pcrglobwb$_{u}tilsDocumentation$

Release 1

Jannis M. Hoch

Feb 03, 2023

CONTENTS:

This is the read-the-docs documentation of `pcrglobwb_utils`, a Python-package containing useful functions and scripts to evaluate output from PCR-GLOBWB.

Note: It is still very much in the testing phase, so no guarantees that it works on any platform and any data!

CHAPTER

ONE

OVERVIEW

Handy functions to work with PCR-GLOBWB input and output.

- Free software: GNU General Public License v3
- Documentation: https://pcrglobwb-utils.readthedocs.io.

1.1 Features

Most mature:

- evaluation of timeseries simulated by PCR-GLOBWB with observations from GRDC.
  - multiple GRDC-stations and properties can be provided via a yml-file.
  - using command line functions and wide range of user-defined options.
- command line functions to validate model output against GRACE and GLEAM data for multiple polygons.

Other:

- aggregating and averaging over time scales.
- water balance assessments of PCR-GLOBWB runs.
- statistical analyses.
- ensemble analysis.

1.2 Credits

Contributions from Jannis M. Hoch (j.m.hoch@uu.nl).

The package structure was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER

TWO

INSTALLATION

2.1 From source

The sources for `pcrglobwb_utils` can be downloaded from the Github repo.

You can clone the public repository:

```
$ git clone git://github.com/JannisHoch/pcrglobwb_utils
```

To avoid conflicting package version numbers, it is advised to create a separate `conda` environnmet for this package:

```
$ conda-env create -f=path/to/pcrglobwb_utils/environment.yml
```

Subsequently, activate this environment with:

```
$ conda activate pcrglobwb_utils
```

Installation of the package is then possible:

```
$ cd path/to/pcrglobwb_utils
$ python setup.py develop
```

Alternatively, you can use:

```
$ cd path/to/pcrglobwb_utils
$ pip install -e path/to/pcrglobwb_utils
```

2.2 From PyPI

`pcrglobwb_utils` can also be installed from PyPI. To do so, use this command:

```
$ pip install pcrglobwb-utils
```

If a specific version is required, then the command would need to look like this:

```
$ pip install pcrglobwb-utils==version
```

CHAPTER

THREE

USAGE

You can either use the functions of pcrglobwb_utils and integrate them into your bespoke workflow. Or you can use some of the pre-built command line functions covering some of the most common workflows.

3.1 Within python

To use `pcrglobwb_utils` in a project:

```
import pcrglobwb_utils
```

You have then all the functions available to be used in a bespoke Python-script for output analysis.

See the jupyter notebook in the *Examples* for more information. They also contain links to interactive versions hosted on myBinder.

3.2 From command line

Alternatively, you can use the command line functionality of `pcrglobwb_utils`. There are currently two kinds of applications for which command line scripts are developed.

First, for validating timeseries of simulated discharge. This can be done using GRDC-data (for selected files or entire batch runs) or by providing observations in an Excel-file. The latter option then requires a geojson-file with the locations of the observation stations in the Excel-file.

For further help about these command line scripts, see

```
$ pcru_eval_tims --help
```

And second, to validate timeseries of any other model output with gridded observations in netCDF-format. The validation will be performed at a user-specified aggregation level. This level is defined by providing a geojson-file containing one or multiple polygons for which the spatial mean is computed per time step and evaluation metrics are computed subsequently.

Top-level information about this command line script can be accessed via

```
$ pcru_eval_poly --help
```

3.2.1 Timeseries analysis

These scripts faciliate the validation of simulated timeseries. This can be done in multiple ways. Either for GRDC-stations and the associated default GRDC file standard. In this case, most of the meta-data can be extracted directly from file. The GRDC-script is limited to evaluating discharge simulations. Or via an Excel-file together with a geojson-file providing the locations of all stations. with the Excel-script, any simulated variable can be evaluated.

Validation with GRDC-data

Code documentation

```
Usage: pcru_eval_tims grdc [OPTIONS] NCF DATA_LOC OUT

Uses pcrglobwb_utils to validate simulated time series (currently only
discharge is supported)  with observations (currently only GRDC) for one or
more stations. The station name and file with GRDC data need to be provided
in a separate yml-file. Per station, it is also possible to provide lat/lon
coordinates which will supersede those provided by GRDC. The script
faciliates resampling to other temporal resolutions.

Returns a csv-file with the evaluated time series (OBS and SIM), a csv-file
with the resulting scores (KGE, R2, RMSE, RRMSE, NSE),  and if specified a
simple plot of the time series. If specified, it also returns a geojson-file
containing KGE values per station evaluated.

NCF: Path to the netCDF-file with simulations.

DATA_LOC: either yaml-file or folder with GRDC files.

OUT: Main output directory. Per station, a sub-directory will be created.

Options:
-v, --var-name TEXT             variable name in netCDF-file
-gc, --grdc-column TEXT         name of column in GRDC file to be read (only
                                used with -f option)
-e, --encoding TEXT             encoding of GRDC-files.
-sf, --selection-file TEXT      path to file produced by pcru_sel_grdc
                                function (only used with -f option)
-t, --time-scale TEXT           time scale at which analysis is performed if
                                resampling is desired. String needs to
                                follow pandas conventions.
-N, --number-processes INTEGER  number of processes to be used in
                                multiprocessing.Pool()- defaults to number
                                of CPUs in the system.
--verbose / --no-verbose        more or less print output.
--help                          Show this message and exit.
```

Settings

There are two options how to use this function. What they have in common is that they read a variable `--var-name` from a netCDF-file `NCF` containing simulated data. The variable name default to 'discharge'.

Also, the command line script will create individual sub-folders per evaluated station in the main output folder `OUT`. Per sub-folder, a csv-file with the compuated metrics will be stored along with the underlying timeseries.

Option 1: Detailed analysis

By providing one yml-file as `DATA_LOC` which has the subsequent structure for each location to be analysed:

```
<location_name>
    file: <path/to/GRDC_file>
    lat: <latitude value>
    lon: <longitude value>
    column: <latitude value>
<location_name>
    file: <path/to/GRDC_file>
    lat: <latitude value>
    lon: <longitude value>
    column: <latitude value>
```

`file` needs to point to the GRDC file corresponding to this station. It can be a relative or absolute path.

