wradlib – An Open Source Library for Weather Radar Data Processing

Thomas
Pfaff

Thomas Pfaff[1], Maik Heistermann[2], Stephan Jacobi[2]

[1]*Institute for Modelling of Water and Environmental Systems, Pfaffenwaldring 61, 70569 Stuttgart, Germany, thomas.pfaff@iws.uni-stuttgart.de*

[2]*Institute of Earth and Environmental Science, Karl-Liebknecht-Str. 24-25, 14476 Potsdam-Golm, Germany, heisterm@uni-potsdam.de, stjacobi@uni-potsdam.de*

(Dated: 27 May 2012)

1. Introduction

Observation data from weather radars is deemed to have great potential in hydrology and meteorology for forecasting of severe weather or floods in small and urban catchments by providing high resolution measurements of precipitation. With the time series from operational weather radar networks attaining lengths suitable for the long term calibration of hydrological models, the interest in using this data is growing. There are, however, several challenges impeding a widespread use of weather radar data.

The first being a multitude of different file formats for data storage and exchange. Although the OPERA project [1] has taken steps towards harmonizing the data exchange inside the European radar network [2], different dialects still exist in addition to a large variety of legacy formats.

The second barrier is what we would like to describe as the product dilemma. A great number of potential applications also implies a great number of different and often competing requirements as to the quality of the data. As an example, while one user might be willing to accept more false alarms in a clutter detection filter, e.g. to avoid erroneous data assimilation results, another might want a more conservative product in order not to miss the small scale convection that leads to a hundred year flood in a small head catchment. A single product from a radar operator, even if created with the best methods currently available, will never be able to accommodate all these needs simultaneously. Thus the product will either be a compromise or it will accommodate one user more than the other. Often the processing chain needs to be in a specific order, where a change of a certain processing step is impossible without affecting the results of all following steps. Sometimes some post-processing of the product might be possible, but if not, the user’s only choice is to either take the product as is or leave it.

If a user should decide that he would take a raw radar product and try to do customized corrections, he is faced with basically reinventing the wheel, writing routines for data I/O, error corrections, georeferencing and visualization, trying to find relevant literature and to extract algorithms from publications, which, often enough, do not offer all implementation details. This takes a lot of time and effort, which could be used much more effectively, if standard algorithms were available in a well documented and easy to use manner.

wradlib is intended to provide this collection of routines and algorithms in order to facilitate and foster the use of weather radar data in as many disciplines as possible including research, application development and teaching.

2. Development Concepts

This section will present an overview of the concepts that are used to provide an efficient and easy to use library with a level of quality necessary for applications in scientific and operational contexts, as well as facilities to interact and collaborate on code and methods.

2.1 Licensing

Since the advent of the internet, society has benefited greatly from open source and community projects, the most notable being the Linux operating system or the PostgreSQL database system. There are also many open source libraries like e.g. the LAPACK [3] linear algebra libraries, which provide significant contributions to scientific software development. The permissive MIT License [4] is used to ensure that students, researchers or companies who contributed to *wradlib* may continue to use their work even for commercial purposes.

2.2 Programming Languages

So far, the main implementation language of *wradlib* is Python [5]. Python is an open source, high level interpreted language with an extensive built-in standard library. It has a clear syntax, is well documented and easy to learn.

Due to its interpreted nature, some operations are not as high-performing as in compiled languages. There is however a mature and well maintained stack of packages like NumPy [6] and SciPy [7] that interface to code optimized for numerical and scientific calculations. Packages like matplotlib [8] and IPython [9] add presentation quality visualization and interactive processing capabilities. Tools to interface with compiled code are available as well, e.g. SWIG [10] or Cython [11] for C/C++, or f2py (Peterson 2009) [12] for FORTRAN (see also [13] for more information on using Python as an integration language).

Interfaces to proprietary high level tools like Matlab™ [14, 15, 16] or IDL™ [17, 18] and open environments such as R [19] exist, but on varying levels of development and maintenance. Contributions in any programming language are welcome. The *wradlib* team will provide support on the integration or porting of the code, if necessary.

