Flyingpigeon Documentation

Release 1.5.1

Nils Hempelmann

Aug 07, 2020

CONTENTS:

Flyingpigeon (the bird) *The pigeon finds its way home over extremely long distances. [..].* (Wikipedia).

A Web Processing Service Testbed.

Flyingpigeon is a server providing a sandbox and test environment for new Web Processing Services (OGC WPS). It is part of the birdhouse ecosystem, which aims to build interoperable services in support to substainable development goals. Once stable, mature and thoroughly tested, Flyingpigeon services are meant to move to stand-alone thematic servers used in production.

The Flyingpigeon software stack as of version 1.0 was published in Computers & Geosciences and can still be accessed from the release page. Meanwhile many of the 1.0 processes have migrated to other birdhouse repositories. For example, processes related to extreme weather event assessment can be found in blackswan, while processes focusing on climate indices will be found in finch.

CHAPTER

ONE

DOCUMENTATION

Learn more about Flyingpigeon in its official documentation at https://flyingpigeon.readthedocs.io.

Submit bug reports, questions and feature requests at https://github.com/bird-house/flyingpigeon/issues

CHAPTER

TWO

CONTRIBUTING

You can find information about contributing in our Developer Guide.

Please use bumpversion to release a new version.

CHAPTER

THREE

LICENSE

Free software: Apache Software License 2.0

CHAPTER

FOUR

CREDITS

This package was created with Cookiecutter and the bird-house/cookiecutter-birdhouse project template.

CHAPTER

FIVE

TABLE OF CONTENT

5.1 Installation

5.1.1 Install from Conda

Warning: TODO: Prepare Conda package.

5.1.2 Install from GitHub

Check out code from the Flyingpigeon GitHub repo and start the installation:

```
$ git clone https://github.com/bird-house/flyingpigeon.git
$ cd flyingpigeon
```

Create Conda environment named *flyingpigeon*:

```
$ conda env create -f environment.yml
$ source activate flyingpigeon
```

Install Flyingpigeon app:

```
$ pip install -e .
OR
make install
```

For development you can use this command:

```
$ pip install -e .[dev]
OR
$ make develop
```

5.1.3 Start Flyingpigeon PyWPS service

After successful installation you can start the service using the `flyingpigeon` command-line.

```
$ flyingpigeon --help # show help
$ flyingpigeon start  # start service with default configuration

OR

$ flyingpigeon start --daemon # start service as daemon
loading configuration
forked process id: 42
```

The deployed WPS service is by default available on:

http://localhost:8093/wps?service=WPS&version=1.0.0&request=GetCapabilities.

Note: Remember the process ID (PID) so you can stop the service with `kill PID`.

You can find which process uses a given port using the following command (here for port 5000):

```
$ netstat -nlp | grep :5000
```

Check the log files for errors:

```
$ tail -f  pywps.log
```

. . . or do it the lazy way

You can also use the `Makefile` to start and stop the service:

```
$ make start
$ make status
$ tail -f pywps.log
$ make stop
```

5.1.4 Run Flyingpigeon as Docker container

You can also run Flyingpigeon as a Docker container.

Warning: TODO: Describe Docker container support.

5.1.5 Use Ansible to deploy Flyingpigeon on your System

Use the Ansible playbook for PyWPS to deploy Flyingpigeon on your system.

5.2 Configuration

5.2.1 Command-line options

You can overwrite the default PyWPS configuration by using command-line options. See the Flyingpigeon help which options are available:

```
$ flyingpigeon start --help
--hostname HOSTNAME        hostname in PyWPS configuration.
--port PORT                port in PyWPS configuration.
```

Start service with different hostname and port:

```
$ flyingpigeon start --hostname localhost --port 5001
```

5.2.2 Use a custom configuration file

You can overwrite the default PyWPS configuration by providing your own PyWPS configuration file (just modifiy the options you want to change). Use one of the existing `sample-*.cfg` files as example and copy them to `etc/custom.cfg`.

For example change the hostname (*demo.org*) and logging level:

```
$ cd flyingpigeon
$ vim etc/custom.cfg
$ cat etc/custom.cfg
[server]
url = http://demo.org:8093/wps
outputurl = http://demo.org:8093/outputs

[logging]
level = DEBUG
```

Start the service with your custom configuration:

```
# start the service with this configuration
$ flyingpigeon start -c etc/custom.cfg
```

5.3 User Guide

- *command line*
- *Python syntax:*

placeholder for some Tutorials how to use the processes of Flyingpigeon.

5.3.1 command line

with birdy (reference to birdy)

5.3.2 Python syntax:

```
"""Python WPS execute"""

from owslib.wps import WebProcessingService, monitorExecution
from os import system
```

```
wps = WebProcessingService(url="http://localhost:8093/wps", verbose=False)
print("Service '{}' is running".format(wps.identification.title))
```

```
Service 'Flyingpigeon' is running
```

```
for process in wps.processes:
    print( '{} : \t {}'.format(process.identifier, process.abstract))
```

```
subset :     Return the data for which grid cells intersect the selected polygon for␣
↪each input dataset as well asthe time range selected.
subset_bbox :        Return the data for which grid cells intersect the bounding box␣
↪for each input dataset as well asthe time range selected.
subset_continents :          Return the data whose grid cells intersect the selected␣
↪continents for each input dataset.
subset_countries :   Return the data whose grid cells intersect the selected␣
↪countries for each input dataset.
pointinspection :    Extract the timeseries at the given coordinates.
subset_WFS :         Return the data for which grid cells intersect the selected␣
↪polygon for each input dataset.
plot_timeseries :    Outputs some timeseries of the file field means. Spaghetti and␣
↪uncertainty plot
```

```
# define some data urls


url1 = 'https://www.esrl.noaa.gov/psd/thredds/fileServer/Datasets/ncep.reanalysis.
↪dailyavgs/surface/slp.2000.nc'
url2 = 'https://www.esrl.noaa.gov/psd/thredds/fileServer/Datasets/ncep.reanalysis.
↪dailyavgs/surface/slp.2001.nc'
url3 = 'https://www.esrl.noaa.gov/psd/thredds/fileServer/Datasets/ncep.reanalysis.
↪dailyavgs/surface/slp.2002.nc'
url4 = 'https://www.esrl.noaa.gov/psd/thredds/fileServer/Datasets/ncep.reanalysis.
↪dailyavgs/surface/slp.2003.nc'
```

```
execute = wps.execute(
    identifier="plot_timeseries", #indices_clipping",
    inputs=[
        ("resource",url1),
        ("resource",url2),
        ("resource",url3),
        ("resource",url4),
        # ("variable" , "slp"),
        ])
```

(continues on next page)

(continued from previous page)

```
monitorExecution(execute, sleepSecs=5)
print(execute.getStatus())

for o in execute.processOutputs:
    print(o.reference)
```

```
 owslib.wps.WPSException : {'code': 'NoApplicableCode', 'locator': 'None', 'text':
↪'Process failed, please check server error log'}
ProcessFailed
```

```
from flyingpigeon.nc_utils import get_coordinates
```

5.4 Developer Guide

Warning: To create new processes look at examples in Emu.

5.4.1 Building the docs

First install dependencies for the documentation:

```
$ make develop
```

Run the Sphinx docs generator:

```
$ make docs
```

5.4.2 Running tests

Run tests using pytest.

First activate the `flyingpigeon` Conda environment and install `pytest`.

```
$ source activate flyingpigeon
$ pip install -r requirements_dev.txt  # if not already installed
OR
$ make develop
```

Run quick tests (skip slow and online):

```
$ pytest -m 'not slow and not online'"
```

Run all tests:

```
$ pytest
```

Check pep8:

```
$ flake8
```

5.4.3 Run tests the lazy way

Do the same as above using the `Makefile`.

```
$ make test
$ make test-all
$ make lint
```

5.4.4 Prepare a release

Update the Conda specification file to build identical environments on a specific OS.

Note: You should run this on your target OS, in our case Linux.

```
$ conda env create -f environment.yml
$ source activate flyingpigeon
$ make clean
$ make install
$ conda list -n flyingpigeon --explicit > spec-file.txt
```

5.4.5 Bump a new version

Make a new version of Flyingpigeon in the following steps:

- Make sure everything is commit to GitHub.
- Update `CHANGES.rst` with the next version.
- Dry Run: `bumpversion --dry-run --verbose --new-version 0.8.1 patch`
- Do it: `bumpversion --new-version 0.8.1 patch`
- ... or: `bumpversion --new-version 0.9.0 minor`
- Push it: `git push`
- Push tag: `git push --tags`

See the bumpversion documentation for details.

5.5 Process Descriptions

Following is a detailed description of processes in Flyingpigeon. As Flyingpigeon is currently dedicated to be the Testbed for Process development, existing processes (like published for flyingpigeon_version_v1.0) might migrate to other birds (WPS services in birdhouse) in upcoming version.

5.5.1 Migrated Processes:

Here comes a list of Processes already developed but currently not available in flyingpigeon. In case the processes were migrated you can see the WPS where you can find them:

Process group	migrated to:	brief description
Analogs of atmospheric flow	BLACKSWAN	Extreme Weather Analytics
Weather Regimes	BLACKSWAN	Extreme Weather Analytics
Climate Indices	FINCH	Climate Monitoring
Species Distribution Models	Disabled	Climate Impact
Segetal Flora	Disabled	Climate Impact

5.5.2 Subset Processes

Generates a polygon subset of input netCDF files. Based on an ocgis call, several pre-defined polygons (e.g. world countries) can be used to generate an appropriate subset of input netCDF files. Spatial subsetting are methods of deriving a new set of data from another set of data using interpolation techniques to generate different spatial or temporal resolutions.

The User can make the principal decisions to define the area or areas to be subsetted and in case of multiple areas wether they should stay in separate files or be merged to an unit.

Point-inspection can be seen as a special form of subsetting. On defined coordinates a 1D time-series will be generated for each coordinate point.

Note: See the *Subset Processes API* for detailed options and data-IO.

