radproc Documentation

Release 0.1.0

Kreklow

May 30, 2018

Contents

Release 0.1.0

Date May 30, 2018

Radproc is an open source Python library intended to faciliate precipitation data processing and analysis for ArcGIS-users. It provides functions for processing, analysis and export of RADOLAN (Radar Online Adjustment) composites and rain gauge data in MR90 format. The German Weather Service (DWD) provides the RADOLAN-Online RW composites for free in the Climate Data Center (ftp://ftp-cdc.dwd.de/pub/CDC/grids_germany/hourly/radolan/) but the data processing represents a big challenge for many potential users. Radproc's goal is to lower the barrier for using these data, especially in conjunction with ArcGIS. Therefore, radproc provides an automated ArcGIS-compatible data processing workflow based on pandas DataFrames and HDF5. Moreover, radproc's arcgis module includes a collection of functions for data exchange between pandas and ArcGIS.

CHAPTER 1

Radproc's Main Features

1.1 Raw Data processing

- Support for the reanalyzed RADOLAN products RW (60 min), YW and RY (both 5 min. resolution)
- Automatically reading in all binary RADOLAN composites from a predefined directory structure
- Optionally clipping the composites to a study area in order to reduce data size
- Default data structure: Monthly pandas DataFrames with full support for time series analysis and spatial location of each pixel
- Efficient data storage in HDF5 format with fast data access and optional data compression
- Easy downsampling of time series
- Reading in DWD rain gauge data in MR90 format into the same data structure as RADOLAN.

1.2 Data Exchange with ArcGIS

- Export of single RADOLAN composites or analysis results into projected raster datasets or ESRI grids for your study area
- Export of all DataFrame rows into raster datasets in a new file geodatabase, optionally including several statistics rasters
- Import of dbf tables (stand-alone or attribute tables of feature classes) into pandas DataFrames
- Joining DataFrame columns to attribute tables
- Extended value extraction from rasters to points (optionally including the eight surrounding cells)
- Extended zonal statistics

1.3 Analysis

- Calculation of precipitation sums for arbitrary periods of time
- Heavy rainfall analysis, e.g. identification, counting and export of rainfall intervals exceeding defined thresholds
- Data quality assessment
- Comparison of RADOLAN and rain gauge data
- *In preparation: Erosivity analysis, e.g. calculation of monthly, seasonal or annual R-factors*

1.3.1 Getting Started

System requirements

To be able to use all features offered by radproc, you need. . .

- a 64-Bit operating system (32-Bit systems can not allocate more than 3 GB memory, which is not sufficient for radar data processing)
- Python version 2.7 (64-Bit). It is strongly recommended to use the Anaconda distribution since this already includes all needed scientific site-packages.
- ArcMap version 10.4 or newer
- ArcGIS 64-Bit background processing
- for processing of RADOLAN data in 5-minute resolution at least 16 GB RAM are recommended

Installation

First, install ArcMap for Desktop and its extension 64-Bit background processing.

Next, download and install the latest Anaconda distribution from https://www.anaconda.com/download/ (Windows, 64-Bit, Python version 2.7).

radproc is currently distributed as wheel file for Python version 2.7 on Windows operating systems (64-Bit only!). You can download the radproc wheel from the GitHub repository at https://github.com/jkreklow/radproc/tree/0.1.0/dist

To install radproc using Anaconda and pip. . .

1. Open the Windows terminal by typing CMD into the Windows search (Administrator rights may be necessary!).
2. Type:

   ```
   pip install C:\path\to\wheelfile\radproc_wheel.whl
   ```

   Now radproc is automatically installed into your Anaconda root environment. You can check by opening Spyder or Jupyter Notebook and entering:

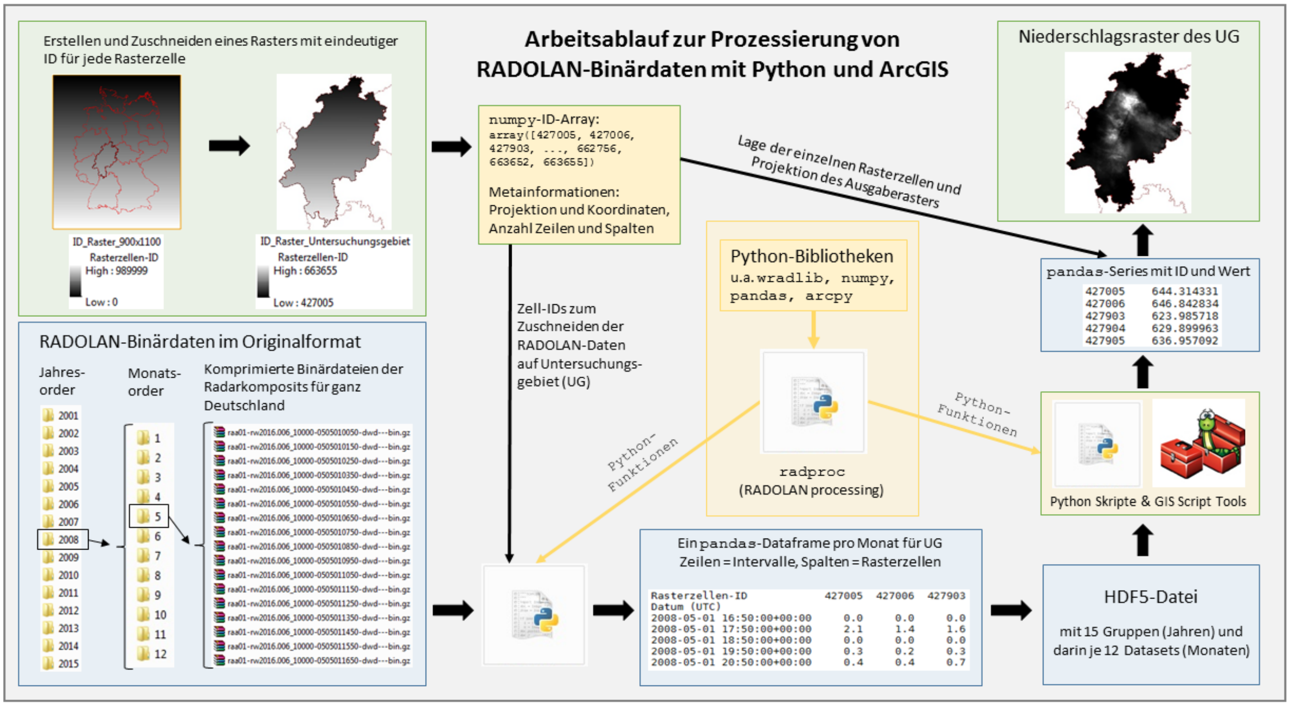
   ```
   import radproc
   ```

3. To enable your Anaconda distribution to "see" the arcpy package from your separate ArcGIS Python installation, you need to **copy the Path file DTBGGP64.pth** which is usually located at *C:\Python27\ArcGISx6410.4\Lib\site-packages* into the corresponding site-packages folder of your Anaconda distribution, e.g. *C:\ProgramData\Anaconda2\Lib\site-packages*

   To check if arcpy is now visible for Anaconda, import arcpy to Spyder or Jupyter Notebook:

```
import arcpy
```

File system description



File system and processing workflow used by radproc.

1.3.2 Tutorials

These tutorials aim to help you getting started with radproc. More tutorials are in progress. . .

Tutorial 1: Raw Data Processing with Radproc

This tutorial will show you how to get started with RADOLAN processing and import your raw hourly RW data into HDF5.

Note: For this approach ArcMap is required!

