# The LOFAR Imaging Cookbook: Manual data reduction with the imaging pipeline

Written by Timothy Garn (and updated by Roberto Francesco Pizzo, with contributions from Vishambhar Nath Pandey, Evert Rol, Anna Scaife, and John Swinbank, on behalf of the LOFAR commissioning teams)*

Version 5.1 + – January 14, 2011

This cookbook describes the process of manually reducing a Measurement Set with the LOFAR imaging pipeline. It is intended to speed up the learning process for future commissioning, by collating various tips, tricks, and solutions in a single place. The LOFAR wiki[1] contains much more information on each stage of data reduction, but might be out of date in many places. The LOFAR forum[2] should also be helpful for commissioning. The contents of this cookbook are an approximation to the correct way of reducing LOFAR data – use with caution.
The softwares that have been designed for LOFAR data reduction are still in development. Sometimes, quicker results might be obtained with other data reduction packages (such as CASA). However, to test and improve the quality of the new software, we strongly encourage the users to follow the proposed way of the cookbook, post results or problems in the LOFAR forum, and talk to the software developers.

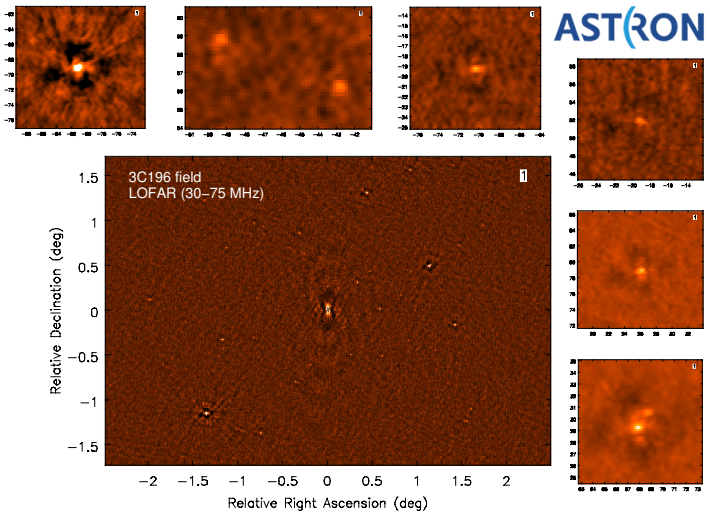



Figure 1: You too can make images like this with LOFAR

*for any suggestions and comments, please contact Roberto Francesco Pizzo, pizzo@astron.nl
[1] http://www.lofar.org/operations/doku.php?id=software:standard_imaging_pipeline
[2] http://usg.lofar.org/forum/

# Changes

The latest released version of the cookbook is available at the web address:

`http://www.astron.nl/radio-observatory/lofar/lofar-imaging-cookbook`

This link is advertised in both the LOFAR forum and the LOFAR wiki. The very latest (development) version of the cookbook can also be found on the USG repository: `http://usg.lofar.org/svn/documents/trunk/Tutorials/Imaging/`. The LOFAR software is continuously improving and, as a consequence, several procedures (and the cookbook itself) continuously change. In the following, we report an overview of the (recent) changes applied to the manual.

## Overview of changes

Most recent changes

2011-01-14 Updated 'Source detection' section (Sect. 9).

2010-12-10 Introduced description of the source detection algoritm PyBDSM (Sect. 9).
Introduced section explaining the gains trnsfer from a calibrator to the target (Sect. 6.6).
Changed download location for Lofar Imaging Cookbook (see above).
Included more scripts in the list of scripts made by the commissioners and useful during the Lofar data reduction. Updated the location where these scripts can be retrieved (Sect. 13.2).
How to use the indirect-read option in RFIconsole now explained in Sect. 5.2.

2010-10-10 Introduced description of the AOFlagger (Sect. 5).
Introduced description of phase-only calibration in BBS (Sect. 6.4).
Introduced a section presenting practical examples of data reduction (Sect. 8).
Described source subtraction in BBS with an example (Sect. 6.9).
Introduced image showing the incidence of RFI at LOFAR frequencies (Fig. 7).

2010-07-22 Major update of BBS: change in the specification of the data selection in the parset files. The new parset files are now available in $\sim$`/pizzo/EXAMPLES/Parset`.
Removal of the 0.5 factor in the definition of the total intensity. No longer inconsistency between BBS and CASA.

New syntax for baseline/antenna selection now available in NDPPP and BBS (see Sect. 4.3).

A new GUI interface is now available in uvplot.py (Sect. 2.3).

All the sections of the manual have been improved, taking into account the suggestions from the users.

2010-07-01 Introduced discussion about spectral line imaging in BBS section

Introduced discussion about flagging of the data after the calibration at the beginning of the BBS section

Inserted comments about what of the imaging pipeline is fully operational and what is still under development.

2010-05-25 Removed appendix with the description of IDPPP. NDPPP is the only flagging/averaging package available now.

2010-05-04 Added a sub section describing how to set default parameters in BBS (Sect. 6.10).

2010-04-21 Added section describing the imaging pipeline. There is now a paragraph on GNU screen in the “Imaging pipeline” section, pointing to the appendix.

Previous changes

- A section describing the outline of the imaging pipeline has been included (Sect. 3).
- Section 6 went through several changes regarding parameters which were out of date in the reported parset files or wrong commands. A new section (Sect. 6.5.1) has been inserted. It concerns a python script which can be used to inspect the gain solutions.
- The section describing the available techniques to inspect the MeasurementSets has been extended including the tool uvplot.py (created by George Heald), which might be useful to rapidly plot the visibilities (Sect. 2.3).

# Contents

# 1 Getting Started

## 1.1 The LOFAR cluster layout

The correlated data coming from the BlueGene/P (BG/P)[3] are stored on a cluster of machines, the so called *offline cluster*. In here the data go through the pipelines, which will perform their standard reduction and calibration. After this pipeline processing, the results are stored in the LOFAR Export (staging) Archive (`LEXAR`).
The LOFAR cluster is divided into 8 subclusters, which are accessible through a frontend (see below). Each subcluster is a processing cluster for a specific commissioning group. Each subcluster has 9 compute nodes (named `lcexxx`) and 3 storage nodes (named `lsexxx`) of 24 TB raw capacity. In total, there are 72 compute nodes and 24 storage nodes. The storage nodes have 4 partitions of 5.1 TB each. Each partition holds a single XFS filesystem. Each filesystem is NFS mounted on all 9 compute nodes in the subcluster (but NOT in other subclusters).
In the following, a few more details of the offline cluster components are reported.

- **Frontend:** it has 2 Intel Xeon L5420 quad core processors, 16 GB internal memory, 2 GbE interfaces and 2 TB disks. There are two identical frontends: `lfe001` and `lfe002`. Both of them serve a specific group of subclusters. The frontends are used to build the software and regulate the workload on the subclusters.
- **Processing units:** the 72 compute elements have 2 Intel Xeon L5420 quad core processors, 16 GB internal memory, 2 GbE interfaces and 1 TB disks. They can be accessed by secure shell and they are grouped.
- **Storage units:** the 24 storage nodes are HP DL180G5 boxes, having 2 Intel Xeon L5420 quad core processors, 16 GB internal memory, 6 GbE network interfaces and a total of 24 TB disk space. The disk space is divided into 4 partitions of 6 disks each. The partitions are called `/data1 - /data4`. They have a XFS filesystem.

## 1.2 Logging on to the offline processing cluster

- Begin by logging on to `portal.lofar.eu`[4]:

  ```
  > ssh -Y <user name>@portal.lofar.eu
  ```

  where `<user name>` is likely to be the user's surname. Type in your default password[5]:

  ```
  > "password"
  ```

  Throughout this cookbook, > is used to indicate a new input line.

  If this is the first time you will be logging onto the cluster, you need to change your password first on the portal:

  ```
  > yppasswd
  ```

[3] It is located in Groningen, The Netherlands.

[4] The actual host name is lfw.lofar.eu (lfw=LOFAR firewall), but this alias will work fine.

[5] Your default password will be communicated to you at the moment of the creation of your Lofar account by Teun Grit, `grit@astron.nl`.

in the usual fashion (old default password, new password, confirm new password). Then you can log in to one of the front end nodes (you may have to log out of and log in again to the portal first).

- Log on to your chosen front end node on lfe001 or lfe002:

```
> ssh -Y lfe001 or lfe002
```

We will choose lfe001 as the default front end node in the rest of the Cookbook.
From here it is possible to show all the available subclusters and their nodes by using the command `showsub`. An example of the output is given below:

```
> showsub
This script shows the subcluster definitions

sub   lce-nodes      lse-nodes      cexec-lce  cexec-lse  group     contact
====  ============   =============  =========  =========  ========  ======
sub1  lce001-lce009  lse001-lse003  lce:0-8    lse:0-2    product.  observers
sub2  lce010-lce018  lse004-lse006  lce:9-17   lse:3-5    TBB       ?
sub3  lce019-lce027  lse007-lse009  lce:18-26  lse:6-8    imaging   Swinbank
sub4  lce028-lce036  lse010-lse012  lce:27-35  lse:9-11   polariz.  de Bruyn
sub5  lce037-lce045  lse013-lse015  lce:36-44  lse:12-14  pulsar    Hessels
sub6  lce046-lce054  lse016-lse018  lce:45-53  lse:15-17  busyweek5 Heald
sub7  lce055-lce063  lse019-lse021  lce:54-62  lse:18-20  develop.  Romein
sub8  lce064-lce072  lse022-lse024  lce:63-71  lse:21-23  busyweek5 Heald
```

To execute a single command on multiple nodes you can use: ”cexec sub<n>: uptime”, ”cexec lse: uptime” or ”cexec lce: uptime”. E.g., show all users on the first 4 nodes of subcluster 3:

```
> cexec sub3:0-4 users
************************ sub3 ************************
--------- lce019---------
heald rol
--------- lce020---------
rol
--------- lce021---------
rol
--------- lce022---------
rafferty rafferty rol
--------- lce023---------
asgekar asgekar
```

Note that you can execute the `cexec` command only for the subcluster(s) you have access to. By default, you will have access to the only subcluster assigned to your LOFAR KSP (see the `group` description obtained through the `showsub` command).
The very first time you run the `cexec` command on a subcluster, you may need to log in to each of the nodes separately to store your SSH key (see Sect 1.3.2). However, you can avoid this by using the procedure described in Sect. 1.3.3.

- Log on to (for example) the imaging subcluster (`sub3`) on one of the lce nodes ranging between `019` and `027`[6]

[6]Note that the subcluster number and the range of lce-nodes might change with the commissioning stage.

```
> ssh -Y <user name>@lce019
```

You can find more information on the cluster here:

`http://www.lofar.org/operations/doku.php?id=public:lofar_cluster`

Note that the `/home/<user name>` directory is visible from any node, but `/data/scratch/<user name>` is machine-specific.

- Best is to use the TCSH shell for your work, since the Cookbook provides little information yet on how to use the bash shell for LOFAR data reduction:

  ```
  > tcsh
  ```

  But see the occasional notes throughout the cookbook for BASH addicts.

Processing can now take place.

## 1.3 Setting up your working environment

### 1.3.1 Login scripts

Log in to the front end cluster and from your `$HOME` directory:

Under the (t)csh shell, you do as follows:

- Make sure to delete any potential .cshrc file on your home directory (check with ls -a)
- `> ln -s /opt/login/cshrc .cshrc`
- Log out and login again; you should see a welcome message.

For BASH, make sure your .bashrc is as clean as possible, that means not cluttered with variables (especially LOFARROOT, LD_LIBRARY_PATH & PYTHONPATH should not have -too many- default settings); although this probably applies to (t)csh as well.

To use a certain package, you can make use of the "`use`" alias, which for BASH is defined as `alias use='source /opt/login/loadpackage.bash'`. To initialize the LOFAR imaging pipeline software, type

```
> use LofIm
```

This software is newly built every day. Sometimes[7] you need a specific "build of the day". For that, use

```
> use LofIm <day>
```

where `<day>` is the week day, i.e.`Sun` or `Mon` or `Tue` etc.

To initialize the CASA software, which could be useful for particular steps of the data reduction, you can type

[7] The build of the current day could be unsuccessful

```
> use Casa
```

For further processing, it is also important to initialize the python packages. Type:

```
> use Pythonlibs
```

Similarly, the Karma software (useful during imaging) and the Pydal libraries (needed to run a few python scripts mentioned in the manual - see Section 13.2) could also be initialized:

```
> use Karma
> use LUS
```

More information can be found on the LOFAR wiki [8].

### 1.3.2 Generation of SSH keys

We use the Secure Shell (SSH) on the LOFAR Central Processing (CEP) to connect to different systems. This page explains how this can be used without having to supply a password each time you want to connect to a system (very useful to run things such as BBS on nodes of a cluster, or other remote machines). With normal SSH you always have to give a password. If you use a private and public key, you can access systems where your public key is in `$HOME/.ssh/authorized_keys` from the system where you have the private key.

- For Linux or OS X:

  From the front end node `lfe001` set up passwordless access to the imaging subcluster `lce` nodes via SSH:

  ```
  > ssh-keygen -tdsa
  ```

  Press enter and again when prompted for a password.

  ```
  > cp ~/.ssh/id_dsa.pub ~/.ssh/authorized_keys
  ```

  `$HOME/.ssh/authorized_keys` must be available on all the machines you need to access with this key; since your home directory is automatically mounted on all the cluster nodes, they should already be accessible—you can copy it to other, external, systems if required. Every time you will be logged into `lfe001`, you will be required to type your password just once for the entire session if you will type:

  ```
  > ‘ssh-agent -c -k‘
  >  ssh-add
  ```

  After pressing enter, you will be asked to type in your password for the current session.

- For Windows and additional information, refer to the LOFAR wiki [9].

[8] `http://www.lofar.org/operations/doku.php?id=public:lle`
[9] `http://www.lofar.org/wiki/doku.php?id=public:ssh-usage`

### 1.3.3 Disable SSH Host Key Checking

Normally, when you first connect to a new host, SSH will prompt you for confirmation of the host key:

```
> ssh lce019
The authenticity of host 'lce019 (10.176.0.19)' can't be established.
RSA key fingerprint is 73:27:96:cd:f5:04:b7:c3:57:47:49:97:8b:87:8b:15.
Are you sure you want to continue connecting (yes/no)?
```

When trying to run a command over many compute (and/or storage) nodes using multiple SSH connections, that can get pretty annoying. To get around it, set `StrictHostKeyChecking` to `no` in `$HOME/.ssh/config`:

```
> cat > ~/.ssh/config
StrictHostKeyChecking no
```

## 1.4 Setting up a personal database for BBS

A personal database is needed on the front end node in order to use BBS on a `lce` node. This need only be done once. The following instructions are taken from the LOFAR wiki[10]:

Log in to `lfe001`, then go through the following steps (where > is the shell prompt):

```
> createdb -U postgres -h <hostname> <databasename>
> createlang -U postgres -h <hostname> plpgsql <databasename>
```

where `<hostname>` is generally `ldb001` and the `databasename` should be your `username`.

Now, download `bbs-sql.tgz`[11], and extract it:

```
> tar xvfz bbs-sql.tgz
or
> tar xvf bbs-sql.tar
> cd bbs-sql
> psql -h <hostname> -U postgres -d <databasename>
```

Inside the sql prompt:

```
> \i create_blackboard.sql
```

You can ignore the warnings that appear on the screen.

To verify that everything went fine, list the schemas that have been created:

```
> \dn
```

[10] `http://www.lofar.org/operations/doku.php?id=engineering:software:tools:bbs`

[11] It can be copied from ~`pizzo/EXAMPLES/Files` or downloaded from the wiki at `http://www.lofar.org/wiki/lib/exe/fetch.php?media=engineering:software:tools:bbs:bbs-sql.tgz` (which requires a wiki username and password).

You should see something like this:

```
        List of schemas
        Name        |  Owner
--------------------+----------
 blackboard         | postgres
 information_schema | postgres
 pg_catalog         | postgres
 pg_toast           | postgres
 pg_toast_temp_1    | postgres
 public             | postgres
(6 rows)
```

Finally, to exit the sql prompt, use:

```
> \q
```

### 1.4.1 Common errors

If you get this error :

```
> <username>@lceXXX: ~\$ createdb -U postgres -h ldb001 <username>
 perl: warning: Setting locale failed.
 perl: warning: Please check that your locale settings:
         LANGUAGE = (unset),
         LC_ALL = (unset),
         LC_PAPER = "nl_NL.UTF-8",
         LC_MONETARY = "nl_NL.UTF-8",
         LANG = "en_US.UTF-8"
      are supported and installed on your system.
 perl: warning: Falling back to the standard locale ("C").
 createdb: database creation failed: ERROR:  database <username> already
 exists
```

then add at the end of your `.bashrc` file (located in your $HOME directory):

```
export LANG=en_US.UTF-8
        export LC_COLLATE=POSIX
        export LC_TIME=POSIX
```

## 1.5 Setting up a working space

As mentioned in Sect. 1.1, the LOFAR data is held on the lse nodes. Each users group has a specific subcluster to work on. For example, the imaging group have storage nodes `lse007, lse008,` and `lse009`. These areas are not used for processing data. For this, you should use one of the `lce` nodes. Depending on the stage of the commissioning, the available computing elements nodes might

change. To inspect which of them you can use, type `showsub` whilst logged into lfe001, as described in Sect. 1.2.

Once you have logged onto a computing node, you should make your own working directory using

```
> mkdir /data/scratch/<username>
```

You can now `cd` into it and use it as your working space.

The data from the storage nodes can be copied to your working area using, for example,

```
> scp -r <user name>@lse007:/data1/<LOFAR dataset>/<subband> .
```

where `<LOFAR dataset>` will look something like `L2010_XXXXX`[12] and `<subband>` is something like `SBXXX.MS`[13].

[12] `"L"` stays for LOFAR, `"2010"` for the year in which the observation was performed, and `XXXXX` is the ID number of the observation, which is assigned to it at the moment of the scheduling.

[13] `"XXX"` is the subband number.

# 2 Data Inspection

Data inspection is essential and can be carried out using either scripts (by means of a python interface to the Measurement Set) or CASA[14].

In this cookbook an example LOFAR subband, SB0.MS at 30MHz of 3C196, observed in September 2009 with 1 core station, 4 remote stations and Effelsberg will be used to illustrate the various stages of the pipeline[15]. Polarisation is XX unless otherwise stated.

## 2.1 Viewing Measurement Set details

The script `msinfo.py` allows you to view details of the contents of a Measurement Set. You can run it by typing:

```
> ~rol/sw/bin/msinfo -V <file name>.MS
```

Previous versions of the cookbook had this and following paths as `/app/scripts/doUSG`. On the new cluster, all `/app` directories have been changed into `/opt`.

An example of the output of the script is given below.

```
        Summary of UV data for /data1/L2009_14319/SB0.MS

  Phase center:            [[ 2.15377266  0.84155074]]    # RA X DEC and exp-
                                                            ressed in radians
  Frequency range (MHz):   30.37  --  30.57
  Wavelength range (m):    9.81  --  9.87
  Time range (MJs):        4759437600.00  --  4759484399.37
  Duration (hrs):          13.00
  # of integrations:       1
  time bin / integration:  1.0
  # of channels:           256
  channel width (KHz):     0.8
  # of polarizations:      4

Antennas & their GPS positions (longitude, latitude):
RS106LBA:    6.985083   52.688684
RS208LBA:    6.918982   52.483598
CS302LBA:    6.849252   52.716224
RS307LBA:    6.680948   52.617727
RS503LBA:    6.850858   52.759804
DE601LBA:    6.883957   50.334350
```

Together with providing important information on the observation details, this script will be useful to test, at later stages, whether the compression in time or frequency of the data has been successful.

A few important caveats when using `msinfo.py` are:

[14] Documentation available at the web address http://casa.nrao.edu/

[15] Some of the figures in this cookbook have been created with scripts written by L. Ker.

- if you split the dataset in sub-time ranges (see Section 4.5.2), `msinfo.py` does not return the corrected timerange, but still returns the whole timerange from the initial dataset. To obtain the right information about the timerange, it is recommended to examine the TIME column in the MS file;
- sometimes the script returns numbers for the duration of the observation which are too large. This bug has been reported.

## 2.2 Pyrap / PyDAL scripts[16]

Pyrap is a python interface to the casacore library, which allows the raw data tables (Measurement Sets) to be manipulated and the data plotted via python scripts (e.g. Figs. 2 and 3). These allow you to customize what is plotted, and can be significantly faster than CASA for plotting large datasets.

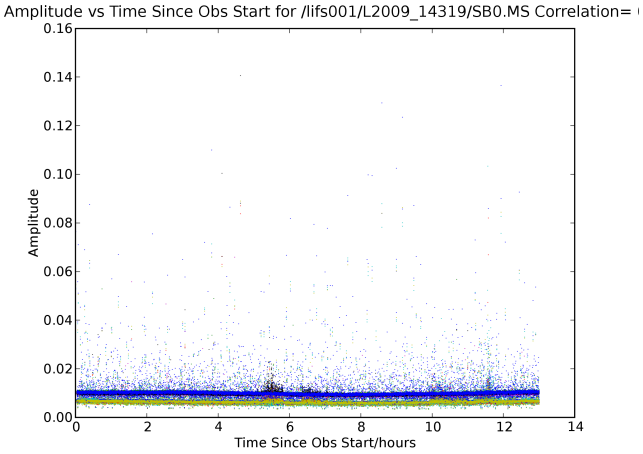



Figure 2: Plotted here using a pyrap script is the amplitude vs time for the SB0.MS 3C196 observation. The high level of RFI is instantly apparent.

Visualizing the data before running the pipeline is essential to pick up any hardware/observation errors. For example, observed data in the past has had gaps present due to correlator errors etc.

## 2.3 Quick baseline-based visibility inspection

Visibilities can be plotted relatively rapidly by making use of the combination of pyrap and the plotting package PGPLOT, both of which work quickly. A script is available which plots visibility data from all baselines in a Measurement Set[17]. It plots either amplitude or phase against time, frequency, or channel number.

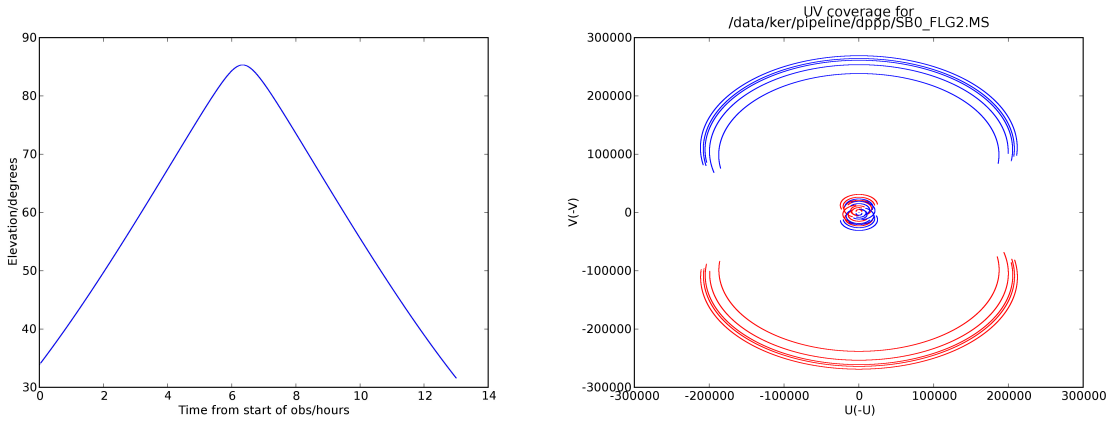
To use it, you should first prepare the environment by typing

```
> source /home/heald/login/usePgplot
```

The script itself is `/home/heald/bin/uvplot.py`, and it will list its (many) options if you use the `-h` flag.

[16] An extensive description of the Pyrap utilities is available at the web address `http://www.astron.nl/casacore/trunk/pyrap/docs/`

[17] `uvplot.py` can be used to plot also the raw visibilities. CASA fails doing that.