In case of polygon subsetting used to subset the shape of e.g. countries or continents, **flyingpigeon contains prepared shapefiles**. To increase the performance the shapefiles had been optimized with the following steps:

Shapefile preparation

This text describes how to prepare, simplify and customize shapefiles from the GADM database. We used GADM version 2.7.

Start by downloading gadm26_levels.gdb, the ESRI shapefile for the whole world that contains all six administration levels. The resulting file is a directory that can be read with qgis 2.8. (Note: for Fedora users, you must have Fedora 22 in order to upgrade to qgis 2.8).

Open shapefile in qgis

To open gadm26_levels.gdb in qgis, follow the steps below:

Add Vector Layer ->

. . . . . Source type = Directory

. . . . . Source:

. . . . . Type: OpenFileGDB

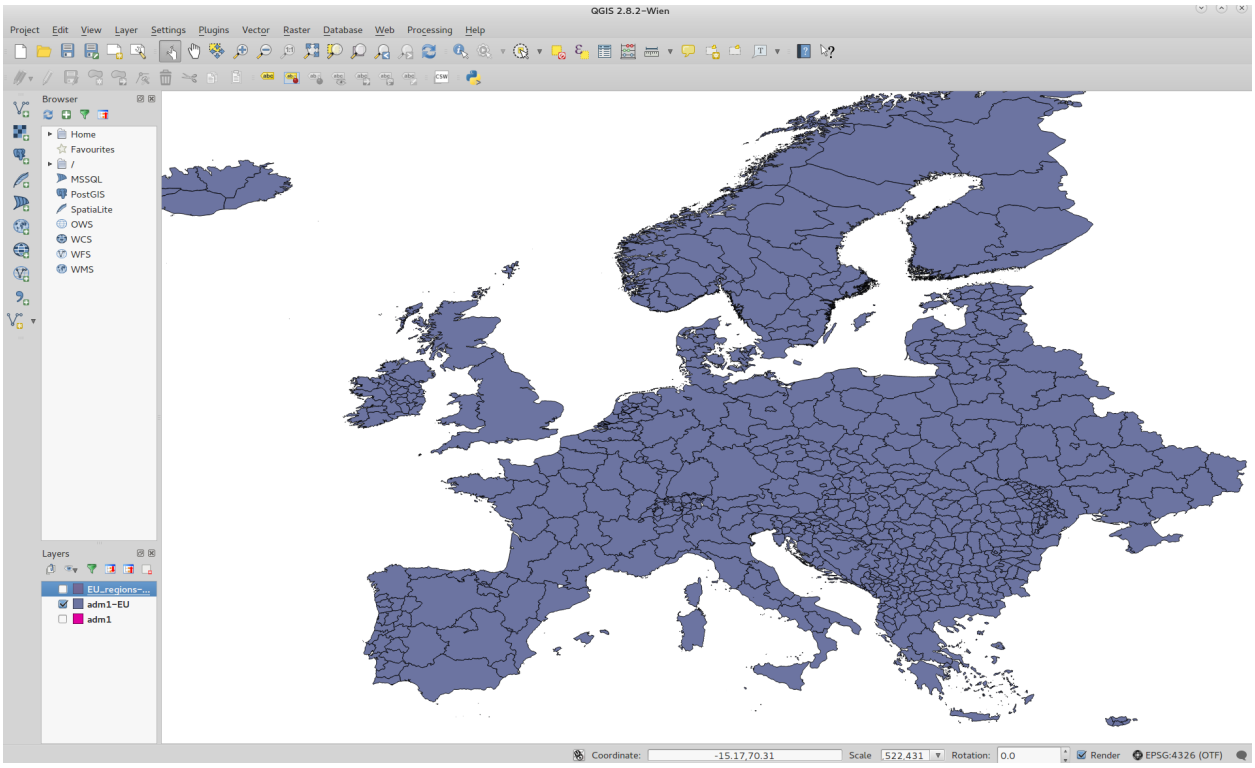
After selecting the directory, select "Open" and a window will appear listing the six levels. Select the level you would like. We used "adm1" which corresponds to the region level (i.e. one level smaller than country level).

Filter the countries you wish to retain

You can use qgis to select countries you wish to extract from the world shapefile. Use the "Advanced Filter (Expression)" option in the Attribute Table. For example, for a selection of EU countries, you can use this expression:

> « ISO » LIKE '%AUT%' OR « ISO » LIKE '%BEL%' OR « ISO » LIKE '%BGR%' OR « ISO » LIKE '%CYP%' OR « ISO » LIKE '%CZE%' OR « ISO » LIKE '%DEU%' OR « ISO » LIKE '%DNK%' OR « ISO » LIKE '%ESP%' OR « ISO » LIKE '%EST%' OR « ISO » LIKE '%FIN%' OR « ISO » LIKE '%FRA%' OR « ISO » LIKE '%GBR%' OR « ISO » LIKE '%GRC%' OR « ISO » LIKE '%HUN%' OR « ISO » LIKE '%HRV%' OR « ISO » LIKE '%IRL%' OR « ISO » LIKE '%ITA%' OR « ISO » LIKE '%LVA%' OR « ISO » LIKE '%LTU%' OR « ISO » LIKE '%LUX%' OR « ISO » LIKE '%MLT%' OR « ISO » LIKE '%NLD%' OR « ISO » LIKE '%POL%' OR « ISO » LIKE '%PRT%' OR « ISO » LIKE '%ROU%' OR « ISO » LIKE '%SVK%' OR « ISO » LIKE '%SVN%' OR « ISO » LIKE '%SWE%' OR « ISO » LIKE '%NOR%' OR « ISO » LIKE '%CHE%' OR « ISO » LIKE '%ISL%' OR « ISO » LIKE '%MKD%' OR « ISO » LIKE '%MNE%' OR « ISO » LIKE '%SRB%' OR « ISO » LIKE '%MDA%' OR « ISO » LIKE '%UKR%' OR « ISO » LIKE '%BIH%' OR « ISO » LIKE '%ALB%' OR « ISO » LIKE '%BLR%' OR « ISO » LIKE '%XKO%'

The Attribute Table will then be updated and you can choose all the rows. This selection will be displayed on the map in the main window of qgis:



You can save this selection in the format of an ESRI Shapefile:

Layer -> Save as -> Save only selected features

Simplify using mapshaper (command line)

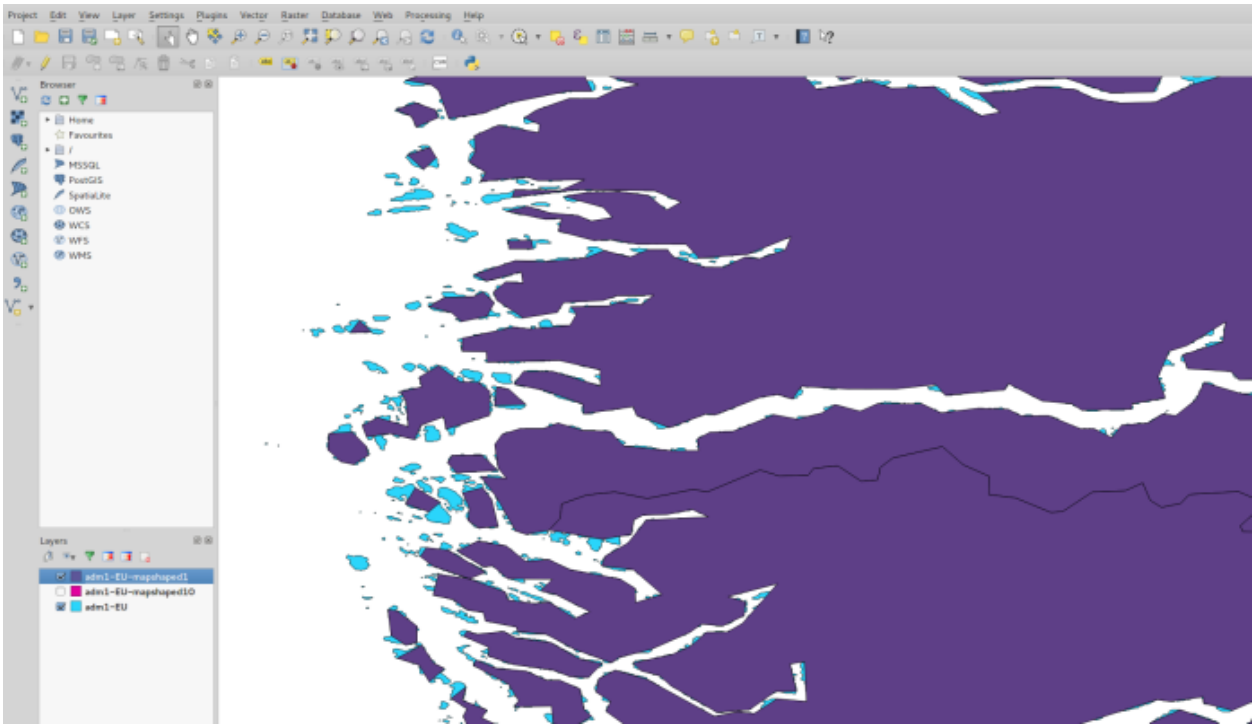
Note that the resulting map is very highly resolved, and it is often necessary to simplify the lines. There is an online tool called mapshaper that we found to be very effective. It can be used both on the command line and as a web GUI.

On the command line, the default options are: Visvalingam Weighted Area, no Snap Vertices. Choose the simplification level, the input and output shapefiles. Here is an example for 1% simplification:

$ mapshaper -i adm1-EU.shp -simplify 1% -o adm1-EU-mapshaped1.shp

=> Repaired 98 intersections; unable to repair 1 intersection.