Building on top of Python makes *wradlib* platform independent provided that package dependencies can be satisfied on the respective platform as well. As the library grows, these dependencies will become more and more. While many Linux distributions provide means to manage dependencies between software packages – like Debian's apt or RedHat's RPM systems – there is no such tool for the Windows operating system. However, there are some scientific Python distributions, which provide a certain set of packages with all dependencies resolved. In order to allow easy installation and usage even on Windows PCs, without actively endorsing this particular distribution, *wradlib* will be maintained to be working with a given version of the python(x,y) [20] distribution. Currently version 2.6.6.1 is supported, but newer versions have been reported to work as well. Some few additional dependencies exist, but these are explicitly given in the documentation.

2.2 Distributed Version Control

Version control is a cornerstone of software development, giving the developer a tool to track changes and undo them, if necessary. Version control and the connected toolsets also enable effective collaboration as integrating code from several developers is possible almost automatically. Distributed version control systems (DVCS) add another level of cooperation as each developer may work with his own repository, having full access to the whole history, merging in changes from others when necessary and pushing out his own changes once these are deemed ready.

wradlib uses the platform independent DVCS mercurial [21]. Starting with mercurial is relatively easy, as is transitioning from older version control systems like subversion due to similar concepts and syntax.

The source code of *wradlib* is hosted at bitbucket [22], a site providing free hosting for open-source projects specialized on mercurial. Bitbucket also offers additional services useful for collaboration and community-building like issue-tracking, RSS and e-mail notifications, interactive comments on changesets, etc.

2.3 Documentation

It is the belief of the authors that the paramount criterion for the usability of a library is its documentation. Documentation should not only enable the potential user to apply the software, but it should also contain key technical and scientific information on the functions employed. Without proper source code documentation, developers – existing and new – will have a harder time to understand what the code actually does or is intended to do, which impedes bug fixing as well as further enhancements. Examples and Tutorials are needed both to guide beginners on their first steps as well as to explain more complex uses, which cannot be covered by the individual function or class documentation. *wradlib* makes extensive use of Python's docstring concept [23] to keep the library documentation as close to the actual code as possible. The software package Sphinx [24] is used to automatically extract these docstrings, to incorporate them in higher level documents (together with examples, tutorials etc.), and finally convert the combined document into presentation formats like html, windows html help or LaTeX. The resulting documentation is then made available for download or can be accessed online.

2.3 Code quality

wradlib's software development principles are oriented along those of other successful open source scientific software packages like NumPy or SciPy. Apart from good documentation, the quality and stability of the code will be assured by a comprehensive set of automated unit tests. Regular meetings of small groups of developers, so called 'sprints', are intended to bring people together to exchange ideas and solve certain problems in a concentrated effort.

The development of the library will be an iterative process. Every algorithm implementation is welcome, regardless of its level of optimization. Improvement will be done, if and when performance becomes an issue.

Table 1 Modules in wradlib

Module	**Purpose**
wradlib.io	Data Input/Output
wradlib.clutter	Clutter identification
wradlib.atten	Attenuation calculations
wradlib.vpr	Vertical profile corrections
wradlib.trafo	Unit transformations
wradlib.zr	Conversions to rainfall rate
wradlib.georef	Georeferencing
wradlib.ipol	Interpolation
wradlib.qual	Measurement quality indicators
wradlib.comp	Composition
wradlib.adjust	Gauge adjustment
wradlib.verify	Verification tools

3. Features by Example

wradlib is divided into several modules according to different steps of a radar processing chain. These modules and the functionality contained within are briefly presented in Table 1. The following sections are intended to give some examples on the functionality of some of the modules. Detailed information can be found in the online documentation at http://wradlib.bitbucket.org.

3.1 Data I/O

The basic data structure used by *wradlib* is the numpy.ndarray. It is a very flexible, yet powerful container for homogeneous data, and has become the standard for most scientific python packages. The wradlib.io module provides functions that load data from files of different formats into numpy.ndarrays. As an example, listing 1 reads the information contained in a DX-product file (one 2-D scan of reflectivity at the lowest elevation following the radar horizon) of the German Weather Service (DWD). It returns the reflectivity in the variable `img` and the data for azimuth and elevation angles per beam in the python dictionary `attrs`.

