BoA User Manual : APEX-MPI-MAN-0018

Version: 2.5 (20.03.2007)

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

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**.

The GNU General Public License

Version 2, June 1991

51 Franklin St, Fifth Floor, Boston, MA 02110-1301, USA

PREAMBLE

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   (a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

   (b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

   (c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole

which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

   (a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   (b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   (c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

   The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

   If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing

the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

   If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

   It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

   This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

   Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

Document history

Revision	Date	Author	Sections/Pages affected	Remarks

Internal document history

Revision	Date	Author	Sections/Pages affected	Remarks
v1.9	30.07.06	CV	all	version prepared for official approval
v1.10	01.08.06	CV	all	minor changes only
v1.11	02.08.06	CV	initial pages	Document history, Related documents, and Definintions added
v2.0	08.12.06	CV	all	new version created with different chapter structure plus updates. Significant changes to Chapters 3 (Cookbook) and 4 (Manual).
v2.1	15.12.06	CV	3	recipes for map, point and focus now present.
v2.2	01.03.07	CV	ch4	ch2 (install) also updated by RS
v2.3	08.03.07	CV	ch3 & 4	re-arrangement and updating
v2.4	09.03.07	CV	all	aesthetic changes to all plus updating of Ch4
v2.5	20.03.07	CV	ch4	brought up-to-date and re-organised

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 in press

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 LABOCA wiki: http://www.astro.uni-bonn.de/ abeelen/labocawiki

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

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 μm atmospheric window and is to be commissioned in September 2006. 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 defines 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	
Help	BoaHelp.py	online help
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 (see Sect. **??**)	collection of utilities
BoaConfig.py (see Sect. **??**)	global parameters definitions
BoaSimulation.py	LABOCA data simulator

2. INSTALLATION

This section describes how to install **BoA** and all required additional software packages, as well as how to update an existing **BoA** version.

2.1 Prerequisites

So far, **BoA** has been installed and tested on Mac OS X and on the following LINUX distributions:

- SuSE 10.0
- Scientific Linux 4.2
- Fedora Core 3
- Debian sid

The following software packages must be installed on a system to be able to install and run **BoA**. (The given version numbers indicate the versions that where used during development and tests with the respective LINUX distribution.)

2.1.1 SuSE 10.0

Table 2.1: Prerequsites 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
subversion	1.2.3-2

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

Table 2.2: Prerequsites for Scientific Linux 4.2

Package	Version
gcc / gccc-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
subversion	1.1.4-2

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

Table 2.3: Prerequsites for Fedora Core 3

Package	Version
gcc / gccc-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
subversion	1.1.4-1.1

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.

Table 2.4: Prerequsites for Debian sid

Package	Version
gcc / gccc-c++	4.1.1-11
compat-g77	3.4.6-10
readline-devel	5.1-7
libpng-devel	1.2.8rel-5.2
xorg-x11-devel	1.0.0-8
findutils	4.2.28-1
subversion	

2.1.4 Debian sid

If a package listed in table 2.4 is missing or needs to be updated, use Debian's package manager *apt*.

For any questions related with Debian, contact Alexandre Beelen.

2.1.5 Mac OS X

BoA has also been successfully installed and running on Apple Macintosh computers. Successful installations were done on a G4 laptop (Powerbook G4 550 MHz) and a G5 desktop (Powermac G5 2x1.8 GHz), under Mac OS X 10.3.5 to 10.3.9. BoA is also running on the new Intel Mac (installed and tested on a MacBook Pro 2.0 GHz, with Mac OS X 10.4.6). For any question related to **BoA** on Mac OS, please contact FredericSchuller (schuller@mpifr-bonn.mpg.de).

The prerequisites specified in table 2.5 refer to Mac OS X 10.4.6.

Table 2.5: Prerequsites for Mac OS X

Package	Version
XCode development tools	
python	2.3.5
gcc	4.0.1
gfortran	i386-apple-darwin8.7.1 gcc-4.2-20060805
fink toolkit	
swig	1.3.20
numeric-py23	23.1
readline	
libpng3	

2.2 Conflicts with other software

During the installation of **BoA** described in section 2.3, a set of programs and libraries (called *external packages*) that **BoA** needs is installed. This makes a complete 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 definded 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 successfull. If this is the case, careful inspection of the environment must be performed. (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** run without any further problems.)

2.3 Obtaining the installation script and packages

The command

```
svn checkout boa-install
```

will download the external packages and the installation script *install.sh* (written by Alexandre Beelen, Thomas Jürges, Frederic Schuller, and Reinhold Schaaf) from **BoA**'s Subversion repository to the directory *boa-install* in your current dirctory. The directory *boa-install* will be created if not allready present.

You are now ready to start the installation. (The **BoA** software itself is not downloaded at this stage. It will be downloaded from the Subversion server during the installation.)

2.4 Installation using the *install.sh* script

Please note that the following instructions relate to the installation of **BoA** under Linux. For any question related to **BoA** on Mac OS, please contact FredericSchuller (schuller@mpifr-bonn.mpg.de).

2.4.1 Running the *install.sh* script

Before running the *install.sh* installation script make sure that you have fulfilled the prerequisites described in Sect. 2.1!

1. Go to the directory where you have downloaded the openboa directory and files. Change into the directory *install* where the installation script *install.sh* is stored.

2. Run the *install.sh* script by typing:

   ```
   ./install.sh
   ```

   This begins the process of installing **BoA**.

   The script will prompt you for some paths (reasonable defaults are offered). If you don't want to use the default path, then please enter your chosen path, e.g. */home/smueller/BoA*, when prompted. (Please note that you have to specify the full path name explicitely: `~/BoA` will not work!)

   The script will create in this installation directory some sub-directories, where all necessary files will be installed. The required disk space is about 200 MB.

   You will also be prompted to enter yes (y) or no (n) for the installation of each software package. For a fresh installation, you should install every package included (even if, say Python, is allready present on your system).

   If you want to install all included packages, you can run the installation by

   ```
   ./install.sh -all
   ```

   The install script will prompt you considerably less often.

   Skipping installation of packages is useful if you resume an aborted installation (see below). If you wish to, you can try to see whether **BoA** works with your preinstalled versions of software; however, that is at your own risk!

3. After the installation is complete, type

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

   or

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

4. You can now run **BoA** by typing `boa` at the prompt!

2.4.2 Details of the installation process

The installation consists of three stages, all of which are performed by the installation script:

- Installation of the external packages necessary for **BoA**
- Installation of BoA itself, including documentation and example FITS files
- Installation of BoA's initialization files .boarc.sh and .boarc.csh

Below you find a description of each of these three stages.

After the installation, you will find a installation log in *install/build.stat*. If the installation fails, the install script will tell you that something went wrong and give you a place where you can find information related with the failure. In addition, you can consult *install/build.stat* for information about the earlier steps of the installation.

Installation of external packages

The installation script will install the following external packages:

Table 2.6: External packages

Package	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	

The installation script prompts you for the location of the external packages. The default should allways be correct.

Next, you are prompted for the directory where **BoA** is to be installed. If this directory allready exists, you must confirm that choice. (This case is necessary to resume an aborted installation or to update the **BoA** software itself. In all other cases, install to a new directory!)

The script then installs all external packages into this directory.

For some packages, (e.g. scipy_obsutils) you are prompted whether you want the package to be updated via CVS. This may not be necessary, so you can safely answer n. If you do update, the installation script provides you the necessary information (CVS login and password). Be aware that the CVS server may be slow or even down. If this is the case, you are prompted after a timeout of about 2 min whether you want to proceed without the CVS update. If you are nervous, cancel the installation with `Ctrl-C` and resume the installation (see below).

