



BoA User Manual : APEX-MPI-MAN-XXXX

Version: 3.0 (03.08.2007)

Authors: C. Vlahakis, M. Albrecht, A. Beelen, F. Bertoldi, R. Schaaf, F. Schuller, et al.



Argelander-
Institut
für
Astronomie



BoA – The Bolometer Data Analysis Software

User and Reference Manual

Purpose

The purpose of this document is to provide a description of the design and usage of the Bolometer Analysis (**BoA**) software package that was designed for the *Large APEX Bolometer Camera* (LABOCA) at **APEX**.

Copyright © 2003 – 2007 MPIfR, AIfA, AIRUB

BoA is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

BoA is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with **BoA**; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307, USA

Related documents

- RD-01** BoA User's manual
 - RD-02** LABOCA design description, APEX-MPI-DSD-0016
 - RD-03** Muders, Hafok, Wyrowski et al., 2006, A&A 454, L25
 - RD-04** The BoA Project: definition, F. Bertoldi et al. (June 2002)
 - RD-05** A future bolometer data analysis software: requirements and definition, F. Bertoldi et al. (June 2002)
 - RD-06** Initial BoA web site: <http://www.openboa.de>
 - RD-07** Boa wiki: <http://www.astro.uni-bonn.de/boawiki/>
-

Definitions

For the following acronyms the understanding shall be:

AIfA	Argelander Institut für Astronomie der Universität Bonn
AIRUB	Astronomisches Institut der Ruhr-Universität Bochum
APECS	APEX Control Software
APEX	Atacama Pathfinder Experiment
ASZCa	APEX SZ Camera
BoA	Bolometer Array Analysis Package
BoGLi	BoA Graphics Library
LABOCA	Large APEX Bolometer Camera
MAMBO	Max-Planck Millimeter Bolometer
MBfits	Multi-beam fits format
MPIfR	Max-Planck-Institut für Radioastronomie, Bonn
MOPSIC	MAMBO data reduction software
NIC	IRAM bolometer reduction package
SURF	SCUBA data reduction software

Contents

I	User's Manual	1
1	Introduction	2
1.1	Philosophy and basic structure	3
2	Installation	6
2.1	Prerequisites	6
2.2	Conflicts with other software	7
2.3	Installing BoA	8
2.4	Uninstalling BoA	15
3	BoA Cookbook	16
3.1	Introduction	16
3.2	Getting started with BoA	16
3.3	Basic BoA commands	17
3.4	BoA commands for coadding data	20
3.5	BoA commands for despiking data	20
3.6	BoA commands for visualising data	21
3.7	Simple example BoA reductions	21
3.8	Pipeline reduction of LABOCA data	24
4	BoA User Manual	29
4.1	About BoA	29
4.2	BoA usage	30
4.3	Making maps	33
4.4	User methods for flagging data	33
4.5	Flatfield and opacity correction	37

4.6	Baseline subtraction, sky removal and statistics	38
4.7	FFT filtering methods	39
4.8	Pointing	40
4.9	Focus	40
4.10	File reading	41
4.11	Controlling graphics display devices	41
4.12	Plotting and displaying data	42
4.13	Data handling	50
4.14	User methods for selecting files and directories	51
4.15	Miscellaneous methods	52
4.16	Scripts	52
4.17	Commands in alphabetical order	55
4.18	Commands in functional order	57
4.19	Abbreviations	60
5	BoGLi: the BoA Graphic Library	62
5.1	Introduction	62
5.2	BoGLi commands	62
5.3	Device handling	63
5.4	Plotting graphics	65
5.5	Keywords	74
II	Reference Manual	76
6	Data Organisation	77
6.1	Data input: the MB-FITS format	77
6.2	BoAData objects	78
6.3	Data output	82
III	All BoAclasses and functions	83
A	BoA Hierarchical Index	84
A.1	BoA Class Hierarchy	84

B	BoA Class Index	85
B.1	BoA Class List	85
C	BoA Class Documentation	86
C.1	boa::BoaDataEntity::BolometerArray Class Reference	86
C.2	boa::BoaDataAnalyser::DataAna Class Reference	91
C.3	boa::BoaDataEntity::DataEntity Class Reference	106
C.4	boa::Bogli::Interface::Fenetre Class Reference	112
C.5	boa::BoaDataAnalyser::FilterFFT Class Reference	114
C.6	boa::BoaFocus::Focus Class Reference	117
C.7	boa::BoaMapping::Image Class Reference	119
C.8	boa::BoaMapping::Kernel Class Reference	123
C.9	boa::BoaMapping::Map Class Reference	124
C.10	boa::BoaPointing::Point Class Reference	128
C.11	boa::BoaDataEntity::ScanParameter Class Reference	131
C.12	boa::BoaSkydip::Skydip Class Reference	139
C.13	boa::BoaDataEntity::Telescope Class Reference	141

Part I

User's Manual

1. INTRODUCTION

The **Atacama Pathfinder Experiment (APEX)**¹ is a 12-meter radio telescope at the best accessible site for submillimeter observations, Llano de Chajnantor in Chile's Atacama desert.



Figure 1.0.1: The APEX telescope at Chajnantor in November 2003

LABOCA is a 295-channel facility bolometer camera for APEX. It operates in the $870\ \mu\text{m}$ atmospheric window and has been commissioned in May 2007. It was built at the MPIfR bolometer lab by Dr. Ernst Kreysa and his staff.

BoA is a newly designed software package for the reading, handling, and analysis of bolometer array data. Its design and implementation is a collaborative effort of scientists at the MPIfR, AIfA and AIRUB that was started in 2002 and in part funded through a "‘Verbundforschung’" grant to the MPIfR and RAIUB. **BoA** is an APEX facility software as part of the LABOCA instrument. The primary goal of **BoA** is to handle data from LABOCA at APEX, both for online visualization and offline processing. **BoA** could also be used to process data acquired with other instruments such as ASZCa at APEX or MAMBO at the IRAM 30-meter telescope. **BoA** includes most of the relevant functionalities of the current reduction packages (MOPSIC, NIC, SURF). The major difference is that **BoA** is written in a programming environment that is easier to modify, maintain, and re-use. Moreover, **BoA** naturally interfaces with APECS and the MBfits format.

¹<http://www.mpifr-bonn.mpg.de/div/mm/apex/>

1.1 Philosophy and basic structure

1.1.1 Philosophy

BoA is designed with two major goals in mind: to provide a comprehensive tool for the reduction and analysis of data from the new generation of bolometer arrays, and to facilitate the extension and modification of the software by any user. **BoA** is intended to combine a simple and intuitive usage with the coverage of all aspects of data reduction, from raw data to final images. The natural choice for the creation of **BoA** is object oriented programming.

1.1.2 Programming language: Python

Most of **BoA** is written in Python, an interpreted, interactive and object-oriented programming language. Python does not adhere to all concepts of object-orientation as strictly as, e.g., C++ does. The resulting shortcomings can be overcome by sticking to some basic programming rules.

Python is a scripting language and as such allows **BoA** to be quickly and easily extended by the user. It also facilitates the wrapping of code written in C/C++ or FORTRAN. To improve execution speed, **BoA** computing-intensive tasks are therefore written in Fortran95.

1.1.3 Basic structure

BoA consists of a set of classes, most of which are defined in dedicated modules (files). In addition, a few functions are defined in separate modules. A detailed description of all classes and methods can be found in Sec. 3. The subdivision was chosen to reach a high modularity and an intuitive grouping of related functionalities within one class.

Two kinds of classes may be distinguished:

- **Data classes:** The `DataEntity` class defines the data structure which is used within **BoA**. Objects of this class contain the raw and reduced data and all relevant parameters of a single scan. This class also defines methods to fill the data object from an MBFITS file. Then, the `DataAna` class inherits from `DataEntity`: it contains all data related methods, plus some methods for data analysis (e.g. flagging, baseline). Then, the `Map` class inherits from `DataAna`: it contains all methods defined in `DataEntity` and `DataAna`, plus specific methods for map processing and display. Finally, classes dedicated to various observing modes inherit from the `Map` class: they contain additional methods specific to a given type of observation. Table 1.1 lists **BoA** data classes, with module names and short descriptions of their responsibilities.
- **Peripheral classes:** All other classes provide methods which either are used by data objects (e.g. `Image` is used within `Map` objects), or provide functionalities on the **BoA** level (e.g. `MessHand`). These classes are summarized in Table 1.2.

Finally, a few functions are defined in separate modules (listed in Table 1.3), which do not define any class. Thus, these functions can easily be imported and run from any level. In particular, the **BoA** Graphic Library (**BoGLi**) is defined in a collection of modules, which can be imported at the python level and do not require **BoA**. A description of **BoGLi** is given in Sect. 5.

Table 1.1: **BoA** data classes

class name	module	purpose
DataEntity	BoaDataEntity.py	data and parameters storage
DataAna	BoaDataAnalyser.py	general data analysis methods
Map	BoaMapping.py	map reduction
Focus	BoaFocus.py	focus reduction
Point	BoaPointing.py	pointing reduction
Sky	BoaSkydip.py	skydip reduction

Table 1.2: Other **BoA** classes

class name	module	purpose
Image	BoaMapping.py	image and axis description
Error	BoaError.py	exception handling
FlagHandler	BoaFlagHandler.py	flag handling
MessHand	BoaMessageHandler.py	message handling
MamboMBFits	MamboMBFits.py	MAMBO to/from MB-Fits conversion
Timing	Utilities.py	benchmarking utilites

In addition, a number of utility and computing routines are written in Fortran modules. These routines are used within Python methods, and should in principle not be called directly by a **BoA** user.

Table 1.3: Other **BoA** modules

module name	purpose
BoGLi (see Sect. 5)	Graphic library
Utilities.py	collection of utilities
BoaConfig.py	global parameters definitions
BoaSimulation.py	LABOCA data simulator

2. INSTALLATION

This section describes how to install **BoA** and all required additional software packages.

2.1 Prerequisites

So far, **BoA** has been successfully installed and used in various LINUX distributions and on Mac OS X.

This documentation describes the installation process on

- SuSE 10.0 (32bit version)
- Scientific Linux 4.2 (32bit version)
- Fedora Core 3 (32bit version)

For installation instructions for other LINUX distributions and for Mac OS X, please consult the **BoA** Wiki page <http://www.astro.uni-bonn.de/boawiki>.

For the LINUX distributions covered by this documentation, the following software packages must be installed prior to the installation of **BoA** to be able to install and run **BoA**. (The given version numbers indicate the versions that were used during development and tests with the respective LINUX distribution.)

2.1.1 SuSE 10.0 (32bit version)

Table 2.1: Prerequisites for SuSE 10.0

Package	Version
gcc / gcc-c++	4.0.2
compat-g77	3.3.5
readline-devel	5.0
libpng-devel	1.2.8
xorg-x11-devel	6.8.2
findutils-locate	4.2.23

Depending on the original setup, some or all of the packages specified in table 2.1 may already be installed on your system. Use SuSE's package manager *YaST* to check if the necessary packages are present and to install or update them if necessary.

2.1.2 Scientific Linux 4.2 (32bit version)

Table 2.2: Prerequisites for Scientific Linux 4.2

Package	Version
gcc / gcc-c++	3.4.4
gcc-g77	3.4.4
readline-devel	4.3
libpng-devel	1.2.7
xorg-x11-devel	6.8.2
findutils	4.1.20

All packages listed in table 2.2 are part of a standard “Workstation” installation. If a package is missing or needs to be updated, use the package manager *yum* that comes with Scientific Linux.

2.1.3 Fedora Core 3 (32bit version)

Table 2.3: Prerequisites for Fedora Core 3

Package	Version
gcc / gcc-c++	3.4.4
gcc-g77	3.4.4
readline-devel	4.3
libpng-devel	1.2.8
xorg-x11-devel	6.8.2
findutils	4.1.20

All packages listed in table 2.3 are part of a standard “Workstation” installation. If a package is missing or needs to be updated, use Fedora's package manager *yum*.

NB: Tests were performed with Fedora installed in a VMWare virtual machine. With its original kernel (version 2.6.9), Fedora run extremley slow. This is a know problem with Fedora Core 3 under VMWare. Updating to kernel version 2.6.12 improved the performance considerably.

2.2 Conflicts with other software

BoA is delivered including a set of programs and libraries (called the **BoA Library**) necessary to run **BoA**. This makes a installation of **BoA** more or less self-contained and reduces the chance of conflicts with other software installed on your system.

However, the behaviour of a system is not only determined by the set of software that is installed on it, but also by the environment that is defined both system-wide and on a per-user basis in various startup scripts. (For a complete list of the startup scripts consult the documentation of your system; most important are the startup scripts of the shell in use. See `man bash` and `man csh`)

During the installation of **BoA**, the installation script tries to set up an environment that allows a smooth installation. When running `boa`, **BoA**'s start-up scripts `.boarc.sh` and `.boarc.csh` try the same. However, there may be situations when this is not successful. If this is the case, careful inspection of the environment must be performed. (To give an example: During the development phase of **BoA**, running **BoA** failed reproducibly on one particular system; after scrutinizing various startup scripts, the cause turned out to be a startup script for IRAF, that changed the C and Fortran compilers. After commenting out the IRAF related lines, **BoA** ran without any further problems.)

2.3 Installing BoA

A complete distribution of **BoA** contains two tar-archives:

- *BoaLib-<DATE>.tgz*, containing the **BoA Library**, a set of programs and libraries necessary to run **BoA**, in versions that have been verified to work properly together with **BoA**'s scripts.
- *Boa-<DATE>.tgz*, containing Python scripts, Fortran programs that provide **BoA**'s core functionality. The archive also contains startup and example scripts as well as **BoA**'s documentation.

(<DATE> indicates the release date of each tar-archive.)

A complete installation of **BoA** includes installation of the **BoA Library** and of **BoA** itself. Both installation steps are described below.

Please note that the following instructions relate to the installation of **BoA** on the LINUX distributions listed in section 2.1. For installation instructions for other LINUX distributions and for Mac OS X, please consult the **BoA** Wiki page <http://www.astro.uni-bonn.de/boawiki>

2.3.1 Installing the BoA Library

The **BoA Library** is contained in the tar-archive *BoaLib-<DATE>.tgz*. It is a set of programs and libraries necessary to run **BoA**, in versions that have been verified to work properly together with **BoA**'s scripts (contained in *Boa-<DATE>.tgz*).

In particular, the **BoA Library** contains the programs and libraries specified in table 2.4.

The installation process will install the **BoA Library** in its own directory tree. The root of this directory tree is referred to as `BOA_LIB_HOME` in this document. If not specified otherwise during the installation process, `BOA_LIB_HOME` is `$HOME/boalib`, where `$HOME` is your home directory.

You do not need root privileges to install the **BoA Library** as long as you install it to a location where you have write permission.

To install the **BoA Library**, proceed as follows:

Table 2.4: Content of **BoA Library**

Program/library	Version
Python	2.3.2
Numeric	23.1
numarray	0.9
swig	1.3.23
Intel FORTRAN	8.1
scipy_distutils	3.3_33.571
f2py	2.44.240_1892
pgplot	5.2
pPGPLOT	1.3
slalib	
pySLALIB	0.4
blas/lapack	
cfitsio	2.49
pCFITSIO	
BoA-FFTW-Numpy	1.0
mpfit	
wcslib	4.1
dchelper	
apexFitsWriter	
apexCalibrator	

1. Unpack the archive to the directory `BoaLib-<DATE>-install` by typing

```
tar xzf BoaLib-<DATE>.tgz
```

and go to this directory:

```
cd BoaLib-<DATE>-install
```

This directory may safely be deleted after the **BoA Library** has been installed.

2. Run the configure script to create the install script:

```
./configure
```

The configure script tests for each software package that is part of the **BoA Library** whether this particular software package is already installed on your computer. If the configure script finds the particular software package on the computer, it tries to find out the version. Only if the configure script finds out that the particular software package is installed in exactly the

version that comes with the **BoA Library**, this particular software package will be skipped in the following installation step. In all other cases (configure cannot locate the software, the versions do not match, configure cannot find out the versions, etc.) configure registers the software package for installation.

You can override this default behaviour with options of the configure script. E.g.

```
./configure --with-python
```

will register Python for installation, even if the correct Python version is already installed on your computer, while

```
./configure --without-python
```

will prevent Python to be installed. Typing

```
./configure --with-all
```

will mark all packages for installation without checking. You can combine these options to finetune the installation:

```
./configure --with-python --without-slablib
```

will work as expected as will

```
./configure --with-all --without-python
```

For a list of all possible options `--with-package` and `--without-package`, type

```
./configure --help
```

The configure script will also specify `BOA_LIB_HOME`, the root of the directory tree where the **BoA Library** will be installed. By default this is `$HOME/boalib` where `$HOME` is your home directory. You can specify a different root using the `--prefix` option as in

```
./configure --prefix=${HOME}/myOwnBoaLibrary
```

3. Run the install script:

```
./install
```

This will install the software packages that were registered for installation by the configure script. If the root of the directory tree `BOA_LIB_HOME` (either `$HOME/boalib` or the directory specified explicitly with `configure --prefix`) already exists, the install script will warn you and query you whether you really want to proceed. If unsure, answer "N"; this will abort the installation. You can rerun the configure script specifying a different prefix and then run install again.

The rest of the installation happens without any input from you. Note that the install process can take about 30 min to complete.

After running this script, the **BoA Library** is installed and ready to be used. You may proceed with the installation of **BoA** itself.

Troubleshooting

In case of errors during the installation, the install script prints out error messages and aborts. Consult the log files that are specified in the error message to find out possible reasons for the failure. Quite often the cause for a failure to install the **BoA Library** are special settings of shell variables (e.g. `PYTHONPATH`) in shell startup scripts. If the reason for the failure is unclear, delete the directory tree with the incomplete installation of **BoA Library**, run

```
./configure --with-all
```

and run the install script again.

If errors still occur, the most probable causes are problems with the environment defined by the set of path settings, shell variables etc. which are set both in system-wide and user-specific startup scripts. (For a complete list of the startup scripts consult the documentation of your system; most important are the startup scripts of the shell in use. See `man bash` and `man csh`.)

During the installation of the **BoA Library**, the installation script tries to set up an environment that allows a smooth installation. However, there may be situations when this is not successful. If this is the case, careful inspection of the environment must be performed.

2.3.2 Installing BoA

The tar-archive *Boa-<DATE>.tgz* contains

- Python scripts, Fortran programs and some related data files necessary to run **BoA**
 - startup scripts for bash and csh to set the correct shell environment to run **BoA**
 - example scripts and related data in MBFits files to be used as cookbook examples for **BoA**
 - scripts for the reduction of Laboca data
-

- rcp files with instrument-specific parameters
- **BoA**'s documentation

Before you can install BoA, you must install the **BoA Library** contained in `BoaLib-<DATE>.tgz` (see section 2.3.1).

By default, the **BoA Library** is installed into a directory tree with root `$HOME/boalib`. However, when installing the **BoA Library** you can choose a different directory tree by specifying the prefix option for the `BoaLib-<DATE>-install/configure` command. The root of this directory tree (either the default `$HOME/boalib` or the one specified by the prefix option of `BoaLib-<DATE>-install/configure`) is referred to as `BOA_LIB_HOME` in this document.

BoA's installation process will install the software in its own directory tree. The default for the root of this tree is `$HOME/boouser` where `$HOME` is your home directory. If you want to install **BoA** to another location, you can specify **BoA**'s root directory during the configuration as described below.

You do not need root privileges to install the software, as long as you install it to a location where you have write permission.

To install BoA, proceed as follows:

1. Unpack the archive to the directory `Boa-<DATE>-install` by typing

```
tar xzf Boa-<DATE>.tgz
```

and go to this directory:

```
cd Boa-<DATE>-install
```

This directory may safely be deleted after the **BoA** has been installed.

2. Run the configure script to create the install script:

In general, the configure script must be run including the specification of the variable `BOA_LIB_HOME`:

```
./configure BOA_LIB_HOME=${BOA_LIB_HOME}
```

where `$BOA_LIB_HOME` is the root of the directory tree to which the **BoA Library** has been installed (see above). `BOA_LIB_HOME` needs not be specified if the **BoA Library** has been installed to the default location (`$HOME/boalib`).

Examples:

- You installed the **BoA Library** without specifying a prefix by running (in `BoaLib-<DATE>-install`)

```
./configure
```

Then you can run the configure script in `Boa-<DATE>-install` also without argument:

```
./configure
```

- You installed the **BoA Library** specifying a prefix by running (in `BoaLib-<DATE>-install`)

```
./configure --prefix=${HOME}/myBoaLib
```

Then you must specify the `BOA_LIB_HOME` argument when running the configure script in `Boa-<DATE>-install`:

```
./configure BOA_LIB_HOME=${HOME}/myBoaLib
```

By default, the configuration scripts registers **BoA**, the startup scripts, the example scripts, and the documentation for installation. You can override this default behaviour with options of the configure script. E.g.

```
./configure --without-examples
```

will prevent the example scripts to be installed. For a list of all possible options `--with-package` and `--without-package`, type

```
./configure --help
```

The configure script will also specify the root of the directory tree where Boa will be installed. By default this is `$HOME/boouser`. You can specify a different root using the `--prefix` option as in

```
./configure --prefix=${HOME}/myOwnBoa
```

3. Run the install script:

```
./install
```

This will install BoA into a directory tree with root `$HOME/boouser` (if the configure command was called without specifying a prefix) or with the root specified by the configure command. The root directory will contain the following subdirectories:

- **boa**: **BoA**'s Python and Fortran code
- **examples**: Example scripts and related data
- **laboca**: Scripts for the reduction of Laboca data
- **rcp**: rcp files with instrument-specific parameters
- **doc**: **BoA**'s documentation

The install script will also create the two startup scripts `.boarc.sh` and `.boarc.csh` in your home directory. (If these files already exist, the existing files are renamed `.bosrc.sh` and `.boarc.csh`.) These files contain definitions of shell variables, path settings, and aliases necessary to run **BoA**.

4. Run **BoA**:

In order to run **BoA**, first run the correct startup script by typing

```
source ~/.boarc.sh (if you are working in bash)
```

or

```
source ~/.boarc.csh (if you are working in csh)
```

(You may include this line into your `.bashrc` or `.cshrc` file to automate this task.)

You can then run **BoA** by typing

```
boa
```

Troubleshooting

In case of errors during the installation, the install script prints out error messages and aborts. Consult the log files that are specified in the error message to find out possible reasons for the failure.

A possible cause for errors during the installation is an incorrect specification of the variable `BOA_LIB_HOME` and/or the `prefix` option when running the configure script. Check your settings and rerun configure and install if necessary.

Other possible causes for problems both during installation and when running **BoA** are conflicts with the environment defined by the set of path settings, shell variables etc. which are set both in system-wide and user-specific startup scripts. (For a complete list of the startup scripts consult the documentation of your system; most important are the startup scripts of the shell in use. See `man bash` and `man csh`.)

During the installation of **BoA**, the installation script tries to set up an environment that allows a smooth installation. When running **BoA**, the start-up scripts `.boarc.sh` and `.boarc.csh` try the same. However, there may be situations when this is not successful. If this is the case, careful

inspection of the environment must be performed. (To give an example: During the development phase of **BoA**, running **BoA** failed reproducably on one particular system; after scrutinizing various startup scripts, the cause turned out to be a startup script for IRAF, that changed the C and Fortran compilers. After commenting out the IRAF related lines, **BoA** ran without any further problems.)

2.4 Uninstalling BoA

To uninstall **BoA**, delete the directory tree into which **BoA** has been installed, and the startup scripts `/.boarc.sh` and `/.boarc.csh`.

To uninstall the **BoA Library**, delete the directory tree to which the **BoA Library** has been installed.

3. BoA COOKBOOK

3.1 Introduction

This cookbook describes basic data reduction using **BoA**. The **BoA** software can be obtained as described in Chapter 2. Currently this cookbook is oriented towards the reduction of data taken with the LABOCA submillimetre array at the APEX telescope.

The cookbook describes how to start up **BoA** for the first time (Section 3.2.1) and describes some example **BoA** sessions, including making a map and solving a pointing and focus (Section 3.7). These example sessions are intended to allow the beginner or occasional user to get on air quickly. Users already familiar with the content of this cookbook can find example pipeline reduction scripts in Section 3.8 and detailed information on **BoA** commands in Chapter 4.

3.2 Getting started with BoA

3.2.1 Starting up BoA

Before you start up **BoA**, make sure that the correct startup script is run. This can either be done manually by typing

```
source ~/.boarc.sh (if you are working in bash)
```

or

```
source ~/.boarc.csh (if you are working in csh)
```

at the command prompt or automatically by inclusion of the proper lines into your `.bashrc` or `.cshrc` file.

The most common way to invoke **BoA** is to simply type

```
boa
```

at the command prompt. **BoA** then prints a welcome message providing version information and changes the prompt to the `boa>` prompt. (Note that you are nevertheless still in the interactive Python layer).

When **BoA** is initiated it imports a set of modules, instantiates the most essential objects and makes the respective methods available using the start script *BoaStart.py*.

Advanced: Invoking BoA from within Python

In certain circumstances, more advanced users may wish to invoke **BoA** from within a Python session. This can be done by typing

```
>>> from BoaStart import *
```

at the Python prompt.

3.2.2 Setup for displaying and reading in data

```
op()                % 1
indir('/home/user/data/') % 2
ils()               % 3
proj('projectID')  % 4
read('filename')    % 5
```

A typical **BoA** session will usually require a data file as input and a graphic output device. Command 1 opens the default graphics device (pgplot). Command 2 sets the desired input directory, i.e. in this case the input data file is located in a directory called */home/user/data/*. The content of this directory can then be listed (command 3). The project ID can also be set (command 4) so that filenames may subsequently be described by just the observation number. Command 5 then reads in the input data file.

Note, these commands and those used in the sections below are abbreviations for the full user method names, as is described in Section [4.2.1](#).

3.2.3 Ending a session

To end a session you can first close the graphics device by typing

```
close()
```

then end the **BoA** session by typing *ctrl+d*.

3.2.4 Getting Help

You can get help on a **BoA** `command()` at any time by typing

```
print command.__doc__
```

at the prompt.

3.3 Basic BoA commands

The main **BoA** data reduction commands are summarised briefly in this section.

There are a few main steps which need to be carried out in order to produce a final reduced LABOCA map. These are `correctOpacity()`, `flatfield()`, `flagC()`, `flagSpeed()`, `flagAccel()`, `medianNoiseRemoval()`, `correlgroup()`, `despike()`, `flattenFreq()`, `base()` and `computeWeight()`. A few additional steps are necessary to perform despiking and to calibrate the map. These are discussed in Section 3.5 and Section 3.8.2. In the pipeline data reduction script shown in Section 3.8 there are also some further steps to take into account some effects related to the instrumental performance.

3.3.1 correctOpacity

Correct for the atmospheric opacity (τ).

3.3.2 flatfield

Divides the signals by bolometer gains to normalise them. There are three choices of flatfield to apply, which can be selected using the *method* keyword. *point* (the default) uses point source relative gains; *median* used correlated noise relative gains; *extend* uses relative gains to extended emission. The default is to process all channels, but if required you can select a list of channels using the *chanList=[]* keyword.

3.3.3 flagC

Assign flags to a list of channels. Supply a list of channels to be flagged in the *chanList=[]* keyword. This can be done both for bad channels and for the dark channels.

3.3.4 flagSpeed

Flag data according to the telescope speed.

3.3.5 flagAccel

Flag data according to the telescope acceleration.

3.3.6 medianNoiseRemoval

Correct for sky noise variations across the array by removing the median noise from the data. The default is to use all channels, but if required you can apply to selected channels only by supply a list of channels in the *chanList=[]* keyword. The keyword *chanRef* is used for computing the relative gains in order to normalise the signals before computing their median. It should be set to -1 to compute the relative gains with respect to the mean signal. It should be set to -2 to compute the relative gains with respect to the median signal. It can also be set to a specific channel number, in which case the relative gains are computed with respect to the signal in that channel. The default is to use the reference

channel that was defined during the observations. The keyword *factor* allows the fraction of skynoise to be subtracted to be set (default is 1, i.e. 100%).

There are two other methods for removing correlated noise, `cnr()` and `pca()` which are currently under development. Contact Frank Bertoldi or Martin Nord for more information.

3.3.7 correlbox

Tests during the commissioning period of LABOCA showed that for LABOCA data correlated noise not only exists across the array but also between groups of channels sharing some parts of the electronics. This task additionally subtracts correlated noise per amplifier box (up to 80 channels connected to the same box).

3.3.8 correlgroup

As for *correlbox*, this task additionally subtracts correlated noise per cable (up to 25 channels connected through the same cable).

3.3.9 flattenFreq

Flatten the $1/f$ part of the FFT using constant amplitude. Use the keyword *below* to set the value below which to filter data and *hiref* to set the value with which amplitudes at $f < below$ will be replaced – the replacement value will be the average value between *below* and *hiref*. The default is to apply this to all channels, but if required you can apply to selected channels only by supplying a list of channels in the *chanList=[]* keyword.

The values for the parameters *below* and *hiref* should be chosen depending on the expected brightness and spatial scale of the sources. Since choice of these parameters will affect the final map care should be taken to choose values which are most appropriate to the particular type of source. See Section 3.8.4 for further details.

3.3.10 base

Perform a polynomial baseline removal on the data. Set the order of the polynomial using the keyword *order* (default is 0). The default is to compute the baseline per subscan, but if this is not required then set the keyword *subscan* to 0. The default is to apply this to all channels, but if required you can apply to selected channels only by supplying a list of channels in the *chanList=[]* keyword.

3.3.11 medianbase

Subtract a zero order baseline (i.e. a constant) to the data. The value to be subtracted is the median value of all unflagged data, per channel and per subscan. It can also be a single value per channel for the full scan, by setting the keyword *subscan* to 0. The default is to apply this to all channels, but it can be restricted to selected channels by supplying a list of channels in the *chanList=[]* keyword.

3.3.12 computeWeight

Computes weights and stores them for use when combining the signals of all bolometers into a map. The default weighting method is $1/rms^2$.

3.3.13 doMap

Build a map in (Az,El) or EQ coordinates. The default is to use all channels, but if required you can select a list of channels using the *chanList=[]* keyword. The oversampling factor can be set using the *oversamp* keyword (default is 2, i.e. use pixels of half the beam size). To compute a beammap set the *beammap* keyword to a value of 1. The coordinate system can be either *HO* (Az,El offsets) or *EQ* (RA, Dec absolute coordinates). The *sizeX* and *sizeY* keywords are used to set the limits of the map in Az and El (or RA and Dec) respectively. See Section 3.8.3 for a description of how setting these keywords is important when coadding multiple maps. Set the *smooth* keyword to smooth with beam. A full list of keywords accepted by `doMap()` can be found in Chapter 4.

3.4 BoA commands for coadding data

3.4.1 mapSum

Coadds together a number of maps (weights and coverage are also coadded). A coadded map with the same WCS and data size is produced.