This produced a simplified map, shown here (purple) superimposed on the original map (blue), zoomed on the coastline of Norway:



Simplify using mapshaper (GUI)

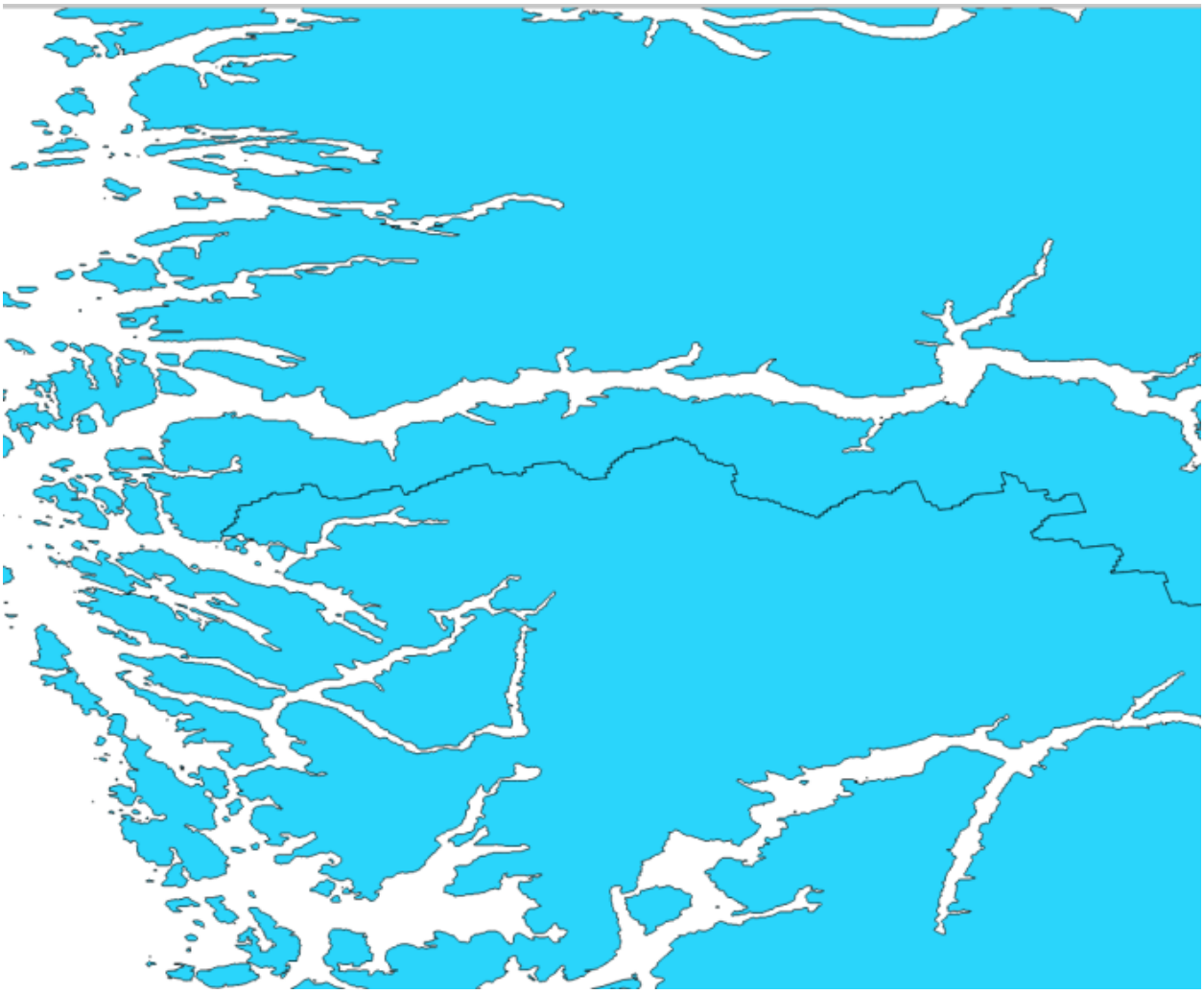
You can test the different simplify options using the mapshaper GUI instead of the command line version. Namely:

Visvalingam Weighted Area | Effective Area

Snap Vertices ON | OFF

Also, the GUI seems to be more successful at repairing all intersections.

The figure below shows the original (cyan), NoSnapVertices-WeightedArea (magenta), and NoSnapVertices-EffectiveArea (purple):



(There were very tiny differences between Snap Vertices vs. No Snap Vertices. Too hard to say from this example if one was better than the other).

Customize shapefile

The shapefile produced from the adm1 level of the ESRI shapefile as described above shows all regions of the selected countries, but when displayed on the screen, some regions were too small both visually and for the resolution of our models (~100 km):

Another issue with using the ESRI file containing all six admin levels is that there is no unique identifier column such as “HASC_1”. For example, for the region Homyel’ (with a ‘ at the end) in Belarus, “HASC_1” = “BY.HO”. Without this field, one would be forced to use “NAME_1” = “Homyel’” to identify this region, but the special character ‘ may cause problems in the python script that reads the file.

So, we need to merge the small country regions together while leaving the larger regions alone.

Steps:

1. We downloaded the GADM shapefile as a single layer (here), and the EU countries were selected as before.

   Note: HASC_1” was NULL for some countires, e.g. FRA, ITA, GBR, BEL. We manually replaced the NULLs with a unique identifier following the convention in the file.

2. We then decided which regions to merge together, and then we formed the following qgis filter expression:

   « ISO » LIKE ‘%CYP%’ OR « ISO » LIKE ‘%IRL%’ OR « ISO » LIKE ‘%MDA%’ OR « ISO » LIKE ‘%BGR%’ OR « ISO » LIKE ‘%XKO%’ OR « ISO » LIKE ‘%CHE%’ OR « ISO » LIKE ‘%LIE%’ OR « ISO » LIKE ‘%DNK%’ OR « ISO » LIKE ‘%HRV%’ OR « ISO » LIKE ‘%BIH%’ OR « ISO » LIKE ‘%SRB%’ OR « ISO » LIKE ‘%PRT%’ OR « ISO » LIKE ‘%MNE%’ OR « ISO » LIKE ‘%ALB%’ OR « ISO » LIKE

'%MKD%' OR « ISO » LIKE '%NLD%' OR « ISO » LIKE '%SVN%' OR « ISO » LIKE '%MDA%' OR « ISO » LIKE '%LUX%' OR « ISO » LIKE '%MLT%' OR « ISO » LIKE '%LIE%' OR « ISO » LIKE '%BIH%' OR « ISO » LIKE '%ROU%' OR « ISO » LIKE '%AUT%' OR « ISO » LIKE '%CZE%' OR « ISO » LIKE '%SVK%' OR « ISO » LIKE '%HUN%'

3. In the main screen of qgis, these selected countries were regrouped with respect to field "ISO" (i.e. country level):

   Vector -> Geoprocessing tools -> Dissolve -> Dissolve field = ISO

4. The other countries (whose regions are large enough to be resolved) were selected in the Attribute Table in the same way, but using ID_1 (corresponding to level adm1) as the identifier.

5. Finally, the two shapefiles were fused together:

   Vector -> Data Management Tools -> Merge shapefiles to one

6. The resulting shapefile was simplified with the mapshaper GUI at 0.1%, which can then be read into the flyingpigeon python scripts.

7. To display in the browser, the shapefile was converted to geojson using ogr2ogr:

   $ ogr2ogr -overwrite -f GeoJSON output.geojson input.shp

Here is the resulting file containing region-level and country-level areas:

5.5.3 Data Visualization:

They are various ways of data visualization. In flyingpigeon are realized the basic ones of creating an ordinary graphic file. It helps to have a quick understanding of your data.

Time series visualization of netCDF files. Creates a spaghetti plot and an uncertainty plot.

Plots are generated based on matplotlib. Appropriate functions are located in eggshell.

Note: See the *Plot Timeseries API* for detailed options and data-IO.

5.5.4 Spatial Analogues

Spatial analogues are maps showing which areas have a present-day climate that is *analogous* to the future climate of a given place. This type of map can be useful for climate adaptation to see how well regions are coping today under specific climate conditions. For example, officials from a city located in a temperate region that may be expecting more heatwaves in the future can learn from the experience of another city where heatwaves are a common occurrence, leading to more proactive intervention plans to better deal with new climate conditions.

Spatial analogues are estimated by comparing the distribution of climate indices computed at the target location over the future period with the distribution of the same climate indices computed over a reference period for multiple candidate regions. A number of methodological choices thus enter the computation:

- Climate indices of interest,
- Metrics measuring the difference between both distributions,
- Reference data from which to compute the base indices,
- A future climate scenario to compute the target indices.

The climate indices chosen to compute the spatial analogues are usually annual values of indices relevant to the intended audience of these maps. For example, in the case of the wine grape industry, the climate indices examined could include the length of the frost-free season, growing degree-days, annual winter minimum temperature andand annual number of very cold days [Roy2017].

The `flyingpigeon.processes.SpatialAnalogProcess` offers six distance metrics: standard euclidean distance, nearest neighbor, Zech-Aslan energy distance, Kolmogorov-Smirnov statistic,Friedman-Rafsky runs statistics and the Kullback-Leibler divergence. A description and reference for each distance metric is given in `flyingpigeon.dissimilarity` and based on [Grenier2013].

The reference data set should cover the target site in order to perform validation tests, and a large area around it. Global or continental scale datasets are generally used, but the spatial resolution should be high enough for users to be able to *recognize* climate features they are familiar with.

Different future climate scenarios from climate models can be used to compute the target distribution over the future period. Usually the raw model outputs are bias-corrected with the observation dataset. This is done to avoid discrepancies that would be introduced by systematic model errors. One way to validate the results is to compute the spatial analog using the simulation over the historical period. The best analog region should thus cover the target site.

The WPS process automatically extracts the target series from a netCDF file using geographical coordinates and the names of the climate indices (the name of the climate indices should be the same for both netCDF files). It also allows users to specify the period over which the distributions should be compared, for both the target and candidate datasets.

An accompanying process `flyingpigeon.processes.PlotSpatialAnalogProcess` can then be called to create a graphic displaying the dissimilarity value. An example of such graphic is shown below, with the target location indicated by a white marker.

Note: See the *Spatial Analogs API* for a description of both processes.