`lat`, `lon`, and `column` are optional settings.

By default, `pcrglobwb_utils` retrieves latitude and longitude information from the meta-data stored in each GRDC-file, and performs a window search around this location to reduce the risk of a mis-match between GRDC coords and location in the model output. In some cases, this may still not be sufficient and hence coordinates can be provided manually via the yaml-file.

GRDC-files have often multiple columns with data. `pcrglobwb_utils` uses `' Calculated'` as default. If another column is supposed to be read, this can be specified here.

Example

In this example, we make use of a yml-file to validate discharge at locations Obidos and Jaturana (both located in the Amazon).

```
Obidos:
    file: 'path/to/files/3629000_Obidos.day'
    column: ' Original'

Jaturana:
    file: 'path/to/files/3627000_Jatuarana.day'
    lon: -59.65
    lat: -3.05
    column: ' Calculated'
```

While we use the GRDC coordinates for Obidos, we specify them for Jaturana. Also, the column to be read in the GRDC-file differs per station.

The daily values are resampled to monthly values in this example.

```
$ yaml_file='path/to/yaml_file.yml'
$ sim='path/to/model_discharge_output.nc'
$ out='./OUT/'
$ pcru_eval_tims grdc $sim $yaml_file $out_dir -t M
```

Option 2: batch analysis

If a batch of stations is to be analysed, it is possible to provide a folder path where GRDC-files are stored as `DATA_LOC`. `pcrglobwb_utils` will then read all files, retrieve meta-data, and perform the analysis. It is possible to only select stations fulfilling certain requirements by providing a file containing selected stations with option `--selection-file`. This has the advantage that not all files need to be specified in a yaml-file, but on the downside gives less possibilites to finetune the analysis. The only thing that can be provided is the column name in the GRDC file batch via `--grdc-column`.

Note: To reduce the risk of stations not being located in the 'right' cell, a window search is automatically performed to find the best matching cell.

In both cases, it is possible to resample simulated and observed data to larger time steps with `--time-scale`.

To speed up computations, it is possible to parallelise the evaluation by specifying a number of cores as `-number-processes`. Note that the number of cores used may be scaled down to either the number of stations available or the number of cores available.

Example

In the example above, both GRDC files are stored in the folder `path/to/files`. Instead of specifying these files manually, we can just analyse the entire folder content.

When analysing many files, it may make sense to parallelise this process, here across 8 cores. And again, we want to perform the analysis at the monthly scale.

```
$ folder='path/to/files/'
$ sim='path/to/model_discharge_output.nc'
$ out='./OUT/'
$ pcru_eval_tims grdc $sim $folder $out_dir -N 8 -t M
```

Validation with Excel-file

If observations are not sources from GRDC, they can be stored in an Excel-file as an alternative.

Attention: This settings is by far less well tested than the use of GRDC data.

Settings

Key inputs are a netCDF-file containing simulated values (`NCF`). With the option `--var-name`, the variable name can be specified. By default, variable 'discharge' will be read.

Observed values are provided with an Excel-file (`XLS`). The file needs to have two or more columns. The first column contains the dates of observed values. All other columns contain then the observed values themselves. The first row must contain the names of the stations to be analysed (except for the first column which does not have to have a header).

The list of stations to be analysed is retrieved from a geojson-file (`LOC`). It contains the locations (lat/lon) of the stations and also a unique identifier per station which must be provided with `--location-id`.

The command line script will create individual sub-folders per evaluated station in the main output folder `OUT`. Per sub-folder, a csv-file with the compuated metrics will be stored along with the underlying timeseries.

With the `--geojson / --no-geojson` switch, a geojson-file will be stored to `OUT` containing KGE values per evaluated station (or not). Defaults to True.

The `--plot` switch activates printing of simple plots of the timeseries per evaluated station.

Note: While the GRDC script works only with simulated discharge, the Excel script provided more freedom and can be used to evaluate any timeseries and variable simulated with PCR-GLOBWB!

Code documentation

```
Usage: pcr_utils_evaluate excel [OPTIONS] NCF XLS LOC OUT

Uses pcrglobwb_utils to validate simulated time series with observations
for one or more stations. The station names and their locations need to be
provided via geojson-file. Observations are read from Excel-file and
analysis will be performed for all stations with matching names in Excel-
file columns and geojson-file. The Excel-file must have only one sheet
with first column being time stamps of observed values, and all other
columns are observed values per station. These columns must have a header
with the station name. The script faciliates resampling to other temporal
resolutions.

Returns a csv-file with the evaluated time series (OBS and SIM),  a csv-
file with the resulting scores (KGE, r, RMSE, NSE),  and if specified a
simple plot of the time series. If specified, it also returns a geojson-
file containing KGE values per station evaluated.

NCF: Path to the netCDF-file with simulations.

XLS: Path to Excel-file containing dates and values per station.

LOC: Path to geojson-file containing location and names of stations.

OUT: Main output directory. Per station, a sub-directory will be created.

Options:
    -v, --var-name TEXT             variable name in netCDF-file
    -id, --location-id TEXT         unique identifier in locations file.
    -t, --time-scale TEXT           time scale at which analysis is performed if␣
↪upscaling is desired: month, year, quarter.
    --plot / --no-plot              simple output plots.
    --geojson / --no-geojson        create GeoJSON file with KGE per GRDC station.
    --verbose / --no-verbose        more or less print output.
    --help                          Show this message and exit.
```

Example

In this example, each station in the geojson-file with a unqiue identifier 'station' will be matched with the columns in the Excel-file to validate simulated sediment transport.

```
$ sim='path/to/model_output.nc'
$ excel='path/to/data.xlsx'
$ loc='path/to/stations.geojson'
$ out='./OUT/'
$ pcr_utils_evaluate excel -v sedimentTransport -id station $sim $excel_file $loc $out
```

3.2.2 Analysis per polygon

Output from PCR-GLOBWB can be validated against any other gridded dataset as long as it is a netCDF-file.

Settings

In each file containing polygons (`PLY`), it is important that each polygon has a unique identifier `ply-id`. For each of the polygons, the script will compute the spatial average for both simulations and observations for each time step. From the resulting timeseries, r, MSE, and RMSE are derived.