Installation of BoA

When the installation of the external packages is complete, the **BoA** software itself is installed. Since it is not included in the boa-install download, it is downloaded from the BoA repository now. (Please be aware that you may have to use your own Subversion login and password here!)

The script prompts you for a directory, where **BoA** is to be installed. You can choose any accessible directory.

After the installation of the **BoA** software, the documentation and example FITS files are downloaded from the **BoA** repository and installed. Again you are prompted whether and where you want these features to be installed.

Installation of BoA's initialization files

As last step, the script installs the initialization files .boarc.sh and .boarc.csh to your home directory. These scripts define a runtime environment for **BoA** (setting shell variables, paths, and aliases) for bash (.boarc.sh) and csh (.boarc.csh). Before running **BoA**, type

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

or

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

You may want to add this to your shell's startup script.

2.5 Resuming an incomplete installation

To resume an incomplete installation, run

```
install/install.sh
```

again. When prompted for the directory to which **BoA** is to be installed, specify the same directory as in the aborted installation. (Do this even if you will not install any external packages; the information is needed for the initialization files!)

You can then safely skip all installation steps, that where performed successfully in the last installation run.

Please be aware that you are prompted for the variable `PGPLOT_DIR` after skipping the installation of pgplot. A reasonable default is offered. However, if you want to use a pre-installed pgplot, you can specify this here.

2.6 Installation FAQ

2.6.1 BoA fails to start

- `ImportError: No module named fUtilities`

the fortran modules have not been compiled. Go to the fortran directory and type `make`

- `ImportError: libifport.so.x`

you dont have the fortran librarie in you `$LD_LIBRARY_PATH`, please source the `boarc.xx` file or check your installation.

2.6.2 I can't open a Graphical Device

- check the pgplot and p_pgplot installation
- if trying to output png files, make sure that libpng was present when compiling pgplot.

2.6.3 Reading a MBFits file fails

- check the cfitisio and pcfitsio installation
- check that the version of `MBFits.xml/$MBFITSXML` you are using match the file you are trying to read

2.7 Updating BoA

Depending on the changes in **BoA** that make an update necessary (or desirable), an update of **BoA** only or a complete update (external packages and **BoA**) may be necessary. Unfortunately, presently there is no systematic way to find out whether a update of the external packages is necessary. The best choice may be first to try an update of **BoA** only, and if you experience problems when running **BoA** after that update, make a complete update of the external packages and **BoA**.

Updating BoAonly

Examine the shell variable BOA_HOME_BOA that is set in the initialization files `~/.boarc.sh` and `~/.boarc.csh`, to find out, where **BoA** is installed. Move to this dirctory:

```
cd $BOA_HOME_BOA
```

Update **BoA** from the **BoA** repository by typing

```
svn update
```

Change to the `fortran` subdirectory and compile **BoA**'s Fortran modules:

```
cd fortran
make clean all
```

Updating external packages and BoA

If you experience problems when running **BoA** after updating **BoA** only, update the external packages as well. For that, follow the instructions in section 2.3 to obtain a new installation script and the external packages from **BoA**'s Subversion repository. Then follow section 2.5 to replace the external packages of your current installation that need to be updated. Do not forget to update **BoA** itself as described in the preceding section.

If this does not result in a working installation, do a fresh installation according to section 2.4, possibly into a new directory.

3. BoACookbook

This section describes how to start up **BoA** for the first time and describes some typical **BoA** sessions, including making a map and solving a pointing and focus. These example sessions are intended to act as recipes to allow the beginner or occasional user to get on air quickly. Users already familiar with the content of this cookbook can find more detailed information in Chapter 4.

3.1 Starting up BoA

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*.

In certain circumstances, more advanced users may wish to invoke **BoA** via a Python session. This can be done in the following ways:

- start a Python session and at the Python prompt type

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

- call Python in interactive mode (*-i*) with the file *BoaStart.py*

  ```
  python -i /home/user/boa/BoaStart.py
  ```

  where */home/user/* is the user path to the *boa* directory.

3.2 Getting started

```
open()                      % 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.

3.3 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*.

3.4 Getting Help

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

```
print command.__doc__
```

at the prompt.

3.5 Example of making a map

```
open()                                  % 1
indir('/home/user/data/')               % 2
ils()                                   % 3
proj('T-77.F-0002-2006')                % 4
read('59491')                           % 5
signal()                                % 6
signal(1)                               % 7
mapping()                               % 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=5)                                                               % 23
mapping()                                                                    % 24
doMap(system='EQ',sizeX=[83.9,83.73],sizeY=[-5.48,-5.28],oversamp=5.)        % 25
smooth(6./3600.)                                                             % 26
mapdisp(caption=data.ScanParam.caption())                                    % 27
close()                                                                      % 28
```

3.5.1 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*.

3.5.2 Visusalising 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.

3.5.3 Basic Processing and Analysis

Usually the processing of the data will begin with subtracting a 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 rcp 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 median noise value. 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 root of the rms of that channel (command 22). Command 23 then unflags the previously flagged source position.

There are then a number of ways to produce a final map. Command 24 produces a quick map in Az-El coordinates. Alternatively command 25 produces a map in EQ coordinates. See Chapter 4 for optional arguments for these and other methods. The map may then be smoothed (command 26) and this smoothed map displayed (command 27).

3.6 Example of solving a pointing

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

The above recipe 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 Example of solving a focus

```
open()                          % 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 recipe 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 Demonstration scripts

The recipes described in Sections 3.5, 3.6 and 3.7 form the basis of three example scripts which are provided in order to demonstrate some of the basic functionalities of **BoA**. These are: *ExampleMap.py*, *ExamplePointing.py* and *ExampleFocus.py* and can be found in the directory */home/user/boa/examples/*. Run the scripts by typing:

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

4. BoAUser 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 4.18.

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 **??**). 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 that *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 4.18.

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)

```
open() or  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))
```

