the plugin type in the table above: *dataset*, *sources*, readers, etc. See the individual documentation for each plugin type for detailed examples and the standard **`Python plugin documentation<https://packaging.python.org/guides/creating-and-discovering-plugins>`_**.

1.31.2 Plugin as YAML files

Todo: This is still a work-in-progress.

Additionally, for *dataset plugins* only, CliMetLab search for known locations to find a YAML file with a name matching the requested dataset. The YAML files are used to create an appropriate class.

CHAPTER

TWO

LICENSE

CliMetLab is available under the open source Apache License.