(a) Elevation vs. time (b) *uv* coverage

Figure 3: Two examples of useful data plotted with pyrap scripts, for the SB0.MS 3C196 observation.

```
> /home/heald/bin/uvplot.py -h
uvplot.py v1.6, 14 April 2010
Usage:  uvplot.py [options]
Options:
-h, --help show this help message and exit
-i INMS, --inms=INMS Input MS to plot [no default]
...
```

One particularly useful option is the `-q` flag. It will give a short listing of crucial information about the Measurement Set specified with `-i`:

```
/home/heald/bin/uvplot.py -i /net/sub3/lse007/data4/L2009\_15697/SBXXX.MS -q
```

This feature will, in particular, report how many timeslots are in the Measurement Set. If this number is larger than a few thousand, you should consider only plotting small timeranges at a time (for speed and clarity of the plots).
An example of output plots is provided in Fig. 4

To simplify the process of specifying plot options, you can optionally use the GUI interface to the plotting program (see Fig. 5). If you pass the `--gui` flag to `uvplot.py`, then a graphical window will appear where you can edit the plot settings. Any other options specified on the command line will show up in the GUI window. Once the settings are specified as you like, click the green "Plot" button at the bottom left. Once the plotting is finished, you can change the options in the interface and plot again. Quit the program with the red button at the bottom right of the interface.

**Troubleshooting**. If you receive the following error

```
RuntimeError: Table DataManager error: Data Manager class LofarStMan is
not registered
```

then probably the daily build is broken and you should revert to a previous build with "`use LofIm <day>`".

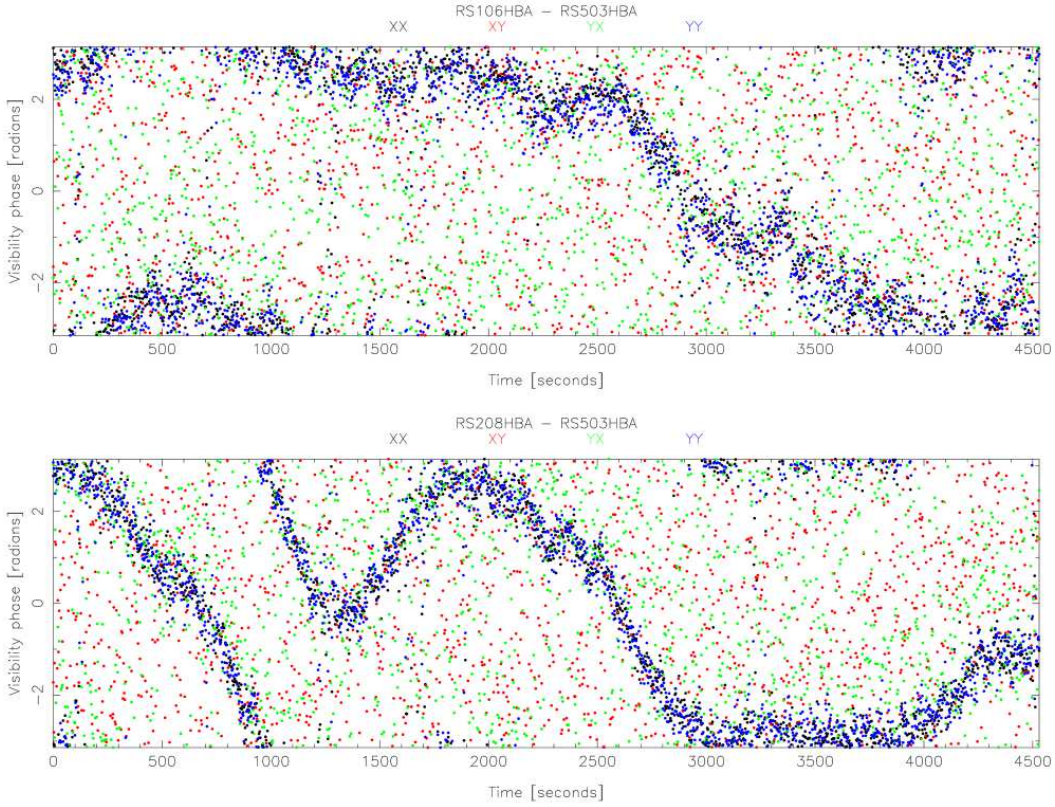



Figure 4: An example of using `uvplot.py`. The command used to generate this plot was `/home/heald/bin/uvplot.py -i /net/sub3/lse007/data4/L2009_15697/SB64.MS -t 0,1500 -y phase -n 1,2 -x time -d output.ps/cps` . The point size has been increased afterward using `/home/heald/bin/embiggen.csh`.

## 2.4 CASA

CASA[18] is the python-based next generation replacement for AIPS/AIPS++. It is however a beta version, and caution must be exercised in its use, as it is still lacking in functionality and has bugs present.
CASA can be used to display the data[19]. Be aware that trying to inspect the raw visibilities with CASA will produce a "`segmentation fault`" error. To avoid this, you should make a copy of the dataset with NDPPP (see the first example parset in Sect. 4.2).

### 2.4.1 Casaviewer

`Casaviewer` can be used to plot the visibilities and look at an image (Sect. 7.6). Once you are sure that you have properly initialized Casa (Sect. 1.3.1), to invoke `casaviewer` just type:

```
> casaviewer
```

[18] `http://casa.nrao.edu/`

[19] Be warned: attempting to display uncompressed data will take a long time (about 20 mins, for a single subband of a 13 hour dataset), while doing this on compressed, single channel data may not show all of the RFI.

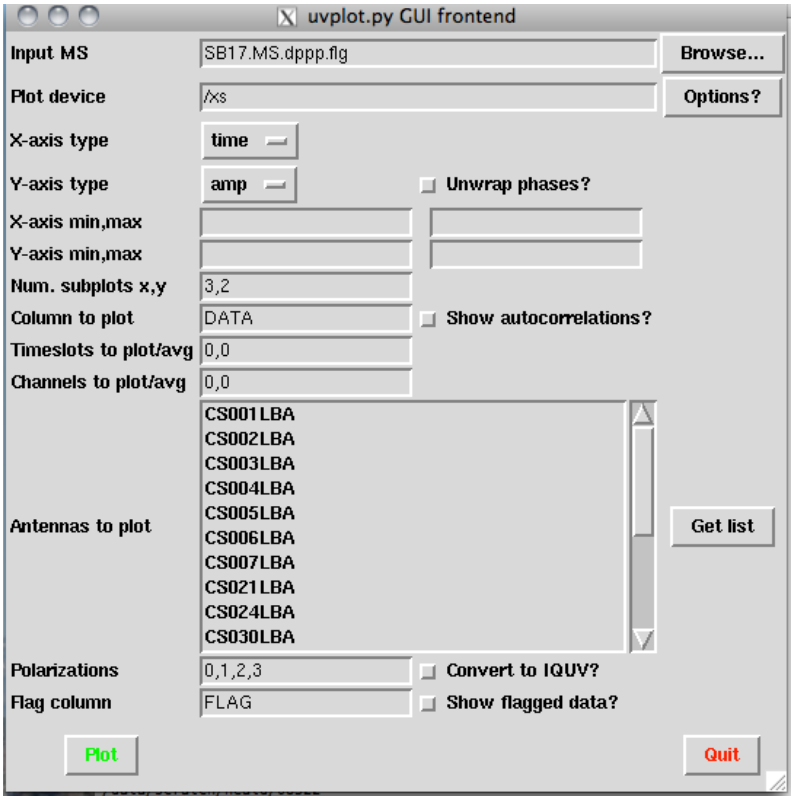



Figure 5: The GUI interface to the plotting program `uvplot.py`.

### 2.4.2 CASA table viewer

To inspect the content of a Measurement Set, use the `casabrowser`:

```
> casabrowser
```

This is useful to understand how the data are contained in the Measurement Set. It is also essential when you suspect that a MS is corrupted. Things to check include whether the time ranges are sensible, and the interval values are consistent.

### 2.4.3 Plotting with CASA

An alternative way of viewing the XX and YY polarisations, plotting time against amplitude, for each set of baselines in turn, is given below.

```
> casapy
```

Inside CASA:

```
> task = 'plotxy'
> vis = '<DPPP output filename>.MS'
> selectdata = True
> correlation = 'XX,YY'
> iteration = 'baseline'
> subplot = 221
```

```
> inp plotxy
> go plotxy
```

When not present, `plotxy` creates a scratch data column in a MS file. This process takes a considerable amount of time and it increases the size of the inspected dataset. To avoid this problem and to be able to inspect the data more quickly, you can use the CASA task `plotms`, which can also be called from outside CASA by typing

```
> casaplotms
```

### 2.4.4 CASA tips

CASA works in a similar way to AIPS[20]. To view the current settings of a task, type `inp <task name>`. To reset all settings, type `default <task name>`. To reload settings from the last run of a task, type `tget <task name>`.

### 2.4.5 CASA bugs

- Trying to inspect the raw data with `casaviewer` or `plotxy` is impossible. It is advised to first `touch` the raw data through NDPPP (see Sect. 4), making a copy of it (see Sect. 4.2).
- A nasty bug exists in the SPLIT task in CASA, whereby attempting to average a Measurement Set in both time and frequency gives a corrupted output Measurement Set. If using the SPLIT task, do not apply any averaging, as further processing with the pipeline will not be possible.
- Many of the plotting options advertised in the help files do not exist yet - another good reason to become familiar with Pyrap!

[20] `http://aips.nrao.edu/`

# 3 Imaging pipeline

While the cookbook deals with all the aspects of the LOFAR image data reduction step by step, eventually the data reduction of LOFAR observations will become automatic and will be performed through the imaging pipeline (see Fig. 6). This contains the various steps taken by the data, from their correlation till the images production.

The data streams from the stations are correlated on the Blue Gene / P (BG/P; OLAP) and then temporarily stored on the Storage Cluster. The Default Pre-Processing Pipeline (DPPP, now known as NDPPP - see Sect. 4) reads the data from Storage and flags and compresses it. The compressed data is again stored and sent to the Compute Cluster for Calibration (BlackBoard Selfcal, Sect 6) and Imaging (CImager, Sect. 7). BBS takes an initial sky model (Local / Global Sky Model; LSM/GSM) and default calibration parameters as input. The updated sky model and calibration parameters are stored. An (corrected) image cube is then constructed by the CImager. This cube is used for finding sources and stored for further astronomical use. Multiple Self-cal calibration and imaging iterations may be performed; these are called Major Cycle loops. The whole pipeline will be controlled by the Monitoring and Control (MAC) software.

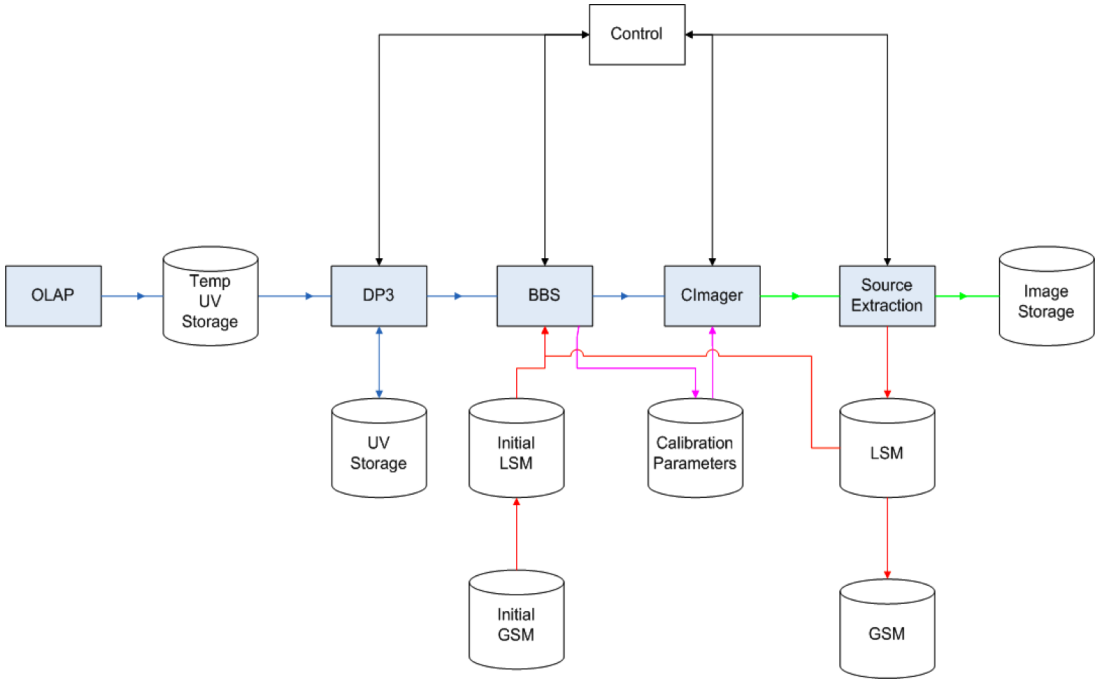



Figure 6: A diagram of the imaging pipeline.

At the moment, many aspects of the LOFAR data reduction are still under development and are being tested. The flagging and compression of the data through NDPPP is currently in a good shape and its details are almost fully understood and tested. BBS and the IMAGER are in an advanced stage of development, but they still need further testing to become fully operational. More still need to be done for the source extraction routine.
The goal of the LOFAR Imaging Cookbook is to explain our understanding of how to use the various softwares developed for LOFAR data reduction. Quicker results could be obtained by using other softwares (such as CASA). However we strongly encourage the users to follow the proposed way of

the cookbook, post results or problems in the LOFAR forum[21], and talk to the software developers (see Sect. 13.3).

## 3.1 Running long-during processes

Running the pipeline, or the individual sub processes, can take a long time. In this case, one could start up the process, redirect the output and errors and put the job in the background. One could even logout (after having "disowned" the job), to return a few hours later to see how the process is doing. For such purpose, one can make use of the `screen` utility, which creates a terminal inside the actual terminal, which behaves independently of the parent one. Logging out of the parent terminal will leave the screen terminal running. Using the `screen` utility can also prevent problems caused by accidental internet disconnections and it allows also to run multiple virtual terminals through one login session. For more details, see Section A at the end of the manual.

[21] http://usg.lofar.org/forum/ . To request a login, contact Teun Grit, `grit@astron.nl`.

# 4 The New Default Pre-Processing Pipeline (NDPPP)

NDPPP (formally known as IDPPP, initial default pre-processing pipeline) is the initial flagging and compression (averaging) routine. The frequencies covered by LOFAR are considerably affected by RFI, both in the low and the high band (see Fig. 7). An efficient cleaning of the data is essential to obtain high quality images.

The input to NDPPP is any (regularly shaped) Measurement Set (MS). Regularly shaped means that all time slots in the MS must contain the same baselines and channels. Furthermore, the MS should contain only one spectral window; for a multiband MS this can be achieved by doing a proper TaQL (Table Query Language) selection beforehand. The data in the given column are piped through the flagging and averaging steps defined in the parset file (See Sect. 4.2) and finally written. This makes it possible to flag at the full resolution, average, flag on a lower resolution scale, perform more averaging, and finally write the data to a new Measurement Set.

The output can be a new Measurement Set, but it is also possible to update the flags in the input MS. When averaging is done, however, the only option is to create a new MS.

For specific questions regarding NDPPP, you can contact the software developer, Ger van Diepen (diepen@astron.nl).

## 4.1 Basic usage

To run NDPPP, you should specify in a `parset` file (see Sect 4.2) which are the parameters that you want to use to flag and/or average the data. You can copy this `NDPPP.parset` file into your working directory by typing:

```
cp ~pizzo/EXAMPLES/Parset/NDPPP.parset .
```

After this preparation, you can run the task by typing:

```
> NDPPP
```

Outputs are printed to screen, including the percentage of data that has been flagged for each antenna. In the next section we will describe the layout of the parset file.

## 4.2 The ParSet File

The steps to perform the flagging and/or compression of the data have to be defined in the parset file. They are executed in the given order, where the data are piped from one step to the other until all data are processed. Each step has a name to be used thereafter as a prefix in the keyword names specifying the type and parameters of the step. An extensive description of the parameters which can be set in the parset file is given in the following sections; here a couple of examples are reported.

The most basic parset file is the following. It copies the DATA column of the MS and flags NaN and infinite data[22].

```
##################################################
#
```

[22] This is the basic parset file that you could use if you want to inspect the raw data with CASA.

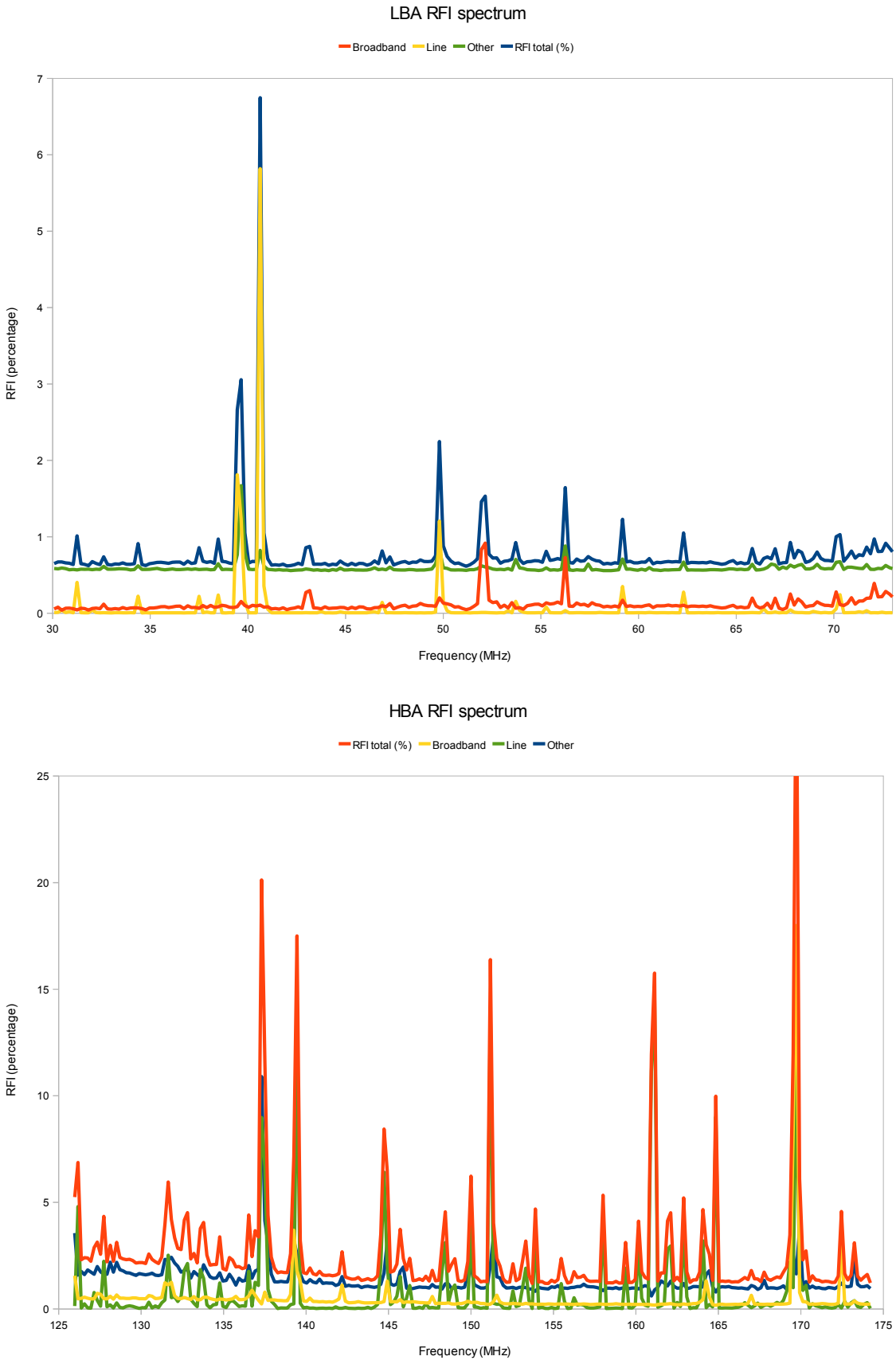



Figure 7: The RFI in the low (top panel) and high (bottom panel) band of LOFAR. The RFI have a broad band, narrow band (line) and other origin. The low-band spectrum is the result of the analysis of 10 minutes LBA observations, while the high-band spectrum derives from the analysis of 6 hours long HBA observations. Curtesy of Andre' Offringa, `offringa@astro.rug.nl`.

```
#  NDPPP.parset
#

msin = ~/SB0.MS
msout = SB0_DPPP.MS
steps=[]
```

The following example is more elaborate. It flags (using a median flagger), averages all channels, flags the result of the average, and finally averages in time. Note that 'msin' and 'msout' can be seen as an implicit first and last step. Note that this is not meant to be the default parset file that you could use on your data. To produce the appropriate parset file to run on your observation, you should first inspect the data and decide which parameters are needed to perform an efficient flagging.

```
#########################################################
#
#  NDPPP.parset
#

msin = ~/SB0.MS
msin.startchan = 8
msin.nchan = 240
msin.datacolumn = DATA     # is the default
msin.autoweight = true   # to calculate the proper weights from the autocorrelations

msout = "SB0_DPPP.MS"      # if empty, the input MS is updated and no averaging
                                                      steps can be done
msout.datacolumn = DATA    # is the default

steps = [preflag,flag1,count,avg1,flag2,avg2,count]   # if defined as [] the MS
                                                        will  be copied and
                                                        NaN/infinite will be
                                                        flagged

preflag.type=preflagger    # This step will flag the autocorrelations. Note  that
                             they are not flagged  by default by NDPPP
preflag.corrtype=auto


flag1.type=madflagger
flag1.threshold=4
flag1.freqwindow=31
flag1.timewindow=5
flag1.correlations=[0,3]   # only flag on XX and YY

avg1.type = squash
avg1.freqstep = 256        # it compresses the data by a factor of 256 in frequency
avg1.timestep = 1          # is the default; it does not compress the data in time

flag2.type=madflagger
```

```
flag2.threshold=3
flag2.timewindow=51

avg2.type = squash
avg2.timestep = 5          # it compresses the data by a factor of 5 in time
```

A description of all the parameters that can be used in NDPPP can be found online at the address `http://www.lofar.org/wiki/doku.php?id=engineering:software:tools:ndppp`.

### 4.2.1 Input / output parameters

A description of the input/output parameters of NDPPP is reported below

msin The `msin` step defines which MS and which DATA column to use. It is possible to skip leading or trailing channels. It sets flags for invalid data (NaN or infinite). Dummy, fully flagged data with correct UVW coordinates will be inserted for missing time slots in the MS. Missing time slots at the beginning or end of the MS can be detected by giving the correct start and end time. This is particularly useful for the imaging pipeline where BBS requires that the MSs of all sub bands of an observation have the same time slots. When updating an MS, those inserted slots are temporary and not put back into the MS.

The input machine will be one of the storage nodes (`lse007 - lse009`). Different sub bands are stored on different nodes, and it may be necessary to search them all for the required data. Measurement sets are stored in the format `LYYYY_XXXXX`, where `L` stands for LOFAR, `YYYY` is the year, and `XXXXX` is the observation number. (i.e. `/net/sub3/lse008/data2/L2009_14319`)

msout The `msout` step defines the output. The input MS is updated if no output name is given.

Data should be written to `/data/scratch/<username>` (which may need creating initially, see Sect. 1.5). Output data should *not* be written back to the storage disks. Also, do not write output data to `/home/<user name>`, as space is very limited on this disk.

You can let `NDPPP` create a so-called VDS file, which tells other data processing programs (notably, `BBS` and `mwimager`) where the data live. You need a so-called cluster description file for this. These can be found in `$LOFARROOT/share` (running `use LofIm` will define the `$LOFARROOT` variable). First check on which subcluster you're working (use `showsub` on the front end node, and see which compute node you've logged into for running `NDPPP`). Then point `msout.clusterdesec` to the corresponding cluster description file. (For the curious, the cluster description is a simple ASCII file that should be straightforward to understand). There is more on cluster description files in the BBS section (Sect. 6).