Even if the list contains only one element the square brackets are mandatory:

```
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 **??**.

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 arguement 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

Several methods are provided for constructing a map.

4.3.1 Making a map

METHOD: `doMap` (*optional arguments*)

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

OPTIONAL ARGUMENTS:

chanList	channels to consider
flag	flag values to consider
oversamp	oversampling factor (beam fwhm / pixel size). Default=2.
beammap	compute a beam map (default: no)
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)
sizeX	limits in Az of the map
sizeY	limits in El of the map
noNan	remove NaN in self.Results?
style	color table to use in image
smooth	do we smooth with beam? (default: no)
noPlot	do not plot the map? (default: no)
caption	plot caption
aspect	keep aspect ratio? (default: yes)
showRms	compute and print rms/beam? (default: yes)
rmsKappa	kappa in kappa-sigma clipping used to compute rms

4.3.2 Making an Az-El map

METHOD: `horizontalMap` (*optional arguments*)

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

OPTIONAL ARGUMENTS:

chanList	channels to consider
flag	flag values to consider
oversamp	oversampling factor (beam fwhm / pixel size). Default=2.
sizeX	limits in Az of the map
sizeY	limits in El of the map
noNan	remove NaN in self.Results?
style	color table to use in image
noPlot	do not plot the map? (default: no)
caption	plot caption

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:

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

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:

chanList	list of channels to be flagged (default: current list)
flag	flag value

4.4.3 Flagging data by time interval

METHOD: `flagMJD` (*optional arguments*)

DESCRIPTION: flag data by MJD interval

OPTIONAL ARGUMENTS:

channel	list of channels to flag (default: 'all')
below	flag data below this value (default end of the scan)
above	flag data above this value (default start of the scan)
flag	flag value to set (default 1)

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:

chanList	list of channels to flag (default: 'all')
Az/El	the horizontal reference position (arcsec for offsets, deg for absolute)
radius	aperture to flag in unit of the reference position
flag	flag to be set (default 5)
offset	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:

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

4.4.6 Flagging subscans

METHOD: `flagSubscan` (*optional arguments*)

DESCRIPTION:flag a list of subscans

OPTIONAL ARGUMENTS:

subList	list of subscan numbers (or single number) to be flagged
flag	flag value to be set

4.4.7 Unflagging data

METHOD: `unflag` (*optional arguments*)

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

OPTIONAL ARGUMENTS:

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

4.4.8 Unflagging a list of channels

METHOD: `unflagChannels` (*optional arguments*)

DESCRIPTION: Unflag a list of channels

OPTIONAL ARGUMENTS:

channel	list of channels to be unflagged (default: current list)

4.5 Baseline subtraction, sky removal and statistics

4.5.1 Computing the Rms in a map

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

DESCRIPTION: compute rms/beam in a map

OPTIONAL ARGUMENTS:

cell size of one beam in pixel
rmsKappa for kappa-sigma clipping before computing rms

4.5.2 Computing weights

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

DESCRIPTION: compute weights and store them in DataWeights attribute

OPTIONAL ARGUMENTS:

method type of weighting (default='rms', i.e. use 1/rms^2)

4.5.3 Median baseline removal

METHOD: `medianBaseline` (*optional arguments*)

DESCRIPTION: remove median value per channel and per subscan

OPTIONAL ARGUMENTS:

chanList list of channels to process (default: all; [] : current list)
subscan compute baseline per subscan (default: yes)

4.5.4 Noise removal

METHOD: `medianNoiseRemoval` (*optional arguments*)

DESCRIPTION: remove median noise from the data.

OPTIONAL ARGUMENTS:

chanList list of channel to flag (default: all; [] : current list)
chanRef reference channel number (default: RefChannel)
computeFF compute skynoise FF (default) or use existing FF_Median?

4.5.5 Polynomial baseline removal

METHOD: `polynomialBaseline` (*optional arguments*)

DESCRIPTION: perform polynomial baseline removal on the data

OPTIONAL ARGUMENTS:

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

4.5.6 Smoothing an image

METHOD: `smoothBy` (*optional arguments*)

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

OPTIONAL ARGUMENTS:

beamSize the FWHM of the smoothing gaussian

4.5.7 Obtaining the statistics

METHOD: `statistics`

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

4.6 Pointing

4.6.1 Solving a pointing

METHOD: `solvePointingOnMap` (*optional arguments*)

DESCRIPTION: compute the offset on the data.Map object

OPTIONAL ARGUMENTS:

gradient	shall we fit a gradient ? (default: no)
circular	fit a cricular gaussian instead of an elliptical gaussian
radius	use only bolo inside this radius (negative means multiple of beam) (default: 10 beams)
Xpos	source position if using fixed position
Ypos	source position if using fixed position
fixedPos	if set, don't fit position, but use Xpos, Ypos
plot	do we plot the results? (default: no)
display	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.

4.7 Focus

The recommended way to conduct Laboca focus observations is to perform a series of n*3 short, symmetric on-offs, e.g. 3 or 6*(4*5sec). For this simply the onoff has to be reduced and then the results can be fitted by a parabola.

4.7.1 Solving a focus

METHOD: `solveFocus`

DESCRIPTION: compute the optimal focus position

4.8 File reading

4.8.1 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')
```

4.9 Controlling graphics display devices

In order to display your data in various ways using the **BoA** plotting methods described in Section 4.10 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 4.10 are thus described in this section.

4.9.1 Opening a plot window

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

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

The default is to open an XWindow. You can use

```
open('?')
```

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

```
open('/ps')
```

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

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

4.9.2 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.

4.9.3 Closing a plot window

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

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

4.9.4 Selecting an open device

METHOD: `selectDev`

DESCRIPTION: select an open device

4.10 Plotting and displaying data

4.10.1 Displaying channel maps

METHOD: `chanMap` (*optional argument*)

DESCRIPTION: plot channel maps

OPTIONAL ARGUMENTS:

chanList	list of channels, of the form [1,2,3]
flag	flag values to consider
oversamp	oversampling factor (beam fwhm / pixel size). Default=2.
center	0/1 if set to 1 it will shift each map by the bolometer offset
	from the fits header. Thereby it shifts the source to the center of each channel map
showRms	compute and print rms/beam? (default: no)
rmsKappa	kappa in kappa-sigma clipping used to compute rms

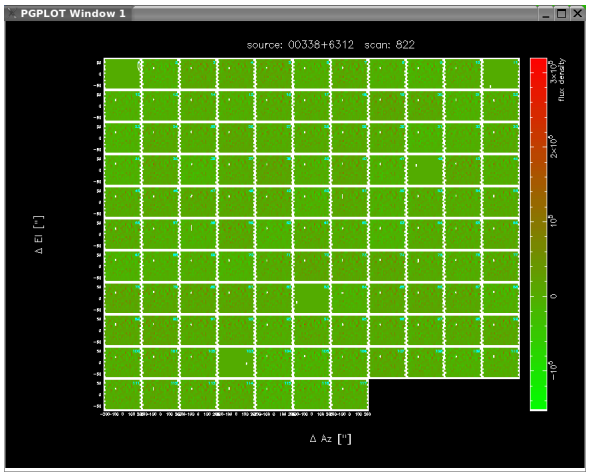

Figure 4.10.1: Default graphical outputs of a channel map of the source 00388+6312, including a wedge.

4.10.2 Displaying/re-displaying a map

METHOD: `display` (*optional arguments*)

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

OPTIONAL ARGUMENTS:

weight	plot the weight map instead of signal map
coverage	plot the rms map instead of signal map
style	the style used for the color (default idl4)
caption	the caption of the plot (default ”)
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)
limitsZ	range of Z values to be plotted (comma separated values, in square brackets)
wedge	draw a wedge ? (default : yes)
aspect	keep the aspect ratio (default : yes)
overplot	should we overplot this image (default : no)
doContour	draw contour instead of map (default : no)
levels	the levels of the contours (default : intensity progression)
labelContour	label the contour (default : no)

Example:

Re-display a map after performing a smoothing (as in example in Section 3.5).

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

4.10.3 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)`

4.10.4 Plotting elevation versus azimuth

METHOD: `plotAzEl` (*optional arguments*)

DESCRIPTION: Plot elevation versus azimuth.

OPTIONAL ARGUMENTS:

flag	flag to be used (default = 0: all valid data; -1: plot all)
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 the plotting related arguments can be found in Section 5.5.

Example:

as for `plotAzimuth()`.

4.10.5 Plotting azimuth and elevation acceleration

METHOD: `plotAzElAcceleration` (*optional arguments*)

DESCRIPTION: Plot azimuth and elevation acceleration.

OPTIONAL ARGUMENTS:

flag	flag to be used (default = 0: all valid data; -1: plot all)
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 the plotting related arguments can be found in Section 5.5.

Example:

as for `plotAzimuth()`.

4.10.6 Plotting elevation offset versus azimuth offset

METHOD: `plotAzElOffset` (*optional arguments*)

DESCRIPTION: Plot elevation offset versus azimuth offset.

OPTIONAL ARGUMENTS:

flag	flag to be used (default = 0: all valid data; -1: plot all)
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 the plotting related arguments can be found in Section 5.5.

Example:

as for `plotAzimuth()`.

4.10.7 Plotting azimuth and elevation speed

METHOD: `plotAzElSpeed` (*optional arguments*)

DESCRIPTION: Plot azimuth and elevation speed.

OPTIONAL ARGUMENTS:

flag	flag to be used (default = 0: all valid data; -1: plot all)
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 the plotting related arguments can be found in Section 5.5.

Example:

as for `plotAzimuth()`.