1. Import radproc

```
In [1]: import radproc as rp
```

2. Unzip Raw Data Archives

The RADOLAN RW product is usually provided as monthly tar.gz archives by the German Weather Service (DWD). These have to be unzipped in order to import the contained hourly binary files. The radproc function

```
unzip_RW_binaries(zipFolder, outFolder)
```

can be used to unzip all archives in one directory into the directory tree format needed by the following radproc functions. Moreover, as the binary files themselves might not be zipped, all binary files are automatically compressed to .gz files to save disk space.

```
In [2]: RW_original = r"O:\Data\RW_archives"
        RW_unzipped = r"O:\Data\RW_unzipped"

        rp.unzip_RW_binaries(zipFolder=RW_original, outFolder=RW_unzipped)
```

Side Note: To unzip the YW or RY products, which might be provided as monthly archives which contain daily archives with the actual binary files, you can use the function

```
unzip_YW_binaries(zipFolder, outFolder)
```

The further processing workflow is the same for all products except that you need more memory space and patience (or a smaller study area) to process the products with higher temporal resolution.

Side Note: If you already have unpacked binary files (e.g. after download of recent RADOLAN-Online files from Climate Data Center) you can skip this step, but you need to make sure that the files are saved in monthly directories (if you have data for more than one month) to make all functions of radproc work correctly.

3. Import Unzipped Data into HDF5

To import all RW data you have just unzipped into an HDF5 file - optionally clipping the data to a study area - you can apply

```
create_idraster_and_process_radolan_data(inFolder, HDFFile, clipFeature=None,␣
↪complevel=9)
```

Behind the scenes, this function will…

- create an ID-raster for Germany in ArcGIS, called *idras_ger*,
- if you specified a Shapefile or Feature-Class as clipFeature: Clip the german ID-raster to the extent of the clipFeature and create a second ID-raster called *idras*,
- import all RADOLAN binary files in a directory tree,
- select the data for your study area based on the generated ID-raster,
- convert the selected data into monthly pandas DataFrames and
- store all DataFrames in the specified HDF5 file.

The result of this function is a HDF5 file with all RADOLAN data of your study area ready for further analysis.

Note: This function works with RADOLAN-Online data as well as with the reanalyzed RADOLAN climatology data, which differ in data size and location. All necessary information are extracted from the RADOLAN metadata or are inherently contained within radproc.

More detailed information on the four function parameters are available in the library reference of the function.

```
In [3]: outHDF = r"O:\Data\RW.h5"
        studyArea = r"O:\Data\StudyArea.shp"

        rp.create_idraster_and_process_radolan_data(inFolder=RW_unzipped, HDFFile=outHDF, clipFeature
```

```
O:\Data\RW_unzipped\2016\5 imported, clipped and saved
```

Now you are ready to start analyzing your data!

Tutorial 2: Aggregation to Precipitation Sums

This tutorial will show you, how to calculate precipitation sums from the data stored in HDF5 and how to export the results to ArcGIS.

To import your RADOLAN data into the necessary HDF5 file, please follow the tutorial on raw data processing.

Example 1: Annual Precipitation Sums

In this example, the annual precipitation sums for the time period from 2001 to 2016 are calculated and exported to ArcGIS.

1. Import radproc

```
In [1]: import radproc as rp
```

2. Load Data from HDF5 and Aggregate to Annual Precipitation Sums

The following function loads precipitation data of the specified time period (2001-2016) from an HDF5 file and generates a DataFrame with annual precipitation sums for every raster cell.

```
In [2]: HDF = r"O:\Data\RW_2001_2016.h5"

        annualSum = rp.hdf5_to_years(HDFFile=HDF, year_start=2001, year_end=2016)
        # Display the first five rows of the new DataFrame
        annualSum.head()

Out[2]: Rasterzellen-ID                427005      427006      427903      427904  \
        Datum (UTC)
        2001-12-31 00:00:00+00:00  783.400024  809.500000  763.299988  773.500000
        2002-12-31 00:00:00+00:00  967.700012  980.799988  951.000000  935.500000
        2003-12-31 00:00:00+00:00  597.200012  614.200012  578.000000  583.000000
        2004-12-31 00:00:00+00:00  795.000000  801.000000  772.000000  768.099976
        2005-12-31 00:00:00+00:00  728.099976  710.799988  678.500000  676.799988

        Rasterzellen-ID                427905      427906      428803      428804  \
        Datum (UTC)
        2001-12-31 00:00:00+00:00  779.700012  788.500000  769.500000  780.400024
        2002-12-31 00:00:00+00:00  947.200012  950.700012  956.000000  960.700012
        2003-12-31 00:00:00+00:00  592.200012  592.299988  573.400024  587.700012
        2004-12-31 00:00:00+00:00  776.600037  792.799988  746.500000  767.099976
        2005-12-31 00:00:00+00:00  730.200012  681.399963  671.400024  664.099976

        Rasterzellen-ID                428805      428806     ...         661855  \
        Datum (UTC)                                           ...
        2001-12-31 00:00:00+00:00  778.700012  783.000000     ...    1066.599976
        2002-12-31 00:00:00+00:00  938.700012  955.400024     ...    1491.599976
        2003-12-31 00:00:00+00:00  587.299988  589.600037     ...     697.600037
        2004-12-31 00:00:00+00:00  779.000000  792.900024     ...     851.599976
        2005-12-31 00:00:00+00:00  658.799988  666.299988     ...     914.799988

        Rasterzellen-ID                 661856       661857       662752       662753  \
```

```
Datum (UTC)
2001-12-31 00:00:00+00:00  1098.699951  1125.400024   994.000000   998.700012
2002-12-31 00:00:00+00:00  1529.500000  1539.199951  1383.099976  1387.500000
2003-12-31 00:00:00+00:00   696.099976   683.700012   670.400024   652.599976
2004-12-31 00:00:00+00:00   872.200012   869.799988   803.000000   802.000000
2005-12-31 00:00:00+00:00   922.900024   911.099976   847.799988   861.500000

Rasterzellen-ID                 662754       662755       662756       663652  \
Datum (UTC)
2001-12-31 00:00:00+00:00  1020.599976  1056.099976  1075.200073   989.200012
2002-12-31 00:00:00+00:00  1407.400024  1478.099976  1488.800049  1380.400024
2003-12-31 00:00:00+00:00   638.400024   663.599976   668.599976   663.099976
2004-12-31 00:00:00+00:00   804.099976   825.099976   841.599976   806.000000
2005-12-31 00:00:00+00:00   868.099976   893.200012   903.299988   850.799988

Rasterzellen-ID                 663655
Datum (UTC)
2001-12-31 00:00:00+00:00  1039.500000
2002-12-31 00:00:00+00:00  1467.099976
2003-12-31 00:00:00+00:00   648.000000
2004-12-31 00:00:00+00:00   813.700012
2005-12-31 00:00:00+00:00   872.500000

[5 rows x 23320 columns]
```

Note: All of radproc's aggregation functions are intended for analysis of longer time periods and currently only work **for entire years starting in January**! To resample smaller time periods, you can e.g. import and resample months with

```
May2016 = rp.load_months_from_hdf5(HDFFile=HDF, year=2016,  months=[5])
freq = 'M' # 'M' for monthly sums, 'D' for daily sums, 'H' for hourly sums
monthSum = May2016.resample(frequency=freq, closed = 'right', label = 'right').sum()
```

3. Export Results into ArcGIS Geodatabase

The following function exports all rows from the DataFrame calculated above into raster datasets in an ArcGIS File Geodatabase. Optionally, different statistics rasters can be created, e.g. the mean or the maximum value of each cell.

```
In [3]: idRaster = r"O:\Data\idras"
        outGDBPath = r"O:\Data"
        GDBName = "Years_01_16.gdb"
        statistics = ["mean", "max"]

        rp.export_dfrows_to_gdb(dataDF=annualSum, idRaster=idRaster, outGDBPath=outGDBPath, GDBName=G
```

The resulting geodatabase will look like this in ArcGIS:

Example 2: Monthly Precipitation Sums

In this example, the monthly precipitation sums for the year 2016 are calculated and exported to ArcGIS.

