Quality Assurance Toolkit Documentation

Documentation of the R-Package qat

Version 0.51 (November 18, 2011)

André Düsterhus (andue@uni-bonn.de)

Contents

1 Introduction

1.1 What's it all about?

This R package should help the user to perform a quality check on data. It is constructed for the analysis of netCDF-data and deliver several checking methods, to test them. The primary focus for these tests are meteorological and climatogical datasets. In the actual available form the package is constructed to work with one dimensional data, like timeseries or vertical profiles.

1.2 Background

The developement of this package is part of the Deutsche Forschungsgesellschaft (DFG/German Science Foundation)-funded project "Publikation Umweltdaten" (Publication of Environmental Data). Its main purpose is, to deliver an environment to publish meteorologic observational data. In this process this package delivers its part in controlling the data basis. Besides performing tests it is also capable to document them and their results.

1.3 Version-Overview

Version 0.1 (September 2010): First checks and a basic structure are available.
Version 0.5 (28 April 2011): Saving structures are implemented. Publication on CRAN.
Version 0.51 (18 November 2011): Corrections for Version R 2.14.

2 Basic structure

2.1 Build up

The package consists of several parts on four different levels. The upper level is the moderator level. This level consists of only one function. With this it is possible to perform a complete quality check on a data vector. The second level is a level, where the processing of workflows take place. The third level is the calling level. This level calls the 4th level functions with a common interface. The 4th and last level is the working level. Here data will be analysed, plotted or transformed. Outside of this structures some additional functions are placed, which got supporting functionality. It is possible to access all functions on every level by themselves. This garantees a highly flexible usage of this package.

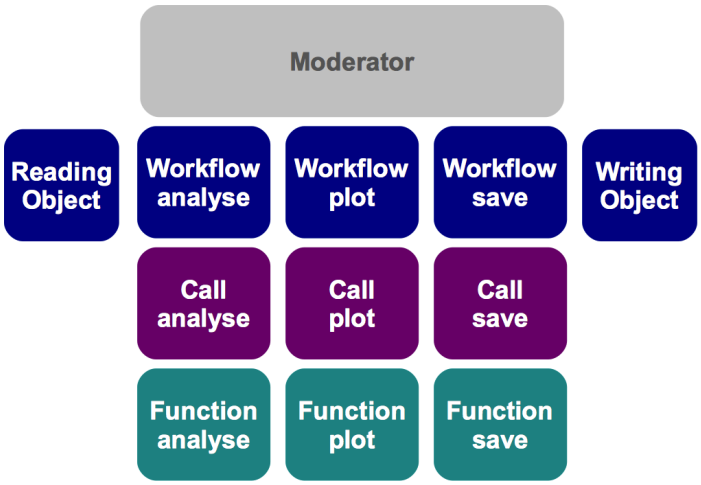

Figure 2.1: Actual structure of the qat-package. Coloured are the elements, which are actually part of the package, gray are the elements, which might be implemented in the future.

2.2 Vertical structure

2.2.1 Level 1: Moderator level

Actually this level is not implemented and the exact specifications are not determined. So the user of this package have to write this level for himself. It should include a reading of the data, callings of the workflow level and eventually a call of a saving routine.

2.2.2 Level 2: Workflow level

At the workflow level three functions are implemented. One controls the analysing of data (qat_run_workflow_check), one plots the results (qat_run_workflow_plot) and the third prepare the data for saving, e.g. in netCDF format (qat_run_workflow_save). The first function is driven by a workflowlist. This consists of names of tests and includes the parameters, which are used by them. To produce such workflowlist from a XML-source, the reading workflow function is implemented (qat_config_read_workflow). The result of the analysis will be given back as a resultlist. The first element consists of the measurement vector, which was analysed and other vectors, which may be used, when the test were performed. The other elements consists of the name of the check, a counter element and the result of the check.
This resultlist is necessary for the plotting workflow. This workflow generates one or more plots for every test into a specified path.
The third workflow generates a savelist from a resultlist. This is necessary, when the results of the tests should be stored in a format like netCDF. This may be done by the function qat_save_result_ncdf. If only the workflow itself is of interest, this can be stored as XML with the function qat_config_write_workflow, which needs only the workflowlist. All three functions work internally with the xml-file "qat_basetools.xml", which lay in the installation path of the package. This lists the implemented methods and define the functions, of the calling level, which belongs to every test. It also got the information on description and algorithm, which may be used by qat_add_all_descriptions and qat_add_all_algorithms.

2.2.3 Level 3: Calling level

The calling level is used to call the checking, plotting and saving functions with a common interface for every check. This interface consists for the checking function of the measurement vector, a part of a workflowlist, an element number, vector for potentional

coordinates (height, time, latitude, longitude), four potential additional vectors, a resultlist with other results and a counter for the latter.
Plotting-calling functions are called with a part of a resultlist, the measurement vector, vector for potentional coordinates (height, time, latitude, longitude), a name for the measurement, name of the directory, where the plot might be stored, a basic name for the file, which will get extended by the calling function, and a potential plotstyle element.
Saving-calling functions are called with a resultlist and a basic unit, which will be modified and stored in the resulting list. As a result the checking function delivers a resultlist, the plotting function produces plots at the given location and the saving functions delivers a savelist.

2.2.4 Level 4: Working level

On the working level the tests will be performed, the plots will be produced and the resultlist transformed to a savelist. For every check exists one checking, one plotting and one saving function, which got each a parameterset of its own needs. The result of the checking function is a resultlist, which is not only filled with the result of the check, but also with the parameters, which are used to perform it. The same is true for the saving functions, which will deliver a savelist element instead.