Listing 1: reading data from a DX file

```
img, attrs = wradlib.io.readDX('/path/to/raa00-dx_10832-0806021720-tur---bin')
```

Currently under implementation are functions to read DWD R-series product files of Cartesian German composite data, Gematronik’s Rainbow version 3 volume files, and the BUFR format. Other formats will be implemented dependent on demand and availability of documentation.

3.2 Error Correction

There are several modules dedicated to the correction of the major sources of error in weather radar data. wradlib.clutter implements functions for clutter detection, wradlib.atten and wradlib.vpr deal with attenuation and errors due to the vertical profile of reflectivity. The function called in listing 2 would identify clutter in a radar image using the filter described in Gabella & Notarpietro (2002) and return the result as a Boolean ndarray.

Listing 2: call to clutter filtering function

```
clutter = wradlib.clutter.filter_gabella(img, tr1=12, n_p=6, tr2=1.1)
```

The behaviour of this filter can be adjusted to the users needs. In this example the three parameters of the filter are set explicitly. If they were not given default values (usually those presented in the respective publication) would have been used.

The Boolean map returned by the function can then be used to correct the original data, for example by interpolation from surrounding bins as shown in listing 3.

Listing 3: using a clutter map to correct data

```
cc = wradlib.ipol.interpolate_polar(img, clutter)
```

For attenuation correction several implementations of the recursive methods presented by Nicol & Austin (2003) and Krämer (2008) exist. An implementation of the method presented by Marzoug & Amayenc (1994) is under way.

3.3 Data Transformations

The wradlib.trafo module contains utility functions to do common unit transformations like converting from and to decibel or transform rainfall intensities to depths depending on the integration interval. Due to the importance and the diversity of algorithms for Z-R-relations, relating functions are collected in the wradlib.zr module. Currently, the standard power-law relation and the 3-part relation currently in operational use by the DWD according to (Bartels et al. 2004) are implemented The clutter corrected dBZ data from Listing 3, for example, can be converted to a 5 minute (300 s) rainfall depth using the code sequence in listing 4.

Listing 4: unit conversion and depth calculation

```
z = wradlib.trafo.idecibel(cc)
i = wradlib.zr.z2r(z)
r = wradlib.trafo.r2depth(i, 300)
```

Additional configuration can be passed to many functions using Python’s keyword mechanism. By default wradlib.zr.z2r uses the parameters a=200 and b=1.6. These can be overridden by explicitly passing them in the function call.

3.4 Georeferencing

Georeferencing is done using the pyproj [25] package, a python wrapper to the well-established PROJ.4 library [26].

Several functions to conveniently call this library, as well as predefined projection strings for common map projections are available. The following code example calculates the radar bin locations of the DWD radar Tuerkheim in Gauss Krueger Zone 3 coordinates based on the polar range and azimuth resolution properties of the radar.

Listing 5: Georeferencing radar data in polar representation

```
1: r = np.arange(1, 129)*1000.
2: az = np.linspace(0, 359, 360)
3: tur_sitecoords = (48.5861, 9.7839)
4: tur_lon, tur_lat = wradlib.georef.polar2centroids(r, az, tur_sitecoords)
5: gk3 = '+proj=tmerc +lat_0=0 +lon_0=9 +k=1 ' + \
6:       '+x_0=3500000 +y_0=0 +ellps=bessel ' + \
7:       '+towgs84=598.1,73.7,418.2,0.202,0.045,-2.455,6.7 ' + \
8:       '+units=m +no_defs'
9: tur_x, tur_y = wradlib.georef.project(tur_lat, tur_lon, gk3)
```

In lines 1 and 2 of listing 5 the scan geometry of the radar is defined consisting of 128 range bins with 1000 m resolution per beam and 360 beams with 1 degree of azimuthal resolution. The location of the radar is defined in line 3 as a latitude/longitude coordinate pair. Line 4 calculates the centroids of all radar bins based on the defined information in lat/lon coordinates using spherical geometry. Lines 5 through 8 define the Gauss-Krueger projection in PROJ.4 syntax and in line 9 the x and y coordinates are calculated. Figure 1 (left) shows the results of the georeferencing and projection for two radars Tuerkheim (tur) and Feldberg (fbg).