1. Import radproc

```
In [4]: import radproc as rp
```

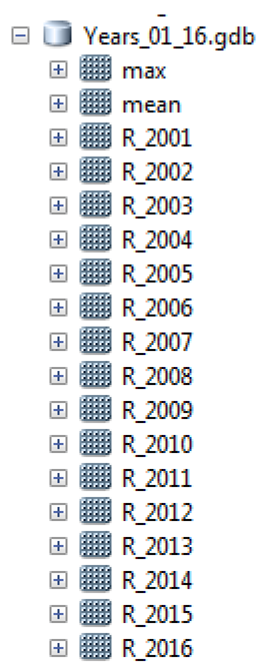


Fig. 1: YearGDB

2. Load Data from HDF5 and Aggregate to Monthly Precipitation Sums

The following function loads precipitation data of the year 2016 from your HDF5 file and generates a DataFrame with monthly precipitation sums for every raster cell.

```
In [5]: HDF = r"O:\Data\RW_2001_2016.h5"

        monthlySum = rp.hdf5_to_months(HDFFile=HDF, year_start=2016, year_end=2016)
        # Display the first five rows of the new DataFrame
        monthlySum.head()
```

```
Out[5]: Rasterzellen-ID               427005     427006     427903     427904  \
        Datum (UTC)
        2016-01-31 00:00:00+00:00  71.400002  71.900002  68.000000  69.099998
        2016-02-29 00:00:00+00:00  68.699997  71.199997  61.500000  64.300003
        2016-03-31 00:00:00+00:00  41.799999  41.299999  40.299999  39.900002
        2016-04-30 00:00:00+00:00  55.700001  56.600002  51.799999  52.400002
        2016-05-31 00:00:00+00:00  38.200001  40.600002  47.299999  44.799999

        Rasterzellen-ID               427905     427906     428803     428804  \
        Datum (UTC)
        2016-01-31 00:00:00+00:00  70.099998  71.699997  65.500000  67.300003
        2016-02-29 00:00:00+00:00  67.300003  68.400002  58.200001  61.700001
        2016-03-31 00:00:00+00:00  40.099998  39.299999  37.700001  38.200001
        2016-04-30 00:00:00+00:00  54.000000  55.299999  50.799999  51.200001
        2016-05-31 00:00:00+00:00  39.400002  34.299999  46.799999  45.400002

        Rasterzellen-ID               428805     428806     ...        661855  \
        Datum (UTC)                                         ...
        2016-01-31 00:00:00+00:00  66.199997  69.800003     ...    128.199997
        2016-02-29 00:00:00+00:00  65.000000  64.300003     ...    105.900002
        2016-03-31 00:00:00+00:00  38.799999  37.500000     ...     65.500000
        2016-04-30 00:00:00+00:00  52.799999  54.400002     ...    108.500000
```

```
2016-05-31 00:00:00+00:00  43.200001  38.500000     ...      121.299995

Rasterzellen-ID                 661856      661857      662752      662753  \
Datum (UTC)
2016-01-31 00:00:00+00:00  129.500000  129.600006  120.199997  121.300003
2016-02-29 00:00:00+00:00  106.199997  110.099998   96.800003   96.699997
2016-03-31 00:00:00+00:00   64.400002   65.199997   70.000000   67.500000
2016-04-30 00:00:00+00:00  112.199997  115.400002  106.700005  106.500000
2016-05-31 00:00:00+00:00  119.300003  115.099998  128.100006  138.199997

Rasterzellen-ID                 662754      662755      662756      663652  \
Datum (UTC)
2016-01-31 00:00:00+00:00  123.500000  125.699997  122.900002  118.099998
2016-02-29 00:00:00+00:00  100.099998  102.599998  101.400002   92.800003
2016-03-31 00:00:00+00:00   64.400002   65.099998   65.099998   69.800003
2016-04-30 00:00:00+00:00  107.699997  112.599998  114.300003  105.199997
2016-05-31 00:00:00+00:00  135.500000  124.300003  128.500000  130.199997

Rasterzellen-ID                 663655
Datum (UTC)
2016-01-31 00:00:00+00:00  120.400002
2016-02-29 00:00:00+00:00   97.400002
2016-03-31 00:00:00+00:00   65.500000
2016-04-30 00:00:00+00:00  108.099998
2016-05-31 00:00:00+00:00  134.399994

[5 rows x 23320 columns]
```

3. Export Results into ArcGIS Geodatabase

The following function exports all rows from the DataFrame calculated above into raster datasets in an ArcGIS File Geodatabase. Optionally, different statistics rasters can be created, in this case the mean, maximum and minimum value of each cell as well as the range per cell.

```
In [6]: idRaster = r"O:\Data\idras"
        outGDBPath = r"O:\Data"
        GDBName = "Months_16.gdb"
        statistics = ["mean", "max", "min", "range"]

        rp.export_dfrows_to_gdb(dataDF=monthlySum, idRaster=idRaster, outGDBPath=outGDBPath, GDBName=
```

The resulting geodatabase will look like this in ArcGIS:

Tutorial 3: Heavy Rainfall Analysis

This tutorial shows how to identify and count heavy rainfall intervals exceeding a specified threshold and export the results to ArcGIS.

Example 1: Identification of Heavy Rainfall Intervals

1. Import radproc

```
In [1]: import radproc as rp
```

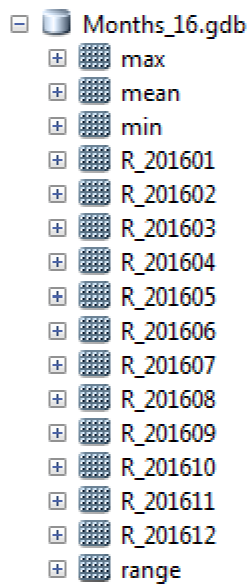


Fig. 2: MonthGDB

2. Identify Heavy Rainfall Intervals

To identify and select all intervals exceeding a rainfall threshold x at least y times in season z, you can use the function

```
find_heavy_rainfalls(HDFFile, year_start, year_end, thresholdValue, minArea, season)
```