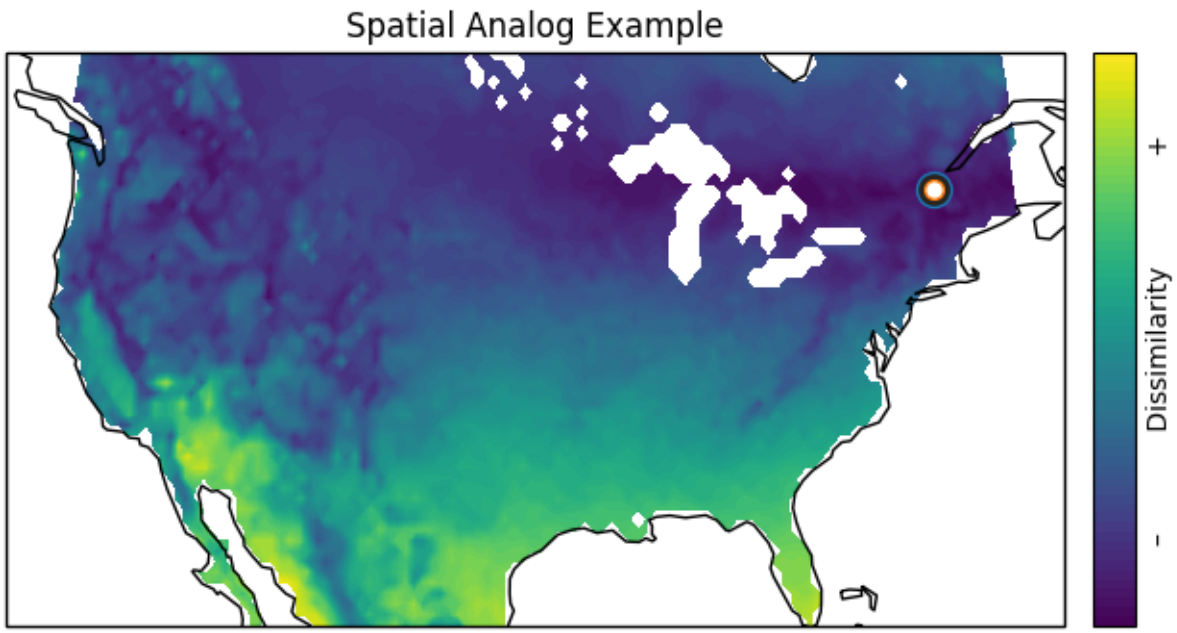


Fig. 1: A map of the dissimilarity metric computed from mean annual precipitation and temperature values in Montreal over the period 1970-1990.

References

5.6 Examples

These examples demonstrates a few features of the Flyingpigeon server.

5.6.1 Subset processes

The WPS flyingpigeon provides several processes to perform spatial subsets of netCDF files:

- **subset_bbox**: Crop netCDF files to a given latitude longitude bounding box.
- **subset_countries**: Crop netCDF files to only the intersection of predefined countries.
- **subset_continents**: Crop netCDF files to only the intersection of predefined continents.
- **subset_wfs_polygon**: Crop netCDF files to only the intersection of given polygons available on a given WFS server.

Note that subsetting operations do not average or reduce results over the selected area, they only return the grid sells intersecting with the area. A mask is applied to cells that are outside the selected area, but inside the rectangular grid required for netCDF files.

[1]:
```
# Import the WPS client and connect to the server
from birdy import WPSClient
import birdy
from os import environ

# To display Images from an url
```

(continues on next page)

(continued from previous page)

```
from IPython.core.display import HTML
from IPython.display import Image

# wait until WPS process is finished
import time
```

[2]:
```
# This cell is for server admnistration test purpose
# Ignore this cell and modify the following cell according to your needs

fp_server = environ.get('WPS_URL')
print(fp_server)  # link to the flyingpigoen server
```

```
http://localhost:8093
```

[3]:
```
# URL to a flyingpigeon server
# fp_server = 'https://pavics.ouranos.ca/twitcher/ows/proxy/flyginpigeon/wps'
```

Connect to the server with birdy client

[4]:
```
# Simple connection (not recommended for larger processing)
fp = WPSClient(fp_server)
# Asyncronous connection with progess status requests
fp_i = WPSClient(url=fp_server, progress=True)
```

Explore the available processes:

[5]:
```
# Enter `fp?` for general exploration on processes provided by flyingpigeon
# fp?

# Or get help for a process in particular
help(fp.subset_continents) # or type: fp.subset_countries?
```

```
Help on method subset_continents in module birdy.client.base:

subset_continents(resource=None, region='Africa', mosaic=False) method of birdy.
↪client.base.WPSClient instance
    Return the data whose grid cells intersect the selected continents for each input␣
↪dataset.

    Parameters
    ----------
    region : {'Africa', 'Asia', 'Australia', 'North America', 'Oceania', 'South␣
↪America', 'Antarctica', 'Europe'}string
        Continent name.
    mosaic : boolean
        If True, selected regions will be merged into a single geometry.
    resource : ComplexData:mimetype:`application/x-netcdf`, :mimetype:`application/x-
↪tar`, :mimetype:`application/zip`
        NetCDF Files or archive (tar/zip) containing netCDF files.

    Returns
    -------
    output : ComplexData:mimetype:`application/x-netcdf`
        NetCDF output for first resource file.
    metalink : ComplexData:mimetype:`application/metalink+xml; version=4.0`
        Metalink file with links to all NetCDF outputs.
```

[11]:
```
# some test data

# 'https://github.com/roocs/mini-esgf-data/raw/master/test_data/group_workspaces/
↪jasmin2/cp4cds1/data/c3s-cordex/output/EUR-11/IPSL/MOHC-HadGEM2-ES/rcp85/r1i1p1/
↪IPSL-WRF381P/v1/day/psl/v20190212/'
# 'psl_EUR-11_MOHC-HadGEM2-ES_rcp85_r1i1p1_IPSL-WRF381P_v1_day_20060101-20101231.nc'


# Specify input files
base_url = 'https://www.esrl.noaa.gov/psd/thredds/fileServer/Datasets/ncep.reanalysis.
↪dailyavgs/surface/'
urls = [base_url + f'slp.{year}.nc' for year in range(2000, 2002)]
```

Subset over a bounding box

Note that all netCDF input files will be subsetted individually.

[7]:
```
# bbox subset of one file without interaction to the server

resp = fp.subset_bbox(
            resource=urls[0], # can be also a list of files but it is recommended to␣
↪run large processes with fp_i
            lon0=20,
            lon1=70,
            lat0=10,
            lat1=50,
            start=None,      # optional to select a time periode
            end=None,
            variable=None,  # can be set if the variable is known, otherwise the␣
↪process detects the variable name
            )

# Check the output files:
bbox = resp.get()
```

[8]:
```
# bbox of multiple files

resp = fp_i.subset_bbox(
            resource=urls,
            lon0=20,
            lon1=70,
            lat0=10,
            lat1=50,
            start=None,      # optional to select a time periode
            end=None,
            variable=None,  # can be set if the variable is known, otherwise the␣
↪process detects the variable name
            )

# using fp_i you need to wait until the processing is complete!

timeout = time.time() + 60*5   # 5 minutes from now

while resp.getStatus() != 'ProcessSucceeded':
    time.sleep(1)
```

(continues on next page)

(continued from previous page)

```
    if time.time() > timeout:  # to avoid endless waiting if the process failed
        break

# Get the output files:
bbox = resp.get()
```

```
HBox(children=(IntProgress(value=0, bar_style='info', description='Processing:'),␣
↪Button(button_style='danger'...
```

[9]:
```
# Plot the test file with the flyingpigeon plot function
out_plot = fp.plot_map_timemean(resource=bbox.output)

# and display the output
Image(url= out_plot.get()[0], width=400)
```

[9]:
```
<IPython.core.display.Image object>
```

[12]:
```
# If you want to download the result files:

# Download files stored in a metalink
# requires https://github.com/metalink-dev/pymetalink
#
from metalink import download
import tempfile


path = tempfile.mkdtemp()
files = download.get(bbox.metalink, path=path, segmented=False, force=True)
len(files)
```

```
Metalink content-type detected.
Downloading to /tmp/tmp_v07l0_1/slp.2000_bbox_subset.nc.
Downloading to /tmp/tmp_v07l0_1/slp.2001_bbox_subset.nc.
```

[12]:
```
2
```

Subset over a continent

As for bounding box subsetting, all netCDF input files are subsetted individually. Here, it is possible to specifiy one or multiple continents. If `mosaic` is `True`, all continents are merged into one shape before subsetting. If `False`, each input file is subsetted over each polygon, such that the number of output files is the number of input files times the number of continents.

The `subset_countries` process works the same with polygons for countries instead of continents.