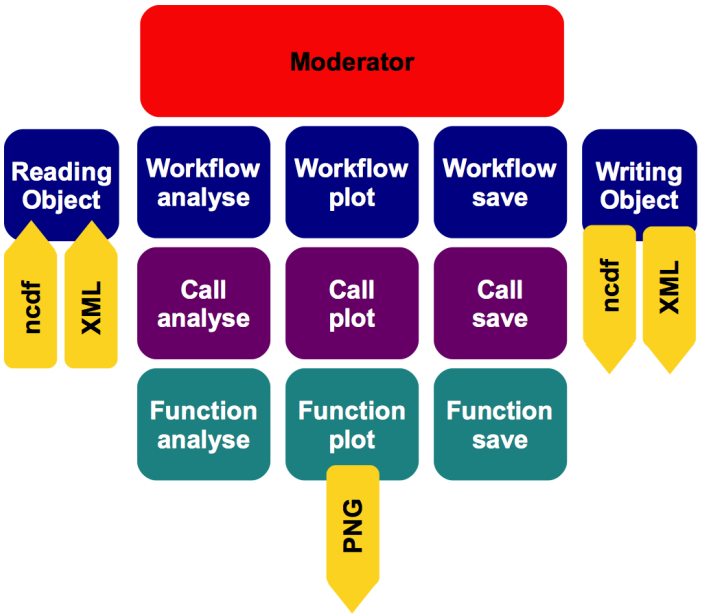

Figure 2.2: Structure of the qat-package. The fileformats are shown as yellow arrows at their point of input respectively output.

3 Control Categories

3.1 Categorie 0: Manipulation functions

3.1.1 Idea

The idea behind manipulation functions are, that the incoming data vector will be manipulated. With the result of these functions the upcoming checks will be performed. For this, once changed data vectors are not reversable.

3.1.2 Functionality

Actually three different manipulation functions are implemented.

3.1.3 Set NaN

The set NaN function ensures, that all elements of a vector, which are equal to a given value will be set to Not a Number (NaN). This may help in further checks to get useful results there.

Calling function

qat_call_set_nans

Calling plot function

Does not exist.

Plotting function

Does not exist.

Calling save function

qat_call_save_set_nans

Savingfunction

qat_save_set_nans_1d
qat_save_set_nans_above_1d
qat_save_set_nans_below_1d

Control function

qat_analyse_set_nans_1d
qat_analyse_set_nans_above_1d
qat_analyse_set_nans_below_1d

3.1.4 Set addup

The set addup function is used to add up successive elements of a vector. The number of elements, which forms a new one in the resulting vector is defined by the parameter block size. The length of the resulting vector is given by the rounded down quotient of the size of the incoming vector and the block size.

Calling function

qat_call_set_addup

Calling plot function

Does not exist.

Plotting function

Does not exist.

Calling save function

qat_call_save_set_addup

Savingfunction

qat_save_set_addup_1d

Control function

qat_analyse_set_addup_1d

3.1.5 Set mean

The set mean function is used to build a mean of successive elenets of a vector. The number of elements, which forms a new one in the resulting vector is defined by the parameter block size. The size of the resulting vector is given by the rounded down quotient of the size of the incoming vector and the block size.

Calling function

qat_call_set_mean

Calling plot function

Does not exist.

Plotting function

Does not exist.

Calling save function

qat_call_save_set_mean

Savingfunction

qat_save_set_mean_1d

Control function

qat_analyse_set_mean_1d

3.2 Categorie 1: LIM, NOC & ROC

3.2.1 Idea

These tests try to find outliners or other errors in the data by using simple tests, which are driven by user specified parameters. The basic idea of these tests are developed by Meek und Hatfield [1994].

3.2.2 Functionality

There are three classes of checks in this category, which give as a main result a flagvector. This flagvector tells the user, if there is a violation of the test rules, which are formed by the given parameters.

3.2.3 LIM-Rule

The LIM-Rule checks, if there is a violation of a determined threshold. This threshold could be static, so that a given minumum and/or maximum value is given. Another static threshold is given by the sigma-version of this check. Here the mean and the standard deviation of the data is calculated and the threshold-values are given by the mean plus and minus the standard deviation. The third version of this tests consists of dynamic threshold. Here every element of the data vector got its counter part of a vector with minumum and a vector with maximum values.

Calling function

qat_call_lim_rule

Calling plot function

qat_call_plot_lim_rule

Plotting function

qat_plot_lim_rule_static_1d
qat_plot_lim_rule_sigma_1d
qat_plot_lim_rule_dynamic_1d

Calling save function

qat_call_save_lim_rule

Saving function

qat_save_lim_rule_static_1d
qat_save_lim_rule_sigma_1d
qat_save_lim_rule_dynamic_1d

Control function

qat_analyse_lim_rule_static_1d
qat_analyse_lim_rule_sigma_1d
qat_analyse_lim_rule_dynamic_1d