3.5 BoA commands for despiking data

3.5.1 flagFractionRms

Flag channels according to rms, with limits depending on median rms of all (yet unflagged) channels. The keyword *ratio* supplies the value above and below which channels will be flagged, as a fraction of the median rms value (i.e. in the form $\text{median} \times \text{ratio}$ and $\text{median}/\text{ratio}$). The default value is 10. The default is to apply this to all channels, but if required you can apply to selected channels only by supplying a list of channels in the *chanList=[]* keyword.

3.5.2 flagRms

Flag channels with rms above and below the rms values specified using the keywords *above* and *below*.

3.5.3 despike

Flag yet unflagged data above and below the given number of times the channel rms specified using the keywords *above* and *below* (e.g. *above=5, below=-3* will flag data below $-3 \times \text{rms}$ and data above $5 \times \text{rms}$).

3.6 BoA commands for visualising data

3.6.1 signal

Plots the time series of the signal.

3.6.2 plotRmsChan

Plots the channel RMS value for all channels (or for the list specified by keyword *chanList*) against channel number.

3.7 Simple example BoA reductions

The following sections show three example reductions of real LABOCA data, which are useful for gaining familiarity with the basic functionalities of **BoA**. The examples show a basic usage of the main commands you will find necessary for reducing your data. Enter the individual commands at the **BoA** prompt for a step-by-step method. You can also try out the examples in an automated way, using the three example scripts provided with your **BoA** installation (*ExampleMap.py*, *ExamplePointing.py* and *ExampleFocus.py*) which can be found in the directory */home/user/boouser/examples/* (if **BoA** was installed to the default location). Run the scripts by typing:

```
execfile('/home/user/boouser/examples/ExampleMap.py')
```

3.7.1 Example 1: making a map

```

op() % 1
indir('/home/user/data/') % 2
ils() % 3
proj('T-77.F-0002-2006') % 4
read('59491') % 5
signal() % 6
signal(1) % 7
doMap() % 8
medianBaseline() % 9
plotRmsChan() % 10
flagRms(above=1) % 11
flagRms(below=0.2) % 12
updateRCP('jup-44830-32-improved.rcp') % 13
flagPos(radius=150.) % 14
base(order=1) % 15
medianNoiseRemoval() % 16
plotRmsChan() % 17
flagRms(above=0.5) % 18
plotRmsChan() % 19
flagC([140,227]) % 20
despike() % 21
computeWeight() % 22
unflag(flag=8) % 23
doMap(system='EQ',sizeX=[83.9,83.73],sizeY=[-5.48,-5.28],oversamp=5.) % 24
smooth(6./3600.) % 25
display(caption=data.ScanParam.caption()) % 26
close() % 27

```

Setting up and accessing the data

The initial steps for starting up a typical session were described in Section 3.2. Command 1 opens the default graphics device and Command 2 sets the desired input directory. The content of this directory is then listed (command 3). The project ID can then be set (command 4) so that filenames may subsequently be described by just the observation number (in this example the file-naming convention is for LABOCA data, and consists of the APEX project ID (*T-77.F-0002-2006*) and the observation number). Command 5 then reads in the input data file for observation *59491*.

Visualising the data

To get a first look at the data you can use command 6 to plot the time series of the data for each pixel, or command 7 to look at the time series of the data for an individual pixel. You can also make a rough map using command 8. These commands will be used again (see below) when the data is processed.

Basic Processing and Analysis

Usually the processing of the data will begin with subtracting a zero-order baseline. This is done with command 9, where the median value per channel and per subscan is removed. To see the data after

baseline subtraction you can use commands 6 and 7 again. Next, command 10 plots the RMS value versus pixel (channel) number. Commands 11 and 12 then flag pixels with RMS values which are higher or lower than the given value as bad. At this point you can use command 10 again to view the remaining unflagged data.

Command 13 reads in the rcv file for calibration. Command 14 then flags the area in which source signal is present, and commands 15 and 16 remove a baseline (using a polynomial fit) and the correlated signal, computed as the median of all signals. The new distribution is then checked with command 19.

Bad channels can be flagged using command 20, and the data then despiked (command 21). If necessary, the despiked data can be examined using commands 6 and 7. Before producing a map the data should be weighted, in this case each channel weighted as the inverse of the square of the rms of that channel (command 22). Command 23 then unflags the previously flagged source position.

Command 24 produces a map in EQ coordinates. See Chapter 4 for optional arguments for these and other methods. The map may then be smoothed (command 25) and this smoothed map displayed (command 26).

3.7.2 Example 2: solving a pointing

```

op() % 1
indir('/home/user/data/') % 2
proj('T-77.F-0002-2006') % 3
read('42947') % 4
signal() % 5
signal(1) % 6
doMap() % 7
medianBaseline() % 8
doMap(oversamp=3) % 9
solvepointing(plot=1) % 10
clear() % 11
read('46117') % 12
medianBaseline() % 13
medianNoiseRemoval() % 14
plotRmsChan() % 15
flagRms(above=20) % 16
doMap(oversamp=3) % 17
solvepointing(plot=1) % 18
close() % 19

```

The above example shows a typical session to solve a pointing. As usual (see Section 3.2) we begin by opening a graphics display device, setting the input directory, and setting the project ID (commands 1,2 and 3). The data file containing the pointing observation is read in (command 4), in this case a strong pointing source (Jupiter). As a first look at the data, the time series of the data for each pixel (command 5) or individual pixel (command 6) can then be plotted. Likewise a rough first-look map can be made (command 7). To construct the map on which to solve the pointing, the median baseline is first removed (command 8) (to see how the signal looks now you can repeat commands 5 and 6). Finally the pointing map is constructed (command 9) and the pointing solved (command 10).

If the pointing source is fainter (in this case an observation of Uranus), some additional steps could

be taken. Following the above example, a graphics display window is already open so we can clear the display using command 11. Command 12 then reads in the data file containing the observation of Uranus. Again, first looks at the data can be made using commands 5, 6 and 7. The median baseline is then removed (command 13), and this time the median noise value is also removed (command 14). You can then check at what RMS most channels are using command 15, then use command 16 to flag channels with RMS values above a certain value (in this case an RMS of 20). Command 15 can then be repeated to see how the data looks now. The pointing map can then be constructed (command 18) and the pointing solved (command 18). Command 19 then closes the graphics display device.

3.7.3 Example 3: solving a focus

```

op() % 1
indir('/home/user/data/') % 2
proj('T-77.F-0002-2006') % 3
read('43275') % 4
solveFocus() % 5

read('46118') % 6
medianBaseline() % 7
medianNoiseRemoval() % 8
solveFocus() % 9
close() % 10

```

The above example shows a typical session to solve a focus. As usual (see Section 3.2) we begin by opening a graphics display device, setting the input directory, and setting the project ID (commands 1,2 and 3). Command 4 the reads in the data file, in this case for a strong source (Jupiter). Command 5 then solves the focus. Command 6 then reads in a new data file, this time for a fainter source (Uranus). This time the median baseline and median noise levels are removed before solving the focus (commands 7, 8 and 9).

3.8 Pipeline reduction of LABOCA data

This section describes the basic steps to reduce LABOCA data. Section 3.8.1 describes the reduction of the skydips to derive the opacity correction, Section 3.8.2 the calibration scheme, Section 3.8.3 describes how to set up a script to reduce your data in an automated way, and Section 3.8.4 the steps needed to carry out a standard data reduction.

3.8.1 Skydip reduction

An example script to reduce Laboca skydips is available in `/home/user/boouser/laboca/reduce-skydip-he3corr.boa`. Laboca skydips consist of two scans: one hot-sky scan for calibration purposes and the skydip itself. In brief the script determines in the first step the zenith sky temperature from the hot-sky scan. Then it calculates the observed sky temperature as a function of the elevation from the 2nd scan. Finally the zenith opacity is fitted to the sky temperature - elevation curve. The second step includes a correction for temperature drifts on the He3 stage of Laboca. These drifts occur because

the Laboca cryostat is strongly tilted during a skydip. Because of the total power design of Laboca, He3 temperature drifts are indistinguishable from variations of the sky emission and this correction is essential for the skydip reduction. The He3 temperatures are stored in the monitor table of the fits file.

To derive the zenith opacity for each target scan, the results from each skydip during the observing run can be stored together with its observing date in a data file. Such a zenith opacity file can easily be created using the `/home/user/boouser/laboca/reduce-skydips-loop.boa` macro. The macro loops over all skydip scans given in the scan list and writes the result to an output file. Note that the scan list should contain only the scan numbers of the hot-sky scans. The function `getTau()` can then be used for any target scan to retrieve the nearest opacity value in time, or a linear interpolation of the zenith opacity from the two bordering skydip scans.

3.8.2 Calibration scheme

The raw units of Laboca data in the fits file are counts which can be converted to the detector output voltage using the function `CntstoV()`. The calibration factor between the detector output voltage and the flux density/beam has been determined during the Laboca commissioning run and is stored in the `VtoJy` variable defined in `/home/user/boouser/laboca/cabling.py`. To calibrate the data to Jy/beam using this standard calibration factor therefore only requires the following steps:

```
read('13690')           % 1
CntstoV(data)           % 2
data.Data *= array(VtoJy,'f') % 3
mjdref = fStat.f_mean(data.ScanParam.MJD) % 4
tau = getTau(mjdref,'linear','Laboca-taus.dat') % 5
data.correctOpacity(tau) % 6
```

Read in a scan (1). Convert the detector counts to detector output voltage (2), convert to Jy/beam (3). The opacity correction is applied based on the observing date as described in Section 3.8.1: determine the observing date (MJD) (4), get a linear interpolation of the zenith tau based on the boardering skydips (5), apply the opacity correction (6).

To test and improve the calibration the BoA installation comes with an example script to reduce the primary (Uranus, Neptune, Mars) and secondary flux calibrators observed during the run (`/home/user/boouser/laboca/reduce-calib-loop.boa`). The fluxes and names of the secondary calibrators are stored in `/home/user/boouser/laboca/secondary-calibrator-flux.boa`. Note that this file contains also the expected fluxes of the primary calibrators which have to be modified according to the observing date (e.g. using Astro in the Gildas software package). The `/home/user/boouser/laboca/reduce-calib-loop.boa` macro loops over all calibration scans given in the scan list, reduces them using the standard calibration (see above) and derives a correction factor for each scan based on the flux in `/home/user/boouser/laboca/secondary-calibrator-flux.boa`. The reduction of each scan uses the `/home/user/boouser/laboca/reduce-calib-map.boa` script. The calibration correction is stored together with the observing date in a file. Similar to the opacity correction the function `getCalCorr()` can then be used to modify the standard calibration based on the observing date for each scan:


```

read('13690') % 1
CntstoV(data) % 2
data.Data *= array(VtoJy,'f') % 3
mjdref = fStat.f_mean(data.ScanParam.MJD) % 4
tau = getTau(mjdref,'linear','Laboca-taus.dat') % 5
data.correctOpacity(tau) % 6
calcorr = getCalCorr(mjdref,'linear','Laboca-calib.dat') % 7
data.Data /= array(calcorr,'f') % 8

```

Steps 1 to 6 are identical to the standard calibration. Step 7 derives a linear interpolation of the calibration correction determined from the two bordering flux calibrator observations. This correction is applied in step 8.

3.8.3 Example reduction script

Here we show a typical pipeline reduction script for a list of scans. Optionally one can apply, similar to the skydip reduction, corrections based on the He3 temperature fluctuation during the scan. Note, however, that most of the signal drifts introduced by these variations strongly correlate among bolometers and are therefore mostly removed by the skynoise removal functions.

```

scans = [13688,13689,13690] % 1
ra1,ra2 = 84.0,83.65 % 2
de1,de2 = -5.75,-4.85
apply_he3corr = 0 % 3
indir('/home/user/data/') % 4
proj('T-77.F-0002-2006')
mapList = []
for num in range(len(scans)): % 5
    s = str(scans[num])
    read(s,readHe=1)
    mjdref = fStat.f_mean(data.ScanParam.MJD)
    tau = getTau(mjdref,'linear','Laboca-taus.dat')
    data.correctOpacity(tau)
    calcorr = getCalCorr(mjdref,'linear','Laboca-calib.dat')
    data.Data /= array(calcorr,'f')
    execfile('reduce-map-weaksource.boa')
    doMap(system='EQ',sizeX=[ra1,ra2],sizeY=[de1,de2])
    mapList.append(data.Map)
ms = mapsum(mapList) % 6
ms.display() % 7
ms.writeFits('output.fits') % 8

```

Read in a list of scans (1). Set the RA and Dec limits which will later be used by `doMap()` (2). Set this parameter to apply a correction for He3 drifts (3). Set parameters for reading in the data (4). Set up a loop to reduce each scan in turn (5). This loop does the following. Read in a scan (omit the `readHe=1` keyword if a He3 correction is not required) and apply the opacity and calibration correction (see Section 3.8.1 & 3.8.2, then carry out the reduction using the script `/home/user/boouser/laboca/reduce-map-weaksource.boa`. Make a map of the reduced scan. All the reduced maps are finally summed into a final coadded map using `mapsum()`. Note that `mapsum` assumes that all maps have the same size, and correspond to the same coordinates on the sky, and so it is important to set values of RA and Dec

limit in the command `doMap()`. This is done at (2). Coordinates are not checked at present. Finally, coadd all the maps (6), display the resulting coadded map (7) and also output it to a fits file (8).

3.8.4 Reducing the data

The following script is `/home/user/boouser/laboca/reduce-map-weaksource.boa` which is called in the above example. It contains all the necessary steps to reduce standard LABOCA data and is optimised for weak sources. For strong or extended sources the same steps can be used, but the values of the parameters for the command `flattenFreq()` should be adjusted accordingly.

Scripts to reduce various types of sources can be found in the directory `/home/user/boouser/laboca`. These are `reduce-map-weaksource.boa`, `reduce-map-strongsource.boa`, `reduce-map-mediumsource.boa`, `reduce-map-extendedsource.boa` and `reduce-map-strongextendedsource.boa`.

```

execfile(os.getenv('BOA_HOME_LABOCA')+'/cabling.py')    % 1
CntstoV(data)                                           % 2
updateRCP('master-laboca-may07.rcp')                  % 3
data.zeroStart()
flatfield()
flagC(resistor)                                         % 4
flagC(cross)
flagC(sealed_may07)

try:                                                     % 5
    tmp = apply_he3corr
except NameError:
    apply_he3corr = 0
if apply_he3corr:
    correctHe3(data)                                    % 5

data.Data *= array(VtoJy,'f')                          % 6
flagSpeed (below=30.)                                  % 7
flagSpeed (above=500.)
flagAccel (above=800.)
flagFractionRms (ratio=5)                              % 8
medianNoiseRemoval (chanRef=-1, factor=0.8, nbloop=5)   % 9
despike (below=-5, above=5)                             % 10
correlbox (data, factor=0.8, nbloop=2)                  % 11
correlgroup (data, factor=0.8, nbloop=2)
flagFractionRms (ratio=5)
despike (below=-3, above=3)
flattenFreq (below=0.3, hiref=0.35)                    % 12
base (order=1, subscan=0)                               % 13
despike (below=-3, above=3)
computeWeight ()                                         % 14

```

Read some LABOCA specific definitions (1) and convert data units to detector output voltage (2). Apply a flat field (3) and flag bad channels (4). If desired, apply a correction for the He3 drift (this requires the data unit to be in detector output voltage) (5). Convert data units into Janskys (6). Flag stationary points and high acceleration in the data (7). Flag dead and very noisy channels (8). Perform a first correlated noise removal on all channels (9) and despike the data (10). Remove correlated noise

by boxes and groups of channels (11). Apply a low frequency filter (exact values for the parameters depends of the type of source you have)(12), remove a first order baseline (13) and compute the weights (14).

4. BoA USER MANUAL

In this chapter you will find information about the structure of **BoA**, how **BoA** can be used, together with detailed descriptions of user methods. Since many user methods have an abbreviated form, these are listed in Section [4.19](#).

4.1 About BoA

In this Section we give a basic overview of the structure of **BoA**. Section [4.1.1](#) gives a brief introduction to the raw data file format, and Section [4.1.2](#) shows an overview of the data structure within **BoA**. More in-depth descriptions are given in Chapter [6](#).

4.1.1 Input data

The data acquired at the APEX telescope are stored in a new file format, known as the MB-Fits format (for Multi-Beam FITS format, see the reference document APEX-MPI-IFD-0002 by Hatchell et al. for details). These files contain:

- the raw data as provided by the Frontend-Backend in use at the telescope
- data associated parameters: time of the observations, positions on the sky...
- a description of the complete Scan (eg. for a map: number of lines, steps between lines...)
- parameters of the receiver channels in the array: relative positions, relative gains

A more complete description of the input data format is given in Sect. [6.1](#).

4.1.2 Internal data handling

Taking full advantage of the object-oriented nature of Python, **BoA** handles data by means of objects of various classes. The primary class for data storage and manipulation is called `DataEntity` (see also Section [6.2.1](#)). This class allows to store the raw data and associated parameters, and it provides methods relevant for any kind of observations (e.g. reading data from an MB-FITS file, plotting the signal as time series, plotting the telescope pattern). The most important attributes of this class are:

- `BolometerArray`: here, the relative positions and gains of the receiver channels are stored, as well as generic informations about the instrument and telescope (name, diameter, coordinates...)

- **ScanParam**: this contains the data associated parameters: coordinates of each point in several systems, timestamps (in LST and MJD), subscans related informations
- **Data**: this is a 2D array (time \times bolometer) which contains the current version of the data. At time of reading, the raw data are stored there; the content of this array is then altered by any processing step
- **DataFlags**, **DataWeights**: 2D arrays, with same size as **Data**, where flagging values and relative weights are stored for each individual data point

For processing different types of observations, **BoA** then provides several classes which inherits from **DataEntity**. Inheritance allows to define a class which contains all attributes and methods of the parent class, plus some specific attributes/methods. The inheritance scheme in **BoA** is as follows:

```
DataEntity < DataAna < Map < Point < Focus
```

When **BoA** is started, one object of class *Focus* is created with name *data*; this is the current data object, on which all reduction procedures can be applied. Additional objects of any data class can be created by the user within one **BoA** session. Then, applying processing methods to a data object with a different name than *data* requires to enter the full syntax (see Chapter ...), including the full name of the method, as opposed to the shortcuts described in Chapters 3 and 4.

Note: Python ensures no real difference between private and public attributes. There are only hidden attributes but this hiding can be overcome easily. Therefore the user might set any attribute directly and call any method. This is not advisable and may easily corrupt the whole **BoA** session. It is more recommendable to just use those methods for which the start script *BoaStart.py* provides abbreviations.

4.2 BoA usage

4.2.1 Methods

BoA tasks are accessed by directly calling the appropriate methods from the interactive Python layer. This ensures the full availability of all Python and ppgplot facilities. As the method names to be called from the Python layer may be rather long, the start script *BoaStart.py* provides a set of convenient abbreviations for those methods which are meant to be called directly by the user (“public” methods). We will therefore refer to these as user methods, a full list of which can be found in Section 4.19.

Example:

The name of the method to open a new graphic device is *DeviceHandler.openDev* and it can be called by

```
DeviceHandler.openDev()
```

or more conveniently by the abbreviations (user methods)

```
op()
```

(note that the parentheses are always mandatory).

4.2.2 Arguments

Nearly all user methods require arguments to be passed. Nevertheless, the methods provide default arguments which thus may be omitted. In this case many methods just supply status information.

Example:

The user method `indir()` sets the desired input directory and requires the directory name as its argument:

```
indir('/home/user/data/')
```

The directory name is a string argument and has to be passed embedded in double or single quotes. Note that for consistency, in the examples throughout this manual we always use single quotes, but these can of course be substituted for double quotes.

Omitting the argument does not change the input directory but instead results in the supply of the current directory name:

```
indir()
```

In case an argument has to be typed more often a Python variable can be used:

```
a='/home/user/data/'  
indir(a)
```

Some methods require a list as argument. In Python a list is embedded in square brackets with a comma as separator. Python provides a variety of functionalities to manipulate lists.

Example:

The user method `signal()` plots the time series of the data (flux density or counts versus time). It allows the user to define the list of channels plotted:

```
signal([18,19,20])
```

To create a list you can use the Python function `range()`:

```
mylist=range(1,163)  
signal(mylist)
```

or:

```
signal(range(1,163))
```

When considering only one element, the square brackets can be omitted:

```
signal(5)
```

User methods can also be called using keyword arguments of the form *keyword = value*.

Example:

By default, the user method `signal()` plots the signal versus time connecting the datapoints with lines:

```
signal()
```

However, if you prefer, for example, to see the individual datapoints without lines, you can modify the value of the *style* argument:

```
signal(style='p')
```

A description of graphics related arguments such as *style* is given in [Section 5.5](#).

4.2.3 Output

Most user methods supply status information as screen output when being called. The amount of information displayed can be restricted using the message handler associated with the main *data* object:

```
data.MessHand.setMaxWeight(4)
```

where the argument is an integer value between 1 and 5, with the following meaning:

- 1: errors, queries
- 2: warnings
- 3: short info
- 4: extended info
- 5: debug

4.3 Making maps

4.3.1 Building a map in (Az,El) or EQ coordinates

METHOD: `doMap` (*optional arguments*)

DESCRIPTION: construct a map in (Az,El) or (RA,Dec) coordinates

OPTIONAL ARGUMENTS:

<i>chanList</i>	channels to consider, of the form [1,2,3] (default: all non-flagged)
<i>channelFlag</i>	plot data from channels flagged or unflagged accordingly
<i>plotFlaggedChannels</i>	channelFlag revers to flagged/unflagged data
<i>dataFlag</i>	plot data flagged or unflagged accordingly
<i>plotFlaggedData</i>	dataFlag revers to flagged/unflagged data
<i>oversamp</i>	oversampling factor (beam fwhm / pixel size). Default=2.
<i>beammap</i>	compute a beam map (default: no)
<i>system</i>	coordinate system, one of 'HO' (Az,El *offsets*) or 'EQ' (RA, Dec absolute coordinates); default = 'HO' optionally 'EQFAST' to do only one rotation on small maps (faster)
<i>sizeX</i>	limits in Az of the map
<i>sizeY</i>	limits in El of the map
<i>limitsZ</i>	limits in pixel values to compute the color scale
<i>style</i>	color table to use in image
<i>smooth</i>	do we smooth with beam? (default: no)
<i>noPlot</i>	do not plot the map? (default: no)
<i>caption</i>	plot caption
<i>aspect</i>	keep aspect ratio? (default: yes)
<i>showRms</i>	compute and print rms/beam? (default: yes)
<i>rmsKappa</i>	kappa in kappa-sigma clipping used to compute rms
<i>derotate</i>	derotate Nasmyth array by Elevation

4.4 User methods for flagging data

4.4.1 Despiking

METHOD: `despike` (*optional arguments*)

DESCRIPTION: Flag yet unflagged data below *below**rms and above *above**rms.

OPTIONAL ARGUMENTS:

<i>chanList</i>	list of channels to be flagged (default: current list)
<i>below</i>	flag data with value < 'below'*rms
<i>above</i>	flag data with value > 'above'*rms
<i>flag</i>	flag values (default: 1 'SPIKE')

METHOD: `iterativeDespike` (*optional arguments*)

DESCRIPTION: Iteratively flag yet unflagged data below *below**rms and above *above**rms.

OPTIONAL ARGUMENTS:

<i>chanList</i>	list of channels to be flagged (default: current list)
<i>below</i>	flag data with value < 'below'*rms
<i>above</i>	flag data with value > 'above'*rms
<i>maxIter</i>	maximum number of iterations (default 100)
<i>flag</i>	flag values (default: 1 'SPIKE')

4.4.2 Flagging a list of channels

METHOD: `flagChannels` (*optional arguments*)

DESCRIPTION: assign flags to a list of channels. To unflag a channel simply flag with flag=0.

OPTIONAL ARGUMENTS:

<i>chanList</i>	list of channels to be flagged (default: current list)
<i>flag</i>	flag value (default: 8 'TEMPORARY')

4.4.3 Flagging data by time interval

METHOD: `flagMJD` (*optional arguments*)

DESCRIPTION: flag data by MJD interval

OPTIONAL ARGUMENTS:

<i>below</i>	flag data below this value (default end of the scan)
<i>above</i>	flag data above this value (default start of the scan)
<i>flag</i>	flag value to be set (default: 8 'TEMPORARY')

METHOD: `flagInTime` (*optional arguments*)

DESCRIPTION: Flag data in time interval.

OPTIONAL ARGUMENTS:

<i>below</i>	flag data below this value (default end of the scan)
<i>above</i>	flag data above this value (default start of the scan)
<i>flag</i>	flag value to be set (default: 8 'TEMPORARY')

4.4.4 Flagging a position on the sky

METHOD: `flagPosition` (*optional arguments*)

DESCRIPTION: flag a position in the sky within a given radius

OPTIONAL ARGUMENTS:

<i>channel</i>	list of channels to flag (default: 'all')
<i>Az/El</i>	the horizontal reference position (arcsec for offsets, deg for absolute)
<i>radius</i>	aperture to flag in unit of the reference position
<i>flag</i>	flag to be set (default 8 'TEMPORARY')
<i>offset</i>	flag on the offsets (default yes)

4.4.5 Flagging channels with certain rms values

METHOD: `flagRms` (*optional arguments*)

DESCRIPTION: flag channels with rms below *below* or above *above*.

OPTIONAL ARGUMENTS:

<i>chanList</i>	list of channel to flag (default: current list)
<i>below</i>	flag channels with rms < 'below'
<i>above</i>	flag channels with rms > 'above'
<i>flag</i>	flag value to set (default: 2 'BAD SENSITIVITY')

METHOD: `flagFractionRms` (*optional arguments*)

DESCRIPTION: flag according to rms, with limits depending on median rms.

OPTIONAL ARGUMENTS:

<i>chanList</i>	list of channel to flag (default: current list)
<i>ratio</i>	channels with rms below median/ratio and above median*ratio will be flagged
<i>flag</i>	flag value to set (default: 2 'BAD SENSITIVITY')
<i>plot</i>	plot the results

4.4.6 Flagging subscans

METHOD: `flagSubscan` (*optional arguments*)

DESCRIPTION: flag a list of subscans

OPTIONAL ARGUMENTS:

<i>subList</i>	list of subscan numbers (or single number) to be flagged
<i>flag</i>	flag value to be set (default: 7 'SUBSCAN FLAGGED')

4.4.7 Flagging speeds

METHOD: `flagSpeed` (*optional arguments*)

DESCRIPTION: Flag data according to telescope speed

OPTIONAL ARGUMENTS:

<i>below</i>	flag data below this value
<i>above</i>	flag data above this value
<i>flag</i>	flag to be set (default 3 'ELEVATION VELOCITY THRESHOLD')

4.4.8 Flagging accelerations

METHOD: `flagAccel` (*optional arguments*)

DESCRIPTION: Flag data according to telescope acceleration

OPTIONAL ARGUMENTS:

below flag data below this value
above flag data above this value
flag flag to be set (default 2 'ACCELERATION THRESHOLD')

4.4.9 Unflagging

METHOD: `unflag` (*optional arguments*)

DESCRIPTION: Unflag data, i.e. reset flags to 0.

OPTIONAL ARGUMENTS:

channel list of channels to be unflagged (default: current list)
flag unflag only this value (default 1)

METHOD: `unflagMJD` (*optional arguments*)

DESCRIPTION: Unflag data in time interval.

OPTIONAL ARGUMENTS:

below unflag data below this value (default end of the scan)
above unflag data above this value (default start of the scan)
flag unflag value to be set (default []: all flag values)

METHOD: `unflagInTime` (*optional arguments*)

DESCRIPTION: Unflag data in time interval.

OPTIONAL ARGUMENTS:

below unflag data below this value (default end of the scan)
above unflag data above this value (default start of the scan)
flag unflag value to be set (default []: all flag values)

METHOD: `unflagPosition` (*optional arguments*)

DESCRIPTION: unflag a position in the sky within a given radius

OPTIONAL ARGUMENTS:

channel list of channels to unflag (default: 'all')
Az/El the horizontal reference position (arcsec for offsets, deg for absolute)
radius aperture to unflag in unit of the reference position
flag unflag to be set (default []: unflag all non-reserved flag values)
offset unflag on the offsets (default yes)

METHOD: `unflagChannels` (*optional arguments*)

DESCRIPTION: Unflag a list of channels

OPTIONAL ARGUMENTS:

chanList list of channels to be unflagged (default: current list)
flag flag values (default []: unset all flags)

METHOD: `unflagSubscan` (*optional arguments*)

DESCRIPTION: unflag a list of subscans

OPTIONAL ARGUMENTS:

subList list of subscan numbers (or single number) to be unflagged
flag flag value to be unset (default []: all flag values)

METHOD: `unflagSpeed` (*optional arguments*)

DESCRIPTION: Unflag data according to telescope speed

OPTIONAL ARGUMENTS:

below unflag data below this value
above unflag data above this value
flag flag to be unset (default []: all flag values)

METHOD: `unflagAccel` (*optional arguments*)

DESCRIPTION: Unflag data according to telescope acceleration

OPTIONAL ARGUMENTS:

below unflag data below this value
above unflag data above this value
flag flag to be unset (default []: all flag values)

4.5 Flatfield and opacity correction

4.5.1 Flatfield

METHOD: `flatfield` (*optional arguments*)

DESCRIPTION: divide signals by bolometer gains to normalise them

OPTIONAL ARGUMENTS:

channel list of channels to process (default: [] = current list)
method choose which flat field to apply:
 point: use point source relative gains (default)
 median: use correlated noise relative gains
 extend: use relative gains to extended emission

4.5.2 Correcting for opacity

METHOD: `correctOpacity` (*optional arguments*)

DESCRIPTION: correct for atmospheric opacity

4.6 Baseline subtraction, sky removal and statistics

4.6.1 Computing the Rms in a map

METHOD: `computeRms` () (*optional arguments*)

DESCRIPTION: compute rms/beam in a map (dispersion between pixels)

OPTIONAL ARGUMENTS:

<i>rmsKappa</i>	for kappa-sigma clipping before computing rms
<i>limitsX</i>	optionally define a sub-region (pixel coord)
<i>limitsY</i>	optionally define a sub-region (pixel coord)

4.6.2 Computing weights

METHOD: `computeWeight` () (*optional argument*)

DESCRIPTION: compute weights and store them in `DataWeights` attribute

OPTIONAL ARGUMENTS:

<i>method</i>	type of weighting (default='rms', i.e. use $1/\text{rms}^2$)
---------------	---

4.6.3 Median baseline removal

METHOD: `medianBaseline` (*optional arguments*)

DESCRIPTION: baseline: remove median value per channel and per subscan

OPTIONAL ARGUMENTS:

<i>channel</i>	list of channels to process (default: [] = current list)
<i>subscan</i>	compute baseline per subscan (default: yes)
<i>order</i>	polynomial order (default: 0)

4.6.4 Skynoise removal

METHOD: `medianNoiseRemoval` (*optional arguments*)

DESCRIPTION: remove median noise from the data

OPTIONAL ARGUMENTS:

<i>chanList</i>	list of channels (default: [] = current list)
<i>chanRef</i>	reference channel number (default: RefChannel)
	–1 = compute relative gains w.r.t. mean signal
	–2 = compute relative gains w.r.t. median signal
<i>computeFF</i>	compute skynoise FF (default) or use existing FF_Median?
<i>factor</i>	fraction of skynoise to be subtracted (default: 1, i.e. 100%)
<i>nbloop</i>	number of iterations (default: 1)

4.6.5 Polynomial baseline removal

METHOD: `polynomialBaseline` (*optional arguments*)

DESCRIPTION: perform polynomial baseline removal on the data

OPTIONAL ARGUMENTS:

<i>channel</i>	list of channels to flag (default: all; [] : current list)
<i>order</i>	polynomial order, >0
<i>subscan</i>	compute baseline per subscan (default: yes)
<i>plot</i>	plot the signal and the fitted polynomials (default: no)
<i>subtract</i>	subtract the polynomial from the data (default: yes)

4.6.6 Smoothing an image

METHOD: `smoothBy` (*optional arguments*)

DESCRIPTION: smooth the image with a 2D gaussian of given FWHM

OPTIONAL ARGUMENTS:

<i>Size</i>	the FWHM of the smoothing gaussian
-------------	------------------------------------

4.6.7 Obtaining the statistics

METHOD: `statistics`

DESCRIPTION: compute mean, median, rms for all scans and subscans for all used channels

4.7 FFT filtering methods

METHOD: `blankFreq` (*optional arguments*)

DESCRIPTION: Permanently remove some frequency interval in the Fourier spectrum of the signal. This is computed subscan by subscan.