The following code will extract all intervals, in which an hourly precipitation of 30 mm is exceeded in at least five cells (these don't need to be adjacent cells!) in May 2016 in Hesse.

```
In [2]: HDF = r"O:\Data\RW.h5"

        hr = rp.find_heavy_rainfalls(HDFFile=HDF, year_start=2016, year_end=2016, thresholdValue=30,
        hr
```

```
Out[2]: Cell-ID                     427005   427006   427903   427904   427905   427906  \
        Date (UTC)
        2016-05-27 16:50:00+00:00      0.0      0.0      0.0      0.0      0.0      0.0
        2016-05-27 17:50:00+00:00      0.0      0.0      0.0      0.0      0.0      0.0
        2016-05-28 14:50:00+00:00      0.0      0.0      0.1      0.1      0.0      0.0
        2016-05-28 15:50:00+00:00      1.2      1.1      1.3      1.2      1.1      1.1
        2016-05-28 17:50:00+00:00      0.0      0.0      0.0      0.0      0.0      0.0
        2016-05-29 16:50:00+00:00      0.0      0.0      0.0      0.0      0.0      0.0
        2016-05-29 23:50:00+00:00      0.0      0.2      0.3      0.2      0.2      0.1
        2016-05-30 00:50:00+00:00      0.0      0.0      0.0      0.0      0.0      0.0

        Cell-ID                     428803   428804   428805   428806     ...     661855  \
        Date (UTC)                                                        ...
        2016-05-27 16:50:00+00:00      0.0      0.0      0.0      0.0     ...        0.0
        2016-05-27 17:50:00+00:00      0.0      0.0      0.0      0.0     ...        0.0
        2016-05-28 14:50:00+00:00      0.0      0.1      0.1      0.0     ...        0.1
        2016-05-28 15:50:00+00:00      1.4      1.1      1.3      1.2     ...        4.7
        2016-05-28 17:50:00+00:00      0.0      0.0      0.0      0.0     ...        4.2
        2016-05-29 16:50:00+00:00      0.0      0.0      0.0      0.0     ...        0.0
        2016-05-29 23:50:00+00:00      0.2      0.1      0.0      0.2     ...        4.9
        2016-05-30 00:50:00+00:00      0.0      0.0      0.0      0.0     ...        4.5
```

```
Cell-ID                      661856  661857  662752  662753  662754  662755  \
Date (UTC)
2016-05-27 16:50:00+00:00       0.0     0.0     0.0     0.0     0.0     0.0
2016-05-27 17:50:00+00:00       0.0     0.0     0.0     0.0     0.0     0.0
2016-05-28 14:50:00+00:00       0.9     1.7     0.4     0.5     0.5     0.6
2016-05-28 15:50:00+00:00       4.0     4.3     9.2     7.8     6.3     3.8
2016-05-28 17:50:00+00:00       4.4     4.3     4.5     5.1     4.5     4.7
2016-05-29 16:50:00+00:00       0.0     0.0     0.0     0.0     0.0     0.0
2016-05-29 23:50:00+00:00       3.9     4.0     4.3     4.4     4.2     4.2
2016-05-30 00:50:00+00:00       3.8     3.5     3.7     3.5     4.2     4.1

Cell-ID                      662756  663652  663655
Date (UTC)
2016-05-27 16:50:00+00:00       0.0     0.0     0.0
2016-05-27 17:50:00+00:00       0.0     0.0     0.0
2016-05-28 14:50:00+00:00       1.1     0.7     0.6
2016-05-28 15:50:00+00:00       3.4    10.6     3.9
2016-05-28 17:50:00+00:00       4.3     5.3     5.2
2016-05-29 16:50:00+00:00       0.0     0.0     0.0
2016-05-29 23:50:00+00:00       3.4     4.3     3.8
2016-05-30 00:50:00+00:00       3.6     4.0     3.6

[8 rows x 23320 columns]
```

3. Export Results into ArcGIS Geodatabase

The following function exports all rows from the resampled daily DataFrame calculated above into raster datasets in an ArcGIS File Geodatabase. Optionally, different statistics rasters can be created, in this case the sum and the maximum value of each cell.

```
In [3]: idRaster = r"O:\Data\idras"
        outGDBPath = r"O:\Data"
        GDBName = "May16_30mm5c.gdb"
        statistics = ["sum", "max"]

        rp.export_dfrows_to_gdb(dataDF=hr, idRaster=idRaster, outGDBPath=outGDBPath, GDBName=GDBName,
```

The new Geodatabase looks like this in ArcGIS:

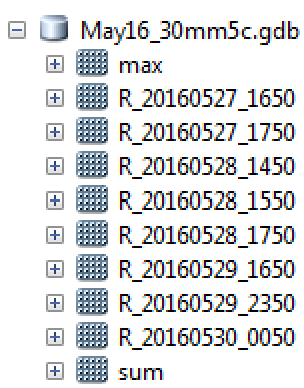


Fig. 3: HR_GDB

Side Note In this example, eight intervals meeting the given criteria have been identified at four days. The following code can be used as a simple approach to obtain daily sums for these events. (Of course this does not take into account that the interval at May 30th is most likely part of the same precipitation event as the ones from May 29th. . . this is a more complicated topic to be adressed in future versions of radproc)

```
In [4]: hr_daily = hr.resample('D').sum()
        hr_daily
```

```
Out[4]: Cell-ID                    427005  427006  427903  427904  427905  427906  \
        Date (UTC)
        2016-05-27 00:00:00+00:00     0.0     0.0     0.0     0.0     0.0     0.0
        2016-05-28 00:00:00+00:00     1.2     1.1     1.4     1.3     1.1     1.1
        2016-05-29 00:00:00+00:00     0.0     0.2     0.3     0.2     0.2     0.1
        2016-05-30 00:00:00+00:00     0.0     0.0     0.0     0.0     0.0     0.0

        Cell-ID                    428803  428804  428805  428806    ...     661855  \
        Date (UTC)                                                   ...
        2016-05-27 00:00:00+00:00     0.0     0.0     0.0     0.0    ...        0.0
        2016-05-28 00:00:00+00:00     1.4     1.2     1.4     1.2    ...        9.0
        2016-05-29 00:00:00+00:00     0.2     0.1     0.0     0.2    ...        4.9
        2016-05-30 00:00:00+00:00     0.0     0.0     0.0     0.0    ...        4.5

        Cell-ID                    661856  661857     662752  662753  662754  \
        Date (UTC)
        2016-05-27 00:00:00+00:00     0.0     0.0   0.000000     0.0     0.0
        2016-05-28 00:00:00+00:00     9.3    10.3  14.099999    13.4    11.3
        2016-05-29 00:00:00+00:00     3.9     4.0   4.300000     4.4     4.2
        2016-05-30 00:00:00+00:00     3.8     3.5   3.700000     3.5     4.2

        Cell-ID                      662755  662756  663652  663655
        Date (UTC)
        2016-05-27 00:00:00+00:00  0.000000     0.0     0.0     0.0
        2016-05-28 00:00:00+00:00  9.099999     8.8    16.6     9.7
        2016-05-29 00:00:00+00:00  4.200000     3.4     4.3     3.8
        2016-05-30 00:00:00+00:00  4.100000     3.6     4.0     3.6

        [4 rows x 23320 columns]
```

For example, the resulting raster dataset for the sum of the two intervals on May 27th looks like this:

Example 2: Counting Heavy Rainfall Intervals

1. Identify and Count Heavy Rainfall Intervals

To count the number of times in which a rainfall threshold x is exceeded at every cell in season z, you can use the function

```
count_heavy_rainfall_intervals(HDFFile, year_start, year_end, thresholdValue, minArea,
↪ season)
```

If you specify a minimum area a > 1, only intervals in which the threshold x is exceeded in at least y cells are taken into account.

The following code will count how many times an hourly precipitation of 10 mm is exceeded at every cell in May 2016 in Hesse.

```
In [5]: HDF = r"O:\Data\RW.h5"

        hr_count = rp.count_heavy_rainfall_intervals(HDFFile=HDF, year_start=2016, year_end=2016, thr
        hr_count
```