Example plots

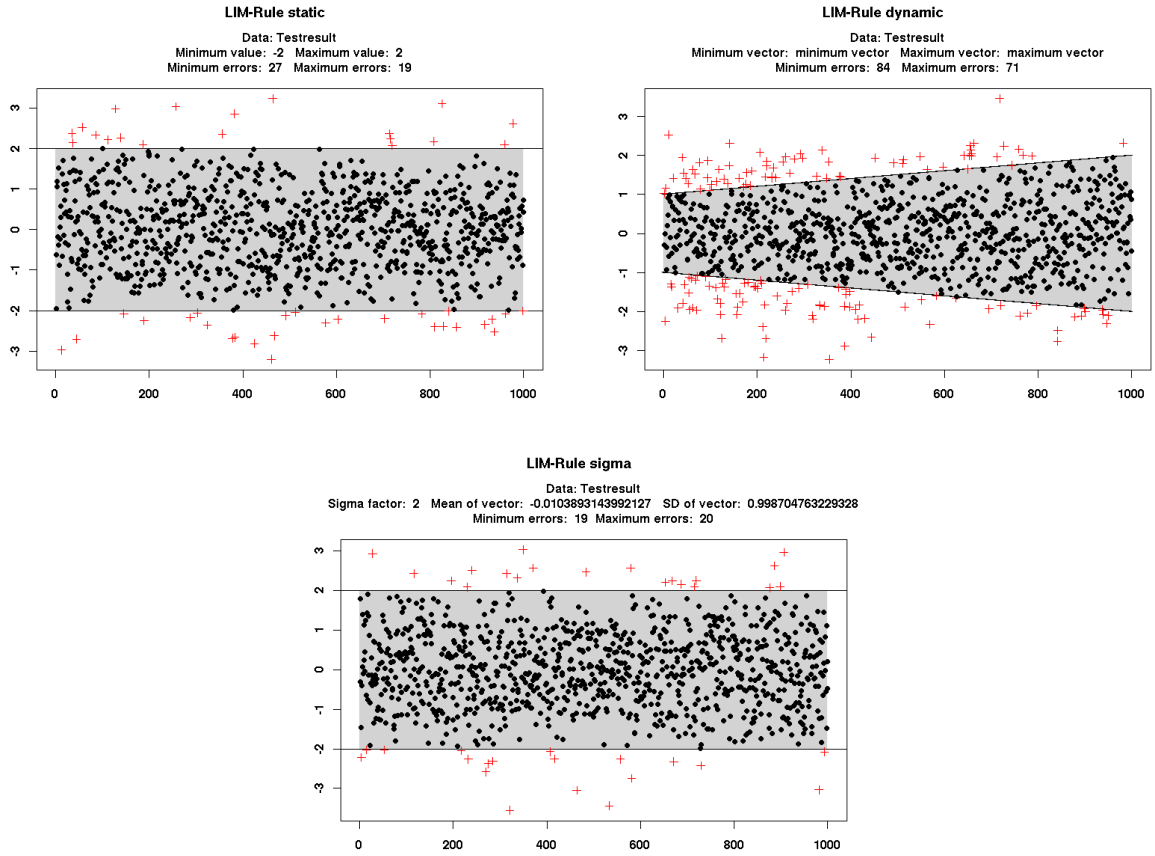

Figure 3.1: The three LIM-checks on a normal distribution. Upper left figure shows the LIM-static check, the upper right figure the LIM-dynamic check and the lower figure the LIM-sigma check.

3.2.4 ROC-Rule

The ROC-Rule checks, if a change between two successive datapoints exceeds a given threshold. This threshold could be static, so that a given maximum upward and/or downward change is fixed by given parameters. In the dynamic implementation of this check every element of the data vector got its counter part of a vector with upward and and a vector with downward values.

Calling function

qat_call_roc_rule

Calling plot function

qat_call_plot_roc_rule

Plotting function

qat_plot_roc_rule_static_1d
qat_plot_roc_rule_dynamic_1d

Calling save function

qat_call_save_roc_rule

Saving function

qat_save_roc_rule_static_1d
qat_save_roc_rule_dynamic_1d

Control function

qat_analyse_roc_rule_static_1d
qat_analyse_roc_rule_dynamic_1d

Example plots

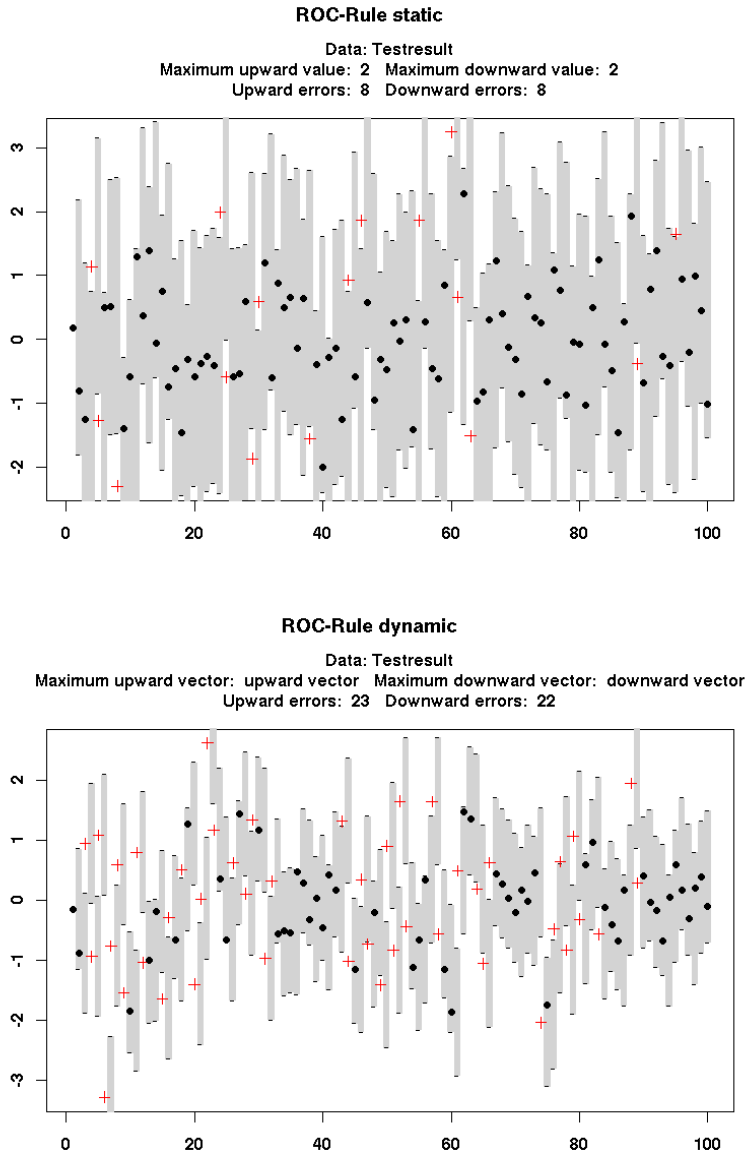

Figure 3.2: Both ROC-checks on a normal distribution. Upper figure shows the ROC-static check, lower figure the ROC-dynamic check.