If the units between simulations and observations are not identical, it is possible to apply a `--conversion-factor` which will be multiplied with the simulated values. The default values is 1.

Should it be required to perform the analysis in another coordinate system than EPSG 4326, it can be provided via `--coordinate-system`. Default is EPSG 4326.

In some instances, anomalies are required instead of the raw timeseries. By specifying `--anomaly`, the scripts derives the anomalies of observation and simulation per time step. Default setting is off.

For some variables, PCR-GLOBWB outputs monthly totals. In case the observations are monthly averages, it is possible to derive monthly values by dividing the total with the number of days per month. This can be activated by setting the `--sum` switch. Default is off.

For a quick visual analysis of the output, it is possible to activate the `--plot` switch. Default is off.

Code documentation

```
Usage: pcr_utils_evaluate poly [OPTIONS] PLY SIM OBS OUT

Computes r, MSE, and RMSE for multiple polygons as provided by a shape-file
between simulated and observed data. Each polygon needs to have a unique
ID. Contains multiple options to align function settings with data and
evaluation properties.

Returns a GeoJSON-file of r, MSE, and RMSE per polygon, and if specified as
simple plot.
Also returns scores of r, MSE, and RMSE per polygon as dataframe.

PLY: path to shp-file or geojson-file with one or more polygons.

SIM: path to netCDF-file with simulated data.

OBS: path to netCDF-file with observed data.

OUT: Path to output folder. Will be created if not there yet.

Options:
    -id, --ply-id TEXT               unique identifier in file containing polygons.
    -o, --obs_var_name TEXT          variable name in observations.
    -s, --sim_var_name TEXT          variable name in simulations.
    -cf, --conversion-factor INT     conversion factor applied to simulated values to␣
↪align variable units.
    -crs, --coordinate-system TEXT   coordinate system.
    --anomaly / --no-anomaly         whether or not to compute anomalies.
    --sum / --no-sum                 whether or not the simulated values are monthly␣
↪totals or not.
    --plot / --no-plot               whether or not to save a simple plot of results.
```

(continues on next page)

(continued from previous page)

```
    --verbose / --no-verbose        more or less print output.
    --help                          Show this message and exit.
```

Example

In this example, simulated `total_thickness_of_water_storage` is validated against `lwe_thickness` from GRACE. Since GRACE data is in [cm], simulated data is converted from [m] by multiplying with 100. Also, the anomaly per time step is determined.

```
$ shp='/path/to/polygon.geojson'
$ obs='/path/to/GRACE.nc'
$ sim='/path/to/model_output.nc'
$ out='./OUT/'

$ pcr_utils_evaluate poly -o lwe_thickness -s total_thickness_of_water_storage -cf 100 -
↪id ID --anomaly $shp $sim $obs $out
```

CHAPTER FOUR

EXAMPLES

Here two examples how *pcrglobwb_utils* can be used. Both can be launched via myBinder in interactive mode (link in notebooks).

4.1 Validating terrestrial water storage and evaporation

Important parts of the water balance and assessment of model performance are the terrestrial water storage (TWS) and evaporation (E). In this example, TWS is validated against GRACE/GRACE-FO data and E is benchmarked with GLEAM data. *pcrglobwb_utils* facilitates the validation process by providing scores per polygon of a shp-file.

4.1.1 Preambule

Loading required packages and showing package versions used.

[1]:
```
%matplotlib inline
import pcrglobwb_utils
import xarray as xr
import pandas as pd
import numpy as np
import geopandas as gpd
import matplotlib
import matplotlib.pyplot as plt
import rasterio as rio
import spotpy as sp
import os, sys
import datetime
```

[2]:
```
print('this notebook was created using Python', str((sys.version)), 'on a', str(sys.
↪platform),'on', datetime.datetime.now())
```

```
this notebook was created using Python 3.8.5 | packaged by conda-forge | (default, Jul␣
↪31 2020, 01:53:45) [MSC v.1916 64 bit (AMD64)] on a win32 on 2021-12-03 17:03:53.367184
```

[3]:
```
pcrglobwb_utils.utils.print_versions()
```

```
pcrglobwb_utils version 0.3.3
pandas version 1.2.4
xarray version 0.18.2
```

(continues on next page)

(continued from previous page)

```
numpy version 1.20.3
geopandas version 0.9.0
rasterio version 1.1.5
rioxarray version 0.4.2
```

4.1.2 Select the area

pcrglobwb_utils applies the TWS and E validation for an area as user-specified by means of a shp-file containing one or more polygons. Per polygon, both simulated and observed data is averaged per time step. With the resulting time series, performance metrics can be computed and visualized. For the example here, the water provinces located in Tanzania are used. Hence, the first step is to load the shp-file.

```
[4]: TZA_waterProvinces = pcrglobwb_utils.spatial_validation.validate_per_shape(shp_fo=
     ↪'example_data/Tanzania_shp/waterProvinces.geojson',
                                                                              shp_key=
     ↪'watprovID')
```

```
reading shp-file C:\Users\hoch0001\Documents\_code\pcrglobwb_utils\examples\example_data\
↪Tanzania_shp\waterProvinces.geojson
```

Quickly inspect the extent information of the loaded data.

```
[5]: TZA_waterProvinces.extent_gdf.head()
```

```
[5]:    OBJECTID                  MAJORBASIN   COUNTRY    REGION  \
     0        12  Africa, East Central Coast  Tanzania    Iringa
     1        13  Africa, East Central Coast  Tanzania     Lindi
     2        14  Africa, East Central Coast  Tanzania   Manyara
     3        15  Africa, East Central Coast  Tanzania     Mbeya
     4        16  Africa, East Central Coast  Tanzania  Morogoro

                                          watprov  watprovID       km2  \
     0    Tanzania_Iringa_Africa, East Central Coast         25   88460.9
     1     Tanzania_Lindi_Africa, East Central Coast         28   85004.1
     2   Tanzania_Manyara_Africa, East Central Coast         29   36772.8
     3     Tanzania_Mbeya_Africa, East Central Coast         30   17342.4
     4  Tanzania_Morogoro_Africa, East Central Coast         31  146718.0

        Shape_Leng  Shape_Area                                           geometry
     0   22.002607    7.195788  POLYGON ((36.29721 -5.12083, 36.29583 -5.12503...
     1   15.485085    6.865913  MULTIPOLYGON (((40.19511 -10.26096, 40.19125 -...
     2   13.405812    2.976842  POLYGON ((37.17537 -2.76838, 37.17193 -2.76958...
     3    8.245606    1.414085  POLYGON ((34.15102 -7.38556, 34.15304 -7.39140...
     4   29.906623   11.605295  MULTIPOLYGON (((39.25203 -8.00781, 39.24966 -8...
```

TWS validation

For validating simulated TWS, netCDF-files of both simulation and observation need to be provided. Per polygon, time series are retrieved and evaluated. This yields a dataframe with R and RMSE value per polygon.