3.5 Interpolation and Composition

wradlib currently implements nearest neighbor, inverse distance and linear (barycentric) interpolation. Currently, various Kriging methods are under development. All methods are provided with a unified interface, so that minimal code changes are necessary to switch between approaches.

These interpolation methods are used by several routines like infilling of missing or filtered data or the transferral from polar to Cartesian representations.

Composition of several radar sites to a common grid can be done using different methods to determine the values of pixels in overlapping regions. This can either be an all-or-nothing decision or a weighted combination. The different criteria like beam height or pulse volume are organized in the wradlib.qual module. The right panel of figure 1 shows the result of a composition done using a weighted combination based on measurement volume.

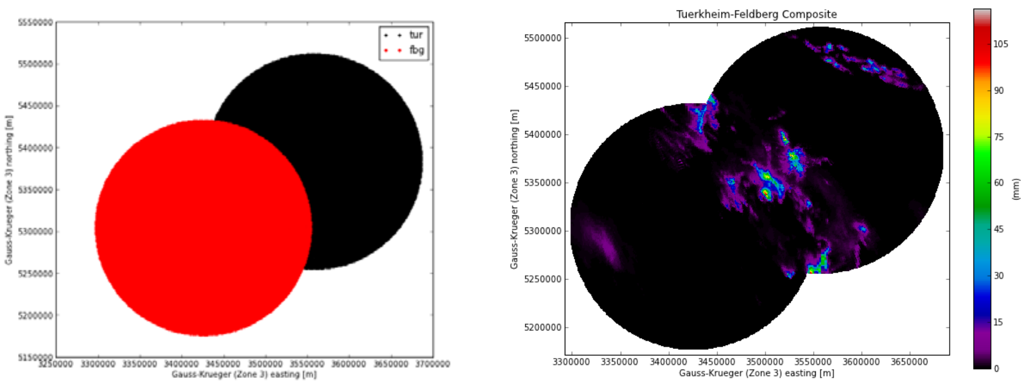

Fig. 1 projected coordinates of radar bins for DWD Radars Tuerkheim(tur) and Feldberg (fbg) (left), composite of 1h accumulation of clutter and attenuation corrected data (right)

3.4 Gauge Adjustment

As gauge adjustment has received increased attention in recent years, and as it is shown to improve at least hydrological modeling results (Heistermann & Kneis, 2011), the wradlib.adjust module has been added in order to provide different adjustment techniques for comparison. At the moment an additive adjustment class has been implemented. We plan to incorporate as many methods as possible in order to enable analyses like that of Goudenhoofdt & Delobbe (2009) for other datasets. Figure 2 shows an example of an additive adjustment performed on a detail of the data presented in figure 1. The current adjustment classes use inverse distance interpolation. As other interpolation methods become available, the classes will be made configurable with respect to which approach is to be used.