### 4.2.2 Flagging

The properties of the flagging performed through NDPPP can be summarized as follows.

- If one correlation is flagged, all correlations will be flagged.
- The msin step flags data containing NaNs or infinite numbers.
- A `PreFlagger` step can be used to flag on time, baseline, elevation, azimuth, simple uv-distance, channel, frequency, amplitude, phase, real, and imaginary. Multiple values (or ranges) can be given for one or more of those keywords. A keyword matches if the data matches one

of the values. The results of all given keywords are AND-ed. For example, only data matching given channels and baselines are flagged.

Keywords can be grouped in a set making it a single (super) keyword. Such sets can be OR-ed or AND-ed. It makes it possible to flag, for example, channel 1-4 for baseline A and channel 34-36 for baseline B. Below it is explained in a bit more detail.

- A `UVWFlagger` step can be used to flag on UVW coordinates in meters and/or wavelengths. It is possible to base the UVW coordinates on a given phase center. If no phase center is given, the UVW coordinates in the input MS are used.

- A `MADFlagger` step can be used to flag on the amplitudes of the data. It flags based on the median of the absolute difference of the amplitudes and the median of the amplitudes. It uses a running median with a box of the given size (number of channels and time slots). It is a rather expensive flagging method with usually good results.

  It is possible to specify which correlations to use in the MADFlagger. Flagging on XX only, can save a factor 4 in performance.

### 4.2.3 Averaging

The properties of the averaging performed through NDPPP can be summarized as follows.

- Unflagged visibility data are averaged in frequency and/or time taking the weights into account. New weights are calculated as the sum of the old weights.

  Some older LOFAR MSs have weight 0 for unflagged data points. These weights are set to 1.

- The UVW coordinates are also averaged (not recalculated).

- It fills the new column LOFAR_FULL_RES_FLAG with the flags at the original resolution for the channels selected from the input MS. It can be used by BBS to deal with bandwidth and time smearing.

- Averaging in frequency requires that the average factor fits integrally. E.g. one cannot average every 5 channels when having 256 channels.

- When averaging in time, dummy time slots will be inserted for the ones missing at the end. In that way the output MeasurementSet is still regular in time.

- An averaged point can be flagged if too few unflagged input points were available

### 4.2.4 Combining PreFlagger keywords into sets

The PreFlagger supports the selection on numerous keywords. See the parset description below for a list of all keywords.
A single keyword can have multiple values, for example `baseline=[ [RT0,RT1],[RT0,RT2] ]` specifies two baselines. A data point matches a keyword if it matches one of the values. This is effectively an OR.
The given keywords are AND-ed, thus a data point is only flagged if all keywords match. It makes it possible to express something like: flag frequencies 20-30 MHZ for baselines containing remote stations like:

```
steps=[flag]
flag.type=preflagger
flag.freqrange=[20..30MHz]
flag.baseline=[RS*]
```

In query languages it is common to combine selections using AND and OR operators. The PreFlagger supports this idiom as well. Multiple PreFlagger sets can be defined, each with its own keywords and values. The sets can be combined using the following operators or their synonyms (in decreasing order of precedence). Parentheses can be used to change the order of precedence.

NOT ! <br>
AND & & & <br>
OR | | | ,

Using such a set expression it is possible to also flag frequencies 110-140 MHz for all core-core baselines.

```
steps=[flag]
flag.type=preflagger
flag.sets=s1 or s2                  # could also be given as s1,s2
flag.s1.freqrange=[20..30MHz]
flag.s1.baseline=[RS*]
flag.s2.freqrange=[110..140MHz]
flag.s2.baseline=[[CS*,CS*]]
```

This example shows that each set has a name (in this case s1 and s2), that has to be used as an extra prefix in the names of the keywords in that set. It makes it possible to nest set expressions to any depth. For example, s2 could have a `sets` keyword as shown below. Note that s2 matches if all its keywords match, thus if the freqrange, baseline, and s2a or s2b matches.

```
steps=[flag]
flag.type=preflagger
flag.sets=s1 or s2                  # could also be given as s1,s2
flag.s1.freqrange=[20..30MHz]
flag.s1.baseline=[RS*]
flag.s2.freqrange=[110..140MHz]
flag.s2.baseline=[[CS*,CS*]]
flag.s2.sets=s2a || s2b
flag.s2.s2a.timeofday=23:55:00..0:05:00   # around midnight
flag.s2.s2b.elevation=0deg..10deg
```

## 4.3 MSSelection, antenna/baseline syntax

In order to select particular baselines, antennas, and baseline lengths, NDPPP uses a new baseline selection syntax, which is common also to CASA. Its properties are reported in the following.

- Whitespace can be given at will.
- The selection is given as a list of groups separated by semicolons. A group can be preceded by an exclamation mark meaning exclusion. It can be used to exclude part of a previous group. If

desired, part of that excluded group can be selected again in a subsequent group.
Note that the OR relation is used for ordinary groups, while AND is used for excluded groups.
For example:

```
group1; group2; !group3; group4; !group5
```

means

```
group1 OR (group2 AND NOT group3) OR (group4 AND NOT group5)
```

- A group contains baseline specifications that can be given in two ways: using antenna names/numbers or using physical baseline length ranges

### 4.3.1 Antenna names/numbers

- A group consists of one or two lists of antenna specifications separated by an operator telling which correlations to use.

  1. & means cross-correlations only.
  2. && means cross- and auto-correlations.
  3. &&& means auto-correlations only (no second list can be given in this case).

  If no second list is given, all baselines between the antennae in the first list are selected, otherwise the baselines between the antennae in the first and second list.
  If a single list without operator is given, all cross-correlation baselines containing the given antennae are selected.

- An antenna list consists of one or more antenna names and/or numbers separated by commas. An antenna number is the index (row number) in the `ANTENNA` sub table of a MeasurementSet.

- An antenna name can contain the following characters:
  alphabetic digit : . _ + −
  The first character cannot be a digit.
  However, any character (thus also pattern characters) can be used in a name if it is escaped by preceding it with a backslash.

- Antennae can be specified with a pattern as used in a shell for file names. Such a pattern has the following special characters:

  1. * means zero or more characters
  2. ? means a single character
  3. square brackets give a choice of characters. A hyphen can be used for ranges and an up-arrow for negation. For example:
     `[a-zA-Z0-9]` a single letter or digit.
     `[abcde]` one of these 5 letters.
     `[^abcde]` not one of these letters.
  4. curly braces indicate a choice of strings (separated by commas). For example:
     `'*{h,cc}'` for any name ending in `h` or `cc`
     As shown in the last example a pattern has to be enclosed in single or double quotes if it contains a comma (or possibly other special characters).

- An antenna name can be given as an extended regular expression if enclosed in slashes[23]. For example:
  `/CS.*HBA.*/` for all HBA core stations.
  Note this could somewhat easier be given as a pattern: `CS*HBA*`

- An antenna number can be a single number or a range with the tilde ($\sim$) as separator. The end value is inclusive. For example:
  `0~5,10~12,18,20`

#### 4.3.2 Physical baseline length

- A group consists of one or more baseline length specifications separated by commas. A specification can be one of the following.

  1. `< value`
     all baselines with physical length $\leq$ the given value.
  2. `> value`
     all baselines with physical length $\geq$ the given value.
  3. `value1 ~ value2`
     all baselines with physical length $\geq$ `value1` and $\leq$ `value2`.

- A value is an integer or floating-point number, optionally followed by a unit. The default unit is m (meter). If in a range `value2` has a unit, `value1` defaults to that unit. It is not allowed to have a unit for `value1`, while not having one for `value2`.

- Note there is some ambiguity between a range of antenna numbers and a range of baseline lengths. A range of integer numbers represents antenna numbers, while a range of floating point numbers represents baseline lengths. An integer number followed by a unit is also seen as a floating-point number.
  A range containing an integer and a floating number is not allowed.

#### 4.3.3 Some examples

Some examples of baseline/antenna selection are reported in the following.

`CS*`
all cross-correlation baselines containing core stations.

`!ES*`
all baselines except the cross-correlation baselines containing European stations.

`CS* &`
all cross-correlations between core stations.

[23] See the man pages on the web (e.g., `http://www.regular-expressions.info`) for more info on regular expressions

```
CS* & RS*
```
all cross-correlations between core and remote stations.

```
CS* & [CR]S*
CS* & CS*,RS*
CS*&; CS*&RS*
```
all cross-correlations between core and core or remote stations. The lines give various ways to specify it (in order of performance).

```
CS* && [RE]S*; !  CS001 & [RE]S*; CS001 & RS001
```
all cross- and auto-correlations between core and remote/European. However, the baselines between `CS001` and remote or European stations are excluded with the exception of the baseline between `CS001` and `RS001`.

```
1∼5
1,2,3,4,5
```
all cross-correlations baselines containing station numbers 1 till 5.

```
100.∼500.m
100.∼500.
100m∼500m
.1 ∼.5km
```
all baselines with a physical length between 100 and 500 meter.

```
<1km; !100∼200m
```
baselines with a length $\leq$ 1000 meter with the exception of lengths between 100 and 200 meter.

```
<1km; !RT[56]
```
baselines with a length $\leq$ 1000 meter except baselines containing RT5 or RT6.

## 4.4 Flag statistics

Several steps shows statistics during output about flagged data points.

- A MADFlagger step shows the percentage of visibilities flagged by that flagging step. It shows:
  1. The percentages per baseline and per station.
  2. The percentages per channel.
  3. The number of flagged points per correlation, i.e. which correlation triggered the flagging. This may help in determining which correlations to use in the MADFlagger.
- A UVWFlagger and PreFlagger step show the percentage of visibilities flagged by that flagging step. It shows percentages per baseline and per channel.
- The `msin` step shows the number of visibilities flagged because they contain a NaN or infinite value. It is shown which correlation triggered the flagging, so usually only the first correlation is really counted.

- A Counter step can be used to count and show the number of flagged visibilities. Such a step can be inserted at any point to show the cumulative number of flagged visibilities. For example, it can be defined as the first and last step to know how many visibilities have been flagged in total by the various steps.

## 4.5 Additional information

### 4.5.1 Manual flagging in CASA

While manual flagging will not be practical once the pipeline is completed, during early stages it may be useful to remove remaining RFI in order to test the calibration or imaging routines. The tasks `flagdata` or `plotxy` may be useful for this. Once the CASA Plotter has loaded and data is visible, click the `Mark Region` button, highlight data that you wish to flag, click the `Flag` button, and `Quit` once you are finished.

Observations at 'low' elevation (below $\sim 30^\circ$ for Cygnus A, and below $\sim 40^\circ$ for 3C196) are sufficiently noisy that they are of limited use. These bad time ranges need to be identified and removed. This could be done through NDPPP, but also by manual flagging in CASA or using the CASA SPLIT task or the python script `split_ms_by_time.py` (Section 4.5.2). Splitting out part of a Measurement Set can be done as part of the distributed pipeline and will most likely be necessary until more robust flagging routines are implemented.

### 4.5.2 Splitting the dataset

If you need to select just a time slice of the MS, you can use TaQL (Table Query Language); this is an SQL-like language which works on MS, and can perform all kinds of selections (and more). A `taql` command line task exists that can perform such a time slice:

```
> taql 'select from dataset.MS where TIME in {MJD(2009/06/11/10:00:00),
   MJD(2009/06/11/13:00:00)} giving selected_data.MS
```

Note the use of the `MJD` function: time in a MS is stored in seconds of Modified Julian Day, and the `MJD` function converts a date (note the CASA date convention: slashes as separator between year, month, day and hour) to seconds MJD. The braces ({}) indicate a closed interval, and thus require a start and end point as their arguments.

If you prefer Python, dislike the SQL syntax, or for whatever other reason, you can use a python script `split_ms_by_time.py` [24] that allows you to select part of the MS.

```
> python split_ms_by_time.py
```

The script consists of only a few lines, and is shown below:

```
#################################################################
# split_ms_by_time.py

#!/usr/bin/python
import pyrap.tables as pt
```

[24] It can be found in ~pizzo/EXAMPLES/Scripts

```
# on the offline cluster if you are in c ot tcsh type:
# use Casa; use Pythonlibs; use LofIm; use Casacore

# Run this program as python split_ms_by_time.py
# or Run it as ./split_ms_by_time.py if the script is executable
# Pandey:v0.0:May2010 contact: pandey@astro.rug.nl

####### START USER ENTRY #########
# Enter the correct input and output table names below
tablename = 'abc_output.MS'
outputname = 'abc_output_junk1.MS'

# Please Enter the start and end times in hours for the output Measurement Set
#     relative to the start of input Measurement Set
# for example start = 1.0 means output Measurement Set will start, 1 hour from
#     the start of input MS
# end = 3.0 will mean that output MS will stop 3 hours from the start of
#    INPUT MS
# So output MS will have 2 hours of data in such a case
start_out = 0.1
end_out = 0.3
####### END USER ENTRY #########

print '################################################'

t = pt.table(tablename)

starttime = t[0]['TIME']
endtime   = t[t.nrows()-1]['TIME']


print '====================='

print 'Input Measurement Set is '+tablename
print 'Start time (sec) = '+str(starttime)
print 'End time   (sec) = '+str(endtime)
print 'Total time duration (hrs)  = '+str((endtime-starttime)/3600)

print '====================='

print 'Output Measurement Set is '+outputname
print 'Start time (relative to input ms start) = '+str(start_out)
print 'End time   (relative to input ms start) = '+str(end_out)
print 'Total time duration (hrs)  = '+str(end_out-start_out)

print '====================='

print 'Now going to do the Querry to select the required time range'
```

```
t1 = t.query('TIME > ' + str(starttime+start_out*3600) + ' && \
  TIME < ' + str(starttime+end_out*3600), sortlist='TIME,ANTENNA1,ANTENNA2')


print 'Total rows in Input MS  = '+str(t.nrows())
print 'Total rows in Output MS = '+str(t1.nrows())

print 'Now Writing the output MS'
t1.copy(outputname, True)
t1.close()
t.close()
print 'Copying Completed... Thanks for using the script '
print '################################################'
```

`start` and `end` define the time range that one wants to extract from the initial dataset, in units of hours starting from the beginning of the observation. For example if your observation consists of 10 hours and you want to extract from the 2nd to the 8th hours the required values are `start = 1`, `end = 8`.

# 5 The AOFlagger - RFI console[25]

The AOFlagger ("RFI console") is a separate, independent flagger. It was originally written for the Epoch of Reionization key science project, which needed a flagger with better accuracy compared to the MADFlagger technique implemented in NDPPP (Sect. 4), but is since then optimized to be accurate for any observation and was therefore put inside the LofIm environment. Several comparisons have been made between the MADFlagger and the AOFlagger, and the consensus is that the AOFlagger is more accurate.

The AOFlagger and the NDPPP's flagger have comparable speed. Currently, AOFlagger is faster on single sub bands, while NDPPP can more easily process multiple sub bands in parallel, and therefore is faster when flagging entire observations. Since both are still in development, this might change in time. Because AOFlagger and NDPPP process the data in a different order, one can unfortunately not be implemented in the other. Flagging through the AOFlagger and averaging the data with NDPPP seems a good strategy and should provide data clean enough to perform the calibration and imaging.

## 5.1 How to run the AOFlagger

The AOFlagger runs on a Measurement Set and updates its flag table. This requires write access to the Measurement Set, thus it can not be directly run on the raw data on the storage nodes. Since the raw Measurement Sets are written using a special read-only storage manager (LofarStMan), the storage manager needs to be changed before running RFI console. This can be done by running the following command (e.g. for SB0.MS):

```
> makeFLAGwritable SB0.MS
```

which gives the following output

```
Successful read/write open of default-locked table SB0.MS: 23 columns,
91341 rows
Created new FLAG column; copying old values ...
FLAG column now stored with SSM to make it writable
```

It might take a few moments to rewrite the flag column. After this, your Measurement Set is ready to be flagged by the AOFlagger:

```
> rficonsole SB0.MS
```

The program will now run the default algorithm on the Measurement Set and output status messages and progress to the console. Depending on the size of the observation, this might take up to several hours (it might therefore be appropriate to run it inside a 'screen', as described in Sect. A). In almost all cases, this should produce flags which will be good enough for imaging.

When the flagger will be done, it will also print the statistics of the flagged data per channel. E.g.:

```
INFO - Summary of RFI per channel: (146,862,207 Hz - 149,342,675 Hz)
INFO - Channel   1-  8:  85.2%  21.4%  10.6%  12.1%  12.5%  10.8%  10.4%  12.3%
INFO - Channel   9- 16:  19.9%  24.0%  20.2%  10.3%  10.4%  11.0%  22.4%  11.4%
```

[25] This section has been adapted from a document provided by Andre' Offringa, offringa@astro.rug.nl

```
[..]
INFO - Channel 241-248:  77.0%  75.4%  74.9%  74.9%  75.1%  75.1%  75.6%  75.6%
INFO - Channel 249-255:  78.1%  76.4%  76.7%  76.5%  75.0%  74.8%  74.8%
```

After RFI console, NDPPP should be used to average the data. When averaging a Measurement Set that has been flagged by AOFlagger, your NDPPP parset should not have any flagging steps in it.

## 5.2 Advanced settings with RFI console

If you find problems related to RFI, or you need to change its settings because you want to run it e.g. on Westerbork data, it is possible to change its strategy by creating a configuration file and change the flagger's settings in it. You can create such a file with:

```
> rfistrategy default mystrategy.rfis
```

This will create a file named `mystrategy.rfis`. Settings can be changed either by:

- adding parameters to the `rfistrategy` executable. Run `rfistrategy` without parameters to get a list of options;
- manually changing the strategy inside the file. The strategy file is an xml text file, and can be edited by hand. It is somewhat hard to read with a normal text editor, but opening the file in firefox shows the structure in a comfortable layout.

Once you have completed the alternative strategy, you can run it with:

```
> rficonsole -strategy mystrategy.rfis SB0.MS
```

The RFI console binary can also take additional parameters. These can be retrieved by running the RFI console program without commands. One useful option is to specify the number of threads to be used. This can be done through the `-j` option:

```
> rficonsole -j 7 SB0.MS
```

In this case, we would use 7 threads to flag the data instead of 4, which is the default. More threads will require more memory. More than 7 threads on the clusters slow down the process, therefore it is not recommended.

Since AOFlagger used to be limited by IO seeking and not by cpu performance, a new approach was implemented in which a measurement set is written to a temporary location in a different order. The increase in speed on large sets is on the order of several factors, typically around 3 or 4 times.

Because this approach uses a very large amount of disk space, it is not the default. In most cases however, people will run it on the lce nodes and there should be enough disk space on them to use this approach. One can then run rficonsole using the indirect baseline reader with the "-indirect-read" parameter, and the measurement set will be rewritten. The syntax will be similar to:

```
offringa@lce032:/data/scratch/offringa/temp$ rficonsole -indirect-read -j 8 SB4.MS
```

Please note that the current working directory will be used as a temporary storage location! Thus by running rficonsole like above, temporary files will be created in /data/scratch/offringa/temp that will take up the amount of space equal to the size of the sub-band (i.e. measurement set). So, do not run this in your home directory but always on the local hd's of the nodes.

## 5.3 Flagging of bad baselines

A new feature has been recently implemented in RFI console. At the end of each flagging run, it finds baselines which seems to behave abnormally. The program tries to establish a smooth RFI vs. baseline-length curve as in Fig. 8, estimates a standard deviation of the baselines to this curve, and clips baselines above some threshold, which is defaulted to 6 times the standard deviation.

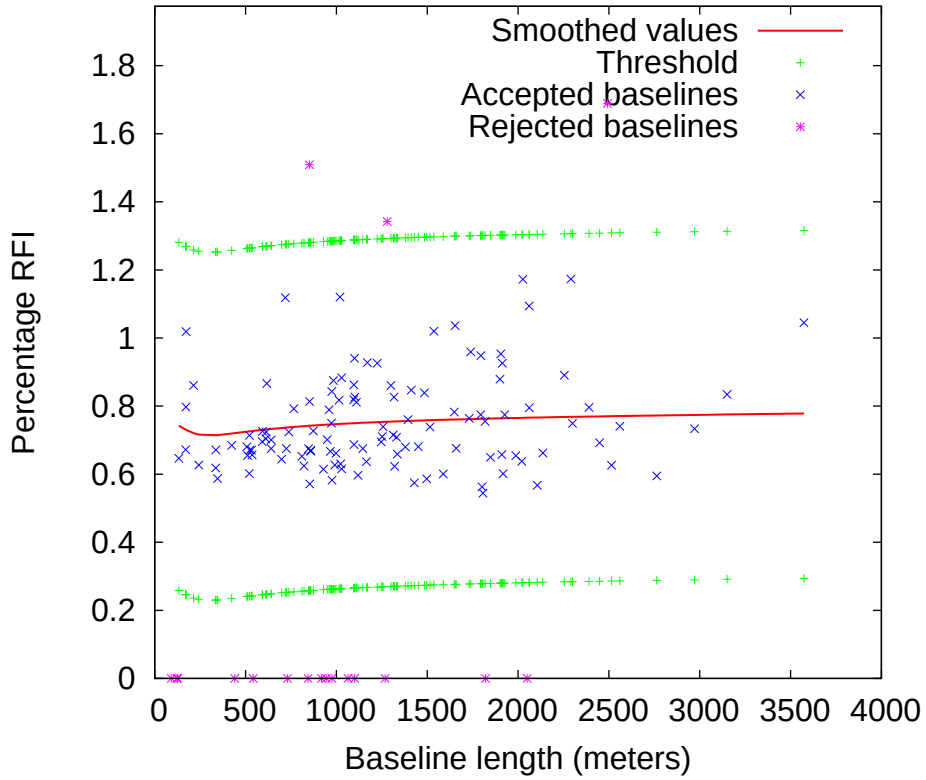



Figure 8: The percentage of RFI vs baseline length for sub band 127 of the observation `L2010_08567`. The bad baselines (in purple) will be reported in the output of RFI console.

Currently, RFI console does not write the flags back to the MS, as it is important to test the new functionality. For the moment, RFI console writes to the terminal which baselines look bad. For the observation analyzed in Fig. 8, the output looks like the following:

```
Baseline CS002LBA x CS501LBA looks bad: 0% rfi (zero or above 40% abs
threshold)
Baseline CS002LBA x CS401LBA looks bad: 0% rfi (zero or above 40% abs
threshold)
Baseline CS002LBA x CS302LBA looks bad: 0% rfi (zero or above 40% abs
threshold)
Baseline CS002LBA x CS301LBA looks bad: 0% rfi (zero or above 40% abs
threshold)
Baseline CS002LBA x CS201LBA looks bad: 0% rfi (zero or above 40% abs
threshold)
Baseline CS002LBA x CS103LBA looks bad: 0% rfi (zero or above 40% abs
threshold)
Baseline CS002LBA x CS032LBA looks bad: 0% rfi (zero or above 40% abs
threshold)
```

```
Baseline CS002LBA x CS101LBA looks bad: 0% rfi (zero or above 40% abs
threshold)
Baseline CS002LBA x CS030LBA looks bad: 0% rfi (zero or above 40% abs
threshold)
Baseline CS002LBA x CS024LBA looks bad: 0% rfi (zero or above 40% abs
threshold)
Baseline CS002LBA x CS021LBA looks bad: 0% rfi (zero or above 40% abs
threshold)
Baseline CS001LBA x CS002LBA looks bad: 0% rfi (zero or above 40% abs
threshold)
Baseline CS002LBA x CS003LBA looks bad: 0% rfi (zero or above 40% abs
threshold)
Baseline CS002LBA x CS004LBA looks bad: 0% rfi (zero or above 40% abs
threshold)
Baseline CS002LBA x CS005LBA looks bad: 0% rfi (zero or above 40% abs
threshold)
Baseline CS002LBA x CS007LBA looks bad: 0% rfi (zero or above 40% abs
threshold)
Estimated std dev for thresholding, in percentage of RFI: 0.09%
Baseline CS030LBA x CS103LBA looks bad: 1.69% rfi, 10.7*sigma away from
est baseline curve
Baseline CS001LBA x CS030LBA looks bad: 1.34% rfi, 6.9*sigma away from
est baseline curve
Baseline CS005LBA x CS201LBA looks bad: 1.51% rfi, 9*sigma away from est
baseline curve
Found 19/136 bad baselines: CS002LBAxCS501LBA, CS002LBAxCS501LBA,
CS002LBAxCS401LBA, CS002LBAxCS302LBA, CS002LBAxCS301LBA,
CS002LBAxCS201LBA, CS002LBAxCS103LBA, CS002LBAxCS032LBA,
CS002LBAxCS101LBA, CS002LBAxCS030LBA, CS002LBAxCS024LBA,
CS002LBAxCS021LBA, CS001LBAxCS002LBA, CS002LBAxCS003LBA,
CS002LBAxCS004LBA, CS002LBAxCS005LBA, CS002LBAxCS007LBA,
CS030LBAxCS103LBA, CS001LBAxCS030LBA, CS005LBAxCS201LBA
```

In the case you would like to apply the flaggings suggested by RFI console to the Measurement Set, you can create a strategy file and change the two occurrences of

```
<flag-bad-baselines>0</flag-bad-baselines>
```

to

```
<flag-bad-baselines>1</flag-bad-baselines>
```

This will make it unnecessary for you to create a new `NDPPP.parset` to perform the baseline flagging.