[6]:
```
watprov_gdf = TZA_waterProvinces.against_GRACE(PCR_nc_fo='example_data/GRACE/
↪totalWaterStorageThickness_monthAvg_output_2010_Tanzania.nc',
                                                GRACE_nc_fo='example_data/GRACE/GRACE_
↪data_2010_Tanzania.nc')
```

```
reading GRACE file C:\Users\hoch0001\Documents\_code\pcrglobwb_utils\examples\example_
↪data\GRACE\GRACE_data_2010_Tanzania.nc
reading PCR-GLOBWB file C:\Users\hoch0001\Documents\_code\pcrglobwb_utils\examples\
↪example_data\GRACE\totalWaterStorageThickness_monthAvg_output_2010_Tanzania.nc
extract raw data from nc-files
clipping nc-files to extent of shp-file
computing R and RMSE for polygon with key identifier watprovID 25
computing R and RMSE for polygon with key identifier watprovID 28
computing R and RMSE for polygon with key identifier watprovID 29
computing R and RMSE for polygon with key identifier watprovID 30
computing R and RMSE for polygon with key identifier watprovID 31
computing R and RMSE for polygon with key identifier watprovID 32
computing R and RMSE for polygon with key identifier watprovID 33
computing R and RMSE for polygon with key identifier watprovID 443
computing R and RMSE for polygon with key identifier watprovID 444
computing R and RMSE for polygon with key identifier watprovID 445
computing R and RMSE for polygon with key identifier watprovID 1021
computing R and RMSE for polygon with key identifier watprovID 1226
computing R and RMSE for polygon with key identifier watprovID 1227
computing R and RMSE for polygon with key identifier watprovID 1228
computing R and RMSE for polygon with key identifier watprovID 1229
```

As *pcrglobwb_utils* returns a geo-dataframe, the R and RMSE values can also be plotted.

[7]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 10))
watprov_gdf.plot(column='R', ax=ax1, cmap='magma', legend=True, legend_kwds={'orientation
↪':'horizontal', 'label':'R'})
watprov_gdf.plot(column='RMSE', ax=ax2, cmap='viridis', legend=True, legend_kwds={
↪'orientation':'horizontal', 'label':'RMSE'})
plt.tight_layout();
```

E benchmark

The same workflow is followed when benchmarking simulated E with GLEAM data.

[8]:
```
watprov_gdf = TZA_waterProvinces.against_GLEAM(PCR_nc_fo='example_data/GLEAM/
↪totalEvaporation_monthTot_output_2010_Tanzania.nc',
                                                GLEAM_nc_fo='example_data/GLEAM/GLEAM_
↪data_2010_Tanzania.nc')
```

```
reading GLEAM file C:\Users\hoch0001\Documents\_code\pcrglobwb_utils\examples\example_
↪data\GLEAM\GLEAM_data_2010_Tanzania.nc
reading PCR-GLOBWB file C:\Users\hoch0001\Documents\_code\pcrglobwb_utils\examples\
↪example_data\GLEAM\totalEvaporation_monthTot_output_2010_Tanzania.nc
```

(continues on next page)

(continued from previous page)

```
extract raw data from nc-files
clipping nc-files to extent of shp-file
computing R and RMSE for polygon with key identifier watprovID 25
computing R and RMSE for polygon with key identifier watprovID 28
computing R and RMSE for polygon with key identifier watprovID 29
computing R and RMSE for polygon with key identifier watprovID 30
computing R and RMSE for polygon with key identifier watprovID 31
computing R and RMSE for polygon with key identifier watprovID 32
computing R and RMSE for polygon with key identifier watprovID 33
computing R and RMSE for polygon with key identifier watprovID 443
computing R and RMSE for polygon with key identifier watprovID 444
computing R and RMSE for polygon with key identifier watprovID 445
computing R and RMSE for polygon with key identifier watprovID 1021
computing R and RMSE for polygon with key identifier watprovID 1226
computing R and RMSE for polygon with key identifier watprovID 1227
computing R and RMSE for polygon with key identifier watprovID 1228
computing R and RMSE for polygon with key identifier watprovID 1229
```

[9]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 10), sharey=True)
watprov_gdf.plot(column='R', ax=ax1, cmap='magma', legend=True, legend_kwds={'orientation
↪':'horizontal', 'label':'R'})
watprov_gdf.plot(column='RMSE', ax=ax2, cmap='viridis', legend=True, legend_kwds={
↪'orientation':'horizontal', 'label':'RMSE'})
plt.tight_layout()
```

4.1.3 Command line functionality

Similar process, but less interactive is the command line functionality. Using `click` it is possible to evaluate model output with any other gridded dataset per polygon.

[10]:
```
pcrglobwb_utils.eval.POLY('example_data/Tanzania_shp/waterProvinces.geojson', 'example_
↪data/GLEAM/totalEvaporation_monthTot_output_2010_Tanzania.nc','example_data/GLEAM/
↪GLEAM_data_2010_Tanzania.nc', './_OUT/TEST', 'watprovID', 'E', 'total_evaporation',␣
↪conversion_factor=100)
```

```
INFO -- start.
INFO -- pcrglobwb_utils version 0.3.3.
INFO -- saving output to folder C:\Users\hoch0001\Documents\_code\pcrglobwb_utils\
↪examples\_OUT\TEST
INFO -- reading observed variable E from example_data/GLEAM/GLEAM_data_2010_Tanzania.nc
INFO -- reading simulated variable total_evaporation from example_data/GLEAM/
↪totalEvaporation_monthTot_output_2010_Tanzania.nc
INFO -- reading polygons from C:\Users\hoch0001\Documents\_code\pcrglobwb_utils\examples\
↪example_data\Tanzania_shp\waterProvinces.geojson
INFO -- evaluating each polygon
INFO -- storing dictionary to C:\Users\hoch0001\Documents\_code\pcrglobwb_utils\examples\
↪_OUT\TEST\output_dict.csv.
INFO -- storing polygons to C:\Users\hoch0001\Documents\_code\pcrglobwb_utils\examples\_
↪OUT\TEST\output_polygons.geojson.
INFO -- done.
INFO -- run time: 0:00:05.927825.
```

```
[11]: import shutil
shutil.rmtree('./_OUT/TEST')
```

4.2 Validating simulated discharge

One typical use case is to validate simulated discharge with observations. In this example, observations from GRDC are used. *pcrglobwb_utils* facilitates reading time series from netCDF-output at either user-specified or GRDC-specified coordinates, plotting time series of simulation and observation, and computing evaluation metrics.