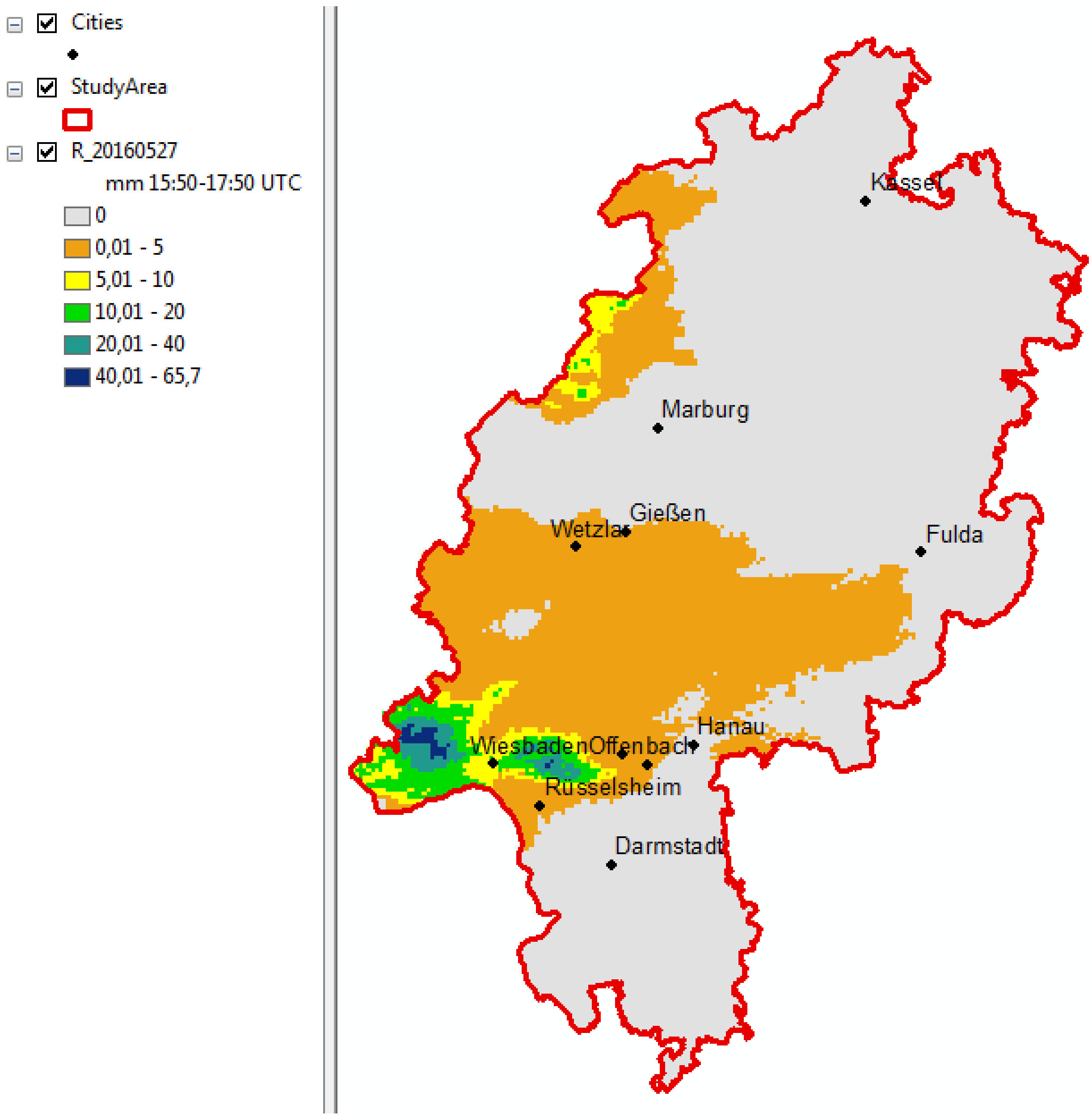


Fig. 4: HR_20160527

```
Out[5]: Cell-ID                    427005  427006  427903  427904  427905  427906  \
        Date (UTC)
        2016-05-31 00:00:00+00:00       0       0       0       0       0       0

        Cell-ID                    428803  428804  428805  428806   ...    661855  \
        Date (UTC)                                                  ...
        2016-05-31 00:00:00+00:00       0       0       0       0   ...         1

        Cell-ID                    661856  661857  662752  662753  662754  662755  \
        Date (UTC)
        2016-05-31 00:00:00+00:00       1       0       1       1       1       1

        Cell-ID                    662756  663652  663655
        Date (UTC)
        2016-05-31 00:00:00+00:00       1       2       1

        [1 rows x 23320 columns]
```

```
In [6]: # print the maximum value
        hr_count.max(axis=1)
```

```
Out[6]: Date (UTC)
        2016-05-31 00:00:00+00:00    4
        Freq: M, dtype: int32
```

2. Export to Raster

As the output DataFrame only has one row (because we only analyzed one month), it can be exported directly with

```
export_to_raster(series, idRaster, outRaster)
```

```
In [7]: rp.export_to_raster(series=hr_count, idRaster=r"O:\Data\idras", outRaster=r"O:\Data\hrcount1(
```

```
Out[7]: 'O:\\Data\\hrcount10mm'
```

1.3.3 Library Reference

Raw Data Processing

Functions for raw data processing.

Unzip, import, clip and convert RADOLAN raw data and write DataFrames to HDF5.

unzip_RW_binaries	Unzips RADOLAN RW binary data saved in monthly .tar or tar.gz archives (e.g.
unzip_YW_binaries	Unzips RADOLAN YW binary data.
radolan_binaries_to_dataframe	Import all RADOLAN binary files in a directory into a pandas DataFrame, optionally clipping the data to the extent of an investigation area specified by an ID array.
radolan_binaries_to_hdf5	Wrapper for radolan_binaries_to_dataframe() to import and **clip all RADOLAN binary files of one month in a directory** into a pandas DataFrame and save the resulting DataFrame as a dataset to an HDF5 file.

Continued on next page

Table 1 – continued from previous page

`create_idraster_and_process_radolan_data`	Convert all RADOLAN binary data in directory tree into an HDF5 file with monthly DataFrames for a given study area.
`process_radolan_data`	Converts all RADOLAN binary data into an HDF5 file with monthly DataFrames for a given study area without generating a new ID raster.

radproc.raw.unzip_RW_binaries

radproc.raw.**unzip_RW_binaries**(*zipFolder*, *outFolder*)

Unzips RADOLAN RW binary data saved in monthly .tar or tar.gz archives (e.g. RWrea_200101.tar.gz, RWrea_200102.tar.gz).

If necessary, extracted binary files are zipped to .gz archives to save memory space on disk. Creates directory tree of style

<outFolder>/<year>/<month>/<binaries with hourly data as .gz files>

Parameters

zipFolder [string] Path of directory containing RW data as monthly tar / tar.gz archives to be unzipped. Archive names must contain year and month at end of basename: RWrea_200101.tar or RWrea_200101.tar.gz

outFolder [string] Path of output directory.

Returns

No return value

radproc.raw.unzip_YW_binaries

radproc.raw.**unzip_YW_binaries**(*zipFolder*, *outFolder*)

Unzips RADOLAN YW binary data. Data have to be saved in monthly .tar or tar.gz archives (e.g. YWrea_200101.tar.gz, YWrea_200102.tar.gz), which contain daily archives with binary files.

If necessary, extracted binary files are zipped to .gz archives to save memory space on disk. Creates directory tree of style

<outFolder>/<year>/<month>/<binaries with data in temporal resolution of 5 minutes as .gz files>

Parameters

zipFolder [string] Path of directory containing YW data as monthly tar / tar.gz archives to be unzipped. Archive names must contain year and month at end of basename: YWrea_200101.tar or YWrea_200101.tar.gz

outFolder [string] Path of output directory.

Returns

No return value

radproc.raw.radolan_binaries_to_dataframe

`radproc.raw.`**`radolan_binaries_to_dataframe`**(*inFolder*, *idArr=None*)

Import all RADOLAN binary files in a directory into a pandas DataFrame, optionally clipping the data to the extent of an investigation area specified by an ID array.

Parameters

inFolder [string] Path to the directory containing RADOLAN binary files. All files ending with '-bin' or '-bin.gz' are read in. The input folder path does not need to have any particular directory structure.

idArr [one-dimensional numpy array (optional, default: None)] containing ID values to select RADOLAN data of the cells located in the investigation area. If no idArr is specified, the ID array is automatically generated from RADOLAN metadata and RADOLAN precipitation data are not clipped to any investigation area.

Returns

(df, metadata) [tuple with two elements:]

df [pandas DataFrame containing. . .]

- RADOLAN data of the cells located in the investigation area
- datetime row index with defined frequency depending on the RADOLAN product and time zone UTC
- ID values as column names

metadata [dictionary] containing metadata from the last imported RADOLAN binary file

Format description and examples

Every row of the output DataFrame equals a precipitation raster of the investigation area at the specific date. Every column equals a time series of the precipitation at a specific raster cell.