OPTIONAL ARGUMENTS:

<i>channel</i>	list of channels to process (default: all)
<i>below</i>	filter data below this value
<i>above</i>	filter data above this value

METHOD: `flattenFreq`(*optional arguments*)

DESCRIPTION: flatten the 1/f part of the FFT using constant amplitude

OPTIONAL ARGUMENTS:

<i>channel</i>	list of channels to process (default: all)
<i>below</i>	filter data below this value
<i>hiref</i>	amplitudes at $f < \text{below}$ will be replaced with the average value between below and hiref

4.8 Pointing

4.8.1 Solving a pointing

METHOD: `solvePointingOnMap`(*optional arguments*)

DESCRIPTION: compute the offset on the data.Map object

OPTIONAL ARGUMENTS:

<i>gradient</i>	shall we fit a gradient ? (default: no)
<i>circular</i>	fit a circular gaussian instead of an elliptical gaussian
<i>radius</i>	use only data points inside this radius (negative means multiple of beam) (default: 10 beams)
<i>Xpos</i>	source position used as first guess
<i>Ypos</i>	source position used as first guess
<i>fixedPos</i>	if set, don't fit position, but use Xpos, Ypos
<i>plot</i>	do we plot the results? (default: no)
<i>display</i>	display the result of the fit (default: yes)

WARNING : No Smoothing should be applied to the map before using this function, or the fitted fwhm will be useless, use fine oversamp to make reasonable fit.

4.9 Focus

4.9.1 Solving a focus

METHOD: `solveFocus`

DESCRIPTION: compute the optimal focus position, by fitting a parabola to the signal versus subreflector position information

4.10 File reading

4.10.1 Reading a FITS file

Reading a FITS file into **BoA** is done with the `read()` command. You may want to define the input directory first:

```
indir('../fits/')      # set the input directory
read('filename')       # read file filename.fits
```

The data are then stored in the default *data* object. It is possible to use several data objects, and to store the content of a file to a user defined object requires the following syntax:

```
data2 = BoaMapping.Map() # define a second data object
                        # of class Map
data2.read('filename')
```

4.11 Controlling graphics display devices

In order to display your data in various ways using the **BoA** plotting methods described in Section 4.12 below, you first need to open a graphics display device (e.g. Xwindows). Graphics display in **BoA** is controlled by a software package called **BoGLi** (the **BoA** Graphic Library), which is described in Chapter 5. A few basic **BoGLi** commands which are needed in order to carry out the **BoA** plotting methods described in section 4.12 are thus described in this section.

4.11.1 Opening a plot window

Opening a graphic device is done with the `openDev()` command:

```
openDev()      # open a device, default: XWindow
op()           # alternatively, use one of the abbreviated commands
```

The default is to open an XWindow. You can use

```
op('?')
```

to get a list of all recognized devices. Alternatively, if you know which device you want you can enter it directly, for example

```
op('/ps')
```

You can also open a named PostScript file, here a colour PostScript file named *signal.ps*, with

```
op('signal.ps/CPS')
```

Note that if no device is already open, **BoA** will automatically the default graphic device at the first time a plotting command is entered.

4.11.2 Clearing a plot window

Clearing a plotting window is done with the `clear()` command:

```
clear()          # clear the active device
```

However, any plot command will first clear the active device before plotting a new graph, unless the *overplot=1* keyword is supplied.

4.11.3 Closing a plot window

Closing a graphic device is done with the `closeDev()` command:

```
closeDev()      # open a device, default: XWindow
```

4.11.4 Selecting an open device

METHOD: `selectDev`

DESCRIPTION: select an open device

4.12 Plotting and displaying data

4.12.1 Plotting channel maps

METHOD: `chanMap(optional argument)`

DESCRIPTION: Compute and plot channel maps in HO offset coordinates

OPTIONAL ARGUMENTS:

<i>chanList</i>	list of channels to consider, of the form [1,2,3]
<i>channelFlag</i>	plot data from channels flagged or unflagged accordingly
<i>plotFlaggedChannels</i>	channelFlag revers to flagged/unflagged data
<i>dataFlag</i>	plot data flagged or unflagged accordingly
<i>plotFlaggedData</i>	dataFlag revers to flagged/unflagged data
<i>oversamp</i>	oversampling factor (beam fwhm / pixel size). Default=2.
<i>sizeX</i>	limits in Az of the map
<i>sizeY</i>	limits in El of the map
<i>limitsZ</i>	
<i>style</i>	color table to use in images
<i>center</i>	if set, it will shift each map by the bolometer offsets. Thereby it shifts the source to the center of each channel map.
<i>showRms</i>	compute and print rms/beam? (default: no)
<i>rmsKappa</i>	kappa in kappa-sigma clipping used to compute rms

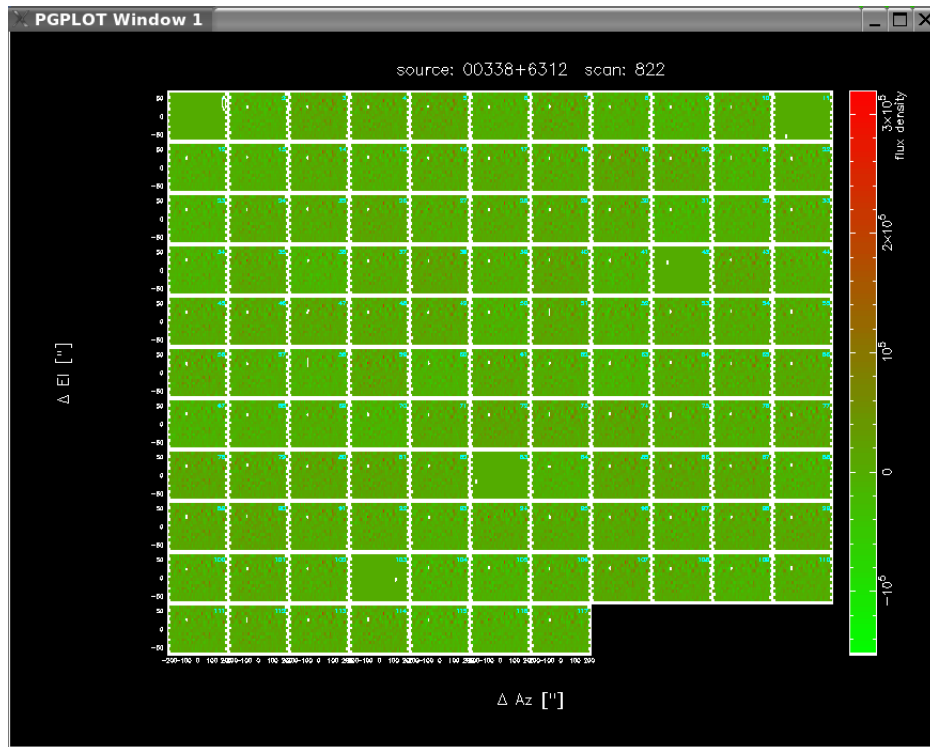


Figure 4.12.1: Default graphical outputs of a channel map of the source 00388+6312, including a wedge.

4.12.2 Displaying/re-displaying a map

METHOD: `display(optional arguments)`

DESCRIPTION: display the reconstructed map in (Az,El)

OPTIONAL ARGUMENTS:

<i>weight</i>	plot the weight map instead of signal map
<i>coverage</i>	plot the rms map instead of signal map
<i>style</i>	the style used for the color (default idl4)
<i>caption</i>	the caption of the plot (default "")
<i>limitsX</i>	range of X values to be plotted (comma separated values, in square brackets)
<i>limitsY</i>	range of Y values to be plotted (comma separated values, in square brackets)
<i>limitsZ</i>	range of Z values to be plotted (comma separated values, in square brackets)
<i>wedge</i>	draw a wedge ? (default : yes)
<i>aspect</i>	keep the aspect ratio (default : yes)
<i>overplot</i>	should we overplot this image (default : no)
<i>doContour</i>	draw contour instead of map (default : no)
<i>levels</i>	the levels of the contours (default : intensity progression)
<i>labelContour</i>	label the contour (default : no)
<i>showRms</i>	compute and display rms/beam? (def: no)

Example:

Re-display a map after performing a smoothing (as in example in Section 3.7.1).

```
smooth(6./3600.) display(caption=data.ScanParam.caption())
```

METHOD: `showMap(optional arguments)`

DESCRIPTION: show the reconstructed map in (Az,El) or (Ra,Dec)

OPTIONAL ARGUMENTS:

<i>style</i>	the style used for the color (default idl4)
<i>caption</i>	the caption of the plot (default "")
<i>limitsX</i>	range of X values to be plotted (comma separated values, in square brackets)
<i>limitsY</i>	range of Y values to be plotted (comma separated values, in square brackets)
<i>limitsZ</i>	range of Z values to be plotted (comma separated values, in square brackets)
<i>doContour</i>	draw contour instead of map (default : no)
<i>wedge</i>	draw a wedge ? (default : yes)
<i>aspect</i>	keep the aspect ratio (default : yes)
<i>showRms</i>	compute and display rms/beam? (def: yes)
<i>rmsKappa</i>	kappa in kappa-sigma clipping used to compute rms

4.12.3 Plot the receiver parameters

METHOD: `plotArray(optional arguments)`

DESCRIPTION: plot the receiver layout

OPTIONAL ARGUMENTS:

overplot *overplot?*
num indicate channel numbers?

Example:

Plot the array layout with receiver numbers indicated. `plotArray(num=1)`

4.12.4 Plotting azimuth versus LST

METHOD: `plotAzimuth(optional arguments)`

DESCRIPTION: Plot the time series of the azimuth, i.e. azimuth versus LST.

OPTIONAL ARGUMENTS:

flag plot data flagged or unflagged accordingly
plotFlagged flag reverts to flagged/unflagged data
limitsX range of X values to be plotted (comma separated values, in square brackets)
limitsY range of Y values to be plotted (comma separated values, in square brackets)
style linestyle to be used ('p' or 'l', for points and solid line respectively)
ci colour index to be used (integer values)
overplot
aspect

A more detailed description of plotting related arguments can be found in Section [5.5](#).

Example:

```
azimuth(style='p', ci=2, limitsY=[-14,-13])
```

Plot azimuth versus LST (note the abbreviated form 'azimuth' used, see Table [4.1](#)). Show individual plotted points (rather than lines), make plotted points red, and only plot azimuth (y axis) from -14 to -13 degrees.

4.12.5 Plotting elevation versus azimuth

METHOD: `plotAzEl(optional arguments)`

DESCRIPTION: Plot elevation versus azimuth.

OPTIONAL ARGUMENTS: as for `plotAzimuth()`

Example:

```
as for plotAzimuth().
```

4.12.6 Plotting azimuth and elevation acceleration

METHOD: `plotAzElAcceleration(optional arguments)`

DESCRIPTION: Plot azimuth and elevation acceleration.

OPTIONAL ARGUMENTS: as for `plotAzimuth()`

Example:

as for `plotAzimuth()`.

4.12.7 Plotting elevation offset versus azimuth offset

METHOD: `plotAzElOffset` (*optional arguments*)

DESCRIPTION: Plot elevation offset versus azimuth offset.

OPTIONAL ARGUMENTS: as for `plotAzimuth()`

Example:

as for `plotAzimuth()`.

4.12.8 Plotting azimuth and elevation speed

METHOD: `plotAzElSpeed` (*optional arguments*)

DESCRIPTION: Plot azimuth and elevation speed.

OPTIONAL ARGUMENTS: as for `plotAzimuth()`

Example:

as for `plotAzimuth()`.

4.12.9 Plotting azimuth offset versus LST

METHOD: `plotAzimuthOffset` (*optional arguments*)

DESCRIPTION: Plot azimuth offset versus LST.

OPTIONAL ARGUMENTS: as for `plotAzimuth()`

Example:

as for `plotAzimuth()`.

4.12.10 Plot flux density of channels versus reference channel

METHOD: `plotCorrel` (*optional argument*)

DESCRIPTION: plot flux density of a list of channels vs. flux density of a reference channel

OPTIONAL ARGUMENTS:

<i>chanRef</i>	reference channel number (default: is the first in chanList)
<i>chanList</i>	list of channels, of the form [1,2,3]
<i>channelFlag</i>	plot data from channels flagged or unflagged accordingly
<i>plotFlaggedChannels</i>	channelFlag reverts to flagged/unflagged data
<i>dataFlag</i>	plot data flagged or unflagged accordingly
<i>plotFlaggedData</i>	dataFlag reverts to flagged/unflagged data
<i>skynoise</i>	plot against the skynoise of chanRef (default : no)
<i>limitsX</i>	range of X values to be plotted (comma separated values in [])
<i>limitsY</i>	range of Y values to be plotted (comma separated values in [])
<i>style</i>	linestyle to be used ('p' or 'l', for points and solid line respectively)
<i>ci</i>	colour index to be used (integer values)
<i>overplot</i>	

4.12.11 Plotting elevation versus LST

METHOD: `plotElevation(optional arguments)`

DESCRIPTION: Plot the time series of the elevation i.e. elevation versus LST.

OPTIONAL ARGUMENTS: as for `plotAzimuth()`

Example:

```
as for plotAzimuth().
```

4.12.12 Plotting elevation offset versus LST

METHOD: `plotElevationOffset(optional arguments)`

DESCRIPTION: Plot elevation offset versus LST.

OPTIONAL ARGUMENTS: as for `plotAzimuth()`

Example:

```
as for plotAzimuth().
```

4.12.13 Plotting the FFT of the signal

METHOD: `plotFFT(optional arguments)`

DESCRIPTION: Plot a Fast Fourier Transform (FFT) of the signal

OPTIONAL ARGUMENTS:

<i>labelX</i>	the X label; default is <i>Frequency [Hz]</i>
<i>labelY</i>	the Y label; default is <i>Amplitude (a.b.u/sqrt(Hz))</i>
<i>limitsX</i>	range of X values to be plotted (comma separated values in [])
<i>limitsY</i>	range of Y values to be plotted (comma separated values in [])
<i>plotphase</i>	plot phase instead of amplitude (default no)
<i>ci</i>	colour index to be used (integer values)
<i>overplot</i>	overplot on previous plot?
<i>windowSize</i>	number of samples on which FFT are computed, then averaged for display (default: 0, no averaging)
<i>windowing</i>	code for the window function applied during computation (default: 3, Hanning function)

Example:

Plot FFT for the first 9 channels.

```
plotFFT(range(10))
```

4.12.14 Plot mean flux versus subscan number

METHOD: `plotMean()` (*optional argument*)

DESCRIPTION: plot mean flux value vs. subscan number

OPTIONAL ARGUMENTS:

<i>chanList</i>	list of channels
<i>map</i>	plot as a 2D map?

4.12.15 Plot mean channel values versus channel number

METHOD: `plotMeanChan()` (*optional argument*)

DESCRIPTION: plot the MEAN value for each subscan against channel number

OPTIONAL ARGUMENTS:

<i>chanList</i>	list of channels
<i>limitsX</i>	range of X values to be plotted (comma separated values in [])
<i>limitsY</i>	range of Y values to be plotted (comma separated values in [])
<i>style</i>	linestyle to be used ('p' or 'l', for points and solid line respectively)
<i>ci</i>	colour index to be used (integer values)
<i>overplot</i>	

4.12.16 Plot flux Rms versus subscan number

METHOD: `plotRms()` (*optional argument*)

DESCRIPTION: plot flux r.m.s. vs. subscan number

OPTIONAL ARGUMENTS:

chanList list of channels
map plot as a 2D map?

4.12.17 Plotting RMS versus channel number

METHOD: `plotRmsChan` (*optional arguments*)

DESCRIPTION: plot the RMS value for each subscan against channel number

OPTIONAL ARGUMENTS:

<i>chanList</i>	list of channels
<i>channelFlag</i>	plot data from channels flagged or unflagged accordingly
<i>plotFlaggedChannels</i>	<i>channelFlag</i> reverts to flagged/unflagged data
<i>dataFlag</i>	plot data flagged or unflagged accordingly
<i>plotFlaggedData</i>	<i>dataFlag</i> reverts to flagged/unflagged data
<i>limitsX</i>	range of X values to be plotted (of the form [1,2])
<i>limitsY</i>	range of Y values to be plotted (of the form [1,2])
<i>style</i>	linestyle to be used ('p' or 'l', for points and solid line respectively)
<i>ci</i>	colour index to be used (integer values)
<i>overplot</i>	
<i>subscan</i>	if 0, plot rms of the complete scan (default); if 1, plot for each subscan and each channel

4.12.18 Display start and end times of subscans

METHOD: `plotSubscan()`

DESCRIPTION: generate a plot showing starting and ending times of subscans

4.12.19 Plot subscans on the Az-El pattern

METHOD: `plotSubscanOffsets()`

DESCRIPTION: Use four colours to show subscans on the Az, El pattern

OPTIONAL ARGUMENTS:

overplot if set, do not plot AzElOffset – assume these have been plotted already

4.12.20 Plotting flux density versus LST

METHOD: `signal` (*optional argument*)

DESCRIPTION: Plot the time series of the flux density i.e. flux density versus LST.

OPTIONAL ARGUMENTS:

<i>chanList</i>	list of channels, of the form [1,2,3]
<i>channelFlag</i>	plot data from channels flagged or unflagged accordingly
<i>plotFlaggedChannels</i>	channelFlag revers to flagged/unflagged data
<i>dataFlag</i>	plot data flagged or unflagged accordingly
<i>plotFlaggedData</i>	dataFlag revers to flagged/unflagged data
<i>skynoise</i>	plot correlated noise (default 0)
<i>caption</i>	plot title, default = scan info
<i>limitsX</i>	range of X values to be plotted (of the form [14,16])
<i>limitsY</i>	range of Y values to be plotted (of the form [14,16])
<i>style</i>	linestyle to be used ('p' or 'l', for points and solid line respectively)
<i>ci</i>	colour index to be used (integer values)
<i>overplot</i>	

A more detailed description of the plotting related arguments can be found in Section 5.5.

Example:

```
signal(chanList=[18,19,20], mjd=1, style='p', ci=2)
signal([18,19,20], mjd=1, style='p', ci=2)
```

4.13 Data handling

4.13.1 Get a list of valid channels

METHOD: `checkChanList (optional argument)`

DESCRIPTION: Return a list of valid channels

OPTIONAL ARGUMENTS:

<i>inList</i>	list of channel numbers to get, or empty list to get the complete list of unflagged channels, or 'all' or 'al' or 'a' to get the complete list of channels
<i>flag</i>	retrieve data flagged or unflagged accordingly
<i>getFlagged</i>	flag revers to flagged/unflagged data

4.13.2 Get pixel values

METHOD: `getPixel()`

DESCRIPTION: get pixel values using mouse

OPTIONAL ARGUMENTS:

nbPix size of area to compute average (default 3x3)

Click left to get one pixel, mid to get average over 9, right to exit (on Data array only).

4.13.3 Print the current list of channels

METHOD: `printCurrChanList()`

DESCRIPTION: Print the current list of channels

4.13.4 Selecting channels

METHOD: `setCurrChanList(optional argument)`

DESCRIPTION: Set a channel or a list of channels to be treated.

OPTIONAL ARGUMENTS:

<i>chanList:</i>	list of channel numbers, of the form: [1,2,3]
'all'... 'al'...'a'	set current list to all possible channels
'?'	get current list of channels (default)

Example:

```
Using the abbreviated form channels() (see Table 4.1): channels([1, 2, 3])  
channels(chanList=[1, 2, 3])  
channels('all')  
channels('?')
```

4.14 User methods for selecting files and directories

4.14.1 Listing the contents of the input directory

METHOD: `listInDir()`

DESCRIPTION: list the contents of the input directory

4.14.2 Resetting the CurrentList

METHOD: `resetCurrentList()`

DESCRIPTION: reset the CurrentList to the complete List

4.14.3 Setting the input directory

METHOD: `setInDir()`

DESCRIPTION: set the input directory

Example:

```
setInDir('inputDirectory')
```

4.14.4 Setting the output directory

METHOD: `setOutDir()`

DESCRIPTION: set the input directory

Example:

```
setOutDir('outputDirectory')
```

4.14.5 Setting the project ID

METHOD: `setProjectID()`

DESCRIPTION: set the current project ID

Example:

```
setProjectID('projectID')
```

4.15 Miscellaneous methods

4.15.1 Updating offset and gain values from a file

METHOD: `updateRCP(rcpFile)`

DESCRIPTION: update only offsets and gains from the content of a file

INPUT:

rcpFile complete name of file to read in

4.16 Scripts

As **BoA** provides the full functionality of Python this allows the use of scripts. Scripts can be run with the `execfile()` function where the name of the file has to be given as string argument. The suffix of the file is arbitrary.

Example:

If you want to have a look at the time series of channels 10 to 30 successively, create the following script with your preferred editor. Note that in Python the contents of the for loop (like if blocks, method definitions, etc.) have to be indented.

```
# testBoa.py
indir('../Fits/')      # set the input directory
read('test')           # read file test.fits
op()                   # open graphic display
```

```
for i in range(10,31): # start a for loop, the indentation in
                        # the following lines is mandatory
sig([i])               # plot time series
raw_input()            # wait for <Return>
```

To run the script type:

```
execfile('testBoa.py')
```

4.16.1 Example scripts

In order to demonstrate some of the basic functionalities of **BoA** three demonstration scripts are provided: *ExampleMap.py*, *ExamplePointing.py* and *ExampleFocus.py*. These can be found in the directory `/home/user/boa/examples/` and are described in detail in Chapter 3. Run the scripts by typing:

```
execfile('/home/user/boa/examples/ExampleMap.py')
```


4.17 Commands in alphabetical order

blankFreq	permanently remove some frequency interval in the Fourier spectrum of the signal
chanMap	plot channel maps
checkChanList	return a list of valid channels
clear	clear the active plot window
closeDev	close one device
computeRms	compute rms/beam in a map
computeWeight	compute weights (default is use $1/\text{rms}^2$)
correctOpacity	correct for atmospheric opacity
despike	flag yet unflagged data below and above given rms values
display	show the reconstructed maps in (Az,El)
doMap	construct a map in (Az,El) coordinates
flagAccel	flag data according to telescope acceleration
flagChannels	flag a list of channels
flagInTime	flag data in time interval
flagMJD	flag data in MJD time interval
flagPosition	flag a position in the sky within a given radius
flagRms	flag channels with rms below or above respective given values
flagSpeed	flag data according to telescope speed
flagSubscan	flag certain subscans
flatfield	divide signals by bolometer gains to normalise them
flattenFreq	flatten the 1/F part of the FFT using constant amplitude
getPixel	get pixel values using mouse
iterativeDespike	iteratively flag yet unflagged data below and above given rms values
listInDir	list the input directory
medianBaseline	baseline: Remove median value per channel and per subscan
medianNoiseRemoval	remove median noise from the data
openDev	open a graphic device
plotArray	plot the receiver layout
plotAzEl	plot elevation versus azimuth
plotAzElAcceleration	plot azimuth and elevation acceleration
plotAzElOffset	plot elevation offset versus azimuth offset
plotAzElSpeed	plot azimuth and elevation speed
plotAzimuth	plot azimuth versus LST
plotAzimuthOffset	plot azimuth offset versus LST

plotCorrel	plot signal vs. reference channel
plotElevation	plot elevation versus LST
plotElevationOffset	plot elevation offset versus LST
plotFFT	plot FFT of signal
plotMean	plot mean flux values vs. subscan numbers
plotMeanChan	plot mean value for each subscan vs. chan. number
plotRms	plot rms flux values vs. subscan numbers
plotRmsChan	plot rms value for each subscan vs. chan. number
plotSubscan	generate a plot showing starting and ending times of sub-scans
plotSubscanOffsets	use four colours to show subscans on the Az, El pattern
polynomialBaseline	remove a polynomial baseline from the data
printCurrChanList	print the current channel list
read	read in a file
resetCurrentList	reset the CurrentList to the complete list
saveMambo	convert MB-Fits file to MAMBO format
selectDev	select an open device
setCurrChanList	select list of channels
setInDir	set the input directory
setOutDir	set the output directory
setProjectID	set the project ID
showMap	show the reconstructed map in (Az,El) or (Ra,Dec)
signal	plot the time series of the data (flux density versus LST)
smoothBy	smooth the image with a 2D gaussian of given FWHM
solveFocus	compute the optimal focus position
solvePointingOnMap	compute the offset on the data.Map object
statistics	prints the statistics
unflag	unflag data, i.e. reset flags to 0
unflagAccel	unflag data according to telescope acceleration
unflagChannels	unflag a list of channels
unflagInTime	unflag data in time interval
unflagMJD	unflag data in time interval
unflagPosition	unflag a position in the sky within a given radius
unflagSpeed	unflag data according to telescope speed
unflagSubscan	unflag a list of subscans
updateRCP	update offsets and gains from the content of a file

4.18 Commands in functional order

4.18.1 Plotting and displaying

chanMap	plot channel maps
display	show the reconstructed maps in (Az,El)
plotArray	plot the receiver layout
plotAzEl	plot elevation versus azimuth
plotAzElAcceleration	plot azimuth and elevation acceleration
plotAzElOffset	plot elevation offset versus azimuth offset
plotAzElSpeed	plot azimuth and elevation speed
plotAzimuth	plot azimuth versus LST
plotAzimuthOffset	plot azimuth offset versus LST
plotCorrel	plot signal vs. reference channel
plotElevation	plot elevation versus LST
plotElevationOffset	plot elevation offset versus LST
plotFFT	plot FFT of signal
plotMean	plot mean flux values vs. subscan numbers
plotMeanChan	plot mean value for each subscan vs. chan. number
plotRms	plot rms flux values vs. subscan numbers
plotRmsChan	plot rms value for each subscan vs. chan. number
plotSubscan	generate a plot showing starting and ending times of sub-scans
plotSubscanOffsets	Use four colours to show subscans on the Az, El pattern
showMap	show the reconstructed map in (Az,El) or (Ra,Dec)
signal	plot the time series of the data (flux density versus LST)

4.18.2 Device handling

clear	clear the active plot window
closeDev	close one device
openDev	open a graphic device
selectDev	select an open device

4.18.3 Pointing and focus

solveFocus	compute the optimal focus position
solvePointingOnMap	compute the offset on the data.Map object

4.18.4 Flagging and despiking data

blankFreq	permanently remove some frequency interval in the Fourier spectrum of the signal
despike	flag yet unflagged data below 'below'*rms and above 'above'*rms
flagAccel	flag data according to telescope acceleration
flagChannels	flag a list of channels
flagInTime	flag data in time interval
flagMJD	flag data by time interval
flagPosition	flag a position in the sky within a given radius
flagRms	flag channels with rms below or above respective given values
flagSpeed	flag data according to telescope speed
flagSubscan	flag certain subscans
flattenFreq	flatten the 1/F part of the FFT using constant amplitude
iterativeDespike	iteratively flag yet unflagged data below and above given rms values
unflag	unflag data
unflagAccel	unflag data according to telescope acceleration
unflagChannels	unflag a list of channels
unflagInTime	unflag data in time interval
unflagMJD	unflag data in time interval
unflagPosition	unflag a position in the sky within a given radius
unflagSpeed	unflag data according to telescope speed
unflagSubscan	unflag a list of subscans

4.18.5 Map making

doMap	construct a map in (Az,El) coordinates
horizontalMap	construct a map in (Az,El) coordinates

4.18.6 Flatfield and opacity correction

correctOpacity	correct for atmospheric opacity
flatfield	divide signals by bolometer gains to normalise them

4.18.7 Baseline subtraction, sky removal and statistics

computeRms	compute rms/beam in a map
computeWeight	compute weights (default is use $1/\text{rms}^2$)
medianBaseline	baseline: Remove median value per channel and per sub-scan
medianNoiseRemoval	remove median noise from the data
polynomialBaseline	remove a polynomial baseline from the data
smoothBy	smooth the image with a 2D gaussian of given FWHM
statistics	prints the statistics

4.18.8 File handling

read	read in a file
saveMambo	convert MB-Fits file to MAMBO format

4.18.9 Data handling

checkChanList	return a list of valid channels
getPixel	allow user to get pixel values using mouse
printCurrChanList	print the current channel list
setCurrChanList	select list of channels

4.18.10 Selecting files and directories

listInDir	list the input directory
resetCurrentList	reset the CurrentList to the complete list
setInDir	set the input directory
setOutDir	set the output directory
setProjectID	set the project ID

4.18.11 Misc.

updateRCP	update offsets and gains from the content of a file
------------------	---

4.19 Abbreviations

As we have noted already, user methods are abbreviations of the full methods. For example, the method `DeviceHandler.openDev()` can be called by the user method `op()`. For further convenience, most user methods can also be called by even shorter abbreviations of the user methods (e.g. `sig()` is all that is needed for `signal()`). A list of user methods and their abbreviations is given in Table 4.1.