4.2.1 Preambule

Loading required packages and showing package versions used.

```
[1]: %matplotlib inline
import pcrglobwb_utils
import xarray as xr
import pandas as pd
import numpy as np
import geopandas as gpd
import matplotlib.pyplot as plt
import rasterio as rio
import spotpy as sp
import os, sys
import datetime
```

```
[2]: print('this notebook was created using Python', str((sys.version)), 'on a', str(sys.
↪platform),'on', datetime.datetime.now())
```

```
this notebook was created using Python 3.8.5 | packaged by conda-forge | (default, Jul␣
↪31 2020, 01:53:45) [MSC v.1916 64 bit (AMD64)] on a win32 on 2021-07-15 10:44:11.157586
```

```
[3]: pcrglobwb_utils.utils.print_versions()
```

```
pcrglobwb_utils version 0.2.4b
pandas version 1.2.4
xarray version 0.18.2
numpy version 1.20.3
geopandas version 0.9.0
rasterio version 1.1.5
rioxarray version 0.4.2
```

4.2.2 Loading GRDC data

For this showcase, we use daily observations at a station in the Amazon River basin. With *pcrglobwb_utils*, it is possible to automatically retrieve the name of the station from file as well as the coordinates associated with this station. Note that the coordinates provided by GRDC do not always coincide with the PCR-GLOBWB river network and thus the location should be double-checked - good that *pcrglobwb_utils* got you covered!

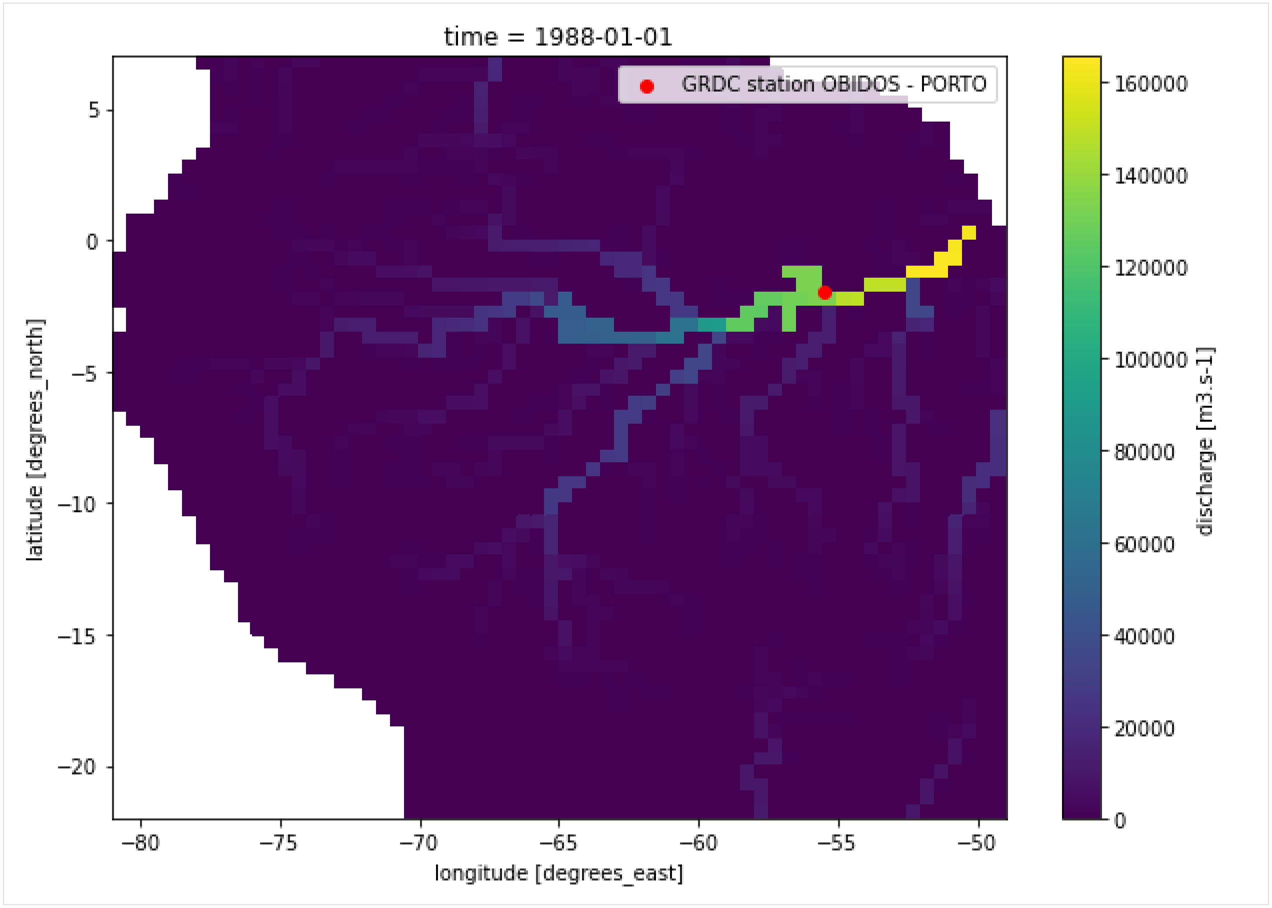
```
[4]: grdc_data = pcrglobwb_utils.obs_data.grdc_data('example_data/GRDC/files/3629000_Obidos.
     ↪day')
```

```
[5]: plot_title, props = grdc_data.get_grdc_station_properties()
     print(plot_title)
     print(props)
```

```
station OBIDOS - PORTO at latitude/longitude -1.947200/-55.511100
{'station': 'OBIDOS - PORTO', 'latitude': -1.9472, 'longitude': -55.5111}
```

With all this information, let's plot the values of the 'discharge' variable from the netCDF-output together with the location of the GRDC-file.