Data can be accessed and sliced with the following Syntax:

df.loc[row_index, column_name]

with row index as string in date format 'YYYY-MM-dd hh:mm' and column names as integer values

Examples::

```
>>> df.loc['2008-05-01 00:50',414773] #--> returns single float value of␣
↪specified date and cell
>>> df.loc['2008-05-01 00:50', :] #--> returns entire row (= raster) of␣
↪specified date as one-dimensional DataFrame
>>> df.loc['2008-05-01', :] #--> returns DataFrame with all rows of␣
↪specified day (because time of day is omitted)
>>> df.loc[, 414773] #--> returns time series of the specified cell as␣
↪Series
```

radproc.raw.radolan_binaries_to_hdf5

`radproc.raw.`**`radolan_binaries_to_hdf5`**(*inFolder*, *HDFFile*, *idArr=None*, *complevel=9*)

Wrapper for radolan_binaries_to_dataframe() to import and **clip all RADOLAN binary files of one month in**

a directory into a pandas DataFrame and save the resulting DataFrame as a dataset to an HDF5 file. The name for the HDF5 dataset is derived from the names of the input folder (year and month).

Parameters

inFolder [string] Path to the directory containing RADOLAN binary files. As the function derives the HDF5 group and dataset names from the directory path, the latter is expected to have the following format:

```
>>> inFolder = "C:/path/to/binaryData/YYYY/MM"  # --> e.g. C:/Data/RADOLAN/2008/5
```

In this example for May 2008, the output dataset will have the path '2008/5' within the HDF5 file.

HDFFile [string] Path and name of the HDF5 file. If the specified HDF5 file already exists, the new dataset will be appended; if the HDF5 file doesn't exist, it will be created.

idArr [one-dimensional numpy array (optional, default: None)] containing ID values to select RADOLAN data of the cells located in the investigation area. If no idArr is specified, the ID array is automatically generated from RADOLAN metadata and RADOLAN precipitation data are not clipped to any investigation area.

complevel [interger (optional, default: 9)] defines the level of compression for the output HDF5 file. complevel may range from 0 to 9, where 9 is the highest compression possible. Using a high compression level reduces data size significantly, but writing data to HDF5 takes more time and data import from HDF5 is slighly slower.

Returns

No return value

Function creates dataset in HDF5 file specified in parameter HDFFile.

radproc.raw.create_idraster_and_process_radolan_data

`radproc.raw.`**`create_idraster_and_process_radolan_data`**(*inFolder*, *HDFFile*, *clipFeature=None*, *complevel=9*)

Convert all RADOLAN binary data in directory tree into an HDF5 file with monthly DataFrames for a given study area.

First, an ID raster is generated and - if you specified a Shapefile or Feature-Class as clipFeature - clipped to your study area. The national ID Raster (idras_ger) and the clipped one (idras) are saved in directory of HDF5 file.

Afterwards, all RADOLAN binary files in a directory tree are imported, clipped to study area, converted into monthly pandas DataFrames and stored in an HDF5 file.

The names for the HDF5 datasets are derived from the names of the input folders (year and month). The directory tree containing the raw binary RADOLAN data is expected to have the following format:

<inFolder>/<year>/<month>/<binaries with RADOLAN data>

–> *<inFolder>*/YYYY/MM

–> *<inFolder>*/2008/5 or *<inFolder>*/2008/05

–> e.g. *C:/Data/RW*/2008/5

In this example, the output dataset will have the path 2008/5 within the HDF5 file. The necessary format is automatically generated by the functions `radproc.raw.unzip_RW_binaries()` and `radproc.raw.unzip_YW_binaries()`.

If necessary, a textfile containing all directories which could not be processed due to data format errors is created in directory of HDF5 file.

Parameters

inFolder [string] Path to the **directory tree** containing RADOLAN binary files. The directory tree is expected to have the following structure: <*inFolder*>/YYYY/MM –> *C:/Data/RADOLAN*/2008/5

HDFFile [string] Path and name of the HDF5 file. If the specified HDF5 file already exists, the new dataset will be appended; if the HDF5 file doesn't exist, it will be created.

clipFeature [string (optional, default: None)] Path to the clip feature defining the extent of the study area. File type may be Shapefile or Feature Class. The clip Feature does not need to be provided in the RADOLAN projection. See below for further details. Default: None (Data are not clipped to any study area)

complevel [interger (optional, default: 9)] defines the level of compression for the output HDF5 file. complevel may range from 0 to 9, where 9 is the highest compression possible. Using a high compression level reduces data size significantly, but writing data to HDF5 takes more time and data import from HDF5 is slighly slower.

Returns

No return value Function creates datasets for every month in HDF5 file specified in parameter HDFFile. Additionally, two ID Rasters and - if necessary - a textfile containing all directories which could not be processed due to data format errors are created in directory of HDF5 file.

Note

See also:

See *File system description* for further details on data processing. If you already have an ID Array available, use `radproc.raw.process_radolan_data()` instead.

Note: The RADOLAN data are provided in a custom stereographic projection defined by the DWD and both ID rasters will automatically be generated in this projection by this function. As there is no transformation method available yet, it is not possible to directly perform any geoprocessing tasks with RADOLAN and geodata with other spatial references. Nevertheless, ArcGIS is able to perform a correct on-the-fly transformation to display the data together. The clip function implemented in radproc uses this as a work-around solution to "push" the clip feature into the RADOLAN projection. Hence, the clipping works with geodata in different projections, but the locations of the cells might be slightly inaccurate.

radproc.raw.process_radolan_data

`radproc.raw.`**`process_radolan_data`**(*inFolder*, *HDFFile*, *idArr=None*, *complevel=9*)

Converts all RADOLAN binary data into an HDF5 file with monthly DataFrames for a given study area without generating a new ID raster.

All RADOLAN binary files in a directory tree are imported, clipped to study area, converted into monthly pandas DataFrame and stored in an HDF5 file.

The names for the HDF5 datasets are derived from the names of the input folders (year and month). The directory tree containing the raw binary RADOLAN data is expected to have the following format:

<inFolder>/<year>/<month>/<binaries with RADOLAN data>

–> *<inFolder>*/YYYY/MM

–> *C:/Data/RADOLAN*/2008/5

In this example, the output dataset will have the path 2008/5 within the HDF5 file.

Additionally, a textfile containing all directories which could not be processed due to data format errors is created in directory of HDF5 file.

Parameters

inFolder [string] Path to the directory containing RADOLAN binary files stored in directory tree of following structure:: *<inFolder>*/YYYY/MM –> *C:/Data/RADOLAN*/2008/5

HDFFile [string] Path and name of the HDF5 file. If the specified HDF5 file already exists, the new dataset will be appended; if the HDF5 file doesn't exist, it will be created.

idArr [one-dimensional numpy array (optional, default: None)] containing ID values to select RADOLAN data of the cells located in the investigation area. If no idArr is specified, the ID array is automatically generated from RADOLAN metadata and RADOLAN precipitation data are not clipped to any investigation area.

complevel [interger (optional, default: 9)] defines the level of compression for the output HDF5 file. complevel may range from 0 to 9, where 9 is the highest compression possible. Using a high compression level reduces data size significantly, but writing data to HDF5 takes more time and data import from HDF5 is slighly slower.

Returns

No return value Function creates datasets for every month in HDF5 file specified in parameter HDFFile. Additionally, a textfile containing all directories which could not be processed due to data format errors is created in HDFFolder.

Notes

See *File system description* for further details on data processing.

Core Functions and Data I/O

Core functions like coordinate conversion and import of ID-array from textfile. Data import from HDF5-file and temporal data aggregation.

`coordinates_degree_to_stereographic`	Converts geographic coordinates [°] to cartesian coordinates [km] in stereographic RADOLAN projection.
`save_idarray_to_txt`	Write cell ID values to text file.
`import_idarray_from_txt`	Imports cell ID values from text file into one-dimensional numpy-array.
`load_months_from_hdf5`	Imports the specified months of one year and merges them to one DataFrame.
`load_month`	Imports the dataset of specified month from HDF5.