4.10.8 Plotting azimuth versus LST

METHOD: `plotAzimuth` (*optional arguments*)

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

OPTIONAL ARGUMENTS:

flag	flag to be used (default = 0: all valid data; -1: plot all)
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 **??**). Show individual plotted points (rather than lines), make plotted points red, and only plot azimuth (y axis) from -14 to -13 degrees.

4.10.9 Plotting azimuth offset versus LST

METHOD: `plotAzimuthOffset` (*optional arguments*)

DESCRIPTION: Plot azimuth offset versus LST.

OPTIONAL ARGUMENTS:

flag	flag to be used (default = 0: all valid data; -1: plot all)
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 the plotting related arguments can be found in Section 5.5.

Example:

as for `plotAzimuth()`.

4.10.10 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:

chanRef	reference channel number (default : is the first in chanList)
chanList	list of channels, of the form [1,2,3]
flag	flag to be used
skynoise	plot against the skynoise of chanRef (default : no)
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	

4.10.11 Plotting elevation versus LST

METHOD: `plotElevation` (*optional arguments*)

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

OPTIONAL ARGUMENTS:

flag	flag to be used (default = 0: all valid data; -1: plot all)
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:

as for `plotAzimuth()`.

4.10.12 Plotting elevation offset versus LST

METHOD: `plotElevationOffset` (*optional arguments*)

DESCRIPTION: Plot elevation offset versus LST.

OPTIONAL ARGUMENTS:

flag	flag to be used (default = 0: all valid data; -1: plot all)
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 the plotting related arguments can be found in Section 5.5.

Example:

as for `plotAzimuth()`.

4.10.13 Plotting the FFT of the signal

METHOD: `plotFFT` (*optional arguments*)

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

OPTIONAL ARGUMENTS:

chanList	list of channels, of the form [1,2,3]
flag	flag to be used
labelX	the X label; default is *Frequency [Hz]*
labelY	the Y label; default is *Amplitude (a.b.u/sqrt(Hz)*
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)
plotphase	plot phase instead of amplitude (default no)
ci	colour index to be used (integer values)
overplot	

Example:

Plot FFT for the first 9 channels.

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

4.10.14 Plot mean flux versus subscan number

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

DESCRIPTION: plot mean flux value vs. subscan number

OPTIONAL ARGUMENTS:

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

4.10.15 Plot mean channel values versus channel number

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

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

OPTIONAL ARGUMENTS:

chanList	list of channels
flag	flag to be used
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	

4.10.16 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?

4.10.17 Plotting RMS versus channel number

METHOD: `plotRmsChan` (*optional arguments*)

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

OPTIONAL ARGUMENTS:

chanList	list of channels
flag	flag to be used
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	
subscan	if 0, plot rms of the complete scan (default); if 1, plot for each subscan and each channel

4.10.18 Display start and end times of subscans

METHOD: `plotSubscan()`

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

4.10.19 Plot subscans on the Az-El pattern

METHOD: `plotSubscanOffsets()`

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

4.10.20 Show a reconstructed map in (Az,El) or (Ra,Dec)

4.10.21 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:

chanList	list of channels, of the form [1,2,3]
flag	flag to be used
mjd	if set, use mjd instead of lst
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	

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)
```

4.11 Data handling

4.11.1 Get a list of valid channels

METHOD: `checkChanList` (*optional argument*)

DESCRIPTION: Return a list of valid channels

OPTIONAL ARGUMENTS:

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
flag	flag value to be selected (default = 0)

4.11.2 Get pixel values

METHOD: `getPixel()`

DESCRIPTION: get pixel values using mouse

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

4.11.3 Print the current list of channels

METHOD: `printChannels()`

DESCRIPTION: Print the current list of channels

4.11.4 Selecting channels

METHOD: `setCurrChanList` (*optional argument*)

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

OPTIONAL ARGUMENTS:

chanList:	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('?')
```

4.12 MB-Fits to FITS file conversion

To convert an MB-Fits file to a FITS file in the MAMBO format you can use the command `mambo()`. The current version does NOT use the data contained in the data object in **BoA**, but reads the input file (with default name = BoaB.currData.FileName) and converts it to the Mambo format. Therefore, this procedure is somewhat decoupled from **BoA**.

4.13 User methods for selecting files and directories

4.13.1 Listing the contents of the input directory

METHOD: `listInDir()`

DESCRIPTION: list the contents of the input directory

4.13.2 Resetting the CurrentList

METHOD: `resetCurrentList()`

DESCRIPTION: reset the CurrentList to the complete List

4.13.3 Setting the input directory

METHOD: `setInDir()`

DESCRIPTION: set the input directory

Example:

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

4.13.4 Setting the output directory

METHOD: `setOutDir()`

DESCRIPTION: set the input directory

Example:

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

4.13.5 Setting the project ID

METHOD: `setProjectID()`

DESCRIPTION: set the current project ID

Example:

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

4.14 Miscellaneous methods

4.14.1 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

4.15 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 succesively, 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')
```

4.15.1 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')
```

4.16 Commands in alphabetical order

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/rms^2)
despike	Flag yet unflagged data below 'below'*rms and above 'above'*rms
display	show the reconstructed maps in (Az,El)
doMap	construct a map in (Az,El) coordinates
flagChannels	flag a list of channels
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
flagSubscan	flag certain subscans
getPixel	get pixel values using mouse
horizontalMap	construct a map in (Az,El) coordinates
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 subscans
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 gived FWHM
solveFocus	compute the optimal focus position
solvePointingOnMap	compute the offset on the data.Map object
statistics	prints the statistics
unflag	unflag data
unflagChannels	unflag a list of channels
updateRCP	update offsets and gains from the content of a file

4.17 Commands in functional order

4.17.1 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 subscans
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)

4.17.2 Device handling

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

4.17.3 Pointing and focus

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

4.17.4 Flagging and despiking data

despike	flag yet unflagged data below 'below'*rms and above 'above'*rms
flagChannels	flag a list of channels
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
flagSubscan	flag certain subscans
unflag	unflag data
unflagChannels	unflag a list of channels

4.17.5 Map making

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

4.17.6 Baseline subtraction, sky removal and statistics

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

4.17.7 File handling

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

4.17.8 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

4.17.9 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

4.17.10 Misc.

updateRCP	update offsets and gains from the content of a file

4.18 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 `open()`. For further convenience, most user methods can also be called by even shorter abbreviations of the user methods (in this example `op()` is all that is needed). A list of user methods and their abbreviations is given in Table 4.1.

Command	Abbreviations
chanMap	ChanMap - chanmap
checkChanList	checkChannels - checkChan
clear	cle - cl
closeDev	close - clo - cls
computeWeight	computeweight - weight
despike	dspike
display	mapdisp - mapdisplay
doMap	domap
findInDir	find - fd
flag	
flagChannels	flagCh - flagC - fCh
flagLon	
flagMJD	
flagPosition	flagPos
flagRms	
flagSubscan	flagSub
getPixel	getPix
listInDir	ils - inls
mapping	horizontalMap - hrMap
maprms	computeRms
medianBaseline	medianBase - medianbase
medianNoiseRemoval	mediannoise
openDev	open - op
plotArray	plotarray
plotAzEl	azel
plotAzElAcceleration	azelaccel - azelac
plotAzElOffset	azeloff - azelo
plotAzElSpeed	azelspeed - azelsp
plotAzimuth	azimuth - azim - az
plotAzimuthOffset	azimuthOffset - azimoff - azo
plotCorrel	plotcorrel - plotcor - plotCor
plotElevation	elevation - elev - el
plotElevationOffset	elevationOffset - eleoff - elo

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
plotFFT	
plotMean	plotmean
plotMeanChan	plotmeanchan
plotRms	plotrms
plotRmsChan	plotrmschan
plotSubscan	plotSub
plotSubscanOffsets	plotSubOff
polynomialBaseline	baseline - base
printCurrChanList	printChannels - printChan
read	
resetCurrentList	resetCurrList - rls
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	

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 pgplot, 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, Simba and Mambo.