## 5.4 Documentation

The properties and performance of the AOFlagger have been described in the following papers:

- Post-correlation radio frequency interference classification methods, Offringa et al., MNRAS, Volume 405, Issue 1, pp. 155-167 -

- A LOFAR RFI detection pipeline and its first results, A.R. Offringa et al., Proceedings of Science, RFI2010

# 6 Calibration with BlackBoard Selfcal (BBS)

BBS controls the self-calibration of a measurement set and can optionally be used for data simulation. Before starting to explain the software details, it is important for the users to know a few caveats :

- Previous versions of BBS and MeqTrees[26] defined the total intensity $I$ as $I = XX + YY$, while most other packages (such as CASA) define it as $I = \frac{XX+YY}{2}$. (Actually, there is a factor of 2 difference in all four Stokes parameters, not only Stokes I). Recently (Mar 16, 2010), the MeqTrees developers decided to switch to the conventional definition (see the discussion at `https://great08.projects.phys.ucl.ac.uk/meqtrees/viewtopic.php?f=57&t=54`). Consequently, BBS has also adopted the conventional definition. This implies that, at the moment, only the CImager (and MWimager, see Sect. 7) uses a different definition. BBS, MeqTrees, and the CASA imager should all be consistent. Fluxes in an image produced by the CASA imager should be as expected, using the CImager they will be twice as high.

- If you are planning to use CASA in the following data reduction, you should run the CASA task "clearcal" on your data. This will reinitialize the calibration columns in your measurement set.

- After running BBS *it is important* to inspect the amplitudes of the CORRECTED_DATA column of the MS file and see whether they show high outliers. If this is the case, the calibration may still be good, but another run of flagging should be applied to the data in order to get rid of the bad points. This can be done through NDPPP, giving as a input column to flag the CORRECTED_DATA column of the Measurement Set.

- BBS now supports the new baseline selection syntax used in NDPPP (see Sect. 4.3). This allows to exclude baselines/stations, select on baseline length, etc...

For specific questions regarding BBS, you could contact the software developers, Joris Van Zwieten (zwieten@astron.nl) and Vishambhar Nath Pandey (pandey@astron.nl).

## 6.1 Basic steps

In order to use BBS, you will need to set up a personal database (see Section 1.4). This only needs to be done once. BBS also requires a file that describes the layout of the cluster nodes (e.g. `sub3.clusterdesc`[27]) and a set of parameters for calibrating in the *uv* plane (`uv-plane-cal.parset`[28]). More details about BBS can be found at the web address `http://www.lofar.org/operations/doku.php?id=engineering:software:tools:bbs` .

For each MS that we want to calibrate it is necessary to create first a `MS.vds` file that describes its contents and location. This can be done by running the following command:

```
> makevds sub3.clusterdesc <directory>/<MS file>
```

After this, all `MS.vds` files need to be combined into a single global `.vds` file, ready for calibration and / or imaging. This is done by typing:

[26] MeqTrees is a software package for implementing Measurement Equations. This makes it uniquely suited for simulation and calibration of radioastronomical data, especially that involving new radio telescopes and observational regimes. For more information, visit `http://www.astron.nl/meqwiki`

[27] You can copy this file from ∼`pizzo`

[28] This can be found in ∼pizzo/EXAMPLES/Parset

```
> combinevds <output file>.gds <vds file 1> [<vds file 2> ...]
```

Instead of specifying the list of input `.vds` files, you could type `*.vds`. Note that the *combine* step is still required even when we want to calibrate only one subband.

To run BBS, type:

```
> calibrate -f --key test --cluster-desc ~/imaging.clusterdesc --db ldb001 \
--db-user postgres <global gds file> uv-plane-cal.parset bbs.skymodel <directory>
```

which is here a single command, spread over multiple lines, indicated by the backslash. Important files/information invoked in this command are:

- `<global gds file>`, which is the combined `.vds` file, created in the previous step. It should be given here by its full path, for example /data/scratch/<username>/<global gds file>. This file is particularly important when you want to run multiple BBS processes at the same time. In this case of multiple processes, you must call them with different names (e.g.`"--key test"`, `"--key test1"`, etc...).
- The `bbs.skymodel` (see Sect. 6.2), which should be in the same directory where you are running the command, otherwise specify the path to its location.
- `<directory>`, which is the working directory containing the measurement sets (probably `/data/scratch/<user name>`). It should be created in every `lce` node that you intend to use. You can use the `cexec` command for this (see Sect. 1.2).
- The `-f` option, which is used to make sure that old sky models (and other old files) are cleared when running BBS on the same dataset again.

You can use `calibrate` on its own to see all the options and the mandatory arguments.

## 6.2 Sky model

The sky model defines what BBS will attempt to replicate during calibration. The example models for 3C196 and Cygnus A are given below[29]:

```
############################################################
# 3C196-bbs.skymodel

# (Name, Type, Ra, Dec, I, Q, U, V, ReferenceFrequency='55.468e6', \
SpectralIndexDegree='0', SpectralIndex:0='0.0', SpectralIndex:1='0.0', \
MajorAxis, MinorAxis, Orientation) = format

3C196, POINT, 08:13:36.062300, +48.13.02.24900, 153.0, 0, 0, 0,  , 1, -0.56, -0.05212

############################################################
# CygA-sky.in
```

[29]The models can be found in ∼pizzo/EXAMPLES/Models. Note that earlier versions of BBS used a slightly different format for this model file, and using the wrong format will lead to errors and a 'kernel failure'. Also, take note that the declination separators have to be dots, not colons, unless you want the declination to be interpreted as hours (i.e., multiplied by a factor 15)

```
# (Name, Type, Ra, Dec, I, Q, U, V, ReferenceFrequency='55.468e6', \
SpectralIndexDegree='0', SpectralIndex:0='0.0', SpectralIndex:1='0.0', \
MajorAxis, MinorAxis, Orientation) = format

CygA.E, POINT, 19:59:31.60000, +40.43.48.3000, 1.25
CygA.W, POINT, 19:59:25.00000, +40.44.15.7000, 1.0
```

Note that the header line beginning '# (Name, ') should be a single line, and has been spread over three lines here for clarity (again indicated by a backslash). The lines in the skymodel file should give the parameters of the model components in the same order as the header. Not all parameters are necessary: only the first five entries are mandatory (up to and including Stokes I). If the other entries in the model component line are omitted, the default values in the header are used. If no default value is specified in the header, zero is assumed.

The total intensity of 3C196 has been set to 153, and so if calibration is successful, after imaging the target source should be unresolved with a flux of 153. Cygnus A has been represented as two point sources, with a flux ratio of 1:1.25.

The spectral index a and the curvature c are defined as follows:

$$\log_{10}(S) = \log_{10}(S_0) + \alpha \log_{10}\left(\frac{\nu}{\nu_0}\right) + c\left[\log_{10}\left(\frac{\nu}{\nu_0}\right)\right]^2 \quad (1)$$

with $\nu_0$ being the reference frequency specified in the header. Alpha corresponds to SpectralIndex:0 and c to SpectralIndex:1. SpectralIndexDegree is the degree of equation 1, i.e. SpectralIndexDegree=1 if both alpha and c are non-zero.

New models have been recently released. For 3C196 we have available a model including 2 sources (`3C196-bbs_2sources.skymodel`); for Cygnus A we have two models, one described by two Gaussians (`CygA_gaussian.skymodel`) and the other by multiple Gaussians (`CygA_multiplegaussian.skymodel`). They can be found in ∼pizzo/EXAMPLES/Models.

## 6.3 The parset files

The parset files[30] define the series of operations that BBS will perform. The BBS parset documentation on the wiki contains extensive information on all the options[31], and it is highly recommended that you obtain a hardcopy of this for future reference.

A parset file contains two sets of instructions, the *Strategy* section, which defines the operations to be carried out, and the *Step* section, which defines a single unit of work in the *Strategy*. In the following sections, three parset files are described in detail. They give a good, simple calibration, that is sufficient for the current stage of commissioning. However the users are encouraged to experiment with different calibration options.

Note that BBS now supports the new baseline selection syntax used in NDPPP (see Sect. 4.3). This allows more freedom when the calibration needs only a selection of antennas and/or baselines.

[30] Examples of these files can be found in ∼/pizzo/EXAMPLES/Parset

[31] `http://www.lofar.org/operations/doku.php?id=engineering:software:tools:bbsconfigurationsyntax`

### 6.3.1 simulation.parset

BBS can be used to simulate a *uv* data set, in order to create the data that LOFAR would actually observe, under ideal conditions and with the chosen observational setup. This gives a quick and easy way of assessing how well calibration has worked on your data.

This is an example parset file to simulate *uv* data for an input measurement set [32]:

```
############################################################
# simulation.parset

### Strategy parameters ###
# Parameters controlling the operations to perform. These parameters
# apply for the whole calibration process.

# Input data column to process
Strategy.InputColumn = DATA

# Time range to process, all if left empty
Strategy.TimeRange = []

# Select a subset of the available baselines (uses CASA baseline selection
# syntax).
Strategy.Baselines = *&

# How much data is loaded into memory in one go - useful for large
# datasets. Units are time stamps. All if zero.
Strategy.ChunkSize = 0

# List of the operations that BBS will perform. The first chunk is
# loaded and operations are performed on it. Then the second one is
# loaded and so on. These are strings that identify a Step, the names
# can be decided by the user. For instance if you want to solve and
# correct for gain and then for bandpass you can call the Steps
# solve_gain, correct_gain, solve_bp, correct_bp. In this example we
# are just simulating data according to the sky model, so we need a
# single step.
Strategy.Steps = [predict]


### Predict step parameters ###
# Parameters controlling the operation of each 'predict' step

# The operation to carry out on the data
Step.predict.Operation = PREDICT

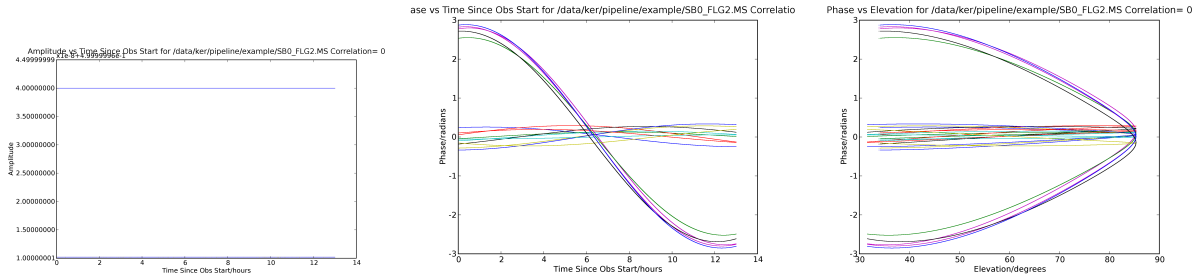
# List of sources to use in model. All in sky model if left empty.
Step.predict.Model.Sources = []

# Measurement set output column to write simulated visibilities to
```

[32] This example can be found in ~pizzo/EXAMPLES/Parset

```
Step.predict.Output.Column = MODEL_DATA
```

An example output of the simulations can be seen in Fig. 9.




(a) Amp. vs. time (b) Phase vs. time (c) Phase vs. elevation

Figure 9: Example outputs for a simulation of the SB0.MS 3C196 observations.

### 6.3.2 uv-plane-cal.parset: GAIN (Im, Re)

This parset[33] performs a simple gain calibration on the data solving for the Real and Imaginary part of the data. The meaning of each parameter is the same as in Section 6.3.1 unless otherwise stated.

```
################################################################
# uv-plane-cal.parset

### Strategy parameters ###
# Parameters controlling the operations to perform

# Input data column to process
Strategy.InputColumn = DATA

# Time range to process, all if left empty
Strategy.TimeRange = []

# Select a subset of the available baselines (uses CASA baseline selection
# syntax).
Strategy.Baselines = *&

# Note that Chunksize must be at least as big as the Cellsize.Time and
# preferably and integer multiple of the value (see below in the Solve step
# parameters).
Strategy.ChunkSize = 0

# Set to true (T) if a global solution will be performed (i.e. if your parset
# contains a key Step.<name>.Solve.CalibrationGroups that is set to something
# other than an empty list ([]).
Strategy.UseSolver = F

# Which steps will be carried out.
```

[33] This example can be found in ∼pizzo/EXAMPLES/Parset

```
# We will be solving, and applying corrections to the data
Strategy.Steps = [solve, correct]


### Solve step parameters ###
# Parameters controlling the operations of each 'solve' step

Step.solve.Operation = SOLVE

# Use 3C196 as a model source
Step.solve.Model.Sources = [3C196]

# There are several ways to model the components. You can specify bandpass,
isotropic or directional gain, enable the beam and enable the ionosphere. Most are
turned off here.
Step.solve.Model.Bandpass.Enable = F
Step.solve.Model.Gain.Enable = T
Step.solve.Model.DirectionalGain.Enable = F
Step.solve.Model.Beam.Enable = F
Step.solve.Model.Ionosphere.Enable = F

# Parameters to solve for: Gains for XX (0:0) and YY (1:1) polarisations.
Step.solve.Solve.Parms = ["Gain:0:0:*", "Gain:1:1:*"]

# Parameters to exclude from fit
Step.solve.Solve.ExclParms = []

# Defines the calibration groups. For example: if you have have 6
# subbands and want solve them in 2 groups you put 3,3. See section on
# running BBS in a distributed fashion.  If empty, do NOT use global
# calibration.
Step.solve.Solve.CalibrationGroups = []

# Data window used to compute a single estimate of the model parameters. This
# determines the "resolution" of the solution (i.e. one solution per channel,
# per second, or one solution per all channels, per 10 minutes, etc.).
# NB. The current solution-based flagging implementation only works for the case
# where CellSize.Freq = 0 and CellSize.Time = 1.
# Solution cell size (no of channels)
Step.solve.Solve.CellSize.Freq = 0
# Solution cell size (no of timeslots)
Step.solve.Solve.CellSize.Time = 1

# Define how many solution cells are simultaneously processed. CellChunkSize is
# in units of timeslots.
# NB. It is recommended that Strategy.ChunkSize is an integer multiple of
# CellChunkSize * CellSize.Time.
# Generally best to set this in the range 10 - 25.
Step.solve.Solve.CellChunkSize = 1
```

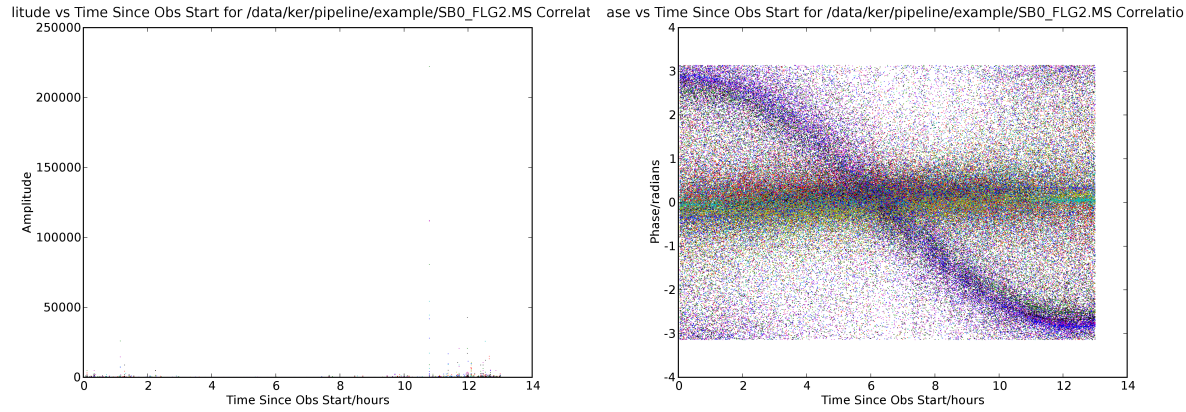
```
# Propagate solutions from one solution cell to the next. The fit uses the
# previous solution as a starting point for the next one. False is
# safer if there are a large number of bad calibration solutions.
Step.solve.Solve.PropagateSolutions = F

# Stop criteria, from CASA libraries -- see wiki for details.
Step.solve.Solve.Options.MaxIter = 20
Step.solve.Solve.Options.EpsValue = 1e-9
Step.solve.Solve.Options.EpsDerivative = 1e-9
Step.solve.Solve.Options.ColFactor = 1e-9
Step.solve.Solve.Options.LMFactor = 1.0
Step.solve.Solve.Options.BalancedEqs = F
Step.solve.Solve.Options.UseSVD = T


### Correct step parameters ###
# Parameters controlling the operations of the 'correct' step, that
# applies the solutions found above.

Step.correct.Operation = CORRECT
Step.correct.Model.Sources = []
Step.correct.Model.Gain.Enable  = T
Step.correct.Output.Column = CORRECTED_DATA
```

The output of the calibration on SB0.MS 3C196 can be seen in Fig. 10.




(a) Amp. vs. time.

(b) Phase vs. time.

Figure 10: 3C196 corrected amplitudes and phases, after calibration. The need for further flagging is clear.

### 6.3.3 uv-plane-calphase.parset: PHASE (Amp, Ph)

This section describes a phase-calibration in BBS for data that have already been calibrated in amplitude. The meanings of each parameter are the same as in Section 6.3.1 and 6.3.2, unless otherwise stated.

The parset example below has been updated as much as possible accordingly to the new parset format. For more details, see `http://usg.lofar.org/forum/index.php?topic=192.0`.

```
#############################################################
# uv-plane-calphase.parset

### Strategy parameters ###
# Parameters controlling the operations to perform

# Now we want to solve and apply the phase solutions to the calibrated
# data, that are in the CORRECTED_DATA column of the MS.
Strategy.InputColumn = CORRECTED_DATA
Strategy.TimeRange = []
Strategy.Baselines = *&
Strategy.UseSolver = F
Strategy.ChunkSize = 0
Strategy.Steps = [solve, correct]


### Solve step parameters ###
# Parameters controlling the operations of each 'solve' step
Step.solve.Operation = SOLVE
Step.solve.Model.Sources = [3C196]
Step.solve.Model.Gain.Enable = T

# To solve for (Amp,Phase) instead of (Re,Im)
Step.solve.Model.Phasors.Enable = T

# To solve only for the Phase gain and not for the Amplitude gains.
Step.solve.Solve.Parms = ["Gain:0:0:Phase:*", "Gain:1:1:Phase:*"]

Step.solve.Solve.ExclParms = []
Step.solve.Solve.CellSize.Freq = 0
Step.solve.Solve.CellSize.Time = 1200
Step.solve.Solve.CellChunkSize = 25
Step.solve.Solve.PropagateSolutions = F
Step.solve.Solve.Options.MaxIter = 20
Step.solve.Solve.Options.EpsValue = 1e-9
Step.solve.Solve.Options.EpsDerivative = 1e-9
Step.solve.Solve.Options.ColFactor = 1e-9
Step.solve.Solve.Options.LMFactor = 1.0
Step.solve.Solve.Options.BalancedEqs = F
Step.solve.Solve.Options.UseSVD = T

### Correct step parameters ###
Step.correct.Operation = CORRECT
Step.correct.Model.Sources = []
Step.correct.Model.Gain.Enable  = T

# Correct for (Amp,Phase) - only Phase in our example - instead of (Re,Im)
Step.correct.Model.Phasors.Enable = T

Step.correct.Output.Column = CORRECTED_DATA
```

## 6.4 Phase-only calibration

A recently implemented feature in BBS is the phase-only calibration. The following parset file[34] performs this operaration. It executes two solving steps: first a gain calibraion for both amplitude and phase, then a directional phase only calibration. The second calibration solves for phase offsets with respect to the first gain solution.

```
### Strategy parameters ###
# Parameters controlling the operations to perform
Strategy.InputColumn = DATA
Strategy.TimeRange = []
Strategy.ChunkSize = 1000
Strategy.UseSolver = F
Strategy.Correlations = []
Strategy.Stations = []
Strategy.Steps = [solve1, solve2]

# First solve step is a regular gain calibration
Step.solve1.Baselines = []
Step.solve1.Model.Sources      = []
Step.solve1.Model.DirectionalGain.Enable  = F
Step.solve1.Model.Gain.Enable  = T
Step.solve1.Model.Phasors.Enable = T
Step.solve1.Correlations= []
Step.solve1.Operation        = SOLVE
Step.solve1.Output.Column    =
Step.solve1.Solve.Parms           = ["Gain:0:0:*", "Gain:1:1:*"]
Step.solve1.Solve.ExclParms       = []
Step.solve1.Solve.CalibrationGroups = []
Step.solve1.Solve.CellSize.Freq          = 1
Step.solve1.Solve.CellSize.Time          = 1
Step.solve1.Solve.CellChunkSize          = 10
Step.solve1.Solve.PropagateSolutions     = F
Step.solve1.Solve.Options.MaxIter        = 20
Step.solve1.Solve.Options.EpsValue       = 1e-9
Step.solve1.Solve.Options.EpsDerivative  = 1e-9
Step.solve1.Solve.Options.ColFactor      = 1e-9
Step.solve1.Solve.Options.LMFactor       = 1.0
Step.solve1.Solve.Options.BalancedEqs    = F
Step.solve1.Solve.Options.UseSVD         = T
Step.solve1.Model.Cache.Enable  = T

Step.solve2.Baselines = []
Step.solve2.Model.Sources      = []

# Enable both gain and directional gain
Step.solve2.Model.DirectionalGain.Enable  = T
Step.solve2.Model.Gain.Enable  = T
```

[34] `uv-plane-calphaseonly.parset`, which can be found in /pizzo/EXAMPLES/Parset

```
Step.solve2.Model.Phasors.Enable = T
Step.solve2.Correlations = []
Step.solve2.Operation        = SOLVE
Step.solve2.Output.Column    =

# Solve only for the phase term of directional gain parameters
# The "Gain:*" are not free parameters now
# the solutions from the previous solve step will be used
Step.solve2.Solve.Parms  = ["DirectionalGain:0:0:Phase:*",
                            "DirectionalGain:1:1:Phase:*"]
Step.solve2.Solve.ExclParms       = []
Step.solve2.Solve.CalibrationGroups = []
Step.solve2.Solve.CellSize.Freq          = 1
Step.solve2.Solve.CellSize.Time          = 1
Step.solve2.Solve.CellChunkSize          = 10
Step.solve2.Solve.PropagateSolutions     = F
Step.solve2.Solve.Options.MaxIter        = 20
Step.solve2.Solve.Options.EpsValue       = 1e-9
Step.solve2.Solve.Options.EpsDerivative  = 1e-9
Step.solve2.Solve.Options.ColFactor      = 1e-9
Step.solve2.Solve.Options.LMFactor       = 1.0
Step.solve2.Solve.Options.BalancedEqs    = F
Step.solve2.Solve.Options.UseSVD         = T
Step.solve2.Model.Cache.Enable  = T
```

## 6.5 Inspecting the gain solutions

### 6.5.1 Parmdbplot

You can inspect the gain solutions using the python script `parmdbplot.py`[35] (recommended).
To start `parmdbplot.py` from the command line, you should first have initialized the LofIm environment and the Pythonlibs (see Sect. 1.3). The you can type, for example

```
> parmdbplot.py SB23.MS.dppp.dppp.dppp.dppp/instrument/
```

This plots the amplitude/phase gain solutions from the calibration for each baseline.