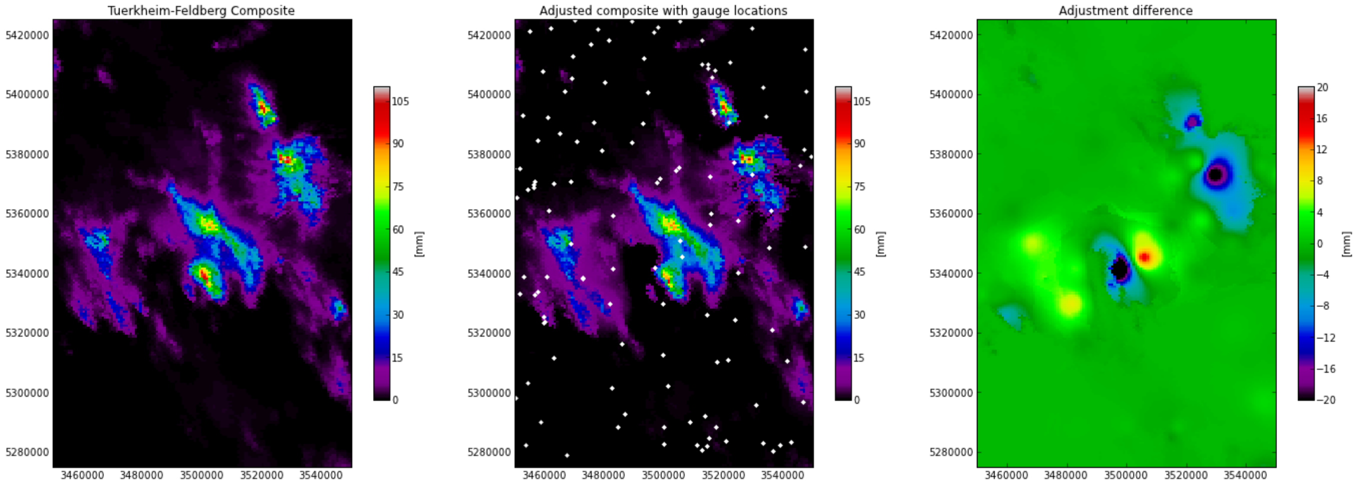

Fig 2 detail of unadjusted (left) adjusted with gauge locations (middle) and difference images of the results of an additive adjustment performed on the data displayed in the right panel of fig. 1

3.5 Visualization

The wradlib.vis module contains several functions and classes to provide both quick diagnostic plots as well as facilities with more control on the plot's appearance. Figure 3 shows a quick diagnostic plot of the DX data mentioned above using the following command of Listing 6. Matplotlib's basemap toolkit will be used to provide support for shape files and mapping based on lat/lon georeferenced data.

Listing 6: Diagnostic plotting of polar data

```
1: wradlib.vis.polar_plot(img)
```

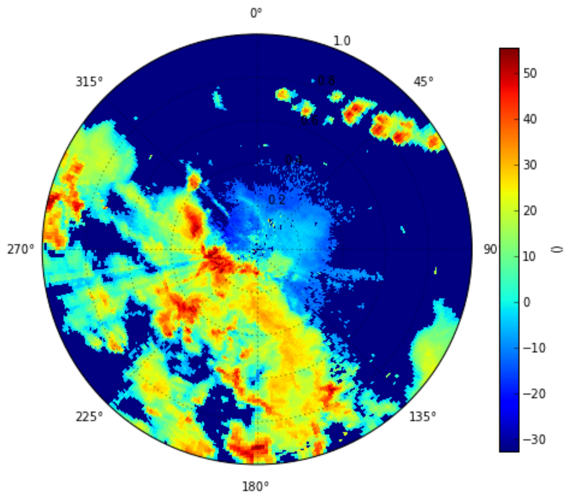

Fig. 3 diagnostic plot of polar reflectivity

5. Outlook

Based on our own experiences when starting out using weather radar data for hydrological modeling, we have implemented an openly available library that, hopefully, over time will provide useful routines for beginners as well as experts. Simple and well documented interfaces should enable users to obtain results quickly, while transparent implementations of common and new algorithms should provide a good basis for comparative studies and further developments. Despite using an interpreted language for development, *wradlib*'s existing algorithms are fast enough to use them for long term analyses and may even work under real time conditions.

Using the capabilities of matplotlib enables interactive as well as presentation quality visualizations of the data, with and without spatial contexts, in the same environment that created the data. Together with the interactive features readily available through the Python interpreter or the IPython shell, *wradlib* may also become a valuable tool for teaching weather radar related topics.

wradlib can also be seen as an approach to increase reproducibility of findings in meteorological and hydrological research. With accessible, well documented and transparent algorithms, results can quickly be verified and possible errors in the original implementation can be identified and removed. New developments can be compared more easily with established methods, if these have been published in an open library like *wradlib*.