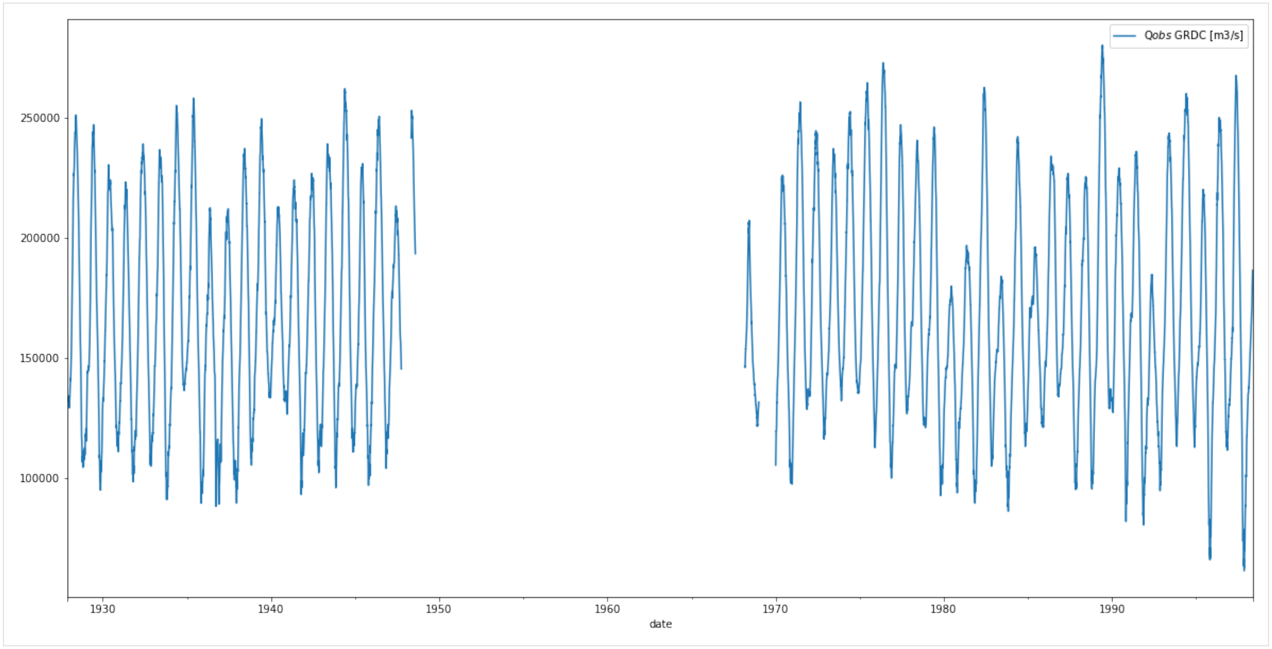
```
[6]: plt.figure(figsize=(10,7))
     pcrglobwb_utils.plotting.plot_var_at_timestep('example_data/GRDC/DUMMY_discharge_
     ↪dailyTot_output.nc',
                                                   var_name='discharge',
                                                   time='1988-01-01')
     plt.scatter(props['longitude'], props['latitude'], c='r', label='GRDC station {}'.
     ↪format(props['station']))
     plt.legend();
```



We have now a lot of meta-data, let's read the actual observed discharge from the file and plot the time series.

```
[7]: df_GRDC, props = grdc_data.get_grdc_station_values(var_name='Q$obs$ GRDC [m3/s]', col_
↪name=' Original')
```

```
[8]: df_GRDC.plot(figsize=(20,10));
```



4.2.3 Loading PCR-GLOBWB data

We now know where the GRDC-station is located, but not to which cell (expressed as row/column combination) this location corresponds. Let's derive this information!

By means of the row/column combination, we can extract the simulated discharge values from the netCDF-file from this cell.

```
[12]: pcr_data = pcrglobwb_utils.sim_data.from_nc('example_data/GRDC/DUMMY_discharge_dailyTot_
      ↪output.nc')
```

```
[13]: row, col = pcr_data.find_indices_from_coords(props['longitude'], props['latitude'])
```

```
[16]: print('The location {} with its latitude {} and longitude {} corresponds to the cell␣
      ↪with row and cell indeces {}'.format(props['station'],
                                                                                          ␣
      ↪                                 props['latitude'],
                                                                                          ␣
      ↪                                 props['longitude'],
                                                                                          ␣
      ↪                                 (row, col)))
```