3.2.5 NOC-Rule

The NOC-Rule checks, if data is variable enough. It shows if succesive data values haven't change for a given number of elements. There is no consideration of NaN-values.

Calling function

qat_call_noc_rule

Calling plot function

qat_call_plot_noc_rule

Plotting function

qat_plot_noc_rule_1d

Calling save function

qat_call_save_noc_rule

Saving function

qat_save_noc_rule_1d

Control function

qat_analyse_noc_rule_1d

Example plots

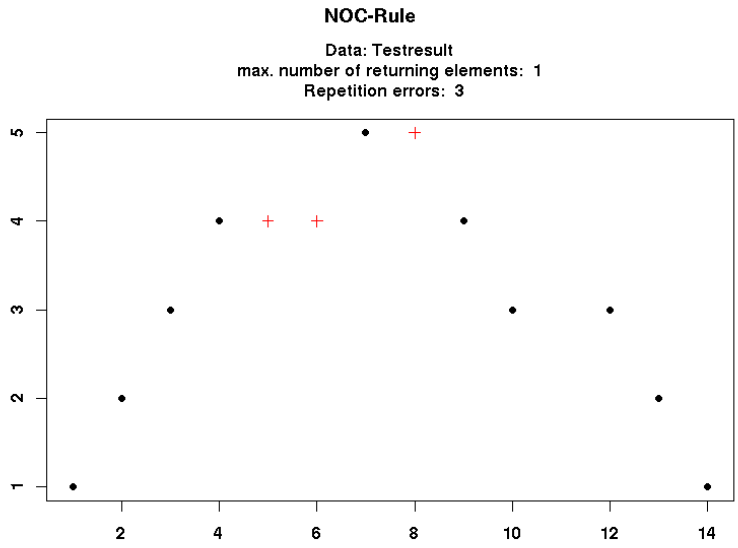

Figure 3.3: The NOC ckeck on a simple dataset.

3.3 Categorie 2: Distribution

3.3.1 Idea

These tests look at the distribiution of the whole or parts of the dataset and deliver some statistical parameters.

3.3.2 Functionality

Actually there are five different versions of distribution tests, which are implemented. All give back the same twelve statstical parameters, when they are usefull: first to fourth moment, standard deviation, skewness, kurtosis, median and the 5%, 25%, 75% and 95% quantile.

3.3.3 Distribution

The distribution-check simply give back a histogram of the dataset. This can be plotted with the plotfunction and also the statistical parameters mentioned above are calculated.

Calling function

qat_call_distribution

Calling plot function

qat_call_plot_distribution

Plotting function

qat_plot_distribution_1d

Calling save function

qat_call_save_distribution

Saving function

qat_save_distribution_1d

Control function

qat_analyse_distribution_1d

Example plots

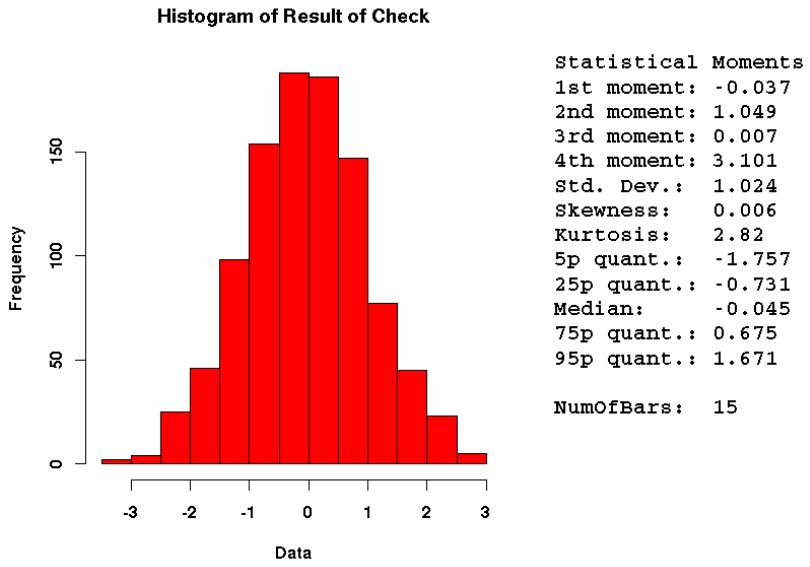

Figure 3.4: The Distribution ckeck on a normal distributed measurement vector with 1000 elements.

3.3.4 Block-Distribution

The block-distribution-check divide the dataset into blocks with the length of the given blocksize parameter and calculate for each block the statistical parameters. The plot function create three plots, which all consists of four timeseries plots of the statistical parameters.