5.2 Command handling

BoGLi has its own command handler. Nevertheless, anytime the **BoA** command handler encounters a graphic command this is automatically passed to the BoGLi command handler. Therefore, the user does not have to care about the separation between **BoA** and **BoGLi** commands. Table 5.1 gives an overview of some of the available commands.

BoGLi provides a variety of attributes that may be changed by the user. The attribute name is then used as command followed by the desired value as argument (see Sect. **??** for details.)

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

`DeviceHandler.openDev`	open a device
`DeviceHandler.closeDev`	close a device
`Plot.clear`	clear the active plot window
`DeviceHandler.selectDev`	select a device
`DeviceHandler.resizeDev`	resize the plotting area, after plot window resized using mouse
`Plot.plot`	make a single plot
`MultiPlot.plot`	plot multiple plots
`Plot.draw`	draw on an image
`MultiPlot.draw`	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. A version of *drivers.list* used for a Linux PC can be found in Sect **??**.

The command handler of **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 types 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:

`DeviceHandler.closeDev(2)`	Close the device with identifier 2
`DeviceHandler.closeDev('all')`	close all devices
`DeviceHandler.closeDev('current')`	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:

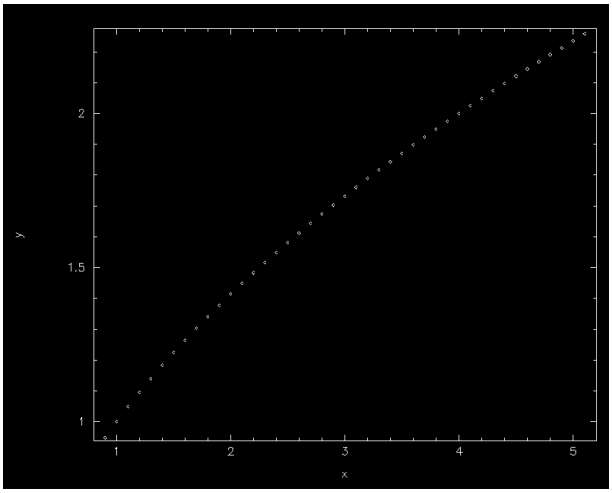

Figure 5.4.1: Example 1 of graphics produced using Plot.plot

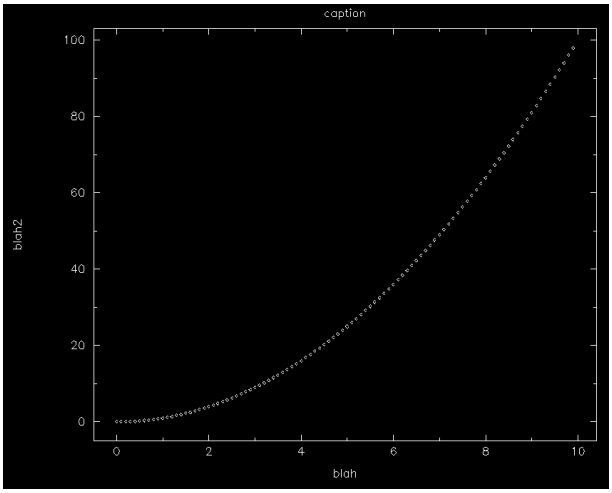

Figure 5.4.2: Example 2 of graphics produced using Plot.plot

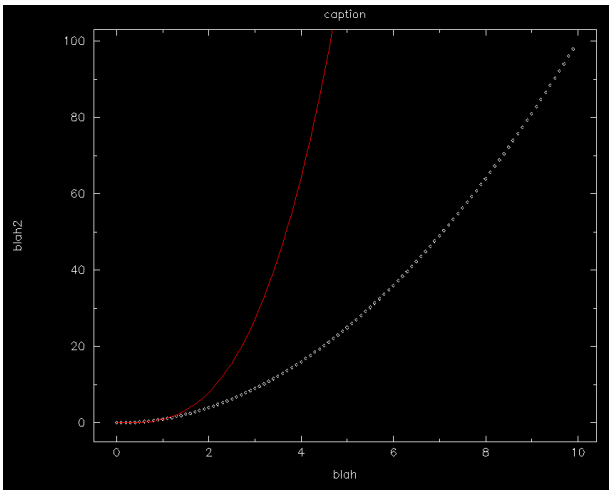

Figure 5.4.3: Example 3 of graphics produced using Plot.plot

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)
width	linewidth (defaut 0 = use previous)
aspect	keep the aspect ratio in 'physical' unit
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 **??**. 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:

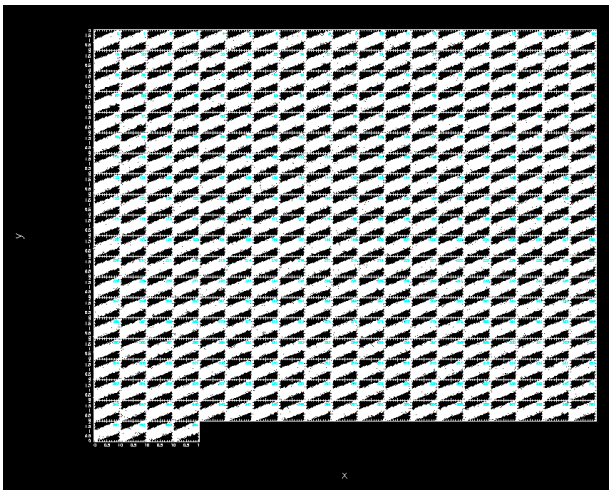

Figure 5.4.4: Example of graphics produced using MultiPlot.plot

```
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 channelssimultaneously.

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

ARGUMENTS:

chanList	list of channels, of the form [1,2,3]
dataX	values to plot along X
dataY	values to plot along Y

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 **??**.

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:

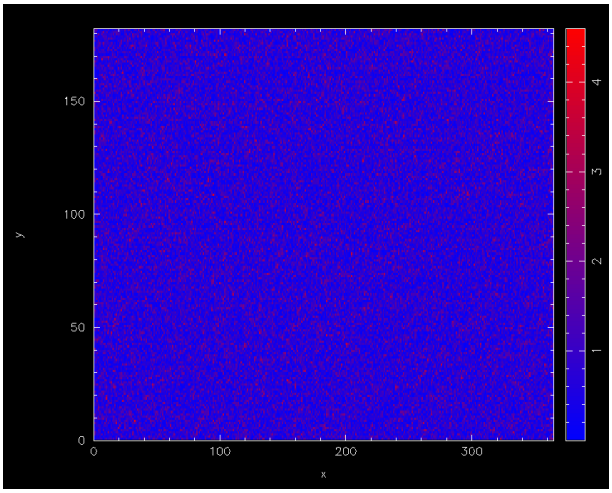

Figure 5.4.5: Example 1 of graphics produced using Plot.draw

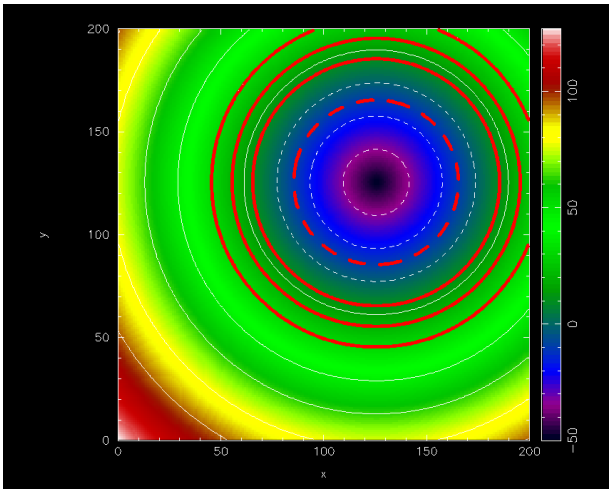

Figure 5.4.6: Example 2 of graphics produced using Plot.draw: drawing contours

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

These arguments are also described in Section **??**.

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=
```

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
```

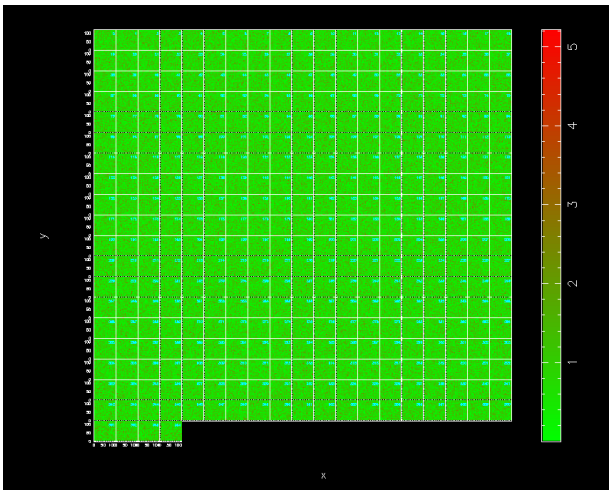

Figure 5.4.7: Example of graphics produced using MultiPlot.draw

```
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.

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.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 **??**.

Example:

```
mapping_array = []
n_map = 365
for i in range(n_map):
    mapping_array.append(Numeric.absolute(RandomArray.standard_normal([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 *linestyle*

('l': line, 'p': point (default), 'b': histogram)

width *linewidth*

(defaut 0 = use previous)

aspect *aspect ratio*

keep the aspect ratio in 'physical' unit

overplot *allow/prohibit overplotting*

set overplot=1 to overplot (default no)

logX *logarithmic scale*

set logX=1 to use a log scale (default no)

logY *logarithmic scale*

set logY=1 to use a log scale (default no)

sizeX *set the 'physical' size of the array*

the 'physical' size of the array (default pixel numbers), defined by the center of the two extreme pixels

sizeY *set the 'physical' size of the array*

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 *draw contours*

set =1 to draw contour instead of map (default no)

levels *set the levels for the contours*

the levels for the contours (default nContour, within plotLimitsZ)

labelContour *label the contours*

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 **??**). 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	Frontent-Backend name
EffectiveFrequency	float	Observing frequency, in Hz
BeamSize	int	Beam size, in arcsec
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

Note: for observations performed with wobbler switching, pairs of ON–OFF integrations are extracted from the Wobbler_Sta attribute, and the phase differences are computed. By default, after reading, only the differentiated signals are stored in the Data attribute. The user can specify the phase number in the read command, in order to get only the 'ON' or the 'OFF' data.

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.

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

Table 6.4: Other attributes of a DataEntity object

Name	Type	Description
FileName	string	Input file name
RefGain	float	Frontend gain/attenuation factor
JyPerCount	float	Counts to Jy conversion factor
Data	float array	Current version of the data
DataBackup	float array	Previous 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	Currrent default flag value when calling a flagging routine
flagValueList	int list	Allowed values for flagging

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:

```
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 that the default BoaData.sav. Then to reload the data object, one has to do:

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

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 (Chapter ...) instead of the abbreviations defined in BoaShortcuts.py.

6.3 Data output

6.3.1 Converting the raw data

BoAprovides a procedure to convert an MB-FITS file to a FITS file with the same format as for MAMBO-ABBA data. The aim of this procedure is to be able to compare the results of a data reduction performed with **BoA**with those obtained with existing packages (e.g. NIC, MOPSI). *Note: This procedure has not been extensively tested recently...*

6.3.2 Saving a map

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) infromations. This is done with the following command:

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

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

7. DEVELOPMENT

7.1 Basic programming rules

7.2 Adding classes

7.3 Adding methods

7.4 Adding Fortran90 code

FB040510

General

We are using Fortran 90/95 subroutines, wrapped to be called from python using the f2py package. This is because f90 code executes much faster than python scripts. There are some subtelties to pay attention to when wrapping fortran code, else you will add large overheads from the py-f90 interface, as arrays are copied and reindexed. For an introduction to F90/95 (only minor differences between the two), I recommend the compact and rather comprehensive (and free!) "Fortran 90 course notes"[1] by AC Marshall from the University of Liverpool. It contains all you probably need to know. I wrote a simple fortran method in BoA/fortran/BoaTest1.f90 to illustrate some basic features and give you a chance to test the wrapper without **BoA**. Look at its header for details. For an online F90/95 language reference[2] the best I found is at the NCSA resources page, describing IBM's XL Fortran for AIX 8.1 – which is close to the Intel compiler.

F90 in BoA

For BoA our general idea is to have one f90.so extension module, which includes all the f90 methods (called subroutines and functions in fortran). This is necessitated by that the f90.data module, which contains much of a scans data, is connected (through an "use data") to the other f90 program modules, and therefore they all need to be linked together.

The f90 methods may be split into different modules (classes) for convenience. We now have the first operational modules BoaF1.f90, BoaChannelAnalyser.f90, BoaBaseLine.f90, and the data module BoaData.f90. Each module may include any number of subroutines or functions. The data module BoaData.f90 is like a common block that contains all the data which does not change during data reduction. All data which does change is passed to the fortran subroutines as call arguments.

[1] http://math.nist.gov/W̃Mitchell/f90course/CourseNotes.pdf

[2] http://www.ncsa.uiuc.edu/UserInfo/Resources/Hardware/IBMp690/IBM/usr/share/man/info/en_US/xlf/html/lr02.HTM#CONTENT

The BoaData.f90 (f90.data from python) module is filled in BoaDataEntity.FillF90. It must be refilled if you change data object, else the fortran methods will work on a different scan. This re-filling must be implemented still. Currently the f90.data is only filled upon read of a new data file.

The CVS directory BoA/fortran contains the fortran source code. You will need to wrap/compile the **BoA** modules on your local system (see below), since it links to local libraries that have no standard address. This will create the extension module f90.so which you import to BoA. From the CVS directory BoA start **BoA**, then

```
>>> from fortran import f90
```

This is how to import any module from a subdirectory, which for this needs to include an empty file __init__.py

The python script fortran/ftest.py contains a series of calls to the fortran subroutines. To run it:

```
>>> read()  # read in some scan
>>> op()    # open plot device
    [enter]
>>> execfile('ftest.py')  # start the script
```

which is followed with lots of output. To illustrate the use of new python methods that use fortran, you find BoA/TestFB.py, which you run like ftest.py. It goes through a number of data reduction steps and plots the data.

Wrapping F90 code with f2py

To wrap the f90 modules to produce f90.so:

```
ifc -c -w svd.f90
f2py -c -m f90 BoaData.f90 BoaF1.f90 BoaChannelAnalyser.f90 BoaBaseLine.f90 s
     or on some installations alternatively:
f2py -c --fcompiler=intel -m f90 BoaData.f90 BoaF1.f90 BoaChannelAnalyser.f90
```

The first command recompiles the svd.o. On the f2py line there are some diagnostic options you may add if you debug your code:

```
-DF2PY_REPORT_ATEXIT : gives time statistics upon exit from python.
-DF2PY_REPORT_ON_ARRAY_COPY=1000 : reports when the f2py interface copies an
-DNUMARRAY : must be used for numarray support. Default is Numeric.
```

If the wrapping fails, one of the following may be wrong:

1. You have not initiated the ifc compiler properly. In your shell initialization file (e.g. .cshrc for tcsh) you need

```
if (-e /opt/intel/compiler60/ia32/bin/ifcvars.sh) then
  source /opt/intel/compiler60/ia32/bin/ifcvars.csh
endif
```

or something equivalent.

2. Your python path does not include the intel fortran compiler:

```
setenv PYTHONPATH ".:/opt/intel/compiler60/ia32/lib/:
                    /usr/local/lib/python2.3:
                    /usr/local/lib/python2.3/site-packages:
                    /home/bertoldi/bin:
                    /opt:
                    /usr/lib"
```

3. You use an old version of f2py.

```
<fortran> f2py -version
2.39.235_1644
```

Once you have successfully imported f90 in BoA, you can inquire about the use of a given method by typing

```
print f90.f1.NAME.__doc__
```

Fortran attributes are called f90.data.name_of_attribute. To inquire which ones are available:

```
boa> print f90.data.__doc__
el - 'f'-array(218)
track_el - 'f'-array(218)
ffcf_gain - 'f'-array(120)
subscan_time - 'f'-array(4)
az_p - 'f'-array(109,3)
lst - 'f'-array(218)
lon_p - 'f'-array(109,3)
track_az - 'f'-array(218)
lat - 'f'-array(218)
az - 'f'-array(218)
lat_p - 'f'-array(109,3)
lst_p - 'f'-array(109,3)
array_gain - 'f'-array(120)
lon - 'f'-array(218)
ffcf_cn - 'f'-array(120)
ut_p - 'f'-array(109,3)
nodding_sta - 'i'-array(218)
subscan_index - 'i'-array(4)
subscan_num - 'i'-array(4)
weights - 'f'-array(0), not allocated
el_p - 'f'-array(109,3)
ut - 'f'-array(218)
wobbler_pos - 'f'-array(218)
```

They are filled in in BoaBusiness.py: BoaB.FillF90

Use f90 methods in BoA

To call a fortran method, here an example:

```
compressed_array,nmax = f90.f1.compress(array,flag_array,0)
```

Two objects are returned as a tuple, an array and an integer. They both are not in the call argument list, they are hidden to python, but are listed in the f90 code call argument list – have a look at the source code.

Limitations

This particular example illustrates one of the limitations of wrapping f90 code: you cannot return an array with a length that is determined upon execution. The wrapper needs to specify the size of an array somehow. It does not have to be fixed, but specified through the size of an input attribute at least. In this example we try to return an array that is a compression of the input array, determined by the condition that the corresponding flag is 0. The trick to still do this here is to return a comressed_array with the same size as array, plus an integer telling the size of the compressed array, so that the final answer is compressed_array[0:nmax].

Fortran vs. C-contiguous

If a Numeric array is proper-contiguous and has a proper type then it is directly passed to the wrapped Fortran function. Otherwise, an element-wise copy of an input array is made and the copy, being proper-contiguous and with proper type, is used as an array argument. There are two types of proper-contiguous Numeric arrays: Fortran-contiguous arrays when data is stored column-wise, i.e. indexing of data as stored in memory starts from the lowest dimension; C-contiguous when data is stored row-wise, i.e. indexing of data as stored in memory starts from the highest dimension. For one-dimensional arrays these notions coincide. To transform input arrays to column major storage order before passing them to Fortran routines, one may use the function as_column_major_storage(<array>) that is provided by all F2PY generated extension modules, such as the BoA f90. If you call a fortran method repeatedly with the same input array, you should convert the array first to avoid conversion by the wrapper interface on each call – which could dominate the execution time here. If you add the option -DF2PY_REPORT_ON_ARRAY_COPY=1000 when wrapping, you will be informed on each copy that the wrapper interface performs. The option -DF2PY_REPORT_ATEXIT gives an execution time summary upon exit that splits up the time used in fortran and in the interface. If the interface time is large or comparable to the fortran execution time, your code is not efficient because it copies arrays too often. Look at examples in BoaBaseLine.py, e.g.:

```
Data = f90.as_column_major_storage(self.Data.Data_Red_p)
Flag = f90.as_column_major_storage(self.Data.Data_Flag_p)
...
for i_ch in ch_range: # loop over channels and phases
    for i_ph in ph_range:
      Data = f90.baseline.addpoly(Data,Poly,Mean,Rms,i_ph,i_ch)
```

The input arrays are copied once into fortran-contiguous arrays before the loop, so in the loop there is no overhead from copying. Note also the general scheme of calling a fortran method here: Data is in- and output argument.

7.5 Interfacing

7.5.1 ScientificPython-2.4.5

ScientificPython is a collection of Python modules that are useful for scientific computing. Almost all modules make extensive use of Numerical Python (NumPy,Numeric), which must be installed prior to Scientific Python. Scientific constist of about one dozen modules, which contain methods written in Python that may come handy, but may be slow. The following lists a number of them.

stat() statistics() command calculates the statistics for all the channels in the range. Using plotmean() plotrms() we can plot mean and RMS values of each channels. The examples are as shows below:

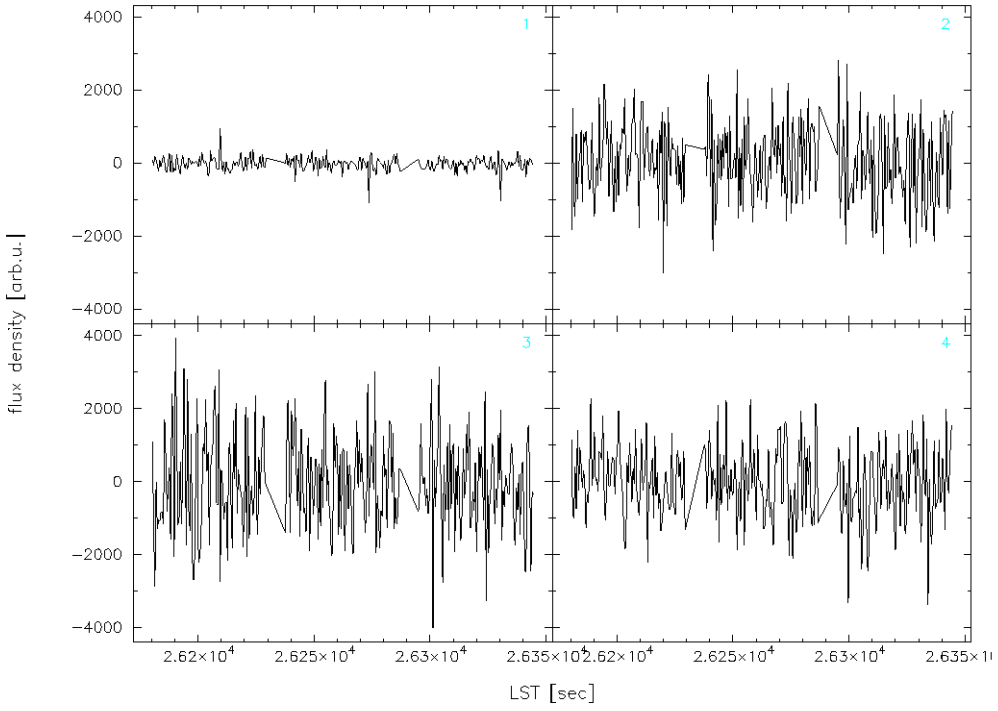

Figure 7.5.1: Plotting the Signal for channels in the range.