The first thing you should see after starting the script should be the main window (see Fig. 11). Here you can select a set of parameters to plot together in a single plot. In the circled portion of the window you can set the resolution in Hz x s on which to evaluate the parameter(s). By default, the plotter tries to find values that will yield a 100x100 grid in frequency x time. Usually, you would want to change the resolution to the sampling intervals used in the observation. If you uncheck the resolution option, the plotter will try to read the resolution from the parmdb itself. However, this is not always set correctly and can lead to a huge amount of data points being generated, which will cause the plotter to get hung up.

Once you click the ”Plot” button, a window similar to the Fig. 12 should pop up. We discuss the controls in the circled portion from left to right. The first is a drop down box that allows you to select

[35] It has been written by Joris van Zwieten, zwieten@astron.nl

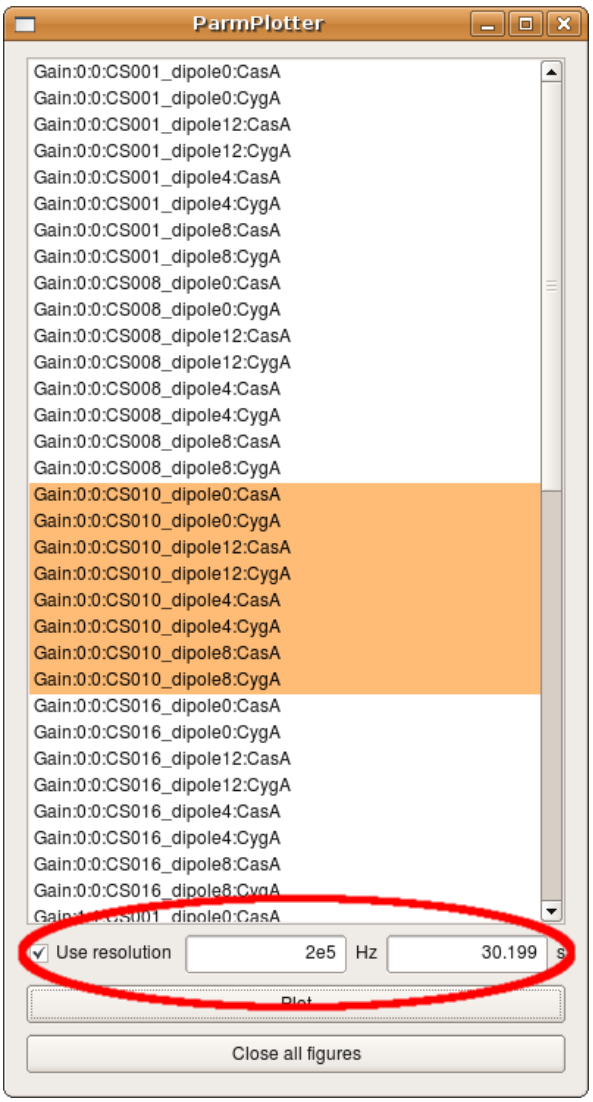



Figure 11: The main window of the `parmdbplot`.

the axis (frequency, time) to slice over. By default, this is set to frequency, which means that the x-axis in the plot is time and that you can step along the frequency axis using the spin control (the second control from the left). Note that the unit of the x-axis in the plot is basically the resolution you provided when creating the plot, i.e. if you specify a resolution of 2 seconds, then the number 10 on the x-axis means 10x2 = 20 seconds. The ”Legend” checkbox allows you turn on/off the legend (which can be quite large and thus obscure the plot, so it is off by default). The ”Polar” checkbox lets you select if the parameter value is plotted as amplitude, phase (the default) or real, imaginary. Finally, the ”Unwrap phase” checkbox will turn on/off phase unwrapping (only relevant when viewing in amplitude/phase mode). The remaining controls on the far right are the default matplotlib controls that allow you to pan, zoom, save the plot, etc.

### 6.5.2 plot.py

To inspect the gain solutions you can also use the python script, `plot.py`. It requires the scripts `solfetch.py` and `solplot.py`[36] to be present in your working directory.

[36]These scripts can be found in ∼pizzo/EXAMPLES/Scripts

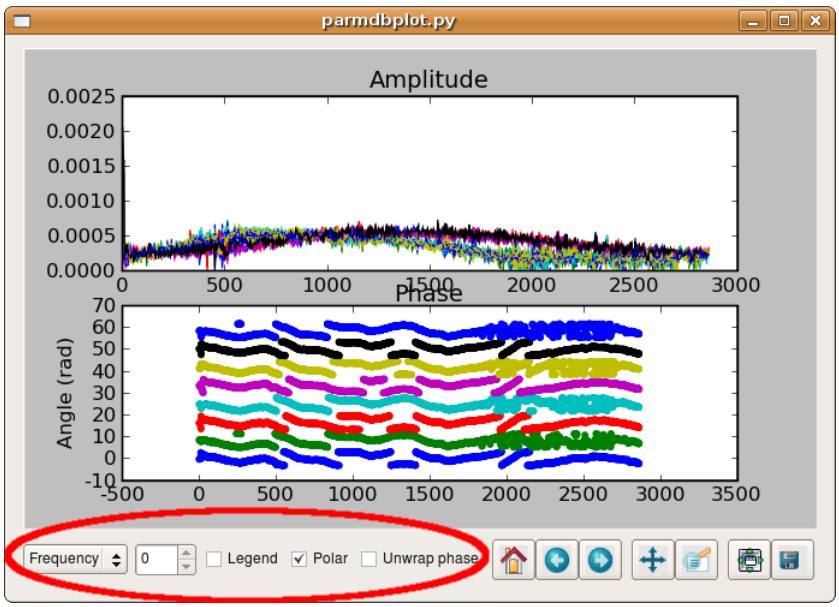



Figure 12: The plot window of the `parmdbplot`.

Station names can be discovered by using `msinfo.py` (Section 2.1).

```
> ipython -pylab
> import lofar.parmdb as parmdb
> import lofar.solutions as sol
> stations = ["CS103LBA","CS302LBA"]    # Just an example, put appropriate
                                          station names here.
> db = parmdb.parmdb("test.MS/instrument") # Put appropriate parmdb name here.
> gain = sol.fetch(db, stations, parm="Gain:1:1")
> sol.plot(gain[0]) # Plot the gain amplitude.
> sol.plot(gain[1]) # Plot the gain phase.
```

This python script plots the amplitude/phase gain solutions from the calibration for each baseline (as shown in Fig. 13). The baselines are plotted with an offset so to be clearer - you can remove this option with `stack=False`. NB. This example *only works* if you solved with direction independent gain, i.e. `Gain`, *not* `DirectionalGain`. If you used `DirectionalGain`, specify a direction as an additional argument to `lofar.solutions.fetch()` and use `DirectionalGain` instead of `Gain` in the ”parm” argument, e.g.:

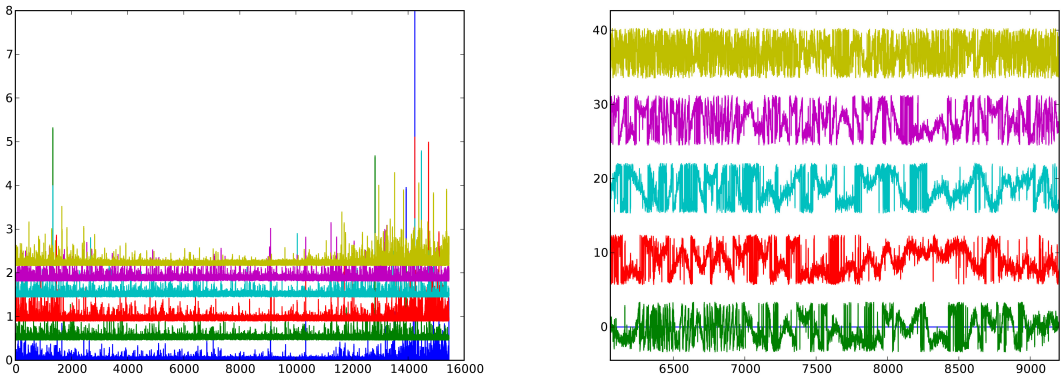
```
> gain = sol.fetch(db, stations, parm="DirectionalGain:1:1", direction="CasA")
```

NB. The `lofar.solutions.fetch()` function *only works* for complex, station (and optionally direction) bound parameters. This is OK when plotting (Directional)Gain solutions, but is not applicable to all possible calibration parameters. The parmdbplot.py script (see following Section) is more general in this respect and should correctly plot any calibration parameter.

Gain:0:0 refers to XX, Gain:1:1 to YY.

## 6.6 Gains transfer from a calibrator to the target source

The gain solutions can be transferred from a calibrator to a target source using `parmdbm`. The procedure is as follows:




(a) Amplitude solutions. (b) Phase solutions.

Figure 13: The amplitude and phase gain solutions for SB0.MS 3C196 observations.

- When calibrating the calibrator observation through BBS, it is important to obtain *time independent* solutions. Moreover, the frequencies should match the target sub bands. For the time independence you should use these settings in the BBS parset[37]:

```
Strategy.ChunkSize = 0
Step.solve.Solve.CellSize.Time = 0
Step.solve.Solve.CellChunkSize = 0
```

- Export solutions so they can be applied to target field.[38] Here is an example:

```
(heald)lce026> parmdbm
log4cplus:WARN Property configuration file "parmdbm.log_prop" not found.
log4cplus:WARN Using basic logging configuration.
Command: open tablename='3c196_1.MS/instrument'
Command: export Gain* tablename='output.table'
Exported record for parameter Gain:0:0:Imag:CS001HBA0
Exported record for parameter Gain:0:0:Imag:CS002HBA0
... more of the same ...
Exported record for parameter Gain:1:1:Real:RS307HBA
Exported record for parameter Gain:1:1:Real:RS503HBA
Exported 104 parms to output.table
Command: exit
```

- Apply gain solutions to target field, using the following flag in the calibrate script:

```
--instrument-db output.table
```

and use a BBS parset which *only* includes a `CORRECT` step. Now you should have a calibrated `CORRECTED_DATA` column which can be imaged.

[37] Note that these settings would be dangerous for a long observation.

[38] See `http://www.lofar.org/operations/doku.php?id=engineering:software:tools:parmdbm` for details.

## 6.7 Solution-based flagging

The solution-based flagger accepts the following parameters:

```
flag(msName, dbName, half_window, threshold, sources=None, storeFlags=True,
updateMain=True, cutoffLow=None, cutoffHigh=None, debug=False)

   msName:         name of the measurement to flag
   dbName:         name of solution parameter database
   half_window:    half the size of the window used in the flagging algorithm
   threshold:      threshold for the flagging algorithm (median of the absolute
                   distance to the median); typical values 2, 3, 4
   sources:        (default None) for directional gains solutions, specify the
                   source directions that should be considered
   storeFlags:     (default True) if set to False, the flags will not be
                   written to the measurement
   updateMain:     (default True) if set to True, both the FLAG and the
                   FLAG_ROW column will be updated if set to False, only the
                   FLAG_ROW column will be updated
   cutoffLow:      (default None) if set, all values less than or equal to
                   cutoffLow will be flagged
   cutoffHigh:     (default None) if set, all values greater than or equal to
                   cutoffHigh will be flagged
   debug:          (default False) if set to True, a plot is generated for each
                   source - station combination that shows what has been flagged.
```

The following values may be good suggestions to start with. Fine tuning will however be needed.

```
half_window = 2000
threshold = 4
cutoffLow = 0
cutoffHigh = 0.1
```

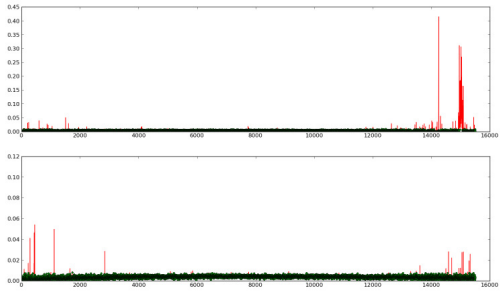
To run the solution-based flagger script:

```
> ipython -pylab
> import lofar.solutions
> lofar.solutions.flag("SB0_FLG2.MS","SB0_FLG2.MS/instrument",2000,4,cutoffLow=0,
cutoffHigh=0.1,storeFlags=False,debug=True)
```

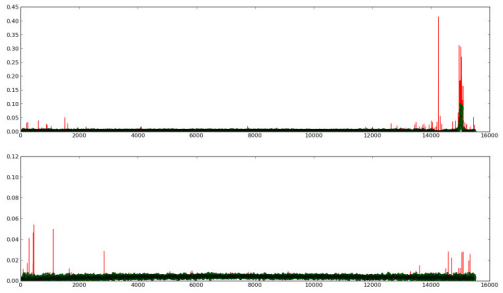
Inspect the plots (e.g. Fig. 14) and adjust the `cutoffLow` and `cutoffHigh` thresholds, `half_window`, and `threshold` parameters as necessary. A large `half_window` will prevent flagging slow variation of the solutions in time. A large `threshold` will prevent flagging low amplitude variations of the solutions.

Rerun this stage until you are happy with the thresholds selected. Then, apply the flags:

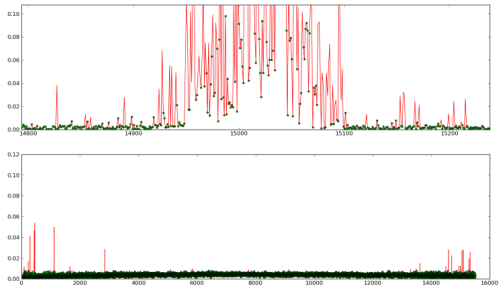
```
> ipython -pylab
> import lofar.solutions
> lofar.solutions.flag("SB0_FLG2.MS","SB0_FLG2.MS/instrument",2000,4,cutoffLow=0,
cutoffHigh=0.1)
```



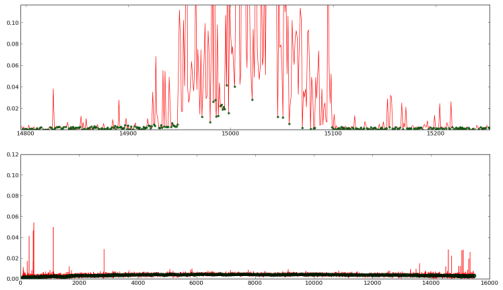
(a) half_window = 2000, threshold = 4.



(b) half_window = 100, threshold = 4.



(c) Zoom - half_window = 100, threshold = 4.



(d) Zoom - half_window = 100, threshold = 1.

Figure 14: Amplitudes (and zoom) after solution-based flagging (a) with `half_window` and `threshold` set correctly; (b) `half_window` set to a too low value; (c) with `half_window` set to a too low value; (d) with `threshold` set to a low value (green points = kept data).

WARNING: The flagging applied cannot be undone, so be sure you are happy with the thresholds and/or make a backup measurement set.
The `lofar.solutions.flag()` function currently only works for *scalar* Gain or *Directional* Gain solutions, where a solution was computed per timeslot (i.e. CellSize.Freq = 0 and CellSize.Time = 1).

## 6.8 Flagging calibrated data

After calibration, it is advised to plot the visibilities to check if they show outliers. If so, the dataset should be flagged. This can be done through NDPPP (see Sect. 4), giving as a input to the flagger the CORRECTED_DATA column of the Measurement Set.

Also the script `CallSolFlag.py`[39] could be used to flag calibrated data in the simple case of flagging points above a given flux-threshold:

```
> CallSolFlag.py <BAND.MS> -l <flux threshold in Jy>
```

## 6.9 Source subtraction[40]

In order to reveal background sources in an image, it is necessary to subtract the strong sources. This can be done by using the following `image-plane-cal.parset`[41].

[39] It can be found in ∼/pizzo/Scripts

[40] This section has been adapted from a document written by Annalisa Bonafede, a.bonafede@jacobs-university.de

[41] Example of these parset files can be found in ∼pizzo/EXAMPLES/Parset

### 6.9.1 image-plane-cal.parset

In the following, we report the parset file and skymodel used for the subtraction of Cas A and Cyg A from the observation of the radio source 3C380.
The parset file performs the following steps: solves for all the sources in the skymodel, using directional gains, then subtracts the sources of the A-team specified in the skymodel file (CygA.E, CygA.W, and CasA in this example ), and finally applies (correct) the gain solutions of the target source: 3C380.

```
#######################################################
#
# image-plane-cal.parset
#

Strategy.ChunkSize = 0
Strategy.Steps = [solve, subtract, correct]

Step.solve.Operation = SOLVE
Step.solve.Model.Sources = []
Step.solve.Model.DirectionalGain.Enable = T
Step.solve.Model.Cache.Enable = T
Step.solve.Solve.Parms = ["DirectionalGain:0:0:*","DirectionalGain:1:1:*"]
Step.solve.Solve.CellSize.Freq = 0
Step.solve.Solve.CellSize.Time = 5
Step.solve.Solve.CellChunkSize = 10
Step.solve.Solve.Options.MaxIter = 20
Step.solve.Solve.Options.EpsValue = 1e-9
Step.solve.Solve.Options.EpsDerivative = 1e-9
Step.solve.Solve.Options.ColFactor = 1e-9
Step.solve.Solve.Options.LMFactor = 1.0
Step.solve.Solve.Options.BalancedEqs = F
Step.solve.Solve.Options.UseSVD = T

Step.subtract.Operation = SUBTRACT
Step.subtract.Model.Sources = [CygA.E, CygA.W, CasA]
Step.subtract.Model.DirectionalGain.Enable = T

Step.correct.Operation = CORRECT
Step.correct.Model.Sources = [3C380]
Step.correct.Model.DirectionalGain.Enable = T
Step.correct.Output.Column = CORRECTED_DATA
```

The skymodel used for calibration is reported below[42]:

```
############################################################
# 3C380-bbs.skymodel
# (Name, Patch, Type, Ra, Dec, I, Q, U, V, ReferenceFrequency='55.468e6', \
```

[42]This model is available in ∼/pizzo/EXAMPLES/Models

```
  SpectralIndexDegree='0', SpectralIndex:0='0.0', SpectralIndex:1='0.0', \
  MajorAxis, MinorAxis, Orientation) = format
CygA.E, , POINT, 19:59:29.99, +40.43.57.53, 4421, 0.0, 0.0, 0.0,73.8e6, 0, -0.7
CygA.W, , POINT, 19:59:23.23, +40.44.23.03, 2998, 0.0, 0.0, 0.0,73.8e6, 0, -0.7
CasA, , POINT, 23:23:24.0,  +58.48.54.0, 20000, 0.0, 0.0, 0.0, 50.e6,  0, 0
3C380, , POINT, 18:29:31.8, +48.44.46.0, 1, 0.0, 0.0, 0.0, 178.e6, 0,-0.7
```

The flux of 3C380 has been set to the arbitrary value of 1 Jy. Its spectral index has been roughly estimated by comparing VLSS and 3C flux. The subtraction can also be performed without specifying the correct flux of the sources and determining the flux of the target source with self calibration.

The resulting image in shown in Fig. 15, compared to the map obtained without subraction.

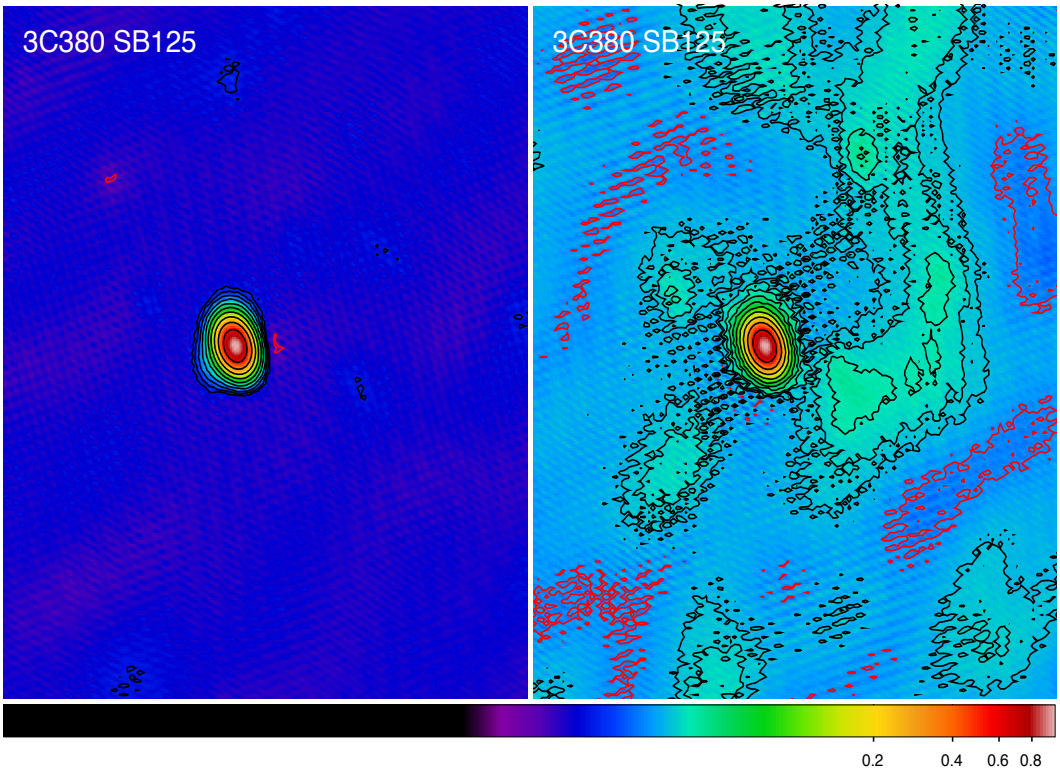



Figure 15: The radio source 3C380 imaged by LOFAR at 135 MHz (observation ID L2010_08567). *Left panel*: self calibration has been applied with directional-gain correction and subtraction of Cyg A and Cas A. *Right panel*: self calibration has been applied without correction. The lowest contour level is at 3 mJy beam$^{-1}$; subsequent contour levels are spaced by factors of $\sqrt{2}$. The resolution is 83.29" $\times$ 59.42". The negative contour is in red.

## 6.10 How to set default parameters

For some circumstances, e.g. simulating differential Faraday rotation, you will need to provide BBS with parameter values which are not defaults. To do this you can create a table to hold your desired parameters. Use the program `parmdbm` (parameter database manager).

```
> parmdbm
```

```
Command: create tablename='mydefault.tab'
Command: adddef RotationMeasure values=-42    # Note the output and make sure
                                              # that it reports "values: -42"
Command: showdef RotationMeasure              # to check that it has worked
Command: exit
```

This creates a new table, which is a directory called `mydefault.tab` in this case. It sets the Faraday rotation at each station to have a value of -42 rad $m^{-2}$. Now when you use BBS via the 'calibrate' script, you have to tell it where to look for these values. That is done by including the extra flag:

```
--instrument-db mydefault.tab
```

before the gds filename in the calibrate command.

Not only does this provide a user-specified value for parameters that you are not solving for, but it also sets the starting value for parameters that you will solve for in BBS.