Calling function

qat_call_block_distribution

Calling plot function

qat_call_plot_block_distribution

Plotting function

qat_plot_block_distribution_1d

Calling save function

qat_call_save_block_distribution

Saving function

qat_save_block_distribution_1d

Control function

qat_analyse_block_distribution_1d

Example plots

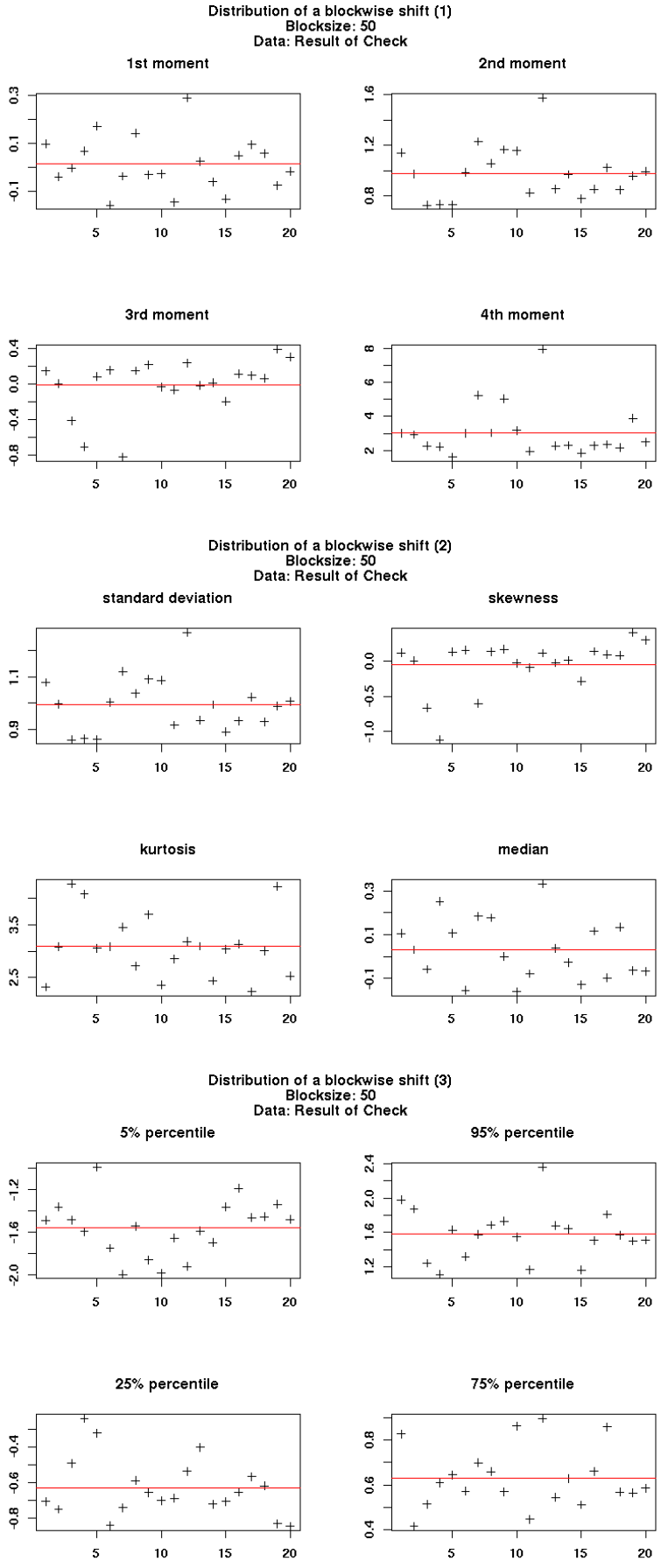

Figure 3.5: The Block-Distribution check on a normal distributed vector with 1000 elements.

3.3.5 Slide-Distribution

The slide-distribution-check use a block with the length of the given parameter blocksize and slide it through the dataset and calculate for each step the statistical parameters. The plot function create three plots, which are all consists of four timeseries plots of the statistical parameters.

Calling function

qat_call_slide_distribution

Calling plot function

qat_call_plot_slide_distribution

Plotting function

qat_plot_slide_distribution_1d

Calling save function

qat_call_save_slide_distribution

Saving function

qat_save_slide_distribution_1d

Control function

qat_analyse_slide_distribution_1d

Example plots

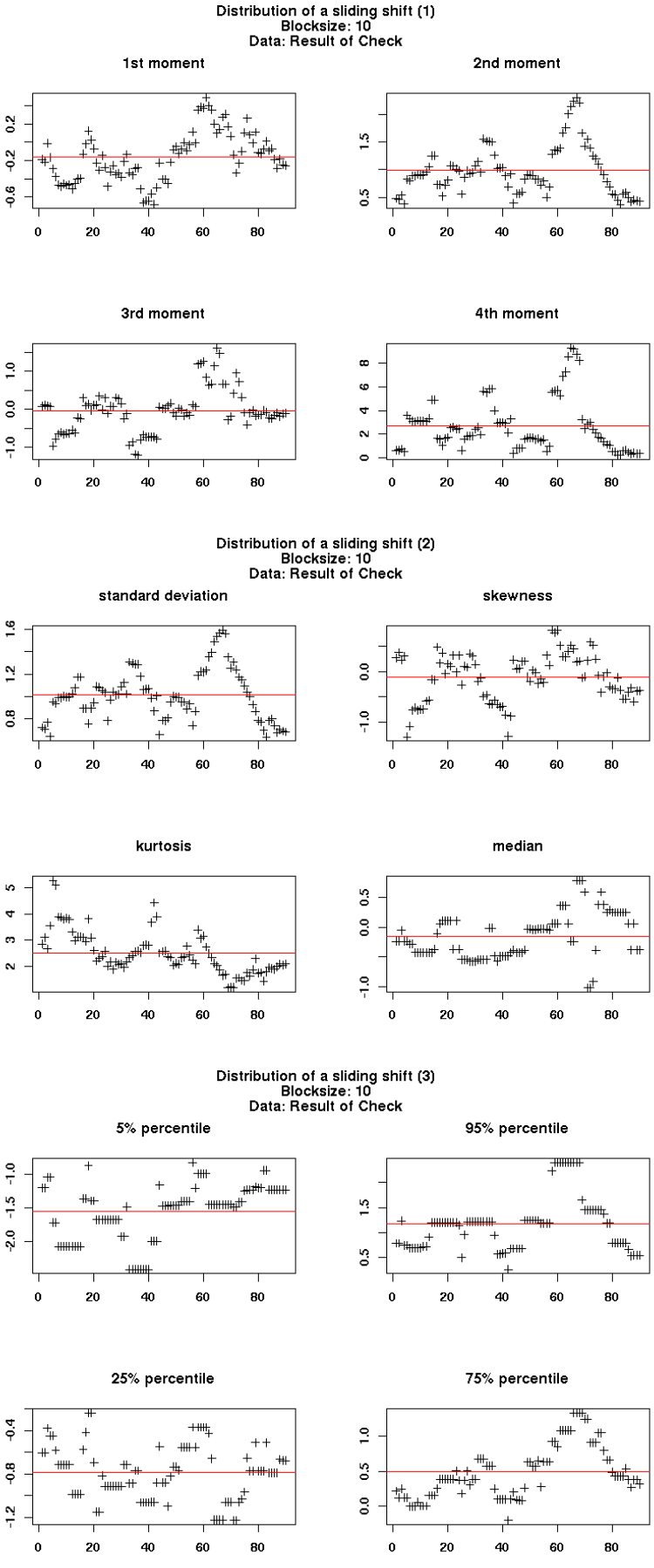

Figure 3.6: Slide-Distribution check on a normal distributed vector with 1000 elements.

3.3.6 Trimmed-Distribution

The trimmed-distribution-check trim in each step the dataset of 1% on both sides of the ordered dataset. For each of this sets the seven statsistical parameters, which are not base on quantiles are calculated. The plot function create a plot, with the mean, standard deviation, skewness and kurtosis for each step.

Calling function

qat_call_trimmed_distribution

Calling plot function

qat_call_plot_trimmed_distribution

Plotting function

qat_plot_trimmed_distribution_1d

Calling save function

qat_call_save_trimmed_distribution

Saving function

qat_save_trimmed_distribution_1d

Control function

qat_analyse_trimmed_distribution_1d

Example plots

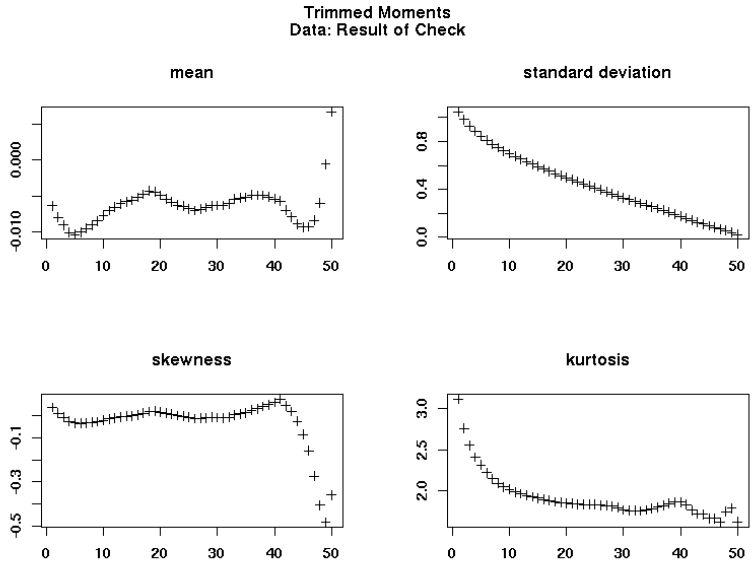

Figure 3.7: Trimmed distribution check on a normal distributed vector with 1000 elements.

3.3.7 Boot-Distribution

The boot-distribution-check bootstrap the dataset for so many times, like it is given by the given parameter bootruns. For each of this sets the statistical parameters are calculated. The plot function create a plot, which consists of twelve boxplots, one for each of the parameters.

Calling function

qat_call_boot_distribution

Calling plot function

qat_call_plot_boot_distribution

Plotting function

qat_plot_boot_distribution_1d

Calling save function

qat_call_save_boot_distribution

Saving function

qat_save_boot_distribution_1d

Control function

qat_analyse_boot_distribution_1d

Example plots

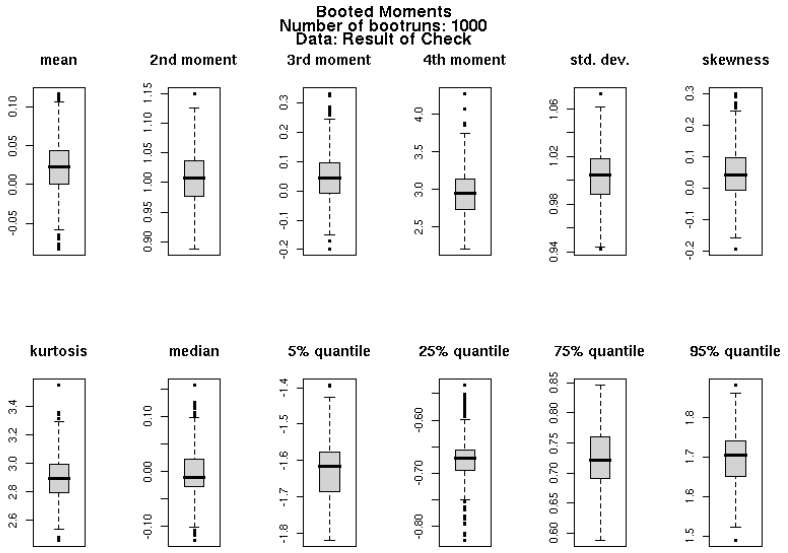

Figure 3.8: Bootstrap distribution test on a normal distributed vector with 1000 elements.

4 Additional Functions

4.1 Reading functions

This package contains reading functions to read in the measurement data. Actually there is only a functionality for netCDF files.

4.1.1 Read in data

Functionality

Read in a file at a given filename and produce a netcdf object.

Function

qat_data_read_ncdf

4.1.2 Number of Variables

Functionality

Look at a given netcdf object and deliver the number of variables.

Function

qat_data_numofvars_ncdf

4.1.3 Names of Variables

Functionality

Look at a given netcdf object and deliver the names of variables.

Function

qat_data_nameofvars_ncdf

4.1.4 Content of Variables

Functionality

Look at a given netcdf object and deliver the content of the variable, which is specified by the parameter numofvar.

Function

qat_data_varcontent_ncdf

4.2 Workflowmanagement

The workflow based on XML files, which are specified by a dtd

4.2.1 Read in workflow

Functionality

Read a workflow of an configuration XML file and give back a workflowlist.

Function

qat_config_read_workflow

4.2.2 Write result/workflow

Functionality

Write a workflow of an workflowlist to a XML file.

Function

qat_result_write

4.2.3 Write result to netCDF

Functionality

Write a workflow of a savelist to a netCDF file.

Function

qat_save_result_ncdf

4.2.4 Run a workflow of checks

Functionality

Run the checks of a workflow and give back a resultlist.

Function

qat_run_workflow_check

4.2.5 Run a workflow of plots

Functionality

Run the workflow, which is saved in a resultlist and produce the plots

Function

qat_run_workflow_plot

4.2.6 Run a workflow to save

Functionality

Run the workflow, which is saved in a resultlist and produce a savelist, which may be used to write a netCDF output of the results.

Function

qat_run_workflow_save

4.2.7 Add a comment to a workflowlist

Functionality

Adds a comment to a specified check at a workflowlist.

Function

qat_add_comment

4.2.8 Add a description to a workflowlist

Functionality

Adds a description to a specified check at a workflowlist.

Function

qat_add_description

4.2.9 Add a algorithm to a workflowlist

Functionality

Adds a algorithm to a specified check at a workflowlist.

Function

qat_add_algorithm

4.2.10 Add all descriptions to a workflowlist

Functionality

Adds the description-information, which are stored in the package to all known checks in a workflowlist.

Function

qat_add_all_descriptions

4.2.11 Add all algorithms to a workflowlist

Functionality

Adds the algorithm-information, which are stored in the package to all known checks in a workflowlist.

Function

qat_add_all_algorithms

5 File Formats

5.1 Workflow-XML

5.1.1 Idea

The workflow-XML-format should enable the user to use the workflow-level. This formate describe the checks, which should be performed by the package and its parameters. It also got the possibility to store more details on the checks. Like comments for a result or informations of the test, like a description and algorithm, itself. For all this service functions are available in the package.

5.1.2 Usage

The most important function in this context is the reading function. This functions read in the workflow and the user can now make use of it.

```
filename <- "myworkflow.xml"
workflowlist <- qat_config_read_workflow(filename)
```

For example it is possible to test a vector with normal distributed values:

```
testvector <- rnorm(100)
rlist <- qat_run_workflow_check(testvector,workflowlist)
```

This will deliver a resultlist, which now can be plotted:

```
qat_run_workflow_plot(rlist, measurement_name="Test", basename="test")
```

With this the plots will be produrced in the same directory. Perhaps some additional details should be included here, for example the descriptions and algorithms for the tests:

```
workflowlist <- qat_add_all_descriptions(workflowlist)
workflowlist <- qat_add_all_algorithms(workflowlist)
```

Also a comment for a result of a test may be a good idea, here a "everything is ok" for the first test in the list:

```
workflowlist <- qat_add_comment(workflowlist, 1, "No problems")
```

At last, the edited workflowlist should be written in a file again. For this there are two possible options. First to write simply the workflow and its additions made to this in a XML file:

```
filename2 <- "myworkflow_result.xml"
qat_config_write_workflow(workflowlist, output_filename=filename2)
```

The other option is to write not only the workflow, but also the results into a netCDF file. For this first a savelist have to be constructed:

```
slist <- qat_run_workflow_save(rlist)
```

Now this savelist can be written together with the measurement vector, which is also stored in the resultlist, and a filename into a new netCDF file:

```
filename3 <- "myworkflow_result.nc"
newtestvector <- rlist[[1]]$measurement_vector
qat_save_result_ncdf(newtestvector, slist, filename3, workflowlist)
```

5.1.3 Description

The data format is a XML-formate. The dtd-file of this looks as follows:

```
<!ELEMENT qatfile (header, workflow)>

<!ELEMENT header (name, type, description, author, date,
(tag)*,version,numofchecks, (config_header)?)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT type (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT tag (#PCDATA)>
<!ELEMENT version (#PCDATA)>
<!ELEMENT numofchecks (#PCDATA)>
<!ELEMENT config_header (name, type, description, author, date,
(tag)*, version, numofchecks)>

<!ELEMENT workflow (procedure, check)>
```

```
<!ELEMENT procedure (method_name, (parameter)*,
(result)?, (description)?, (algorithm)?)>
<!ELEMENT method_name (#PCDATA)>
<!ELEMENT parameter (parameter_name, parameter_value)>
<!ELEMENT parameter_name (#PCDATA)>
<!ELEMENT parameter_value (#PCDATA)>
<!ELEMENT result ((comment_on_result)?,(result_file)*)>
<!ELEMENT comment_on_result (#PCDATA)>
<!ELEMENT result_file (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT algorithm (#PCDATA)>

<!ELEMENT check (method_name, (parameter)*, (result)?,
(description)?, (algorithm)?)>
```

Basically this formate consists of two parts, a header and a workflow. The header stores information about the file, like the author, date or version informations. In the section of the workflow checks can be defined, with the needed parameters. These can also include description, algorithm or result informations.

5.1.4 Example

This example is a simple configuration file, which can be used to test a normal distributed vector.

```
<qatfile>
<header>
<name>Testconfiguration</name>
<type>configuration</type>
<description>This workflow tests normal distributed vectors</description>
<author>N. N.</author>
<date>2010-08-02</date>
<tag>normal</tag>
<tag>testconfig</tag>
<version>0.1</version>
<numofchecks>10</numofchecks>
</header>
<workflow>
```

```
<check>
<method_name>lim</method_name>
<parameter>
<parameter_name>minimum_value</parameter_name>
<parameter_value>-2</parameter_value>
</parameter>
<parameter>
<parameter_name>maximum_value</parameter_name>
<parameter_value>2</parameter_value>
</parameter>
<parameter>
<parameter_name>sigma_factor</parameter_name>
<parameter_value>2.3</parameter_value>
</parameter>
</check>
<check>
<method_name>roc</method_name>
<parameter>
<parameter_name>downward_value</parameter_name>
<parameter_value>1.5</parameter_value>
</parameter>
<parameter>
<parameter_name>upward_value</parameter_name>
<parameter_value>2</parameter_value>
</parameter>
<parameter>
<parameter_name>upward_vector</parameter_name>
<parameter_value>vec3</parameter_value>
</parameter>
<parameter>
<parameter_name>downward_vector</parameter_name>
<parameter_value>vec4</parameter_value>
</parameter>
<parameter>
<parameter_name>upward_vector_name</parameter_name>
<parameter_value>lin vec 1 to 3</parameter_value>
</parameter>
```

<parameter>
<parameter_name>downward_vector_name</parameter_name>
<parameter_value>lin vec 3 to 1</parameter_value>
</parameter>
</check>
<check>
<method_name>noc</method_name>
<parameter>
<parameter_name>max_return_elements</parameter_name>
<parameter_value>1</parameter_value>
</parameter>
</check>
<check>
<method_name>dist</method_name>
<parameter>
<parameter_name>numofbars</parameter_name>
<parameter_value>10</parameter_value>
</parameter>
</check>
<check>
<method_name>blockdist</method_name>
<parameter>
<parameter_name>blocksize</parameter_name>
<parameter_value>50</parameter_value>
</parameter>
</check>
<check>
<method_name>slidedist</method_name>
<parameter>
<parameter_name>blocksize</parameter_name>
<parameter_value>100</parameter_value>
</parameter>
</check>
<check>
<method_name>trimmeddist</method_name>
</check>
<check>

```
<method_name>bootdist</method_name>
<parameter>
<parameter_name>bootruns</parameter_name>
<parameter_value>100</parameter_value>
</parameter>
</check>
</workflow>
</qatfile>
```

5.2 Plotstyle-XML

5.2.1 Idea

To change the general look of the graphs it is possible to change their style. To do it for all functions with the same mechanism the plotstyle element was included. To create this list it is possible to read in the here described plotstyle-XML

5.2.2 Usage

This is controled by the function qat_style_plot. The returned list of this function is a so called plotstyle element. It is possible to read a plotstyle-XML in with the following call:

```
filename <- "myplotstyle.xml"
plotstyle <- qat_style_plot(filename)
```

The list plotstyle got now the information of the style, which is stored in myplotstyle.xml. If the standard style should be used, it is either possible to call the function without a plotstyle or let the plotting functions in the qat package do it for you.

```
plotstyle <- qat_style_plot()
```

The latter is the standard in all functions.

5.2.3 Description

The data format is a XML-formate. The dtd-file of this looks as follows:

```
<!ELEMENT qat_plotstyle (parameter*)>
```

```
<!ELEMENT parameter (parameter_name, parameter_value)>
<!ELEMENT parameter_name (#PCDATA)>
<!ELEMENT parameter_value (#PCDATA)>
```

5.2.4 Example

The following XML-Sheme shows the standard settings:

```
<qat_plotstyle>
<parameter>
<parameter_name>basecolor</parameter_name>
<parameter_value>white</parameter_value>
</parameter>
<parameter>
<parameter_name>frontcolor</parameter_name>
<parameter_value>black</parameter_value>
</parameter>
<parameter>
<parameter_name>plotcolormain</parameter_name>
<parameter_value>red</parameter_value>
</parameter>
<parameter>
<parameter_name>plotcolorminor</parameter_name>
<parameter_value>black</parameter_value>
</parameter>
<parameter>
<parameter_name>plotcolorbackground</parameter_name>
<parameter_value>lightgrey</parameter_value>
</parameter>
<parameter>
<parameter_name>fontcolor</parameter_name>
<parameter_value>black</parameter_value>
</parameter>
<parameter>
<parameter_name>plotpointminor</parameter_name>
<parameter_value>20</parameter_value>
</parameter>
<parameter>
```

<parameter_name>plotpointmain</parameter_name>
<parameter_value>3</parameter_value>
</parameter>
</qat_plotstyle>

Acknowledgement

This work was funded by the Deutsche Forschungsgemeinschaft (DFG) within the Projekt "Publikation Umweltdaten" (Publication of Environmental Data).

List of Figures

Bibliography

[Meek und Hatfield 1994] Meek, D. W. ; Hatfield, J. L.: Data Quality Checking for Single Station Meteorological Databases. In: *Agricultural and Forest Meteorology* 69 (1994), Jun, Nr. 1-2, S. 85–109. – ISSN 0168-1923