Command	Abbreviations
<code>chanMap</code>	<code>ChanMap</code> - <code>chanmap</code>
<code>checkChanList</code>	<code>checkChannels</code> - <code>checkChan</code>
<code>clear</code>	<code>cle</code> - <code>cl</code>
<code>closeDev</code>	<code>close</code> - <code>clo</code> - <code>cls</code>
<code>computeRms</code>	<code>maprms</code>
<code>computeWeight</code>	<code>computeweight</code> - <code>weight</code>
<code>correlatedNoiseRemoval</code>	<code>cnr</code> - <code>CNR</code>
<code>corrPCA</code>	<code>corr pca</code> - <code>pca</code> - <code>PCA</code>
<code>despike</code>	<code>dspike</code>
<code>display</code>	<code>mapdisp</code> - <code>mapdisplay</code>
<code>doMap</code>	<code>mapping</code> - <code>doMap</code> - <code>domap</code>
<code>dumpData</code>	<code>dump</code>
<code>findInDir</code>	<code>find</code> - <code>fd</code>
<code>flag</code>	
<code>flagChannels</code>	<code>flagCh</code> - <code>flagC</code> - <code>fCh</code>
<code>flagLon</code>	
<code>flagMJD</code>	
<code>flagPosition</code>	<code>flagPos</code>
<code>flagRms</code>	
<code>flagSubscan</code>	<code>flagSub</code>
<code>flatfield</code>	<code>flat</code>
<code>getPixel</code>	<code>getPix</code>
<code>iterativeDespike</code>	<code>itDespike</code>
<code>listInDir</code>	<code>ils</code> - <code>inls</code>
<code>mapSum</code>	<code>mapsum</code>
<code>medianBaseline</code>	<code>medianBase</code> - <code>medianbase</code>
<code>medianNoiseRemoval</code>	<code>mediannoise</code>
<code>correctOpacity</code>	<code>opacity</code> - <code>opac</code>
<code>openDev</code>	<code>op</code>
<code>plotArray</code>	<code>plotarray</code>
<code>plotAzEl</code>	<code>azel</code>
<code>plotAzElAcceleration</code>	<code>azelaccel</code> - <code>azelac</code>
<code>plotAzElOffset</code>	<code>azeloff</code> - <code>azelo</code>

Table 4.1: List of user methods with abbreviations. Don't forget to add the round brackets () at the end of the commands.

Command	Abbreviations
plotAzElSpeed	azelspeed - azelsp
plotAzimuth	azimuth - azimuth - az
plotAzimuthOffset	azimuthOffset - azimuthoff - azo
plotCorrel	plotcorrel - plotcor - plotCor
plotElevation	elevation - elev - el
plotElevationOffset	elevationOffset - eleoff - elo
plotFFT	
plotMean	plotmean
plotMeanChan	plotmeanchan
plotRms	plotrms
plotRmsChan	plotrmschan
plotSubscan	plotSub
plotSubscanOffsets	plotSubOff
pointSize	
polynomialBaseline	baseline - base
printCurrChanList	printChannels - printChan
readRCPfile	readRCP - rcf
read	
removeScans	remove - rs
resetCurrentList	resetCurrList - rls
restoreData	restore
selectDev	device - dev
selectInDir	select - slt
setCurrChanList	channels - channel - chan
setInDir	indir - ind
setInFile	infile - inf
setOutFile	outfile - outf
setOutDir	outdir - outd
setProjectID	setproj - proj
showMap	
signal	signa - sign - sig
solveFocus	solvefocus - solveFoc - solvefoc
solvePointingOnMap	solvepointing - solvepoint - solvepoin - solvepoi
smoothBy	smooth
statistics	stats - stat
unflag	
unflagChannels	unflagCh - unflagC - ufCh
updateRCP	
zoom	

Table 4.1: *continued*

5. BoGLi: THE BoA GRAPHIC LIBRARY

5.1 Introduction

The **BoA** Graphic Library (**BoGLi**) is an object-oriented software package for the graphical display of data. It is written in Python and uses `ppgplot`, the python binding to `pgplot`. The main parts (classes) of the software are self-consistent and may independently be used from any python programme. Nevertheless, **BoGLi** comes with features which especially customise its use for the display of astronomical data from multi-channel receivers. Its main goal is to provide a graphic tool tailored for the use with **BoA** for the display of data from LABOCA and other bolometer arrays.

5.2 BoGLi commands

Table 5.1 gives an overview of some of the available commands. **BoGLi** commands provide a variety of keywords that may be changed by the user (see Sect. 5.5 for details).

Table 5.1: List of useful **BoGLi** commands.

<code>DeviceHandler.openDev</code>	open a device
<code>DeviceHandler.closeDev</code>	close a device
<code>Plot.clear</code>	clear the active plot window
<code>DeviceHandler.selectDev</code>	select a device
<code>DeviceHandler.resizeDev</code>	resize the plotting area, after plot window resized using mouse
<code>Plot.plot</code>	make a single plot
<code>MultiPlot.plot</code>	plot multiple plots
<code>Plot.draw</code>	draw on an image
<code>MultiPlot.draw</code>	draw on plots of multiple channels

5.3 Device handling

BoGLi is based on `pgplot` and as a consequence the number and type of available devices depends on the actual configuration. A list of supported devices is given at <http://www.astro.caltech.edu/tjp/pgplot/devices.html>. During installation the device drivers have to be selected by editing the file `drivers.list`. As many device drivers are available on selected operating systems only, you should ensure that drivers you do not want are commented out (place `!` in column 1) to avoid installation failures.

BoGLi provides a set of commands to manage output devices. A detailed description of these commands is given below.

5.3.1 Opening a plot window

DESCRIPTION: Open a graphics device for `pgplot` output and make it the current device. The default, when no argument is provided, is to open an XWindow.

USAGE: `DeviceHandler.openDev (optional argument)`

The relevant abbreviations can also be used (see Table 4.1).

OPTIONAL ARGUMENT: *pgplot device type*

If the device is opened successfully, it becomes the selected device to which graphics output is directed until another device is selected (see 5.3.4) or the device is closed (see 5.3.2). If no device argument is specified PGPLOT will open the default graphics device (an XWINDOW). Alternatively, the graphics device may be selected using any of the following as arguments:

- (1) A complete device specification of the form `'device/type'` or `'file/type'`, where `/type` is one of the allowed PGPLOT device types (installation-dependent, e.g. `/xwindow`) and `'device'` or `'file'` is the name of a graphics device or disk file appropriate for this type. The `'device'` or `'file'` may contain `'/'` characters; the final `'/'` delimits the `'type'`. If necessary to avoid ambiguity, the `'device'` part of the string may be enclosed in double quotation marks.

Example: `'plot.ps/ps'`, `'dir/plot.ps/ps'`, `'"dir/plot.ps"/ps'`,
`'user:[tjp.plots]plot.ps/PS'`

- (2) A device specification of the form `'/type'`, where `/type` is one of the allowed PGPLOT device types, e.g. `/xwindow`. PGPLOT supplies a default file or device name appropriate for this device type.

Example: `'/ps'` (PGPLOT interprets this as `'pgplot.ps/ps'`)

- (3) A device specification with `'/type'` omitted; in this case the type is taken from the environment variable `PGPLOT_TYPE`, if defined (e.g., `setenv PGPLOT_TYPE PS`). Because of possible confusion with `'/'` in file-names, omitting the device type in this way is not recommended.

Example: `'plot.ps'` (if `PGPLOT_TYPE` is defined as `'ps'`, PGPLOT interprets this as `'plot.ps/ps'`)

- (4) A blank string (`' '`); in this case, `PGOPEN` will use the value of environment variable `PGPLOT_DEV` as the device specification, or `'/NULL'` if the environment variable is undefined.

Example: ' ' (if PGPLOT_DEV is defined)

- (5) A single question mark, with optional trailing spaces, i.e. ('?'). In this case, PGPLOT will prompt the user to supply the device specification, with a prompt string of the form 'Graphics device/type (? to see list, default XXX):' where 'XXX' is the default (value of environment variable PGPLOT_DEV).

Example: ' ? '

- (6) A non-blank string in which the first character is a question mark (e.g. '?Device: '); in this case, PGPLOT will prompt the user to supply the device specification, using the supplied string as the prompt (without the leading question mark but including any trailing spaces).

Example: '?Device specification for PGPLOT: '

In cases (5) and (6), the device specification is read from the standard input. The user should respond to the prompt with a device specification of the form (1), (2), or (3). If the user enters a question-mark in response to the prompt, a list of available device types is displayed and the prompt is re-issued. If the user supplies an invalid device specification, the prompt is re-issued. If the user responds with an end-of-file character, e.g., ctrl-D in UNIX, program execution is aborted; this avoids the possibility of an infinite prompting loop. A programmer should avoid use of PGPLOT-prompting if this behavior is not desirable.

The device type is case-insensitive (e.g., '/ps' and '/PS' are equivalent). The device or file name may be case-sensitive in some operating systems.

5.3.2 Closing a plot window

DESCRIPTION: Close a plotting device. The default, where no argument is supplied, is to close the current device.

USAGE: `DeviceHandler.closeDev (optional argument)`

OPTIONAL ARGUMENT:

device number (integer)
'all'
'current'...'curre'...'cur'

Example:

<code>DeviceHandler.closeDev(2)</code>	Close the device with identifier 2
<code>DeviceHandler.closeDev('all')</code>	close all devices
<code>DeviceHandler.closeDev('current')</code>	close current device (the default if no argument specified)

5.3.3 Clearing a plot window

DESCRIPTION: Clear the output of the current device. To clear the output of a different device change to that device first (see 5.3.4).

USAGE: `Plot.clear()`

5.3.4 Selecting a device

DESCRIPTION: Select an open device for graphical output. The selected device has to be previously opened with *open* (see 5.3.1).

USAGE: `DeviceHandler.selectDev(argument)`

ARGUMENT: *device number* (integer)

Example:

```
DeviceHandler.selectDev(2)  Make device number 2 the current device for  
                           graphical output
```

5.3.5 Resizing a device

DESCRIPTION: Resize the plotting area after resizing of the graphics display window using the mouse. This is applicable to some interactive devices (e.g. /xwindow).

USAGE: `DeviceHandler.resizeDev()`

5.4 Plotting graphics

This section lists some of the graphics plotting capabilities of **BoGLi**.

5.4.1 Plotting single plots

DESCRIPTION: Make a single plot of x versus (optional) y.

USAGE: `Plot.plot(dataX, [dataY, limitsX, limitsY, labelX, labelY, caption, style, ci, width, overplot, aspect, logX, logY, nodata])`

ARGUMENTS:

dataX values to plot along X

dataY values to plot along Y (optional - default: plot dataX vs. running number)

OPTIONAL ARGUMENTS:

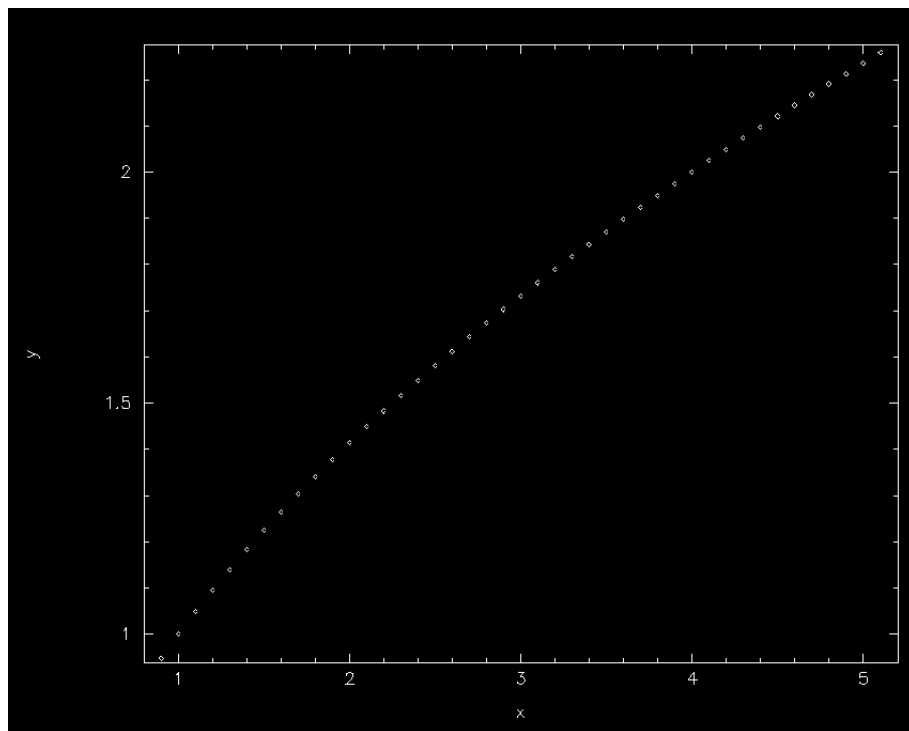


Figure 5.4.1: Example 1 of graphics produced using Plot.plot

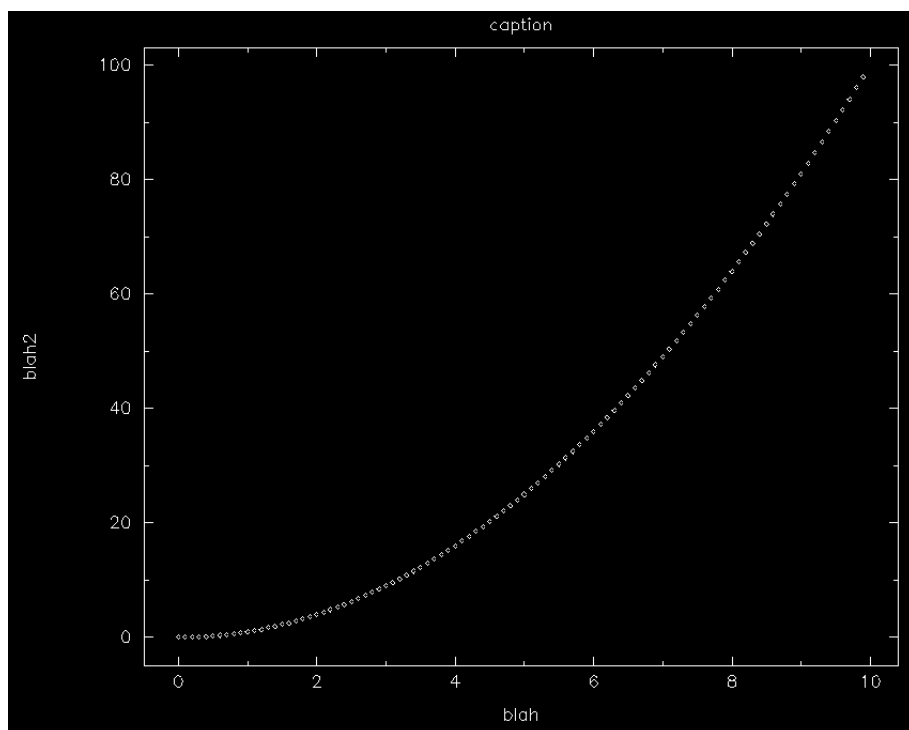


Figure 5.4.2: Example 2 of graphics produced using Plot.plot

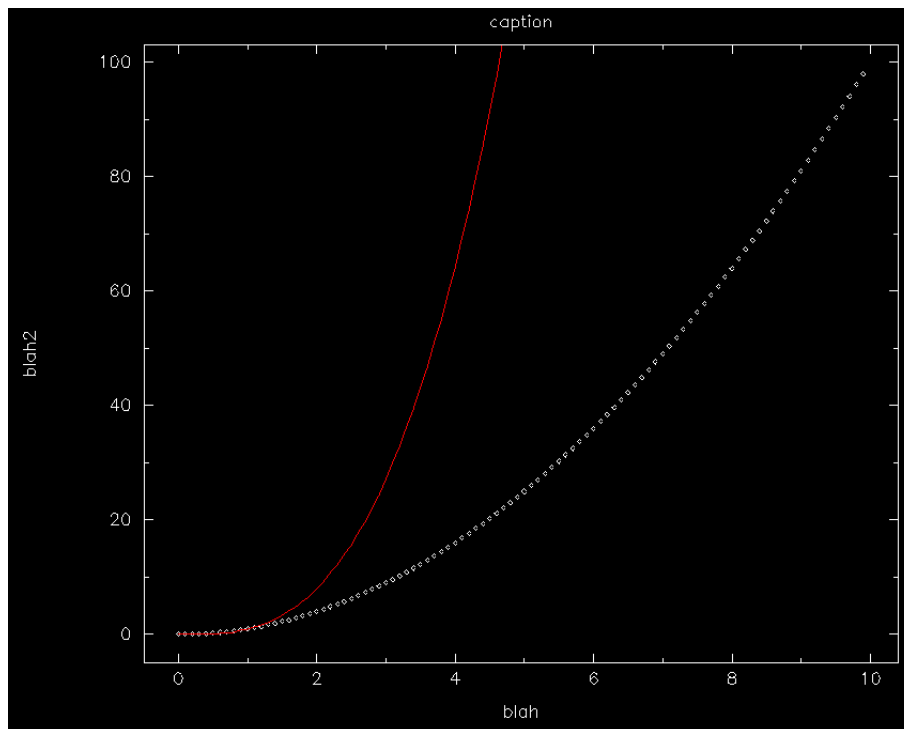


Figure 5.4.3: Example 3 of graphics produced using Plot.plot

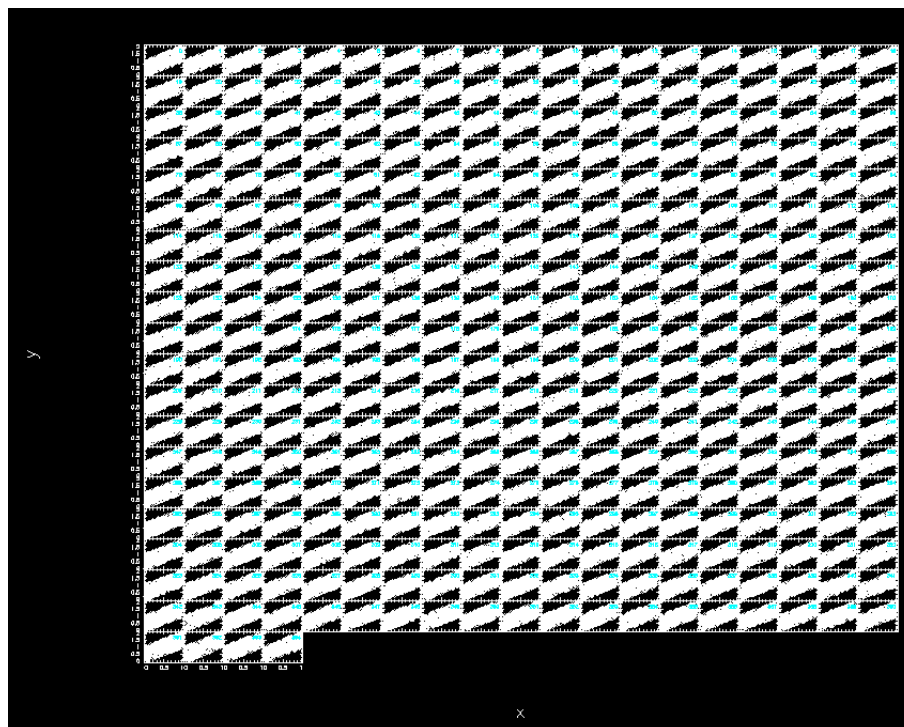


Figure 5.4.4: Example of graphics produced using MultiPlot.plot

<i>limitsX</i>	limits to use in X for the plot
<i>limitsY</i>	limits to use in Y for the plot
<i>labelX</i>	x label (default 'x')
<i>labelY</i>	y label (default 'y')
<i>caption</i>	the caption of the plot (default '')
<i>style</i>	the style used for the plot ('l': line, 'p': point (default), 'b': histogram)
<i>ci</i>	color index (default 1)
<i>width</i>	linewidth (default 0 = use previous)
<i>aspect</i>	keep the aspect ratio in 'physical' unit
<i>overplot</i>	set overplot=1 to overplot (default no)
<i>logX</i>	set logX=1 to use a log scale (default no)
<i>logY</i>	set logY=1 to use a log scale (default no)

These are also described in Section 5.5. Note *dataY* is also optional – if no *dataY* is supplied the default is to plot *dataX* versus running number.

Example:

```
x = Numeric.array(range(100), Numeric.Float)/10
```

```
Plot.plot(x, Numeric.sqrt(x), limitsX=[1, 5])
```

Note that Y limits are then computed according to this X range.

The graphic output produced in this case is shown in Figure 5.4.1.

Example:

```
Plot.plot(x, x*x, labelX='blah', labelY='blah2', caption='caption')
```

Note that plot clear the screen first, you need to use the new 'overplot' keyword (see below).

The graphic output produced in this case is shown in Figure 5.4.2.

Example:

```
Plot.plot(x, x*x*x, overplot=1, ci=2, style='l')
```

The graphic output produced in this case is shown in Figure 5.4.3.

5.4.2 Plotting multiple channels

DESCRIPTION: Make a plot of x versus (optional) y for several channels simultaneously.

USAGE: `MultiPlot.plot(chanList, dataX, dataY, [limitsX, limitsY, labelX, labelY, caption, style, ci, overplot, logX, logY, nan])`

ARGUMENTS:

chanList list of labels, of the form [1,2,3] or ['A','B','C']
dataX values to plot along X (list of lists, or list of arrays)
dataY values to plot along Y (list of lists, or list of arrays)

OPTIONAL ARGUMENTS:

limitsX limits to use in X for the plot
limitsY limits to use in Y for the plot
labelX x label (default 'x')
labelY y label (default 'y')
caption the caption of the plot (default '')
style the style used for the plot ('l': line, 'p': point (default), 'b': histogram)
ci color index (default 1)
overplot set overplot=1 to overplot (default no)
logX set logX=1 to use a log scale (default no)
logY set logY=1 to use a log scale (default no)

These are also described in [Section 5.5](#).

Example:

```
n_point = 365
chanlist=range(n_point)

x2 = RandomArray.random([n_point,n_point])
y2 = RandomArray.random([n_point,n_point])

MultiPlot.plot(chanlist,x2,y2+x2,style='p')
```

The graphic output produced in this case is shown in [Figure 5.4.4](#).

5.4.3 Drawing on an image

DESCRIPTION: Draw on an image

USAGE: `Plot.draw(map_array, [sizeX, sizeY, WCS, limitsX, limitsY, limitsZ, nan, labelX, labelY, caption, style, contrast, brightness, wedge, overplot, aspect, doContour, levels, labelContour])`

ARGUMENTS:

map_array map to display

OPTIONAL ARGUMENTS:

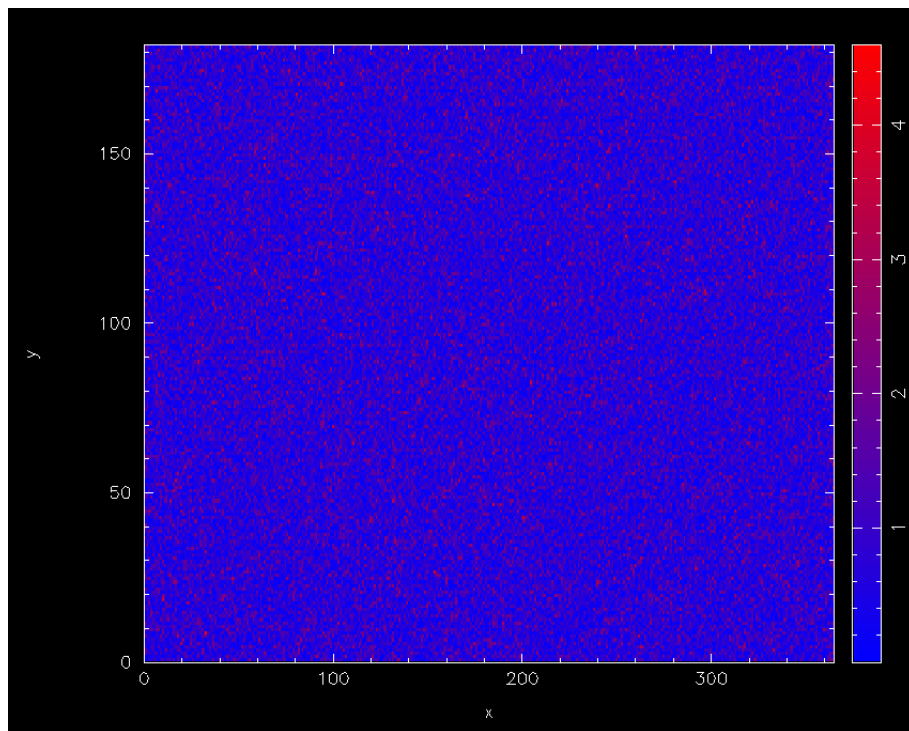


Figure 5.4.5: Example 1 of graphics produced using Plot.draw

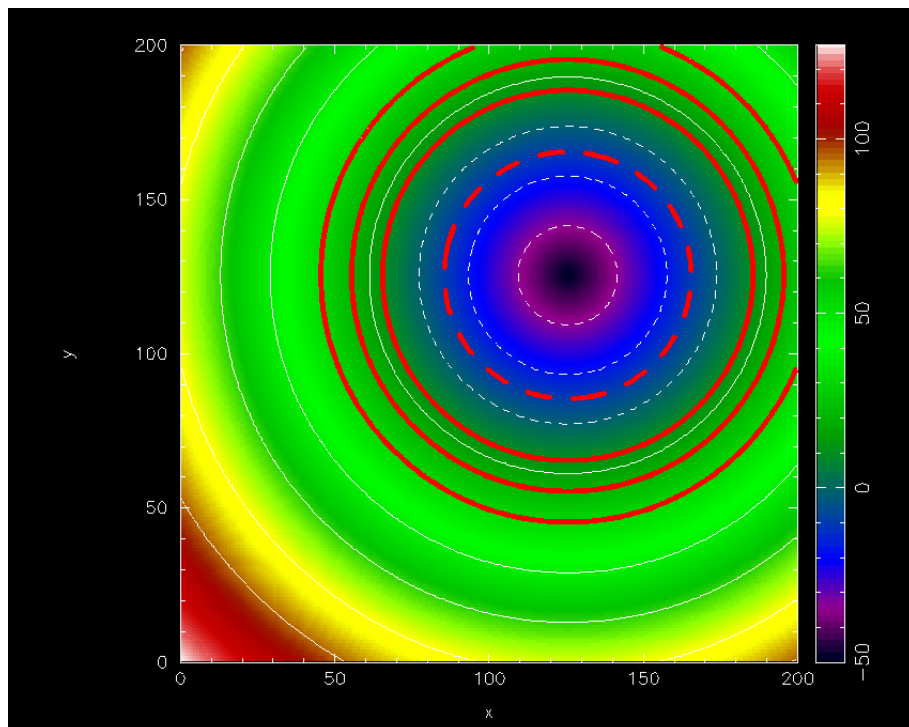


Figure 5.4.6: Example 2 of graphics produced using Plot.draw: drawing contours

<i>sizeX</i>	the 'physical' size of the array (default pixel numbers), defined by the center of the two extreme pixels
<i>sizeY</i>	the 'physical' size of the array (default pixel numbers), defined by the center of the two extreme pixels
<i>limitsX</i>	limits to use in X for the plot
<i>limitsY</i>	limits to use in Y for the plot
<i>nan</i>	set =1 if NaN are present in the array
<i>labelX</i>	x label (default 'x')
<i>labelY</i>	y label (default 'y')
<i>caption</i>	the caption of the plot (default '')
<i>style</i>	the color used for the plot (default 'g2r', see <code>Plot.setImaCol()</code>)
<i>wedge</i>	set <code>wedge=1</code> to draw a wedge (default no)
<i>aspect</i>	keep the aspect ratio in 'physical' unit
<i>overplot</i>	set <code>overplot=1</code> to overplot (default no)
<i>doContour</i>	set =1 to draw contour instead of map (default no)
<i>levels</i>	the levels for the contours (default <code>nContour</code> , within <code>plotLimitsZ</code>)
<i>labelContour</i>	set =1 to label the contours (default no)

These arguments are also described in Section 5.5.

Example:

```
n_point = 365
mapping=Numeric.absolute(RandomArray.standard_normal([n_point,n_point/2]))
Plot.draw(mapping, style='b2r', wedge=1)
```

You can also define 'physical' unit for your plot and still use `limitsX/Y` and `aspect`:

```
Plot.draw(mapping, sizeX=[-1,1], sizeY=[-2,2], limitsY=[-1,1], aspect=1, wedge=1)
```

The graphic output produced in this case is shown in Figure 5.4.5.

Example: You can also use `Plot.draw()` to plot contours.

```
def dist(x,y):
    return (x-125)**2+(y-125)**2
image = Numeric.sqrt(Numeric.fromfunction(dist, (200,200)))-50
Plot.draw(image, wedge=1, aspect=1, style='rainbow') # display an image
Plot.draw(image, doContour=1, overplot=1)           # overlay some contours
Plot.contour['color'] = 2                           # change the colour and
Plot.contour['linewidth'] = 10                      # linewidth attributes
Plot.draw(image, doContour=1, overplot=1, levels=[-10,10,20,30])
    # plot some more contours with the new attributes
```

The graphic output produced in this case is shown in Figure 5.4.6.

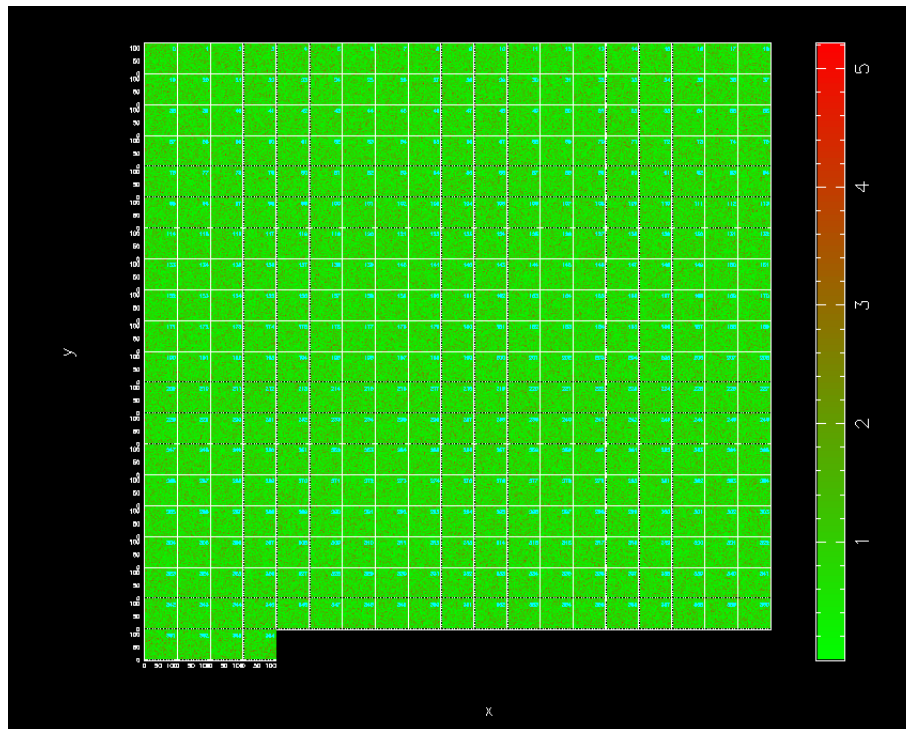


Figure 5.4.7: Example of graphics produced using MultiPlot.draw

5.4.4 Drawing on plots of multiple channels

DESCRIPTION: Draw on a multi-channel image

USAGE: `MultiPlot.plot.draw(chanList, map_arrays, [sizeX, sizeY, WCS, limitsX, limitsY, limitsZ, nan, labelX, labelY, caption, style, contrast, brightness, wedge, overplot])`

ARGUMENTS:

chanList list of channels
map_arrays lits of map to display

OPTIONAL ARGUMENTS:

sizeX the 'physical' size of the array (default pixel numbers)
sizeY the 'physical' size of the array (default pixel numbers)
limitsX limits to use in X for the plot
limitsY limits to use in Y for the plot
labelX x label (default 'x')
labelY y label (default 'y')
caption the caption of the plot (default ' ')
style the color used for the plot (default 'g2r', see `Plot.setImaCol()`)
wedge set wedge=1 to draw a wedge (default no)
overplot set overplot=1 to overplot (default no)

These are also described in [Section 5.5](#).