Having started the project in October 2011, *wradlib*'s development is still at the beginning. While we have tried to cover the whole radar processing chain, there are only few algorithms for each task to choose from. We hope that this will change in the future as more contributions will be done by us or others. In addition to the development and maintenance of algorithms, we would like to use *wradlib* to build an active community of people working with weather radar data. To enable exchange among its members, we have set up the groups *wradlib-users@googlegroups.com* for questions related to the application of the library and *wradlib-dev@googlegroups.com* for discussions about the development of new features.

We are aware of the fact that *wradlib* is not the only attempt on creating a common, freely available software library for radar data processing. According to [27] there has been work towards a common software library as part of the OPERA project. However, it could not be determined, whether this library is available publicly or only for project members.

Since January 2012, Version 1.0 of the BALTRAD [28] open source distribution and processing software is available, which adheres to similar ideas as *wradlib*. Its scope is wider, including facilities for operational data exchange throughout a radar network. We will check whether and how efforts can be joined, in order to create the most comprehensive and powerful openly available set of tools for the advancement of the use of weather radar data in the earth sciences.

References

Bartels H. et al., 2004: Projekt RADOLAN – Routineverfahren zur Online-Aneichung der Radarniederschlagsdaten mit Hilfe von automatischen Bodenniederschlagsstationen (Ombrometer), Technical Report

Gabella M., Notarpietro R., 2002: Ground clutter characterization and elimination in mountainous terrain. In *Proceedings of ERAD. Delft*, Copernicus GmbH, 305-311

Goudenhoofdt E., Delobbe L., 2009: Evaluation of radar-gauge merging methods for quantitative precipitation estimates. *Hydrology and Earth System Sciences*, **13(2)**, 195-203

Heistermann M., Kneis D., 2011: Benchmarking quantitative precipitation estimation by conceptual rainfall-runoff modelling. *Water Resources Research*, **47(6)**

Krämer S., 2008: Quantitative Radardatenaufbereitung für die Niederschlagsvorhersage und die Siedlungsentwässerung. Dissertation, Gottfried Wilhelm Leibniz Universität Hannover

Marzoug M., Amayenc P., 1994: A Class of Single-and Dual-Frequency Algorithms for Rain-Rate Profiling from a Spaceborne Radar. Part I: Principle and Tests from Numerical Simulations. *Journal of Atmospheric and Oceanic Technology*, **11(6)**, 1480–1506

Nicol J.C., Austin G.L., 2003: Attenuation correction constraint for single-polarisation weather radar. *Meteorological Applications*, **10(4)**, 345-354

Peterson P., 2009: F2PY: a tool for connecting Fortran and Python programs. *International Journal of Computational Science and Engineering*, **4(4)**, 296

URL references (accessed 2012-05-27)

[1] http://www.knmi.nl/opera
[2] http://www.knmi.nl/opera/opera3/OPERA_2008_03_WP2.1b_ODIM_H5_v2.1.pdf
[3] http://www.netlib.org/lapack
[4] http://www.opensource.org/licenses/MIT
[5] http://python.org
[6] http://numpy.scipy.org
[7] http://www.scipy.org
[8] http://matplotlib.sourceforge.net
[9] http://ipython.org
[10] http://www.swig.org
[11] http://cython.org
[12] http://www.scipy.org/F2py
[13] http://docs.scipy.org/doc/numpy/user/c-info.python-as-glue.html
[14] http://mlabwrap.sourceforge.net
[15] http://claymore.engineer.gvsu.edu/~steriana/Python/pymat.html
[16] http://code.google.com/p/python-matlab-wormholes
[17] http://astronomy.sussex.ac.uk/~anthonys/pidly
[18] http://www.cacr.caltech.edu/~mmckerns/pyIDL.html
[19] http://rpy.sourceforge.net/
[20] http://www.pythonxy.com
[21] http://mercurial.selenic.com
[22] https://bitbucket.org
[23] http://en.wikipedia.org/w/index.php?title=Docstring&oldid=469570236
[24] http://sphinx.pocoo.org
[25] http://code.google.com/p/pyproj/
[26] http://trac.osgeo.org/proj/
[27] http://www.knmi.nl/opera/opera1/WG1_1e2_CommonSoftwareLibrary.pdf
[28] http://git.baltrad.eu