Continued on next page

Table 2 – continued from previous page

load_years_and_resample	Imports all months of the specified years, merges them together to one DataFrame and resamples the latter to [annual | monthly | daily | hourly] precipitation sums.
hdf5_to_years	Wrapper for load_years_and_resample() to import all months of the specified years, merge them together to one DataFrame and resample the latter to annual precipitation sums.
hdf5_to_months	Wrapper for load_years_and_resample() to import all months of the specified years, merge them together to one DataFrame and resample the latter to monthly precipitation sums.
hdf5_to_days	Wrapper for load_years_and_resample() to import all months of the specified years, merge them together to one DataFrame and resample the latter to daily precipitation sums.
hdf5_to_hours	Wrapper for load_years_and_resample() to import all months of the specified years, merge them together to one DataFrame and resample the latter to hourly precipitation sums.
hdf5_to_hydrologicalSeasons	Calculates the precipitation sums of the hydrological summer and winter seasons (May - October and November - April).

radproc.core.coordinates_degree_to_stereographic

radproc.core.**coordinates_degree_to_stereographic**(*Lambda_degree*, *Phi_degree*)

Converts geographic coordinates [°] to cartesian coordinates [km] in stereographic RADOLAN projection.

Parameters

Lambda_degree [float] Degree of latitude [°N / °S]

Phi_degree [Float] Degree of longitude [°E / °W]

Returns

(x, y) [Tuple with two elements of type float] Cartesian coordinates x and y in stereographic projection [km]

radproc.core.save_idarray_to_txt

radproc.core.**save_idarray_to_txt**(*idArr*, *txtFile*)

Write cell ID values to text file.

Parameters

idArr [one-dimensional numpy array] containing ID values of dtype int32

txtFile [string] Path and name of a new textfile to write cell ID values into. Writing format: One value per line.

Returns

No return value

radproc.core.import_idarray_from_txt

radproc.core.**import_idarray_from_txt**(*txtFile*)

Imports cell ID values from text file into one-dimensional numpy-array.

Parameters

txtFile [string] Path to a textfile containing cell ID values. Format: One value per line.

Returns

idArr : one-dimensional numpy-array of dtype int32

radproc.core.load_months_from_hdf5

radproc.core.**load_months_from_hdf5**(*HDFFile, year, months=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]*)

Imports the specified months of one year and merges them to one DataFrame.

Parameters

HDFFile [string] Path and name of the HDF5 file containing monthly datasets.

year [integer] Year for which data are to be loaded.

months [list of integers (optional, default: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])] Months for which data are to be loaded.

Returns

df : pandas DataFrame

radproc.core.load_month

radproc.core.**load_month**(*HDFFile*, *year*, *month*)

Imports the dataset of specified month from HDF5.

Parameters

HDFFile [string] Path and name of the HDF5 file containing monthly datasets.

year [integer] Year for which data are to be loaded.

month [integer] Month for which data are to be loaded.

Returns

df : pandas DataFrame

radproc.core.load_years_and_resample

radproc.core.**load_years_and_resample**(*HDFFile*, *year_start*, *year_end=0*, *freq='years'*)

Imports all months of the specified years, merges them together to one DataFrame and resamples the latter to [annual | monthly | daily | hourly] precipitation sums.

Parameters

HDFFile [string] Path and name of the HDF5 file containing monthly datasets.

year_start [integer] First year for which data are to be loaded.

year_end [integer (optional, default: start_year)] Last year for which data are to be loaded.

freq [string (optional, default: "years")] Target frequency. Available frequencies for downsampling:

"years", "months", "days", "hours"

Returns

df [pandas DataFrame] resampled to the target frequency and containing [annual | monthly | daily | hourly] precipitation sums.

Examples

The mean annual precipitation sum can be calculated with the following syntax:

```
>>> import radproc as rp
>>> meanPrecip = rp.load_years_and_resample(r"C:\Data\RADOLAN.h5", 2010, 2015,
↪"years").mean()
# The resulting pandas Series can be exported to an ESRI Grid:
>>> rp.export_to_raster(series=meanPrecip, idRaster=rp.import_idarray_from_
↪raster(r"C:\Data\idras"), outRaster=r"P:\GIS_data\N_mean10_15")
```

Note: All resampling functions set the label of aggregated intervals at the right, hence every label describes the precipitation accumulated in the previous interval period.

radproc.core.hdf5_to_years

radproc.core.**hdf5_to_years**(*HDFFile*, *year_start*, *year_end=0*)

Wrapper for load_years_and_resample() to import all months of the specified years, merge them together to one DataFrame and resample the latter to annual precipitation sums.

Parameters

HDFFile [string] Path and name of the HDF5 file containing monthly datasets.

year_start [integer] First year for which data are to be loaded.

year_end [integer (optional, default: start_year)] Last year for which data are to be loaded.

Returns

df [pandas DataFrame] resampled to annual precipitation sums.

Note: All resampling functions set the label of aggregated intervals at the right, hence every label describes the precipitation accumulated in the previous interval period.

radproc.core.hdf5_to_months

`radproc.core.hdf5_to_months(HDFFile, year_start, year_end=0)`

Wrapper for load_years_and_resample() to import all months of the specified years, merge them together to one DataFrame and resample the latter to monthly precipitation sums.

Parameters

HDFFile [string] Path and name of the HDF5 file containing monthly datasets.

year_start [integer] First year for which data are to be loaded.

year_end [integer (optional, default: year_start)] Last year for which data are to be loaded.

Returns

df [pandas DataFrame] resampled to monthly precipitation sums.

Note: All resampling functions set the label of aggregated intervals at the right, hence every label describes the precipitation accumulated in the previous interval period.

radproc.core.hdf5_to_days

`radproc.core.hdf5_to_days(HDFFile, year_start, year_end=0)`

Wrapper for load_years_and_resample() to import all months of the specified years, merge them together to one DataFrame and resample the latter to daily precipitation sums.

Parameters

HDFFile [string] Path and name of the HDF5 file containing monthly datasets.

year_start [integer] First year for which data are to be loaded.

year_end [integer (optional, default: start_year)] Last year for which data are to be loaded.

Returns

df [pandas DataFrame] resampled to daily precipitation sums.

Note: All resampling functions set the label of aggregated intervals at the right, hence every label describes the precipitation accumulated in the previous interval period.

radproc.core.hdf5_to_hours

`radproc.core.hdf5_to_hours(HDFFile, year_start, year_end=0)`

Wrapper for load_years_and_resample() to import all months of the specified years, merge them together to one DataFrame and resample the latter to hourly precipitation sums.

Parameters

HDFFile [string] Path and name of the HDF5 file containing monthly datasets.

year_start [integer] First year for which data are to be loaded.

year_end [integer (optional, default: start_year)] Last year for which data are to be loaded.

Returns

df [pandas DataFrame] resampled to hourly precipitation sums.

Note: All resampling functions set the label of aggregated intervals at the right, hence every label describes the precipitation accumulated in the previous interval period.

Note: For comparisons between hourly RW data and gauge data/YW data resampled to hours, keep in mind, that hours in RW always start at hh-1:50 whereas the resampled hours begin at hh:00.

---

radproc.core.hdf5_to_hydrologicalSeasons

`radproc.core.hdf5_to_hydrologicalSeasons(HDFFile, year_start, year_end=0)`

Calculates the precipitation sums of the hydrological summer and winter seasons (May - October and November - April).

Imports all months of the specified years, resamples them to monthly precipitation sums, merges them together to one DataFrame and resamples the latter to half-annual precipitation sums. Note: The Data are truncated to the period May of year_start to October of year_end before resampling!

Parameters

HDFFile [string] Path and name of the HDF5 file containing monthly datasets.

year_start [integer] First year for which data are to be loaded. The months January to April of this year are not contained in the precipitation sums!

year_end [integer (optional, default: start_year)] Last year for which data are to be loaded. The months November and December of this year are not contained in the precipitation sums!