Example:

```
mapping_array = []
n_map = 365
for i in range(n_map):
    mapping_array.append(Numeric.absolute(RandomArray.standard_normal([120,120])))
MultiPlot.draw(range(n_map), mapping_array, wedge=1)
```

The graphic output produced in this case is shown in [Figure 5.4.7](#).

5.5 Keywords

BoGLi provides a variety of parameters which allow the graphical output to be customised, as regards primitives such as colours, linestyles, character sizes, as well as text output and general appearance.

ci *colour index*

The colour index is an integer in the range 0 to a device-dependent maximum. The default colour index is 1, usually white on a black background for monitor displays or black on a white background for printed hardcopies. Colour index 0 corresponds to the background colour. If the requested color index is not available on the selected device, colour index 1 will be used.

ls *line style*

The line style is an integer in the range 1 to 5 with the following codes:

- 1: full line
- 2: dashed
- 3: dot-dash-dot-dash
- 4: dotted
- 5: dash-dot-dot-dot

The line style does not affect graph markers, text, or area fill.

lw *line width*

The line width is specified in units of 1/200 (0.005) inch (about 0.13 mm) and must be an integer in the range 1-201. This parameter affects lines, graph markers and text.

limitsX *limits to use in X for the plot*

limitsY *limits to use in Y for the plot*

labelX *x label*
(default 'x')

labelY *y label (default 'y')*

caption *caption label*
(default ' ')

style	<i>linestyle</i> (<i>'l'</i> : line, <i>'p'</i> : point (default), <i>'b'</i> : histogram)
width	<i>linewidth</i> (default 0 = use previous)
aspect	<i>aspect ratio</i> keep the aspect ratio in 'physical' unit
overplot	<i>allow/prohibit overplotting</i> set overplot=1 to overplot (default no)
logX	<i>logarithmic scale</i> set logX=1 to use a log scale (default no)
logY	<i>logarithmic scale</i> set logY=1 to use a log scale (default no)
sizeX	<i>set the 'physical' size of the array</i> the 'physical' size of the array (default pixel numbers), defined by the center of the two extreme pixels
sizeY	<i>set the 'physical' size of the array</i> the 'physical' size of the array (default pixel numbers), defined by the center of the two extreme pixels
nan	set =1 if NaN are present in the array
wedge	set wedge=1 to draw a wedge (default no)
doContour	<i>draw contours</i> set =1 to draw contour instead of map (default no)
levels	<i>set the levels for the contours</i> the levels for the contours (default nContour, within plotLimitsZ)
labelContour	<i>label the contours</i> set =1 to label the contours (default no)

Part II

Reference Manual

6. DATA ORGANISATION

6.1 Data input: the MB-FITS format

A complete description of the Multi-Beam FITS Raw Data Format is given in the reference document APEX-MPI-IFD-0002. In this section, we only give a brief description of this file format.

6.1.1 The hierarchy for a full scan

For a given observing sequence, corresponding to one scan, a set of tables are generated and stored in a hierarchical way in the MB-FITS format. Three tables are created on top of this hierarchy, where informations related to the full scan are gathered:

- Primary header: here, some general informations are stored, such as telescope name, project ID, date of observation start, versions of MB-FITS format and FitsWriter software
- SCAN-MBFITS: the header of this table contains a description of the scan pattern (type, geometry, line length in case of a raster map...), the source name and coordinates, together with a description of the referential used, and some generic informations about the telescope (coordinates, pointing coefficients). In addition, a binary table lists the names of frontend-backend (hereafter FEBE) combinations in use for this observation.
- FEBEPAR-MBFITS: one such table is created for each FEBE in use (in general, only one FEBE is active for bolometer observing). It contains the FEBE name and the number of available channels for this FEBE in its header. The associated binary table gives all relevant information about the instrument: relative gains, positions, gain/attenuation factors, polarisation angles...

6.1.2 Tables for each subscan

For each subscan within a scan, three tables are generated:

- MONITOR-MBFITS: this table gathers all the monitoring information sent by the control system during the observation. Each datapoint has an associated timestamp in MJD. In particular, this monitor stream contains commanded and actual telescope positions sampled every 48 ms. It also contains data related to the weather conditions, the subreflector angle and position, and the LST values.
- DATAPAR-MBFITS: this table also contains the telescope positions, subreflector angles and positions, and LST values, but interpolated to the timestamps corresponding to the data stream.

It also contains a PHASE column, which can for example contains a succession of “ON” and “OFF” for a wobbler-switching observation.

- **ARRAYDATA-MBFITS**: here the raw data are stored. While some basic informations are stored in the header (e.g. central frequency of the observation), the binary table only contains two columns: the timestamps (in MJD), and a vector with length equal to the number of channels in use containing the raw data for each integration.

Note: in case several FEBE are in use at the same time, then a DATAPAR table and an ARRAYDATA table are generated for each subscan and for each FEBE.

6.2 BoAData objects

The manipulation of data within BoA is done with data objects of one class that inherits from the DataEntity class (Sect. 4.1.2; see also Section 6.2.1). Such objects contain the current version of the data, as well as associated parameters related to the scan and to the bolometer array. On top of this, the DataAna and Map classes define additional attributes, as described in the next subsections.

6.2.1 DataEntity

A DataEntity object has a number of attributes, listed in the following tables. Two of them are objects of classes BolometerArray and ScanParameter.

BolometerArray

The BolometerArray object defines the attributes listed in Table 6.1. They are read in from the file, or computed when reading, except for CurrChanList (contains the current list of channels on which any processing or plotting function is applied) and Flags (can be altered by the user).

Telescope

Attributes of a Telescope object are shown in Table 6.2.

ScanParam

Attributes of the ScanParam object (class ScanParameter) are listed in Table 6.3.

Data arrays

In addition to the scan parameters and bolometer array related informations, a DataEntity object contains some general informations about the observation, and 2D arrays of data and related numbers, with sizes number of pixels in use \times number of integrations. These are described in Table 6.4.

Table 6.1: Attributes of a BolometerArray object

Name	Type	Description
Telescope	object	see Table 6.2
FeBe	string	Frontend-Backend name
EffectiveFrequency	float	Observing frequency, in Hz
BeamSize	int	Beam size, in arcsec
BEGain	float	backend gain factor
FEGain	float	frontend gain factor
NChannels	int	Total number of pixels in the instrument
Gain	float array	1D array with relative gains (flat field)
Offsets	float array	relative (X,Y) offsets, in arcsec
Channel_Sep	float array	matrix of channel to channel separations, in arcsec
TransmitionCurve	float array	
Flags	int array	Flag value for each channel (0 = unflagged)
RefChannel	int	Reference channel number
NUsedChannels	int	Number of channels in use for this observation
UsedChannels	int array	List of channels in use for this observation
CurrChanList	int array	Current list of channel numbers

Table 6.2: Attributes of a Telescope object

Name	Type	Description
Name	str	Telescope name, e.g. APEX-12m
Diameter	float	Antenna diameter, in m
Latitude	float	Latitude, in deg
Longitude	float	Longitude, in deg
Elevation	float	Elevation, in m

6.2.2 DataAna

On top of the DataEntity, the DataAna layer defines additional attributes, related to statistics and flagging of the data. They are listed in Table 6.5.

6.2.3 Map

Finally, any kind of observation is stored in **BoA** in a Map object, that defines many methods for data reduction (see the Appendix for reference). It also contains an attribute called 'Map', of class Image, where the results of a map-making routine are stored.

6.2.4 Storing a data object

At any time during a **BoA** session, the user can dump the content of the current data object to a file. It can later be loaded again into **BoA**, in order to continue with the data reduction. This is done with:

Table 6.3: Attributes of the ScanParam object

Name	Type	Description
ScanNum	int	Scan number
ScanType	string	Scan type, e.g. 'FOCUS—Z
ScanMode	string	Scan mode, e.g. 'RASTER'
ScanDir	string	Scanning direction
Line_Len	float	Line length for a raster, in arcsec
Line_Ysp	float	Y-step between lines in a raster, in arcsec
Az_Vel	float	Scanning speed in Az, in arcsec/s
Object	string	Target name
Basis	tuple	Pair of strings describing basis frame - e.g. ('RA— —SFL', 'DEC— —SFL')
Coord	tuple	Target coordinates in basis frame
Date_Obs	string	Date of observation
Equinox	float	Equinox
Nula, Nule	floats	X, Y pointing settings at scan start
Colstart	float	Focus-Z setting at scan start
DeltaCA, DeltaIE	floats	Accumulated pointing corrections CA and IE
NObs	int	Number of subscans
SubscanNum	int list	Subscans numbers
SubscanIndex	int array	Integration numbers at subscans starts and ends
SubscanEpo	float array	Epochs of subscans starts, in year
SubscanTime	float array	LST times of subscans starts, in s
SubscanType	string list	Types of subscans - e.g. 'ON', or 'REF'
WobUsed	int	Boolean: is a wobbler used?
WobCycle	float	Wobbler period, in s
WobblerPos	float array	Wobbler positions, in arcsec
WobThrow	float	Wobbler throw, in arcsec
WobblerSta	string list	Wobbler status
Nodding_Sta	int array	Nodding status
WobMode	string	Wobbler mode, e.g. 'SQUARE'
AddLonWT	int	Wobbler throw to be added in Az, in arcsec
AddLatWT	int	Wobbler throw to be added in El, in arcsec
OnOffPairs	int list	List of pairs of integration numbers (if wobbler)
Nint	int	Number of integrations
Baslon, Baslat	float arrays	Absolute coordinates in basis frame, in deg
Track_Az, Track_El	float arrays	Tracking errors in Az and El, in arcsec
Lon, Lat	float arrays	Offsets w.r.t. the source in Az and El, in deg
FocX, FocY, FocZ	float arrays	Subreflector positions in X, Y, Z, in mm
PhiX, PhiY	float arrays	Subreflector rotation angles in X and Y, in deg
Az, El	float arrays	Absolute coordinates in Az, El, in deg
Lonpole, Latpole	float array	Coordinates in user frame of basis pole
Rot	float array	Rotation angle between user and basis frames, in deg
MJD	float array	Timestamps in MJD, in days
UT	float array	Timestamps in UTC, in s
LST	float array	Timestamps in LST, in s
Flags	int array	Flagging in time domain (0 = unflagged)

Table 6.4: Other attributes of a DataEntity object

Name	Type	Description
FileName	string	Input file name
JyPerCount	float	Counts to Jy conversion factor
Data	float array	Current version of the data
DataWeights	float array	Relative weights of the datapoints
DataFlags	array	Flagging of individual datapoints (0 = unflagged)
CorMatrix	float array	Channel to channel correlation matrix
FFCF_Gain	float array	1D array of relative gains (flat field) derived from skynoise
FFCF_CN	float array	Channel to channel correlated skynoise
SkyNoise	float array	Skynoise present in the signal

Table 6.5: Other attributes of a DataAna object

Name	Type	Description
ChanMean	float array	Mean values of signal per channel
ChanRms	float array	R.M.S of signal per channel
ChanMed	float array	Median values of signal per channel
ChanMean_s	float array	Mean values of signal per channel and per subscan
ChanRms_s	float array	R.M.S. of signal per channel and per subscan
ChanMed_s	float array	Median values of signal per channel and per subscan
flagValue	int	Current default flag value when calling a flagging routine
flagValueList	int list	Allowed values for flagging


```
boa> dump()  
boa< I: current data successfully written to BoaData.sav
```

or:

```
boa> dump('myMap.data')  
boa< I: current data successfully written to myMap.data
```

to give another filename than the default `BoaData.sav`. Then to reload the data object, one has to do:

```
boa> dd = newRestoreData()
```

or:

```
boa> dd = newRestoreData('myMap.data')
```

Note: it is not possible in its present state to apply this restore method to the default *data* object. Therefore, after reloading a data object to a new variable (*dd* in the above example), one has to use the extended syntax (see Appendix) instead of the abbreviations defined in `BoaShortcut.py`.

6.3 Data output

Once a mapping observation has been read in and processed with **BoA**, the user can store the results, i.e. a map in sky coordinates, in FITS file with standard 2D FITS images, including header with World Coordinate System (WCS) informations. This is done with the following command:

```
boa> data.Map.writeFits()          # default file name: boaMap.fits  
boa> data.Map.writeFits('LABOCA_1234.fits') # give a file name
```

The resulting FITS file will contain three images, displaying the Intensity, the Weights and the Coverage of the current map. The content of each image is identified by the FITS keyword `EXTNAME`.

Part III

All BoAclasses and functions

A. BOA HIERARCHICAL INDEX

A.1 BoA Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

boa::BoaDataEntity::BolometerArray	86
boa::BoaDataEntity::DataEntity	106
boa::BoaDataAnalyser::DataAna	91
boa::BoaMapping::Map	124
boa::BoaPointing::Point	128
boa::BoaFocus::Focus	117
boa::BoaSkydip::Skydip	139
boa::Bogli::Interface::Fenetre	112
boa::BoaDataAnalyser::FilterFFT	114
boa::BoaMapping::Image	119
boa::BoaMapping::Kernel	123
boa::BoaDataEntity::ScanParameter	131
boa::BoaDataEntity::Telescope	141

B. BOA CLASS INDEX

B.1 BoA Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

boa::BoaDataEntity::BolometerArray	86
boa::BoaDataAnalyser::DataAna	91
boa::BoaDataEntity::DataEntity	106
boa::Bogli::Interface::Fenetre	112
boa::BoaDataAnalyser::FilterFFT	114
boa::BoaFocus::Focus	117
boa::BoaMapping::Image	119
boa::BoaMapping::Kernel	123
boa::BoaMapping::Map	124
boa::BoaPointing::Point	128
boa::BoaDataEntity::ScanParameter	131
boa::BoaSkydip::Skydip	139
boa::BoaDataEntity::Telescope (If <code>__tag__[12] == '\$Name: APECS':</code> <code>boa_version = __-</code> <code>tag__[7:-2]</code> else: <code>boa_version = 'Unknown'</code> print " version: %s" % <code>boa_version</code>)	141

C. BOA CLASS DOCUMENTATION

C.1 boa::BoaDataEntity::BolometerArray Class Reference

C.1.1 Detailed Description

NAM: BolometerArray (class)

DES: Define all the useful parameters of a bolometer array

Public Member Functions

- def [__init__](#)
- def [__str__](#)
- def [checkChanList](#)
- def [flag](#)
- def [flipOffsets](#)
- def [get](#)
- def [getChanIndex](#)
- def [getChanSep](#)
- def [plotArray](#)
- def [plotGain](#)
- def [printCurrChanList](#)
- def [readAdditionnalIndexFile](#)
- def [readAsciiRcp](#)
- def [readRCPfile](#)
- def [rotateArray](#)
- def [selectAdditionnalIndex](#)
- def [setCurrChanList](#)
- def [unflag](#)
- def [updateRCP](#)
- def [writeAsciiRcp](#)
- def [writeRCPfile](#)

C.1.2 Member Function Documentation

def boa::BoaDataEntity::BolometerArray::__init__ (self)

DES: Instanciation of a BolometerArray object

def boa::BoaDataEntity::BolometerArray::__str__ (self)

DES: Defines a string which is shown when the print instruction is used.

def boa::BoaDataEntity::BolometerArray::checkChanList (self, inList, flag = [], getFlagged = 0)

DES: Return a list of valid channels

INP: (int list/string) inList: list of channel numbers to get, or
empty list to get the complete list of unflagged channels, or
'all' or 'al' or 'a' to get the complete list of channels
(integer list) flag : retrieve data flagged or unflagged accordingly
(log) getFlagged : flag revers to flagged/unflagged data

flag	getFlagged	Retrieve..
'None'	0	all data
[]	0	unflagged data (default)
[]	1	data with at least one flag set
1	0	data with flag 1 not set
1	1	data with flag 1 set
[1,2]	0	data with neither flag 1 nor flag 2 set
[1,2]	1	data with either flag 1 or flag 2 set

OUT: (int list) list of channel numbers

def boa::BoaDataEntity::BolometerArray::flag (self, chanList = [], flag = 1)

DES: assign flags to a list of channels

INP: (integer array) chanList: list the channels to be flagged
(integer list) flag : flag values (default 1)

def boa::BoaDataEntity::BolometerArray::flipOffsets (self)

DES: flips the sign in Az/El of channel offsets. Used to convert (old) APEX-SZ scans into the same convention as for LABOCA

INP:

def boa::BoaDataEntity::BolometerArray::get (self, dataType, flag = [], getFlagged = 0)

DES: get bolometers offsets or gain according to flag

INP: (string) dataType : type of data

(integer list) flag : retrieve data flagged or unflagged accordingly

(log) getFlagged : flag revers to flagged/unflagged data

flag	getFlagged	Retrieve..
'None'	0	all data
[]	0	unflagged data (default)
[]	1	data with at least one flag set
1	0	data with flag 1 not set
1	1	data with flag 1 set
[1,2]	0	data with neither flag 1 nor flag 2 set
[1,2]	1	data with either flag 1 or flag 2 set

OUT: (float array) : the requested data

def boa::BoaDataEntity::BolometerArray::getChanIndex (self, chanList = [])

DES: convert from physical channel number to index in UsedChannel

INP: (i list) chanList : the physical channel number

OUT: (i list) the corresponding index (-1 if failed)

def boa::BoaDataEntity::BolometerArray::getChanSep (self, chanList = [])

DES: return the channel separation in both direction from the reference channel

def boa::BoaDataEntity::BolometerArray::plotArray (self, overplot = 0, num = 0, limitsX = [], limitsY = [], ci = 3)

DES: plot the receiver parameters

INP: (optional) overplot (logical) = overplot?

(optional) num (logical) = indicate chan numbers?

def boa::BoaDataEntity::BolometerArray::plotGain (self, style = 'idl4')

DES: plot the gain of the Array

INP: (str) style : the style to be used (default idl4)

WAR: the bolometer without know offsets should be flagged

def boa::BoaDataEntity::BolometerArray::printCurrChanList (self)

DES: print the current channel list in somehow "clever" way

OUT: a string representing the current channel list

def boa::BoaDataEntity::BolometerArray::readAdditionnalIndexFile (self, indexFile = 'match.dat', refColumn = 0, indexColumn = 1, comment = ' !')

DES: Read a list of additional index from an ASCII file, to be used with selectAdditionnalIndex

INP: indexFile : the name of the file to read the ...
 refColumn : the column of channel number and ... (default 0, the first column)
 indexColumn : the column to match the channel with (default 1, the second column)
 comment : comment character (default ' !')

def boa::BoaDataEntity::BolometerArray::readAsciiRcp (self, filename = 'boa.rcp')

DES: update receiver channel offsets from a simple ascii file
 channelNumber AzOffset ElOffset Major(FWHM) Minor(FWHN) Tilt Gain
 with unit of arcsec and degree

INP: (string) filename: the filename to read in

def boa::BoaDataEntity::BolometerArray::readRCPfile (self, rcpFile)

NAM: readRCPfile (method)

DES: update Receiver Channel Parameters (attributes Offsets,
 Gain and ChannelSep) from the content of a file.
 Also read beam shape if available

INP: (string) rcpFile: complete name of file to read in

def boa::BoaDataEntity::BolometerArray::rotateArray (self, angle)

DES: rotate array offsets by a given angle

INP: (float) angle (in degree)

def boa::BoaDataEntity::BolometerArray::selectAdditionnalIndex (self, value = None)

DES: Select according to the additionnal Index

INP: (s) value : the value to test

def boa::BoaDataEntity::BolometerArray::setCurrChanList (self, chanList = ' ?')

DES: set list of channels to be treated

INP: (int list/string) chanList = list of channels, or string '?'
 to get current list of channels, or string 'a' or 'al' or 'all'
 to set current list to all possible channels. Default: '?'

def boa::BoaDataEntity::BolometerArray::unflag (self, chanList = [], flag = [])

DES: unflags to a list of channels

INP: (integer array) chanList : list of channels to be unflagged (default all)
 (integer list) flag : flag values (default []: unset all flags)

def boa::BoaDataEntity::BolometerArray::updateRCP (self, rcpFile, scale = 1.)

NAM: updateRCP

DES: update only offsets and gains from the content of a file

INP: (string) rcpFile: complete name of file to read in
 (float) scale: scale factor to tune initial guess ASZCA rcp FB20070324

def boa::BoaDataEntity::BolometerArray::writeAsciiRcp (self, rcpFile = 'boa.rcp')

NAM: writeRCPfile (method)

DES: store current Receiver Channel Parameters (Offsets,
 Gain) to a file with mopsi like format

INP: (string) rcpFile: complete name of output file

def boa::BoaDataEntity::BolometerArray::writeRCPfile (self, rcpFile = 'rcpBoa.rcp')

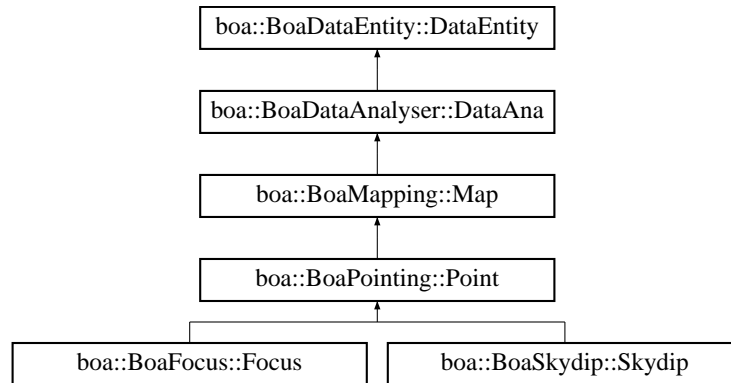
NAM: writeRCPfile (method)

DES: store current Receiver Channel Parameters (Offsets,
 Gains, Beam shape) to a file with mopsi like format

INP: (string) rcpFile: complete name of output file

C.2 boa::BoaDataAnalyser::DataAna Class Reference

Inheritance diagram for `boa::BoaDataAnalyser::DataAna`:



C.2.1 Detailed Description

DES: An object of this class is responsible for the flagging of individual channels, i.e. it sets the values in the `Channel_Flag` array of the corresponding `DataEntity` object. It provides methods to derive the rms of each channel and to automatically search for bad or noisy channels. Channels might be flagged according to a given input file. This object provides methods to derive the correlation matrix.

Public Member Functions

- def `__init__`
- def `bandRms`
- def `blankFreq`
- def `computeCorTimeShift`
- def `computeWeight`
- def `computeWeights`
- def `correctOpacity`
- def `correlatedNoiseRemoval`
- def `corrPCA`
- def `corrPCA_old`
- def `despike`
- def `flag`
- def `flagAccel`
- def `flagAutoRms`
- def `flagChannels`
- def `flagFractionRms`

-
- def [flagInTime](#)
 - def [flagLon](#)
 - def [flagMJD](#)
 - def [flagPosition](#)
 - def **flagRms**
 - def [flagSpeed](#)
 - def [flagSubscan](#)
 - def [flagTurnaround](#)
 - def [flatfield](#)
 - def [flattenFreq](#)
 - def [iterativeDespike](#)
 - def [medianBaseline](#)
 - def [medianFilter](#)
 - def [medianNoiseRemoval](#)
 - def [plotCorDist](#)
 - def [plotCorMatrix](#)
 - def [plotDataGram](#)
 - def [plotFFT](#)
 - def [plotMean](#)
 - def [plotMeanChan](#)
 - def [plotRms](#)
 - def [plotRmsChan](#)
 - def [polynomialBaseline](#)
 - def [read](#)
 - def [readFFCF](#)
 - def [rebin](#)
 - def [reduceFreq](#)
 - def [slidingRms](#)
 - def [taperFreq](#)
 - def [timeshiftAzEl](#)
 - def [timeShiftChan](#)
 - def [timeShiftChanList](#)
 - def [unflag](#)
 - def [unflagAccel](#)
 - def **unflagChannels**
 - def [unflagInTime](#)
 - def [unflagLon](#)
 - def [unflagMJD](#)
 - def [unflagPosition](#)
 - def [unflagSpeed](#)
 - def [unflagSubscan](#)
 - def [unflagTurnaround](#)
 - def [writeFFCF](#)
 - def [zeroStart](#)
-

C.2.2 Member Function Documentation

def boa::BoaDataAnalyser::DataAna::__init__ (self)

DES: initialise an instance

Reimplemented from [boa::BoaDataEntity::DataEntity](#).

Reimplemented in [boa::BoaFocus::Focus](#), [boa::BoaPointing::Point](#), [boa::BoaMapping::Map](#), and [boa::BoaSkydip::Skydip](#).

def boa::BoaDataAnalyser::DataAna::bandRms (self, chanList = [], low = 1., high = 10., channelFlag = [], getFlaggedChannels = 0, dataFlag = [], getFlaggedData = 0, windowSize = 0, windowing = 3)

DES: compute rms in some spectral range

INP: (i list) chanList : list of channels

(f) low, high : range limits (in Hz)

(int list) chan = channel list

(integer list) channelFlag : retrieve data from channels flagged or unflagged accordingly

(log) getFlaggedChannels : channelFlag revers to flagged/unflagged data

(integer list) dataFlag : retrieve data flagged or unflagged accordingly

(log) getFlaggedData : dataFlag revers to flagged/unflagged data

flag	getFlagged	Retrieve..
'None'	0	all data
[]	0	unflagged data (default)
[]	1	data with at least one flag set
1	0	data with flag 1 not set
1	1	data with flag 1 set
[1,2]	0	data with neither flag 1 nor flag 2 set
[1,2]	1	data with either flag 1 or flag 2 set

(f) windowSize : optional window size to compute FFTs

(i) windowing : function type for windowing (see applyWindow)

def boa::BoaDataAnalyser::DataAna::blankFreq (self, channel = 'all', below = '?', above = '?')

DES: Permanently remove some frequency interval in the Fourier spectrum of the signal. This is computed subscan by subscan.

INP: (int list) channel = list of channel to flagprocess (default: all)

(float) below = filter data below this value

(float) above = filter data above this value

def boa::BoaDataAnalyser::DataAna::computeCorTimeShift (self, shiftAz, shiftEl, chanList = [], refChan = -1, distlim = -1.)

DES: computes mean of absolute correlations for all channel pairs with mutual distance smaller than distlim, given time shifts in azimuth and elevation directions. To be used by timeshiftAzEl.

```

INP: (f)      shiftAz : time shift in azimuth. unit: milliseconds per arcsecond
(f)      shiftEl : time shift in elevation. unit: milliseconds per arcsecond
(i list) chanList : the list of channels to consider
(i)      refChan : reference channel (timeshift 0)
(f)      distlim : consider only correlations on bolometer separations
                smaller than this value (arcseconds)

```

def boa::BoaDataAnalyser::DataAna::computeWeight (self, method = 'rms')

DES: compute weights and store them in DataWeights attribute

INP: (str) method : type of weighting (default='rms', i.e. use $1/\text{rms}^2$)

def boa::BoaDataAnalyser::DataAna::computeWeights (self, chanList = [], minCorr = 0., a = 0.95, b = 2.0, core = 10., beta = 2.)

DES: compute correlation and weight matrix of the used channels

Weight is a non-linear rescaling of the correlation coefficient

```
weight_nm = ( CM_nm - a * min_m( CM_nm ) )**b
```

an additionnal weighting factor is applied with channel separation

```
weight_nm = weight_nm * 1.0 / ( 1 + ( dist_nm / core )**beta )
```

```

INP: (i list)      : chanList restrict the computation to certain channel (default : all used chan
(f) minCorr : minimum correlation coefficient (default:0, should be positiv)
(f) a       : parameter for weights, usually = 0.90-0.98
(f) b       : parameter for weights, usually = 1
(f) core    : core radius in arcmin for radial weighting (weight = 0.5)
(f) beta    : beta for beta profile for radial weighting

```

def boa::BoaDataAnalyser::DataAna::correctOpacity (self, tau = 0.)

DES: correct for atmospheric opacity

def boa::BoaDataAnalyser::DataAna::correlatedNoiseRemoval (self, chanList = [], threshold = 1.e-3, iterMax = 4, plot = 0, coreRadius = 30, chanRef = 17)

DES: remove the correlated noise from the data

```

INP: (i list) chanList : list of channel to flag (default: all; [] : current list)
(f)      threshold : threshold value for the Flat Field Correction Factor (in %, default 1.e-3)
(i)      iterMax  : maximum number of iteration
(i)      plot     : plot or not to plot (def 0)
(i)      coreRadius: core radius for weight taper beta profile
(i)      chanRef  : reference channel to start with

```

def boa::BoaDataAnalyser::DataAna::corrPCA (self, chanList = [], order = 1, subscan = 0)

DES: remove the correlated noise from the data
by principal component analysis, subscan by suscan
INP: (i list) chanList : list of channel to flag
(i) order : number of principal components to remove
(l) subscan : do the PCA subscan by subscan? default no
---negative value means choose the optimal number---not yet!!!!

def boa::BoaDataAnalyser::DataAna::corrPCA_old (self, chanList = [], order = 1)

DES: remove the correlated noise from the data
by principal component analysis
INP: (i list) chanList : list of channel to flag
(i) order : number of principal components to remove
---negative value means choose the optimal number---not yet!!!!

def boa::BoaDataAnalyser::DataAna::despike (self, chanList = [], below = -5, above = 5, flag = 1)

DES: Flag yet unflagged data below 'below'*rms and above 'above'*rms.
INP: (i list) chanList : list of channel to flag (default: current list)
(f) below : flag data with value < 'below'*rms
(f) above : flag data with value > 'above'*rms
(i list) flag : flag values (default: 1 'SPIKE')

def boa::BoaDataAnalyser::DataAna::flag (self, dataType = "", channel = 'all', below = '??', above = '??', flag = 8)

DES: flag data based on dataType, general flagging routine, may be slow
INP: (s) dataType : flag based on this dataType
(i list) channel : list of channel to flag (default: all)
(f) below : flag dataType < below (default max; or 5*RMS)
(f) above : flag dataType > above (default min; or -5*RMS)
(i) flag : flag value (default 8 'TEMPORARY')

below and above should be in unit of the flagged data,
except for 'Lon' and 'Lat' where they should be in arcsec

def boa::BoaDataAnalyser::DataAna::flagAccel (self, channel = 'all', below = '??', above = '??', flag = 2)

DES: Flag data according to telescope acceleration
INP: (float) below = flag data below this value
(float) above = flag data above this value
(int) flag = flag to be set (default 2 'ACCELERATION THRESHOLD')

```
def boa::BoaDataAnalyser::DataAna::flagFractionRms ( self, chanList = [], ratio = 10., flag
= 2, plot = 0)
```

DES: flag according to rms, with limits depending on median rms

INP: (i list) chanList : list of channel to flag (default: current list)
(f) ratio : channels with rms below median/ratio and above
median*ratio will be flagged
(i) flag : value of flag to set (default: 2 'BAD SENSITIVITY')
(b) plot : plot the results

```
def boa::BoaDataAnalyser::DataAna::flagInTime ( self, dataType = 'LST', below = '?',  
above = '?', flag = 8)
```

DES: Flag data in time interval

INP: (float) below = flag data below this value (default end of the scan)
(float) above = flag data above this value (default start of the scan)
(int) flag = flag to be set (default: 8 'TEMPORARY')

```
def boa::BoaDataAnalyser::DataAna::flagLon ( self, channel = 'all', below = '?', above =  
'?', flag = 8)
```

NAM: flagLon (method)

DES: Flag data in Longitude interval

INP: (int list) channel = list of channel to flag (default: all)
(float) below = flag data below this value
(float) above = flag data above this value
(int) flag = flag to be set (default 8 'TEMPORARY')

```
def boa::BoaDataAnalyser::DataAna::flagMJD ( self, below = '?', above = '?', flag = 8)
```

DES: Flag data in time interval

INP: (float) below = flag data below this value (default end of the scan)
(float) above = flag data above this value (default start of the scan)
(int) flag = flag to be set (default: 8 'TEMPORARY')

```
def boa::BoaDataAnalyser::DataAna::flagPosition ( self, channel = 'all', Az = 0, El = 0,  
radius = 0, flag = 8, offset = 1)
```

DES: flag a position in the sky within a given radius

INP: (int list) channel : list of channel to flag (default: 'all')
(float) Az/El : the horizontal reference position (arcsec for offsets, deg for absolute)
(float) radius : aperture to flag in unit of the reference position
(int) flag : flag to be set (default 8 'TEMPORARY')
(logical) offset : flag on the offsets (default yes,)

def boa::BoaDataAnalyser::DataAna::flagSpeed (self, below = ' ?', above = ' ?', flag = 3)

DES: Flag data according to telescope speed
 INP: (float) below = flag data below this value
 (float) above = flag data above this value
 (int) flag = flag to be set (default 3 'ELEVATION VELOCITY THRESHOLD')

def boa::BoaDataAnalyser::DataAna::flagSubscan (self, subList, flag = 7)

DES: flag subscans
 INP: (int list) subList = list of subscan numbers (or single number)
 to be flagged
 (int) flag = value of flags to set (default: 7 'SUBSCAN FLAGGED')

def boa::BoaDataAnalyser::DataAna::flagTurnaround (self, flag = 1)

NAM: flagTurnaround (method)
 DES: flag subscans where azimuth offset changes sign
 INP: (int) flag = flag to be set (default 1 'TURNAROUND')

def boa::BoaDataAnalyser::DataAna::flatfield (self, chanList = [], method = 'point')

DES: divide signals by bolo gains to normalise them
 INP: (i list) channel: list of channels to process (default: [] = current list)
 (str) method : choose which flat field to apply:
 - point [default] = use point source relative gains
 - median = use correlated noise relative gains
 - extend = use relative gains to extended emission

def boa::BoaDataAnalyser::DataAna::flattenFreq (self, channel = 'all', below = 0.1, hiref = 1., optimize = 1, window = 4)

DES: flatten the 1/F part of the FFT using constant amplitude
 INP: (int list) channel = list of channels to process (default: all)
 (float) below = filter data below this value
 (float) hiref = amplitudes at $f < \text{below}$ will be replaced with
 the average value between below and hiref

def boa::BoaDataAnalyser::DataAna::iterativeDespike (self, chanList = [], below = -5, above = 5, flag = 1, maxIter = 100)

DES: Iteratively flag yet unflagged data below 'below'*rms and above 'above'*rms.
 INP: (i list) chanList : list of channel to flag (default: current list)
 (f) below : flag data with value < 'below'*rms
 (f) above : flag data with value > 'above'*rms
 (i) maxIter : maximum number of iteration (default 100)
 (i list) flag : flag values (default: 1 'SPIKE')

def boa::BoaDataAnalyser::DataAna::medianBaseline (self, chanList = [], subscan = 1, order = 0)

DES: baseline: Remove median value per channel and per subscan
 INP: (i list) channel : list of channels to process (default: [] = current list)
 (1) subscan : compute baseline per subscan (default: yes)
 (i) order : polynomial order (default: 0)

def boa::BoaDataAnalyser::DataAna::medianFilter (self, chanList = [], window = 20, subtract = 1)

DES: median filtering: remove median values computed over sliding window
 INP: (i list) channel: list of channels to process (default: [] = current list)
 (i) window: number of samples to compute median
 (1) subtract: subtract from data? (default: yes)

def boa::BoaDataAnalyser::DataAna::medianNoiseRemoval (self, chanList = [], chanRef = 0, computeFF = 1, factor = 1., nbloop = 1)

DES: remove median noise from the data
 INP: (i list) chanList : list of channels (default: [] = current list)
 (int) chanRef : reference channel number (default: RefChannel;
 -1 = compute relative gains w.r.t. mean signal
 -2 = compute relative gains w.r.t. median signal
 (log) computeFF : compute skynoise FF (def.) or use existing FF_Median?
 (float) factor : fraction of skynoise to be subtracted (default: 100%)
 (int) nbloop : number of iterations (default: 1)

def boa::BoaDataAnalyser::DataAna::plotCorDist (self, chanList = [], average = 1, upperlim = -1., check = 1, style = 'p', ci = 1, overplot = 0, limitsX = [], limitsY = [], pointsiz = 3., plot = 1)

DES: plot correlations (correlation matrix) as a function
 of channel separation
 INP: (i list) chanList : the list of channels to plot
 (i) average : number of data to average over (for easier viewing)
 (f) upperlim : return only distances in arcsec below this value
 (negative value means no limit, which is the default)
 (1) check : check the chanList first (default: yes)
 (1) plot : actually produce a plot? (default: yes)

def boa::BoaDataAnalyser::DataAna::plotCorMatrix (self, chanList = [], check = 1, distance = 0, weights = 0, xLabel = 'Channels', style = 'idl4', limitsZ = [])

DES: plot the correlation matrix
 INP: (i list) chanList : the list of channel to plot

```
(1)      check : check the chanList first ( default : yes)
(1)      distance : sort the second dimension by distance (default : no)
(1)      weights : plot weights instead of correlation matrix (default: no)
```

**def boa::BoaDataAnalyser::DataAna::plotDataGram (self, chanNum = -1, flag = [],
plotFlagged = 0, n = 512, limitsZ = [])**

DES: plot FFT of signal

INP: (i) chanNum : channel number to plot

(integer list) flag : plot data flagged or unflagged accordingly

(log) plotFlagged : flag revers to flagged/unflagged data

flag	plotFlagged	Plot..
'None'	0	all data
[]	0	unflagged data (default)
[]	1	data with at least one flag set
1	0	data with flag 1 not set
1	1	data with flag 1 set
[1,2]	0	data with neither flag 1 nor flag 2 set
[1,2]	1	data with either flag 1 or flag 2 set

(i) n : Number of points for the ffts

(2f) limitsZ : limits for the color scale

**def boa::BoaDataAnalyser::DataAna::plotFFT (self, chanList = [], channelFlag = [],
plotFlaggedChannels = 0, dataFlag = [], plotFlaggedData = 0, limitsX = [], limitsY = [],
style = 'l', ci = 1, overplot = 0, logX = 1, logY = 1, windowSize = 0, windowing = 3)**

DES: plot FFT of signal

INP: (i list) chanList : list of channels

(integer list) channelFlag : plot data from channels flagged or unflagged accordingly

(log) plotFlaggedChannels : channelFlag revers to flagged/unflagged data

(integer list) dataFlag : plot data flagged or unflagged accordingly

(log) plotFlaggedData : dataFlag revers to flagged/unflagged data

flag	plotFlagged	Plot..
'None'	0	all data
[]	0	unflagged data (default)
[]	1	data with at least one flag set
1	0	data with flag 1 not set
1	1	data with flag 1 set
[1,2]	0	data with neither flag 1 nor flag 2 set
[1,2]	1	data with either flag 1 or flag 2 set

limits, style, ci...: plot parameters (see MultiPlot.plot)

**def boa::BoaDataAnalyser::DataAna::plotMean (self, chanList = [], channelFlag = [],
plotFlaggedChannels = 0, dataFlag = [], plotFlaggedData = 0, limitsX = [], limitsY = [],
style = 'l', ci = 1, overplot = 0, map = 0)**

DES: plot mean flux value vs. subscan number

TODO: flag handling not implemented yet

INP: (int list) chanList = list of channels

```

(integer list) channelFlag : plot data from channels flagged or unflagged accordingly
(log)   plotFlaggedChannels : channelFlag revers to flagged/unflagged data
(integer list)   dataFlag : plot data flagged or unflagged accordingly
(log)   plotFlaggedData : dataFlag revers to flagged/unflagged data
      flag | plotFlagged | Plot..
      'None' | 0         | all data
      []      | 0         | unflagged data (default)
      []      | 1         | data with at least one flag set
      1       | 0         | data with flag 1 not set
      1       | 1         | data with flag 1 set
      [1,2]   | 0         | data with neither flag 1 nor flag 2 set
      [1,2]   | 1         | data with either flag 1 or flag 2 set
(logical)      map = plot as a 2D map?

```

**def boa::BoaDataAnalyser::DataAna::plotMeanChan (self, chanList = [], channelFlag = [],
plotFlaggedChannels = 0, dataFlag = [], plotFlaggedData = 0, limitsX = [], limitsY = [],
style = 'p', ci = 1, overplot = 0)**

DES: PLOtting the MEAN value for each subscan against channel number.

**def boa::BoaDataAnalyser::DataAna::plotRms (self, chanList = [], channelFlag = [],
plotFlaggedChannels = 0, dataFlag = [], plotFlaggedData = 0, limitsX = [], limitsY = [],
style = 'l', ci = 1, overplot = 0, map = 0)**

DES: plot flux r.m.s. vs. subscan number

TODO: flag handling not implemented yet

INP: (int list) chanList = list of channels

```

(integer list) channelFlag : plot data from channels flagged or unflagged accordingly
(log)   plotFlaggedChannels : channelFlag revers to flagged/unflagged data
(integer list)   dataFlag : plot data flagged or unflagged accordingly
(log)   plotFlaggedData : dataFlag revers to flagged/unflagged data
      flag | plotFlagged | Plot..
      'None' | 0         | all data
      []      | 0         | unflagged data (default)
      []      | 1         | data with at least one flag set
      1       | 0         | data with flag 1 not set
      1       | 1         | data with flag 1 set
      [1,2]   | 0         | data with neither flag 1 nor flag 2 set
      [1,2]   | 1         | data with either flag 1 or flag 2 set
(logical)      map = plot as a 2D map?

```

**def boa::BoaDataAnalyser::DataAna::plotRmsChan (self, chanList = [], channelFlag = [],
plotFlaggedChannels = 0, dataFlag = [], plotFlaggedData = 0, limitsX = [], limitsY = [],
style = 'p', ci = 1, overplot = 0, subscan = 0, logY = 0)**

DES: PLOtting the RMS value for each subscan against channel number.

INP: (logical) subscan: if 0, plot rms of the complete scan, if 1,

plot for each subscan and each channel

(integer list) channelFlag : plot data from channels flagged or unflagged accordingly

```

(log) plotFlaggedChannels : channelFlag revers to flagged/unflagged data
(integer list) dataFlag : plot data flagged or unflagged accordingly
(log) plotFlaggedData : dataFlag revers to flagged/unflagged data
                        flag | plotFlagged | Plot..
                        'None' | 0 | all data
                        [] | 0 | unflagged data (default)
                        [] | 1 | data with at least one flag set
                        1 | 0 | data with flag 1 not set
                        1 | 1 | data with flag 1 set
                        [1,2] | 0 | data with neither flag 1 nor flag 2 set
                        [1,2] | 1 | data with either flag 1 or flag 2 set

```

def boa::BoaDataAnalyser::DataAna::polynomialBaseline (self, chanList = [], order = 0, subscan = 1, plot = 0, subtract = 1)

DES: polynomial baseline removal on the Data.

INP: (i list) channel : list of channel to flag (default: all; [] : current list)
 (i) order : polynomial order, >0
 (l) subscan : compute baseline per subscan (default: yes)
 (l) plot : plot the signal and the fitted polynomials (default: no)
 (l) subtract : subtract the polynomial from the data (default: yes)

def boa::BoaDataAnalyser::DataAna::read (self, inFile = "", febe = "", baseband = 0, subscans = [], update = 0, phase = 0, channelFlag = 1, integrationFlag = 9, blanking = 1, readHe = 0, readAzEl0 = 0, readT = 0, readWind = 0, readBias = 0)

DES: fill a BoA data object from an MB-FITS file

INP: (string) inFile : path to the dataset to be read
 (string) febe : FE-BE name to select
 (int) baseband : baseband to select
 (int list) subscans : subscan numbers to read (default: all)
 (logical) update : if true, do not reset previous entity object
 (int) phase : phase to be stored (default: phase diff)
 (log) blanking : automatic flagging of blanked data (default: yes)
 channelFlag (i list) : flag for not connected feeds (default: 1 'NOT CONNECTED')
 integrationFlag (i list) : flag for blanked integrations (default: 9 'BLANK DATA')
 (log) readHe : do we need the He3 temperatures? (default: no)
 (log) readAzEl0 : do we read monitor Az,El at start? (default: no)
 (logical) readT : do we read T_amb from monitor? (def: no)
 (logical) readWind : do we read wind speed, dir... (def: no)
 (logical) readBias : do we need ASZCa bias settings? (def: no)
 OUT: (int) status : 0 if reading ok, <> 0 if an error occurred
 Possible error codes are:
 -1 = file could not be opened
 -2 = something wrong with FEBE
 -3 = something wrong with basebands
 -4 = something wrong with subscans

def boa::BoaDataAnalyser::DataAna::readFFCF (self, inFile = 'ffcf.txt')

NAM: readFFCF (method)

DES:

INP: (string) inFile: complete name of file to read in

def boa::BoaDataAnalyser::DataAna::rebin (self)

DES: average integrations 2 by 2

**def boa::BoaDataAnalyser::DataAna::reduceFreq (self, channel = 'all', center = 50.,
width = 1., factor = 10., optimize = 1, window = 4)**

DES: Permanently reduce some frequency interval in the Fourier spectrum of the signal. This is computed subscan by subscan.

INP: (int list) channel = list of channel to process (default: all)

(f) center = central frequency, in Hz

(f) width = line FWHM

(f) factor = attenuation factor

**def boa::BoaDataAnalyser::DataAna::slidingRms (self, nbInteg = 10, channel = [], flag = [],
getFlagged = 0)**

NAM: slidingRms (method)

DES: compute rms in a sliding window

INP: (int) nbInteg : number of elements on which one rms is computed (= window size)

(i list) channel : list of channel to flag (default: all; [] : current list)

(integer list) flag : retrieve data flagged or unflagged accordingly

(log) getFlagged : flag revers to flagged/unflagged data

flag	getFlagged	Retrieve..
'None'	0	all data
[]	0	unflagged data (default)
[]	1	data with at least one flag set
1	0	data with flag 1 not set
1	1	data with flag 1 set
[1,2]	0	data with neither flag 1 nor flag 2 set
[1,2]	1	data with either flag 1 or flag 2 set

OUT: (array) the rms are returned

**def boa::BoaDataAnalyser::DataAna::taperFreq (self, channel = 'all', above = '?', N = 2,
window = 4)**

DES: Permanently taper off Fourier spectrum above given value of the signal

INP: (int list) channel = list of channel to flagprocess (default: all)

(float) above = filter data above this value

(int) N = Butterworth steepenes order

def boa::BoaDataAnalyser::DataAna::timeshiftAzEl (self, chanList = [], refChan = -1, check = 1, distlim = 300., shiftmax = 10.)

DES: computes time shifts of all channels, with respect to a reference channel, which MAXIMIZES the correlated noise across the array

INP: (i list) chanList : the list of channels to consider
 (i) refChan : reference channel (will get timeshift=0)
 (l) check : check the chanList first (default: yes)
 (f) distlim : consider only correlations on bolometer separations smaller than this value (arcseconds)
 (f) shiftmax : maximum timeshift (absolute value) in milliseconds per arcsecond

def boa::BoaDataAnalyser::DataAna::timeShiftChan (self, chan, step)

DES: time shift channel by step

INP: (i) chan : channel number
 (i) step : number of time stamps

def boa::BoaDataAnalyser::DataAna::timeShiftChanList (self, chanList, steps)

DES: time shift list of channels by list of steps

INP: (i list) chan : channel list
 (i list) steps : list of number of time stamps

def boa::BoaDataAnalyser::DataAna::unflag (self, channel = [], flag = [])

NAM: unflag (method)

DES: Unflag data, i.e. reset to 0.

INP: (i list) channel : list of channel to flag (default: current list)
 (i) flag : unflag only this value (default []: all non-reserved flag values)

def boa::BoaDataAnalyser::DataAna::unflagAccel (self, channel = 'all', below = '??', above = '??', flag = [])

DES: Unflag data according to telescope acceleration

INP: (float) below = unflag data below this value
 (float) above = unflag data above this value
 (int) flag = flag to be unset (default []: all flag values)

def boa::BoaDataAnalyser::DataAna::unflagInTime (self, dataType = 'LST', below = '?', above = '?', flag = [])

DES: Unflag data in time interval

INP: (float) below = unflag data below this value (default end of the scan)
 (float) above = unflag data above this value (default start of the scan)
 (int) flag = flag to be unset (default []: all flag values)

def boa::BoaDataAnalyser::DataAna::unflagLon (self, channel = 'all', below = '?', above = '?', flag = [])

NAM: unflagLon (method)

DES: Unflag data in Longitude interval

INP: (int list) channel = list of channel to flag (default: all)
 (float) below = flag data below this value
 (float) above = flag data above this value
 (int) flag = flag to be unset (default []: all non-reserved flag values)

def boa::BoaDataAnalyser::DataAna::unflagMJD (self, below = '?', above = '?', flag = [])

DES: Unflag data in time interval

INP: (float) below = flag data below this value (default end of the scan)
 (float) above = flag data above this value (default start of the scan)
 (int) flag = flag to be unset (default []: all flag values)

def boa::BoaDataAnalyser::DataAna::unflagPosition (self, channel = 'all', Az = 0, El = 0, radius = 0, flag = [], offset = 1)

DES: unflag a position in the sky within a given radius

INP: (int list) channel : list of channel to unflag (default: 'all')
 (float) Az/El : the horizontal reference position (arcsec for offsets, deg for absolute)
 (float) radius : aperture to unflag in unit of the reference position
 (int) flag : unflag to be set (default []: unflag all non-reserved flag values)
 (logical) offset : unflag on the offsets (default yes,)

def boa::BoaDataAnalyser::DataAna::unflagSpeed (self, below = '?', above = '?', flag = [])

DES: Unflag data according to telescope speed

INP: (float) below = unflag data below this value
 (float) above = unflag data above this value
 (int) flag = flag to be unset (default []: all flag values)

def boa::BoaDataAnalyser::DataAna::unflagSubscan (self, subList, flag = [])

DES: unflag subscans

INP: (int list) subList = list of subscan numbers (or single number)
 to be unflagged

(int) flag = value of flags to unset (default []: all flag values)

def boa::BoaDataAnalyser::DataAna::unflagTurnaround (self, flag = [])

NAM: unflagTurnaround (method)

DES: unflag subscans where azimuth offset changes sign

INP: (int) flag = flag to be unset (default []: all flag values)

def boa::BoaDataAnalyser::DataAna::writeFFCF (self, outFile = 'ffcf.txt')

NAM: writeFFCF (method)

DES: store current correlated noise flat field to a file

INP: (string) file: complete name of output file

def boa::BoaDataAnalyser::DataAna::zeroStart (self, chanList = [], subscan = 0)

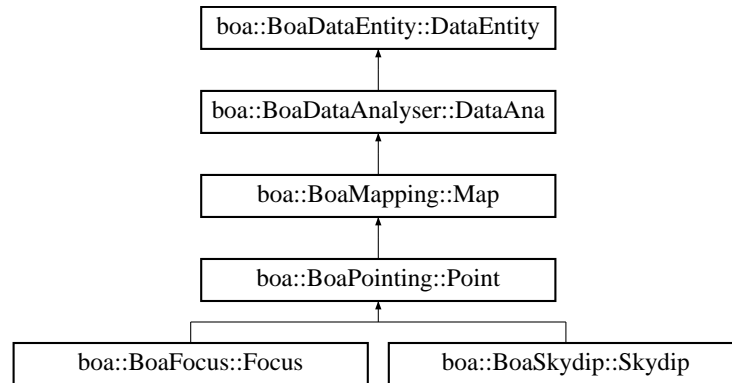
DES: make signal start at zero

INP: (i list) channel : list of channels to process (default: [] = current list)

(l) subscan : compute baseline per subscan (default: no)

C.3 boa::BoaDataEntity::DataEntity Class Reference

Inheritance diagram for boa::BoaDataEntity::DataEntity::



C.3.1 Detailed Description

NAM: DataEntity (class)

DES: Objects of this class store the data and associated parameters of a scan, which can contain several observations (or subscans). They also contain additional arrays in which the current results of the data reduction are stored. This class also provides the interface between the MB-FITS files and BoA, by the means of the fillFromMBFits() method.

Public Member Functions

- def `__add__`
- def `__init__`
- def `__str__`
- def `backup`
- def `dumpData`
- def `getChanData`
- def `getChanListData`
- def `loadExchange`
- def `plotCorrel`
- def `read`
- def `reset`
- def `restore`
- def `restoreData`
- def `saveExchange`
- def `saveMambo`
- def `selectPhase`

- def [signal](#)
- def [signalHist](#)

Public Attributes

- **CorrelatedNoise**
- **Data**
- **DataBackup**
- **DataFlags**
- **DataWeights**

C.3.2 Member Function Documentation

def boa::BoaDataEntity::DataEntity::__init__ (self)

DES: Instanciation of a new DataEntity object.
All attributes are defined and set to default values.

Reimplemented in [boa::BoaDataAnalyser::DataAna](#), [boa::BoaFocus::Focus](#), [boa::BoaPointing::Point](#), [boa::BoaMapping::Map](#), and [boa::BoaSkydip::Skydip](#).

def boa::BoaDataEntity::DataEntity::__str__ (self)

DES: Defines a string which is shown when the print instruction is used. It contains the sizes and typecodes of all attributes.

def boa::BoaDataEntity::DataEntity::backup (self)

DES: backup the data

def boa::BoaDataEntity::DataEntity::dumpData (self, fileName = 'BoaData.sav')

DES: save the current DataEntity object to a file
INP: (string) fileName: name of the output file
optional - default value = 'BoaData.sav'

def boa::BoaDataEntity::DataEntity::getChanData (self, dataType = ' ', chan = 'None', flag = [], getFlagged = 0, flag2 = None, subscans = [])

DES: get data for one channel
INP: (string) dataType : type of data
(int) chan : channel number

```

(integer list) flag : retrieve data flagged or unflagged accordingly
(log)      getFlagged : flag revers to flagged/unflagged data
            flag      | getFlagged | Retrieve..
            'None'    | 0          | all data
            []        | 0          | unflagged data (default)
            []        | 1          | data with at least one flag set
            1         | 0          | data with flag 1 not set
            1         | 1          | data with flag 1 set
            [1,2]     | 0          | data with neither flag 1 nor flag 2 set
            [1,2]     | 1          | data with either flag 1 or flag 2 set
(int list) subscans : list of wanted subscan (default all)
OPT: (int array)  flag2 : second array of flags to check
OUT: (float)      array : data of one channel

```

**def boa::BoaDataEntity::DataEntity::getChanListData (self, type = ' ', chanList = [],
channelFlag = [], getFlaggedChannels = 0, dataFlag = [], getFlaggedData = 0, dataFlag2 =
None, subscans = [])**

```

DES: get data for list of channels
INP: (string) type = type of data
      (int list) chan = channel list
      (integer list) channelFlag : retrieve data from channels flagged or unflagged accordingly
      (log)      getFlaggedChannels : channelFlag revers to flagged/unflagged data
      (integer list)      dataFlag : retrieve data flagged or unflagged accordingly
      (log)      getFlaggedData : dataFlag revers to flagged/unflagged data
            flag      | getFlagged | Retrieve..
            'None'    | 0          | all data
            []        | 0          | unflagged data (default)
            []        | 1          | data with at least one flag set
            1         | 0          | data with flag 1 not set
            1         | 1          | data with flag 1 set
            [1,2]     | 0          | data with neither flag 1 nor flag 2 set
            [1,2]     | 1          | data with either flag 1 or flag 2 set
      (int array) dataFlag2 = second array of flags to check (optional)
OUT: (list of float arrays) = data of the input list of channels

```

def boa::BoaDataEntity::DataEntity::loadExchange (self, fileName = "")

```

DES: read information from a Fits file for exchange with other
      reduction packages into the DataEntity object
INP: (str) fileName: name of the Fits file

```

**def boa::BoaDataEntity::DataEntity::plotCorrel (self, chanRef = -1, chanList = [],
channelFlag = [], plotFlaggedChannels = 0, dataFlag = [], plotFlaggedData = 0, skynoise =
0, limitsX = [], limitsY = [], style = 'p', ci = 1, overplot = 0)**

```

DES: plot flux density of a list of channels vs. flux density of a
      reference channel
INP: (int)      chanRef = reference channel number (default : is the first in chanList)

```

```

(int list) chanList = list of channels
(integer list) channelFlag : plot data from channels flagged or unflagged accordingly
(log) plotFlaggedChannels : channelFlag revers to flagged/unflagged data
(integer list) dataFlag : plot data flagged or unflagged accordingly
(log) plotFlaggedData : dataFlag revers to flagged/unflagged data
      flag | plotFlagged | Plot..
      'None' | 0 | all data
      [] | 0 | unflagged data (default)
      [] | 1 | data with at least one flag set
      1 | 0 | data with flag 1 not set
      1 | 1 | data with flag 1 set
      [1,2] | 0 | data with neither flag 1 nor flag 2 set
      [1,2] | 1 | data with either flag 1 or flag 2 set
(1) skynoise = plot against the skynoise of chanRef (default : no)

```

def boa::BoaDataEntity::DataEntity::read (self, inFile = "", febe = "", baseband = 0, subscans = [], update = 0, phase = 0, channelFlag = 1, integrationFlag = 9, readHe = 0, readAzEl0 = 0, readT = 0, readWind = 0, readBias = 0)

DES: fill a data entity object
 INP: (string) inFile: path to the dataset to be read
 (int list) subscans : subscan numbers to read (default: all)
 (logical) update : if true, do not reset previous entity object
 (int) phase : phase to be stored (default: phase diff)
 channelFlag (i list) : flag for not connected feeds (default: 1 'NOT CONNECTED')
 integrationFlag (i list) : flag for blanked integrations (default: 9 'BLANK DATA')
 (logical) readHe : do we read LABOCA He3 tempe? (def: no)
 (logical) readAzEl0 : do we read monitor Az, El(0)? (def: no)
 (logical) readT : do we read T_amb from monitor? (def: no)
 (logical) readWind : do we read wind speed, dir...? (def: no)
 (logical) readBias : do we need ASZCa bias settings? (def: no)
 OUT: (int) status : 0 if reading ok, <> 0 if an error occurred
 (see BoaDataAnalyser.read for error codes description)

def boa::BoaDataEntity::DataEntity::reset (self)

DES: Reset all attributes - useful before reading a new file

def boa::BoaDataEntity::DataEntity::restore (self)

DES: backup the data

def boa::BoaDataEntity::DataEntity::restoreData (self, fileName = 'BoaData.sav')

DES: restore a DataEntity object previously saved in a file, and set it as the currData attribute of Boab
 INP: (string) fileName: name of the input file
 optional - default value = 'BoaData.sav'

def boa::BoaDataEntity::DataEntity::saveExchange (self, fileName = "", overwrite = 0)

DES: save information from the DataEntity object to a
Fits file for exchange with other reduction packages
INP: (str) fileName: name of the Fits file (optional)
(log) overwrite: Overwrite existing file (optional)

def boa::BoaDataEntity::DataEntity::saveMambo (self, inName = "", outName = "")

DES: convert an MB-Fits file to the MAMBO FITS format, readable
by MOPSIC
INP: (str) inName: name of the MB-Fits file (optional)
(str) outName: name of the MAMBO output file (optional)

def boa::BoaDataEntity::DataEntity::selectPhase (self, phase)

NAM: selectPhase (method)
DES: Keep only Data(ON) or Data(OFF)
INP: (int) phase: phase to keep, 1=ON, 2=OFF

**def boa::BoaDataEntity::DataEntity::signal (self, chanList = [], channelFlag = [],
plotFlaggedChannels = 0, dataFlag = [], plotFlaggedData = 0, limitsX = [], limitsY = [],
style = 'l', ci = 1, overplot = 0, plotMap = 0, skynoise = 0, caption = "")**

DES: plot time series of flux density
INP: (int list) chanList = list of channels
(integer list) channelFlag : plot data from channels flagged or unflagged accordingly
(log) plotFlaggedChannels : channelFlag revers to flagged/unflagged data
(integer list) dataFlag : plot data flagged or unflagged accordingly
(log) plotFlaggedData : dataFlag revers to flagged/unflagged data

flag	plotFlagged	Plot..
'None'	0	all data
[]	0	unflagged data (default)
[]	1	data with at least one flag set
1	0	data with flag 1 not set
1	1	data with flag 1 set
[1,2]	0	data with neither flag 1 nor flag 2 set
[1,2]	1	data with either flag 1 or flag 2 set

(logical) skynoise = plot correlated noise (default 0)
(str) caption = plot title, default = scan info

**def boa::BoaDataEntity::DataEntity::signalHist (self, chanList = [], channelFlag = [],
plotFlaggedChannels = 0, dataFlag = [], plotFlaggedData = 0, limitsX = [], limitsY = [], ci =
1, overplot = 0, caption = "", nbin = 60, fitGauss = 0, subtractGauss = 0, logY = 0)**

DES: plot histogram of flux density time series

```

INP: (int list) chanList = list of channels
      (integer list) channelFlag : plot data from channels flagged or unflagged accordingly
      (log) plotFlaggedChannels : channelFlag revers to flagged/unflagged data
      (integer list) dataFlag : plot data flagged or unflagged accordingly
      (log) plotFlaggedData : dataFlag revers to flagged/unflagged data
          flag | plotFlagged | Plot..
          'None' | 0 | all data
          [] | 0 | unflagged data (default)
          [] | 1 | data with at least one flag set
          1 | 0 | data with flag 1 not set
          1 | 1 | data with flag 1 set
          [1,2] | 0 | data with neither flag 1 nor flag 2 set
          [1,2] | 1 | data with either flag 1 or flag 2 set
(int) nbin = number of bins in histogram
(l) fitGauss = fit a gaussian to the data?
(l) subtractGauss = subtract gaussian from the data?
(str) caption = plot title, default = scan info

```

C.4 boa::Bogli::Interface::Fenetre Class Reference

C.4.1 Detailed Description

```
classe Fenetre - parametres et methodes pour les boites et boutons
attributs:
int forme      : 0=cercle 1=rectangle 2=rectangle transparent
list pos       : positions (X,Y) des centres dans la fenetre
list float/tuple size: rayon ou (largeur,hauteur)
list label     : messages a apparaitre (vecteur de string) dans ou pres d'une fenetre
tuple txtpos   : position des labels relativement a pos
int font       : taille des caracteres
int coltxt     : couleur du texte
int colfond    : couleur de fond des boutons
int family     : police de caracteres
```

Public Member Functions

- def `__init__`
- def `dessine`
- def `saisie`

Public Attributes

- `colfond`
- `coltxt`
- `family`
- `font`
- `forme`
- `label`
- `pos`
- `size`
- `txtpos`

C.4.2 Member Function Documentation

```
def boa::Bogli::Interface::Fenetre::dessine ( self, new = 0)
```

method `dessine`

INP: `new` : efface la fenetre si non nul
OUT: aucune

def boa::Bogli::Interface::Fenetre::saisie (self)

method saisie

INP: aucune

OUT: choix : selection (numero du bouton)

C.5 boa::BoaDataAnalyser::FilterFFT Class Reference

C.5.1 Detailed Description

DES: To easily do FFT filtering

INF: make the assumption that the input signal is real, so do not care about negative frequencies...

Public Member Functions

- def `__init__`
- def `blankAmplitude`
- def `doDataGram`
- def `doFFT`
- def `invFFT`
- def `plotDataGram`
- def `plotFFT`
- def `reduceAmplitude`
- def `taperAmplitude`

Public Attributes

- **Amplitude**
- **DataGram**
- **Freq**
- **N**
- **OutX**
- **OutY**
- **Phase**
- **Power**
- **SamplFreq**
- **Timing**
- **X**
- **Y**

C.5.2 Member Function Documentation

def `boa::BoaDataAnalyser::FilterFFT::blankAmplitude (self, below = ' ?', above = ' ? ')`

DES: blank the amplitude below and/or after a certain frequency

```
def boa::BoaDataAnalyser::FilterFFT::doDataGram ( self, interpolate = 0, n = 1024, window = 4)
```

DES : Compute the Datagram of the data

```
INP : (l) interpolate : force an interpolation to be done
      (default no : check of timing quality better then 0.1 %)
      (i) n             : Number of points for the ffts
```

```
def boa::BoaDataAnalyser::FilterFFT::doFFT ( self, interpolate = 0, windowing = 4,
windowSize = 0, Xstart = 0, Xend = 0)
```

DES : perform all the necessary steps to do a forward FFT

```

INP : (1) interpolate : force an interpolation to be done
      (default no : check of timing quality better than 0.1 %)
      (i) windowing : windowing function used to compute FFTs (default: Hamming)
      (i) windowSize : length of chunks to compute FFT on and to average
      (default: 0 = compute on the entire data serie)
      (i) Xstart, Xend: optional indices for using only part of the data

```

```
def boa::BoaDataAnalyser::FilterFFT::invFFT ( self, windowing = 4 )
```

DES : perform all the necessary steps to do a backward FFT

```
def boa::BoaDataAnalyser::FilterFFT::plotDataGram ( self, interpolate = 0, n = 1024,
window = 4, limitsZ = [ ] )
```

DES: Plot the Datagram of the Data

INP (i) n Number of points for the ffts

```
def boa::BoaDataAnalyser::FilterFFT::plotFFT ( self, plotPhase = 0, labelX
= 'Frequency [Hz]', labelY = 'Amplitude (a.b.u/sqrt(Hz))', \
limitsX=[], limitsY = [], logX = 1, logY = 1, overplot = 0, ci = 1)
```

DES: Plot the fft

```

INP: (str)  labelX/Y   : the X/Y label
      (2d f) limitsX/Y : the plot limits for X/Y
      (bol) plotPhase  : plot phase instead of amplitude (default no)

```

```
def boa::BoaDataAnalyser::FilterFFT::reduceAmplitude ( self, center = 50 ., width = 1 .,  
factor = 10 ., dB = 0)
```

DES: multiply the Fourier spectrum with a filter function

```

DBS: multiply the Fourier spectrum with a filter function
INP: (f) center: central frequency, in Hz
      (f) width: window FWHM
      (f) factor: attenuation factor
      (l) dB : is factor expressed in dB? (default: no)

```

def boa::BoaDataAnalyser::FilterFFT::taperAmplitude (self, above = ' ? ', N = 2)

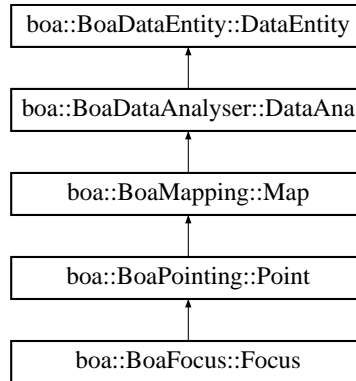
DES: Butterworth taper the amplitude above a certain frequency

INP: (f) above: frequency above which to taper

 (f) N: steepness parameter

C.6 boa::BoaFocus::Focus Class Reference

Inheritance diagram for `boa::BoaFocus::Focus::`



C.6.1 Detailed Description

NAM: `Focus` (class)

DES: An object of this class is responsible for the focus reduction of single or multiple scans and provides the offsets.

Public Member Functions

- `def __init__`
- `def reduce`
- `def solveFocus`

C.6.2 Member Function Documentation

`def boa::BoaFocus::Focus::__init__ (self)`

DES: Initialise an instance

Reimplemented from [boa::BoaPointing::Point](#).

`def boa::BoaFocus::Focus::reduce (self, datasetName = "", obstoProc = [])`

DES: Process a Focus scan - this method is called by the `apexCalibrator`

INP: (string) `datasetName`: path to the dataset to be reduced

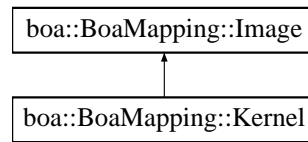
(i list) `obstoProc`: list of subscans to consider (default: all)

def boa::BoaFocus::Focus::solveFocus (self)

DES: compute the optimal focus position

C.7 boa::BoaMapping::Image Class Reference

Inheritance diagram for `boa::BoaMapping::Image`:



C.7.1 Detailed Description

NAM: `Image` (class)

DES: An object of this class describes an image and its axis

Public Member Functions

- `def __init__`
- `def __str__`
- `def computeRms`
- `def computeRmsBeam`
- `def computeWCS`
- `def display`
- `def dumpMap`
- `def extractSource`
- `def getPixel`
- `def meanDistribution`
- `def physicalCoordinates`
- `def rmsDistribution`
- `def smoothBy`
- `def smoothWith`
- `def wcs2phy`
- `def wcs2pix`
- `def writeFits`
- `def zoom`

Public Attributes

- **BeamSize**
 - **Coverage**
 - **Data**
 - **Header**
 - **RmsBeam**
-

- WCS
- Weight

C.7.2 Member Function Documentation

def boa::BoaMapping::Image::__str__ (self)

DES: Defines a string, shown when the print instruction is used.

def boa::BoaMapping::Image::computeRms (self, rmsKappa = 2, limitsX = [], limitsY = [])

DES: compute rms/beam in a map (dispersion between pixels)

INP: (f) rmsKappa: for kappa-sigma clipping before computing rms
(i lists) limitsX/Y: optionally define a sub-region (pixel coord)

def boa::BoaMapping::Image::computeRmsBeam (self, cell = 3, rmsKappa = 2, limitsX = [], limitsY = [])

DES: compute rms/beam in a map (smoothed at beam resolution)

INP: (f) cell: size of one beam in pixel
(f) rmsKappa: for kappa-sigma clipping before computing rms
(i lists) limitsX/Y: optionally define a sub-region (pixel coord)

def boa::BoaMapping::Image::computeWCS (self, pixelSize, sizeX = [], sizeY = [], minmax = [])

DES: fill main WCS keywords according to pixel size and map limits

INP: (int) pixelSize = size of pixel in arcsecond
(float) sizeX = map limits in azimuth, in arcsecond
(float) sizeY = map limits in elevation, in arcsecond
(float) minmax = [minAzoff,maxAzoff,minEloff,maxEloff] in this order

def boa::BoaMapping::Image::display (self, weight = 0, coverage = 0, style = 'idl4', caption = "", wedge = 1, aspect = 1, overplot = 0, doContour = 0, levels = [], labelContour = 0, limitsX = [], limitsY = [], limitsZ = [], showRms = 0, rmsKappa = 2)

DES: show the reconstructed maps in (Az,El)

INP: (boolean) weigth,coverage : plot the rms or weight map instead of signal map
(string) style : the style used for the color (default idl4)

(string) caption : the caption of the plot (default '')
(flt array) limitsX/Y/Z : the limits in X/Y/intensity
(boolean) wedge : draw a wedge ? (default : yes)
(boolean) aspect : keep the aspect ratio (default : yes)
(boolean) overplot : should we overplot this image (default : no)

```

(boolean) doContour      : draw contour instead of map (default : no)
(float array) levels     : the levels of the contours
                        (default : intensity progression)
(boolean) labelContour   : label the contour (default : no)
(boolean) showRms        : compute and display rms/beam? (def: yes)

```

def boa::BoaMapping::Image::dumpMap (self, fileName = 'BoaMap.sav')

```

DES: save an Image instance to a file
INP: (string) fileName: name of the output file
      (default = 'BoaMap.sav')

```

def boa::BoaMapping::Image::extractSource (self, gradient = 0, circular = 0, radius = -10, Xpos = 0., Ypos = 0., fixedPos = 0)

```

DES: fit a 2D Gaussian on a map -

```

def boa::BoaMapping::Image::getPixel (self, nbPix = 3)

```

DES: allow user to get pixel values using mouse
INP: (int) nbPix : size of area to compute average (default 3x3)

```

def boa::BoaMapping::Image::meanDistribution (self, cell = 3, limitsX = [], limitsY = [])

```

DES: compute and plot the distribution of means in the map
INP: (int) cell : size of cells on which mean values are computed (default: 3x3)
      (i lists) limitsX/Y: optionally define a sub-region (pixel coord)

```

def boa::BoaMapping::Image::physicalCoordinates (self)

```

DES: return arrays with physical units corresponding to the map

```

def boa::BoaMapping::Image::rmsDistribution (self, cell = 3)

```

DES: compute and plot the distribution of rms in the map
INP: (int) cell : size of cells on which rms are computed (default: 3x3)

```

def boa::BoaMapping::Image::smoothBy (self, Size)

```

DES: smooth the image with a 2D gaussian of given FWHM
INP: (float) Size : the FWHM of the smoothing gaussian

```

def boa::BoaMapping::Image::smoothWith (self, kernel)

DES: smooth the image with the given kernel
 INP: (kernel) : the kernel

def boa::BoaMapping::Image::wcs2phy (self, i, j)

DES: Convert from pixel coordinates to physical (world) coordinates
 INP: float (i,j) : the pixel coordinates to convert from
 OUT: float (X,Y) : the physical coordinates

We should switch to libwcs at some point

def boa::BoaMapping::Image::wcs2pix (self, X, Y)

DES: Convert from physical coordinates described by self.WCS
 to pixel coordinates
 INP: float (X,Y) : the physical coordinates to convert from
 OUT: float (i,j) : the pixel coordinates

We should switch to libwcs at some point

def boa::BoaMapping::Image::writeFits (self, outfile = 'boaMap.fits', overwrite = 0, limitsX = [], limitsY = [], intensityUnit = "Jy/beam")

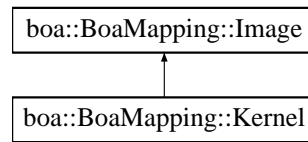
DES: store the current map (2D array with WCS info) to a FITS file
 INP: (str) outfile: output file name (default boaMap.fits)
 (bool) overwrite: overwrite existing file -
 default = 0: do not overwrite existing file
 (f list) limitsX/Y: optional map limits (in world coordinates)
 (string) intensityUnit: optional unit of the intensity (default: "Jy/beam")

def boa::BoaMapping::Image::zoom (self, mouse = 1, style = 'idl4', wedge = 1, limitsZ = [], aspect = 1, limitsX = [], limitsY = [], caption = None, doContour = 0, levels = [], showRms = 1, rmsKappa = 2)

DES: allow the user to select a region in the map to zoom in
 INP: (bool) mouse: use the mouse? (default: yes)
 (other parameters: same as display)

C.8 boa::BoaMapping::Kernel Class Reference

Inheritance diagram for boa::BoaMapping::Kernel::



C.8.1 Detailed Description

NAM: Kernel (class)
DES: define a kernel

Public Member Functions

- def [__init__](#)

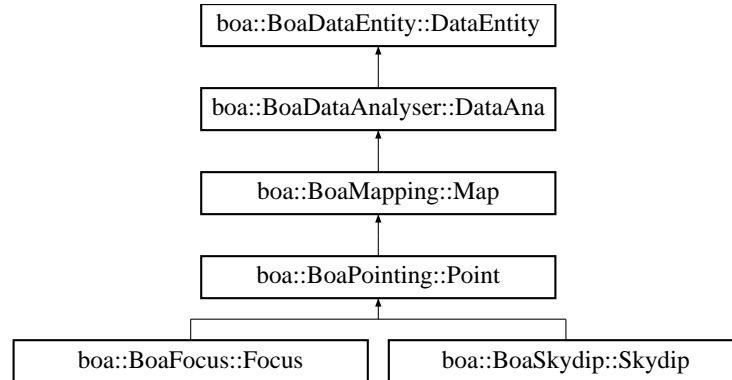
C.8.2 Member Function Documentation

def `boa::BoaMapping::Kernel::__init__ (self, pixelSize, beamSize)`

DES: Initialise an instance of a Kernel class
INP: (float) pixelSize: the physical size of a pixel
 (float) beamSize : the beam FWHM in the same unit

C.9 boa::BoaMapping::Map Class Reference

Inheritance diagram for `boa::BoaMapping::Map`:



C.9.1 Detailed Description

NAM: `Map` (class)

DES: An object of this class is responsible for the restoration of mapping data of single or multiple files.

Public Member Functions

- `def __init__`
- `def beamMap`
- `def chanMap`
- `def computeRmsFromMap`
- `def displayMap`
- `def doMap`
- `def getPixelFromMap`
- `def plotBoloRms`
- `def reduce`
- `def showMap`
- `def smoothMap`
- `def zoom`

C.9.2 Member Function Documentation

`def boa::BoaMapping::Map::__init__ (self)`

DES: Initialise an instance.

Reimplemented from `boa::BoaDataAnalyser::DataAna`.

Reimplemented in `boa::BoaFocus::Focus`, `boa::BoaPointing::Point`, and `boa::BoaSkydip::Skydip`.

```
def boa::BoaMapping::Map::beamMap ( self, chanList = [], channelFlag = [],
plotFlaggedChannels = 0, dataFlag = [], plotFlaggedData = 0, oversamp = 2.0, sizeX = [],
sizeY = [], style = 'idl4')
```

DES: build a beam map in (Az,El) coordinates

INP: (int list) chanList = channels to consider

(integer list) channelFlag : plot data from channels flagged or unflagged accordingly

(log) plotFlaggedChannels : channelFlag revers to flagged/unflagged data

(integer list) dataFlag : plot data flagged or unflagged accordingly

(log) plotFlaggedData : dataFlag revers to flagged/unflagged data

flag	plotFlagged	Plot..
'None'	0	all data
[]	0	unflagged data (default)
[]	1	data with at least one flag set
1	0	data with flag 1 not set
1	1	data with flag 1 set
[1,2]	0	data with neither flag 1 nor flag 2 set
[1,2]	1	data with either flag 1 or flag 2 set

(float) oversamp = oversampling factor (beam fwhm / pixel size). Default=2.

(list float) sizeX = limits in Az of the map

(list float) sizeY = limits in El of the map

```
def boa::BoaMapping::Map::chanMap ( self, chanList = [], channelFlag = [],
plotFlaggedChannels = 0, dataFlag = [], plotFlaggedData = 0, oversamp = 1., sizeX = [],
sizeY = [], style = 'idl4', limitsZ = [], center = 0, showRms = 0, rmsKappa = 2.)
```

DES: Compute and plot channel maps in HO offset coordinates

INP: (i list) chanList = channels to consider

(integer list) channelFlag : plot data from channels flagged or unflagged accordingly

(log) plotFlaggedChannels : channelFlag revers to flagged/unflagged data

(integer list) dataFlag : plot data flagged or unflagged accordingly

(log) plotFlaggedData : dataFlag revers to flagged/unflagged data

flag	plotFlagged	Plot..
'None'	0	all data
[]	0	unflagged data (default)
[]	1	data with at least one flag set
1	0	data with flag 1 not set
1	1	data with flag 1 set
[1,2]	0	data with neither flag 1 nor flag 2 set
[1,2]	1	data with either flag 1 or flag 2 set

(float) oversamp = oversampling factor (beam fwhm / pixel size). Default=2.

(2xfloat) sizeX = limits in Az of the map

(2xfloat) sizeY = limits in El of the map

(str) style = color table to use in images

(logical) center = if set, it will shift each map by the bolometer offsets.

Thereby it shifts the source to the center of each channel map.

(logical) showRms = compute and print rms/beam? (default: no)

(float) rmsKappa = kappa in kappa-sigma clipping used to compute rms

```
def boa::BoaMapping::Map::computeRmsFromMap ( self, rmsKappa = 2, limitsX = [],
limitsY = [])
```

DES: compute rms/beam in a map (dispersion between pixels)

INP: (f) rmsKappa: for kappa-sigma clipping before computing rms

(i lists) limitsX/Y: optionally define a sub-region (pixel coord)

```
def boa::BoaMapping::Map::doMap ( self, chanList = [], channelFlag = [],
plotFlaggedChannels = 0, dataFlag = [], plotFlaggedData = 0, oversamp = 2.0, beammap =
0, system = 'HO', sizeX = [], sizeY = [], limitsZ = [], style = 'idl4', wedge = 1, smooth =
0, noPlot = 0, caption = None, aspect = 1, showRms = 1, rmsKappa = 2., derotate = 0)
```

DES: reconstruct a map in (Az,El) coordinates combining bolometers

INP: (int list) chanList = channels to consider

(integer list) channelFlag : plot data from channels flagged or unflagged accordingly

(log) plotFlaggedChannels : channelFlag revers to flagged/unflagged data

(integer list) dataFlag : plot data flagged or unflagged accordingly

(log) plotFlaggedData : dataFlag revers to flagged/unflagged data

flag	plotFlagged	Plot..
'None'	0	all data
[]	0	unflagged data (default)
[]	1	data with at least one flag set
1	0	data with flag 1 not set
1	1	data with flag 1 set
[1,2]	0	data with neither flag 1 nor flag 2 set
[1,2]	1	data with either flag 1 or flag 2 set

(float) oversamp = oversampling factor (beam fwhm / pixel size). Default=2.

(log) beammap = compute a beam map (default: no)

(str) system = coord. system, one of 'HO' (Az,El *offsets*) or 'EQ'

(RA, Dec absolute coord.) - default = 'HO'

optionally 'EQFAST' to do only one rotation

on small maps (faster)

(list float) sizeX = limits in Az of the map

(list float) sizeY = limits in El of the map

(logical) noNan = remove NaN in self.Results?

(str) style = color table to use in image

(logical) smooth = do we smooth with beam? (default: no)

(logical) noPlot = do not plot the map? (default: no, i.e. yes we plot)

(str) caption = plot caption

(logical) aspect = keep aspect ratio? (default: yes)

(logical) showRms = compute and print rms/beam? (default: yes)

(float) rmsKappa = kappa in kappa-sigma clipping used to compute rms

(int) derotate = derotate Nasmyth array by Elevation

```
def boa::BoaMapping::Map::getPixelFromMap ( self, nbPix = 3)
```

DES: allow user to get pixel values using mouse

INP: (int) nbPix : size of area to compute average (default 3x3)

```
def boa::BoaMapping::Map::plotBoloRms ( self, smoothFactor = 1.5, style = 'idl4', limitsX
= [], limitsY = [], limitsZ = [], caption = ")
```

DES: plot the array with color scale showing rms

INP: (float) smoothFactor: the map is smooted by this factor x beam
 style, limits? : plot parameters

```
def boa::BoaMapping::Map::reduce ( self, datasetName = "", obstoProc = [], update = 0)
```

DES: Process a map scan - this method is called by the apexCalibrator

INP: (string) datasetName: path to the dataset to be reduced
 (i list) obstoProc: list of subscans to consider (default: all)

Reimplemented in [boa::BoaSkydip::Skydip](#).

```
def boa::BoaMapping::Map::showMap ( self, style = 'idl4', wedge = 1, limitsZ = [], aspect
= 1, limitsX = [], limitsY = [], caption = None, doContour = 0, levels = [], showRms = 1,
rmsKappa = 2)
```

DES: show the reconstructed map in (Az,El) or (Ra,Dec)

```
def boa::BoaMapping::Map::smoothMap ( self, Size)
```

DES: smooth the image with a 2D gaussian of given FWHM

INP: (float) Size : the FWHM of the smoothing gaussian

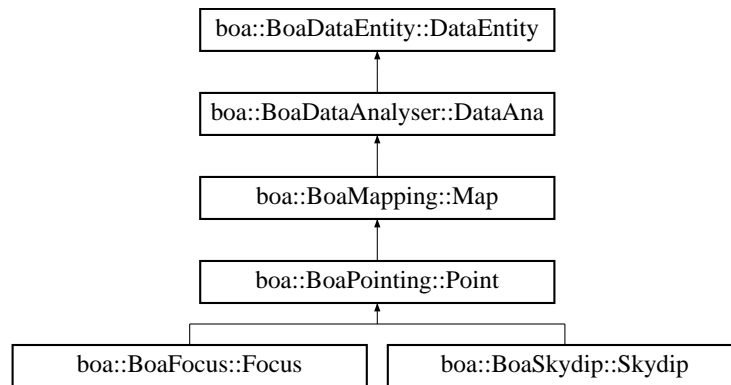
```
def boa::BoaMapping::Map::zoom ( self, mouse = 1, style = 'idl4', wedge = 1, limitsZ = [],
aspect = 1, limitsX = [], limitsY = [], caption = None, doContour = 0, levels = [], showRms =
1, rmsKappa = 2)
```

DES: allow the user to select a region in the map to zoom in

INP: (bool) mouse: use the mouse? (default: yes)
 (other parameters: same as showMap)

C.10 boa::BoaPointing::Point Class Reference

Inheritance diagram for `boa::BoaPointing::Point::`



C.10.1 Detailed Description

NAM: `Point` (class)

DES: An object of this class is responsible for the reduction of pointing scan(s)

Public Member Functions

- [def `__init__`](#)
- [def `arrayParameters`](#)
- [def `iterMap`](#)
- [def `showPointing`](#)
- [def `solvePointing`](#)
- [def `solvePointingOnMap`](#)
- [def `updateArrayParameters`](#)
- [def `writeModelData`](#)

C.10.2 Member Function Documentation

def `boa::BoaPointing::Point::__init__` ([self](#))

DES: Initialise an instance.

Reimplemented from [boa::BoaMapping::Map](#).

Reimplemented in [boa::BoaFocus::Focus](#), and [boa::BoaSkydip::Skydip](#).

def boa::BoaPointing::Point::arrayParameters (self, chanList = [], gradient = 0, circular = 0, radius = 0, plot = 0)

DES: determine the array parameters from the data

INP: (i list) chanList : the channel list to be used (default: current list)
 (1) gradient : remove a background gradient in the data (default: no)
 (1) circular : fit a cricular gaussian instead of an elliptical gaussian

def boa::BoaPointing::Point::iterMap (self, chanList = [], phase = 0, flag = 0, sizeX = [], sizeY = [])

DES: reconstruct a map in (Az,El) coordinates combining bolometers
 and using varying scale to zoom on signal

INP: (int list) chanList = channels to consider
 (int) phase = phase to plot
 (int) flag = flag values to consider
 (list float) sizeX = limits in Az of the map
 (list float) sizeY = limits in El of the map

def boa::BoaPointing::Point::showPointing (self, plot = 1, display = 1, noMap = 0, caption = "", aspect = 1, style = 'idl4', limitsZ = [])

DES: display results of last solvePointing (in text, and on the map if plot=1)

INP: (logical) plot : display the results on a map (default: no)
 (logical) display : display the result on screen (default: yes)

def boa::BoaPointing::Point::solvePointing (self, chanList = [], gradient = 0, circular = 0, radius = -5, Xpos = 0., Ypos = 0., fixedPos = 0, plot = 0, display = 1, caption = "", aspect = 1)

DES: compute the offset

INP: (int list) chanList: list of channels to be used (default: all)
 (boolean) gradient: shall we fit a gradient ? (default: no)
 (boolean) circular: fit a cricular gaussian instead of an elliptical gaussian
 (float) radius : use only bolo inside this radius (negative means multiple of beam) (de
 (float) Xpos, Ypos : source position if using fixed position
 (boolean) fixedPos : if set, don't fit position, but use Xpos, Ypos
 (boolean) plot : do we plot the results? (default: no)
 (boolean) display : display the result of the fit (default: yes)
 OUT: store in self.PoitingResult the results of the fit (i.e. all parameters
 as computed by mpfit routine). If mpfit failed, then self.PoitingResult
 is set to -1

def boa::BoaPointing::Point::solvePointingOnMap (self, gradient = 0, circular = 0, radius = -10, Xpos = 0., Ypos = 0., fixedPos = 0, plot = 0, display = 1, caption = "", aspect = 1, style = 'idl4')

DES: compute the offset on the data.Map object

```

INP: (boolean) gradient: shall we fit a gradient ? (default: no)
      (boolean) circular: fit a cricular gaussian instead of an elliptical gaussian
      (float)    radius  : use only bolo inside this radiu
                      (negative means multiple of beam - default: 10 beams)
      (float) Xpos,Ypos : source position if using fixed position
      (boolean) fixedPos : if set, don't fit position, but use Xpos, Ypos
      (boolean) plot     : do we plot the results? (default: no)
      (boolean) display  : display the result of the fit (default: yes)
OUT: store in self.PointingResult the results of the fit (i.e. all parameters
      as computed by mpfit routine). If mpfit failed, then self.PoitingResult
      is set to -1

```

```

WARNING : No Smoothing should be applied to the map
before using this function, or the fitted fwhm will be
useless, use fine oversamp to make reasonable fit

```

def boa::BoaPointing::Point::updateArrayParameters (self, filename = None)

```

DES: Update the Parameters Offsets with the computed values
INP: (str) filename : optional output file name

```

def boa::BoaPointing::Point::writeModelData (self)

```

Generate one line to be written in the .dat file used for
determining pointing model

```

C.11 boa::BoaDataEntity::ScanParameter Class Reference

C.11.1 Detailed Description

NAM: ScanParameter (class)

DES: Define all parameters (coordinates, time) for a scan

Public Member Functions

- def `__add__`
- def `__init__`
- def `__str__`
- def `caption`
- def `computeAzElOffsets`
- def `computeOnOff`
- def `computeParAngle`
- def `computeRa0De0`
- def `computeRaDec`
- def `computeRaDecOffsets`
- def `findSubscan`
- def `findSubscanFB`
- def `flag`
- def `get`
- def `he3SmoothInterpolate`
- def `plotAzEl`
- def `plotAzElAcceleration`
- def `plotAzElOffset`
- def `plotAzElSpeed`
- def `plotAzimuth`
- def `plotAzimuthOffset`
- def `plotElevation`
- def `plotElevationOffset`
- def `plotSubscan`
- def `plotSubscanOffsets`
- def `selectPhase`
- def `unflag`

Public Attributes

- `Az`
 - `AzOff`
 - `BasLat`
 - `BasLon`
-

- Dec
- DecOff
- El
- ElOff
- Flags
- FocX
- FocY
- FocZ
- LatOff
- LatPole
- LonOff
- LonPole
- LST
- MJD
- NInt
- NObs
- NoddingState
- ParAngle
- PhiX
- PhiY
- RA
- RAOff
- Rot
- SubscanIndex
- UT
- WobblerPos

C.11.2 Member Function Documentation

def boa::BoaDataEntity::ScanParameter::__init__ (self)

DES: Instanciation of a new ScanParameter object

def boa::BoaDataEntity::ScanParameter::__str__ (self)

DES: Defines a string, shown when the print instruction is used.

def boa::BoaDataEntity::ScanParameter::caption (self)

DES: Return a short caption of the scan

def boa::BoaDataEntity::ScanParameter::computeAzElOffsets (self)

DES: compute telescope Az, El offsets w.r.t. the source, using antenna Az, El and RA, Dec of the source

def boa::BoaDataEntity::ScanParameter::computeOnOff (self)

DES: determine ON-OFF pairs from content of WobblerSta, and fill OnOffPairs attribute with pairs of integration numbers.
The result is a 2 x Nb_Integ. array of integers.

def boa::BoaDataEntity::ScanParameter::computeParAngle (self)

DES: compute parallactic angle

def boa::BoaDataEntity::ScanParameter::computeRa0De0 (self)

DES: compute source coordinates in equatorial system

def boa::BoaDataEntity::ScanParameter::computeRaDec (self)

DES: compute telescope RA, Dec positions from Az, El

def boa::BoaDataEntity::ScanParameter::computeRaDecOffsets (self)

DES: compute telescope RA, Dec offsets w.r.t. the source

def boa::BoaDataEntity::ScanParameter::findSubscan (self, threshold = 1.)

DES: compute subscan indices from steps in az, el
INP: (float) threshold = value (in arcsec²) of (d_az² + d_el²) step
used to detect turnovers / stationnary points

def boa::BoaDataEntity::ScanParameter::findSubscanFB (self, azMax = 1000.)

DES: compute subscan indices from steps in az, el
INP: (float) azMax = azimuth offset where subscans are marked

def boa::BoaDataEntity::ScanParameter::flag (self, dataType = "", below = ' ? ', above = ' ? ', flag = 1)

DES: flag data based on dataType

INP: (float) below : flag dataType < below (default max)
 (float) above : flag dataType > above (default min)
 (integer list) flag : flag values (default 1)

below and above should be in unit of the flagged data,
 except for 'Lon' and 'Lat' where they should be in arcsec

def boa::BoaDataEntity::ScanParameter::get (self, dataType = ' ', flag = [], getFlagged = 0, subscans = [])

DES: get data of the ScanParam class

INP: (string) dataType : type of data

LST MJD Az El AzOff ElOff focX focY focZ
 (integer list) flag : retrieve data flagged or unflagged accordingly
 (log) getFlagged : flag revers to flagged/unflagged data

flag		getFlagged		Retrieve..
'None'		0		all data
[]		0		unflagged data (default)
[]		1		data with at least one flag set
1		0		data with flag 1 not set
1		1		data with flag 1 set
[1,2]		0		data with neither flag 1 nor flag 2 set
[1,2]		1		data with either flag 1 or flag 2 set

(i list) subscans : optionnally select subscan(s)

OUT: (float array) : the requested data

returned data are in the stored unit except for offsets which are
 converted to arcsec

def boa::BoaDataEntity::ScanParameter::he3SmoothInterpolate (self, flag = [], getFlagged = 0)

DES: this is a *function* which *returns* an array with He3 temperatures
 interpolated to the data timestamps, with a smoothing (boxcar window
 applied) before interpolating

INP: (integer list) flag : retrieve data flagged or unflagged accordingly
 (log) getFlagged : flag revers to flagged/unflagged data

flag		getFlagged		Retrieve..
'None'		0		all data
[]		0		unflagged data (default)
[]		1		data with at least one flag set
1		0		data with flag 1 not set
1		1		data with flag 1 set
[1,2]		0		data with neither flag 1 nor flag 2 set
[1,2]		1		data with either flag 1 or flag 2 set

OUT: (f array) interpolated He3 temperatures are returned

def boa::BoaDataEntity::ScanParameter::plotAzEl (self, flag = [], plotFlagged = 0, limitsX = [], limitsY = [], style = 'l', ci = 1, overplot = 0, aspect = 1)

DES: plot azimuth vs. elevation

INP: (int list) flag : plot data flagged or unflagged accordingly

(log) plotFlagged : flag reverses to flagged/unflagged data

flag	plotFlagged	Plot..
'None'	0	all data
[]	0	unflagged data (default)
[]	1	data with at least one flag set
1	0	data with flag 1 not set
1	1	data with flag 1 set
[1,2]	0	data with neither flag 1 nor flag 2 set
[1,2]	1	data with either flag 1 or flag 2 set

def boa::BoaDataEntity::ScanParameter::plotAzElAcceleration (self, flag = [], plotFlagged = 0, limitsX = [], limitsY = [], style = 'l', ci = 1, overplot = 0, aspect = 1)

DES: plot azimuth and elevation acceleration

INP: (int list) flag : plot data flagged or unflagged accordingly

(log) plotFlagged : flag reverses to flagged/unflagged data

flag	plotFlagged	Plot..
'None'	0	all data
[]	0	unflagged data (default)
[]	1	data with at least one flag set
1	0	data with flag 1 not set
1	1	data with flag 1 set
[1,2]	0	data with neither flag 1 nor flag 2 set
[1,2]	1	data with either flag 1 or flag 2 set

INP: (int) flag : flag to be plot (default 0 : valid data, -1 plot all)

def boa::BoaDataEntity::ScanParameter::plotAzElOffset (self, flag = [], plotFlagged = 0, limitsX = [], limitsY = [], style = 'l', ci = 1, overplot = 0, aspect = 0)

DES: plot elevation offset versus azimuth offset

INP: (int list) flag : plot data flagged or unflagged accordingly

(log) plotFlagged : flag reverses to flagged/unflagged data

flag	plotFlagged	Plot..
'None'	0	all data
[]	0	unflagged data (default)
[]	1	data with at least one flag set
1	0	data with flag 1 not set
1	1	data with flag 1 set
[1,2]	0	data with neither flag 1 nor flag 2 set
[1,2]	1	data with either flag 1 or flag 2 set

def boa::BoaDataEntity::ScanParameter::plotAzElSpeed (self, flag = [], plotFlagged = 0, limitsX = [], limitsY = [], style = 'l', ci = 1, overplot = 0, aspect = 1)

DES: plot azimuth and elevation speed

```

INP: (int list)  flag : plot data flagged or unflagged accordingly
      (log) plotFlagged : flag revers to flagged/unflagged data
          flag | plotFlagged | Plot..
          'None' | 0 | all data
          [] | 0 | unflagged data (default)
          [] | 1 | data with at least one flag set
          1 | 0 | data with flag 1 not set
          1 | 1 | data with flag 1 set
          [1,2] | 0 | data with neither flag 1 nor flag 2 set
          [1,2] | 1 | data with either flag 1 or flag 2 set

```

def boa::BoaDataEntity::ScanParameter::plotAzimuth (self, flag = [], plotFlagged = 0, limitsX = [], limitsY = [], style = 'l', ci = 1, overplot = 0, aspect = 1)

DES: plot time series of azimuth

```

INP: (int list)  flag : plot data flagged or unflagged accordingly
      (log) plotFlagged : flag revers to flagged/unflagged data
          flag | plotFlagged | Plot..
          'None' | 0 | all data
          [] | 0 | unflagged data (default)
          [] | 1 | data with at least one flag set
          1 | 0 | data with flag 1 not set
          1 | 1 | data with flag 1 set
          [1,2] | 0 | data with neither flag 1 nor flag 2 set
          [1,2] | 1 | data with either flag 1 or flag 2 set

```

def boa::BoaDataEntity::ScanParameter::plotAzimuthOffset (self, flag = [], plotFlagged = 0, limitsX = [], limitsY = [], style = 'l', ci = 1, overplot = 0, aspect = 1)

DES: plot time series of azimuth offset

```

INP: (int list)  flag : plot data flagged or unflagged accordingly
      (log) plotFlagged : flag revers to flagged/unflagged data
          flag | plotFlagged | Plot..
          'None' | 0 | all data
          [] | 0 | unflagged data (default)
          [] | 1 | data with at least one flag set
          1 | 0 | data with flag 1 not set
          1 | 1 | data with flag 1 set
          [1,2] | 0 | data with neither flag 1 nor flag 2 set
          [1,2] | 1 | data with either flag 1 or flag 2 set

```

def boa::BoaDataEntity::ScanParameter::plotElevation (self, flag = [], plotFlagged = 0, limitsX = [], limitsY = [], style = 'l', ci = 1, overplot = 0, aspect = 1)

DES: plot time series of elevation

```

INP: (int list)  flag : plot data flagged or unflagged accordingly
      (log) plotFlagged : flag revers to flagged/unflagged data
          flag | plotFlagged | Plot..
          'None' | 0 | all data
          [] | 0 | unflagged data (default)

```

[]		1		data with at least one flag set
1		0		data with flag 1 not set
1		1		data with flag 1 set
[1,2]		0		data with neither flag 1 nor flag 2 set
[1,2]		1		data with either flag 1 or flag 2 set

def boa::BoaDataEntity::ScanParameter::plotElevationOffset (self, flag = [], plotFlagged = 0, limitsX = [], limitsY = [], style = 'l', ci = 1, overplot = 0, aspect = 1)

DES: plot time series of elevation offset

INP: (int list) flag : plot data flagged or unflagged accordingly

(log) plotFlagged : flag reverses to flagged/unflagged data

flag		plotFlagged		Plot..
'None'		0		all data
[]		0		unflagged data (default)
[]		1		data with at least one flag set
1		0		data with flag 1 not set
1		1		data with flag 1 set
[1,2]		0		data with neither flag 1 nor flag 2 set
[1,2]		1		data with either flag 1 or flag 2 set

def boa::BoaDataEntity::ScanParameter::plotSubscan (self)

DES: generate a plot showing starting and ending times of subscans

def boa::BoaDataEntity::ScanParameter::plotSubscanOffsets (self, overplot = 0)

DES: Use four colours to show subscans on the Az, El pattern

INP: (logical) overplot : if set, do not plot AzElOffset - assume these have been plotted already

def boa::BoaDataEntity::ScanParameter::selectPhase (self, phase)

NAM: selectPhase (method)

DES: Keep only parameters (times, positions) associated with Data(ON) or Data(OFF)

INP: (int) phase: phase to keep, 1=ON, 2=OFF

def boa::BoaDataEntity::ScanParameter::unflag (self, dataType = "", below = ' ?', above = ' ?', flag = [])

DES: unflag data based on dataType

INP: (float) below : unflag dataType < below (default max)

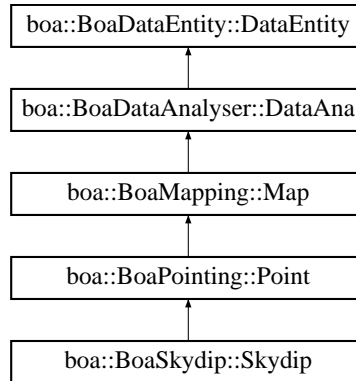
(float) above : unflag dataType > above (default min)

(integer list) flag : flag values (default []: unset all flags)

below and above should be in unit of the flagged data,
except for 'Lon' and 'Lat' where they should be in arcsec

C.12 boa::BoaSkydip::Skydip Class Reference

Inheritance diagram for `boa::BoaSkydip::Skydip::`



C.12.1 Detailed Description

NAM: `Skydip` (class)

DES: An object of this class is responsible for the reduction of skydips and provides zenithal opacity

Public Member Functions

- def [__init__](#)
- def [findElAccSubscan](#)
- def [findElSubscan](#)
- def [reduce](#)
- def [solveSkydip](#)

C.12.2 Member Function Documentation

def `boa::BoaSkydip::Skydip::__init__` (self)

DES: Initialise an instance

Reimplemented from [boa::BoaPointing::Point](#).

def `boa::BoaSkydip::Skydip::findElAccSubscan` (self)

determine subscans by looking at elevation acceleration

def boa::BoaSkydip::Skydip::findElSubscan (self, tolerance = 0.2, delta = 5.e-4)

DES: Determine subscans by looking at elevation

INP: (f) tolerance: tolerance (in deg) to assume that we're still in the same subscan

(f) delta : minimum variation of El (in deg) for the steps between subscans

def boa::BoaSkydip::Skydip::reduce (self, datasetName = "", obstoProc = [], blind = 317)

DES: Process a skydip scan - this method is called by the apexCalibrator

INP: (string) datasetName: path to the dataset to be reduced

(i list) obstoProc: list of subscans to consider (default: all)

(i) blind: channel number of the ref. (blind) bolo

!!! SPECIFIC TO LABOCA !!!

Reimplemented from [boa::BoaMapping::Map](#).

def boa::BoaSkydip::Skydip::solveSkydip (self)

DES: compute the zenithal opacity

C.13 boa::BoaDataEntity::Telescope Class Reference

C.13.1 Detailed Description

```
if __tag__[:12] == '$Name: APECS': boa_version = __tag__[7:-2] else: boa_version = 'Unknown'
print " version: %s" % boa_version
```

```
NAM: Telescope (class)
DES: Define all the useful parameters of a telescope
```

Public Member Functions

- def [__init__](#)
- def [__str__](#)
- def [set](#)

C.13.2 Member Function Documentation

def boa::BoaDataEntity::Telescope::__init__ ([self](#))

DES: Instanciation of a Telescope object

def boa::BoaDataEntity::Telescope::__str__ ([self](#))

DES: Defines a string which is shown when the print instruction is used.

def boa::BoaDataEntity::Telescope::set ([self](#), [name](#) = "", [diameter](#) = 0.0, [longitude](#) = 0.0, [latitude](#) = 0.0, [elevation](#) = 0.0)

DES: set all the parameters

Index

- `__init__`
 - `boa::BoaDataAnalyser::DataAna`, 93
 - `boa::BoaDataEntity::BolometerArray`, 87
 - `boa::BoaDataEntity::DataEntity`, 107
 - `boa::BoaDataEntity::ScanParameter`, 132
 - `boa::BoaDataEntity::Telescope`, 141
 - `boa::BoaFocus::Focus`, 117
 - `boa::BoaMapping::Kernel`, 123
 - `boa::BoaMapping::Map`, 124
 - `boa::BoaPointing::Point`, 128
 - `boa::BoaSkydip::Skydip`, 139
- `__str__`
 - `boa::BoaDataEntity::BolometerArray`, 87
 - `boa::BoaDataEntity::DataEntity`, 107
 - `boa::BoaDataEntity::ScanParameter`, 132
 - `boa::BoaDataEntity::Telescope`, 141
 - `boa::BoaMapping::Image`, 120
- `arrayParameters`
 - `boa::BoaPointing::Point`, 128
- `backup`
 - `boa::BoaDataEntity::DataEntity`, 107
- `bandRms`
 - `boa::BoaDataAnalyser::DataAna`, 93
- `beamMap`
 - `boa::BoaMapping::Map`, 124
- `blankAmplitude`
 - `boa::BoaDataAnalyser::FilterFFT`, 114
- `blankFreq`
 - `boa::BoaDataAnalyser::DataAna`, 93
- `boa::BoaDataAnalyser::DataAna`, 91
- `boa::BoaDataAnalyser::DataAna`
 - `__init__`, 93
 - `bandRms`, 93
 - `blankFreq`, 93
 - `computeCorTimeShift`, 93
 - `computeWeight`, 94
 - `computeWeights`, 94
 - `correctOpacity`, 94
 - `correlatedNoiseRemoval`, 94
 - `corrPCA`, 94
 - `corrPCA_old`, 95
 - `despike`, 95
 - `flag`, 95
 - `flagAccel`, 95
 - `flagFractionRms`, 95
 - `flagInTime`, 96
 - `flagLon`, 96
 - `flagMJD`, 96
 - `flagPosition`, 96
 - `flagSpeed`, 96
 - `flagSubscan`, 97
 - `flagTurnaround`, 97
 - `flatfield`, 97
 - `flattenFreq`, 97
 - `iterativeDespike`, 97
 - `medianBaseline`, 97
 - `medianFilter`, 98
 - `medianNoiseRemoval`, 98
 - `plotCorDist`, 98
 - `plotCorMatrix`, 98
 - `plotDataGram`, 99
 - `plotFFT`, 99
 - `plotMean`, 99
 - `plotMeanChan`, 100
 - `plotRms`, 100
 - `plotRmsChan`, 100
 - `polynomialBaseline`, 101
 - `read`, 101
 - `readFFCF`, 101
 - `rebin`, 102
 - `reduceFreq`, 102
 - `slidingRms`, 102
 - `taperFreq`, 102
 - `timeshiftAzEl`, 102
 - `timeShiftChan`, 103
 - `timeShiftChanList`, 103

- unflag, 103
- unflagAccel, 103
- unflagInTime, 103
- unflagLon, 104
- unflagMJD, 104
- unflagPosition, 104
- unflagSpeed, 104
- unflagSubscan, 104
- unflagTurnaround, 105
- writeFFCF, 105
- zeroStart, 105
- boa::BoaDataAnalyser::FilterFFT, 114
- boa::BoaDataAnalyser::FilterFFT
 - blankAmplitude, 114
 - doDataGram, 114
 - doFFT, 115
 - invFFT, 115
 - plotDataGram, 115
 - plotFFT, 115
 - reduceAmplitude, 115
 - taperAmplitude, 115
- boa::BoaDataEntity::BolometerArray, 86
- boa::BoaDataEntity::BolometerArray
 - __init__, 87
 - __str__, 87
 - checkChanList, 87
 - flag, 87
 - flipOffsets, 87
 - get, 87
 - getChanIndex, 88
 - getChanSep, 88
 - plotArray, 88
 - plotGain, 88
 - printCurrChanList, 88
 - readAdditionnalIndexFile, 88
 - readAsciiRcp, 89
 - readRCPfile, 89
 - rotateArray, 89
 - selectAdditionnalIndex, 89
 - setCurrChanList, 89
 - unflag, 89
 - updateRCP, 90
 - writeAsciiRcp, 90
 - writeRCPfile, 90
- boa::BoaDataEntity::DataEntity, 106
- boa::BoaDataEntity::DataEntity
 - __init__, 107
 - __str__, 107
 - backup, 107
 - dumpData, 107
 - getChanData, 107
 - getChanListData, 108
 - loadExchange, 108
 - plotCorrel, 108
 - read, 109
 - reset, 109
 - restore, 109
 - restoreData, 109
 - saveExchange, 109
 - saveMambo, 110
 - selectPhase, 110
 - signal, 110
 - signalHist, 110
- boa::BoaDataEntity::ScanParameter, 131
- boa::BoaDataEntity::ScanParameter
 - __init__, 132
 - __str__, 132
 - caption, 132
 - computeAzElOffsets, 132
 - computeOnOff, 133
 - computeParAngle, 133
 - computeRa0De0, 133
 - computeRaDec, 133
 - computeRaDecOffsets, 133
 - findSubscan, 133
 - findSubscanFB, 133
 - flag, 133
 - get, 134
 - he3SmoothInterpolate, 134
 - plotAzEl, 134
 - plotAzElAcceleration, 135
 - plotAzElOffset, 135
 - plotAzElSpeed, 135
 - plotAzimuth, 136
 - plotAzimuthOffset, 136
 - plotElevation, 136
 - plotElevationOffset, 137
 - plotSubscan, 137
 - plotSubscanOffsets, 137
 - selectPhase, 137
 - unflag, 137
- boa::BoaDataEntity::Telescope, 141
- boa::BoaDataEntity::Telescope
 - __init__, 141

-
- `__str__`, 141
 - `set`, 141
 - `boa::BoaFocus::Focus`, 117
 - `boa::BoaFocus::Focus`
 - `__init__`, 117
 - `reduce`, 117
 - `solveFocus`, 117
 - `boa::BoaMapping::Image`, 119
 - `boa::BoaMapping::Image`
 - `__str__`, 120
 - `computeRms`, 120
 - `computeRmsBeam`, 120
 - `computeWCS`, 120
 - `display`, 120
 - `dumpMap`, 121
 - `extractSource`, 121
 - `getPixel`, 121
 - `meanDistribution`, 121
 - `physicalCoordinates`, 121
 - `rmsDistribution`, 121
 - `smoothBy`, 121
 - `smoothWith`, 121
 - `wcs2phy`, 122
 - `wcs2pix`, 122
 - `writeFits`, 122
 - `zoom`, 122
 - `boa::BoaMapping::Kernel`, 123
 - `boa::BoaMapping::Kernel`
 - `__init__`, 123
 - `boa::BoaMapping::Map`, 124
 - `boa::BoaMapping::Map`
 - `__init__`, 124
 - `beamMap`, 124
 - `chanMap`, 125
 - `computeRmsFromMap`, 125
 - `doMap`, 126
 - `getPixelFromMap`, 126
 - `plotBoloRms`, 126
 - `reduce`, 127
 - `showMap`, 127
 - `smoothMap`, 127
 - `zoom`, 127
 - `boa::BoaPointing::Point`, 128
 - `boa::BoaPointing::Point`
 - `__init__`, 128
 - `arrayParameters`, 128
 - `iterMap`, 129
 - `showPointing`, 129
 - `solvePointing`, 129
 - `solvePointingOnMap`, 129
 - `updateArrayParameters`, 130
 - `writeModelData`, 130
 - `boa::BoaSkydip::Skydip`, 139
 - `boa::BoaSkydip::Skydip`
 - `__init__`, 139
 - `findElAccSubscan`, 139
 - `findElSubscan`, 139
 - `reduce`, 140
 - `solveSkydip`, 140
 - `boa::Bogli::Interface::Fenetre`, 112
 - `dessine`, 112
 - `saisie`, 112
 - `caption`
 - `boa::BoaDataEntity::ScanParameter`, 132
 - `chanMap`
 - `boa::BoaMapping::Map`, 125
 - `checkChanList`
 - `boa::BoaDataEntity::BolometerArray`, 87
 - `computeAzElOffsets`
 - `boa::BoaDataEntity::ScanParameter`, 132
 - `computeCorTimeShift`
 - `boa::BoaDataAnalyser::DataAna`, 93
 - `computeOnOff`
 - `boa::BoaDataEntity::ScanParameter`, 133
 - `computeParAngle`
 - `boa::BoaDataEntity::ScanParameter`, 133
 - `computeRa0De0`
 - `boa::BoaDataEntity::ScanParameter`, 133
 - `computeRaDec`
 - `boa::BoaDataEntity::ScanParameter`, 133
 - `computeRaDecOffsets`
 - `boa::BoaDataEntity::ScanParameter`, 133
 - `computeRms`
 - `boa::BoaMapping::Image`, 120
 - `computeRmsBeam`
 - `boa::BoaMapping::Image`, 120
 - `computeRmsFromMap`
 - `boa::BoaMapping::Map`, 125
 - `computeWCS`
 - `boa::BoaMapping::Image`, 120
 - `computeWeight`
 - `boa::BoaDataAnalyser::DataAna`, 94
 - `computeWeights`
-

-
- boa::BoaDataAnalyser::DataAna, 94
 - correctOpacity
 - boa::BoaDataAnalyser::DataAna, 94
 - correlatedNoiseRemoval
 - boa::BoaDataAnalyser::DataAna, 94
 - corrPCA
 - boa::BoaDataAnalyser::DataAna, 94
 - corrPCA_old
 - boa::BoaDataAnalyser::DataAna, 95
 - despike
 - boa::BoaDataAnalyser::DataAna, 95
 - dessine
 - boa::Bogli::Interface::Fenetre, 112
 - display
 - boa::BoaMapping::Image, 120
 - doDataGram
 - boa::BoaDataAnalyser::FilterFFT, 114
 - doFFT
 - boa::BoaDataAnalyser::FilterFFT, 115
 - doMap
 - boa::BoaMapping::Map, 126
 - dumpData
 - boa::BoaDataEntity::DataEntity, 107
 - dumpMap
 - boa::BoaMapping::Image, 121
 - extractSource
 - boa::BoaMapping::Image, 121
 - findElAccSubscan
 - boa::BoaSkydip::Skydip, 139
 - findElSubscan
 - boa::BoaSkydip::Skydip, 139
 - findSubscan
 - boa::BoaDataEntity::ScanParameter, 133
 - findSubscanFB
 - boa::BoaDataEntity::ScanParameter, 133
 - flag
 - boa::BoaDataAnalyser::DataAna, 95
 - boa::BoaDataEntity::BolometerArray, 87
 - boa::BoaDataEntity::ScanParameter, 133
 - flagAccel
 - boa::BoaDataAnalyser::DataAna, 95
 - flagFractionRms
 - boa::BoaDataAnalyser::DataAna, 95
 - flagInTime
 - boa::BoaDataAnalyser::DataAna, 96
 - flagLon
 - boa::BoaDataAnalyser::DataAna, 96
 - flagMJD
 - boa::BoaDataAnalyser::DataAna, 96
 - flagPosition
 - boa::BoaDataAnalyser::DataAna, 96
 - flagSpeed
 - boa::BoaDataAnalyser::DataAna, 96
 - flagSubscan
 - boa::BoaDataAnalyser::DataAna, 97
 - flagTurnaround
 - boa::BoaDataAnalyser::DataAna, 97
 - flatfield
 - boa::BoaDataAnalyser::DataAna, 97
 - flattenFreq
 - boa::BoaDataAnalyser::DataAna, 97
 - flipOffsets
 - boa::BoaDataEntity::BolometerArray, 87
 - get
 - boa::BoaDataEntity::BolometerArray, 87
 - boa::BoaDataEntity::ScanParameter, 134
 - getChanData
 - boa::BoaDataEntity::DataEntity, 107
 - getChanIndex
 - boa::BoaDataEntity::BolometerArray, 88
 - getChanListData
 - boa::BoaDataEntity::DataEntity, 108
 - getChanSep
 - boa::BoaDataEntity::BolometerArray, 88
 - getPixel
 - boa::BoaMapping::Image, 121
 - getPixelFromMap
 - boa::BoaMapping::Map, 126
 - he3SmoothInterpolate
 - boa::BoaDataEntity::ScanParameter, 134
 - invFFT
 - boa::BoaDataAnalyser::FilterFFT, 115
 - iterativeDespike
 - boa::BoaDataAnalyser::DataAna, 97
 - iterMap
 - boa::BoaPointing::Point, 129
 - loadExchange
 - boa::BoaDataEntity::DataEntity, 108
-

- meanDistribution
 - boa::BoaMapping::Image, [121](#)
 - medianBaseline
 - boa::BoaDataAnalyser::DataAna, [97](#)
 - medianFilter
 - boa::BoaDataAnalyser::DataAna, [98](#)
 - medianNoiseRemoval
 - boa::BoaDataAnalyser::DataAna, [98](#)
 - physicalCoordinates
 - boa::BoaMapping::Image, [121](#)
 - plotArray
 - boa::BoaDataEntity::BolometerArray, [88](#)
 - plotAzEl
 - boa::BoaDataEntity::ScanParameter, [134](#)
 - plotAzElAcceleration
 - boa::BoaDataEntity::ScanParameter, [135](#)
 - plotAzElOffset
 - boa::BoaDataEntity::ScanParameter, [135](#)
 - plotAzElSpeed
 - boa::BoaDataEntity::ScanParameter, [135](#)
 - plotAzimuth
 - boa::BoaDataEntity::ScanParameter, [136](#)
 - plotAzimuthOffset
 - boa::BoaDataEntity::ScanParameter, [136](#)
 - plotBoloRms
 - boa::BoaMapping::Map, [126](#)
 - plotCorDist
 - boa::BoaDataAnalyser::DataAna, [98](#)
 - plotCorMatrix
 - boa::BoaDataAnalyser::DataAna, [98](#)
 - plotCorrel
 - boa::BoaDataEntity::DataEntity, [108](#)
 - plotDataGram
 - boa::BoaDataAnalyser::DataAna, [99](#)
 - boa::BoaDataAnalyser::FilterFFT, [115](#)
 - plotElevation
 - boa::BoaDataEntity::ScanParameter, [136](#)
 - plotElevationOffset
 - boa::BoaDataEntity::ScanParameter, [137](#)
 - plotFFT
 - boa::BoaDataAnalyser::DataAna, [99](#)
 - boa::BoaDataAnalyser::FilterFFT, [115](#)
 - plotGain
 - boa::BoaDataEntity::BolometerArray, [88](#)
 - plotMean
 - boa::BoaDataAnalyser::DataAna, [99](#)
 - plotMeanChan
 - boa::BoaDataAnalyser::DataAna, [100](#)
 - plotRms
 - boa::BoaDataAnalyser::DataAna, [100](#)
 - plotRmsChan
 - boa::BoaDataAnalyser::DataAna, [100](#)
 - plotSubscan
 - boa::BoaDataEntity::ScanParameter, [137](#)
 - plotSubscanOffsets
 - boa::BoaDataEntity::ScanParameter, [137](#)
 - polynomialBaseline
 - boa::BoaDataAnalyser::DataAna, [101](#)
 - printCurrChanList
 - boa::BoaDataEntity::BolometerArray, [88](#)
 - read
 - boa::BoaDataAnalyser::DataAna, [101](#)
 - boa::BoaDataEntity::DataEntity, [109](#)
 - readAdditionnalIndexFile
 - boa::BoaDataEntity::BolometerArray, [88](#)
 - readAsciiRcp
 - boa::BoaDataEntity::BolometerArray, [89](#)
 - readFFCF
 - boa::BoaDataAnalyser::DataAna, [101](#)
 - readRCPfile
 - boa::BoaDataEntity::BolometerArray, [89](#)
 - rebin
 - boa::BoaDataAnalyser::DataAna, [102](#)
 - reduce
 - boa::BoaFocus::Focus, [117](#)
 - boa::BoaMapping::Map, [127](#)
 - boa::BoaSkydip::Skydip, [140](#)
 - reduceAmplitude
 - boa::BoaDataAnalyser::FilterFFT, [115](#)
 - reduceFreq
 - boa::BoaDataAnalyser::DataAna, [102](#)
 - reset
 - boa::BoaDataEntity::DataEntity, [109](#)
 - restore
 - boa::BoaDataEntity::DataEntity, [109](#)
 - restoreData
 - boa::BoaDataEntity::DataEntity, [109](#)
 - rmsDistribution
 - boa::BoaMapping::Image, [121](#)
 - rotateArray
 - boa::BoaDataEntity::BolometerArray, [89](#)
-

-
- saisie
 - boa::Bogli::Interface::Fenetre, [112](#)
 - saveExchange
 - boa::BoaDataEntity::DataEntity, [109](#)
 - saveMambo
 - boa::BoaDataEntity::DataEntity, [110](#)
 - selectAdditionnalIndex
 - boa::BoaDataEntity::BolometerArray, [89](#)
 - selectPhase
 - boa::BoaDataEntity::DataEntity, [110](#)
 - boa::BoaDataEntity::ScanParameter, [137](#)
 - set
 - boa::BoaDataEntity::Telescope, [141](#)
 - setCurrChanList
 - boa::BoaDataEntity::BolometerArray, [89](#)
 - showMap
 - boa::BoaMapping::Map, [127](#)
 - showPointing
 - boa::BoaPointing::Point, [129](#)
 - signal
 - boa::BoaDataEntity::DataEntity, [110](#)
 - signalHist
 - boa::BoaDataEntity::DataEntity, [110](#)
 - slidingRms
 - boa::BoaDataAnalyser::DataAna, [102](#)
 - smoothBy
 - boa::BoaMapping::Image, [121](#)
 - smoothMap
 - boa::BoaMapping::Map, [127](#)
 - smoothWith
 - boa::BoaMapping::Image, [121](#)
 - solveFocus
 - boa::BoaFocus::Focus, [117](#)
 - solvePointing
 - boa::BoaPointing::Point, [129](#)
 - solvePointingOnMap
 - boa::BoaPointing::Point, [129](#)
 - solveSkydip
 - boa::BoaSkydip::Skydip, [140](#)
 - taperAmplitude
 - boa::BoaDataAnalyser::FilterFFT, [115](#)
 - taperFreq
 - boa::BoaDataAnalyser::DataAna, [102](#)
 - timeshiftAzEl
 - boa::BoaDataAnalyser::DataAna, [102](#)
 - timeShiftChan
 - boa::BoaDataAnalyser::DataAna, [103](#)
 - timeShiftChanList
 - boa::BoaDataAnalyser::DataAna, [103](#)
 - unflag
 - boa::BoaDataAnalyser::DataAna, [103](#)
 - boa::BoaDataEntity::BolometerArray, [89](#)
 - boa::BoaDataEntity::ScanParameter, [137](#)
 - unflagAccel
 - boa::BoaDataAnalyser::DataAna, [103](#)
 - unflagInTime
 - boa::BoaDataAnalyser::DataAna, [103](#)
 - unflagLon
 - boa::BoaDataAnalyser::DataAna, [104](#)
 - unflagMJD
 - boa::BoaDataAnalyser::DataAna, [104](#)
 - unflagPosition
 - boa::BoaDataAnalyser::DataAna, [104](#)
 - unflagSpeed
 - boa::BoaDataAnalyser::DataAna, [104](#)
 - unflagSubscan
 - boa::BoaDataAnalyser::DataAna, [104](#)
 - unflagTurnaround
 - boa::BoaDataAnalyser::DataAna, [105](#)
 - updateArrayParameters
 - boa::BoaPointing::Point, [130](#)
 - updateRCP
 - boa::BoaDataEntity::BolometerArray, [90](#)
 - wcs2phy
 - boa::BoaMapping::Image, [122](#)
 - wcs2pix
 - boa::BoaMapping::Image, [122](#)
 - writeAsciiRcp
 - boa::BoaDataEntity::BolometerArray, [90](#)
 - writeFFCF
 - boa::BoaDataAnalyser::DataAna, [105](#)
 - writeFits
 - boa::BoaMapping::Image, [122](#)
 - writeModelData
 - boa::BoaPointing::Point, [130](#)
 - writeRCPfile
 - boa::BoaDataEntity::BolometerArray, [90](#)
 - zeroStart
 - boa::BoaDataAnalyser::DataAna, [105](#)
 - zoom
 - boa::BoaMapping::Image, [122](#)
-

boa::BoaMapping::Map, [127](#)