The default instrument parameter database that `calibrate` creates if you do not specify your own, is created as follows:

```
> parmdbm

Command: create tablename="instrument"
Command: adddef Gain:0:0:Ampl  values=1.0
Command: adddef Gain:1:1:Ampl  values=1.0
Command: adddef Gain:0:0:Real  values=1.0
Command: adddef Gain:1:1:Real  values=1.0
Command: adddef DirectionalGain:0:0:Ampl  values=1.0
Command: adddef DirectionalGain:1:1:Ampl  values=1.0
Command: adddef DirectionalGain:0:0:Real  values=1.0
Command: adddef DirectionalGain:1:1:Real  values=1.0
Command: adddef AntennaOrientation values=5.497787144
Command: exit
```

Please note that if you create your own instrument parameter database, you will also want to include the `adddef` commands above to set the correct defaults for `Gain`, `DirectionalGain`, and `AntennaOrientation`. Otherwise, solving for `Gain` and `DirectionalGain` will *not* work!

## 6.11 Running BBS in a distributed fashion / global calibration

BBS can run on individual subbands, as previously described, but can also calibrate several subbands at once (a 'global' calibration), in a manner illustrated by Fig. 16. To run a BBS global calibration, you need a working directory (with exactly the same path) on each of the `lce` nodes that contain the subbands.

Run `makevds` on each of the subbands in turn. Copy all the `.MS.vds` files to one location and run `combinevds` on all. Run the calibrate script using the output file of `combinevds`.

To perform a global calibration, change the `Strategy.UseSolver` line in the calibration parset file to `Strategy.UseSolver = T` and update the `Step.solve.Solve.CalibrationGroups = []` line to the desired number of kernels and calibration groups (see the BBS wiki page for more details).

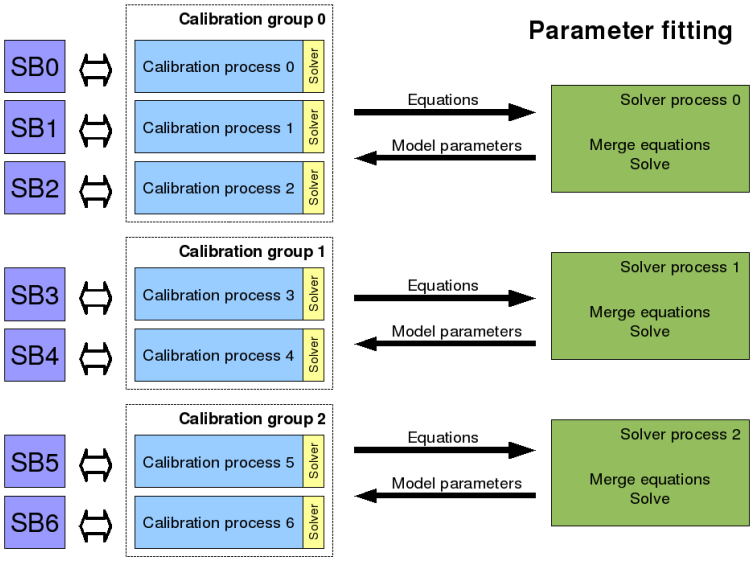



Figure 16: Illustration of how BBS runs in a distributed fashion (taken from the LOFAR wiki).

## 6.12 BBS troubleshooting

- After calibrating many frequency channels (e.g. for spectral line imaging purposes), the spectral profile could show an artificial sin-like curve. This is due to the fact that BBS applied a single solution to all the input channels. To avoid this, it is important to set the parameter `Step.solve.Solve.CellSize.Freq` to a value higher than 0, indicating the number of channels that BBS will try to find a solution for (0 means one solution for all the channels). You can inspect the solutions with `parmdbplot` to judge if this is required.
- BBS seems to have some problems in finding reasonable amplitude gain solutions when the value of `Step.solve.Solve.CellSize.Time` is not 1. Developers are working on this.
- If BBS does not work, it is likely a problem with the version used, or other components of the pipeline, i.e. NDPPP.
- The `<key name>*.log` files produced by BBS can give a very useful indication of when/where things went wrong. Look at these first if you experience a crash. In the following some error messages are reported together with the solution we have found to fix them.

### 6.12.1 Common errors

- `[FAIL] error: setupsourcedb or remote setupsourcedb-part process(es) failed`

  Solution: Check your sky model file.

- `> ERROR clean database for key default failed`

  Solution: Check if you can reach the database server on ldb001 and make sure that you created your personal database correctly. When in doubt, recreate your personal database as described in Section 1.4).

If BBS crashes for any reason, be sure to kill all BBS processes before running again (e.g. `killall GlobalControl`, `killall KernelControl`, `killall SolverControl`).

# 7 Imaging with MWImager

After calibration, an image can be created using `MWimager`[43]. It is worth noting that MWImager is still under development and the imaging of LOFAR data could also be done using CASA (task CLEAN). However, we encourage the users to use MWimager and get in contact with the software developers[44] to help the improvement.

MWimager is a wrapper around underlying programs doing the imaging. These are:

- cimager: a new imager developed at ATNF for the ASKAP observations.
- lwimager: an imager using the CASA imaging libraries.

MWimager forms an image cube from the visibility data in a LOFAR dataset. Similar to NDPPP and BBS, it operates in a distributed way using the so-called `.gds` file.

## 7.1 Running MWimager

As mentioned before, MWimager acts as a front end to two imagers: by default, it uses the ASKAP imager (the actual program is `cimager`), developed by the ASKAP people for both LOFAR and ASKAP. This is the imager intended to be used in the LOFAR imaging pipeline, and should be tested most often. As a fallback option, there is the CASA imager (`lwimager`), which is similar to the interactive imager in casapy, but runs standalone.

Both imagers are capable of the following:

- imaging from the DATA, MODEL_DATA, or CORRECTED_DATA column
- imaging individual channels
- multi-frequency synthesis (N visibility channels to M image channels)
- imaging of all Stokes
- wide-field imaging using facets or W-projection (or combination)
- deconvolution (though not all clean algorithms are supported by both imagers).

Furthermore MWimager is capable of applying direction-dependent corrections (beam, ionosphere).

MWimager requires the same cluster description file (e.g. `sub3.clusterdesc`) as BBS, a parameter set file (`.parset`) to tell it how to operate (see Sect. 7.2), and various scripts that actually run the program. Moreover, the file called `.casarc`[45] is also required in the home directory. It points to various calibration tables, necessary to run the imager.
Once you have copied and adjusted the parameters in your parset file `MWimager.parset` (see section below), run the following commands to create an image:

```
> mwimager mwimager.parset imaging.clusterdesc
```

[43] Master-worker imager, where master refers to the front end, and worker to the compute nodes
[44] Ger van Diepen, diepen@astron.nl
[45] You can copy this file from `/home/rol/.casarc`

If you want to run the CASA imager, use the `-casa` option:

```
> mwimager -casa mwimager.parset imaging.clusterdesc
```

The imager can take 10 seconds to several hours, depending on the size of the image you are trying to make. This is because it needs to correct for the w-plane, which becomes increasingly more computationally intensive for larger images. Therefore, always try and start with a small image.

Some parameters to create a relatively quick, small, and crude image (see e.g. Fig. 17) are given below.

## 7.2 Parset files

Below is an example parset for the MWimager[46].

```
############################################################
# mwimager.parset 1

Images.stokes = [I]
Images.shape = [512, 512]
Images.cellSize = [20, 20]
Images.ra = 08h13m36.06
Images.dec = 48.13.02.25
Images.directionType = J2000
Solver.type = Dirty
Solver.algorithm = Hogbom
Solver.niter = 100
Solver.gain = 0.2
Solver.verbose = True
Solver.scales = [0, 3]
Gridder.type = WProject
Gridder.wmax = 10000
Gridder.nwplanes = 257
Gridder.oversample = 1
Gridder.maxsupport = 40
Gridder.limitsupport = 0
Gridder.cutoff = 0.001
Gridder.nfacets = 1
Gridder.nchan = 1
Gridder.padding = 1
dataset = <file name>.gds
datacolumn = CORRECTED_DATA
minUV = 1.0
ncycles = 0
restore = True
restore_beam = [0.003, 0.003, 0]
```

This creates a Stokes I (total intensity) image, $512 \times 512$ pixel in size, with every pixel sampling 20 arcsec. The image size needs to be kept reasonably small in order for imaging to only take a short

[46] An example of this parset file can be found in ~pizzo/EXAMPLES/Parset

time. The location of the centre of the image needs setting specifically (to be the position of your target source). As the parameters are set above, no cleaning is performed and clean components are therefore not restored.

If you are using international baselines, the increased baseline length will require you to increase the values of `wmax` and `maxsupport` (and the processing time will increase accordingly), or you'll get an error when running the imager. Set these values to e.g. `wmax = 50000` and `maxsupport = 1024`, and try again.

Some further notes:

nwplanes The number of nwplanes has to be odd, While the ASKAP imager will work with an even number of images, the CASA imager requires an odd number of w-planes.

restore Set this to `true`, even for a dirty image. This appears to be some oddity in the ASKAP imager: when `restore = False`, you will end up with an image with only zeros in it.

ncycles When trying to clean the image, do not forget to set this value to 1 or higher: this is the number of major clean cycles (compared to `niter` for the number of minor clean cycles.) Note that the CASA imager ignores `ncycles`, but the ASKAP imager does not.

output files Both the ASKAP and CASA imager will produce a number of output files (unfortunately with different naming conventions, though this will avoid overriding files). These files contain e.g. the clean model, the PSF model used in cleaning, a weights image, a dirty image, and a restored image.

Imaging can take up a lot of memory, especially when creating a large image. It is therefore possible for the processing cluster to run out of memory if multiple subbands are imaged at the same time, on the same machine. Be sure to keep an eye out on this. Either use `top` to interactively trace the memory usage, or use a command like `ps xo "%cpu,%mem,user,args" | grep -E '(\-\-\-lce|cimager \-inputs)'` for a quick look.

If `MWimager` is killed, `cimager` could be still running on one or more compute nodes, consuming memory and CPU power. Check if this is the case and also kill any `cimager` job before re-running `MWimager` again; e.g. `killall cimager`.

## 7.3 Cleaning

Deconvolution of an image has not yet been extensively tested. The essential parameters are the (obvious) following ones (others as above, or whatever preferred):

```
############################################################
# mwimager.parset 1

Solver.type = Clean
Solver.algorithm = Hogbom
Solver.niter = 100
Solver.gain = 0.2
Solver.verbose = True
Solver.scales = [0, 3]
ncycles = 1
restore = True
restore_beam = [0.003, 0.003, 0]
```

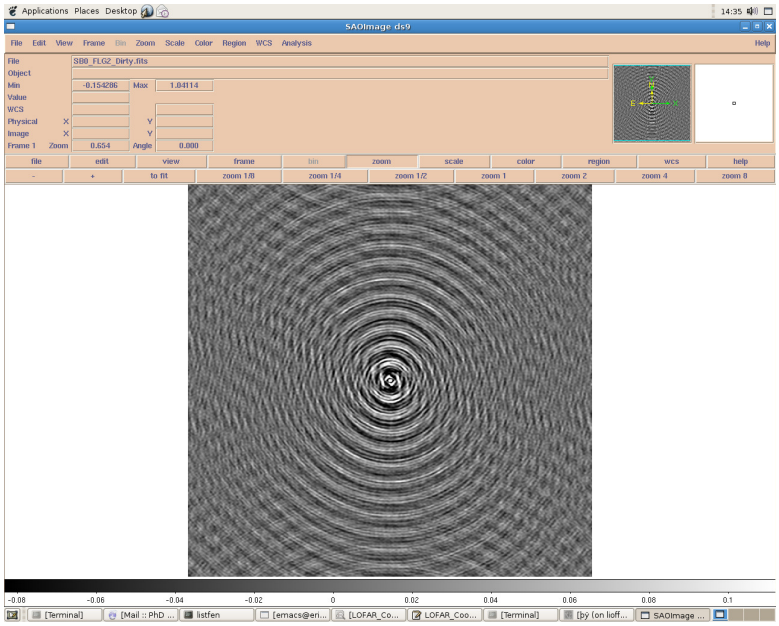


Figure 17: A simple image of 3C196 using the prescription above.

Algorithms that can be used are Hogbom and Multiscale (note: case-sensitive!). For the latter, you will need to set the scales using `Solver.scales`. The default is `[0, 3, 10]`.

The restoring beam has to be specified manually. The scale is in degrees (although one can append `arcsec` after the value), so the above example has a circular restoring beam of 10 arcseconds radius and a position angle of 0.

Do not forget to set `ncycles` to at least 1: this is the number of major cycles.

The cleaning will clean the entire image, or rather, the inner quarter. If you want to clean more, increase the image size or use the `padding` parameter; you likely will have to increase the `maxsupport` parameter (and memory and CPU usage will go up).

## 7.4 Facetting

When creating a large image, memory consumption may become an issue. To avoid this, one can use facetting. This will, however, increase the computation time by a large amount, so is only recommended when really needed.

To use facetting, set the image shape to the *total* image size, and specify the number of facets:

```
################################################################
# mwimager.parset 1

Images.nfacets = 11
Images.shape = [2816, 2816]
```

The above will produce an image using 11 by 11 facets (so 121 total), each facet having a size of 256 pixels on a side (note, however, that a 2816 x 2816 pixel image can still be easily created using w-projection without facetting).

If you are using facetting, you probably have to make sure the cleaning process is used correctly: this

communicates across the facets, so sidelobes get cleaned. But since only the inner section of a facet gets cleaned, setting the `padding` parameter to 2 or more will be essential.

Once you have started off the imager for your large image, have a check that things do not run out of memory, and then you will probably want to do something else: if it is a really big image (in pixel size), it may take several hours (be sure to check occasionally if things did not stall).

## 7.5 Process information from the imager

The cimager can produce a lot of output. You may not want to see all of this on your screen, but still want to have a logging file which has all the details, to browse through after the imaging has finished. The cimager can therefore use a logging configuration file, which sets up the level of output and where the output needs to go (e.g. file and/or stdout). An example file, `/home/rol/askap.log_cfg`, can be used. This file will need to reside in the directory where the cimager process is run, so copy it to your working directory if you would like to use it. Modify it to your needs (hopefully the syntax is relatively obvious).

## 7.6 Displaying the output image

There are two simple ways to display the output image: (i) in CASA; (ii) converting it to FITS and using the viewer of your choice (e.g. `kvis`, see below).

- To display the image in CASA, type:

  ```
  > use Casa
  > casaviewer
  ```

  Select the file, and click Raster Image (on the right-hand panel) to display the image.

- To convert the image to FITS:

  ```
  > image2fits in=<input image> out=<output image>
  ```

  where `<input image>` is the output from `MWimager`, likely to be in the format `image.i.<input MS name>_Dirty`, and `<output image>` is your desired FITS file name.

  Once you have a FITS image, you can display it using the Karma[47] tool `kvis`:

  ```
  >use Karma
  >kvis
  ```

## 7.7 Identifying background source positions

The following steps allow you to compare the positions of background sources in an image with their correct locations (taken from e.g. VLSS, WENSS). To see the sources, you need to create an image with the main target (in our case 3C196) already subtracted out (e.g. Fig. 18).

- Convert the image to `.fits` format

[47] http://www.atnf.csiro.au/computing/software/karma/

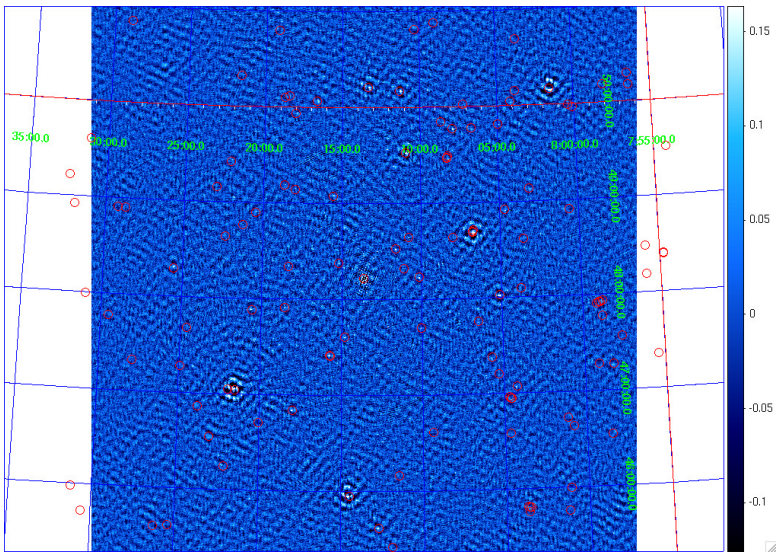



Figure 18: This is a 17 subband, 5 station image made of the 3C196 field (3C196 subtracted out) during Busy Week 1. Background sources are clearly apparent. Credit George Heald.

- Open it in `ds9`
- Analysis/Catalogs/Search for Catalogues
- Enter the desired survey in the Name or Designation box and click Retrieve.
- Select the survey from the list & click Catalog.

Check carefully if the source positions agree – some of the images produced at Busy Week 1 indicated a rotation in the image, which may have been due to a NDPPP error.

## 7.8 Running MWimager in a distributed fashion

MWimager can be run in a distributed fashion on the cluster nodes in a similar way to BBS. Use the `.gds` file as the input in the parset file.

## 7.9 Imager troubleshooting

(errors are broken across lines, indicated with a backslash)

- An error along the lines of:

```
MeasIERS::fillMeas(MeasIERS::Files, Double) (file measures/Measures\
/MeasIERS.cc, line 309): Requested data table TAI_UTC cannot be found\
 in the searched directories:
./
./data/
/home/rol/aips++/data//ephemerides/
MeasTable::dUTC(Double) (file measures/Measures/MeasTable.cc,\
 line 6295): Cannot read leap second table TAI_UTC
```

```
Unused gridder
Unexpected exception in /opt/LofIm/daily/Mon/askapsoft/bin/cimager:\
 2010-04-05 22:01:20 SEVERE MeasTable::dUTC(Double) (file\
 measures/Measures/MeasTable.cc, line 6295) Cannot read leap\
 second table TAI_UTC
```

This indicates a missing `.casarc` file in your home directory. See instructions at the start of the imaging section.

- ```
  Askap error in /opt/LofIm/daily/Mon/askapsoft/bin/cimager:\
   Overflowing convolution function for w-plane 0 - increase\
   maxSupport or decrease overSample; support=5 oversample=1 nx=10\
   ('support*itsOverSample<nx/2' failed) (thrown in gridding/\
  WProjectVisGridder.cc:268)
  ```

  You will need to increase the maxsupport parameter, or decrease the oversample parameter. Since this error shows up pretty early in the process, you can increase maxsupport by trial by steps of 2, not by 10 (the imager will run when increasing by a factor of 10, but may be much slower than a successful run after an increase of only 2).

- ```
  Askap error in /opt/LofIm/daily/Mon/askapsoft/bin/cimager: W scaling\
   error: recommend allowing larger range of w, you have w=-21.1118\
   wavelengths ('result > -1' failed) (thrown in gridding/\
  WDependentGridderBase.cc:88)
  ```

  This error probably occurs if you are using data with long baselines (e.g., international stations). If your default `wmax` is 10000, try increasing to 50000.

- If you are running the MWimager scripts from your own directory, i.e. copied from a commissioner at Busy Week 1, and you encounter error messages about not being able to find things in /data/ker/... then you need to edit the scripts, and change the paths to your own directory. However it is simpler (and preferable) to follow the method above for running MWimager, in order to use the most recent version of the software.

- ```
  > /opt/casacore/bin/image2fits in=image out=out.fits
  /opt/casacore/bin/image2fits: error while loading shared libraries:
  libcasa_lattices.so: cannot open shared object file: No such file
  or directory
  ```

  Solution: your environment is incorrectly set. Use either `use LofIm` or use the script at `/home/rol/sw/bin/image2fits` instead (note: the latter doesn't require the `in=` and `out=` keywords).

- ```
  > /opt/casacore/bin/image2fits in= image-name out=out.fits
  > Input::Put: parameter image.<image name>_Dirty is unknown
  ```

  Solution: This command is both space- and case-sensitive, check that you have entered it correctly.

- ```
  > log4cplus:ERROR No appenders could be found for logger
    (LCS.Common.EXCEPTION)
  > log4cplus:ERROR Please initialize the log4cplus system properly
  > Unable to open file /home/<user login>/MWimager.parset
  ```

  Solution: check that you have copied the `mwimager.parset` file to your working directory.
  ```

# 8 Practical examples

## 8.1 Calibration of 3C66 data obtained with LOFAR

In this section we will describe a way to get started calibrating the field centered on the radio source 3C66 observed by LOFAR. The data reduction is specifically written with dataset ID L2010_07096 in mind. With the exception of the CASA imager described at the end, everything is done using LOFAR software tools. The people who have contributed to this data reduction are George Heald, Ger de Bruyn, Björn Adebahr, and Monica Trasatti. The final image is reported in Fig. 19.

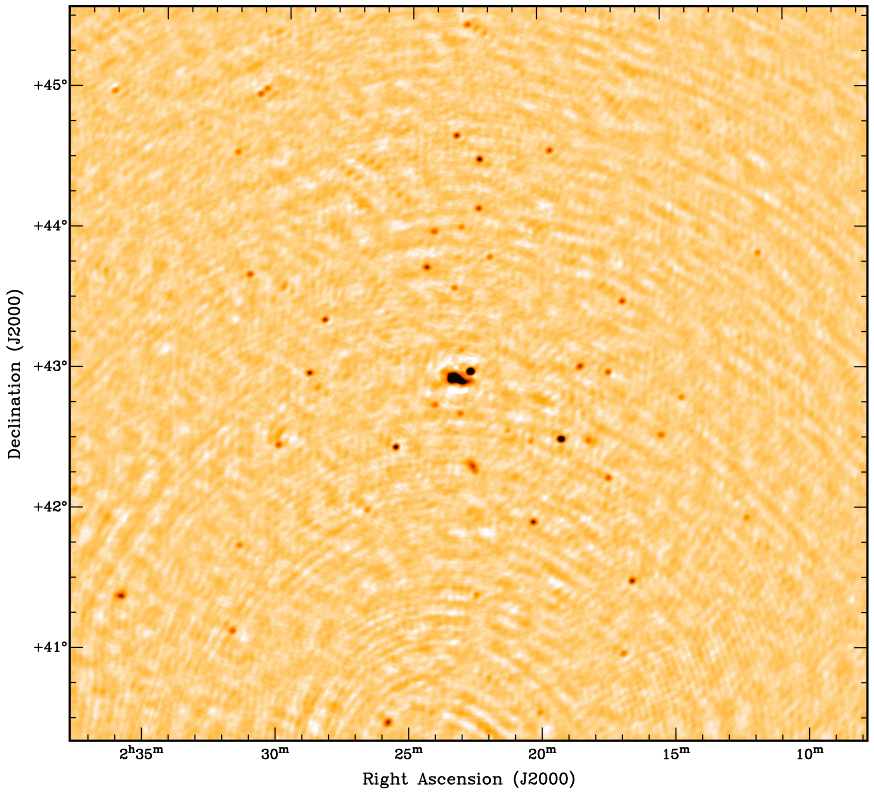



Figure 19: The field of 3C66 observed by LOFAR. The image was formed from a single sub band at 138 MHz, and has a dynamic range of 560. The ring-shaped artifacts are caused by 3C65, which is located just beyond the southern edge of the image.

### Data description

The data are in the high band and cover approximately the frequency range 125–174 MHz. The system was set up in its HBA_DUAL mode so that all HBA ears in the core are correlated separately. Thus, although only 16 stations (11 core + 5 remote) were used for the observation, there appear to be 27 stations in the Measurement Sets. The duration of the observation is 6 hours, and the integration time is 3 seconds. Each of the 248 sub bands contains 256 channels and spans 0.2 MHz. All four polarizations are recorded.

### Flagging and data compression

NDPPP can be used to flag and compress the data. The following settings are suggested for this data

reduction. It should be noted that we flag baselines shorter than 400 meters, and this step may be unnecessary.