There are two outputs: * a netCDF file to have a quick test to check if the process went according to the users needs * a metalink file with the list of all output files

[13]:
```
# Run the process
resp = fp_i.subset_continents(resource=urls, region=['Europe', 'Africa'], mosaic=True)

# using fp_i you need to wait until the processing is complete!
timeout = time.time() + 60*5   # 5 minutes from now

while resp.getStatus() != 'ProcessSucceeded':
    time.sleep(1)
```

(continues on next page)

(continued from previous page)

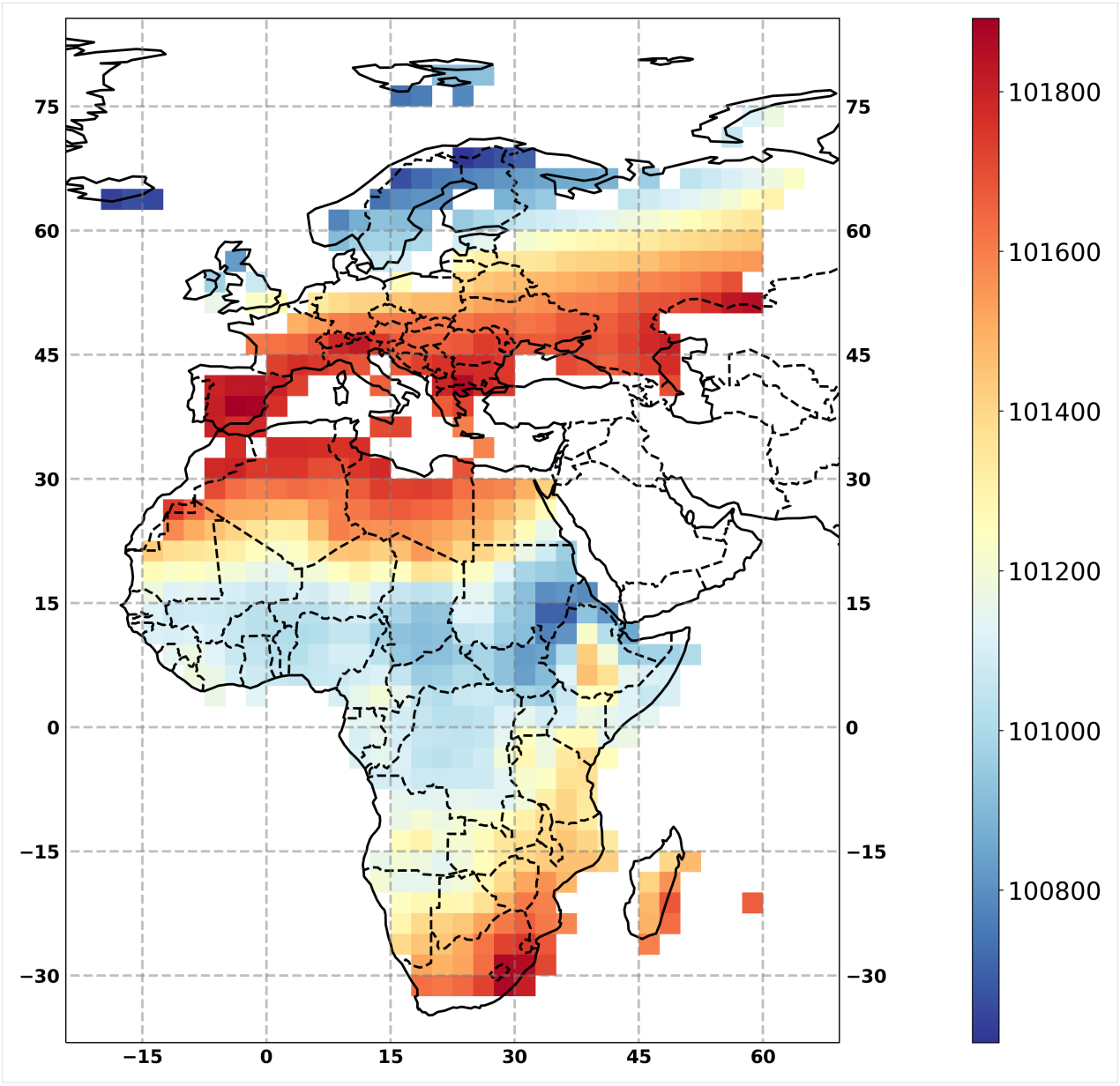
```
    if time.time() > timeout:  # to avoid endless waiting if the process failed
        break

# Check the output files:
cont = resp.get()
```

```
HBox(children=(IntProgress(value=0, bar_style='info', description='Processing:'),␣
↪Button(button_style='danger'...
```

[14]:
```
# plot the test file (url of output) with the flyingpigon plot function
resp = fp.plot_map_timemean(resource=cont.output)
```

[15]:
```
# The plot process returnes one graphic file.
# It can be plotted directly by asking birdy to get python objects, instead of links␣
↪to files.
resp.get(asobj=True).plotout_map
```

[15]:



Subset with WFS server

[16]:

```
inputs = dict(nc="https://pavics.ouranos.ca/twitcher/ows/proxy/thredds/dodsC/
↪birdhouse/nrcan"
                    "/nrcan_northamerica_monthly"
                    "/tasmax/nrcan_northamerica_monthly_2015_tasmax.nc",
                typename="public:USGS_HydroBASINS_lake_na_lev12",
                geoserver="https://pavics.ouranos.ca/geoserver/wfs",
                fid1="USGS_HydroBASINS_lake_na_lev12.67061",
                fid2="USGS_HydroBASINS_lake_na_lev12.67088",
                mosaic=False)
```

(continues on next page)

(continued from previous page)

```
# resp =  fp_i.subset_wfs_polygon(resource=inputs['nc'],
#                                 typename=inputs['typename'],
#                                 featureids=inputs['fid1'],
#                                 geoserver=inputs['geoserver'],
#                                 mosaic=False,
#                                 start=None,
#                                 end=None,
#                                 variable=None,
#                                 )
```

[17]:
```
# # using fp_i you need to wait until the processing is complete!
# timeout = time.time() + 60*5   # 5 minutes from now

# while resp.getStatus() != 'ProcessSucceeded':
#     time.sleep(1)
#     if time.time() > timeout:  # to avoid endless waiting if the process failed
#         break

# outputs = resp.get()
```

[ ]:
```
```

5.6.2 Plot processes

The WPS flyingpigeon provides several processes to perform plot subsetts of netCDF files.

They are several processes to perfom timeseries graphics as well as spatial visualisations as maps.

[1]:
```
# birdy client for communication with the server:
from birdy import WPSClient
from os import environ

# handling files and folders
from os import path, listdir
from urllib import request

# wait until WPS process is finished
import time

# to display external png graphics in notebook:
from IPython.display import Image
from IPython.core.display import HTML
```

[2]:
```
# This cell is for server admnistration test purpose
# Ignore this cell and modify the following cell according to your needs

fp_server = environ.get('FYINGPIGEON_WPS_URL')
print(fp_server)  # link to the flyingpigoen server
```

```
http://localhost:8093/wps
```

[3]:
```
# URL to a flyingpigeon server
# fp_server = 'https://pavics.ouranos.ca/twitcher/ows/proxy/flyginpigeon/wps'
```

[4]:
```
fp_i = WPSClient(fp_server, progress=True)
fp = WPSClient(fp_server)
```

read in the required files:

this is an example of an local installation, where local files are processed

indices were calculated with the birdhouse WPS finch: https://finch.readthedocs.io/en/latest/processes.html

[5]:
```
# read in the existing indices based on bias_adjusted tas files:
tas_NER = '/home/nils/data/example_data/NER/'
tasInd_NER = [ tas_NER+f for f in listdir(tas_NER) if '.nc' in f ]

tasInd_NER.sort()
```

[ ]:
```
pip install pymetalink
```

[6]:
```
# frequencies
freq=['yr','mon']

# precipitation indices
# pr_indices = ['prcptot','rx1day','wetdays','cdd','cwd','sdii','rx5day']
tas_indice = 'tg_mean'

# titles = ['Somme annuelle des précipitations',
#           'Jours de plus fortes précipitations',
#           'Nombre de jours humide',
#           'Journées consécutives de sécheresse',
#           'Jours humides consécutifs',
#           "Index d'intensité de précipitations",
#           'Somme max. sur 5 jours consécutifs',
#           'Températures moyennes annuelles']

dates = ['1976-01-01', '2005-12-31', '2036-01-01', '2065-12-31', '2071-01-01', '2099-
↪12-30']
```

[7]:
```
# find ensemble files for one indice based on pr files:

tasInd_NER = [ tas_NER+f for f in listdir(tas_NER) if '.nc' in f ]


resource = [f for f in tasInd_NER if tas_indice in f ] # and '_yr_' in f
```

Spaghetti Plot

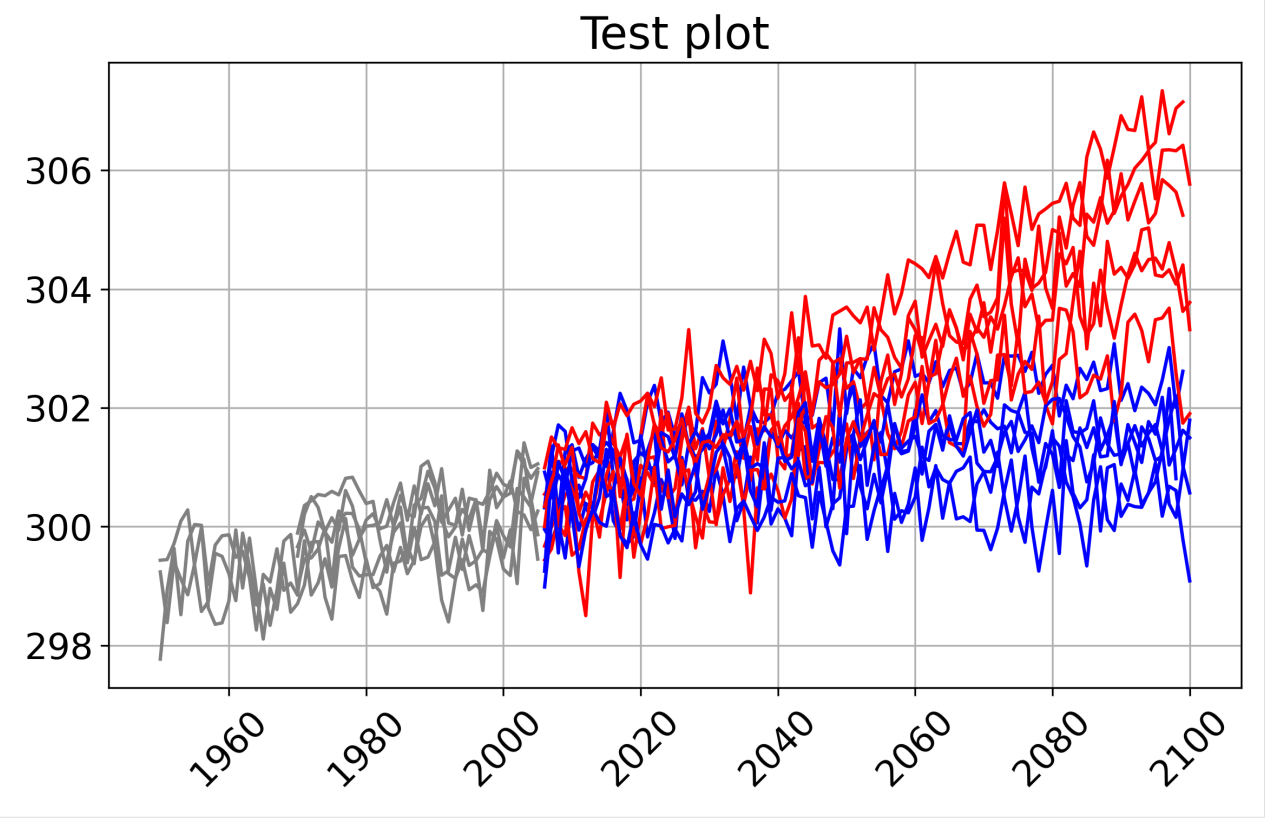
A simple way of visualisation of an ensemble of datasets. The plot visualises historical and rcp runs in different colors

[8]:
```
out_plot = fp.plot_spaghetti(resource=resource, title='Test plot',
                             delta = -273.15, # to convert K to C
                             figsize='9,5',# ymin=0, ymax=14  #
                             )
len(out_plot.get())
```

[8]:
```
1
```

[9]:
```
# display the output graphic url:
out_plot.get(asobj=True).plotout_spaghetti
# or with:
# Image(out.get()[0], width=400)
```

[9]:



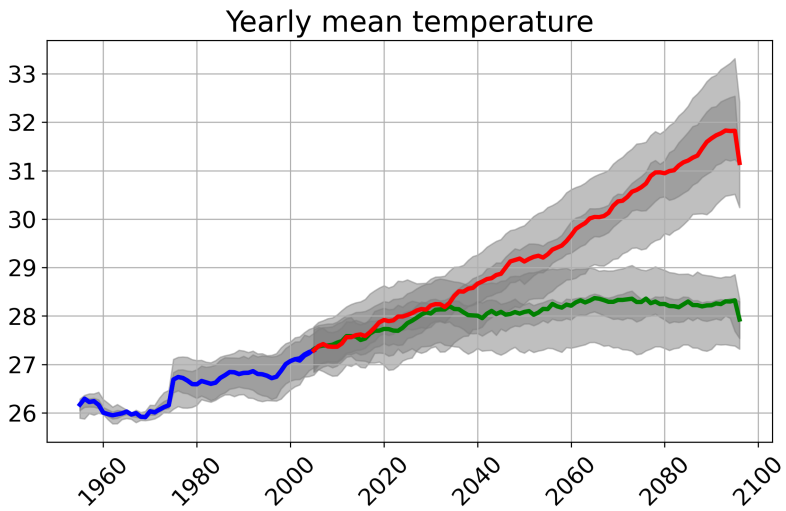
RCP uncertainty timeseries

An ensemble of indice files will be visualised as median and the appropriate uncertainties seperatled by historical and rcp runs.

[10]:
```
resp = fp.plot_uncertaintyrcp(resource=resource, title='Yearly mean temperature',␣
↪delta = -273.15,
                              figsize='9,5', # ymin=20, ymax=100  #
                             )
```

[11]:
```
# onother way to display the output graphic.

# download the file
out_file = '/tmp/ts_uncertainty_tg-yr.png'
request.urlretrieve(resp.get()[0], out_file)

# display the graphic:
Image(out_file, width=400)
```

[11]:

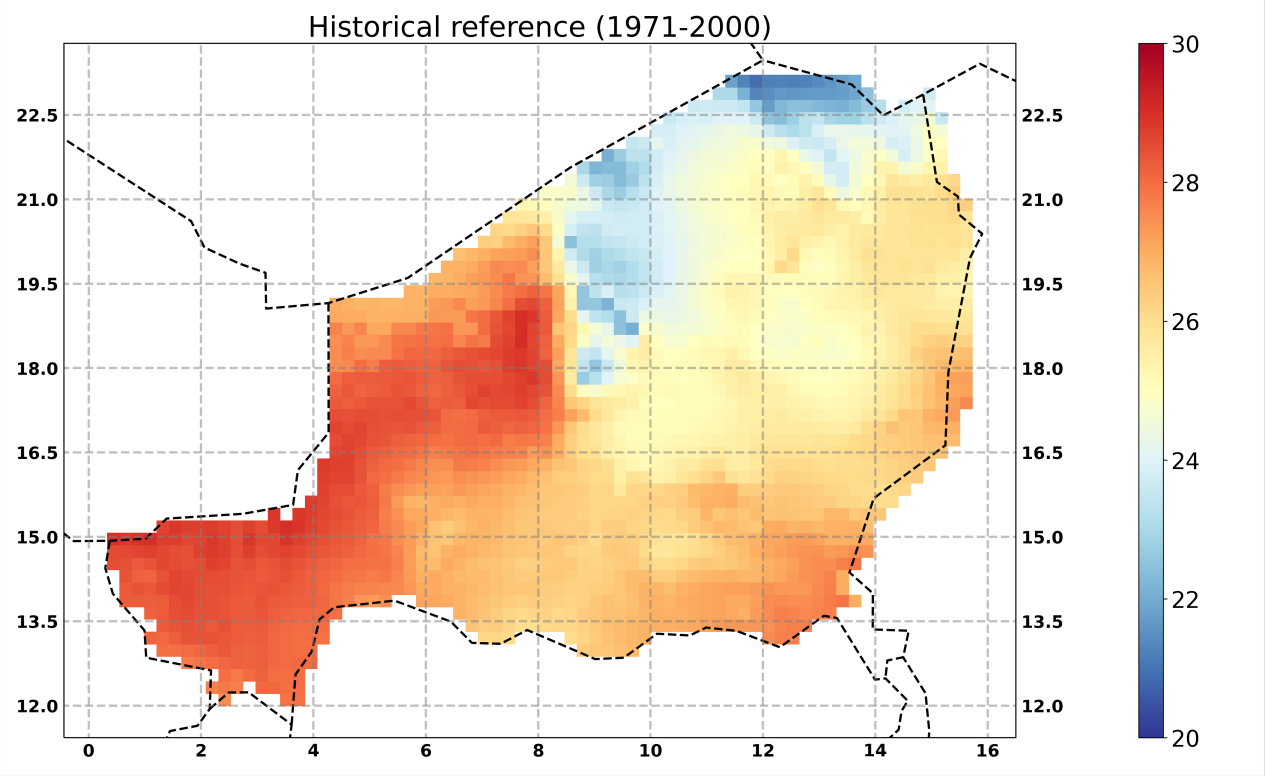


Plot a map

Spatial visualisation of data as a mean over the time.

[12]:
```
# select only the historical runs of the indices ensemble
hist = [f for f in resource if 'historical' in f]

# process the data of the years 1971-2000
out = fp.plot_map_timemean(resource=hist, title='Historical reference (1971-2000)',
                           delta = -273.15,
                            datestart='1971-01-01', dateend='2000-12-31',  # subset␣
↪a time range
                            vmin=20,  vmax=30,
#                             cmap=''
                          ) #

# display the output graphic
Image(out.get()[0], width=600)
```

[12]:



plot a climate change signal

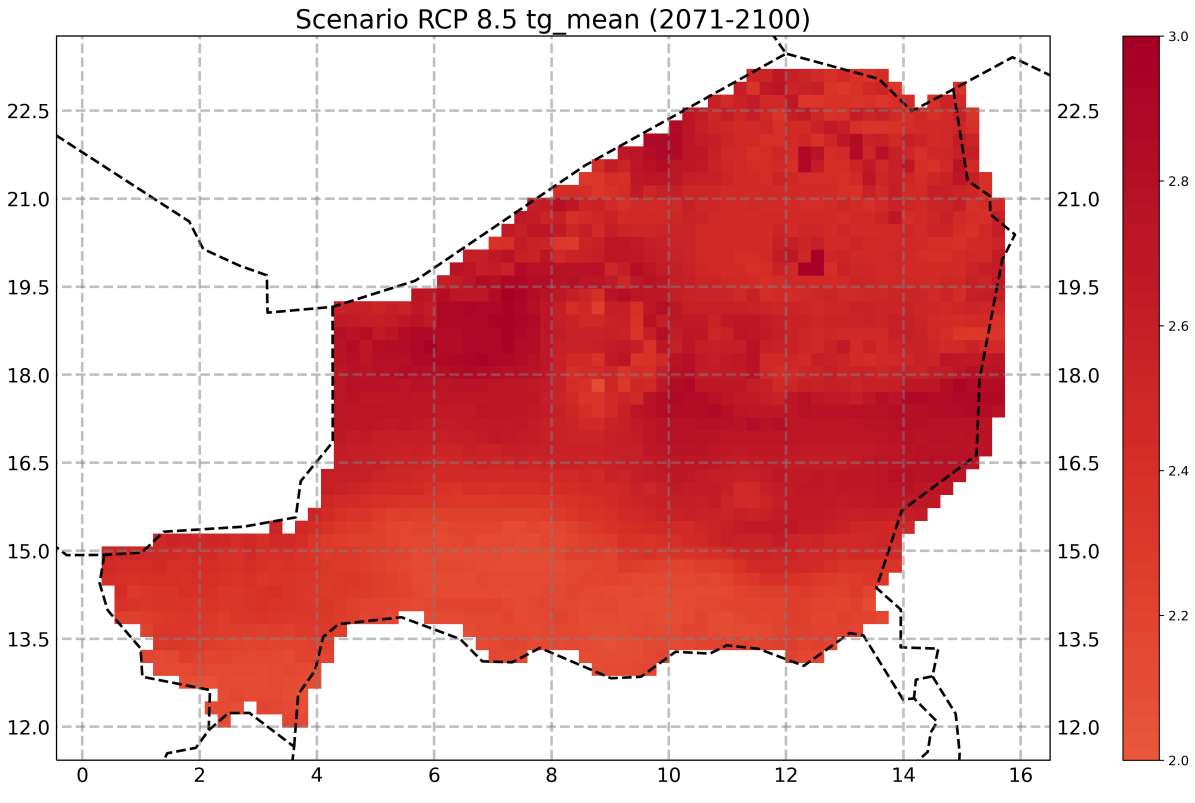
calculation of the differnence in a climate signal between a future projection to a reference periode

[13]:
```
dates = ['1971-01-01', '2000-12-31', '2036-01-01', '2065-12-31', '2071-01-01', '2100-
↪12-31']

ref = [f for f in tasInd_NER if tas_indice in f  and 'historical' in f] # and fr in f
proj = [f for f in tasInd_NER if tas_indice in f and 'rcp85' in f]

ref.sort()
proj.sort()
```

[14]:
```
resp = fp_i.climatechange_signal(resource_ref=ref,
                                 resource_proj=proj,
#                                  variable='tg-mean',
                                 title='Scenario {} {} ({})'.format('RCP 8.5', tas_
↪indice, '2071-2100'),
                                 datestart_ref=dates[0],
                                 dateend_ref=dates[1],
                                 datestart_proj=dates[2],
                                 dateend_proj=dates[3],
#                                  vmin=0 , vmax=7,
#                                  cmap='BrBG'
                                )
```

```
HBox(children=(IntProgress(value=0, bar_style='info', description='Processing:'),␣
↪Button(button_style='danger'...
```

[15]:
```
# using fp_i you need to wait until the processing is complete!

timeout = time.time() + 60*1   # 1 minute from now

while resp.getStatus() != 'ProcessSucceeded':
    time.sleep(1)
    if time.time() > timeout:  # to avoid endless waiting if the process failed
        break

Image(resp.get()[2], width=600)
```

[15]:


[ ]:

5.7 Processes API

- *Subset Processes API*
- *Plot Timeseries API*
- *Spatial Analogs API*

5.7.1 Subset Processes API

Sub-setting is performed with ocgis. Appropriate functions are located in eggshell.

class flyingpigeon.processes.wps_pointinspection.**PointinspectionProcess**
pointinspection Point Inspection (v0.10)

Extract the timeseries at the given coordinates.

Parameters

- **resource** (*application/x-netcdf*, *application/x-tar*, *application/zip*) – NetCDF files or archive (tar/zip) containing NetCDF files.
- **coords** (*string*) – Comma-seperated tuple of WGS85 lon, lat decimal coordinates (e.g. 2.356138, 48.846450).

Returns **tarout** – Tar archive containing one CSV file per input file, each one storing time series column-wise for all point coordinates.

Return type *application/x-tar*

References

- LSCE
- Doc

class flyingpigeon.processes.wps_subset_wfs_polygon.**SubsetWFSPolygonProcess**
subset-wfs-polygon Subset (v0.2)

Return the data for which grid cells intersect the selected polygon for each input dataset as well asthe time range selected.

Parameters

- **resource** (*application/x-netcdf*, *application/x-ogc-dods*) – NetCDF file or OPEnDAP url pointing to netCDF file.
- **typename** (*string, optional*) – Name of the layer in GeoServer.
- **featureids** (*string, optional*) – fid(s) of the feature in the layer.
- **geoserver** (*string, optional*) – Typically of the form http://host:port/geoserver/wfs
- **start** (*dateTime, optional*) – Initial datetime for temporal subsetting.
- **end** (*dateTime, optional*) – Final datetime for temporal subsetting.

- **variable** (*string, optional*) – Name of the variable in the NetCDF file.Will be guessed if not provided.

Returns

- **output** (*application/x-netcdf*) – NetCDF output for first resource file.
- **metalink** (*application/metalink+xml; version=4.0*) – Metalink file with links to all NetCDF outputs.

class flyingpigeon.processes.wps_subset_continents.**SubsetcontinentProcess**

subset_continents Subset Continents (v0.11)

Return the data whose grid cells intersect the selected continents for each input dataset.

Parameters

- **region** (*string*) – Continent name.
- **resource** (*application/x-netcdf*, *application/x-tar*, *application/zip*) – NetCDF Files or archive (tar/zip) containing netCDF files.

Returns

- **output** (*application/x-netcdf*) – NetCDF output for first resource file.
- **metalink** (*application/metalink+xml; version=4.0*) – Metalink file with links to all NetCDF outputs.

References

- Doc

class flyingpigeon.processes.wps_subset_countries.**SubsetcountryProcess**

subset_countries Subset Countries (v0.11)

Return the data whose grid cells intersect the selected countries for each input dataset.

Parameters

- **region** (*string*) – Country code, see ISO-3166-3: https://en.wikipedia.org/wiki/ISO_3166-1_alpha-3#Officially_assigned_code_elements
- **resource** (*application/x-netcdf*, *application/x-tar*, *application/zip*) – NetCDF Files or archive (tar/zip) containing NetCDF files.

Returns

- **output** (*application/x-netcdf*) – NetCDF output for first resource file.

- **metalink** (*application/metalink+xml; version=4.0*) – Metalink file with links to all NetCDF outputs.

References

- Doc

class flyingpigeon.processes.wps_subset_bbox.**SubsetBboxProcess**

subset_bbox Subset netCDF file on bounding box (v0.2)

Return the data for which grid cells intersect the bounding box for each input dataset as well asthe time range selected. This implies that the centroid ofa grid cell can be outside the bounding box.

Parameters

- **resource** (*application/x-netcdf*, *application/x-ogc-dods*) – NetCDF file or OPEnDAP url pointing to netCDF file.
- **lon0** (*float*) – Minimum longitude.
- **lon1** (*float*) – Maximum longitude.
- **lat0** (*float*) – Minimum latitude.
- **lat1** (*float*) – Maximum latitude.
- **start** (*dateTime, optional*) – Initial datetime for temporal subsetting.
- **end** (*dateTime, optional*) – Final datetime for temporal subsetting.
- **variable** (*string, optional*) – Name of the variable in the NetCDF file.Will be guessed if not provided.

Returns

- **output** (*application/x-netcdf*) – NetCDF output for first resource file.
- **metalink** (*application/metalink+xml; version=4.0*) – Metalink file with links to all NetCDF outputs.

References

- Doc

5.7.2 Plot Timeseries API

class flyingpigeon.processes.wps_plot_timeseries.**PlottimeseriesProcess**
plot_timeseries Graphics (timeseries) (v0.11)

Outputs some timeseries of the file field means. Spaghetti and uncertainty plot

Parameters

- **resource** (*application/x-netcdf*, *application/x-tar*, *application/zip*) – NetCDF Files or archive (tar/zip) containing NetCDF files. (Info)
- **variable** (*string, optional*) – Variable to be expected in the input files (variable will be detected if not set)

Returns

- **plotout_spagetti** (*image/png*) – Visualisation of single variables as a spaghetti plot
- **plotout_uncertainty** (*image/png*) – Visualisation of single variables ensemble mean with uncertainty

References

- Doc

5.7.3 Spatial Analogs API

class flyingpigeon.processes.wps_spatial_analog.**SpatialAnalogProcess**
spatial_analog Spatial analog of a target climate. (v0.2)

Spatial analogs based on the comparison of climate indices. The algorithm compares the distribution of the target indices with the distribution of spatially distributed candidate indices and returns a value measuring the dissimilarity between both distributions over the candidate grid.

Parameters

- **candidate** (*application/x-netcdf*, *application/x-tar*, *application/zip*) – NetCDF files or archive (tar/zip) storing the candidate indices. The output will be stored on this grid. (Info)
- **target** (*application/x-netcdf*, *application/x-tar*, *application/zip*) – NetCDF files or archive (tar/zip).containing netcdf files storing the target indices. (Info)
- **location** (*string*) – Geographical coordinates (lon,lat) of the target location.
- **indices** (*string*) – One or more climate indices to use for the comparison.
- **dist** (*{'seuclidean', 'nearest_neighbor', 'zech_aslan', 'kolmogorov_smirnov', 'friedman_rafsky', 'kldiv'}, optional*) – Dissimilarity metric comparing distributions.
- **dateStartCandidate** (*dateTime, optional*) – Beginning of period (YYYY-MM-DD) for candidate data. Defaults to first entry.

- **dateEndCandidate** (*dateTime, optional*) – End of period (YYYY-MM-DD) for candidate data. Defaults to last entry.
- **dateStartTarget** (*dateTime, optional*) – Beginning of period (YYYY-MM-DD) for target data. Defaults to first entry.
- **dateEndTarget** (*dateTime, optional*) – End of period (YYYY-MM-DD) for target data. Defaults to last entry.

Returns **output** – Dissimilarity between target at selected location and candidate distributions over the entire grid.

Return type *application/x-netcdf*

References

- Doc

class flyingpigeon.processes.wps_plot_spatial_analog.**PlotSpatialAnalogProcess**

plot_spatial_analog Map of dissimilarity values calculated by the spatial_analog process. (v0.1)

Produce map showing the dissimilarity values computed by the spatial_analog process as well as indicating by a marker the location of the target site.

Parameters

- **resource** (*application/x-netcdf*, *application/x-tar*, *application/zip*) – Dissimilarity between target at selected location and candidate distributions over the entire grid. (Info)
- **fmt** (*{'png', 'pdf', 'svg', 'ps', 'eps'}, optional*) – Figure format
- **title** (*string, optional*) – Figure title.

Returns **output_figure** – Map of dissimilarity values.

Return type *image/png*, *application/pdf*, *image/svg+xml*, *application/postscript*

References

- Doc

5.8 Tutorials

5.8.1 Subset Processes of Flyingpigeon

Subset process as ipython notebook:

https://github.com/bird-house/notebooks/blob/master/flyingpigeon_birdy_subset.ipynb

The WPS flyingpigeon provides several processes to perform spatial subsetts of netCDF files. This are:

- **bounding box:** reduces the input netCDF files to a given latitude longitude bounding box.
- **country or continent subset:** to reduce netCDF files to only the intersection of selected polygons. Countries and Continents are predifined.
- **WFS subset:** reduce netCDF files to only the intersection of given polygons available on a given WFS server.

```
# import the WPS client and connet to the server
from birdy import WPSClient
import birdy

fp_server = 'http://localhost:8093/wps'   # finch

# simple connection (not recommended for larger processing)
fp = WPSClient(fp_server)

# asyncron connection with progess status requests
fp_i = WPSClient(url=fp_server, progress=True)
```

Explore the available processes:

```
# fp? for general exploration on processes provided by flyingpigeon
fp?

# of check out a process in detail:
help(fp.subset_continents) # or type: fp.subset_countries?
```

```
Help on method subset_continents in module birdy.client.base:

subset_continents(resource=None, region='Africa', mosaic=False) method of␣
↪birdy.client.base.WPSClient instance
    Return the data whose grid cells intersect the selected continents for␣
↪each input dataset.

    Parameters
    ----------
    region : {'Africa', 'Asia', 'Australia', 'North America', 'Oceania',␣
↪'South America', 'Antarctica', 'Europe'}string
        Continent name.
    mosaic : boolean
        If True, selected regions will be merged into a single geometry.
    resource : ComplexData:mimetype:application/x-netcdf, application/x-tar,␣
↪application/zip
        NetCDF Files or archive (tar/zip) containing netCDF files.

    Returns
    -------
    output : ComplexData:mimetype:application/x-netcdf
        NetCDF output for first resource file.
    metalink : ComplexData:mimetype:application/metalink+xml; version=4.0
        Metalink file with links to all NetCDF outputs.
```

```
# point out some input files:

url1 = 'https://www.esrl.noaa.gov/psd/thredds/fileServer/Datasets/ncep.reanalysis.
↪dailyavgs/surface/slp.2000.nc'
url2 = 'https://www.esrl.noaa.gov/psd/thredds/fileServer/Datasets/ncep.reanalysis.
↪dailyavgs/surface/slp.2001.nc'
url3 = 'https://www.esrl.noaa.gov/psd/thredds/fileServer/Datasets/ncep.reanalysis.
↪dailyavgs/surface/slp.2002.nc'
url4 = 'https://www.esrl.noaa.gov/psd/thredds/fileServer/Datasets/ncep.reanalysis.
↪dailyavgs/surface/slp.2003.nc'
```

Call a continent subset process

All netCDF input files will be subsetted seperatly and depending on mosic=True or Fales the selected polygons are given as seperated files or one output file per input file including one intersection of all selected polygons.

subset_countries is working in the same principe

```
# run the process
out = fp_i.subset_continents(resource=[url1, url2, url3, url4], region=['Europe',
↪'Africa'], mosaic=True)

# You need to wait until the processing is done!
```

```
HBox(children=(IntProgress(value=0, bar_style='info', description='Processing:'),␣
↪Button(button_style='danger'...
```

There are two outputs: * a netCDF file to have a quick test to check if the process went according to the users needs * a metalink file with the list of all output files

```
# check the output files:
out.get()
```

```
subset_continentsResponse(
    output='http://127.0.0.1:8093/outputs/f232a4ba-67a0-11ea-a160-9cb6d08a53e7/slp.
↪2000_EuropeAfrica.nc',
    metalink='http://127.0.0.1:8093/outputs/f232a4ba-67a0-11ea-a160-9cb6d08a53e7/
↪input.meta4'
)
```

```
# plot the test file with the flyingpigeon plot function
out = fp_i.plot_map_timemean(resource=out.get()[0])
```

```
HBox(children=(IntProgress(value=0, bar_style='info', description='Processing:'),␣
↪Button(button_style='danger'...
```

```
# the plot process returnes one graphic file
out.get()
```

```
plot_map_timemeanResponse(
    plotout_map='http://127.0.0.1:8093/outputs/bdbf0876-67a1-11ea-9e91-9cb6d08a53e7/
↪tmp_5ahujnj.png'
)
```

```
from IPython.display import Image
from IPython.core.display import HTML
Image(url= out.get()[0], width=400)
```

5.9 Changes

5.9.1 1.6 (2020-06-10)

- remove eggshell dependency
- notebooks are part of the test suite
- improved plot processes
- remove mosaic option for subset processes
- polygon subset processes files separately instead of an entire data-set at once
- multiple outputs listed in Metalink output
- update pywps to 4.2.3
- use cruft to keep up-to-date with the cookie-cutter template

5.9.2 1.5.1 (2019-11-11)

- Add Postgres library to docker image.
- Pin PyWPS 4.2.3.

5.9.3 1.5 (2019-10-01)

- Update from cookiecutter template.
- Pin PyWPS 4.2.2.

5.9.4 1.4.2 (2019-09-18)

- Fixed the logic of the nc resource handler in subset (#288).
- Various documentation fixes.

5.9.5 1.4.1 (2019-05-20)

- Subset processes enabled (#274).
- Point-inspection process enabled (#271).
- Spatial-analog process enabled (#280).
- Fixed docs build on RTD (#279).

5.9.6 1.4.0 (2018-12-03)

New FlyingPigeon without buildout deployment (#265).

5.9.7 1.3.0 (2018-12-03)

Release with merged processes from PAVICS projects.

- Merged processes from Ouranos/PAVICS fork (#262).
- Multiple fixes.

warning: This is not a functional release due to unresolved issues and dependency conflicts. The release is kept as reference for the available functionality.

5.9.8 1.2.1 (2018-09-14)

Bug-fix release:

- disabled many processes due to conda dependency conflicts (#261).
- simplified *buildout.cfg* (#245).
- tests for *subset_countries* added (#237).
- numerous others fixes.

5.9.9 1.2.0 (2018-04-04)

Issues:

- Fixed abstract for CSV files output in pointinspection process: #216
- snappy installation is optional: #229
- Disabled sphinx buildout configuration: #227
- Fixed test failures: #210 and #224
- Fixed codacy report: #211
- Fixed readthedocs build: #207

5.9.10 1.1.0 (2017-12-22)

- disabled analogs processes (using castf90) . . . moved to black-swan.
- added new spatial analogs process.
- added initial version of satellite processes using scihub.coperniucs data.
- updated weatherregimes processes.

5.9.11 1.0.3 (2017-12-21)

- fixed sphinx build.

5.9.12 1.0.2 (2017-12-20)

- updated conda environment.
- fixed pytest configuration.
- updated travis link in Readme.

5.9.13 1.0.1 (2017-11-14)

- fixed version number
- fixed changes formatting
- display version number in service title

5.9.14 1.0.0 (2017-11-01)

- code adapted to pywps4
- ocgis v2 depoyed
- Tests for components
- Version published in Computers & Geosciences

Set of processes

Base processes:

- Fetch resources
- Fetch GBIF Species Coordination
- Subset Polygons
- Point Inspection
- Timeseries visualisation
- Climate Indices Calculation

Climate Impact:

- Species Distribution Model
- Segetal Flora Calculation

Extreme Weather Events Assessment:

- Analogs of Circulation for reanalyzes Datasets
- Analogs of Circulation for model data
- Analogs of Circulation Comparison between reanalyzes and climate model data
- Analogs output data visualisation

- Weather regime Determination for reanalyzes Datasets
- Weather regime Determination for model datasets
- Weather regime projections (based on previous analyses)

5.9.15 0.11.0 (2017-07-11)

converted processes to pywps-4 from next:

- subsetting countries, continents and european regions
- climate indices (daily percentiles, single variable)
- species distribution model
- land-sea mask
- point inspection
- fetch resources

5.9.16 0.10.1 (2017-07-11)

- disabled bbox parameter . . . needs to be fixed in OWSLib.
- updated titles of analogs processes.
- updated version in docs.
- disabled wps_gbiffetch test . . . was stalled.

5.9.17 0.10.0 (2017-07-10)

- Translate code pywps4 conform
- Climate indices dailypercentile
- Climate Fact sheet Generator
- R plot for SDM response cuvres running under CentOS
- Species distribution model Processes modularized in five processes
- Direction switch for analogs comparison process

5.9.18 0.9.1 (2016-11-16)

- modularisation of segetalflora process
- docker update

5.9.19 0.9.0 (2016-09-08)

- Subset points
- Subset European regions
- Subset world countries
- Subset continents
- Analogues for reanalyses datasets
- Analogues for model datasets
- Analogues for comparison model to reanalyses datasets
- Species Distribution Model based on GBIF CSV file
- Species Distribution Model with GBIF search included
- Weather regimes for reanalyses datasets
- Weather regimes for model datasets
- Weather regimes for model datasets with centroids trained on reanalyses datasets
- Segetalflora
- Initial spatial analogues process
- Climate indices (simple)
- Climate indices (percentile-based)
- Download resources
- Initial ensembles robustness
- Plots for time series

5.9.20 0.2.0 (2016-07-15)

- analogs detection and viewer.
- timeseries plot.
- indices calculation with icclim.
- subsetting for countries and regions.
- weather regimes.
- SDM: species distribution model for tree species based on GBIF.
- species biodiversity of segetal flora.

5.9.21 0.1.0 (2014-09-04)

Paris Release

- moved code to github.
- Initial Release.

5.10 Credits

5.10.1 Development Lead

- Nils Hempelmann <info@nilshempelmann.de>

5.10.2 Contributors

- Carsten Ehbrecht <ehbrecht@dkrz.de>
- David Huard <huard.david@ouranos.ca>
- Cathy Nangini
- Trevor James Smith

This package was created with Cookiecutter and the bird-house/cookiecutter-birdhouse project template.

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

Flying Pigeon (the bike) *Flying Pigeon is a Chinese bicycle company [..]. The Flying Pigeon is the most popular vehicle ever.* (Wikipedia)

Flying Pigeon (the bird) *The pigeon finds its way home over extremely long distances. [..].* (Wikipedia).

BIBLIOGRAPHY

[Roy2017] Roy, P., Grenier, P., Barriault, E. et al. Climatic Change (2017) 143: 43. doi:10.1007/s10584-017-1960-x

[Grenier2013] Grenier, P., A.-C. Parent, D. Huard, F. Anctil, and D. Chaumont, 2013: An assessment of six dissimilarity metrics for climate analogs. J. Appl. Meteor. Climatol., 52, 733–752, doi:10.1175/JAMC-D-12-0170.1

INDEX

P

S