You need to import Numeric for Scientific. You can access the methods by importing the class or all methods:

```
>>> from Numeric import *
>>> import Scientific.Statistics
>>> Scientific.Statistics.median([1,2,3,5,6])
3.0
```

or alternatively

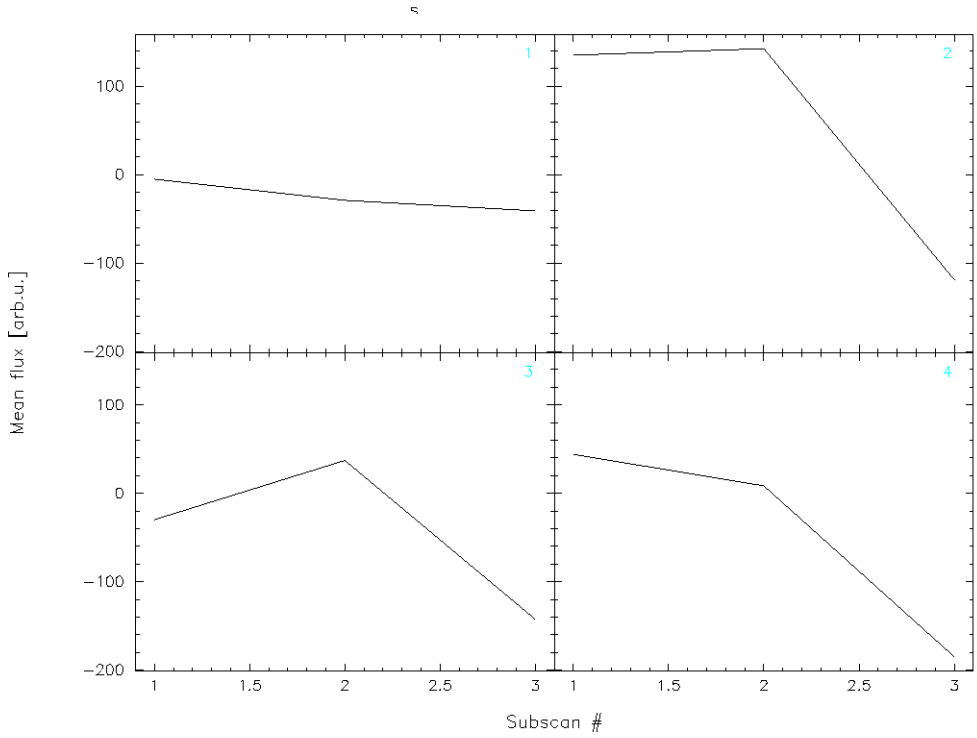

Figure 7.5.2: Plotting the Mean values of signal.

```
>>> from Scientific.Statistics import *
>>> median([1,2,3,5,6])
3.0
```

Available method in class Scientific.Statistics:

```
moment(data, order, about=None, theoretical=1)
mean(data)
weightedMean(data, sigma)
variance(data)
standardDeviation(data)
median(data)
mode(data)
normalizedMoment(data, order)
skewness(data)
kurtosis(data)
correlation(data1, data2)
```

There are also two classes for histograms:

```
Histogram
```

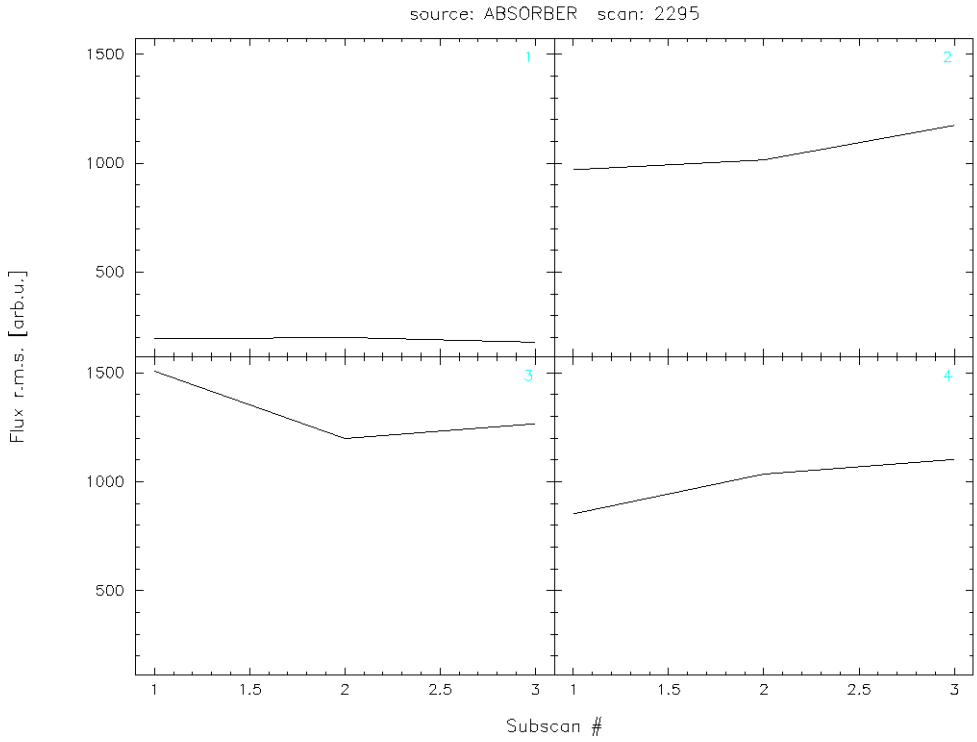

Figure 7.5.3: Plotting the RMS values of signal.

```
WeightedHistogram(Histogram)
```

The following explains only those Scientific methods which are useful for Boa. Consult the scripts or the (very sparse) documentation for more info.

Scientific.Statistics.median

Description: Computes the median of a 1-d array.

Example:

```
>>> median([1,2,3,5,6])
3.0
```

Scientific.Statistics.mean

Description: Returns the mean (average value) of a 1-d array.

Example:

```
>>> mean([1,2,3,5,6])
3.3999999999999999
```

Scientific.Statistics.correlation

Description: Computes the correlation coefficient between two 1-dim arrays a and b according to

$$c_{ab} = \frac{\langle (a-\bar{a})(b-\bar{b})\rangle}{\langle (a-\bar{a})^2\rangle^{1/2}\langle (b-\bar{b})^2\rangle^{1/2}} \tag{7.5.1}$$

Example:

```
>>> correlation([1,2,3,4,5],[1,2,3,4,5])
1.0
>>> correlation([1,2,3,4,5],[1,2,3,5,5])
0.96476382123773219
>>> correlation([1,2,3,4,5],[5,4,3,2,1])
-1.0
```

Scientific.Functions.LeastSquares

Description: General non-linear least-squares fit using the Levenberg-Marquardt algorithm and automatic derivatives. The parameter model specifies the function to be fitted. It will be called with two parameters: the first is a tuple containing all fit parameters, and the second is the first element of a data point (see below). The return value must be a number. Since automatic differentiation is used to obtain the derivatives with respect to the parameters, the function may only use the mathematical functions known to the module FirstDerivatives. The parameter parameter is a tuple of initial values for the fit parameters. The parameter data is a list of data points to which the model is to be fitted. Each data point is a tuple of length two or three. Its first element specifies the independent variables of the model. It is passed to the model function as its first parameter, but not used in any other way. The second element of each data point tuple is the number that the return value of the model function is supposed to match as well as possible. The third element (which defaults to 1.) is the statistical variance of the data point, i.e. the inverse of its statistical weight in the fitting procedure. The function returns a list containing the optimal parameter values and the chi-squared value describing the quality of the fit.

Example:

```
>>> from Numeric import exp
>>> def f(param, t):
...     return param[0]*exp(-param[1]/t)
...
>>> data = [(100, 4.999e-8),(200, 5.307e+2),
  (300, 1.289e+6),(400, 6.559e+7)]
```

```
>>> print leastSquaresFit(f, (1e13,4700), data)
([8641551709749.7666, 4715.4677901570467], 1080.2526437958597)
```