```
########################################
#
#     NDPPP.parset
#
########################################

msin = /net/sub4/lse010/data2/L2010_07096/SB63.MS
msin.startchan = 8
msin.nchan = 240
msin.datacolumn = DATA     # is the default

msout = "SB63_DPPP.MS"
msout.datacolumn = DATA    # is the default

steps = [uvwflag,count,flag1,count,avg1,flag2,count]

uvwflag.type=uvwflagger
uvwflag.uvmmin=400

flag1.type=madflagger
flag1.threshold=3
flag1.freqwindow=31
flag1.timewindow=5
flag1.correlations=[0,3]   # only flag on XX and YY

avg1.type = squash
avg1.freqstep = 16
avg1.timestep = 1          # is the default

flag2.type=madflagger
flag2.threshold=2
flag2.timewindow=51
```

Note that the entry `flag2.timewindow=51` can be replaced by larger values up to the total duration of the observation, depending on the source structure in the field. This can help in removing RFI deriving from sunrise/sunset. In some cases (for instance those with a bright dominant point source), the amplitude is not expected to vary very much with time. By not increasing the `timewindow` parameter, data associated with sunrise/sunset may not be flagged because the associated rise and fall of the visibility amplitudes are too slow.
Ger de Bruyn recommends a compression of a factor of 16 in frequency. By discarding the first and last 8 channels (as recommended above), 15 channels would be present in the output sub bands. With this set up, it should be possible to subtract 3C65[48] more completely than we were able to manage (it is several degrees away and we induced significant bandwidth smearing by averaging so many channels).

[48] We do not describe here how to go about subtracting 3C65, but future versions of the cookbook will cover this topic.

It is worth noting that the AOFlagger (see Sect. 5) has recently proved to give much better flagging results than NDPPP. Therefore, you could alternatively use this flagger and then NDPPP only for the data compression. A few sub bands processed in this way are available on lce029 at the location `/data/safe/3C66data`.

After running AOFlagger and/or NDPPP, you may want to load CASA and use the command `clearcal`, which will prevent the CASA imager from overwriting the gain-corrected data column (`CORRECTED_DATA`) produced by the calibration[49]:

```
taskname            = "clearcal"
vis                 =  "SB63_DPPP.MS"
field               =  ""
spw                 =  ""
```

## Preliminary model and calibration

The preliminary model does not have to be as perfect as it had to be when LOFAR consisted of about 5–7 stations. Then it was essential to provide BBS with a first calibration model that was extremely close to reality.

The model that we have used to get started is the following:

```
# (Name, Type, Ra, Dec, I, Q, U, V, ReferenceFrequency='60e6', \
     SpectralIndexDegree='0', SpectralIndex:0='0.0', MajorAxis, \
     MinorAxis, Orientation) = format
# The above line defines the field order and is required.

3C66A, POINT, 02:22:39.6115, +43.02.07.799, 18.46, 0.0, 0.0, 0.0, '138e6'
3C66B, GAUSSIAN, 02:23:18.4, +42.59.37.2, 37.0, 0.0, 0.0, 0.0, '138e6',\
                '0', '0.0', 200.0, 112.5, 96.3
```

Single lines are spread over multiple lines separated by backslash. The flux of 3C66A should be more or less correct. The settings for the Gaussian description of 3C66B were obtained by calculating a power-law interpolation between the WENSS and NVSS images of the source, and subsequently doing a by-eye estimation of the axis ratio and position angle. We estimated that the total flux of the source was approximately twice the peak flux of 3C66A, which was sufficient to produce a reasonable first image.

In general, for faint sources it is best to use reference values from NED[50]. Extended structures like 3C66B seem to be better represented by Gaussians. A good way to proceed is to try to find an image of the target field in NED and apply a Gaussian to it, simulating the effect of the LOFAR station beam. This will help you to estimate the source flux values.

It should be noted that for more than one strong source in the field, the relative flux between the sources is a critically important factor. Moreover, for the first self-calibration it seems that that it is

[49] Strictly speaking, this is no longer necessary if you use `calready=False` in the CASA imager settings, but if you forget and leave `calready=True`, then CASA will overwrite your `CORRECTED_DATA` column unless you first run `clearcal`.

[50] `http://nedwww.ipac.caltech.edu/`

better to start with a model containing about half of the flux in the field and include only the strongest sources in it. Then, when proceeding in self calibration iterations, more flux and more sources can be added.

The calibration has been performed by using the following BBS parset file:

```
################################################
#
#   BBS.parset
#
################################################

Strategy.ChunkSize = 0
Strategy.Steps = [solve, correct]

Step.solve.Operation = SOLVE
Step.solve.Model.Sources = []
Step.solve.Model.Gain.Enable = T
Step.solve.Model.Cache.Enable = T
Step.solve.Solve.Parms = ["Gain:0:0:*","Gain:1:1:*"]
Step.solve.Solve.CellSize.Freq = 0
Step.solve.Solve.CellSize.Time = 1
Step.solve.Solve.CellChunkSize = 10
Step.solve.Solve.Options.MaxIter = 20
Step.solve.Solve.Options.EpsValue = 1e-9
Step.solve.Solve.Options.EpsDerivative = 1e-9
Step.solve.Solve.Options.ColFactor = 1e-9
Step.solve.Solve.Options.LMFactor = 1.0
Step.solve.Solve.Options.BalancedEqs = F
Step.solve.Solve.Options.UseSVD = T

Step.correct.Operation = CORRECT
Step.correct.Model.Sources = []
Step.correct.Model.Gain.Enable = T
Step.correct.Output.Column = CORRECTED_DATA
```

Note that `Step.solve.Solve.CellSize.Time = 1` represents the size of the solution interval for calibration in units of integration time. It may be useful to start with a high solution interval (like 300) and lower it to 1 during the the self-calibration iterations. For example, you could successively use values of 300, 100, 50, 10, and 1. This seems to be an important point, as for a complex field the first model used for calibration is not that accurate.

After running BBS, you should inspect the solutions using `parmdbplot.py`, and the `CORRECTED_DATA` column using `uvplot.py` (see Sect. 6.5.1 and 2.3, respectively). Inevitable bad solutions producing corrected data with very high amplitude will appear. To flag the data suffering from bad solution, you should run NDPPP, preferably with the following settings:

```
msin = SB62_DPPP.MS
```

```
msin.datacolumn = CORRECTED_DATA
msout=

steps = [flag1]

flag1.type=madflagger
flag1.threshold=3
flag1.timewindow=501
```

## Imaging

To image the field, you can load CASA again and use the task `clean`. This performs also good wide field imaging, as well as deconvolution. The task settings that we used are:

```
taskname            =         "clean"
vis                 =         'SB63_DPPP.MS'
imagename           =         ['iter09']
outlierfile         =           ''
field               =           ''
spw                 =           ''
selectdata          =          False
mode                =          'mfs'
nterms              =            1
reffreq             =           ''
gridmode            =         'widefield'
    wprojplanes     =          256
      facets        =            1
niter               =          2000
gain                =          0.1
threshold           =         '0.0mJy'
psfmode             =         'clark'
imagermode          =          ''
multiscale          =           []
interactive         =           True
mask                =           []
imsize              =         [2048, 2048]
cell                =     ['15.0arcsec', '15.0arcsec']
phasecenter         =           ''
restfreq            =           ''
stokes              =          'I'
weighting           =       'natural'
uvtaper             =         False
modelimage          =            ''
restoringbeam       =          ['']
pbcor               =         False
minpb               =           0.2
calready            =          True
async               =          False
```

You will notice that the `interactive` parameter is set to `True`, which allows you to fiddle around with the mask. In this mode you can also modify the number of CLEAN components and iterations on-the-fly.

## Calibration cycle

In order to self calibrate, you should now take the clean components that you just obtained and feed them back in to BBS. This can be achieved by using a program provided by Joris van Zwieten. It is available at ∼`heald/bin/casapy2bbs` and can be used through the following command:

```
> ~heald/bin/casapy2bbs iter09 iter09.catalog
```

Note that the name of the clean component image produced by `widefield` is just the string given in parameter `imagename` and all other images produced by `widefield` are called `<imagename>.*`. The output of the script, in this case `iter09.catalog`, can then be used as a new skymodel for BBS.

Following a new calibration, you can make a new image and loop as many times as you like around the major cycle: convert clean component lists to a new BBS skymodel, run BBS again, and produce a new image.

If you want to look at your image in kvis, you will have to first make a fits file:

```
image2fits in=iter09.image out=iter09.image.fits
```

To increase the sensitivity of the final image, you can combine several sub bands. Currently, this can be done in the image plane by using the scripts `average.py` or `average_weights.py`[51]. However, a better final result is obtained by combining the sub bands in the uv domain. This can be done by using Oleg Smironov's script, which can be obtained by typing from your shell:

```
> svn co svn://lofar9.astron.nl/var/svn/repos/trunk/Owlcat
```

After this command has completed, you can run:

```
> Owlcat/merge-ms.py -s output.MS SB*.MS
```

[51] Available in `/pizzo/EXAMPLES/Scripts`

# 9 Source detection: PyBDSM

This section has been written by David Rafferty, rafferty@strw.leidenuniv.nl.

## 9.1 Introduction

PyBDSM (**Py**thon **B**lob **D**etection and **S**ource **M**anagement) is a Python source-finding software package written by Niruj Mohan and Alexander Usov and currently maintained by David Rafferty (rafferty@strw.leidenuniv.nl). PyBDSM can process FITS and CASA images and can output source lists in BBS, FITS, ASCII and ds9 region formats. It can be used in the ipython environment or in python scripts. Details of the algorithms used in PyBDSM can be found in Section 2.7 of the Anaamika manual located at http://www.strw.leidenuniv.nl/~mohan/anaamika_manual.pdf (the Anaamika package includes the Fortran version of PyBDSM, as well as other related software, and is no longer being maintained).

## 9.2 Setup

Currently, PyBDSM is installed on the offline cluster in /home/rafferty/PyBDSM.[52] To initialize your environment for PyBDSM, run:

```
> use LofIm
> use PythonLibs
> source /home/rafferty/PyBDSM/init_pybdsm
```

PyBDSM should now be visible to Python and can be imported into Python scripts or ipython with the command "import bdsm."

## 9.3 Usage

A standard analysis is done using the bdsm.process_image() task. The general call is (with defaults shown):

```
img = bdsm.process_image(input_file, beam=None, freq=None,
thresh_isl=3.0, thresh_pix=5.0, thresh_gaus=None, use_rms_map=None,
rms_box=None, extended=False, gaussian_maxsize=10.0)
```

where "input_file" is the name of the image and the other (optional) arguments are defined as follows:

**beam** The restoring beam, defined as (major axis FWHM [deg], minor axis FWHM [deg], angle east from north of major axis [deg]). If not given (i.e., beam = None, the default), the beam is read from the image header.

**freq** The frequency in Hz of input image (use a list if input image is a cube: e.g., [30e6, 35e6, 72e6]). If not given (i.e., freq = None, the default), the frequency is read from the image header.

**thresh_isl** The threshold for island detection in number of times the background rms (default = 3.0).

[52] Eventually, PyBDSM will be part of the nightly build; at that point, it may be initialized with "use LUS."

**thresh_pix** At least one pixel in each island must exceed this threshold of number of times the background rms (default = 5.0).

**thresh_gaus** The peak threshold in sigma for adding more Gaussians to an island. If not given (the default), the threshold will be estimated automatically.

**use_rms_map** Use a 2-D map for the background rms or a constant value? None/True/False (default = None = determine automatically). If set to None or True, the rms_map is calculated using a moving-window algorithm, where the mean and rms are calculated within a window of a size (box_size $\times$ box_size), and the window is stepped withing the image by steps of size box_step. The box_size and box_step can be calculated automatically or set with the "rms_box" option.

**rms_box** The box size and box step to use when constructing the rms map, defined as (box_size [pixels], box_step [pixels]). If set to None (the default), PyBDSM estimates the box automatically (in this case, see the log file for the box information).

**gaussian_maxsize** Maximum allowable size of fitted Gaussians in number of beam areas (default = 10.0).

**extended** If significant extended emission is expected, try setting this option to True (default = False). This option simply sets "use_rms_map = False" (to use a constant rms rather than a 2-D map that may be strongly biased by the extended emission) and "gaussian_maxsize = 100" (to allow extended Gaussians to be fit).

A few other, more specialized, options are available. Enter "bdsm.process_image?" in ipython for a description of all current options.

If the fit is successful, PyBDSM will return an "Image" object (in this example named "img") which contains the results of the fit (among many other things). A log file with additional information about the fit will be saved in the current directory. A number of tools are available for examining the fit and writing out the source list, residual image, etc. These tools are methods of the Image object returned by bdsm.process_image() and are described below:

**img.showfit()** This method shows a quick summary of the fit by plotting the input image with the islands and Gaussians found, along with the model and residual images.

**img.showrms()** Either prints the rms value used (if constant) or shows the 2-D rms map, along with the input image with the islands and Gaussians overlaid.

**img.write_model_img(filename)** Write the model image to a FITS file.

**img.write_resid_img(filename)** Write the residual image to a FITS file.

**img.write_rms_img(filename)** Write the background 2-D rms image to a FITS file.

**img.write_ch0_img(filename)** Write the ch0 image (the image used for detection) to a FITS file.[53]

**img.write_gaul(filename, format)** This method writes the Gaussian list to the given filename with the given format. Available formats are "ascii" (simple ASCII text file; the default), "bbs" (BBS sky model file), "fits" (FITS file), and "ds9" (ds9 region file). The information included in the output varies depending on the format used: with the ASCII and FITS formats, all Gaussian parameters are included; with the BBS and ds9 formats, only a subset of parameters are included. Additionally, in the BBS and ds9 output files, sources with sizes smaller than the beam are denoted as point sources.

[53]Typically the ch0 image is just the original input image. However, if the input image is a 3- or 4-D cube, the ch0 image is the result of an average over the frequency axis of the cube.

For details of the entries in the Gaussian lists made with img.write_gaul(), see Section 9.5.

## 9.4 Examples

### 9.4.1 Simple Example

A simple example of using PyBDSM on a LOFAR image (an HBA image of 3C61.1) is shown below.

```
> ipython
Python 2.5.2 (r252:60911, Jan 20 2010, 23:14:04)
Type "copyright", "credits" or "license" for more information.
IPython 0.8.1 -- An enhanced Interactive Python.
?       -> Introduction to IPython's features.
%magic  -> Information about IPython's 'magic' % functions.
help    -> Python's own help system.
object? -> Details about 'object'. ?object also works, ?? prints more.

In [1]: import bdsm
In [2]: img_3C61 = bdsm.process_image('sb48.fits')
Opened sb48.fits
Image size : (256, 256) pixels
Beam shape (major, minor, pos angle) : (0.002916, 0.002654, -173.36) degrees
Number of islands found : 4
Fitting island #3...
Number of Gaussians found : 12
In [3]: img_3C61.showfit()
```

The figure made by img_3C61.showfit() is shown in Figure 20. In the plot window, one can zoom in, save the plot to a file, etc. The list of best-fit Gaussians found by PyBDSM may be written to a file for use in other programs, such as BBS, as follows:

```
In [4]: img_3C61.write_gaul(format='bbs')
--> Wrote BBS sky model sb48.pybdsm.gaul.sky
```

The output Gaussian lists contain source positions, fluxes, etc. (see Section 9.5 for details).

### 9.4.2 Image with Artifacts

Occasionally, a PyBDSM run with the default parameters does not produce good results. For example, if there are significant deconvolution artifacts in the image, the "thresh_isl," "thresh_pix," or "rms_box" parameters might need to be changed to prevent PyBDSM from fitting Gaussians to such artifacts. An example of running PyBDSM with the default parameters on such an image is shown in Figure 21. It is clear that a number of spurious sources are being detected. Simply raising the threshold for island detection (using the "thresh_isl" parameter) would remove these sources but would also remove many real but faint sources in regions of low rms. Instead, by setting the "rms_box" parameter to better match the typical scale over which the artifacts vary significantly, one obtains much better results. In this example, the scale of the regions affected by artifacts is approximately 20 pixels, whereas PyBDSM used a "rms_box" of 61 pixels when run with the default parameters (as reported in the log file), resulting in an rms map that is over-smoothed. Therefore, one should call bdsm.process_image() as follows:

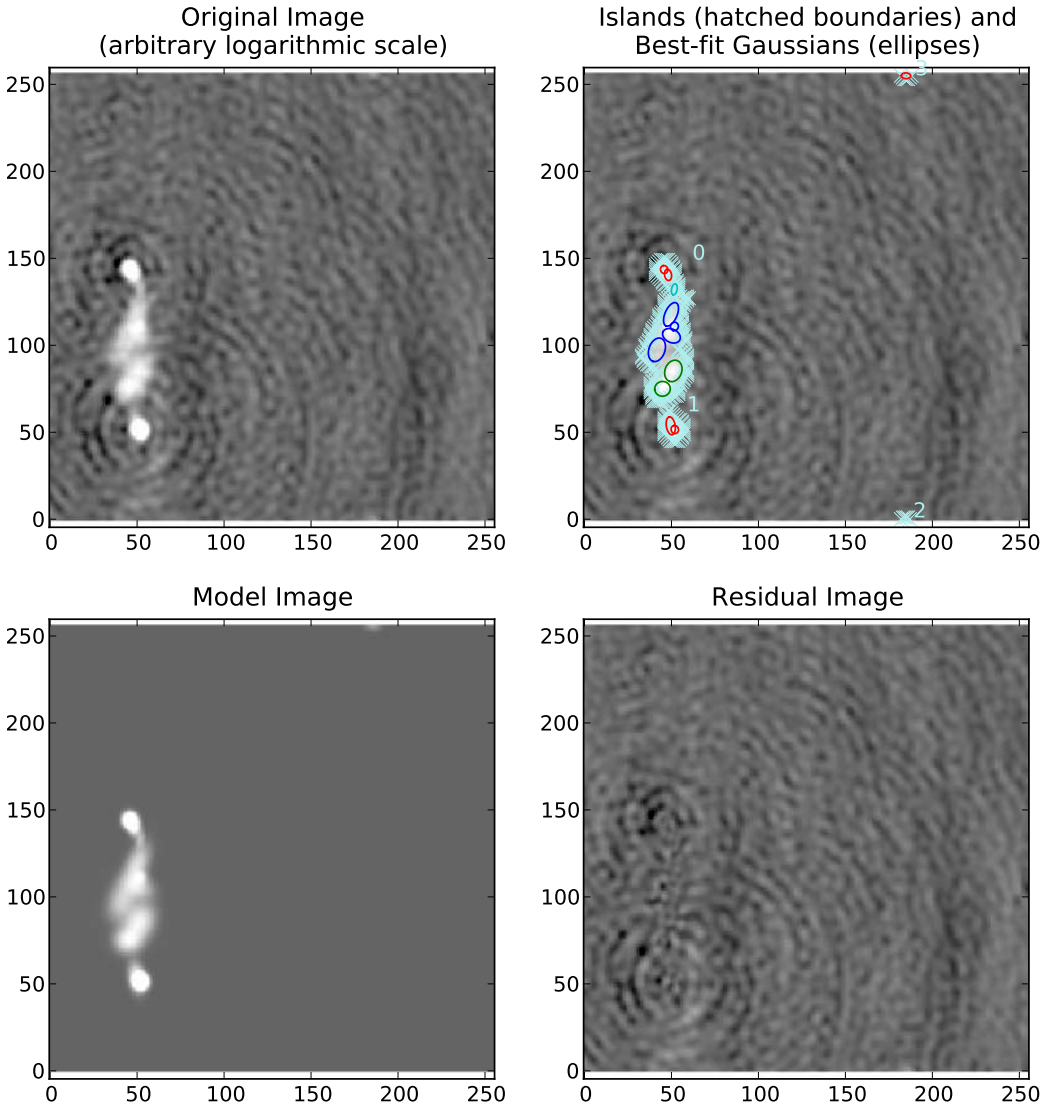



Figure 20: Output of img_3C61.showfit(), showing the original image with and without sources, the model image, and the residual (original minus model) image. Boundaries of the islands of emission found by PyBDSM are shown in light blue (with the corresponding island index), and the fitted Gaussians are shown for each island as ellipses (the sizes of which correspond to the FWHMs of the Gaussians).