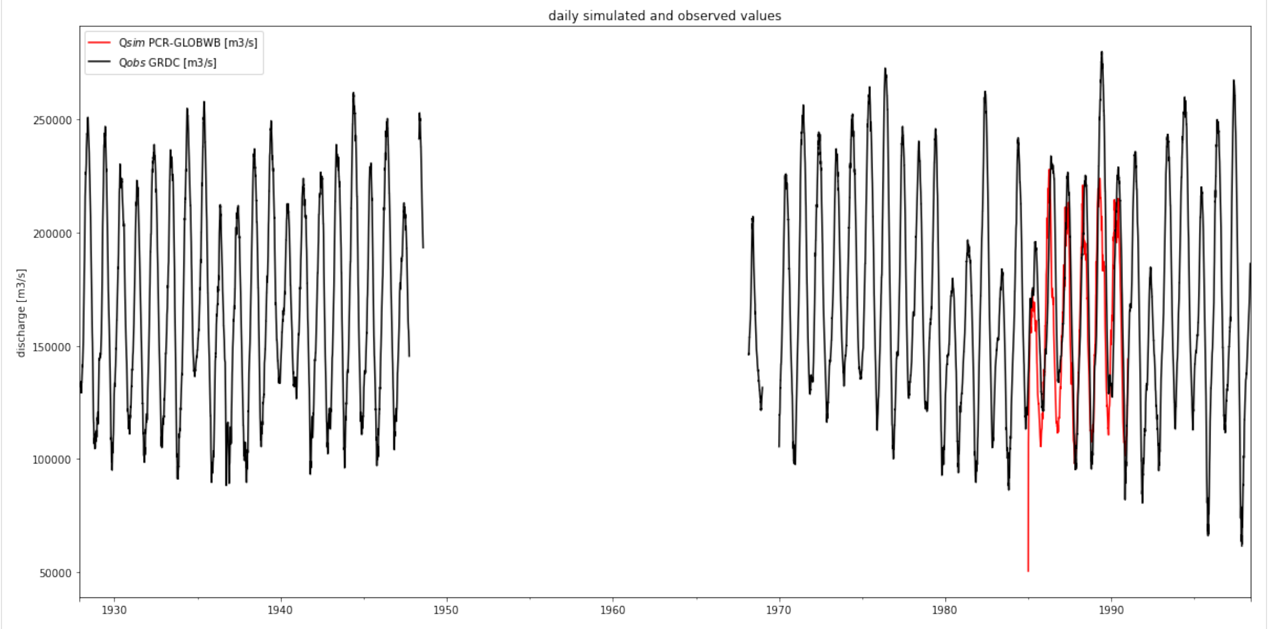
```
The location OBIDOS - PORTO with its latitude -1.9472 and longitude -55.5111 corresponds␣
↪to the cell with row and cell indeces (17, 50)
```

```
[14]: q_sim_obidos = pcr_data.read_values_at_indices(row, col, plot_var_name='Q$sim$ PCR-
      ↪GLOBWB [m3/s]', plot=False)
```

4.2.4 Validating simulated discharge with observations

Visualizing time series of simulation and observation shows a) that our example simulation data covers only a small part of the available observations, and b) that simulated discharge agrees quite good with GRDC data.

[15]:
```
fig, ax = plt.subplots(1, 1, figsize=(20,10))
q_sim_obidos.plot(ax=ax, c='r')
df_GRDC.plot(ax=ax, c='k')
ax.set_ylabel('discharge [m3/s]')
ax.set_xlabel(None)
plt.legend()
plt.title('daily simulated and observed values');
```



pcrglobwb_utils allows for validating simulations with observations and returns the KGE (optionally with all components), NSE, RMSE and coefficient of determination r2.

[19]:
```
scores = pcr_data.validate_results(df_GRDC, out_dir='./_OUT', return_all_KGE=False)
```

[20]:
```
scores
```

[20]:
```
        KGE       NSE           MSE          RMSE        R2
0  0.712195  0.496277  9.555419e+08  30911.840814  0.599577
```

[ ]:

CHAPTER FIVE

API-DOCS

This section contains the Documentation of the Application Programming Interface (API) of `pcrglobwb_utils`.

5.1 Observed data

There are function to load (meta-)data of observations from file:

5.1.1 From GRDC data

For reading GRDC data, working with a python-object can be done like this.

5.1.2 From other sources

5.2 Simulated data

5.2.1 From netCDF files

`pcrglobwb_utils` has a dedicated class to extract values from a netCDF-file for a given location. Also, the timeseries can be resampled in time.

5.2.2 Functions

5.3 Analysing ensembles

If we have output at one location from various runs, for example for future climate scenarios, it can be useful to analyse the mean, max, and min thereof. Also, one may want to vizualize it. Besides, it is also possible to determine the long-term monthly average and plot it.

5.4 Time functions

Output from PCR-GLOBWB (particularly discharge) is often at the daily resolution.

In case statistics or aggregations in time are needed, those functions could be used:

`time_funcs.`**`calc_monthly_climatology`**(*df_in: DataFrame*, *col_name=None*) → DataFrame

Calculates the climatological mean of each month across a timeseries at sub-monthly timestep.

Parameters

- **`df_in`** (*`pd.DataFrame`*) – dataframe containing timeseries at sub-monthly timestep.
- **`col_name`** (*`str, optional`*) – name of column to be considered only. Defaults to None.

Returns
dataframe containing mean of each month.

Return type
pd.DataFrame

`time_funcs.`**`resample_time`**(*df: DataFrame*, *resampling_period: str*) → DataFrame

Resamples a dataframe in time. The resampling duration is set with 'time' and needs to follow pandas conventions. Output needs to be combined with a statistic, such as ".mean()".

Parameters

- **`df`** (*`pd.DataFrame`*) – dataframe to be resampled.
- **`resampling_period`** (*`str`*) – resampling duration.

Returns
actually returns a pd.core.resample.DatetimeIndexResampler

Return type
pd.DataFrame

`time_funcs.`**`resample_to_annual`**(*df: DataFrame*, *stat_func='mean'*, *suffix=None*) → DataFrame

Resamples a timeseries at sub-annual time step to annual values. A range of annual statistics can be chosen. If desired, the column name of the returned dataframe can contain a suffix for better distinguishment. By default, column names are unaltered.

Parameters

- **`df`** (*`pd.DataFrame`*) – dataframe containing timeseries. Note, only tested with dataframes containing one column.
- **`stat_func`** (*`str, optional`*) – Statistical method to be used. Either 'mean', 'max', 'min' or 'sum'. Defaults to 'mean'.
- **`suffix`** (*`str, optional`*) – Suffix to be added to column of returned dataframe. Defaults to None.

Returns
dataframe containing resampled timeseries.

Return type
pd.DataFrame

`time_funcs.`**`resample_to_month`**(*df: DataFrame*, *stat_func='mean'*, *suffix=None*) → DataFrame

Resamples a timeseries at sub-monthly time step to monthly values. A range of monthly statistics can be chosen. If desired, the column name of the returned dataframe can contain a suffix for better distinguishment. By default, column names are unaltered.

Parameters

- **df** (*pd.DataFrame*) – dataframe containing timeseries. Note, only tested with dataframes containing one column.
- **stat_func** (*str, optional*) – Statistical method to be used. Either 'mean', 'max', 'min' or 'sum'. Defaults to 'mean'.
- **suffix** (*str, optional*) – Suffix to be added to column of returned dataframe. Defaults to None.

Returns
dataframe containing resampled timeseries.

Return type
pd.DataFrame

5.5 Water Balance

As the model already indicates (PCR-GLOB Water Balance), it's all about the water balance. While the water balance computations are not direct output of PCR-GLOBWB, their components can be retrieved from the log-file.