Returns

df [pandas DataFrame] resampled to precipitation sums of hydrological summer and winter seasons. In contrast to most other resampling functions from radproc, the index labels the beginning of each resampling period, e.g. the index 2001-05-01 describes the period from May to October 2001.

Note: All resampling functions set the label of aggregated intervals at the right, hence every label describes the precipitation accumulated in the previous interval period.

Heavy Rainfall Analysis

Module for heavy rainfall analysis.

- identify and select all intervals in which a specified precipitation threshold is exceeded
- count the number of threshold exceedances

find_heavy_rainfalls	Creates a DataFrame containing all heavy rainfalls (intervals) exceeding a specified threshold intensity value.
count_heavy_rainfall_intervals	Creates a DataFrame containing the sum of all heavy rainfalls intervals exceeding a specified threshold intensity value.

radproc.heavyrain.find_heavy_rainfalls

radproc.heavyrain.**find_heavy_rainfalls**(*HDFFile*, *year_start*, *year_end*, *thresholdValue*, *minArea*, *season*)

Creates a DataFrame containing all heavy rainfalls (intervals) exceeding a specified threshold intensity value.

- rainfall intensity
- minimum area (number of cells) where intensity has to be exceeded
- season / time period

Parameters

HDFFile [string] Path and name of the HDF5 file containing monthly pandas DataFrames with precipitation data.

year_start [integer] First year for which data are to be loaded.

year_end [integer] Last year for which data are to be loaded.

thresholdValue [integer] Rainfall intensity threshold value.

minArea [integer] Minimum area where intensity threshold value has to be exceeded.

season [string or list] Season / Time period to analyse. Can be a list with integer values from 1 to 12 or a string describing the season. The following strings are possible: ["Year" | "May - October" | "November - April" | "January/December" | "Jan" | "Feb" | "Mar" | "Apr" | "May" | "Jun" | "Jul" | "Aug" | "Sep" | "Oct" | "Nov" | "Dec"]

Returns

heavy_rains [pandas DataFrame] containing all intervals meeting the given criteria.

radproc.heavyrain.count_heavy_rainfall_intervals

radproc.heavyrain.**count_heavy_rainfall_intervals**(*HDFFile*, *year_start*, *year_end*, *thresholdValue*, *minArea*, *season*)

Creates a DataFrame containing the sum of all heavy rainfalls intervals exceeding a specified threshold intensity value.

Search parameters are

- rainfall intensity
- minimum area (number of cells) where intensity has to be exceeded
- season / time period

Parameters

HDFFile [string] Path and name of the HDF5 file containing monthly pandas DataFrames with precipitation data.

year_start [integer] First year for which data are to be loaded.

year_end [integer] Last year for which data are to be loaded.

thresholdValue [integer] Rainfall intensity threshold value.

minArea [integer] Minimum area (number of cells) where intensity threshold value has to be exceeded.

season [string or list] Season / Time period to analyse. Can be a list with integer values from 1 to 12 or a string describing the season. The following strings are possible: ["Year" | "May - October" | "November - April" | "January/December" | "Jan" | "Feb" | "Mar" | "Apr" | "May" | "Jun" | "Jul" | "Aug" | "Sep" | "Oct" | "Nov" | "Dec"]

Returns

interval_count [pandas DataFrame] containing the sum of all intervals meeting the given criteria. Temporal resolution depending on the given season.

RADOLAN Binary File Import

Copy of all functions necessary for reading in RADOLAN composite files, taken from module wradlib.io (version=0.9.0)

(Heistermann, M., Jacobi, S., and Pfaff, T. 2013: Technical Note: An open source library for processing weather radar data (wradlib), Hydrol. Earth Syst. Sci., 17, 863-871)

Copying these functions was necessary in order to reduce number of dependencies and avoid conflicts arising between different GDAL installations required for other wradlib modules and ArcGIS.

read_RADOLAN_composite	Read quantitative radar composite format of the German Weather Service

radproc.wradlib_io.read_RADOLAN_composite

radproc.wradlib_io.**read_RADOLAN_composite**(*fname*, *missing=-9999*, *loaddata=True*)

Read quantitative radar composite format of the German Weather Service

The quantitative composite format of the DWD (German Weather Service) was established in the course of the RADOLAN project and includes several file types, e.g. RW, YW, RY, RX, RZ, and many, many more. (see format description on the RADOLAN project homepage).

Originally, the national RADOLAN composite was a 900 x 900 grid with 1 km resolution and in polar-stereographic projection. But for the reanalysis within the radar climatology project finished in 2017 the national grid was extended to 1100 x 900 cells. There are other grid resolutions for other composites, too (e.g. PC, PG).

This function already evaluates and applies the so-called PR factor which is specified in the header section of the RADOLAN files. The raw values in an RY oder YW file are in the unit 0.01 mm/5min, while read_RADOLAN_composite returns values in mm/5min (i. e. factor 100 higher). The factor is also returned as part of attrs dictionary under keyword "precision".

fname [string] path to the composite file

missing [int] value assigned to no-data cells

loaddata [bool] True | False, If False function returns (None, attrs)

output [tuple] tuple of two items (data, attrs):

- data : numpy array of shape (number of rows, number of columns)
- attrs : dictionary of metadata information from the file header

DWD MR90 Gauge Data Processing

Collection of functions for processing DWD rain gauge data in MR90 format.

Convert gauge data to pandas DataFrames with same format as RADOLAN data and saves them as HDF5 datasets.

stationfile_to_df	Import a textfile with DWD rain gauge data in MR90 format into a one-column pandas DataFrame.
summarize_metadata_files	Import all metafiles and summarizes metadata in a single textfile.
dwd_gauges_to_hdf5	Import all textfiles containing DWD rain gauge data in MR90 format from input folder into a DataFrame and save it as monthly HDF5 datasets.

radproc.dwd_gauge.stationfile_to_df

radproc.dwd_gauge.**stationfile_to_df**(*stationfile*)

Import a textfile with DWD rain gauge data in MR90 format into a one-column pandas DataFrame.

Downsample frequency from 1 to 5-minute intervals to adjust temporal resolution to best-resolved RADOLAN data produt YW. Convert time zone to UTC.

Parameters

stationfile [string] Path and name of textfile containing rain gauge measurements.

Returns

df [one-column pandas DataFrame] with data imported from stationfile downsampled to 5-minute intervals.

radproc.dwd_gauge.summarize_metadata_files

`radproc.dwd_gauge.`**`summarize_metadata_files`**(*inFolder*)

Import all metafiles and summarizes metadata in a single textfile.

Metadata include information on station number and name, geographic coordinates and height above sea level.

Parameters

inFolder [string] Path of directory containing metadata files for DWD gauges.

Returns

summaryFile [string] Path and name of output summary file created.

radproc.dwd_gauge.dwd_gauges_to_hdf5

`radproc.dwd_gauge.`**`dwd_gauges_to_hdf5`**(*inFolder*, *HDFFile*)

Import all textfiles containing DWD rain gauge data in MR90 format from input folder into a DataFrame and save it as monthly HDF5 datasets.

Frequency is downsampled from 1 to 5-minute intervals to adjust temporal resolution to RADOLAN product YW. Time zone is converted from MEZ to UTC.

Parameters

inFolder [string] Path of directory containing textfiles with DWD rain gauge data in MR90 format.

HDFFile [string] Path and name of the HDF5 file. If the specified HDF5 file already exists, the new dataset will be appended; if the HDF5 file doesn't exist, it will be created.

Returns

None Save monthly DataFrames to specified HDF5 file.

Note

To import gauge data from HDF5, you can use the same functions from radproc.core as for RADOLAN data since both are stored the same data format and structure.

1.3.4 Indices and tables

- genindex
- modindex
- search

Python Module Index

r

Index

C

D

F

H

I

L

P

R

S

U