```
In [2]: img_3C196 = bdsm.process_image('3C196.image', rms_box=(20, 10))
```

Here the rms map is computed using a box of 20 pixels in size with a step size of 10 pixels (i.e., the box is moved across the image in 10-pixel steps). See Figure 22 for a summary of the results.

### 9.4.3 Image with Extended Emission

If there is extended emission that fills a significant portion of the image, the background rms map will likely be biased high in regions where extended emission is present (this can be checked with the "img.showrms()" method). Setting "extended=True" in the bdsm.process_image() call will force PyBDSM to use a constant rms value across the whole image (i.e., "use_rms_map=False") and will allow for large Gaussians ("gaussian_maxsize" is set to 100 times the beam area). Depending on the source structure, the "thresh_isl", "thresh_pix", and "thresh_gaus" parameters may also have to be adjusted as well to ensure that PyBDSM finds and fits islands of emission properly. An example analysis of an image with significant extended emission is shown in Figure 23.

## 9.5 Gaussian List Description

The ASCII and FITS Gaussian lists made with img.write_gaul() include the following entries for each Gaussian:

**Gaul_id** Index number of the Gaussian.

**Island_id** Index number of the island the Gaussian belongs to. Islands can be fit with multiple Gaussians.

**Flag** If flag is zero, the Gaussian is valid. For the meaning of nonzero flags, see Section 2.5.2 of `http://www.strw.leidenuniv.nl/~mohan/anaamika_manual.pdf`.

**Total_flux** Total flux, in Jy.

**Err_total_flux** Error in total flux, in Jy.

**Peak_flux** Peak flux density, in Jy/beam.

**Err_peak_flux** Error in peak flux density, in Jy/beam.

**RA** R.A. of center of fitted Gaussian, in degrees.

**Err_RA** Error in R.A., in degrees.

**DEC** Dec of center of fitted Gaussian, in degrees.

**Err_DEC** Error in Dec, in degrees.

**Xpos** X-position of center of fitted Gaussian, in pixels.

**Err_xpos** Error in x-position, in pixels.

**Ypos** Y-position of center of fitted Gaussian, in pixels.

**Err_ypos** Error in y-position, in pixels.

**Bmaj_fw** FWHM of major axis of fitted Gaussian, in arcsec.

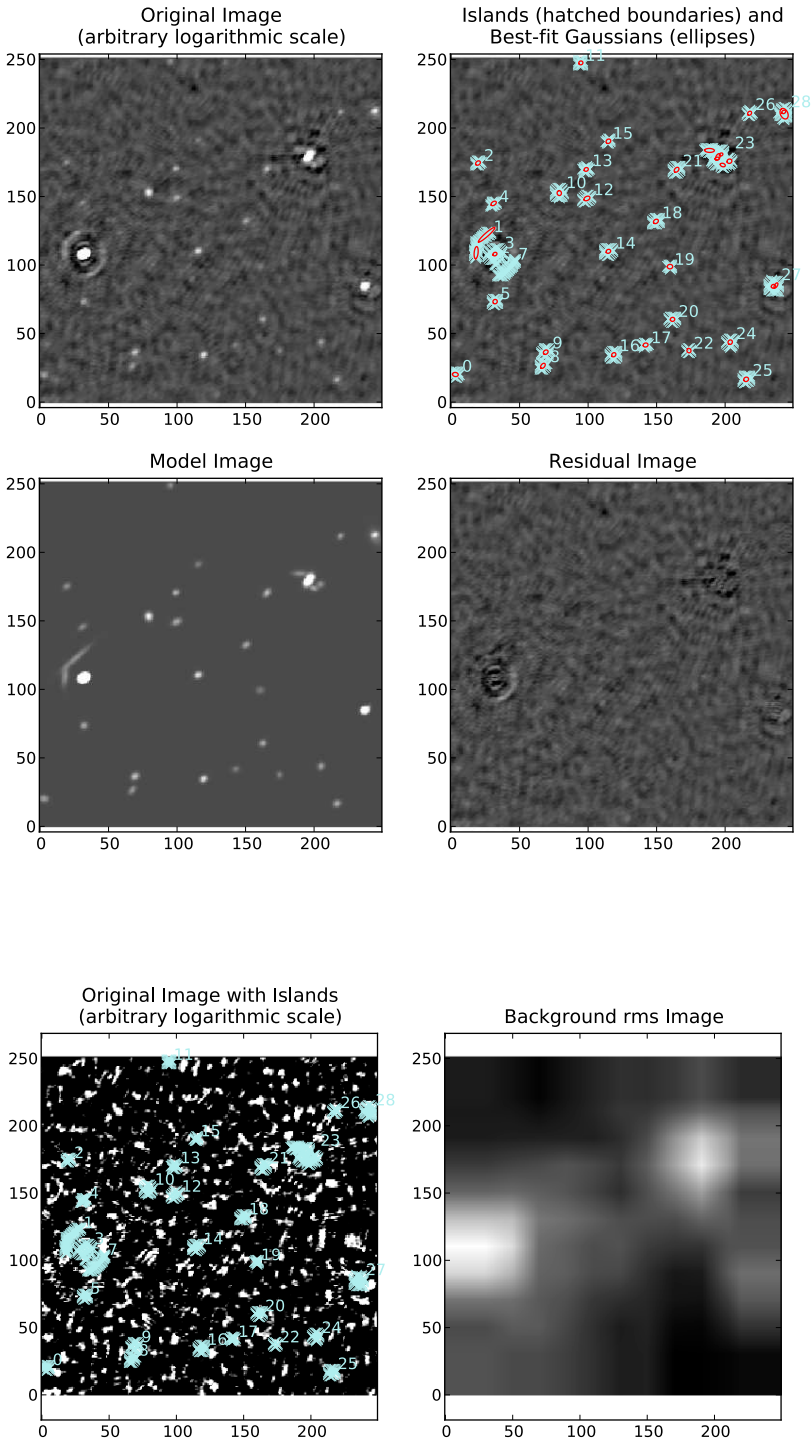



Figure 21: Example fit with default parameters of an image with strong artifacts around bright sources. A number of artifacts near the bright sources are picked up as sources. The background rms map for the same region (produced using "img.showrms()") is shown in the lower panel: the rms varies fairly weakly across the image, whereas ideally it would increase strongly near the bright sources (reflecting the increased rms in those regions due to the artifacts).

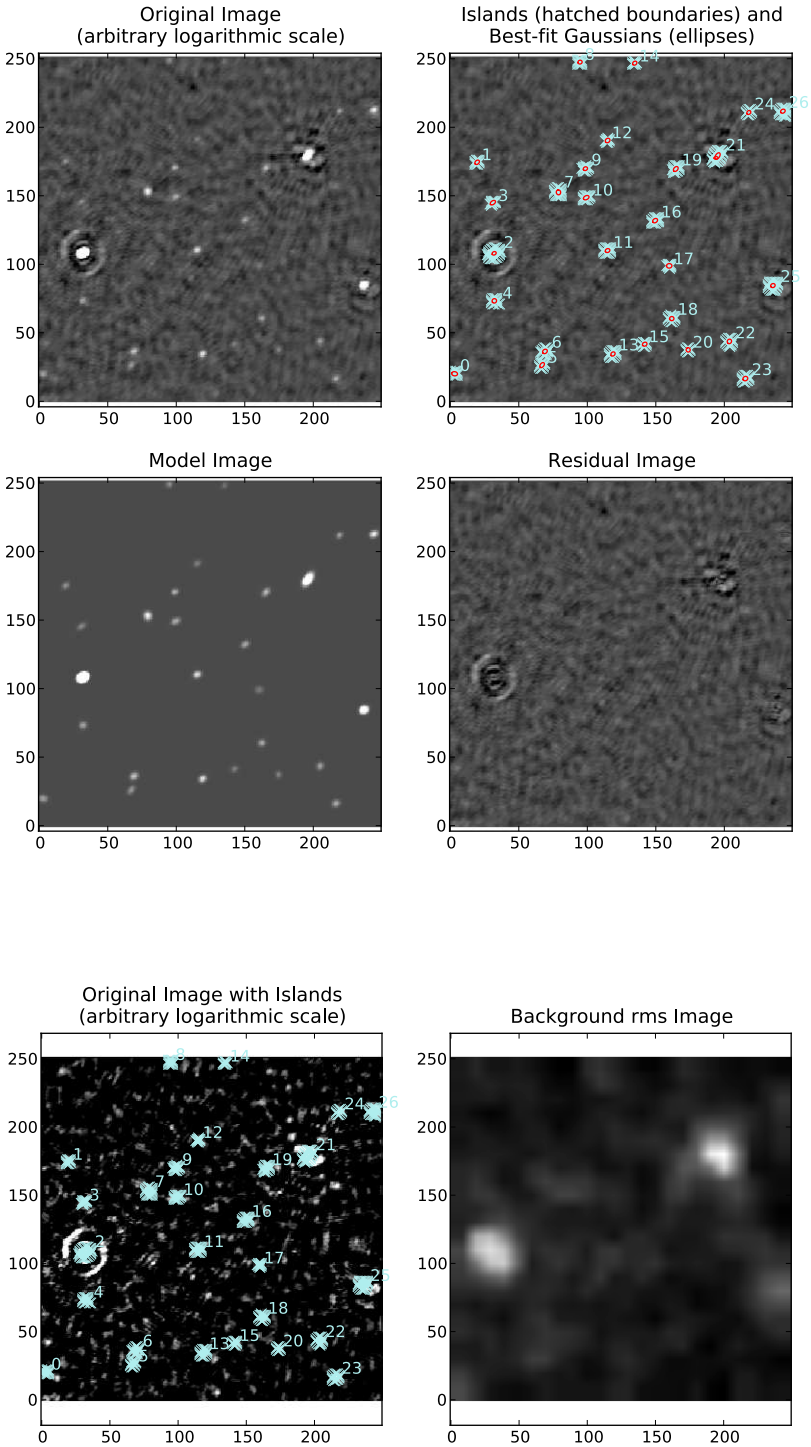



Figure 22: The same as Figure 21. but with "rms_box=(20, 10)". The rms map now varies on scales similar to that of the regions affected by the artifacts, and both bright and faint sources are recovered properly.

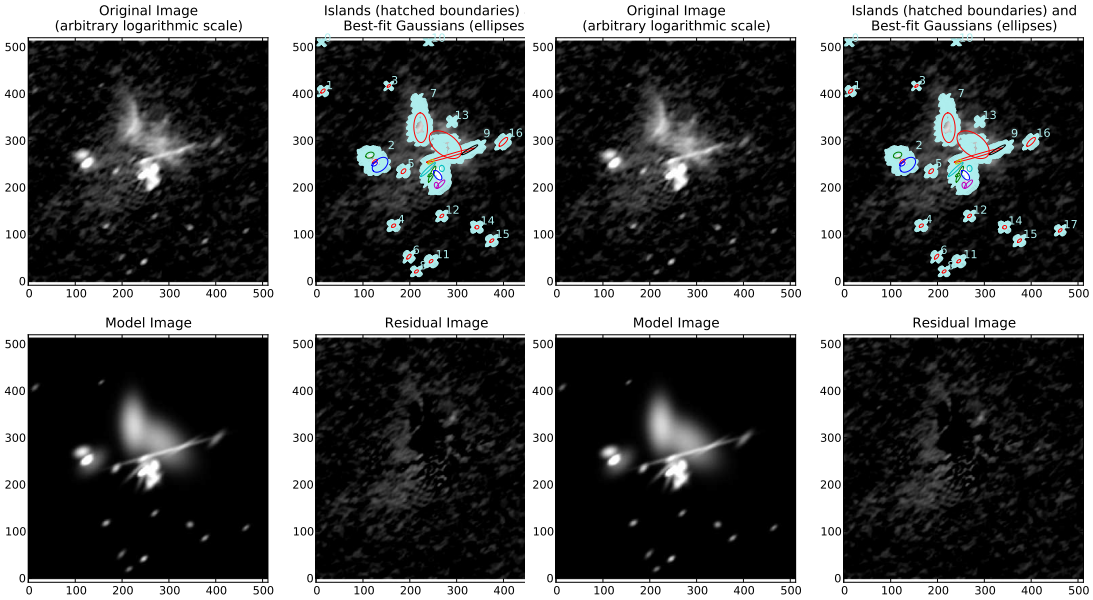



Figure 23: Example fit of an image of A2256 using the default settings of bdsm.process_image() (*left*) and using the "extended = True" option (*right*).

**Err_bmaj** Error in FWHM of major axis, in arcsec.

**Bmin_fw** FWHM of minor axis of fitted Gaussian, in arcsec.

**Err_bmin** Error in FWHM of minor axis, in arcsec.

**Bpa** Position angle of fitted Gaussian, east from north, in degrees.

**Err_bpa** Error in position angle, in degrees.

# 10 Running the pipeline in distributed mode

An automatic way of performing a LOFAR data reduction exists and a few people use it successfully. For all the other users, distributed methods for running BBS & MWImager have been covered in the relevant sections(Sect. 6.11 and Sect. 7.8). Regarding NDPPP, if you need to flag/average all the subbands of an observation, please contact Roberto Pizzo (pizzo@astron.nl) or John Swinbank (swinbank@transientskp.org), who can help you with that. A documentation on how to run the pipeline yourself will become available soon.

# 11 Poorly documented stages in this cookbook

Sections of this cookbook that are known to need further updates include:

Section 4 (NDPPP):

- 'Optimal' flagging methods are still wanted.

Section 6 (BBS):

- Better sky models are needed
- An update to the solution-based flagging script is needed.

Section 7 (MWImager):

- MWImager needs to be properly tested by the users and then more documentation could be added.

Section 10 (Distributed pipeline operations):

- Documentation on how to run the imaging pipeline needs to be added

General:

- Improved images / screen grabs are needed to illustrate important stages of the pipeline – ideally, one specific measurement set and all the required parameters to create an example image, so that later commissioners can attempt to replicate everything successfully.

# 12 General troubleshooting

The process of commissioning by definition involves a great deal of troubleshooting! Here we list some non software-specific generic errors which have been encountered, and some checks/solutions. Please, suggest updates of this section throughout the busy weeks, so as we have an up to date record of common errors/problems which people encounter – these are by far the most time consuming aspect of running the pipeline.

## 12.1 Common errors

Common errors can happen during the processing of the data. The most typical ones could be due to:

- A full home directory. The disk with the home directories (at time of writing) is still fairly full. Don't copy Measurement Sets to your home directory, use scratch instead. Check:

  ```
  > df -h
  ```

  and then possibly

  ```
  > du -h ~
  ```

- Not having the `.tcsh` file in your home directory on login. Check: Ensure .tcsh is present in your home directory.
- Failing to initialise the LOFAR environment/Pythonpath, or running into commands or libraries that are not found. See if a version of an earlier day works for you:

  ```
  use LofIm <day>
  ```

## 12.2 Version errors

Each section of the pipeline is still being actively worked on, and updated versions of NDPPP, BBS, and MWImager are incorporated into the daily builds. However, this also comes with the disadvantage that what may have worked yesterday, may not necessarily work today! If you encounter a crash with NDPPP/BBS/MWImager, with a previously working set of parameters, try running the software version from the day before:

```
(14th October)
> NDPPP
New error appears, due to NDPPP being actively worked on...

(Use the version from the day before)
> use LofIm <yesterday>
>NDPPP
NDPPP terminated normally
```

and similarly for BBS, MWImager. If this works, please report the crash & error message to the relevant developer, as this implies a buggy version of the software has made it into the daily build.

If you have spotted a bug, reported it to the developers and they have provided a fix that day, you may need to source the development version of the LOFAR environment again.

```
> use LofIm <yesterday>
```

# 13 Useful resources

## 13.1 Webpages

The LOFAR wiki is a key resource, and you need an account to access the software areas. You can register for an account here: `http://www.lofar.org/operations/doku.php?id=start&do=register`

The LOFAR forum is also useful for getting rapid feedback and suggestions: `http://usg.lofar.org/forum`

Essential pages on the wiki are:

Main imaging wiki page: `http://www.lofar.org/wiki/doku.php?id=software:standard_imaging_pipeline`

NDPPP: `http://www.lofar.org/wiki/doku.php?id=engineering:software:tools:ndppp`

BBS: `http://www.lofar.org/operations/doku.php?id=engineering:software:tools:bbs`

BBS parset parameters: `http://www.lofar.org/operations/doku.php?id=engineering:software:tools:bbsconfigurationsyntax`

MWimager: `http://www.lofar.org/wiki/doku.php?id=engineering:software:tools:mwimager`

cimager: `http://www.lofar.org/wiki/doku.php?id=engineering:software:tools:cimagerdoc` (useful for some of the more cryptic mwimager parameters.)

## 13.2 Useful analysis scripts

A compilation of some practical python scripts is available at the wiki web page

http://www.lofar.org/operations/doku.php?id=engineering:software:tools

You can copy them directly from `/opt/tools/cookbook`. For a few of these scripts, you will need to initialize the PyDAL libraries:

```
> use LUS
```

The scripts provided are[54]:

- autoflagger.py: flags autocorrelations in a Measurement Set
- average.py: averages images from multiple sub bands together
- average_weights.py: averages images weighting them by the inverse of their variance.
- baseline.py: plots amplitude/phase vs time/uvdistance/elevation
- CallSolFlag.py: flags calibrated data

[54] If you have other scripts that could be useful for other commissioners, please contact Roberto Pizzo at `pizzo@astron.nl`.

- closure.py: prints closure phase vs time/elevation for selected antennas
- coordinates_mode.py: routines to work with astronomical coordinates
- plot.py: inspect gain solutions
- solfetch.py: modules required for solflag.py
- solflag.py: carries out solution-based flagging
- solplot.py: modules required for solflag.py
- uvcoverage.py: plots the uv coverage for a Measurement
- plot_flag.py: plots "images" of frequency versus time on a baseline-by-baseline basis, with the pixel values equal to the visibility amplitudes
- img2fits.py: converts CASA images to fits images
- compare_gaincal.py: plots CASA and BBS gain solutions against each other for comparison. It can also plot CASA vs CASA and BBS vs BBS. Only supports gain solutions, and only if a solution was computed for each integration time
- traces.py: plots L,M tracks for the zenith, azimuth and elevation of the NCP, CasA, CygA, and the target against time for a given MS or time range. Observer location is fixed to Dwingeloo. It is easy to add other sources of interest, or to modify the observer location, but it does require editing the Python code. The script is useful to check the elevation of possible interfering sources like CasA and CygA
- casapy2bbs: written by Joris van Zwieten. Converts a clean component image produced by casa into a skymodel file readable by BBS. See also modelclip.py.
- embiggen.csh: increases the size of plotted points in postscript files. Useful when producing ps output from e.g. uvplot.py.
- lin2circ.py: given a Measurement Set with a DATA column given in XX,XY,YX,YY correlations, converts to circular correlations RR,RL,LR,LL and writes them to a column in the Measurement Set.
- modelclip.py: sorts a skymodel file with respect to Stokes I flux, and truncates the list of sources such that N% of the total flux is kept in the model (where N is specified on the command line). Useful for clean component skymodels produced by e.g. casapy2bbs.
- msHistory.py: prints information from the HISTORY table of a Measurement Set. Useful for obtaining a quick listing of the parset values used in e.g. NDPPP.
- plotElevation.py: given a Measurement Set, plots the elevation of the target source as a function of time
- split_ms_by_time.py: extracts part of a Measurement Set (selected by timerange) and writes out to a new Measurement Set. Optionally excludes selected antennas.
- uvcov.py: plots uv coverage for one or more Measurement Sets. If all Measurement Sets are for the same source at the same time (in other words are different subbands of the same observation), then use the '-s' option to save a lot of time. Do NOT use that option if the input Measurement Sets are not coincident in time.

- uvplot.py: plots data from a Measurement Set in several combinations, in a per-baseline fashion. Not as flexible as casaplotms, but should be faster.
- uvrms.py: performs RM Synthesis on the data in a Measurement Set.

In addition, there is the `msinfo` script to inspect a Measurement Set. This can be downloaded from the USG repository (`http://usg.lofar.org/svn/code/trunk/src/contrib/data_inspection/`); `msinfo` requires the `msinfo.py` and `argparse.py` Python script and module. An existing version on the cluster can be found at `/home/rol/sw/bin/msinfo`. (The aforementioned USG repostiroy directory also contains some older data inspection scripts. These may, however, not function properly anymore on the new cluster, because of outdated dependencies.)

## 13.3 Contact points

Some key contact points

- LOFAR Imaging Cookbook - Roberto Francesco Pizzo (pizzo@astron.nl)
- NDPPP - Ger van Diepen (diepen@astron.nl), Vishambhar Nath Pandey (pandey@astron.nl), and Renting (renting@astron.nl)
- AOFlagger - Andre' Offringa (offringa@astro.rug.nl)
- BBS - Joris van Zwieten (zwieten@astron.nl) and Vishambhar Nath Pandey (pandey@astron.nl)
- MWImager - Ger van Diepen (diepen@astron.nl) and Evert Rol (evert.astro@googlemail.com)
- Distributed Pipeline Running - John Swinbank (swinbank@transientskp.org)
- Pyrap/CASA - Ger van Diepen (diepen@astron.nl)

# 14 Acknowledgments

This cookbook has been written by people who are learning LOFAR data reduction (plus a couple of LOFAR-overloaded summer students), rather than by anyone who has been involved in developing the pipeline (see the wiki for that) – as such, any and everything in it might still be incorrect. Please send comments and suggestions of improvements to Roberto Francesco Pizzo (pizzo@astron.nl).

The processing described in this cookbook resulted first from the first, second, and third Imaging Busy Weeks at ASTRON, and was also based on previous suggestions from Louise Ker and Francesco de Gasperin.

The contact points for the various versions of the LOFAR Imaging Cookbook are listed below.

Version 1.0: Timothy Garn

Version 1.1: Louise Ker (lmk@roe.ac.uk)

Version 1.2: Annalisa Bonafede (a.bonafede@jacobs-university.de)

Version 2.0: Emanuela Orru' (e.orru@astro.ru.nl) & Fabien Batejat (fabien.batejat@chalmers.se)

Version 2.1: Roberto Francesco Pizzo (pizzo@astron.nl)

Version 2.2: Roberto Francesco Pizzo (pizzo@astron.nl)

Version 2.3: Roberto Francesco Pizzo (pizzo@astron.nl)

Version 3.0: Roberto Francesco Pizzo (pizzo@astron.nl)

Version 4.0: Roberto Francesco Pizzo (pizzo@astron.nl)

Version 5.0: Roberto Francesco Pizzo (pizzo@astron.nl)

Version 5.1: Roberto Francesco Pizzo (pizzo@astron.nl)

# A GNU screen

For some long running processes, as the flagging (NDPPP, see Sect. 4), the calibration (BBS, see Sect. 6) or creating a large image, it can be handy to have a session running on one of the cluster nodes, and then leave that running the background overnight. This can be done by putting the task in the background and use a command like `disown`. Sometimes, however, it is more convenient to have a program running the in the foreground. In such a case, it is difficult to log out without killing the process. GNU screen to the rescue.

GNU screen (short: screen) is a utility that creates a virtual terminal that stands on its own, and can be put in the background or foreground at will. Multiple terminals ("tabs") are also possible. A disadvantage of GCN screen is that it will not use with X-windows (forwarding) and scrolling back for previous output does not always work. For a long-during command line task, however, it is very useful.

Have a first look at screen using the help option:

```
screen -help
Use: screen [-opts] [cmd [args]]
 or: screen -r [host.tty]

Options:
-a            Force all capabilities into each window's termcap.
-A -[r|R]     Adapt all windows to the new display width & height.
-c file       Read configuration file instead of '.screenrc'.
-d (-r)       Detach the elsewhere running screen (and reattach here).
-dmS name     Start as daemon: Screen session in detached mode.
-D (-r)       Detach and logout remote (and reattach here).
-D -RR        Do whatever is needed to get a screen session.
-e xy         Change command characters.
-f            Flow control on, -fn = off, -fa = auto.
-h lines      Set the size of the scrollback history buffer.
-i            Interrupt output sooner when flow control is on.
-l            Login mode on (update /var/run/utmp), -ln = off.
-list         or -ls. Do nothing, just list our SockDir.
-L            Turn on output logging.
-m            ignore $STY variable, do create a new screen session.
-O            Choose optimal output rather than exact vt100 emulation.
-p window     Preselect the named window if it exists.
-q            Quiet startup. Exits with non-zero return code if unsuccessful.
-r            Reattach to a detached screen process.
-R            Reattach if possible, otherwise start a new session.
-s shell      Shell to execute rather than $SHELL.
-S sockname   Name this session <pid>.sockname instead of <pid>.<tty>.<host>.
-t title      Set title. (window's name).
-T term       Use term as $TERM for windows, rather than "screen".
-U            Tell screen to use UTF-8 encoding.
-v            Print "Screen version 4.00.03 (FAU) 23-Oct-06".
-wipe         Do nothing, just clean up SockDir.
-x            Attach to a not detached screen. (Multi display mode).
-X            Execute <cmd> as a screen command in the specified session.
```

The detach and attach commands are the 'background' and 'foreground' commands. Now start screen for real (from a frontend or compute node):

```
> screen
```

You will get a new terminal, but otherwise everything looks the same. You can exit this 'new' terminal using `exit` or control-D (depending on your shell). It is more fun, however, to detach the screen session, while running a command in the foreground. So simply execute `sleep` for half a minute:

```
> sleep 30
```

and detach screen, by typing `control-a control-d`. All screen commands start with a special control key, which by default is control-a. You should now have been returned to your previous terminal. Now list the available screen instances:

```
> screen -list
There is a screen on:
        8090.pts-64.lfe001        (Detached)
1 Socket in /var/run/screen/S-rol.
```

This shows you which screen(s) are running. You can have multiple screens, although that may be confusing. Now re-attach to this screen:

```
screen -r
```

You will get back to the screen terminal, which may still be running `sleep` (or it might just have ended).

Now create a new 'tab' inside screen. Use `control-a control-c` (c for 'create'). You end up in the new tab. If you want to switch to the previous tab, use `control-a 0`; the numbers 0 to 9 specify a tab. You can also use `control-a ¨` to get a list of tabs, and use the arrows keys & enter to select the tab (in case you have more than ten tabs). Lastly, to toggle back and forth between two tabs, you can simply type `control-a control-a`.

In case you accidentially loose your internet connection, your session to your the LOFAR cluster will quit. 'Screen', however, will still be running. If you login again to the machine where you started screen, you can see it:

```
> screen -list
There is a screen on:
        8090.pts-64.lfe001        (Attached)
1 Socket in /var/run/screen/S-rol.
```

Note that it says Attached; you can't simply attached to it using `screen -r`, you first have the detach the (now defunct) old screen session, and then reattach to it:

```
screen -d -r
```

Inside screen, there are a lot more commands that you can use. Type `control-a ?` to get a help view (space or enter to exit). The above sample session should keep you going for most tasks anyway. Note that you can easily ssh to machines from within screen (just like normal), although forwarding

X ('-Y' option) won't work. So starting screen from the frontend node and then logging in to one or more compute nodes to run NDPPP, BBS and/or mwimager can work nicely.

For people who don't like the the control-a shortcut, you can redefine this key in a file called `.screenrc` (in your home directory. Enter the following line:

```
escape ^Jj
```

If you have problems with the delete key, try this in `.screenrc`:

```
bindkey -k kD
bindkey -d -k kD
```

There are many other options; see for example `http://www.emacswiki.org/emacs/GnuScreen`.