`class water_balance.`**`water_balance`**`(`*fo*`)`

Annual water balance information of a PCR-GLOBWB run. Data is retrieved from the log-file of this run.

Parameters
fo (*str*) – path to log-file

`bar_plot``(`***kwargs*`)`

Creates a bar plot of water balance components per year. This adds to the regular plotting options with pandas dataframes.

`get_annual_values``()`

Get annual values for a range of water balance components by parsing the log-file.

Returns
dataframe containing annual values of water balance components

Return type
dataframe

5.6 Functions for validation

5.6.1 Timeseries

GRDC

The top-level to evalute simulated timeseries with GRDC observations can be used via command line. See _usage_timeseries.

GSIM

Todo: Would be very nice to have

EXCEL

Todo: Documentation needs to be added still.

5.6.2 Polygons

With `pcrglobwb_utils` it is possible to validate spatially-varying PCR-GLOBWB output against a range of datasets. Per domain, area averages per timestep are computed and timeseries validated.

`class spatial_validation.`**`validate_per_shape`**(*shp_fo*, *shp_key*, *crs='epsg:4326'*, *out_dir=None*)

Initializing object for validating output for area(s) provided by shp-file. If the shp-file contains multiptle (polygon) geometries, validation is performed per individual geometry. Per geometry, r and RMSE are determined.

Parameters

- **`shp_fo`** (*`str`*) – Path to shp-file defining the area extent for validation.
- **`shp_key`** (*`str`*) – Column name in shp-file to be used as unique identifier per entry in shp-file.
- **`crs`** (*`str, optional`*) – Definition of projection system in which validation takes place. Defaults to 'epsg:4326'.
- **`out_dir`** (*`str, optional`*) – Path to output directory. In None, then no output is stored. Defaults to None.

Methods Summary

`against_GLEAM`(PCR_nc_fo, GLEAM_nc_fo[, ...])	With this function, simulated land surface evaporation (or another evaporation output) from PCR-GLOBWB can be validated against evaporation data from GLEAM (or any other evaporation data in GLEAM).
`against_GRACE`(PCR_nc_fo, GRACE_nc_fo[, ...])	With this function, simulated totalWaterStorage output from PCR-GLOBWB can be validated against GRACE-FO observations.

Methods Documentation

against_GLEAM(*PCR_nc_fo*, *GLEAM_nc_fo*, *PCR_var_name='total_evaporation'*, *GLEAM_var_name='E'*, *convFactor=1000*)

With this function, simulated land surface evaporation (or another evaporation output) from PCR-GLOBWB can be validated against evaporation data from GLEAM (or any other evaporation data in GLEAM). Works with monthly totals and computes monthly area averages per time step from it.

Parameters

- **PCR_nc_fo** (*str*) – Path to netCDF-file containing evaporation output from PCR-GLOBWB.
- **GLEAM_nc_fo** (*str*) – Path to netCDF-file containing GLEAM evaporation data.
- **PCR_var_name** (*str, optional*) – netCDF variable name in PCR-GLOBWB output. Defaults to 'land_surface_evaporation'.
- **GLEAM_var_name** (*str, optional*) – netCDF variable name in GLEAM data. Defaults to 'E'.
- **convFactor** (*int, optional*) – conversion factor to convert PCR-GLOBWB units to GLEAM units. Defaults to 1000.

Returns
containing data of shp-file appended with columns for R and RMSE per entry.

Return type
geo-dataframe

against_GRACE(*PCR_nc_fo*, *GRACE_nc_fo*, *PCR_var_name='total_thickness_of_water_storage'*, *GRACE_var_name='lwe_thickness'*, *convFactor=100*)

With this function, simulated totalWaterStorage output from PCR-GLOBWB can be validated against GRACE-FO observations. Yields timeseries of anomalies. Works with monthly averages and computes monthly area averages per time step from it.

Parameters

- **PCR_nc_fo** (*str*) – Path to netCDF-file containing totalWaterStorage output from PCR-GLOBWB.
- **GRACE_nc_fo** (*str*) – Path to netCDF-file containing GRACE-FO data
- **PCR_var_name** (*str, optional*) – netCDF variable name in PCR-GLOBWB output. Defaults to 'total_thickness_of_water_storage'.
- **GRACE_var_name** (*str, optional*) – netCDF variable name in GRACE-FO data. Defaults to 'lwe_thickness'.
- **convFactor** (*int, optional*) – conversion factor to convert PCR-GLOBWB units to GRACE-FO units. Defaults to 100.

Returns
containing data of shp-file appended with columns for R and RMSE per entry.

Return type
geo-dataframe

CHAPTER SIX

ABOUT

6.1 Authors

6.1.1 Development Lead

- Jannis M. Hoch <j.m.hoch@uu.nl>

6.1.2 Contributors

- Niko Wanders (Utrecht University)

6.2 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

6.2.1 Types of Contributions

Report Bugs

Report bugs at https://github.com/JannisHoch/pcrglobwb_utils/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

Write Documentation

pcrglobwb_utils could always use more documentation, whether as part of the official pcrglobwb_utils docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/JannisHoch/pcrglobwb_utils/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

6.2.2 Get Started!

Ready to contribute? Here's how to set up `pcrglobwb_utils` for local development.

1. Fork the `pcrglobwb_utils` repo on GitHub.
2. Clone your fork locally

```
$ git clone git@github.com:your_name_here/pcrglobwb_utils.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development

```
$ mkvirtualenv pcrglobwb_utils
$ cd pcrglobwb_utils/
$ python setup.py develop
```

4. Create a branch for local development

```
$ git checkout -b name-of-your-bugfix-or-feature
```

Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox

```
$ flake8 pcrglobwb_utils tests
$ python setup.py test or pytest
$ tox
```

To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

6.2.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.8 or higher, and for PyPI. Check https://travis-ci.com/JannisHoch/pcrglobwb_utils/pull_requests and make sure that the tests pass for all supported Python versions.

6.2.4 Tips

To run a subset of tests:

```
$ pytest tests.test_pcrglobwb_utils
```

6.2.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bump2version patch # possible: major / minor / patch
$ git push
$ git push --tags
```

6.3 License

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX