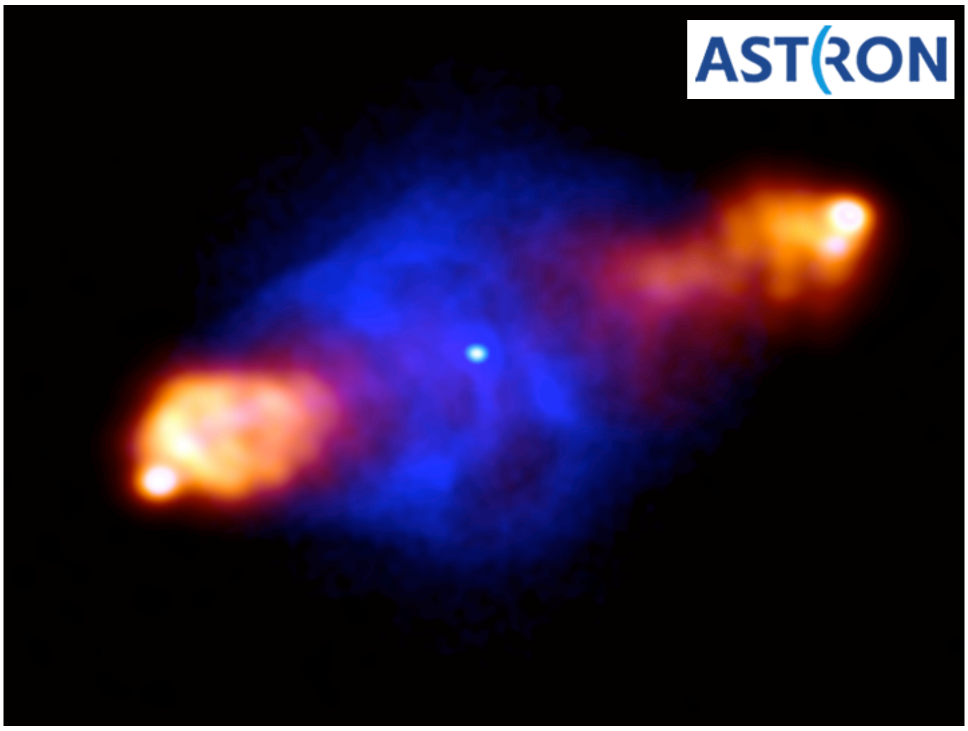
THE LOFAR IMAGING COOKBOOK: Manual data reduction with the imaging pipeline

Version 18.0

November 11, 2015



Edited by Aleksandar Shulevski

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 The LOFAR wiki[1] contains much more information on each stage of the data reduction, but might be out of date in many places. 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 and talk to the software developers in case you experience problems or have any questions.

For any suggestions or comments regarding this manual, you are kindly requested to post an issue in the LOFAR issue tracker[2]. As a reference person, please select A. Shulevski, who will either solve the issue or report it to the maintainer of a specific chapter of the manual. Please read the instructions regarding the issue tracker on the LOFAR wiki[3] beforehand.

The Lofar Imaging Cookbook was edited by Timothy Garn before he passed away.

COVER ILLUSTRATION:

LOFAR is being used to study supermassive black holes and the affect they have on their local environment. A classic example of an active galaxy is Cygnus A, which lies in a nearby cluster of galaxies at a distance of about 700 million lightyears. At the center of this galaxy is a powerful active nucleus that emits jets of plasma at relativistic speeds. An early LOFAR image at 240 MHz shown here that these jets extend far beyond the stellar part of the galaxy, up to 200 thousand light years from the center, before abruptly interacting with the intra-cluster medium at impact points called hotspots. (Image credits: J. McKean and M. Wise, ASTRON).

Editor in chief: Aleksandar Shulevski[4]

Authors: Roberto F. Pizzo, Ger van Diepen, Tammo Jan Dijkema, G. Heald, Francesco de Gasperin, M. Iacobelli, John McKean, Maaijke Mevius, André Offringa, Emanuela Orrú, David Rafferty, Cyril Tasse, Bas van der Tol, Valentina Vacca, Nicolas Vilchez, Reinout van Weeren, Wendy Williams and Sarod Yatawatta, on behalf of the Lofar commissioning team.

[1]`http://www.lofar.org/operations/doku.php?id=software:standard_imaging_pipeline`
[2]`https://support.astron.nl/lofar_issuetracker/`
[3]`http://www.lofar.org/operations/doku.php?id=maintenance:lofar_issue_tracker`
[4]`shulevski[at]astron[dot]nl`

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 on 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 the changes

Most recent changes

2015-11-11
- updated the Getting Started chapter (Chapter 1);
- updated the LoSoTo chapter (Chapter 8);
- added new DPPP calibration chapter (Chapter 6);
- added new RM Synthesis chapter (Chapter 13);
- changed the references for the LOFAR analysis scripts to the GitHub repository instead of CEP3.

2015-06-19
- update of the Getting Started chapter (Chapter 1).
- updated the DPPP chapter, improved the baseline selection documentation, added documentation for the up-arrow character and added a few examples (Chapter 5);
- update of the LoSoTo chapter (Chapter 8);
- update of the Sagecal chapter (Chapter 9);
- update of the AW Imager chapter (Chapter 10);
- update of the Source Detection chapter (Chapter 11);
- update of the Sky Model Construction chapter (Chapter 14);
- update of the Useful resources chapter (Chapter 16)

2015-02-24
- Renaming on NDPPP with DPPP through the Cookbook;
- added to the DPPP chapter a description on how to use steps coded in Python (Sect. 5.4.1);
- update of the Selfcal chapter (Chapter 12);
- update of the LoSoTo chapter (Chapter 8);
- update of the BBS chapter (Chapter 7), including a description on how to use `parmexportcal`.

2014-11-04
- general update of the entire manual to make it compatible with the architecture of the CEP3 commissioning cluster.

2014-04-22
- corrections to the BBS chapter (Chapter 7);
- corrections to the Selfcal chapter (Chapter 12).

Contents

1 Getting Started[5]

1.1 The LOFAR cluster layout

The correlated data coming from the Cobalt[6] correlator are stored on a cluster of machines called CEP2 cluster (see Sect. 1.1.1 for an extensive overview). CEP2 has been added to the LOFAR offline system at the beginning of 2011. Till March 2011, another cluster of computing machines was used to store and process the data (CEP1). Nowadays, CEP2 is normally used by the Radio Observatory to process the data through the initial stages of the data reduction (flagging and averaging of the visibilities), while another CEP facility is currently used by both the commissioners and LOFAR users to manually play with the data and understand which strategy to use for the calibration and the imaging: the new commissioning cluster CEP3 (see Sect. 1.1.2). CEP2 will be soon replaced by another CEP facility, providing technologies that are not available on CEP2, especially with respect to resource management. The CEP4 cluster is currently being installed with the aim to have it ready for production before the end of 2015. In the following sections we will focus on discussing the architecture of CEP facilities as well as their usage policies.

1.1.1 CEP2

The LOFAR CEP2 cluster is composed of 100 compute nodes (`locus001-100`). A detailed description of all the packages available on the new cluster and on its network interface can be found on line at http://www.lofar.org/wiki/doku.php?id=operations:phase2cluster. In the following, a few more details on this cluster components are given.

- **Frontend:** it has 2 lhn head nodes (called lhnxxx; lhn001 is the main head node for users). Each head node (lhn001 & lhn002) consists of: 64 GB memory 16 CPU's @ 2.3 GHz, 512 GB Solid State Disc, 6 TB RAID5 disks.
- **Processing units:** the 100 locus compute nodes with 21TB each (called locusxxx). Each locusxxx compute nodes consists of: 64 GB memory, 24 CPU's (6 quad cores @ 2.1 GHz), 21 TB disk space, XFS filesystem, 40 Gbps Infiniband network interface.
- **4 router nodes:** Each of the 4 router nodes connect a quarter of the compute nodes to the CEP network switches. They are also used to interface the Ethernet Fiber connections to the Infiniband interconnect of the cluster. They can be seen as network equipment by the user.
- **Infiniband interconnect using Ethernet protocol:**

USAGE POLICY: *CEP2 is meant for storing the observations and process them through the Lofar pipelines but not for commissioning work. For that, commissioners can use CEP3 with prior coordination with the Radio Observatory (sciencesupport[at]astron[dot]nl). Since the CEP2 cluster is meant to be a machine dedicated to run the pipeline, the Observatory will give access to it just to a few selected people (exclusively to perform MSSS- related work). Due to the limited disk space, CEP2 is not intended for long term storage of raw data and/or intermediate products.*

[5]This chapter is maintained by M. Iacobelli, `iacobelli[at]astron[dot]nl`.
[6]It is located in Groningen, The Netherlands.

1.1.2 CEP3

The CEP3 cluster is available since November 2014 and allows running science processing close to the CEP2 facility as well as for commissioning work. An overview of the hardware and software as well as of major policies and procedures is available at http://www.lofar.org/operations/doku.php?id=cep3:start. The cluster consists of 24 equivalent servers with the following specifications:

- DELL PowerEdge R720 Rack server
- Dual Intel Xeon e5 2660 v2 processors (10 cores each)
- 128 GB memory
- 4x 8 TB internal disk configured in RAID 6 (24 TB net capacity)
- 10 GE data interconnect
- 1 GE management network

In the future the servers can be fitted with up to two GPU cards (e.g. NVIDIA K20X).

Two head nodes (lhd001 & lhd002) are available for logging in and (limited) interactive development and processing purposes. Access to the CEP3 system can only be done through the lhd002 head node, Access to the twenty worker nodes is managed through a job management system. Users are required to submit requests for processing jobs/sessions on the worker nodes. In general, data will be distributed across the local disks on the worker nodes and processing jobs are distributed accordingly.

USAGE POLICY: *Observing, CEP2 processing time and the use of CEP3 are allocated by the LOFAR Programme Committee and the ILT director during the regular proposal evaluation stages, or under Director's Discretionary Time.*
Access and use of CEP3 is under the sole control of the Radio Observatory's Science Support Group (SSG). Access for Users will be granted only at the discretion of the Science Support Group. Users should conform to the access, resource allocation and data deletion policies issued by the SSG at all times.
Users awarded with access to CEP3 will be able to access the cluster for a limited period of time (4 weeks by default). The awarded period starts from the moment the user's data is copied from CEP2 (after Radio Observatory pipeline processing) to CEP3. At the beginning of a Cycle, users can derive this timeline by checking the observing schedule, which is available at http://www.astron.nl/radio-observatory/cycles/cycles. Access timelines related to observing programs involving observations spread in time will be discussed between the PI and Science Support. After the granted period on CEP3 has expired, all user's data products generated on the cluster will be automatically and promptly removed, to enable new users to have enough disk space to perform their data reduction.
Extensions to the default 4-week period will be granted only in exceptional circumstances and only if properly justified through a formal request to be sent to sciencesupport@astron.nl no later than 1 week before the expiration of your access privileges. Monitoring of node usage during allocated time will be performed and the evaluation of extension requests will be based on such statistics.

1.2 Logging on to CEP3

As mentioned above, normal users have access only to CEP3 head node(s), while the access to processing nodes is controlled using the Slurm cluster management software. In the head node users can experience the quality of the data and understand the best approach to use in the lof node(s) for the calibration and imaging of the visibilities. After Science Support has set up a reservation on a particular processing node(s), you should have a reservationID needed for setting up access to the working node(s)

- To access CEP3, begin by logging on to `portal.lofar.eu`[7]:

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

  where `<user name>` is likely to be the user's surname. Type in your default password[8]:

  ```
  > "password"
  ```

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

- You can now log in to the front end (you are requested to use `lhd002`)[9]:

  ```
  > ssh -Y lhd002
  ```

  If this is the first time you will be logging onto the cluster, you are advised to change your password by typing:

  ```
  > yppasswd
  ```

  in the usual fashion (old default password, new password, confirm new password).

- In order to log on to (for example) `lof019`[10], start a Slurm job from the head node `lhd002`. Any job will do, but we advice starting an interactive bash shell:

  ```
  > use Slurm
  > srun --reservation=<reservationID> -N<nr of nodes> bash -i
  ```

  This should give you a prompt (or more than one if you have more nodes on your job). Once you have the Slurm job running and log using ssh-keys enabled (see Sect. 1.3.2 for instructions), you are allowed standard SSH access with X-forwarding to the reserved nodes from the head node starting from a new terminal screen on the head node `lhd002`:

  ```
  > ssh -XY <user name>@lof019
  ```

  Once on the compute node, you will be located in your home directory (`/home/<user name>`), which is visible from any node.

For more information on the cluster architecture/properties, you can visit:

http://www.lofar.org/operations/doku.php?id=public:lofar_cluster
http://www.lofar.org/operations/doku.php?id=cep3:start

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

[8]Your default password will be communicated to you at the moment of the creation of your Lofar account by Teun Grit (`grit[at]astron[dot]nl`).

[9]You may have to log out of and log in again to the portal first

[10]The Radio Observatory will assign you with a suitable `lof` node to work on

1.3 Setting up your working environment

1.3.1 Login scripts

After an account is created, you will have a separate CEP3 `$HOME` directory. At the first login, it will be empty and needs to be setup properly to be able to use the provided tools and programs on CEP3. Log in to the front end cluster and from your `$HOME` directory follow the approaches reported below.

Create a link to the systems login scripts:

- Make sure to delete any potential `.profile` or `.cshrc` file on your home directory (check with `ls -a`)

-
  ```
   > ln -s /opt/cep/login/bashrc $HOME/.profile
  ```
  or
  ```
   > ln -s /opt/cep/login/cshrc $HOME/.cshrc
  ```
  depending on whether you use a BASH login shell or a (T)CSH login shell.
- Now log in 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.

While the python packages are initialised by default, to use a certain package you can make use of the "`use`" alias. To initialise the LOFAR imaging pipeline software together with and linked against casacore, casarest, and pyrap type

```
 > use Lofar
```

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

```
 > use Casa
```

On CEP3 AIPS version 31DEC14 is available. To be able to use you must contact Marco Iacobelli, Teun Grit or Arno Schoenmakers to add you to the user group "`aipsgrp`". Then you can execute

```
 > use AIPS
```

You can then start AIPS with the normal startup command, e.g.

```
 > aips tpok tv=local
```

SageCal and auxiliary scripts have been installed, using OpenBlas. To activate them type

```
 > use Lofar

 > use Sagecal
```

The self-calibration tool of the Standard Imaging Pipeline (see Sect. **??**) is installed and can be activated by typing:

```
> use Pyselfcal
```

The WSClean wide-field imager has been installed. To activate it type:

```
> use Wsclean
```

The COmplex Half-Jacobian Optimization for N-directional EStimation (CohJones) peeling tool and auxiliary scripts have been installed. To activate it type:

```
> use CohJones
```

To activate Duchamp, type

```
> use Duchamp
```

To activate DS9, just type

```
> use DS9
```

Similarly, the Karma software (useful during imaging), can also be initialised:

```
> use Karma
```

To run a few scripts needed during the data reduction (see Sect. 16.2) without having to copy them into your working directory, just type:

```
> use Lofar ;  use Cookbook
```

To activate a collection of python scripts from the Imaging commissioning, type

```
> use Scripts
```

To extract TEC, vTEC, Earth magnetic field and Rotation Measures from GPS and WMM data for radio interferometry observations, the RMextract tool is installed. To activate it type:

```
> use Lofar
```

```
> use RMextract
```

Finally, you can initialise scripts to perform RM-Synthesis with

```
use RMSynthesis
```

and activate Pyrmsynth 1.2.1 by executing rm_synthesis.py (Option –help gives options). Also available is library rm_tools (use `import rm_tools` within a python script).

You can also create a file in your home directory (`.mypackages`) that contains a list of all packages to initialize at login time. For example, if this file contains the line

```
Casa Lofar
```

the login scripts will initialize the CASA and the LOFAR imaging pipeline software for you at login time.

On the LOFAR wiki more information can be found about the LOFAR login environment. Also, an updated list of the software packages installed on CEP3 is found here.

Processing can now take place. Once you have logged onto this compute node, you should create your own working directory using

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

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

You can copy in here the data provided by the Radio Observatory by e.g. typing:

```
> scp -r <user name>@lhd002:/data/<user>/<LOFAR dataset> .
```

where `<LOFAR dataset>` has the syntax `LXXXXX`[11].

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 (Sect. 7) 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 `lhd002` set up passwordless access to the `lof` nodes (if you have a job running to provide you access) via SSH: Go to, or create a directory

  ```
  $HOME/.ssh
  ```

  ```
  > ssh-keygen -t dsa
  ```

  Press enter and again when prompted for a password. Now you have a file id_rsa.pub in this directory. Copy it to authorized_keys.

  ```
  > 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.

- For Windows and additional information, refer to the LOFAR wiki [12].

[11] "L" stays for LOFAR, `XXXXX` is the ID number of the observation, which is assigned to it at the moment of the scheduling

[12] `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 lof019
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
```

1.3.4 Data copy from and to CEP3 cluster

Data transfers from CEP2 to CEP3 should always be coordinated with the Radio Observatory (sciencesupport[at]astron[dot]nl). Data retrieval from LTA locations as well as from/to other computing facilities is also possible as detailed at http://www.lofar.org/wiki/doku.php?id=cep3:userdata. Although access to the CEP3 systems can only be done through the lhd002 head node, data transfers to the outside world can be done directly. Because data will be transferred via the LOFAR portal, care should be taken not to flood the available network bandwidth with the public internet. Thus we recommend to limit the bandwidth to not disturb the portal access of other users. Note that the portal capacity is 120MB/s.

2 Data Inspection[13]

Data inspection is essential for a proper data reduction and can be carried out using either scripts (by means of a python interface to the Measurement Set) or CASA[14]. In this Chapter, we summarize the various tools that can be used to inspect the LOFAR visibilities at the beginning and during the processing of your data.

2.1 Viewing Measurement Set details

The program `msoverview` provides you with details on the contents of a Measurement Set, no matter if it is raw or it has been already processed. You are advised to use this script when you start working on your data. You can run it by typing:

```
> msoverview in=some.MS verbose=T
```

where the `verbose` parameter allows you to have more detailed information about the observation, as the used antennas and their positions.

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

```
msoverview: Version 20110407GvD
================================================================================
MeasurementSet Name:  /data1/L2011_24909/L24909_SB000_uv.MS      MS Version 2
================================================================================
           This is a raw LOFAR MS (stored with LofarStMan)

Observer: unknown     Project: LOFAROPS
Observation: LOFAR
Antenna-set: HBA_ZERO

Data records: 47250       Total integration time = 125.174 seconds
Observed from   31-Mar-2011/14:04:59.0   to   31-Mar-2011/14:07:04.2 (UTC)

Fields: 1
  ID   Code Name          RA             Decl            Epoch
  0         BEAM_0        04:37:04.0000 +29.40.14.0000 J2000
   (nVis = Total number of time/baseline visibilities per field)

Spectral Windows:  (1 unique spectral windows and 1 unique polarization
                                                                 setups)
  SpwID  #Chans Frame Ch1(MHz)    ChanWid(kHz)TotBW(kHz)  Ref(MHz)    Corrs
  0          64 TOPO  114.942932  3.05175781  195.3125    115.039062  XX  XY
                                                                      YX  YY

Antennas: 27:
  ID   Name      Station   Diam.    Long.         Lat.
  0    CS001HBA0 LOFAR     0.0  m   +006.52.07.1  +52.43.34.7
```

[13]This chapter is maintained by R. F. Pizzo, `pizzo[at]astron[dot]nl`

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

```
  1    CS002HBA0 LOFAR    0.0  m   +006.52.07.6  +52.43.46.8
  2    CS003HBA0 LOFAR    0.0  m   +006.52.10.9  +52.43.51.8
  3    CS004HBA0 LOFAR    0.0  m   +006.52.03.0  +52.43.47.8
  4    CS005HBA0 LOFAR    0.0  m   +006.52.08.8  +52.43.42.5
  5    CS006HBA0 LOFAR    0.0  m   +006.52.17.0  +52.43.43.7
  6    CS007HBA0 LOFAR    0.0  m   +006.52.15.3  +52.43.51.1
  7    CS017HBA0 LOFAR    0.0  m   +006.52.38.3  +52.43.52.0
  8    CS021HBA0 LOFAR    0.0  m   +006.51.46.1  +52.43.54.4
  9    CS024HBA0 LOFAR    0.0  m   +006.52.27.5  +52.43.19.7
  10   CS026HBA0 LOFAR    0.0  m   +006.52.54.0  +52.43.50.0
  11   CS030HBA0 LOFAR    0.0  m   +006.51.37.7  +52.44.12.4
  12   CS032HBA0 LOFAR    0.0  m   +006.51.39.2  +52.43.38.6
  13   CS101HBA0 LOFAR    0.0  m   +006.52.50.8  +52.44.11.3
  14   CS103HBA0 LOFAR    0.0  m   +006.53.45.2  +52.43.48.8
  15   CS201HBA0 LOFAR    0.0  m   +006.52.55.0  +52.43.39.2
  16   CS301HBA0 LOFAR    0.0  m   +006.52.07.1  +52.43.11.7
  17   CS302HBA0 LOFAR    0.0  m   +006.50.53.7  +52.42.57.6
  18   CS401HBA0 LOFAR    0.0  m   +006.51.24.1  +52.43.42.8
  19   CS501HBA0 LOFAR    0.0  m   +006.51.57.9  +52.44.29.9
  20   RS106HBA  LOFAR    0.0  m   +006.59.05.6  +52.41.21.6
  21   RS205HBA  LOFAR    0.0  m   +006.53.50.8  +52.40.17.6
  22   RS208HBA  LOFAR    0.0  m   +006.55.10.4  +52.29.03.1
  23   RS306HBA  LOFAR    0.0  m   +006.44.32.3  +52.42.18.5
  24   RS307HBA  LOFAR    0.0  m   +006.40.54.8  +52.37.02.5
  25   RS406HBA  LOFAR    0.0  m   +006.45.04.2  +52.49.59.6
  26   RS503HBA  LOFAR    0.0  m   +006.51.04.8  +52.45.33.2

The MS is fully regular, thus suitable for BBS
   nrows=47250  ntimes=125  nbaselines=378  nband=1
```

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

2.2 Pyrap / PyDAL scripts

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. 1 and 2). These allow you to customize what is plotted, and can be significantly faster than CASA for plotting large datasets.

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.
An extensive description of the Pyrap utilities is available at the web address

http://www.astron.nl/casacore/trunk/pyrap/docs/

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

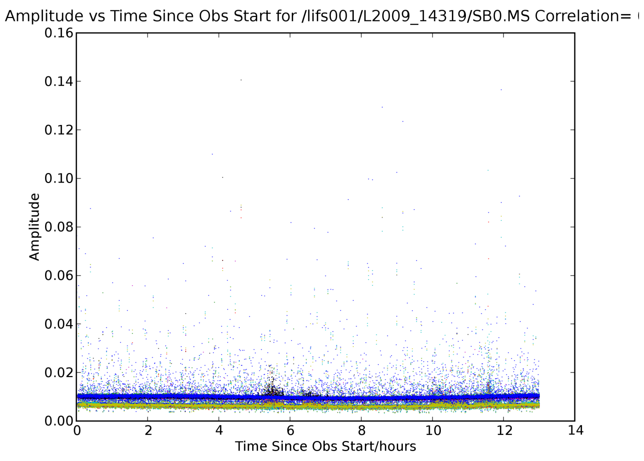
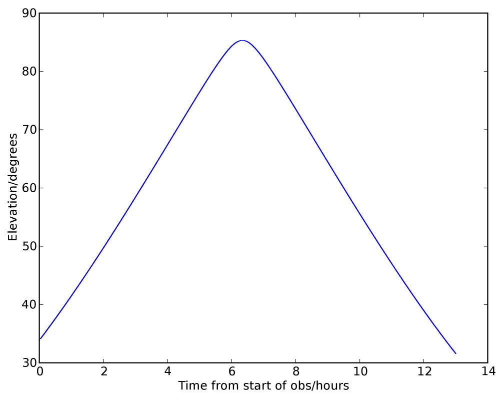
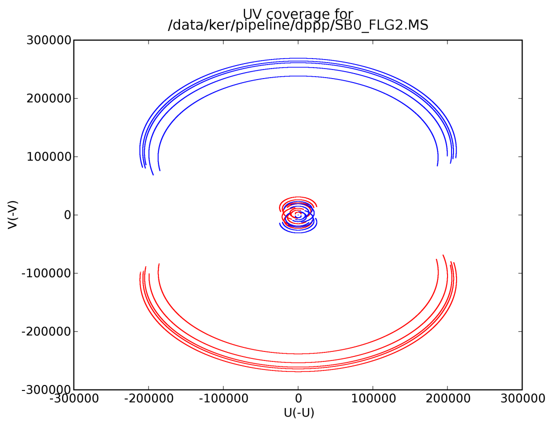


Figure 1: Plotted here using a pyrap script is the amplitude vs time for the SB0.MS 3C196 observation. The high level of RFI is instantly apparent.



(a) Elevation vs. time



(b) *uv* coverage

Figure 2: Two examples of useful data plotted with pyrap scripts, for the SB0.MS 3C196 observation.

from all baselines in a Measurement Set[15]. It plots either amplitude or phase against time, frequency, or channel number.

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

```
> use Cookbook
> use Lofar
```

The script itself is `uvplot.py`[16], and it will list its (many) options if you use the `-h` flag.

```
> uvplot.py -h
uvplot.py v1.6, 14 April 2010
Usage:  uvplot.py [options]
Options:
-h, --help show this help message and exit
```

[15]`uvplot.py` can also be used to plot the raw visibilities and may be significantly faster than the PLOTMS task in CASA.

[16]The script has been created by G. Heald.

```
-i INMS, --inms=INMS Input MS to plot [no default]
...
```

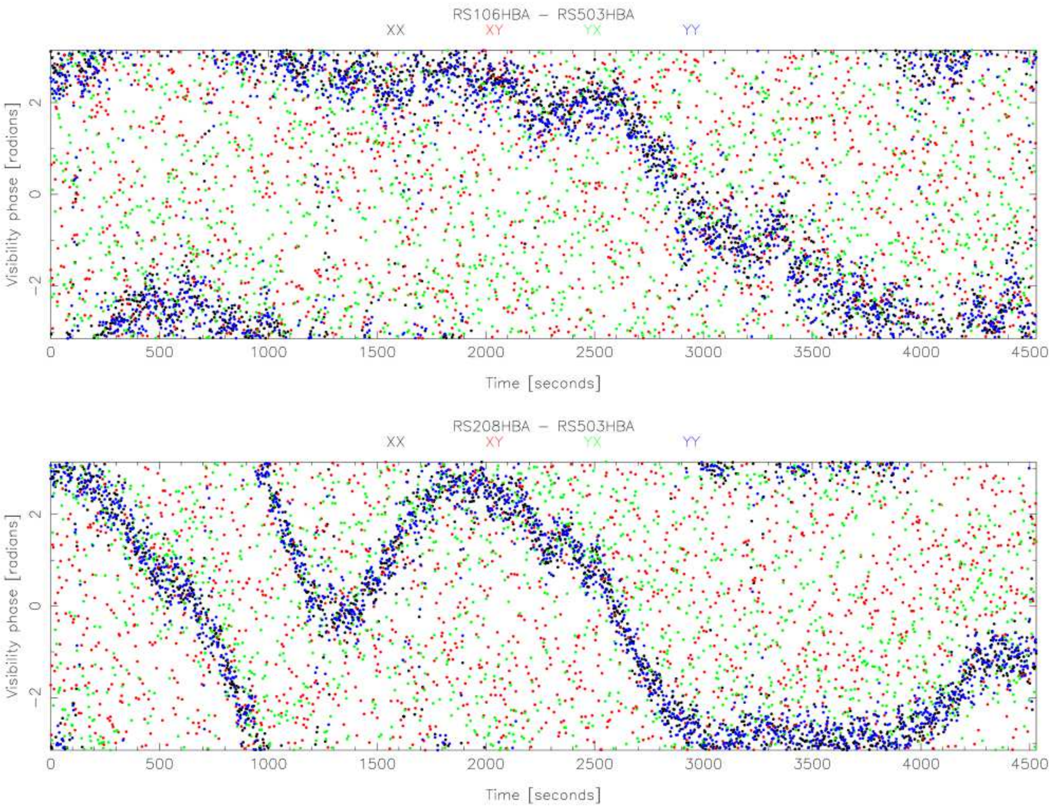


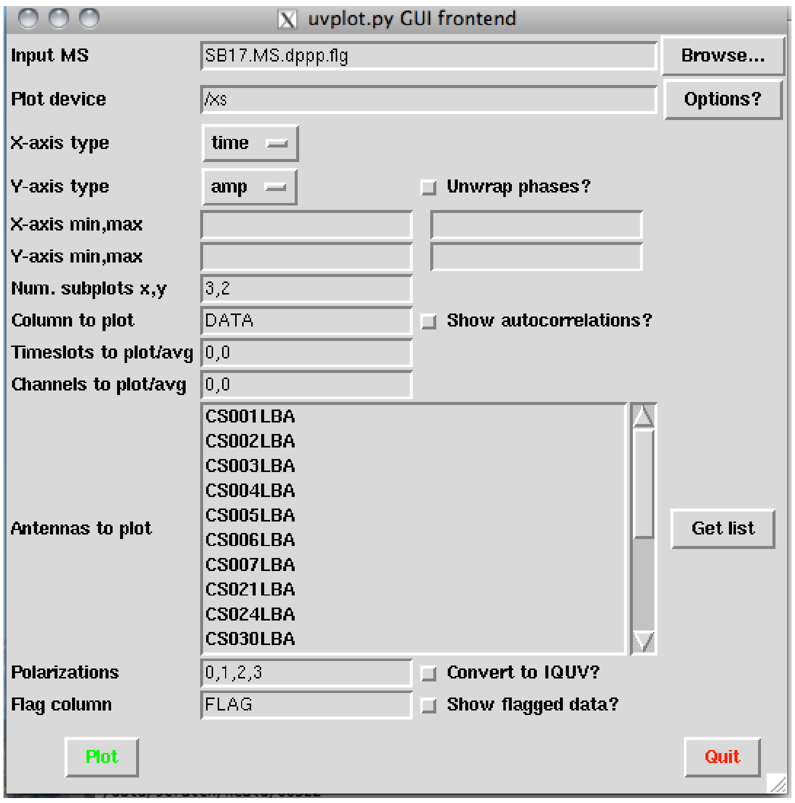
Figure 3: An example of using `uvplot.py`. The command used to generate this plot was `uvplot.py -i /data/scratch/pipeline/L2009_15697/SB64.MS -t 0,1500 -y phase -n 1,2 -x time -d output.ps/cps` .

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

```
uvplot.py -i /data/scratch/pipeline/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. 3

To simplify the process of specifying plot options, you can optionally use the GUI interface to the plotting program (see Fig. 4). 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.



Input MS	SB17.MS.dppp.flg	Browse...
Plot device	/xs	Options?
X-axis type	time	
Y-axis type	amp	Unwrap phases?
X-axis min,max		
Y-axis min,max		
Num. subplots x,y	3,2	
Column to plot	DATA	Show autocorrelations?
Timeslots to plot/avg	0,0	
Channels to plot/avg	0,0	
Antennas to plot	CS001LBA CS002LBA CS003LBA CS004LBA CS005LBA CS006LBA CS007LBA CS021LBA CS024LBA CS030LBA	Get list
Polarizations	0,1,2,3	Convert to IQUV?
Flag column	FLAG	Show flagged data?

Figure 4: The GUI interface to the plotting program `uvplot.py`.

2.4 CASA

CASA[17] is the python-based next generation replacement for AIPS/AIPS++ and can be used to display the data[18]. 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 DPPP (see the first example parset in Sect. 5.2). Moreover, when opening with CASA a MeasurementSet observed between May and October 2011, you will get the following error:

```
Unrecognized mount type
```

This is due to the fact that the MS writer version used during those months was specifying the antenna mount as FIXED, and not as ALT-AZ, which is CASA friendly. To solve this problem, you can run the following `taql` command on your MS (`your.MS`):

```
> taql 'update your.MS/ANTENNA set MOUNT="X-Y" '
```

2.4.1 Casaviewer

To initialize the CASA software, type

```
> use Casa
```

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

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

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

```
> casaviewer
```

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[19] 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>[21].

[19]We caution users that the task plotxy is no longer being developed and will be deprecated in favor of plotms in future releases of CASA.

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

[21]For more detailed information and examples of how to use CASA, we direct users to the CASA Guides: http://casaguides.nrao.edu/index.php?title=Main_Page

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 DPPP (see Sect. 5), making a copy of it (see Sect. 5.2).
- 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.

We recommend that CASA users regularly check the list of known issues maintained by the NRAO for each officially released version. This information can be found by visiting the CASA releases page[22], selecting the relevant version of CASA, and then clicking on the Known Issues link.

2.5 The Drawer[23]

The Drawer is a useful algorithm that can be adopted to quickly inspect a MeasurementSet and investigate which sources are contributing to the visibilities. The software automatically converts the fringes seen in the visibilities to locations in the sky, having the advantage that (i) it works very well on the raw data and therefore it can be used before any calibration, (ii) it is very fast to recover spatial information on the half sphere centered on the phase center of the observation (one can generally generate an all sky plot in less than a few minutes). The concept behind the Drawer was already known and used in AIPS (task FRMAP).
As the fringes ”produced” by each individual baseline are rotating on the sky, each source modulates the visibility, depending on its distance from the phase center (far away sources give a higher fringe rate). The Drawer performs an FFT of the visibility of each given baseline in a particular timeslot, along the time axis, and finds the dominant frequency. From that value, and from the ”speed” of the given baseline in the uv plane, it solves a simple equation and derives a line on the sky. Per baseline, it reflects all the possible places where the source producing the given detected modulation could be. The lines of all the baselines/timeslots are then gridded onto an image. The pixel values do not reflect the flux of the sources, but the log of the occurrence of fringe finding. The current version of the software does not deal with data chunks yet, i.e. it first reads the whole MS and puts the visibilities into memory. Therefore it performs quicker on averaged datasets containing a few channels.

2.5.1 Examples

First, the LofIm and Pythonlibs environment need to be properly set:

```
> use Lofar
> use Cookbook
```

The help file of drawMS is self contained:

```
> drawMS -h

Options:
  --version             show program's version number and exit
```

[22]http://casa.nrao.edu/release.shtml
[23]This sub section was kindly provided by Cyril Tasse (cyril[dot]tasse[at]obspm[dot]fr)

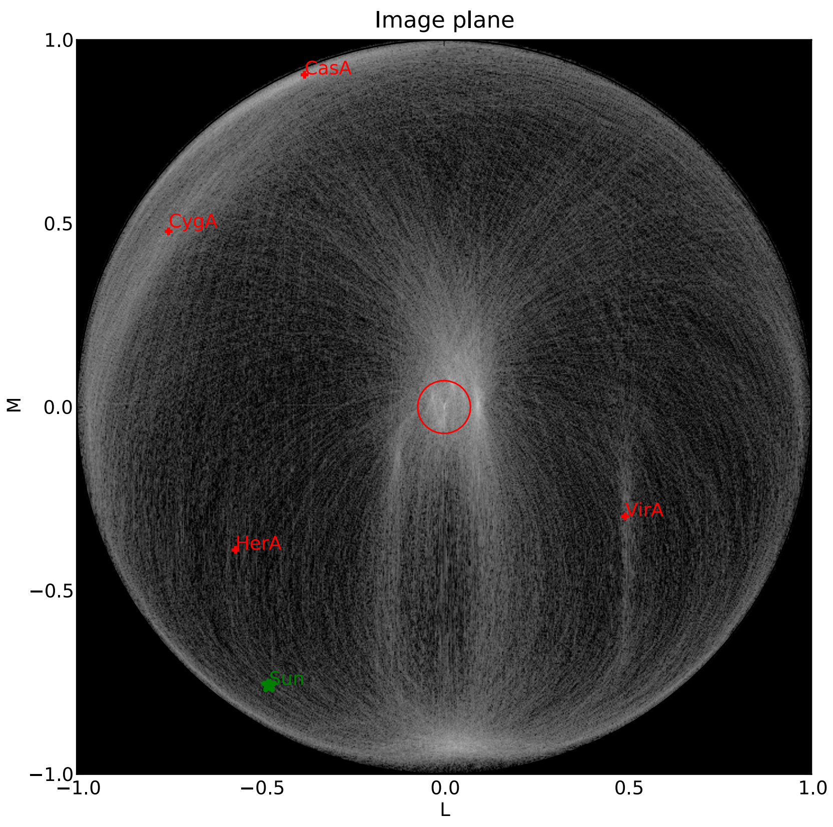


Figure 5: drawMS is a simple algorithm that allows to quickly recover spatial information on the sources that have the brightest apparent flux. In this example, drawMS is run on the raw data of an observation of the Bootes field. One can clearly see the contribution from CasA, CygA, and TauA, while there is no direct contribution from the Sun.

```
-h, --help            show this help message and exit

* Necessary options:
  Won't work if not specified.

  --ms=MS             Input MS to draw [no default]

* Data selection options:
  ColName is set to DATA column by default, and other parameters select
  all the data.

  --ColName=COLNAME   Name of the column to work on. Default is DATA. For
                      example: --ColName=CORRECTED_DATA
  --uvrange=UVRANGE   UV range (in meters, not in lambda!). Default is
                      0,10000000. For example: --uvrange=100,1000
  --wmax=WMAX         Maximum W distance. Default is 10000000.
  --timerange=TIMERANGE
                      Time selection range, in fraction of total observing
                      time. For example, --timerange=0.1,0.2 will select the
                      second 10% of the observation. Default is 0,1.
  --AntList=ANTLIST   List of antennas to compute the lines for. Default is
                      all. For example: --AntList=0,1,2 will plot 0-n, 1-n,
                      2-n
```

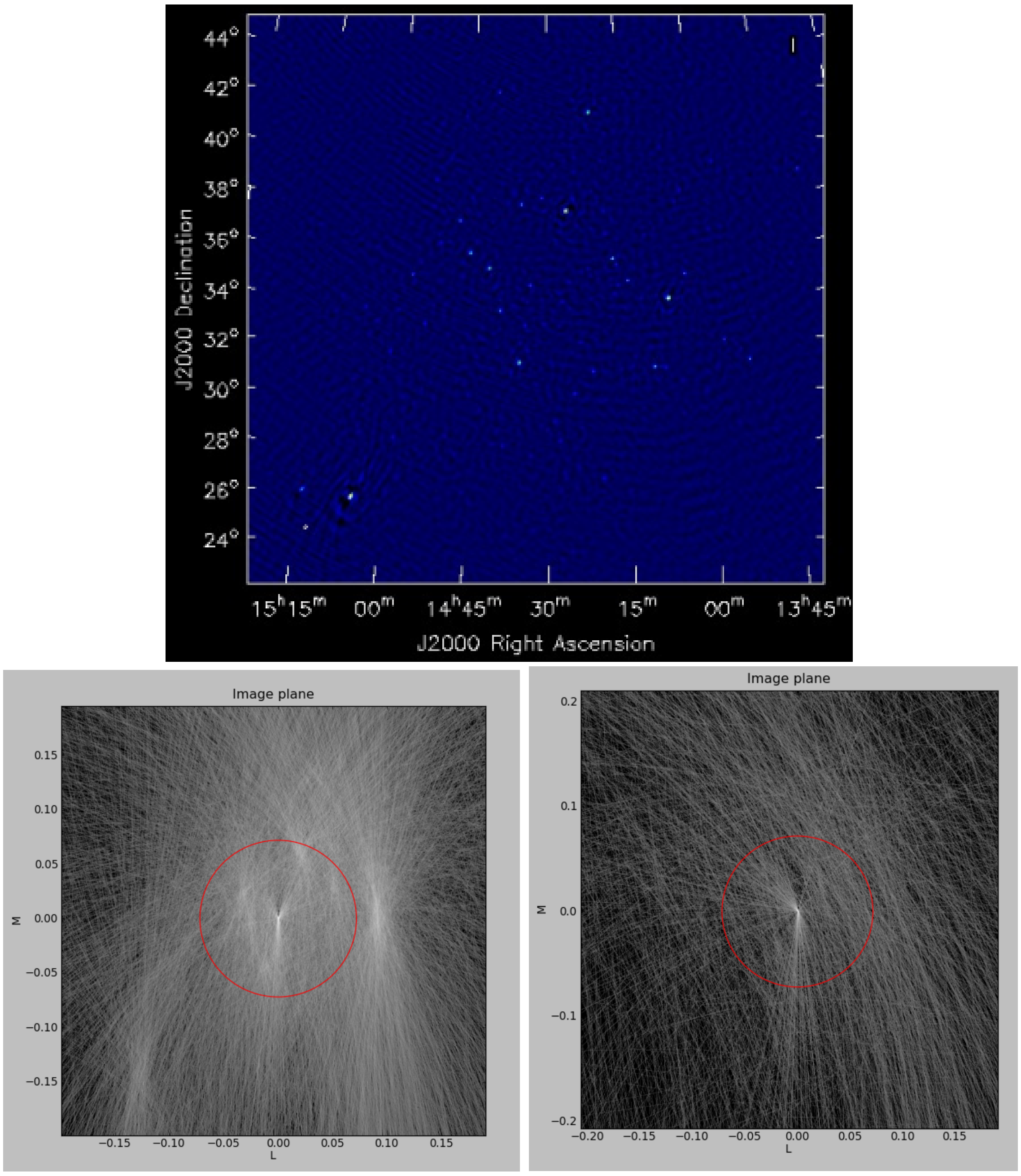


Figure 6: *Top panel*: a very wide field image of the Bootes field (a few SBs), which took a long time to be imaged because of the large field of view. *Bottom panel*: drawMS outputs were generated in less than 5 minutes. Line plots show the overdensities corresponding to real sources in the image. We can see that the coherency in the data gets partly lost before and after sunrise. This is probably due to turbulence in the ionosphere.

```
  --FillFactor=FILLFACTOR
                          The probability of a baseline/timeslot to be
                          processed. Default is 1.0. Useful when large dataset
                          are to be drawn. For example --FillFactor=0.1 will
                          result in a random selection of 10% of the data

* Algorithm options:
  Default values should give reasonable results, but all of them have
  noticeable influence on the results
```

```
    --timestep=TIMESTEP
                        Time step between the different time chunks of which
                        the drawer does the fft. Default is 500.
    --timewindow=TIMEWINDOW
                        Time interval width centered on the time bin
                        controlled by --timestep. If not defined then it is
                        set to --timestep.
    --snrcut=SNRCUT     Cut above which the fringe is drawn. Default is 5.0.
    --maskfreq=MASKFREQ
                        When a fringe is found, it will set the fft to zero in
                        that 1D pixel range. Default is 2.0.
    --MaxNPeaks=MAXNPEAKS
                        Maximum number of fringes it will find per baseline
                        and timeslot. Default is 7.
    --NTheta=NTHETA     Number of angles in the l-m plane the algorithm will
                        solve for. Default is 20.

  * Fancy options:
    Plot NVSS sources, or make a movies.

    --RadNVSS=RADNVSS   Over-plot NVSS sources within this radius. Default is
                        0 (in beam diameter).
    --SlimNVSS=SLIMNVSS
                        If --RadNVSS>0, plot the sources above this flux
                        density. Default is 0.5 Jy.
    --MovieName=MOVIENAME
                        Name of the directory that contains the movie (.mpg),
                        the individual timeslots (.png), and the stack
                        (.stack.png). Each page correspond to the data
                        selected by --timewindow, separated by --timestep. For
                        example --MovieName=test will create a directory
                        "dMSprods.test". Default is None.
```

As explained in the help file, default values should give reasonable results, but all of them have noticeable influence on the results. However, some handy parameters that are often used are the following: **ColName** (”DATA” by default, or ”CORRECTED_DATA”), **FillFactor** (less lines, but speedup the calculus), **RadNVSS** (to display the location of NVSS sources), **MovieName** (to generate a time-movie), and **timewindow/timestep** (see help file).

Here are a few examples of drawMS possible usage. For the plot of Fig. 5, on the raw data:

```
/home/tasse/drawMS/drawMS --ms=name.MS
```

For the plot of Fig. 6, for the first part of the run, on the raw data:

```
drawMS --ms=name.MS --timerange=0.0,0.5 --FillFactor=0.5
```

and for the second part of the run:

```
drawMS --ms=name.MS --timerange=0.5,1.0 --FillFactor=0.5
```

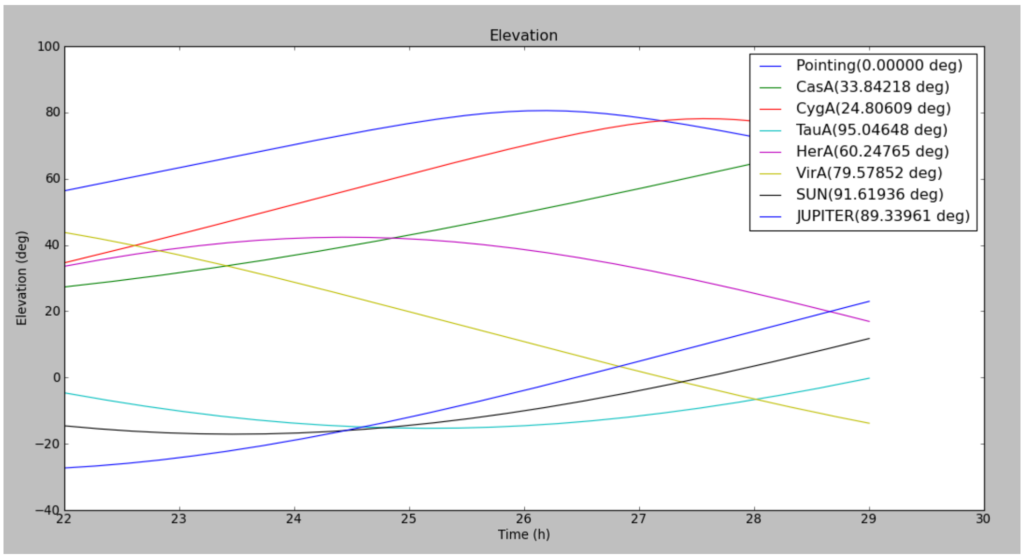


Figure 7: The elevation and angular distance of the A-team from the target field centered on B1835+62. The plot was obtained with `plot_Ateam_elevation.py`.

Time sliced animations (movies) can contain a lot of information, sometimes hard to interpret. The following command should produce something sensible:

```
drawMS --ms=name.MS --snrcut=3 --timestep=100
--timewindow=300 --uvrange=100,100000 --MovieName=test
```

2.6 Plot_Ateam_elevation.py

The low frequency radio sky is dominated by a few bright sources that form the so called A-team: CasA, CygA, VirA, TauA, HydA, HerA. The removal of these sources from the target visibilities is very important in order to achieve high dynamic range images. A useful script providing an overview of the elevation and distance of the A-team from a given target is `plot_Ateam_elevation.py`. On CEP3, you can initialize this script by typing:

```
use Cookbook
```

The output of the script (see e.g. Fig. 7) should be analyzed to understand which of the A-team sources should be subtracted from a given observation. The algorithm performing the subtraction is called demixing and is described in Sect. 5.1.10.

2.7 Useful tools to handle Measurement Sets

2.7.1 Concatenating subbands

Sometimes it is necessary to concatenate multiple subbands (e.g. to form a single image). This can be done in a very efficient way using the `msconcat` function in `pyrap.tables`.

```
> pt.msconcat (['ms1','ms2','ms3'], newms)
```

It concatenates the MSs given in the list and creates a new MS. The concatenation is done in a so-called virtual way, which makes it very fast. It also has the advantage that if data in the original MSs change (e.g., by a BBS run) the concatenated MS sees those changes as well, so there is no need to concatenate again. However, they must be re-concatenated if another column (e.g., CORRECTED_DATA) gets added to the orginal MSs.

Note that in general the resulting MS cannot be used in the CASA environment. However, it can be used in the casa table viewer.

2.7.2 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

# on the offline cluster if you are in c ot tcsh type:
# use Casa; use Pythonlibs; use Lofar; 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[at]astro.rug[dot]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
```

[24]it is initialized with `'use Cookbook'`

```
#     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`.

2.7.3 Converting MS times to a friendly format

It is possible to convert the times given in a Measurement Set from MJD (Modified Julian Date) to a more friendly format by using the TaQL command

```
dateList = pt.taql('calc ctod(mjdtodate([select TIME from ~/SB000.MS]))')
```

This will return a list with date/time strings for all cells in the TIME column. You can select only row 10 like e.g.

```
dateList = pt.taql('calc ctod(mjdtodate([select TIME from
                    ~/GER.MS limit 1 offset 10]))')
```

If you have a time in a python variable `timevar`, you can use something like

```
dateList = pt.taql('calc ctod(mjdtodate($timevar s))')
```

The `s` is needed to tell that the MJD is in seconds.

In the first examples TaQL sees in the table that the unit of TIME is seconds.

Of course, you can give any other selection in the select command.

See http://www.astron.nl/casacore/trunk/casacore/doc/notes/199.html for more info on TaQL.

3 Imaging pipeline[25]

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 Standard Imaging Pipeline. The Standard Imaging Pipeline is still under development, and its first version deployed in LOFAR Version 1.0 and 2.0 is graphically illustrated in Fig. 8.
The first standard data processing steps are encapsulated within a sub-pipeline called the Pre-processing Pipeline, which consists of two steps:

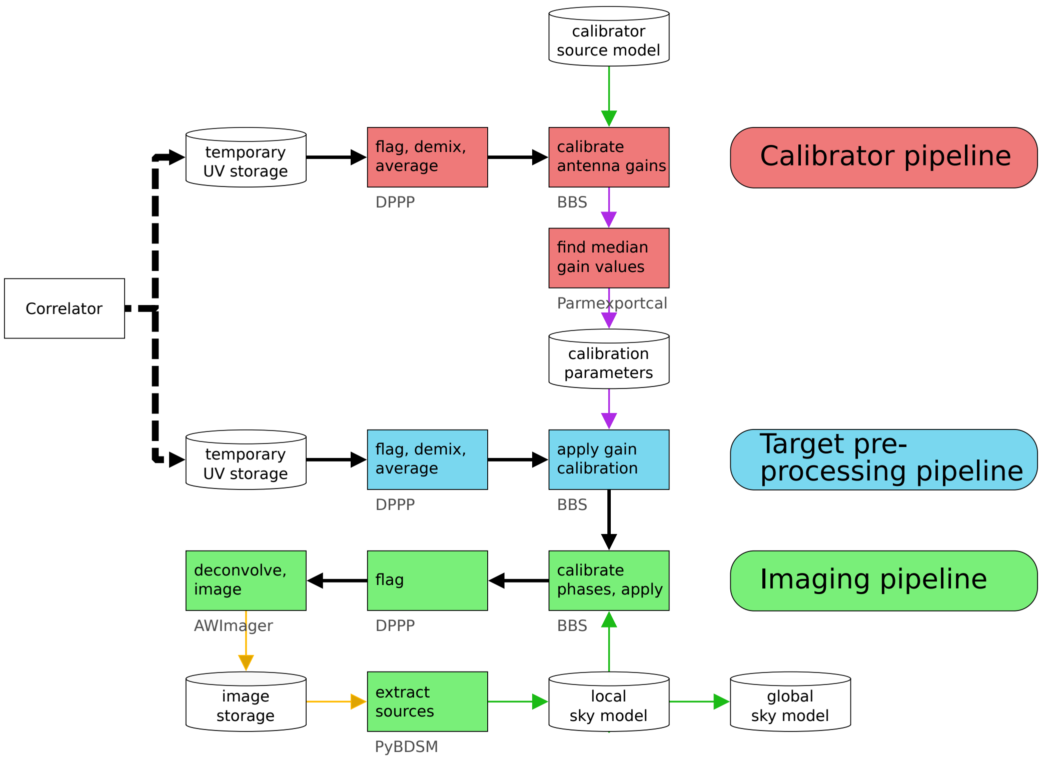


Figure 8: A diagram of the Standard Imaging Pipeline.

- The Calibrator Pre-Processing Pipeline, which flags the data in time and frequency, and optionally averages them in time, frequency, or both (the software that performs this step is labeled DPPP - New Default Pre-Processing Pipeline, see Sect. 5). This stage of the processing also includes a subtraction of the contributions of the brightest sources in the sky (the so called "A-team": Cygnus A, Cassiopeia A, Virgo A, etc...) from the visibilities through the 'demixing' algorithm (B. van der Tol, PhD thesis). Eventually, the solutions for the calibrator are computed, using the BlackBoard Selfcal (BBS) system (Sect. 7) - specifically developed for LOFAR.
- The Target Pre-processing pipeline performs RFI excision and removes the contributions of the A-team from the visibilities. This step is followed by the BBS calibration, which applies the externally generated solutions for the calibrator to the target field. In the future, the calibration of the complex station gains will be achieved using a local sky model (LSM). This will be generated from the Global Sky Model (GSM), which will soon be provided by the MSSS survey.

Following the pre-processing stage, the calibrated data are further processed in the Imaging Pipeline, which begins with an imaging step that uses the AWImager (Sect. 10). AWImager is able to perform

[25]This chapter is maintained by R. F. Pizzo, `pizzo[at]astron[dot]nl`

both W-projection and A-projection, a scheme that can potentially take into account all direction-dependent effects in the deconvolution step. Source finding software (Sect. 11) is used to identify the sources detected in the image, and generate an updated local sky model. In future versions of the pipeline one or more ‘major cycle’ loops of calibration (with BBS), flagging, imaging, and LSM updates will then be performed. At the end of the process, the final LSM will be used to update the GSM, and final image products will be produced.

3.1 Running long-duration 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.

4 The AOFlagger[26]

The frequencies covered by LOFAR are considerably affected by RFI, both in the low and the high band (see Fig. 9 and 10). An efficient cleaning of the data is essential to obtain high quality images.

The AOFlagger (the algorithm executed by "RFI Console") is an 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 DPPP (Chapter 5), 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.

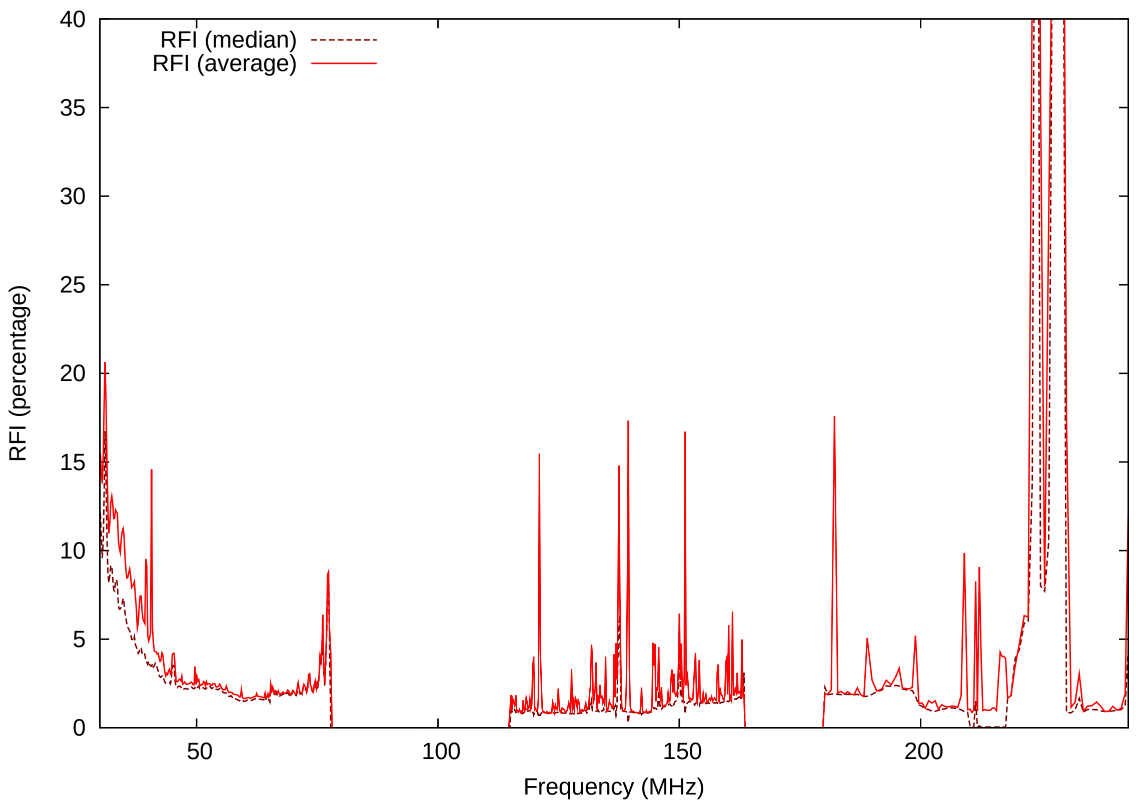
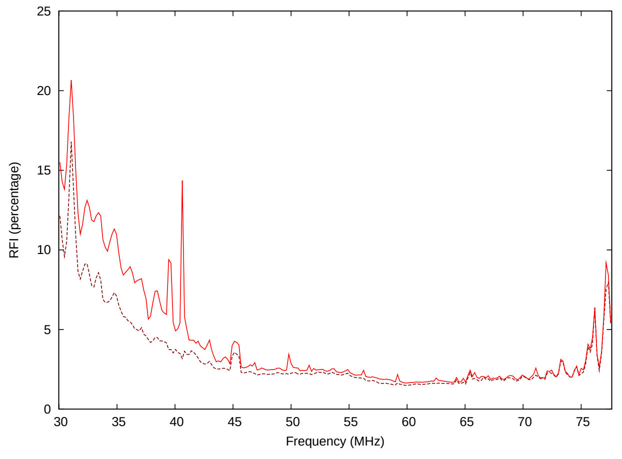


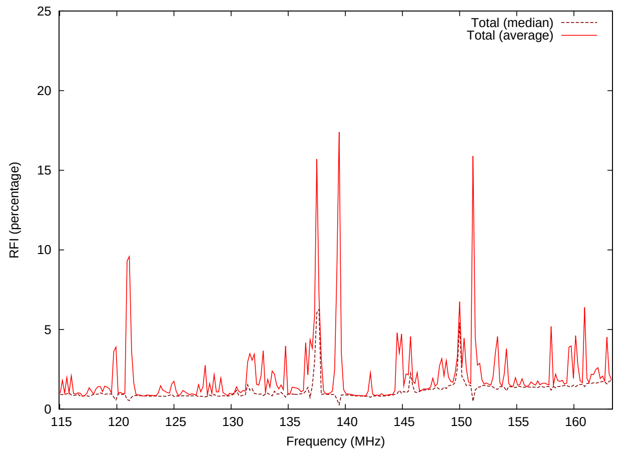
Figure 9: The RFI distribution in most of the observable frequencies of LOFAR. It combines data from four observations, whose RFI spectra have been given in more detail in figure 10. In most of the frequencies, the RFI situation is benign and cause only a few percent of data loss. Towards 30 MHz and lower, RFI become harder to deal with. The high end of the HBA contain a few strong and broadband digital broadcast transmitters.

Using the AOFlagger to flag the data and DPPP to average them afterwards seems a good strategy and it should provide data clean enough to perform the calibration and the imaging. Recently, the flagger has also been incorporated into DPPP (see Chapter 5). If one wants to use the default flagging settings, it is recommended that DPPP is used directly with the AOFlagger option, because it is faster and uses less resources compared to executing the steps separately. For details on how to run RFIconsole within DPPP, please see Chapter 5. In the following sections, the details regarding the AOFlagger as

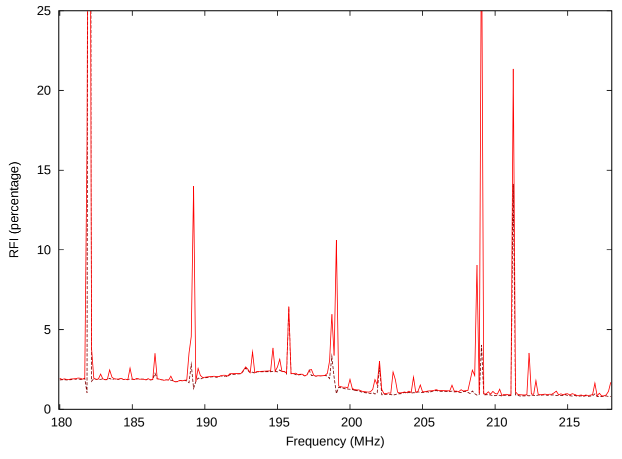
[26]The author of this Chapter is André Offringa (`andre[dot]offringa[at]anu[dot]edu[dot]au`).



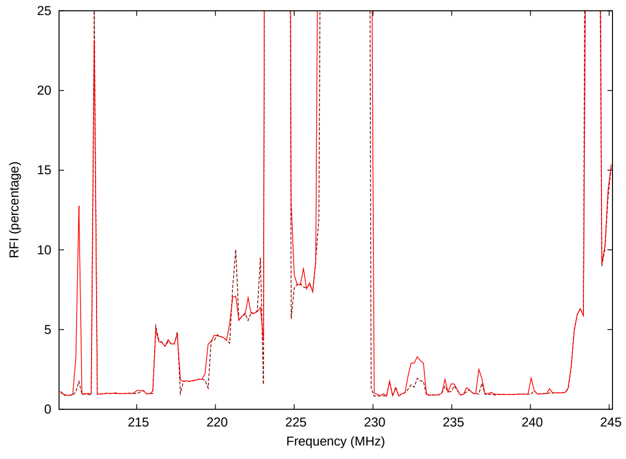
(a) LBA (6 hour observation)



(b) HBA low (24 hour observation)



(c) HBA mid (1 hour observation)



(d) HBA high (6 hour observation)

Figure 10: The RFI in the LBA and HBA bands of LOFAR. The HBA has been split into three parts (HBA low, mid and high), which were observed independently. Towards lower frequencies (below 30 MHz), the data are more affected by interferences. Parts of the HBA high range (around 225 MHz) are contaminated by several broadband digital broadcast transmitters. Please note that it is unknown whether these observations are representative of the RFI distribution in LOFAR bands. Also note that they were partly performed during daytime.

independent routine are summarized.

4.1 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 the flagger. 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. The fastest mode of RFI Console, called the indirect read mode, will rewrite the data set to a temporary location. This location will be the path from where you start RFI Console. Therefore, you have to make sure you start RFI Console from a scratch directory, for example:

```
 > mkdir /data/scratch/offringa/temp
 > cd /data/scratch/offringa/temp
```

Now, you are ready to flag the set:

```
offringa@lce032:/data/scratch/offringa/temp$ rficonsole -indirect-read 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 further reduction.

Please note that the current working directory will be used as a temporary storage location! Thus by running RFI Console 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.

Once the flagger has finished, it will print statistics of the flagged data per channel and polarization. E.g.:

```
Summary of RFI per channel: (166,700,363 Hz - 166,894,149 Hz)
Channel   1-  8:   7.8%   7.8%   7.7%   7.7%   7.7%   7.7%   7.6%   7.6%
Channel   9- 16:   7.6%   7.6%   7.6%   7.6%   7.6%   7.7%   8.3%   8.7%
[..]
Channel 241-248:   7.5%   7.5%   7.5%   7.5%   7.5%   7.5%   7.6%   7.6%
Channel 249-255:   7.6%   7.6%   7.6%   7.6%   7.6%   7.5%   7.3%
Polarization statistics: XX: 3.3%, XY: 3.3%, YX: 3.3%, YY: 3.4%
```

After having ran RFI Console, the next step will normally be to use DPPP to average the data. When averaging a measurement set that has been flagged by the AOFlagger, your DPPP parset should not have any flagging steps in it.

4.2 Advanced settings with RFI Console

In this chapter, some techniques will be described which might be helpful when the default strategy is not good enough or when you would like to analyze the RFI and/or the flag results more closely.

4.2.1 Visualizing RFI and flags

A useful tool to analyze the RFI and flags in a measurement set is the "RFI Gui". To start it, make sure you have your X forwarded and start it with:

```
> rfigui
```

In the RFI Gui, there are options to alter flagging parameters, compare flags of different strategies, image individual baselines and several plotting options. For further explanation, refer to the RFI Gui tutorial and the references therein, which can be found on the following address:

http://www.astro.rug.nl/rfi-software/gui-tutorial.html

4.2.2 Changing flagging parameters

If you would like to change the flagger settings outside of the RFI Gui, it is possible to change RFI Console's 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 in two ways, either by:

- adding parameters to the `rfistrategy` executable. Run `rfistrategy` without parameters to get a list of options;
- or manually changing the file and altering the parameters. The strategy file is an xml text file, and can be edited by hand with any text editor such as nano or emacs.

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

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

4.2.3 RFI Console's parameters

The RFI Console program 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 -indirect-read -j 8 SB0.MS
```

In this case, we would use 8 threads to flag the data instead of 4, which is the default. More threads will require more memory. More than 8 threads on the clusters slow down the process, therefore it is not recommended.

Another option is to flag based on data in the `CORRECTED_DATA` column, instead of the default `DATA` column. This can be done with the `-column` option:

```
> rficonsole -column CORRECTED_DATA SB0.MS
```

With this option, you can flag the corrected data created by BBS. While this fixes bad solutions, flagging corrected data (with any flagger) might not be a good approach, thus this option is BEING TESTED.

4.2.4 Using the direct reading mode

Since AOFlagger used to be limited by IO seeking and not by cpu performance, the indirect read 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. If you do not want RFI Console to rewrite the set, you can leave out the "indirect-read" option.

4.2.5 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. 11, 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.

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. 11, the output looks like this:

```
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)
[...]
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,
```

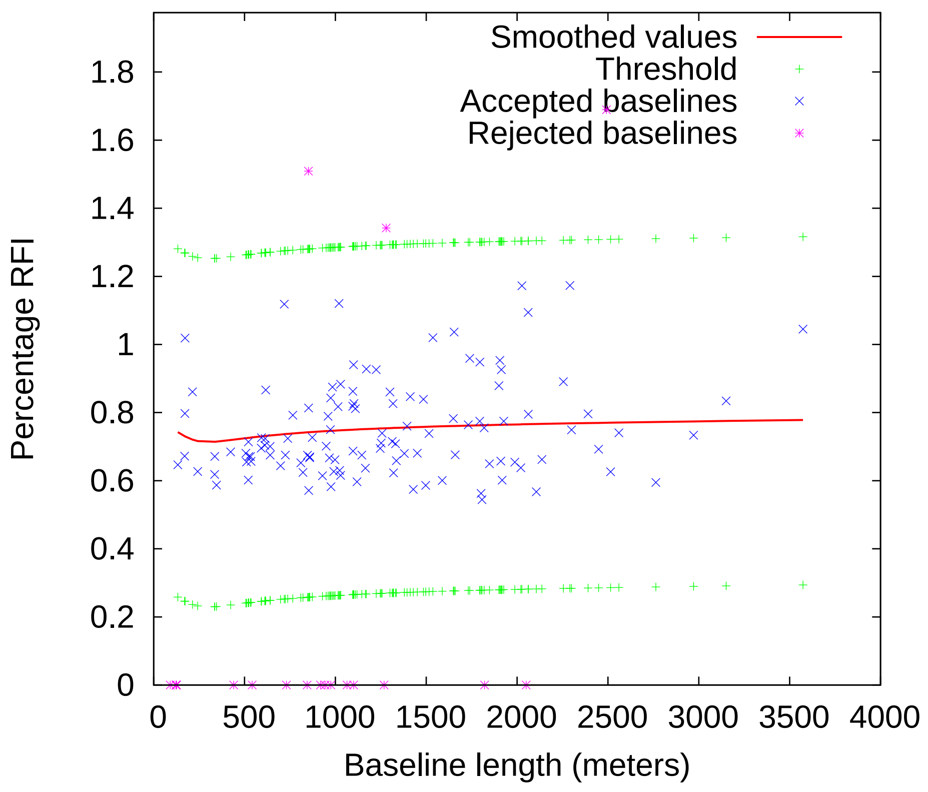


Figure 11: The percentage of RFI vs baseline length for sub band 127 of the observation `L2010_08567`. The bad baselines (in purple) will be reported in the output of RFI Console.

```
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 `DPPP.parset` to perform the baseline flagging.

4.3 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 (http://arxiv.org/abs/1002.1957).
- A LOFAR RFI detection pipeline and its first results, A.R. Offringa et al., Proceedings of Science, RFI2010 (http://arxiv.org/abs/1007.2089).

5 The Default Pre-Processing Pipeline (DPPP)[27]

DPPP (default pre-processing pipeline) is the flagging and averaging routine for LOFAR data. It is capable of flagging the data (using various flaggers, including the AOFlagger - Sect. 4) and average them to produce data sets clean of RFI and ready for the calibration.

Recently DPPP has been updated to incorporate a fast gain calibration step. Calibration with DPPP is described in Chapter 6.

DPPP can perform the following operations (see following sections for more details):

1. Flagging (automatic or manual)
2. Averaging in time and/or frequency
3. Phase shift to another phase center
4. Count flags and writing the counts into a table for plotting purposes.
5. Combine subbands into a single MeasurementSet
6. Demix and subtract A-team sources
7. Add stations to form a superstation
8. Filter out baselines and/or channels
9. Predict a sky model
10. Apply the LOFAR beam model
11. Do gain calibration
12. Apply calibration solutions
13. An arbitrary step defined in user code (C++ or Python)

Each of these operations works on the output of the previous one in a streaming way (thus without intermediate writes to disk). The steps can be combined in any way. The same type of step can be used multiple times with probably different parameters.

The ability to execute an arbitrary DPPP step is implemented by means of dynamically loadable shared libraries. It is described in more detail in Sect. 5.4.

The input to DPPP is any (regularly shaped) MeasurementSet (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 specifying which spectral window to use. The data in the given column are piped through the flagging and averaging steps defined in the parset file (See Sect. 5.2) and finally written. This makes it possible to e.g. flag at the full resolution, average, flag on a lower resolution scale, perform more averaging, and finally write the data to a new MeasurementSet.

The output can be a new MeasurementSet, but it is also possible to update the flags or data in the input. When averaging or phase-shifting to another phase center is done, the only option is to create a new MeasurementSet.

[27]This section is maintained by Ger van Diepen (`diepen[at]astron[dot]nl`) and David Rafferty (`rafferty[at]strw[dot]leidenuniv[dot]nl`).

It is possible to combine multiple MeasurementSets into a single spectral window. In this way multiple subbands can be combined into a single MeasurementSet. Note this is different from pyrap's `msconcat` command, because `msconcat` keeps the individual spectral windows, while DPPP combines them into one.

Detailed information on DPPP can be found here[28]. For specific questions regarding this software, you can contact the software developers, Tammo Jan Dijkema (dijkema[at]astron.nl) or Ger van Diepen (diepen[at]astron.nl).

5.1 Various ways to use DPPP

Several operations can be done using DPPP varying from copying a MeasurementSet to combining several MeasurementSets and from simple or advanced flagging to phase-shifting the data to another phase reference direction.

5.1.1 Basic usage

To run DPPP, you should specify in a `parset` file (see Sect 5.2) which are the operations to perform on the data and which are the parameters that you want to use. You can copy this `DPPP.parset` file into your working directory by typing:

```
cp /globaldata/COOKBOOK/Parset/DPPP.parset .
```

After this preparation, you can make the LOFAR software available and run the task by typing:

```
> use LofIm
> DPPP
```

If no argument is given, DPPP will execute the `DPPP.parset` file. If a parset file has a different name, it can be given as argument to the DPPP command like:

```
> DPPP some.parset
```

It is possible to override any in the parset file by adding them to the command line:

```
 > DPPP some.parset msin=other.MS msout=other-avg.MS
```

Outputs are printed to screen, including the percentage of data that have been flagged for each antenna and frequency channel.
The next sections contain some examples of parset files. They only show a subset of the available parameters.

5.1.2 Copy a MeasurementSet and calculate weights

The raw LOFAR MeasurementSets are written in a special way. To make them appear as ordinary MeasurementSets a special so-called storage manager has been developed, the `LofarStMan`. This storage manager is part of the standard LOFAR software, but not of a package like CASA. To be able to use CASA to inspect the raw LOFAR data, a copy of the MeasurementSet has to be made which can be done with a parset like:

[28]`http://www.lofar.org/wiki/doku.php?id=public:user_software:ndppp`

```
msin = in.ms
msin.autoweight = true
msout = out.ms
steps = []
```

Apart from copying the input MS, it will also flag NaNs and infinite data values. Furthermore the second line means that proper data weights are calculated using the auto-correlations. The latter is very useful, because the LOFAR online system only calculates simple weights from the number of samples used in a data value.

5.1.3 Count flags

The percentages of flagged data per baseline, station, and frequency channel can be made visible using:

```
msin = in.ms
msout =
steps = [count]
```

The output parameter is empty meaning that no output will be written. The `steps` parameter defines the operations to be done which in this case is only counting the flags.

5.1.4 Preprocess a raw LOFAR MS

The following example is much more elaborate and shows how a typical LOFAR MS can be preprocessed.

```
msin = in.ms
msin.startchan = nchan/32    #1
msin.nchan = nchan*30/32
msin.autoweight = true
msout = out.ms
steps = [flag,avg]           #2
flag.type = aoflagger        #3
flag.memoryperc = 25
avg.type = average           #4
avg.freqstep = 60
avg.timestep = 5
```

1. Usually the first and last channels of a raw LOFAR dataset are bad and excluded. Because the number of frequency channels of a LOFAR observations can vary (typical 64 or 256), an elaborate way is used to specify the first and number of channels to use. They are specified as expressions where the 'variable' `nchan` is predefined as the number of input channels.
2. Two operations have to be done: flagging followed by averaging. The parameters for these steps are specified thereafter using the step name. Note that a step name can be anything, but for clarity meaningful names should be used.
3. A step is defined by various parameters. Their names have to be prefixed with the step name. In this way it is possible to have multiple steps of the same type in a single DPPP run.

First the type of step has to be defined, because the name is only a name. In this case the aoflagger is used, an advanced data flagger developed by André Offringa. This flagger works best for large time windows, so it tries to collect as many data in memory as possible. However, to avoid memory problems this step will not use more than 25% of the available memory.

4. The next step is averaging the data, 60 channels and 5 time slots to a single data point. Averaging is done in a weighted way. The new weight is the sum of the original weights.

In DPPP the data flows from one step to another. In this example the flow is read-flag-average-write. The data of a time slot flows to the next step as soon as a step has processed it. In this case the flag step will buffer a lot of time slots, so it will take a while before the average step receives data. Using its parset parameters, each step decides how many data it needs to buffer.

5.1.5 Update flags using the preflagger

The preflagger step in DPPP makes it possible to flag arbitrary data, for example baselines with international stations.

```
msin = in.ms
msout =
steps = [flag]
flag.type = preflagger
flag.baseline = ![CR]S*
```

No output MS is given meaning that the input MS will be updated. Note that the `msout` always needs to be given, so one explicitly needs to tell that an update should be done.
The preflagger makes it possible to flag data on various criteria. This example tells that baselines containing a non core or remote station have to be flagged.
Note that in this way the baselines are flagged only, not removed. The `Filter` step described hereafter can be used to remove baselines or channels.

5.1.6 Remove baselines and/or channels

The filter step in DPPP makes it possible to remove baselines and/or leading or trailing channels. In fact, it should be phrased in a better way: to keep baselines and channels. An output MS name must be given, because data are removed physically.

```
msin = in.ms
msout = out.ms
steps = [filter]
filter.type = filter
filter.baseline = [CR]S*
```

If this would be the only step, it has the same effect as using `msselect` with `deep=true`.
The filter step might be useful to remove, for example, the superterp stations after they have been summed to a single superstation using a `stationadder` step.

The filter step has the capability (using the `remove` parameter) to remove the stations not being used from the ANTENNA subtable (and other subtables) and to renumber the remaining stations. This will also remove stations filtered out in previous steps (e.g., `msselect`), even if the filter step itself does not filter on baseline. In this way it can be used to 'normalize' a MeasurementSet.

5.1.7 Combining stations into a superstation

The stationadder step in DPPP makes it possible to add stations incoherently forming a superstation. This is particularly useful to combine all superterp stations, but can, for instance, also be used to add up all core stations.
This step does not solve or correct for possible phase errors, so that should have be done previously using BBS. However, this might be added to a future version of DPPP.

```
msin = in.ms
msout = out.ms
steps = [add]
add.type = stationadder
add.stations = {CSNew:[CS00[2-6]*]}
```

This example adds stations CS002 till CS006 to form a new station CSNew. The example shows that the parameter value needs to be given in a Python dict-like way. The stations to be added can be given as a vector of glob patterns. In this case only one pattern is given. Note that the wildcard asterix is needed, because the station name ends with LBA or HBA (or even HBA0 or HBA1).

In the example above the autocorrelations of the new station are not written. That can be done by setting parameter `autocorr`. By default they will be calculated from by summing the autocorrelations of the input stations. By setting parameter `sumauto` to false, they are calculated from the crosscorrelations of the input stations.

5.1.8 Update flags for NaNs

Currently it is possible that BBS writes NaNs in the CORRECTED_DATA column. Such data can easily be flagged by DPPP.

```
msin = in.ms
msin.datacolumn = CORRECTED_DATA
msout = .
steps = []
```

DPPP will always test the input data column for NaNs. However, if no steps are specified, DPPP will not update the flags in the MS. An update can be forced by defining the output name as a dot. Giving the output name the same name as the input has the same effect.

5.1.9 Creating another data column

When updating a MeasurementSet, it is possible to specify another data column. This can, for instance, be used to clone the data column.

```
msin = in.ms
msin.datacolumn = DATA
msout = .
msout.datacolumn = CORRECTED_DATA
steps = []
```

In this way the MeasurementSet will get a new column CORRECTED_DATA containing a copy of DATA. It can be useful when thereafter, for example, a python script operates on CORRECTED_DATA.

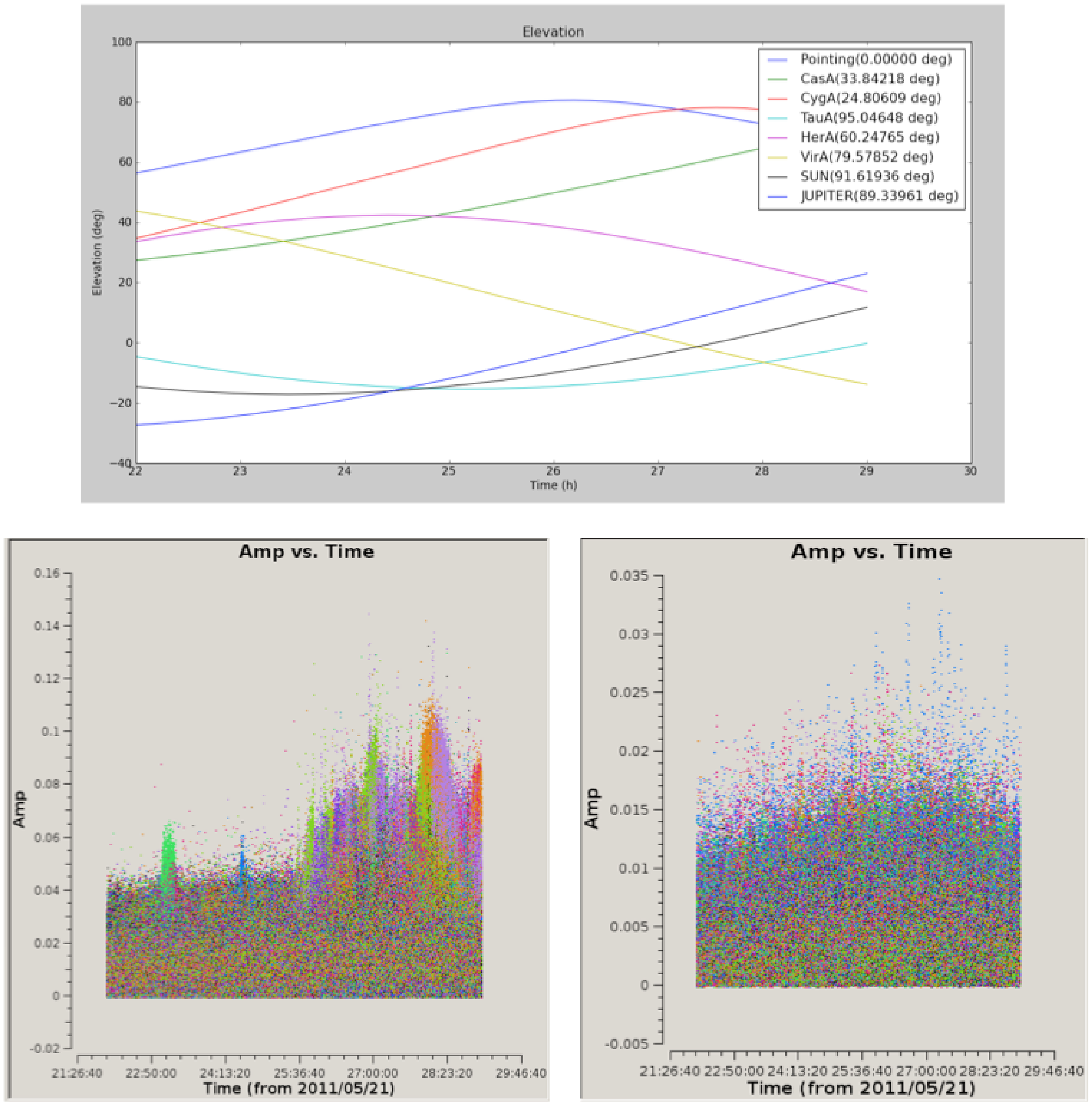


Figure 12: *Top:*the elevation and angular distance of the A-team from the target field centered on B1835+62. This plot shows that during this observation CygA and CasA are very high in elevation and pretty close to our target (24 and 33 degrees, respectively). *Bottom left:* target visibilities before demixing - the interference of Cas A and Cyg A with the target visibilities is the cause of the bump in the data in the second part of the observation. *Bottom right:* the contributions of the two A-tam sources is gone. This is particularly evident in the second part of the observation.

5.1.10 Demixing

The so-called "demixing" procedure should be applied to all LBA (and sometimes HBA) data sets to remove from the target visibilities the interference of the strongest radio sources in the sky (the so called A-team: CasA, CygA, VirA, etc...). Removing this contribution is essential to make it possible to properly calibrate the target field. To understand whether demixing is needed for your data, you are suggested to inspect the elevation of the A-team sources during your observation. By combining this information with the angular distance of the A-team from your target, you can have a clear picture of how critical is to apply this algorithm to your data to improve the calibration and imaging of the visibilities. This overview is provided by the script `plot_Ateam_elevation.py`, which is described in Sect. 2.6.

There are two ways to do the demixing:

- The old demixer will demix in the same way for the entire observation without taking temporal variations into account. One can define:
  - The baselines to use.
  - The (A-team) sources to solve for and to subtract.
  - If the target has to be ignored, solved or deprojected.
  - Possibly different time and frequency averaging factors to use for demix and subtract.
- Recently a new demixing scheme (designed by Reinout van Weeren) has been added to DPPP. Basically it works the same as the old demixer, but for each time window it estimates the data by evaluating a rough model of the A-team sources and target. Using those data it tests which sources have to be demixed, which baselines should be used, and if the target has to be ignored, solved, or deprojected. The LOFAR beam is taken into account in estimate, solve, and subtract. In this scheme one can specify:
  - A detailed model of the A-team sources to be used in the solve/subtract.
  - A rough model of the A-team sources to be used in the estimation. If not given, the detailed model is used.
  - A model of the target field which can be obtained using e.g. gsm.py.
  - The baselines to be used in the demixer. Note that the estimation step might exclude baselines for a given time window.
  - Various threshold and ratio values to test which sources, etc. to use.

Below, an example old demixer parset is given:

```
msin = in.ms
msout = out.ms
steps = [demix]
demix.type = demixer
demix.subtractsources=[CygA, CasA, VirA]
demix.targetsource=3C196
demix.freqstep=16
demix.timestep=10
```

Following this example, the source models of CygA, CasA, and VirA will be subtracted with the gain solutions calculated for them. The target source model is also used to get better gain solutions for the A-team sources.

If no source model is given for the target, the target direction is projected away when calculating the gains. This should not be done if an A-team source is close to the target. Currently, Science Support is investigating how close it can be. If too close, one should specify

```
demix.ignoretarget=true
```

Examples of demixing performance on real data are given in Figure 12.

5.1.11 Combine MeasurementSets

For further processing it can be useful to combine preprocessed and calibrated LOFAR MeasurementSets for various subbands into a single MeasurementSet. In this way BBS can run faster and can a single image be created from the combined subbands.

```
msin = somedirectory/L23456_SAP000_SB*_uv.MS.dppp
msin.datacolumn = CORRECTED_DATA
msin.baseline = [CR]S*&
msout = L23456_SAP000_SBcomb_uv.MS.dppp
steps = []
```

The first line shows that a wildcarded MS name can be given, so all MeasurementSets with a name matching the pattern will be used. The data of all subbands are combined into a single subband and the meta frequency info will be updated accordingly.
The second line means that the data in the CORRECTED_DATA column will be used and written as the DATA column in the output MS.
The third line means that only the cross-correlations of the core and remote stations are selected and written into the output MS. Note this is different from flagging the baselines as shown in the preflagger example. Input selection means that non-matching baselines are fully omitted, while the preflagger only flags baselines.
Note that no further operations are needed, thus no steps are given. However, it is perfectly possible to include any other step. In this case one could use `count`.

It is important to note that subbands to be combined should be consecutive, thus contiguous in frequency. Otherwise BBS might not be able to handle the MS. This means that the first and last channels of an MS should not be removed, but flagged instead using the preflagger.

5.1.12 Advanced multi-step example

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 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.

```
#####################################################
#
#  DPPP.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]

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

# Detect RFI using a median flagger
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         # synonym for average
avg1.freqstep = 256        # average a factor of 256 in frequency
avg1.timestep = 1          # is the default; no averaging in time

# Do another median flagging step on the averaged data
flag2.type=madflagger
flag2.threshold=3
flag2.timewindow=51

# Compress in time.
avg2.type = squash
avg2.timestep = 5          # average a factor of 5 in time
```

Note that the `count` step counts percentages of data flagged until this step. Each flag step also shows percentages of data flagged, but only by that flag step.
Because the default step type is the name of the step, the count step does not need further parameters. However, it is possible to specify some.

5.2 The ParSet File

As shown in the examples in the previous section, the steps to perform the flagging and/or averaging of the data have to be defined in the parset file. The steps 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.

The name of the parset file needs to be given as the first (and only) argument to the DPPP command. It defaults to `DPPP.parset`.

A description of all the parameters that can be used in DPPP can be found online at the address

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

5.2.1 Input / output parameters

A description of the input/output parameters of DPPP is given 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.

When combining multiple MSs into a single one, the names of the input MSs can be given in two ways using the `msin` argument.

- The name can be wildcarded as done in, say, bash using the characters *, ?, [], and/or . The directory part of the name cannot be wildcarded though. For example,

  ```
  msin=L23456\_SAP000\_SB*\_uv.MS
  ```

- A list of MS names can be given like `msin=[in1.ms, in2.ms]`.

The MSs will be ordered in frequency unless `msin.orderms=false` is given.

It is possible to select baselines to use from the input MS. If a selection is given, all baselines not selected will be omitted from the output. Note this is different from the Preflagger where data flags can be set, but always keeps the baselines.

LOFAR data are written and processed on the CEP2 cluster in Groningen. This cluster consists of the head node lhn001 and the compute/storage nodes locus001..100. Different subbands are stored on different nodes, and it may be necessary to search them all for the required data. MeasurementSets are named in the format `LXXXXX_SAPnnn_SBmmm_uv.MS`, where `L` stands for LOFAR, `XXXXX` is the observation number, `nnn` is the subarray pointing (beam), and mmm is the subband.
For example, `/data/L32667_SAP000_SB010_uv.MS`

msout The `msout` step defines the output. The input MS is updated if an emtpy output name is given.

Data should be written to `/data/scratch/<username>` (which may need creating initially). 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 `DPPP` 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 `/globaldata/COOKBOOK/files`.
(For the curious, the cluster description is a simple ASCII file that should be straightforward to understand).

5.2.2 Flagging

The properties of the flagging performed through DPPP 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 (or unflag) 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.
- An `AOFlagger` step can be used to flag using the AOFlagger. Usually it is faster than using `rficonsole` itself, because it does not reorder the data. Instead it flags in a user-defined time window. It is possible to specify a time window overlap to reduce possible edge effects. The larger the time window, the better the flagging results. It is possible to specify the time window by means of the amount of memory to be used.
  The flagging strategy can be given in an rficonsole strategy file. Such a file should not contain a 'baseline iteration' command, because DPPP itself iterates over the baselines. Default strategy files exist for LBA and HBA observations (named LBAdefault and HBAdefault).
  Note that the `AOFlagger` flags more data if there is a large percentage of zero data in the time window. This might happen if zero data is inserted by DPPP for missing time slots in a MeasurementSet or for missing subbands when concatenating the MeasurementSets of multiple subbands.
  By default the QUALITY subtables containing flagging statistics are written. They can be inspected using `aoqplot`.

5.2.3 Averaging

The properties of the averaging performed through DPPP 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

5.2.4 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	!	
AND	&&	&
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
```

5.3 MSSelection, antenna/baseline syntax

In order to select particular baselines, antennas, and baseline lengths, DPPP uses the CASA baseline selection syntax. 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

5.3.1 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 inh 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[29]. 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 (∼) as separator. The end value is inclusive. For example:
  `0~5,10~12,18,20`

- An antenna name, pattern, or regex can be preceded by a up-arrow / caret (^) meaning negation. For example:
  `^[CR]S*` for all international stations (i.e., no CS* and RS*)
  Note there is quite a difference between the negations with ^ and !. The first one applies to a station name/pattern, the second to the entire group. For example:

`^CS*&`	selects all cross-correlations between non-core stations.
`!CS*&`	selects all baselines not being a cross-correlation between core stations.
`!CS*&&*`	selects all baselines except those between a core and any other station.

  In the second example the auto-correlations between core stations are also selected as well as the baselines between core stations and remote or international stations. This is not the case in the first example. The third example is almost the same as the first one, but it also selects the auto-correlations of the remote and international stations.

5.3.2 Physical baseline length

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

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

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.

5.3.3 Some examples

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

```
    CS*
```
all baselines containing core stations.

```
    ^[CR]S*&
```
all cross-correlation baselines between international stations.

```
    ^[CR]S*&; !DE601* & DE605*
```
all cross-correlation baselines between international stations, except Effelsberg-Jülich.

```
    !CS*
```
all baselines not being a cross-correlation between core stations.

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

```
    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).

```
    ![CR]S* &&
```
all baselines containing European stations

```
    CS* && RS*; !  CS001 & RS*; CS001 & RS001
```
all cross- and auto-correlations between core and remote. However, the baselines between `CS001` and remote 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.

5.4 Arbitrary User DPPP Step

Besides the predefined DPPP steps like AOFlagger, Averager, etc., it is possible to use any user-defined DPPP step. Such a step has to reside in a shared library, that will dynamically be loaded by DPPP. The name of such a shared library has to be the step type name (without the prefix lib). DPPP will try to load a library libxxx.so (or .dylib on OS-X) when a step type xxx is unknown.

To make this a bit more flexible it is possible to define multiple steps in a single shared library. In such a case the step type name has to consist of 2 parts separated by a dot. The first part is the library name, the second part the step type in that library.
For example:

```
steps=[averager, mystep1, mystep2]
mystep1.type = mystep.stepa
mystep2.type = mystep.stepb
```

defines two user steps. Both step implementations reside in library `libmystep.so`. An example of a dynamically loaded step can be found in the LOFAR source code repository in LOFAR/CEP/DPPP/TestDynDPPP.

5.4.1 User Step in Python

The mechanism described above is used to make it possible to implement a user step in Python. The step type has to be `pythondppp` which has the effect that the library `libpythondppp.so` will be loaded. This library will look for two specific parset keys:

- `stepname.python.module`
  defines the Python module to be loaded. If the module name is not given, it defaults to the class name.

- `stepname.python.class`
  defines the class in that module to be used.

Note it is possible that a single Python module contains multiple classes.

The Python class has to be derived from `DPStep` which forms the superclass for all DPPP steps implemented in Python. A few functions must be implemented in the subclass. They are similar to the virtual functions in the C++ base class DPStep.

- `__init__(self, parsetDict)`
  is the constructor. It receives a dict containing the parset keys and values for this step. It must call the constructor of the superclass and can thereafter process the parset.
- `updateInfo(self, dpinfo)`
  handles the meta data. It receives a dict containing the meta data describing the observation. The names of the keys in the dict are the same as the names of the data members in the C++ class DPInfo without the prefix `its`. The step must return a dict with possibly changed meta data. Usually this dict can be empty, but when e.g. averaging is done, the step changes meta data like number of channels which must be returned in the dict.
  The function can allocate the arrays needed in the `process` function described below.
- `process(self, time, exposure)`
  is the heart of the step class. It is called for each time slot to process data for the baselines, channels, and polarisations in that time slot. It receives the time (MJD in sec) and exposure (sec). The data that are needed can be obtained using the functions `getData`, `getFlags`, `getWeights`, and/or `getUVW`. These functions read into existing numpy arrays which can be created (e.g., by `updateInfo`) using the functions `makeDataArray`, etc..
  The function `processNext` must be called at the end to execute the process function in the next DPPP step. It must supply a dict containing changed data. The keys in the dict are TIME, EXPOSURE, DATA, FLAGS, WEIGHTS, and/or UVW.
  Similar to the behaviour in existing steps like Averager and AOFlagger, a step can buffer data as needed. Once a time slot is ready for the next step, it must call `processNext`. In this call a dict containing all data fields must be supplied, because the C++ layer does not have the old data anymore.

The following functions have a default implementation in the superclass, but can/must be implemented in the subclass if needed.

- `finish(self)`
  is called at the very end of the processing. If data were buffered, it can be used to process the last data (and call processNext for them). Note that the C++ layer will call `finish` of the next step.
- `needVisData(self)`
  This function only needs to be implemented in a subclass if the step does not need the visibility data.
- `needWrite(self)`
  This function needs to be implemented in a subclass if it changes data (visibility data, flags, weights, and/or UVW) that needs to be written (or possibly updated) in the MeasurementSet.
- `show(self)`
  can show the parameters of the step. It should return a string (with newlines) that will be printed

by the C++ layer. An empty string is not printed.
The implementation in the superclass shows all parset keys, so usually this function does not need to be implemented.

- `showCounts(self)`
  can show possible counts (e.g., nr of flags set). It should return a string (with newlines) that will be printed by the C++ layer. An empty string is not printed.
- `showTimings(self, elapsedDuration)`
  can show possible timings of parts in this step. It should return a string (with newlines) that will be printed by the C++ layer. An empty string is not printed. It receives the elapsed time (sec) this entire step took. This time is always printed, so usually this function does not need to be implemented.
- `addToMS(self, msname)`
  can add information to the output MeasurementSet. This function will only be needed in very special cases where dedicated info needs to be added to the MS.

An example DPStep subclass can be found in the LOFAR source code repository in LOFAR/CEP/DPPP/PythonDPPP/test/tPythonStep.py. Its parset defines an increment that is applied to the TIME and UVW. It prints the sum of the various data arrays.
The file is shown below. Further down a possible parset is shown.

```
from lofar.pythondppp import DPStep
from lofar.parameterset import parameterset

class tPythonStep(DPStep):
    def __init__(self, parsetDict):
        # The constructor gets the subset of the DPPP parset containing
        # all keys-value pairs for this step.
        # Note: the superclass constructor MUST be called.
        DPStep.__init__(self, parsetDict)
        parset = parameterset(parsetDict)
        self.itsIncr = parset.getDouble('incr', 1)

    def updateInfo(self, dpinfo):
        # This function must be implemented.
        self.itsInfo = dpinfo
        # Make the arrays that will get the input buffer data from
        # the getData, etc. calls in the process function.
        self.itsData = self.makeArrayDataIn()
        self.itsFlags = self.makeArrayFlagsIn()
        self.itsWeights = self.makeArrayWeightsIn()
        self.itsUVW = self.makeArrayUVWIn()
        # Return the dict with info fields that change in this step.
        return {};

    def process(self, time, exposure):
        # This function must be implemented.
        # First get the data arrays needed by this step.
        self.getData (self.itsData);
        self.getFlags (self.itsFlags);
        self.getWeights (self.itsWeights);
        self.getUVW (self.itsUVW);
        # Process the data.
        print "process tPythonStep", time-4.47203e9, exposure, self.itsData.sum(),
              self.itsFlags.sum(), self.itsWeights.sum(), self.itsUVW.sum()
        # Execute the next step in the DPPP pipeline. TIME,UVW are changed.
        return self.processNext ({'TIME': time+self.itsIncr,
                                  'UVW': self.itsUVW+self.itsIncr})

    def finish(self):
        # Finish the step as needed.
        # This function does not need to be implemented.
        # Note: finish of the next step is called by the C++ layer.
        print "finish tPythonStep"

    def showCounts(self):
        # Show the counts of this test.
        # This function does not need to be implemented.
        return "   **showcounttest**"

    def addToMS(self, msname):
        # Add some info the the output MeasurementSet.
        # This function does not need to be implemented.
        print "addToMS tPythonStep", msname
```

The following parset executes the Python step twice. Note that for the first time the increment is not defined, so the default 1 will be used. Also note that, as shown in the second step, it is not needed to define the Python module because its name is the same as the class.

```
msin=tPythonStep_tmp.MS
msout=tPythonStep_tmp.msout
msout.overwrite=T

steps='[pystep1,pystep2]'

pystep1.type=PythonDPPP          # case-insensitive
pystep1.python.module=tPythonStep
pystep1.python.class=tPythonStep
pystep1.somekey=somevalue        # unused key

pystep2.type=pythondppp
pystep2.python.class=tPythonStep
pystep2.incr=3.5
```

5.5 Flag statistics

Several steps shows statistics during output about flagged data points.

- A MADFlagger and AOFlagger step show 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.

Furthermore the AOFlagger step will by default write some extra QUALITY subtables in the output MeasurementSet containing statistical information about its performance. These quality data can be inspected using the `aoqplot` tool.

5.6 Analyzing the data quality with `aoqplot`

Once you have succesfully run DPPP on the measurement sets in your observation, it is recommended that you validate the results of the flagger and get an impression of the quality of the full observation. For this, the `aoqplot` tool that is part of the LofIm build can be used.

Basically, the tool allows you to plot standard deviations and RFI percentages and some other quantities, over time, frequency, baselines and the time-frequency domain of the full observation, i.e., over all sub-bands. Since an observation can be several terabytes of data, the performance of standard tools to perform such analysis is an issue, and the `aoqplot` was designed to overcome this problem. Fig. 13 shows an example of the `aoqplot` interface, plotting the data standard deviations of an LBA set over the entire frequency range. It also allows one to plot differential statistics. “Differential” in this context means that the standard deviation is calculated over the difference between adjacent channels. Therefore, they quantify the noise, because the difference of signal in adjacent (3 kHz) channels is tiny and can be neglected. Differential quantities are prefixed with a “D”, such as DMean and DStdDev.

5.6.1 Usage

If your environment allows (see below), one can get the statistic plots of an observation of a DPPPed set, with the following command:

```
aoqplot yourobservation.gvds
```

The gvds file is given by the observatory and describes the observation, in particular the locations of the measurement sets. The `aoqplot` tool will now connect to all the nodes with an ssh connection, start a bash-session on the node and start a client there that connects back to your node and sends

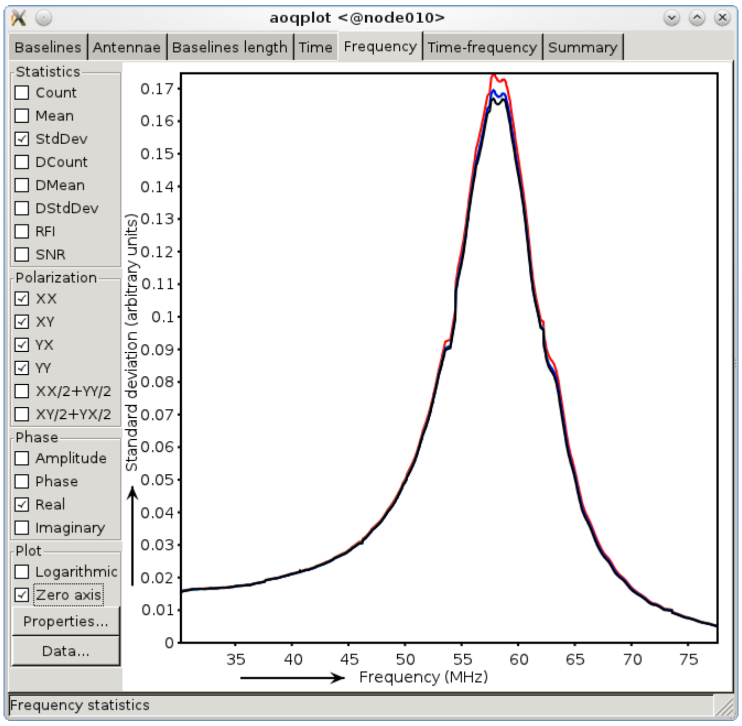


Figure 13: The `aoqplot` window showing the standard deviation of the data over frequency of a full observation.

the quality tables. Once all the tables have been received, a window will appear containing the plots. Collecting all the tables takes typically less than 5 seconds.

The use of ssh and bash requires that you should be able to ssh to all the involved nodes without manual intervention, and that the client (called "`aoremoteclient`", part of the LofIm daily build) is directly in your bash path after ssh-ing. Thus, after "`ssh locus001`" (or lce001), you should be able to start `aoremoteclient` within bash without a "use LofIm". The easiest way of doing this is by putting `LofIm` in `~/.mypackages`. If you have problems running the `aoqplot` due to your environment, please let us know.

When a node is not answering, or there is some error with the measurement set, the set will be skipped, an error will be given describing the problem and the statistics will be collected without those measurement sets. If none of the measurement sets can be queried, you will see a dialog window with many error messages and the window will not appear thereafter. If you can not determine the cause of this, please let us know. The software is currently (January 2012) still experimental.

The `aoqplot` can also be used on individual sub-bands by putting the measurement set filename in the command line, e.g.:

```
aoqplot SB000.MS
```

5.6.2 Analyzing the statistics

The `aoqplot` tool provides the following statistics:

- **Count**: the number of samples that are left after flagging of the data. This should normally be fairly constant over time, frequency and baselines, apart from a few imprints of RFI that lower the number of available samples. Since the flags of the complex values of different polarizations are normally equal, there's no use in looking at this statistic for polarizations or real/imaginary components indivually.

- **Mean**: the mean of the data. If you are observing a strong source (such as a calibrator), this value should contain structure over time, frequency and baselines. Note that if you for example plot the mean over time, each sample in the plot shows the mean of that timestep over all baselines and frequencies. Therefore, if your source is not in the phase centre, it will be supressed and can even be averaged out, because sources outside the phase centre contribute sinusoidally and will cancel out. If your source *is* in the phase centre, the Mean is a very good representation of the strength of the signal. Together with an estimate of the noise, this can be used to calculate the signal-to-noise ratio during the observation. If you know the approximate flux density of the source, you can estimate the gain during the observation and, together with an estimate of the noise, calculate a rough estimate of the system noise. Cross polarizations can be checked to see if there was significant differential Faraday rotation during the observation.

- **StdDev**: the standard deviation of the data after flagging. The standard deviation should not have significant imprints of RFI. In good data, one generally sees about three significant spikes in HBA (in $\pm$ 115-163 MHz) and zero spikes in LBA (>30 MHz, an example is given in Fig. 13). The standard deviation is rather sensitive for low-level RFI, and a few RFI spikes do not seem to hurt calibration at this point (please report if you think otherwise). If there are time or frequency ranges at which the standard deviation is significantly different, try to select different polarizations and use the different domains (time-frequency, baseline, time, frequency, ...) to see if you can localize the guilty data range. The position of the Sun and the Milky Way in the sky can significantly change the standard deviation. Because the StdDev includes the variance of the signal, it is recommended also to look at the DStdDev.

- **DCount**, **DMean** and **DStdDev** are similar to the above statistics, but are calculated over the differences of samples (after flagging) in adjacent channels. They contain therefore very little contribution of the signal, and can be used to get an accurate estimate of the noise. They have been normalized to represent the same units as their counterpart values. The DMean should be close to zero, as the signal should be subtracted out, and the noise should average out (it is mainly there because it is easy to calculate, but it is often more helpful to look at Mean and DStdDev).

- **RFI**: the amount of RFI found by the flagger. The 'base level' of RFI is 2–5%, but can contain a few spikes over time or frequency that go up to 20%–100% at times. This is normally not a problem. Sub-bands or stations with significant different RFI levels (either 0% or $>\sim$ 5%) often indicate an issue with the station. Such problems are often also reflected in the standard deviations. Different polarizations and real/imaginary values have equal RFI ratios.

- **SNR**: the signal-to-noise ratio. It is calculated by Mean / DStdDev. This value is only accurate if you are observing a source in the phase centre, due to the reasons mentioned in the paragraph for the Mean value.

5.6.3 Background information

The `aoqplot` tool works together with DPPP. Recent versions (>21 December 2011) of DPPP will add so-called quality statistic tables to a measurement set. These tables circumvent having to read the entire DATA column of a measurement set to get the basic statistics. The way they are stored is described in the quality statistics proposal written by André Offringa[30]. Because the statistics plotting

[30] `offringa[at]astro[dot]rug[dot]nl`

tool require these tables, you can not directly plot statistics of measurement sets that are averaged by an older DPPP, or have not been averaged at all.

The statistics are calculated individually for the real and complex values. This is not common when treating complex values, but does allow easy interpretation. This means that μ_r and σ_r, the real mean and real standard deviations respectively, are calculated as:

$$\mu_r = \frac{1}{N} \sum_{x \in X} \mathtt{real}(x) \tag{1}$$

$$\sigma_r^2 = \frac{1}{N} \sum_{x \in X} \left(\mathtt{real}(x) - \mu_r\right)^2 \tag{2}$$

If you select “amplitude” in the `aoqplot` user interface, the actual plotted quantity is:

$$|\sigma| = \sqrt{\sigma_r^2 + \sigma_i^2}, \tag{3}$$

i.e., the amplitude of the standard deviation of the real and imaginary components, not the standard deviation over the amplitudes. The same holds for the “XX+YY” and “XY+YX” check boxes, which represent the sum of the statistic, not the statistic over the sums.

If, for some reason, you want to use `aoqplot`, but do not want to use DPPP to average the data, a different way of adding the required quality statistics to a measurement set is by using the `aoquality` tool, part of the LofIm build. The general usage is:

```
 aoquality collect SB000.MS
```

The `aoquality` also has some options for retrieving statistics on the command line. Run `aoquality` without parameters to get a list of options.

5.7 Additional information: 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. Flagging tasks in CASA include `FLAGDATA` and `FLAGCMD` for command-line based flagging. The task `PLOTMS` offers GUI-based flagging. `PLOTXY` can also be used for manual flagging, but users should be aware that it is being deprecated in favor of `PLOTMS` and may not be available in future releases of CASA. Once the CASA `PLOTMS` 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.

CASA also provides two algorithms, RFLAG and TFCROP, for automatic RFI flagging. These algorithms are available as options within the `FLAGDATA` task. For more information on their usage, we suggest users consult Chapter 3 of the latest version of the CASA Cookbook.

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 DPPP, but also by manual flagging in CASA or using the CASA `SPLIT`[31] task or the python script `split_ms_by_time.py` (Section 2.7.2). Splitting out part of a MeasurementSet can be done as part of the distributed pipeline and will most likely be necessary until more robust flagging routines are implemented.

[31]The `SPLIT` task will be deprecated in favor of the `MSTRANSFORM` task beginning with CASA v.4.1.0.

6 Gain calibration with DPPP[32]

The most common gain calibration procedures can be performed with DPPP. Using DPPP (instead of BBS, Chapter 7) makes a considerable difference in speed: DPPP calibration is at least 10 times faster.

However, BBS can handle more calibration scenarios, which sometimes makes it necessary to use BBS.

The calibration problem that DPPP can solve is the following: find a set of Jones matrices $\{\mathbf{G}_p\}$ (one for every station p) which corrects the measured visibilities $\mathbf{V}_{pq}$ to closely resemble the model visibilities $\mathbf{M}_{pq}$ (for all baselines pq), i.e. minimize

$$\|\mathbf{V}_{pq} - \mathbf{G}_p \mathbf{M}_{pq} \mathbf{G}_q^H\| \tag{4}$$

The matrices $\mathbf{G}$ will be referred to as *gain matrices* although they correct for more than just electrical gains.

6.1 Calibration variants

There are various options to restrict the shape of $\mathbf{G}$. The main difference is whether to solve for the amplitude of the solutions or only for the phase. Also, the number of free parameters can be restricted.

Shape	Calibration type	Free parameters
$\mathbf{G}_p = \begin{pmatrix} A_{xx}^{(p)} e^{\phi_{xx}^{(p)}} & A_{xy}^{(p)} e^{\phi_{xy}^{(p)}} \\ A_{yx}^{(p)} e^{\phi_{yx}^{(p)}} & A_{yy}^{(p)} e^{\phi_{yy}^{(p)}} \end{pmatrix}$	diagonal	4
$\mathbf{G}_p = \begin{pmatrix} A_{xx}^{(p)} e^{\phi_{xx}^{(p)}} & 0 \\ 0 & A_{yy}^{(p)} e^{\phi_{yy}^{(p)}} \end{pmatrix}$	diagonal	4
$\mathbf{G}_p = \begin{pmatrix} e^{\phi_{xx}^{(p)}} & 0 \\ 0 & e^{\phi_{yy}^{(p)}} \end{pmatrix}$	phaseonly	2
$\mathbf{G}_p = \begin{pmatrix} e^{\phi^{(p)}} & 0 \\ 0 & e^{\phi^{(p)}} \end{pmatrix}$	scalar phase	1

6.2 Make a skymodel

To perform a calibration, you need a sky model (see Section). You can get one from the catalogs in `gsm.py` (see Section), or make your own. To make the sky model (in text format) readable by DPPP, it needs to be converted from plain text to a *sourcedb*. That is done with the program `makesourcedb`. Usually the sourcedb is called 'sky' and copied into the data set you're reducing. If you put it elsewhere, it is customary to give it the extension `.sourcedb`.

[32]This section is maintained by Tammo Jan Dijkema (`dijkema[at]astron[dot]nl`).

```
makesourcedb in=my.skymodel out=L123.MS/sky format='<'
```

The part `format='<'` is necessary to convince makesourcedb that the format is given by the first line in the file.

It is also possible to calibrate on model visibilities – in this case no sky model is necessary. See the online documentation of DPPP, the parameter to look for is `usemodelcolumn`.

6.3 Calibration

To perform a phase only calibration, the following parset can be given to DPPP.

```
msin=L123.MS
msout=
steps=[gaincal]
gaincal.sourcedb=L123.MS/sky
gaincal.parmdb=L123.MS/instrument
gaincal.caltype=phaseonly
gaincal.solint=2
```

The part `solint=2` specifies that we only want one solution for every two time intervals. This can improve the signal to noise ratio – but one should have a physical argument that tells that the solutions do not change within the solution interval. Currently one solution is computed that is assumed constant over the entire band, more flexibility will be added in the future.

The parset above performs a phase only calibration, and stores the calibration result in the *parmdb* (parameter database) `L123.MS/instrument`. This file will be created if it is not there yet – in fact it is better if it is not there yet. Note that it is a convention to save the calibration tables in a file (casa table) called `instrument` in the data set being reduced. If you store it outside the data set, the convention is to give it the extension `.parmdb`.

The solution table can (and should!) be inspected with `parmdbplot.py`, see Section 7.12. Note that this calibration step does not yet change the visibilities.

6.4 Applying solutions

To apply the calibration solutions in DPPP, the step *applycal* can be used. The following parset applies the solutions that were obtained by gaincal[33].

```
msin=L123.MS
msout=.
msout.datacolumn=CORRECTED_DATA
steps=[applycal]
gaincal.parmdb=L123.MS/instrument
```

It is a convention to write the output to the column `CORRECTED_DATA`, to avoid changing the original data column `DATA`.

6.5 Transferring solutions and the beam

When transferring solutions from a calibrator to a target, the sensitivity of the beam across the sky needs to be taken into account: the instrument does not have the same sensitivity at the position of the

[33]Ideally, applycal could be called within the same call of DPPP, so that the visibilities are read only once. This has not been implemented yet.

calibrator field as at the position of the target field. You can compensate for this by using a model for the LOFAR beam. Effectively, then instead of equation 4 the following equation is solved for $\mathbf{G}_p$:

$$||\mathbf{V}_{pq} - \mathbf{G}_p \mathbf{B}_p \mathbf{M}_{pq} \mathbf{B}_q^H \mathbf{G}_q^H|| \tag{5}$$

In the case of transferring solutions, the calibration is usually about the amplitude and the calibration type should be either `diagonal` or `fulljones`.

A parset for calibrating on the calibrator field, taking the beam into account, is given below:

```
msin=L123.MS
msout=
steps=[gaincal]
gaincal.sourcedb=L123.MS/sky
gaincal.parmdb=L123.MS/instrument
gaincal.caltype=diagonal
gaincal.usebeammodel=true
```

6.6 Applying the beam

When applying the solutions of the calibrator to the target, you should probably not apply the beam, so that another round of calibration is possible afterwards. Only after you are done with all calibration, the beam should be applied (just before imaging). Applying the beam is possible with

```
msin=L123.MS
msin.datacolumn=CORRECTED_DATA
msout=.
msout.datacolumn=CORRECTED_DATA
steps=[applybeam]
```

7 Calibration with BBS[34]

BBS is a software package that was designed for the calibration and simulation of LOFAR data. This section provides a practical guide to reducing LOFAR data with BBS. Do not expect a description of the optimal way to calibrate LOFAR data that works on all possible fields. Much still has to be learned about the reduction of LOFAR data. This chapter should rather be viewed as a written record of the experience gained so far, through the efforts of many commissioners and developers.

NOTE: the most common calibration scenarios can now be performed in DPPP, which is a lot faster (at least 10 times). Please first see if your calibration can be done with DPPP, see Chapter 6.

7.1 Overview

The sky as observed by LOFAR is the ”true” sky distorted by characteristics of the instrument (station beam, global bandpass, clock drift, . . .) and the environment (ionosphere). The goal of calibration (and imaging) is to compute an accurate estimate of the “true” sky from the distorted measurements.

The influence of the instrument and the environment on the measured signal can be described by the *measurement equation*. Calibration involves solving the following inverse problem: Given the measured signal (observations) and a parameterized measurement equation (model), find the set of parameter values that minimizes the difference between the model and the observations.

The core functionality of BBS can conceptually be split into two parts. One part concerns the simulation of visibilities given a model. The other part concerns estimating best-fit parameter values given a model and a set of observed visibilities. This is an iterative procedure: A set of simulated visibilities is computed using the current values of the parameters. Then, the values of the parameters are adjusted to minimize the difference between simulated visibilities and observed visibilities, and an updated set of simulated visibilities is computed. This continues in a loop until a convergence criteria is met.

BBS has two modes of running. The first is to run as a stand-alone tool to calibrate one subband. If a single subband does not contain enough signal, BBS offers the possibility to use data from multiple subbands together for parameter estimation. This is called *global* parameter estimation. In this chapter, we will focus on the stand-alone version; only in Section 7.10 we will treat the global solve.

As input, BBS requires an observation (one or more subbands / measurement sets), a source catalog (see Section 7.3), a table of initial model parameters (see Section 7.5), and a configuration file (also called *parset*, see Section 7.8) that specifies the operations that need to be performed on the observation as a whole. As output, BBS produces a processed observation, a table of estimated model parameters, and a bunch of log files.

7.2 Usage

To calibrate a single MS, execute the `calibrate-stand-alone` script on the command line:

```
> calibrate-stand-alone -f <MS> <parset> <source catalog>
```

The important arguments provided to the `calibrate-stand-alone` script in this example are:

- `<MS>`, which is the name of the MS to process.

[34]This section is maintained by Tammo Jan Dijkema (`dijkema@astron.nl`). Most of it is written by Joris van Zwieten and Reinout van Weeren.

- `<parset>`, which defines the reduction (see Section 7.8).
- `<source catalog>`[35], which defines a list of sources that can be used for calibration (see Section 7.3).
- The `-f` option, which overwrites stale information from previous runs.

You can run the script without arguments for a description of all the options and the mandatory arguments, such as the `-v` option to get a more verbose output.

7.3 Source catalog

The source catalog defines the sources that can be used (referred to) in the reduction. It is a plain text file that is translated by the `makesourcedb` tool into a form that BBS can use. (Internally, the script converts the catalog file to a sourcedb by running `makesourcedb` on it, before starting BBS.)

A catalog file can be created by hand using a text editor. Using tools like `awk` and `sed`, it is relatively straight-forward to convert existing text based catalogs into `makesourcedb` format. Various catalog files created by commissioners have been collected at `/globaldata/COOKBOOK/Models` on the CEP cluster.

Alternatively, a catalog file in `makesourcedb` format can be created using the `gsm.py` tool. This tool extracts sources in a cone of a given radius around a given position on the sky from the *Global Sky Model* or GSM. The GSM contains all the sources from the VLSS, NVSS, and WENSS survey catalogs. See Section 7.4 for more information about the GSM and `gsm.py`.

Below is an example catalog file for 3C196. In this example, 3C196 is modelled as a point source with a flux of 153 Jy and a spectral index of -0.56 with a curvature of -0.05212 at a reference frequency of 55.468 MHz.

```
# (Name,Type,Ra,Dec,I, ReferenceFrequency='55.468e6', SpectralIndex) = format
3C196, POINT, 08:13:36.062300, +48.13.02.24900, 153.0, , [-0.56, -0.05212]
```

Another example catalog file describes a model for Cygnus A. It is modelled as two point sources that represent the Eastern and the Western lobe respectively, with a flux ratio of 1:1.25.

```
# (Name, Type, Ra, Dec, I) = 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
```

For each source, values should be specified for the fields listed in the header line, in the same order. The only mandatory fields are: `Name`, `Type`, `Ra`, and `Dec`. 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).

If a field is left blank (e.g. the `ReferenceFrequency` in the 3C196 example above), the default value specified in the header line is used. If this is not available, then the application defined default value is used instead.

When a catalog contains sources defined by shapelets as well as point sources or Gaussian sources, it is easiest to create two catalog files (one containing the shapelet sources and one containing the rest). Using the `append` option of the `makesourcedb` tool, a single `parmdb` containing all the sources can then be created and fed to the `calibrate-stand-alone` script using the `--sourcedb` option.

[35]When the MS already contains a sourcedb, it is not necessary to specify a new source catalog if you're calibrating on the same sourcedb.

For more information about the recognized fields, default values, and units, please refer to the `makesourcedb` documentation on the LOFAR wiki[36].

7.3.1 Gaussian sources

A Gaussian source can be specified as follows.

```
# (Name, Type, Ra, Dec, I, Q, U, V, ReferenceFrequency='60e6', \
     SpectralIndex='[0.0]', MajorAxis, MinorAxis, Orientation) = format

sim_gauss, GAUSSIAN, 14:31:49.62,+13.31.59.10, 5,,,,,[-0.7], 96.3, 58.3, 62.6
```

Note that the header line beginning with "`# (Name, ...`" must be a single line, and has been spread over multiple lines here for clarity (indicated by a backslash). There is a known issue about the definition of the `Orientation`. This is only relevant if the Gaussian is not symmetric, i.e. the major axis and minor axis differ, and the Gaussian is far away from the phase center. Until this has been fixed, only use Gaussians that are close to the phase center.

7.3.2 Spectral index

The spectral index used in the source catalog file is defined as follows:

$$\log_{10}(S) = \log_{10}(S_0) + c_0 \log_{10}\left(\frac{\nu}{\nu_0}\right) + c_1\left[\log_{10}\left(\frac{\nu}{\nu_0}\right)\right]^2 + \ldots + c_n\left[\log_{10}\left(\frac{\nu}{\nu_0}\right)\right]^{n+1} \quad (6)$$

with ν_0 being the reference frequency specified in the `ReferenceFrequency` field, c_0 the spectral index, c_1 the curvature, and $c_2, \ldots, c_n$ the higher order curvature terms. The `SpectralIndex` field should contain a list of coefficients $[c_0, \ldots, c_n]$[37].

7.3.3 Rotation measure

For polarized sources, Stokes Q and U fluxes can be specified explicitly, or implicitly by specifying the intrinsic rotation measure RM, the polarization angle χ_0 at $\lambda = 0$, and the polarized fraction p.

In the latter case, Stokes Q and U fluxes at a wavelength λ are computed as:

$$\begin{aligned} Q(\lambda) &= p\, I(\lambda)\, \cos(2\chi(\lambda)) && (7) \\ U(\lambda) &= p\, I(\lambda)\, \sin(2\chi(\lambda)) && (8) \\ \chi(\lambda) &= \chi_0 + RM\, \lambda^2 && (9) \end{aligned}$$

Here, $I(\lambda)$ is the total intensity at wavelength λ, which depends on the spectral index of the source.

The intrinsic rotation measure of a source can be specified by means of the field *RotationMeasure*. The polarization angle χ_0 at $\lambda = 0$ and the polarized fraction p can be specified in two ways:

- Explicitly by means of the fields *PolarizationAngle* and *PolarizedFraction*.
- Implicitly, by means of the fields Q, U, and *ReferenceWavelength*.

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

[37]The old format of specifying the spectral index as the number of coefficients minus one (i.e. n), followed by $n+1$ separate fields `SpectralIndex:0`, `SpectralIndex:1`, ..., `SpectralIndex:n` is not supported anymore.

When specifying Q and U at a reference wavelength λ_0, the polarization angle χ_0 at $\lambda = 0$, and the polarized fraction p will be computed as:

$$\chi_0 = \frac{1}{2}\tan^{-1}(\frac{U}{Q}) - \lambda_0^2\, RM \tag{10}$$

$$p = \frac{\sqrt{(Q^2+U^2)}}{I(\lambda_0)} \tag{11}$$

Here, $I(\lambda_0)$ is the total intensity at reference wavelength λ_0, which depends on the spectral index of the source. Note that the reference wavelength λ_0 must be > 0 if the source has a spectral index.

7.4 GSM[38]

The *Global Sky Model* or GSM is a database that contains the reported source properties from the VLSS, WENSS and NVSS surveys. The `gsm.py` script can be used to create a catalog file in `makesourcedb` format (see Section 7.3) from the information available in the GSM. As such, the `gsm.py` script serves as the interface between the GSM and BBS.

A catalog file created by `gsm.py` contains the VLSS sources that are in the field of view. For every VLSS source, counterparts in the other catalogs are searched and associated depending on criteria described below. Spectral index and higher-order terms are fitted according to equation 6 in Section 7.3.2.

On CEP, there is a GSM database instance and is loaded with all the sources and their reported properties from the VLSS, WENSS and NVSS surveys. The WENSS survey is actually split up into two catalogs according to their frequencies: A main ($\delta \leq 75.8$, $\nu = 325$ MHz) and a polar ($\delta \geq 74.5$, $\nu = 352$ MHz) part. The VLSS and NVSS surveys are taken at 73.8 MHz and 1400 MHz, respectively.

The python wrapper script `gsm.py` can be used to generate a catalog file in `makesourcedb` format and can be run as:

```
> gsm.py outfile RA DEC radius [vlssFluxCutoff [assocTheta]]
```

The input arguments are explained in Table 1.

Parameter	Unit	Description
`outfile`	string	Path to the `makesourcedb` catalog file. If it exists, it will be overwritten.
`RA, DEC`	degrees	Central position of conical search.
`radius`	degrees	Extent of conical search.
`vlssFluxCutoff`	Jy	Minimum flux of VLSS sources in `outfile`. Defaults to 4 Jy.
`assocTheta`	degrees	Search radius centred on VLSS source for which counterparts are searched. Defaults to 0.00278 degrees (10 arcsec, taking into account the VLSS resolution of 80 arcsec).

Table 1: The parameters and criteria that are used for creating the initial Global Sky Model.

The `gsm.py` script calls the function `expected_fluxes_in_fov()` in `gsmutils.py` that does the actual work. It makes a connection to the GSM database and selects all the VLSS sources that

[38]This section was contributed by Bart Scheers.

fulfill the criteria. The area around every found VLSS source (out to radius `assocTheta`) is searched for candidate counterparts in the other surveys. The dimensionless distance association parameter, $\rho_{i,\star}$, is used to quantify the association of VLSS source−candidate counterpart further. It weights the positional differences by the postion errors of the pair and follows a Rayleigh distribution (De Ruiter et al., 1977). A value of 3.717 corresponds to an acceptance of missing 0.1% genuine source associations (Scheers, 2011). The dimensionless radius is not an input argument to `gsm.py`, but it is to the above mentioned function. For completeness, we give its definition below:

$$\rho_{i,\star} \equiv \sqrt{\frac{(\alpha_i \cos\delta_i - \alpha^\star \cos\delta^\star)^2}{\sigma^2_{\alpha_i} + \sigma^2_{\alpha^\star}} + \frac{(\delta_i - \delta^\star)^2}{\sigma^2_{\delta_i} + \sigma^2_{\delta^\star}}}, \qquad (12)$$

Here the sub- and superscripts $\star$ refers to the VLSS source and i to its *candidate* counterpart in one of the other surveys.

After being associated (or not), the corresponding fluxes and frequencies are used to fit the spectral-index and higher-order terms according to equation 6 in Section 7.3. Therefore, we use the python `numpy.poly1d()` functions. If no counterparts were found a default spectral index of −0.7 is assumed.

Another optional argument when calling the function `expected_fluxes_in_fov()` in `gsmutils.py` is the boolean `storespectraplots`. When true and not performance driven, this will plot all the spectra of the sources in the catalog file, named by their VLSS name.

Special cases

There might be cases that a VLSS source has more than one WENSS counterpart. This might occur when the mutiple subcomponents of a multicomponent WENSS source are associated to the VLSS source. WENSS sources that are flagged as a subcomponent (`'C'`) are omitted in the inclusion. Only single component WENSS sources (`'S'`) and multicomponent WENSS sources (`'M'`) are included in the counterpart search.

7.5 Model parameters

When calibrating, we try to estimate parameters in the measurement equation. The values of these model parameters are stored in a so-called “parmdb” (table). Usually, this parmdb is stored inside a measurement set and is called `instrument`. To inspect or create a parmdb, use the command `parmdbm`[39]. To view the contents of a parmdb, use the tool `parmdbplot.py` (section 7.12).

A parmdb can contain two sorts of parameters: normal parameters that are both time and frequency dependent, and *default parameters* that are neither frequency nor time dependent. The default parameters can be used as fallback if a model parameter is not known, but they can also be used in some schemes for transferring solutions, see section 7.14.

Before even starting the actual BBS, there need to be values in the parmdb, because they are used as starting values by BBS. For gains (`Gain` and `DirectionalGain`), the default starting value of 0 is not adequate. For this reason, the `calibrate-stand-alone` script implicitly creates a default parmdb that contains initial values of 1 for these parameters.

In most circumstances, these defaults are fine. However, there are cases (e.g. when simulating differential Faraday rotation) in which you will want to provide BBS with your own specific default values. For example:

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

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

This creates a new parmdb, which is called `myparmdb` in this case. Differential Faraday rotation at each station is initialized to -42.0 rad m^{-2}. If you run BBS using the `calibrate-stand-alone` script, use the `--parmdb` option to use this parmdb.

This way you can provide default values for parameters that you have not (yet) estimated. These same values are used as an initial guess when you do try to estimate them. Please note that if you create your own parmdb, you will almost always want to include the default `adddef` commands listed below to set the correct defaults for `Gain`, `DirectionalGain`. Otherwise, estimating these parameters will not work correctly. The default parameters are automatically created if you use `calibrate-stand-alo-ne` with the option `-f` or `--replace-parmdb`

```
adddef Gain:0:0:Ampl  values=1.0
adddef Gain:1:1:Ampl  values=1.0
adddef Gain:0:0:Real  values=1.0
adddef Gain:1:1:Real  values=1.0
adddef DirectionalGain:0:0:Ampl  values=1.0
adddef DirectionalGain:1:1:Ampl  values=1.0
adddef DirectionalGain:0:0:Real  values=1.0
adddef DirectionalGain:1:1:Real  values=1.0
```

7.6 Model

As already mentioned, BBS consists of two parts: a part to solve equations and a part to simulation of visibilities given a sky model. This section is about the latter. BBS uses the measurement equation, which is an equation that describes all effects that happen to the signal that was sent by the sky, before they are captured in your data. All these effects are Jones matrices: 2×2 matrices that work on the two components (the two polarizations) of your data. An important thing to note is that these Jones matrices do not commute: the order in which they are matters.

The most commonly used effects that BBS can handle are given in Table 2, in the order that they are applied (from sky to antenna). The direction dependent effects are different for each *patch* you specify in your source model.

The two most commonly used effects, Gain and DirectionalGain, have only one option: `Phasors`. When set to `True` (or `T`), the gains are expressed like $A \cdot e^{i\phi}$, otherwise they are specified in the form $a + b \cdot i$. While mathematically equivalent, this does make a difference because the solver in BBS solves for real variables. When you are solving for phases or amplitudes only, it is necessary to specify `Phasors = True`. Specify this like

```
Model.Gain.Phasors = T
```

In older versions, the option to specify the gains as amplitude/phase could only be defined for the whole model expression as a whole, like `Model.Phasors.Enable=True`. This is still supported: it sets the phasors option for all (directional and non-directional) gains in the model.

For configuration of the beam model that is used, see Section 7.6.1.

Effect	Description	
`ScalarPhase`	A phase that is equal for both dipoles	direction dependent
`Rotation`	Faraday Rotation without frequency dependency	direction dependent
`FaradayRotation`	Faraday Rotation	direction dependent
`DirectionalTEC`	TEC (ionosphere), see 7.8.5	direction dependent
`Beam`	The LOFAR beam model. See 7.6.1	direction dependent
`DirectionalGain`	Directional gain	direction dependent
`CommonScalarPhase`	Scalar Phase	direction independent
`CommonRotation`	Rotation	direction independent
`TEC`	TEC (ionosphere), see 7.8.5	direction independent
`Gain`	Gain	direction independent
`Clock`	Clock	direction independent

Table 2: Effects that BBS handles. The first half are direction dependent effects (DDEs), which means that the effect is different for each patch. The bottom effects are direction independent effects (DIEs).

The only other model parameter that has a configuration option is `Clock`: you can specify whether the two dipoles in an antenna should have the same clock or a separate one. The option is `Split`. By default this is false, so the dipoles are assumed to share a clock. Specify the following to use a separate clock for each dipole:

```
Model.Clock.Split = T
```

7.6.1 Beam model

The beam model tries to emulate all kinds of distortions to the signal that are caused by the beam of the station. These effects are split into two parts: the *element beam*, which is the response of a single dipole, and the *array factor*, which emulates the effect of combining the signal of the many dipoles in a station. In HBA, the array factor model also models the effect of the analog tile beam former.

To have a look at different elements of the beam, you can specifically use only the element beam or only the array factor (if you don't know the details, you need both the element beam and the array factor, which is the default). The options are:

```
Model.Beam.Mode = ELEMENT       # only element beam
Model.Beam.Mode = ARRAY_FACTOR  # only array factor
Model.Beam.Mode = DEFAULT       # both element beam and array factor (default)
```

The tile beam former in the HBA tiles forms a beam for a certain reference frequency. When modeling this beam, the beam model should of course do this for the same frequency. Usually, this works automatically: the reference frequency is stored in the measurement set. Things are different when you compress a number of subbands as channels into one measurement set (usually done with DPPP). Then each 'channel' was beamformed at a different reference frequency. In this case, the reference frequency is only right for one of the 'channels'. To handle this case well, there is an option that tells the beam model to use the channel frequency (which is usually the center frequency of the compressed subband). This option is:

```
Step.Solve.Model.Beam.UseChannelFreq = T
```

Note that the beam model is a direction dependent effect like any other in BBS. That means that over a patch, the beam is assumed to be constant (it is evaluated at the centroid of the patch). This may change in the future.

7.7 Solver

BBS performs parameter estimation on the measurement equation to find parameters that best match the observed visibilities. To improve signal to noise, one can assume that the parameters are constant for a number of time samples or a number of frequencies. In this way, there are more measurements available to estimate the same parameter. To specify these, use `Step.<name>.Solve.CellSize.Time` and `Step.<name>.Solve.CellSize.Freq`. The unit of `CellSize.Time` is number of time slots, so `CellSize.Time=1` corresponds to the correlator integration time. If `CellSize.Time=0`, one solution is calculated for the entire scan.

The underlying solver is a Levenberg-Marquardt solver. Several parameters exist to this solver, however the defaults should be fine.

7.8 Example reductions

The sequence of operations that BBS will perform on the data are defined in a configuration or *parset* file[40]. The BBS documentation on the LOFAR wiki documents all the options (see the LOFAR wiki[41]), and it is highly recommended that you obtain a hardcopy of this for future reference.

A BBS parset file consists of two sections: The *Strategy* section, which defines the operations (or *steps*) to be carried out, and the *Step* section, which defines the details of each step. The following sections describe a few typical reductions along with the corresponding parset.

The parsets shown in the following sections are intentionally verbose. Often, default settings have been included for clarity. For example, the default input column is DATA. The line
`Strategy.InputColumn = DATA` is therefore redundant and can be left out.

7.8.1 Simulation

Given a source catalog and (optionally) a table of model parameters, BBS can be used to compute simulated *uv*-data (without noise). Simulated data can sometimes be useful as a debugging aid. Imaging simulated data can provide an impression of what you would expect to see with an ideal telescope under ideal conditions and without noise. Comparing observed data to simulated data can provide useful clues, although in practice this is limited to cases where the signal to noise ratio of the observed data is high.

Simulated data produced by BBS (or any other software package) can also be used as model data during calibration using the same parset syntax as described in Section 7.11.

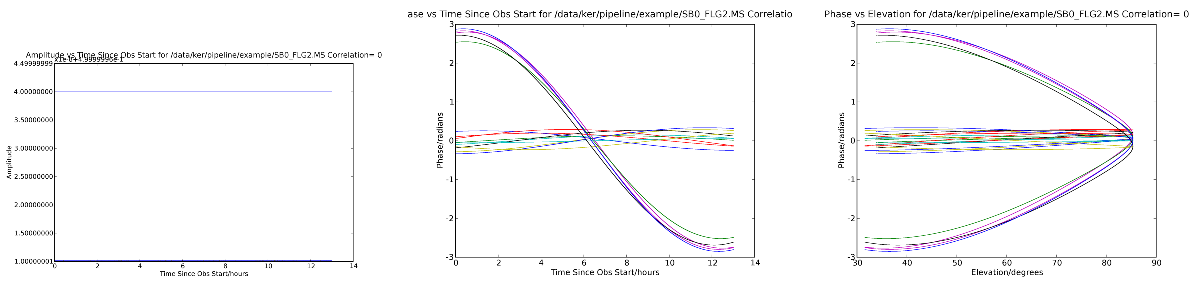
An example parset file[42] to simulate *uv*-data for all the sources in the source catalog is shown below.

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

[40] Examples of these files can be found in `/globaldata/COOKBOOK/Parset`.
[41] http://www.lofar.org/operations/doku.php?id=engineering:software:tools:bbs
[42] This example can be found in `/globaldata/COOKBOOK/Parset/simulation.parset`.

```
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 = 100

# 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 = []

# MS output column to write simulated visibilities to
Step.predict.Output.Column = MODEL_DATA
```

The example output of these simulations is shown in Figure 14.



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

Figure 14: Example outputs for a simulation of the SB0.MS 3C196 observations.

7.8.2 Gain calibration (direction independent)

The following parset[43] describes a direction independent gain calibration. The meaning of each parameter is the same as in Section 7.8.1 unless otherwise stated. Enabling / disabling of model components can be either done by a short-hand `T` and `F` which is equivalent to explicit `True` and `False`.

The output of the calibration on a single subband centered on 3C196 is shown in Figure 15.

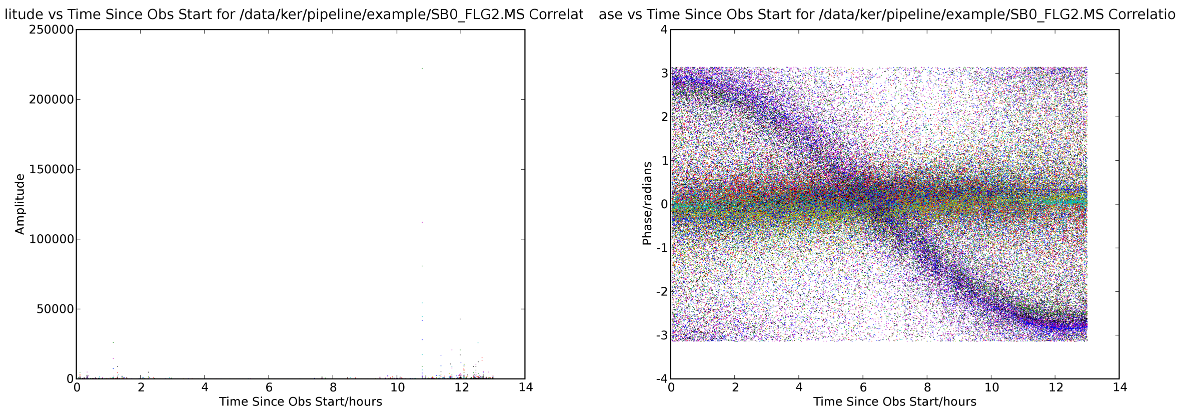
[43]This example can be found in `/globaldata/COOKBOOK/Parset/uv-plane-cal.parset`.

```
##############################################################
# 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 should be at least as big as the Cellsize.Time (or
# zero) and preferably and integer multiple of CellChunkSize * CellSize.Time
# (see below in the SOLVE step parameters). All if zero.
Strategy.ChunkSize = 100

# 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.
# 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 from the name tags in the sky model file
Step.solve.Model.Sources = [3C196]

# Cache unaffected terms between iterations. Speeds up the computation, but
# increases memory usage.
Step.solve.Model.Cache.Enable = T

# Individual terms of the measurement equation can be enabled or disabled.
# Most are turned off here.
Step.solve.Model.Gain.Enable = T
Step.solve.Model.DirectionalGain.Enable = F

# When only estimating the diagonal elements of the Gain matrix (as in this
# example), or when the model of the target field contains multiple sources,
# the beam model should be enabled.
Step.solve.Model.Beam.Enable = T

# 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 (subset of those selected by Step.solve.Solve.Parms).
Step.solve.Solve.ExclParms = []

# Defines the calibration groups. For example: If you have have 6 subbands
# and  want to solve them in 2 groups you put 3,3. 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.).
#
# CellSize.Freq is specified in number of channels, CellSize.Time in number
# of time slots. A value of zero means all, i.e. using the values given below
# a single solution will be computed per time slot using data from all
# channels.
#
# 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 time cells.
# NB. It is recommended that Strategy.ChunkSize is an integer multiple of
# CellChunkSize * CellSize.Time (or zero). Generally best to set this in the
# range 10 - 25.
Step.solve.Solve.CellChunkSize = 10

# 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 = 50
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 = [3C196]
Step.correct.Model.Gain.Enable  = T
Step.correct.Model.Beam.Enable = T
Step.correct.Output.Column = CORRECTED_DATA
```

The CORRECT step performs a correction of the *uv*-data for a particular direction. This can be done for exactly *one* source from the skymodel. If `Model.Sources` contains multiple sources, BBS will throw an exception, because it cannot correct for more than one direction at a time.

BBS also accepts an empty source list in the CORRECT step. In that case it will correct for the phase center direction of the MS. This does then, of course, not include direction dependent effects such as `DirectionalGain`, `DirectionalTEC`, et cetera, which are inherently bound to a patch name and therefore can only be corrected for if a patch name is specified. This implicit behaviour must be kept in mind when correcting your data.

Note that a CORRECT step cannot be “undone”. If a CORRECTED_DATA column is used for further calibration later on, one has to be aware of the consequences. For example, if in the first correct the beam was enabled, this prevents proper use of the beam in the following steps[44].



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

Figure 15: 3C196 corrected amplitudes and phases, after calibration. The need for further flagging is clear.

7.8.3 Gain calibration (direction independent, phase or amplitude only)

The parset file to perform a (direction independent) phase or amplitude calibration differs from the parset file for regular gain calibration (see Section 7.8.2) on three important points.

Phase calibration will be used as an example. First, the complex gain parameters must be decomposed into their amplitude, phase components instead of their real, imaginary components (the default). This can be achieved by setting `Model.Phasors.Enable = T`. Second, only the phase component should be estimated. This can be achieved by setting `Solve.Parms = ["Gain:0:0:Phase:*", "Gain:1:1:Phase:*"]`. Third, amplitude differences should be ignored when computing the difference between model visibilities and observed visibilities. This can be achieved by setting `Solve.Mode = PHASE`.

Amplitude calibration is specified similarly, except that in this case only the amplitude component should be estimated (`Solve.Parms = ["Gain:0:0:Ampl:*", "Gain:1:1:Ampl:*"]`) and phase differences should be ignored (`Solve.Mode = AMPLITUDE`).

The value of `Solve.Mode` controls the way in which the difference between the (complex) model visibilities and (complex) observed visibilities is computed. The default is to take both phase and amplitude differences into account (`Solve.Mode = COMPLEX`). Alternatively, `Solve.Mode = PHASE` ignores amplitude differences, while `Solve.Mode = AMPLITUDE` ignores phase differences.

For direction independent gain, the measurement equation can be separated into an equation of amplitudes and an equation of phases. Therefore, it makes sense to minimize phase differences when solving for phase only (and similarly for amplitude). This is equivalent to the phase (or amplitude) calibration that is available in other calibration packages (e.g. CASA).

For direction dependent gain, however, the model visibilities are the vector sum of multiple components (sources), each multiplied by its own gain term. Changing the phase of the gain of a single component changes both the phase *and the amplitude* of the sum. Minimizing either phase or amplitude differences is *incorrect* in this case, because amplitude and phase are interdependent.

[44]This is important when doing a self calibration. In that case only the CORRECTED_DATA column should be used for imaging, while a next calibration step should go back to take the DATA column as input to refine the calibration.

In summary, unless you know what you are doing, be careful setting Solve.Mode when using anything other than direction independent gain.

The parset shown below is an example parset for phase calibration[45].

```
############################################################
# uv-plane-cal-phaseonly.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 should be at least as big as the Cellsize.Time (or
# zero) and preferably and integer multiple of CellChunkSize * CellSize.Time
# (see below in the SOLVE step parameters). All if zero.
Strategy.ChunkSize = 100

# 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.
# 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 from the name tags in the sky model file
Step.solve.Model.Sources = [3C196]

# Cache unaffected terms between iterations. Speeds up the computation, but
# increases memory usage.
Step.solve.Model.Cache.Enable = T

# Individual terms of the measurement equation can be enabled or disabled.
# Most are turned off here.
Step.solve.Model.Phasors.Enable = T
Step.solve.Model.Gain.Enable = T

# Minimize phase differences. Use AMPLITUDE for amplitude calibration (and
# don't forget to change Step.solve.Solve.Parms below).
Step.solve.Solve.Mode = PHASE

# Estimate the phase of the gains for the XX (0:0) and YY (1:1) polarisations.
# Use ["Gain:0:0:Ampl:*", "Gain:1:1:Ampl:*"] for amplitude calibration (and
```

[45]This parset can be found in /globaldata/COOKBOOK/Parset/uv-plane-cal-phaseonly.parset.

```
# don't forget to change Step.solve.Solve.Mode above).
Step.solve.Solve.Parms = ["Gain:0:0:Phase:*", "Gain:1:1:Phase:*"]

# Parameters to exclude (subset of those selected by Step.solve.Solve.Parms).
Step.solve.Solve.ExclParms = []

# Defines the calibration groups. For example: If you have have 6 subbands
# and want to solve them in 2 groups you put 3,3. 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.).
#
# CellSize.Freq is specified in number of channels, CellSize.Time in number
# of time slots. A value of zero means all, i.e. using the values given below
# a single solution will be computed per time slot using data from all
# channels.
#
# 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 (or zero). Generally best to set this in the
# range 10 - 25.
Step.solve.Solve.CellChunkSize = 10

# 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 = 50
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 for the direction of <source>

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

7.8.4 Gain calibration (direction dependent) with source subtraction

This section has been adapted from a document written by Annalisa Bonafede[46]. In the following, we report the parset[47] file and skymodel used for the subtraction of Cas A and Cyg A from the observation of the radio source 3C380.

The parset includes the following steps:

- Solve for the gain in the direction of each source in the source catalog.
- Subtract the sources CygA.E, CygA.W, and CasA, each with their own individual gain.
- Correct the data for the gain in the direction of 3C380 (the target).

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

Strategy.ChunkSize = 100
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 = 50
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 source catalog used for calibration is reported below[48]:

```
############################################################
# 3C380-bbs.skymodel
# (Name, Type, Ra, Dec, I, ReferenceFrequency, SpectralIndex) = format

CygA.E, POINT, 19:59:29.99, +40.43.57.53, 4421, 73.8e6, [-0.7]
CygA.W, POINT, 19:59:23.23, +40.44.23.03, 2998, 73.8e6, [-0.7]
CasA, POINT, 23:23:24.0, +58.48.54.0, 20000
```

[46]a.bonafede[at]jacobs-university[dot]de

[47]This example can be found in /globaldata/COOKBOOK/Parset/image-plane-cal.parset.

[48]This source catalog is available in /globaldata/COOKBOOK/Models.

```
3C380, POINT, 18:29:31.8, +48.44.46.0, 1, 178.e6, [-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 Figure 16, compared to the map obtained without subtraction.

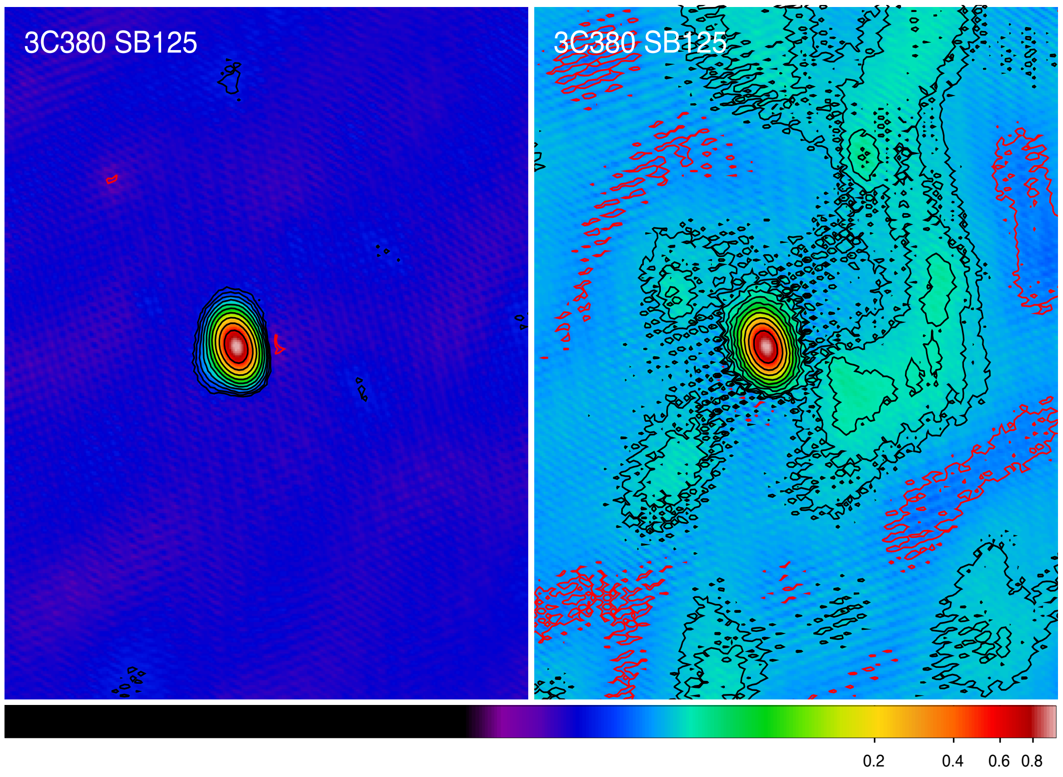


Figure 16: The radio source 3C380 imaged by LOFAR at 135 MHz (observation ID L2010_08567). *Left panel*: Self calibration has been applied with directional-gain correction and subtraction of Cyg A and Cas A. *Right panel*: Self calibration has been applied without correction. The lowest contour level is at 3 mJy beam^{-1}; subsequent contour levels are spaced by factors of $\sqrt{2}$. The resolution is 83.29" × 59.42". The negative contour is in red.

7.8.5 Differential TEC

The electrons in the ionosphere introduce a frequency dependent phase shift in the radio waves passing through the atmosphere. The strength of this effect depends on the 'total electron content' (TEC) along the the line between the receiving station and the source (the *line of sight*). Determining the (absolute) TEC value of patches in the atmosphere is extremely difficult, and is usually based on measuring the frequency dependent reception time of GPS signals and subsequent application of an ionospheric model.

The determination of the *differential TEC*, which is the TEC value difference between two stations on a baseline, is easier. This does not allow you to map an ionospheric phase screen, but it does provide a rough estimate of the behaviour of the ionosphere (which can be used in later to be available ionospheric models for LOFAR).

Differential TEC can be used to describe the frequency dependence of the phase solutions. Since the effect of the ionosphere is more severe at low frequencies, this is of advantage to LBA observations. Estimating differential TEC allows for more accurate phase solution, and, when using global parameter estimation (see Section 7.10), more subbands can be grouped together.

To include differential TEC in the model you must include the following keys in the parset:

```
Step.<name>.Model.TEC.Enable = T
Step.<name>.Solve.Parms = [TEC:*]
```

Warning: to fit TEC or directional TEC the effect of clocks needs to be sorted out. So either in a previous calibration or in the same calibration you need to calibrate for `Clock` as well.

To make use of the estimated differential TEC values, subsequent steps must also include differential TEC in the model (using `Step.<name>.Model.TEC.Enable = T`). The script `Solution_Plotter.py`[49] can be used to plot differential TEC solutions per baseline, when run on an MS containing TEC solutions.

7.9 Tweaking BBS to run faster

BBS can take a long time to calibrate your huge dataset. Luckily, there are some ways to tune it. You have to know a bit about how BBS works to do this.

BBS views the data as a grid of time slots times vs channels (see Figure 17). A solution cell is defined as the number of timeslots times the number of channels on which a constant parameter solution is computed. For example, if you specify `CellSize.Freq = 1`, a different solution is computed for every channel independently. In Figure 17, the solution cells are outlined with blue lines.

Because the evaluation of the model is computationally expensive, and a lot of intermediate results can be reused, the model is evaluated for a number of cells simultaneously. There is a trade-off here: if one cell would converge to a solution very slowly, the model is evaluated for all the cells in the cell chunk, even the ones that have converged. The number of cells in a cell chunk is specified by `CellChunkSize`, which specifies the *number of solution cells in the time direction*. Dependent on the amount of frequency cells, a value of `CellChunkSize = 10 -- 25` could be good.

First, BBS loads a lot of timeslots into memory. The amount of timeslots is specified by `ChunkSize`. If you specify `ChunkSize = 0` then the whole MS will be read into memory, which will obviously not work if your data is 80 Gb large. Depending on the number of baselines and channels, usually 100 is a good choice. Monitor the amount of memory that is used (for example by running `top`) to see if you're good. The `ChunkSize` should be an integer multiple of `CellSize.Time` $\times$ `CellChunkSize`, so that in every chunk the same number of cell chunks are handled and all cell chunks have the same size.

Multithreading

BBS can do a bit of multithreading. Be warned in advance that if you use *N* threads, BBS will most likely not run *N* times faster. Again, a bit of tweaking may be necessary.

The multithreading is done on solve part, not on the model building part. So it works only on problems that are 'solve-dominated'. The multithreading is performed over the solution cells. For this to give any speedup, there need to be at least as many solution cells as the number of threads. You usually get a decent speed-up if the number of solution cells is about 5–6 times the number of threads. So if you

[49] Available from the LOFAR-Contributions GitHub repository at: https://github.com/lofar-astron/LOFAR-Contributions

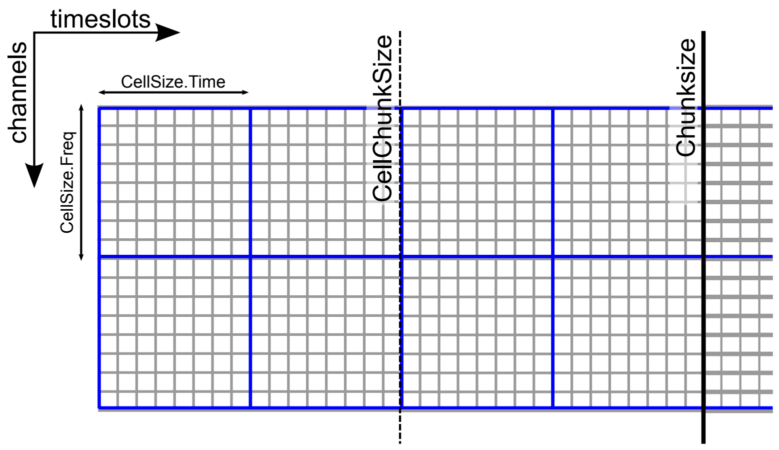


Figure 17: The different solve domains and chunk parameters for BBS. In this example, `CellSize.Time=8`, `CellSize.Freq=8`, `ChunkSize=32`, `CellChunkSize=2` (do not use these values yourself, they are probably not good for actual use).

solve over all frequencies (the number of frequency cells is 1), `CellChunkSize = 5` × `nThreads` should give you some speedup.

You can instruct `calibrate-stand-alone`[50] to run with multithreading with the parameter `-t` or `--numthreads`:

```
calibrate-stand-alone --numthreads 8 <MS> <parset> <source catalog>
```

7.10 Global parameter estimation

BBS was designed to run on a compute cluster, calibrating across multiple subbands which reside on separate compute nodes. BBS consists of three separate executables: `bbs-controller`, `bbs-reducer`, and `bbs-shared-estimator`. The `bbs-controller` process monitors and controls a set of `bbs-reducer` processes, and possibly one or more `bbs-shared-estimator` processes. Each *subband* is processed by a separate `bbs-reducer` process. When working with `calibrate-stand-alone`, the script actually launches one `bbs-reducer` for you. To setup a calibration across subbands, the script `calibrate` can be used, which sets up the appropriate processes on different nodes.

Each `bbs-reducer` process computes a set of equations and sends this to the `bbs-shared-estimator` process assigned to the group it is part of. The `bbs-shared-estimator` process merges the set of equations with the sets of equations it receives from the other `bbs-reducer` processes in the group. Once it has received a set of equations from all the `bbs-reducer` in its group, the `bbs-shared-estimator` process computes a new estimate of the model parameters. This is sent back to all `bbs-reducer` processes in the group and the whole process repeats itself until convergence is reached.

7.10.1 Setting up your environment

Before using the distributed version of BBS, you will have to set up a personal database. In principle, this needs to be done only once. You only have to recreate your database when the BBS SQL code has changed, for example to support new functionality. Such changes are kept to a minimum.

Also, on each `lce` or `locus` node that you are going to use you need to create a working directory. Make sure you use the same path name on all the nodes . This has to be done only once.

[50]Currently, it is not possible to combine multithreading with a global solve or with the `calibrate` script.

The distributed version of BBS requires a file that describes the compute cluster (an example of such a file is `cep1.clusterdesc`[51]) and a configuration or *parset*[52] file that describes the reduction.

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

```
> makevds cep1.clusterdesc <directory>/<MS>
```

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

```
> 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 required even if we want to calibrate a single subband only, although normally you would calibrate a single subband using the stand-alone version of BBS (see Section 7.2).

7.10.2 Usage

To calibrate an observation with the distributed version of BBS *on the Groningen cluster*, execute the `calibrate` script on the command line:

```
> calibrate -f --key <key> --cluster-desc ~/imaging.clusterdesc --db ldb001 \
--db-user postgres <gds file> <parset> <source catalog> <working directory>
```

which is a single command, spread over multiple lines, as indicated by a backslash. The important arguments provided to the `calibrate` script in this example are:

- `<key>`, which is a single word that identifies the BBS run. If you want to start a BBS run *while another run is still active*, make sure that the runs have *different* keys (using this option).
- `<gds file>`, which contains the locations of all the MS (subbands) that constitute the observation. It should be given here by its full path, for example /data/scratch/<username>/<global gds file>.
- `<parset>`, which defines the reduction (see Section 7.8).
- `<source catalog>`, which defines a list of sources that can be used for calibration (see Section 7.3).
- `<working directory>`, which is the working directory where BBS processes will be run and where the logs will be written (usually `/data/scratch/<user name>`). It should be created on each compute node that you intend to use. You can use the `cexec` command for this (see Section 1.2).
- The `-f` option, which overwrites stale information from previous runs.

[51]You can copy this file from `/globaldata/COOKBOOK/Files`.

[52]Examples can be found in `/globaldata/COOKBOOK/Parset`.

You can run the calibrate script without arguments for a description of all the options and the mandatory arguments. You can also use the -v option to get a more verbose output. Note that the arguments of calibrate are very much like[53] those of calibrate-stand-alone.

The calibrate script does not show progress information, which makes it difficult to estimate how long a BBS run will take to complete. One way around this is to log on to one of the compute nodes where BBS is processing, change to your local working directory, and monitor one of the bbs-reducer log files. These log files are named <key>_kernel_<pid>.log. The default key is default, and therefore you will often see log files named e.g. default_kernel_12345.log. The following command will print the number of times a chunk of visibility data was read from disk:

```
> grep "nextchunk" default_kernel_12345.log | wc -l
```

You can compute the total number of chunks by dividing the total number of time stamps in the observation by the chunk size specified in the parset (Strategy.ChunkSize). Of course, if you make the chunk size large, it may take BBS a long time to process a single chunk and this way of gauging progress is not so useful. Generally, it is not advisable to use a chunk size larger than several hundreds of time samples. This will waste memory. A chunk size smaller than several tens of time stamps is also not advisable, because it leads to inefficient disk access patterns.

7.10.3 Defining a global solve

Solving across multiple subbands can be useful if there is not enough signal in a single subband to achieve reasonable calibration solutions, i.e. the phase and amplitude solutions look more or less random. Basically, there should be enough source flux in the field for calibration. Of course, one first has to verify that the sky model used for calibration is accurate enough, because using an inaccurate sky model can also result in bad calibration solutions.

To enable global parameter estimation, include the following keys in your parset. Note that the value of the Step.<name>.Solve.CalibrationGroups key is just an example. This key is described in more detail below.

```
Strategy.UseSolver = T
Step.<name>.Solve.CalibrationGroups = [3,5]
```

The Step.<name>.Solve.CalibrationGroups key specifies the partition of the set of all available subbands into separate *calibration groups*. Each value in the list specifies the *number* of subbands that belong to the same calibration group. Subbands are ordered from the lowest to the highest starting frequency. The sum of the values in the list *must* equal the total number of subbands in the gds file. By default, the CalibrationGroups key is set to the empty list. This indicates that there are no interdependencies and therefore each subband can use its own solver. Global parameter estimation will *not* be used in this case (even if Strategy.UseSolver = T).

When using global parameter estimation, it is important to realize that drifting station clocks and the ionosphere cause frequency dependent phase changes. Additionally, due to the global bandpass, the effective sensitivity of the telescope is a function of frequency. Therefore, at this moment, it is not very useful to perform global parameter estimation using more than about 1–2 MHz of bandwidth (5–10 consecutive subbands).

A few examples of using global parameter estimation with 10 bands would be:

[53]Actually, both scripts use different terminology, sky-db in the one is called sourcedb in the other. This will be fixed.

- Estimate parameters using all bands together:

  ```
  Step.<name>.Solve.CalibrationGroups = [10]
  ```

- Estimate parameters for the first 5 bands together, and separately for the last 5 bands together:

  ```
  Step.<name>.Solve.CalibrationGroups = [5,5]
  ```

- Do not use global parameter estimation (even if `Strategy.UseSolver = T`):

  ```
  Step.<name>.Solve.CalibrationGroups = []
  ```

- Estimate parameters for band 0 separately, bands 1–3 together, bands 4–5 together, and bands 6–9 together (note that 1+3+2+4=10, the total number of subbands in our example).

  ```
  Step.<name>.Solve.CalibrationGroups = [1,3,2,4]
  ```

7.11 Pre-computed visibilities

Diffuse, extended sources can only be approximately represented by a collection of point sources. Exporting clean-component (CC) models from catalogues (e.g. VLSS, WENSS) or first iteration major cycle selfcal images tend to contain many CCs, ranging up to 50,000. By using `casapy2bbs.py` these can be imported into a BBS catalog file, but processing these can take long and memory requirements might not even allow their usage at all.[54]

An alternative lies in (fast) Fourier transforming model images directly into *uv*-data columns. These can then be used in BBS as model data. The import can be done with a tool called `addUV2MS`.

The first argument is the MS to which the *uv*-data is to be added. The second argument is a CASA image (extension .model). The filename of the image is stripped of its leading path and file extension. This is then the column name it is identified by in the parset, and can be seen in the MS. For example:

```
> addUV2MS -w 512 L24380SB030uv.MS.dppp.dppp $HOME/Images/3C196_5SBs.model
```

This will create a column of name `3C196_5SBs`, containing the *uv*-data FFT'ed from this model image, using 512 w-projection planes. You can run `addUV2MS` multiple times with different images, or also with several images as additional command arguments, to create more than one *uv*-data column. While this works in principle with normal images, it is advisable to use clean component model images generated by CASA. A few notes of caution though:

- The image must have the same phase center as the MS, because the internal CASA-routine which is used, does not do any phase shifting.
- For wide field images you should set an appropriate value for the number of w-planes used in the w-projection term (default=128).
- Direction dependent effects cannot be handled for "large" images. This means the image has to be small on the scale over which the direction dependent effects change.

[54]On CEP3, catalog files with up to ca. 10,000 CCs can be processed until the nodes run out of memory.

- `addUV2MS` temporarily overwrites the frequency in the input images to match that in the MS. It restores the original frequency, but only one (multi-frequency channel images aren't supported). Aborting the run may leave you then with a model image having an incorrect frequency entry.
- Successive runs with the same input model image will overwrite the data in the respective column.

The column name generated by `addUV2MS` can be used in the BBS parset as:

```
Step.<name>.Model.Sources = [@columnname]
```

You can use `casabrowser` to find out which data columns are available in a given MS. Be careful about dots in the filename, and verify that your model refers to the name of the created column. Running `addUV2MS -h` will give you more information about its usage.

You can use `cexecms` to add model *uv*-data to more than one MS, for example:

```
> cexecms "addUV2MS -w 512 <FN> $HOME/Images/3C1965_SBs.model" \
"/data/scratch/pipeline/L2011_08175/*"
```

7.12 Inspecting the solutions

You can inspect the solutions using the python script `parmdbplot.py`. To start `parmdbplot.py` from the command line, you should first initialize the LofIm environment (see Section 1.3). Then you can type, for example:

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

The first thing you should see after starting the script should be the main window (see Figure 18). Here you can select a set of parameters of the same type to be plotted together in a single plot. Note that some features will not be available if you select multiple parameters. Parmdbplot is able to properly handle the following solution types: Gain, DirectionalGain, CommonRotationAngle, RotationAngle, CommonScalarPhase, ScalarPhase, Clock, TEC, RM. The last three (Clock, TEC and RM) are properly converted into a phase – they are stored differently in the parmdb internally.

The "Use resolution" option is best left *unchecked*. If it is checked, the plotter tries to find a resolution that will yield a 100×100 grid in frequency $\times$ time. Usually, you just want to use the sampling intervals that are present in the parmdb. In that case, leave the box unchecked.

Once you click the "Plot" button, a window similar to the Figure 19 should pop up. We discuss the controls on the top of the window from left to right. The first is a drop down box that allows you to select 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).

The "Legend" checkbox allows you turn the legend on or off (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. The "Unwrap phase" checkbox will turn phase unwrapping on or off. The button "Block y-axis" is useful when stepping over multiple frequency solutions: if checked, the y-scale will remain the same for all plots. "Use points" can be checked if you want points (instead of lines) for amplitude plots.

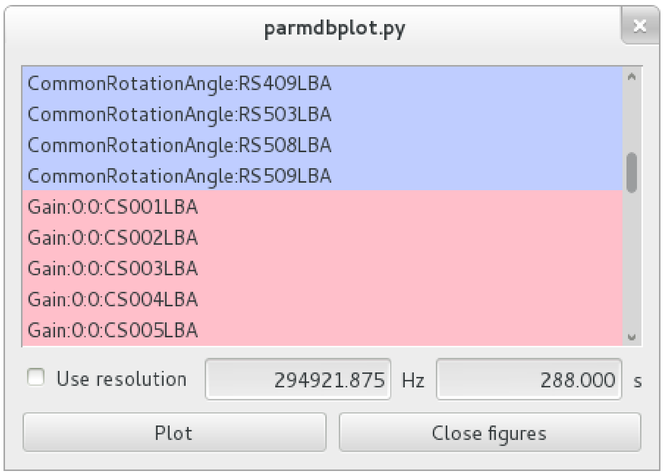


Figure 18: The main window of `parmdbplot`.

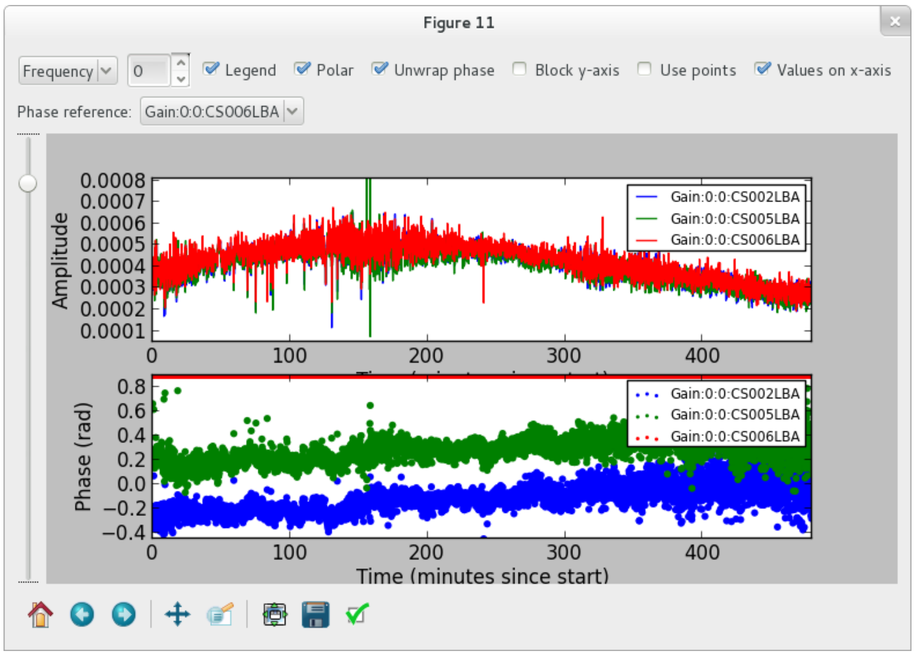


Figure 19: The plot window of `parmdbplot`.

The last checkbox lets you choose the unit for the x-axis. By default, it is the sample number. If you chose, in the main window, to use a time resolution of 2 seconds, then the number 10 on the x-axis means $10 \times 2 = 20$ seconds. If you enable “Values on x-axis”, it will show the number of minutes since the start of the observation.

The “Phase reference” drop down box allows you to select the parameter used as the phase reference for the phase plots (the phase reference is only applied in amplitude/phase mode).

When the phase reference is set, one can use the “phase sum” drop-down menu (see Fig. 20). This menu allows the user to select among all the other parameters related to the same antenna and add the phases to those plotted. All the selected phases are referenced to the ”phase reference” antenna. This procedure is useful if one solves for Phase and Clock (or TEC) and wants to look at the global phase effect. Note that the phase effects are just added together with not specific order. This is only physically correct if the effects commute.

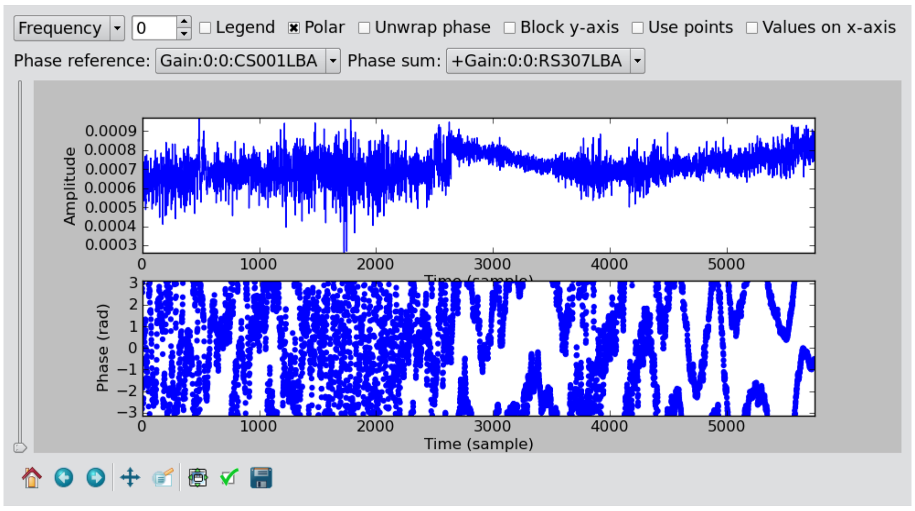


Figure 20: The plot window of parmdbplot with a phase sum

The slider on the left of the plots allows you to make an exponential zoom to the median value of amplitude plots. This can be helpful if you have one outlier in the solution which sets the scale to 1000 when you are actually interested in the details around 0.001.

The controls on the bottom of the window are the default matplotlib controls that allow you to pan, zoom, save the plot, and so on.

For a quick overview of solutions, and to compare lots of solutions at a glance, you can also use the `solplot.py`[55] script by George Heald:

```
>solplot.py -q *.MS/instrument/
```

Running the script with `-h` will produce an overview of possible options.

7.13 The global bandpass

This section describes estimates of the global bandpass for the *LBA* band and *HBA* bands. The bandpass curves were estimated from the BBS amplitude solutions for several observations of a calibrator source (Cygnus A or 3C196).

7.13.1 LBA

The global bandpass has been determined for the LBA band between 10-85 MHz by inspecting the BBS solutions after calibration of a 10-minute observation of Cygnus A. Calibration was performed using a 5-second time interval on data flagged for RFI (with RFIconsole), demixed, and compressed to one channel per sub band. The LBA beam was enabled during the calibration. The bandpass was then derived by calculating the median of the amplitude solutions for each sub band over the 10-minute observation, after iterative flagging of outliers.

Figure 21 shows the amplitudes found by BBS for each sub band, normalized to 1.0 near the peak at ≈ 58 MHz. The time and elevation evolution of the bandpass has also been investigated. In general, the bandpass is approximately constant on average over the elevation range probed by these observations, implying that the effects of the beam have been properly accounted for.

[55]Available at the LOFAR-Contributions GitHub repository: https://github.com/lofar-astron/LOFAR-Contributions

7.13.2 HBA

The global bandpass for the HBA bands was determined in the same basic way as the LBA global bandpass. Three one-hour observations of 3C196 from April 2012 were used (one each for HBA-low, HBA-mid, and HBA-high). No demixing was done. A two-point-source model was used for calibration. The resulting bandpass is shown in Figure 22. Note that several frequency intervals in the HBA-high observation were affected by severe RFI.

7.14 Gain transfer from a calibrator to the target source[56]

In order to calibrate a target field without an *a priori* model, one way forward is to observe a well-known calibrator source, use it to solve for station gains, and apply those to the target field in order to make a first image and begin self-calibrating. There are two tested methods for utilizing calibrator gains in LOFAR observations. Additional methods may become possible later.

- Observe a calibrator source before a target observation, using the same frequency settings, with a short time gap between calibrator and target. This is the same approach as is used in traditional radio telescopes and allows using the full bandwidth in the target observation. However, it is not suggested for long target observations, because the calibrator gains may only remain valid for a relatively short time.
- Observe a calibrator in parallel with a target observation, using the same frequency settings for both beams. This has the disadvantage that half of the bandwidth is lost in the target observation, but the time variation of the station gains will be available.

The recommended approach for dealing with both of these cases is outlined below.

7.14.1 The "traditional" approach

When doing calibration transfer in the normal way, i.e. the calibrator and target are observed at different times, some work needs to be done before the gain solutions of the calibrator can be transferred to the target. This is because the calibrator solutions tell the gain error of the instrument at the time the calibrator was observed, and in principle not of when the target was observed. The implicit assumption of the traditional approach is that the gain solutions are constant in time. The frequencies of the calibrator observation should in principle match those of the target.

The above means that the calibration of the calibrator should lead to only one solution in time. This can be achieved by setting `CellSize.Time` to `0`. After this is done, the validity of this solution should be extended to infinity by using the `export` function of `parmdbm`. Full documentation for that program is available on the LOFAR wiki[57].

In order to achieve time independence you should use these settings in the BBS parset[58]:

```
Strategy.ChunkSize = 0                # Load the entire MS in memory
Step.<name>.Solve.CellSize.Time = 0   # Solution should be constant in time
Step.<name>.Solve.CellChunkSize = 0
```

Export the calibrator solutions so that they can be applied to target field (see the LOFAR wiki for details). For example:

[56] George Heald (`heald[at]astron[dot]nl`) contributed to the writing of this section.
[57] `http://www.lofar.org/operations/doku.php?id=engineering:software:tools:parmdbm`
[58] Note that these settings would be dangerous for a long observation!

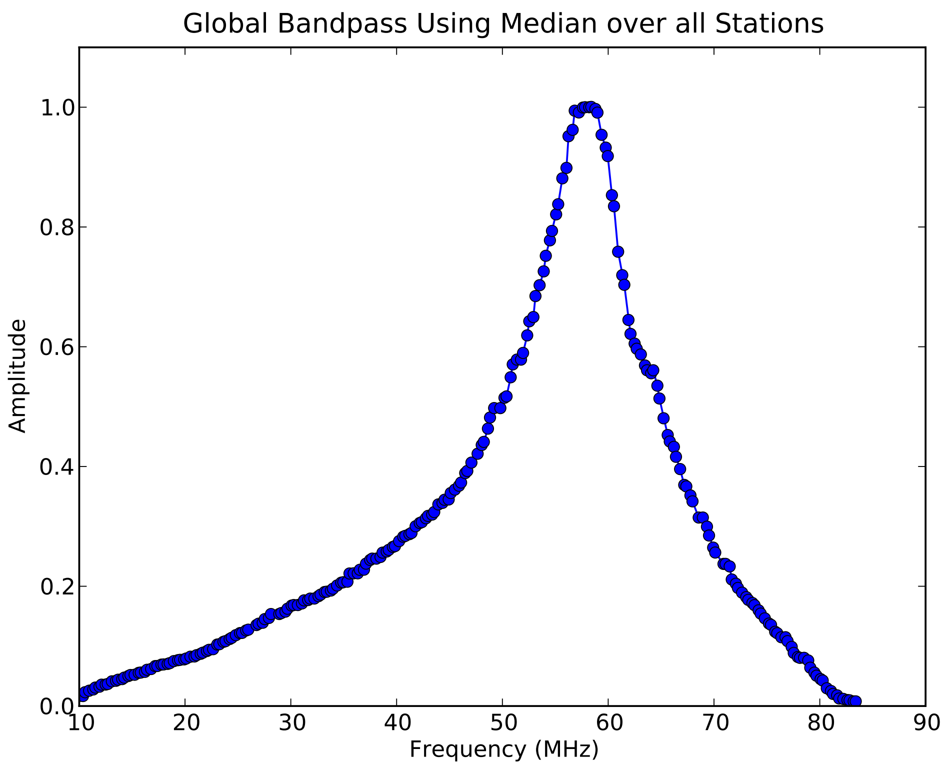
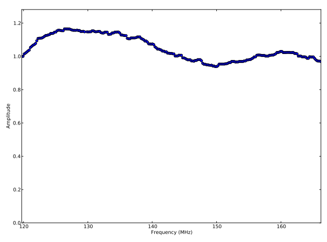
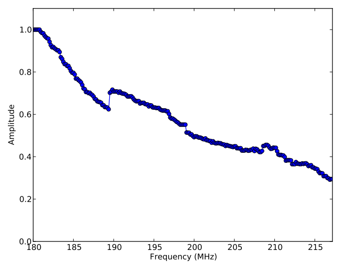


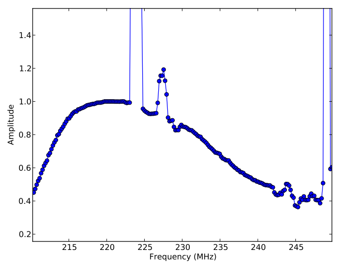
Figure 21: The global bandpass in LBA between 10–85 MHz.



(a) HBA-low



(b) HBA-mid



(c) HBA-high

Figure 22: The global bandpass for the HBA.

```
> parmdbm
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
```

An alternative scheme to make the solutions of the calibrator observation time independent is to have a smaller `CellSize.Time`, and afterwards take the median[59]. This way, time cells where calibration failed do not affect the solutions. To follow this scheme, calibrate the calibrator as you would do normally, for example with `CellSize.Time=5`. To make the solutions time independent, use the tool `parmexportcal.py` (see `parmexportcal --help` or its documentation[60]. For example, if you have calibrated the calibrator and stored the solution in `cal.parmdb`, you can take the median amplitudes as follows:

```
parmexportcal in=cal.parmdb out=cal_timeindependent.parmdb
```

By default, `parmexportcal` takes the median amplitude, and the last phase. This is because the phase always varies very fast, and taking the median does not make sense. One can also chose not to transfer the phase solution of the calibrator at all, by setting `zerophase` to false in `parmexportcal`.

More advanced schemes for processing calibrator solutions before transferring them to the target can be applied using LoSoTo, see section 8. An example could be flagging the calibrator solution, and then averaging it (instead of taking the median).

To apply the gain solutions to target field, one can use BBS. For the `calibrate-stand-alone` script, use the `--parmdb` option. Use a BBS parset which *only* includes a `CORRECT` step. Now you should have a calibrated `CORRECTED_DATA` column which can be imaged.

7.14.2 The LOFAR multi-beam approach

Unlike other radio telescopes, LOFAR has the ability to observe in multiple directions at once. We are currently experimenting with transferring station gains from one field (a calibrator) to a target field. So far it seems to work quite well, with limitations described below. The requirements for this technique are:

- The calibrator and target beams should be observed simultaneously, with the same subband frequency. Future work may change this requirement (we may be able to interpolate between calibrator subbands), but for now the same frequencies must be observed in both fields.
- Any time and/or frequency averaging performed (before these calibration steps) on one field must be done in exactly the same way for the other field.
- The calibrator beam should not be too far from the target beam in angular distance. Little guidance is currently available for the definition of "too far", but it appears that a distance of 10 degrees is fine at $\nu = 150$ MHz, while 40 degrees (at the same frequency) might well be too far. Future experiments should clarify this limitation.

[59]This is the calibration scheme that the LOFAR pipelines follow.
[60]http://www.lofar.org/operations/doku.php?id=engineering:software:tools:parm-export

- In HBA, the distance limit is also driven by the flux density of the calibrator, and the attenuation by the tile beam. For a good solution on the calibrator, ensure that the sensitivity is sufficient to provide at least a signal-to-noise ratio of $2-3$ per visibility. The tile FWHM sizes and station SEFDs are available online.

In order to do the calibration, and transfer the resulting gains to the target field, follow these steps:

1. Calibrate the calibrator using BBS. Ensure that the beam is enabled:

   ```
   Step.<name>.Model.Beam.Enable = T
   ```

   The time and frequency resolution of the solutions can be whatever is needed for the best results.

2. (Optional) Perform some corrections to the calibrator solutions, for example flagging, smoothing or interpolation. This is best done in LoSoTo, see section 8.

3. Apply the gain solutions to the target field. For the `calibrate` script, use the `--instrument-db` option. For the `calibrate-stand-alone` script, use the `--parmdb` option. Use a BBS parset which *only* includes a `CORRECT` step. Again, ensure that the beam is enabled. The source list for the `CORRECT` step should be left empty:

   ```
   Step.<name>.Model.Sources = []
   ```

4. Before proceeding with imaging and self-calibration, it may be advisable to flag and copy the newly created `CORRECTED_DATA` column to a new dataset using DPPP.

7.15 Post-processing

The calibrated data produced by BBS can contain outliers that have to be flagged to produce a decent image. It is recommended to visually inspect the corrected visibilities after calibration. Outliers can be flagged in various ways, for instance using AOFlagger (see Section 4) or DPPP (see Section 5).

The script `CallSolFlag.py`[61] can also be used to flag the calibrated data. Even though the name suggests differently, this script simply flags all the visibilities in the `CORRECTED_DATA` column that exceed the specified flux threshold.

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

7.16 Troubleshooting

- Bugs should be reported using the LOFAR issue tracker[62].

- If BBS crashes for any reason, be sure to kill all BBS processes (`bbs-controller`, `bbs-reducer`, and `bbs-shared-estimator`) on all of the nodes you were working on before running again.

[61] Available from the LOFAR-Contributions GitHub repository at: https://github.com/lofar-astron/LOFAR-Contributions. It can also be invoked directly after having initialized the `Tools` packages (see Sect. 1.3).

[62] http://support.astron.nl/lofar_issuetracker

- 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.<name>.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.

- The `<key name>*.log` files produced by BBS may provide useful information about what went wrong. Inspect these first when BBS has crashed. The log files from `bbs-reducer` are usually located on the compute nodes.

- BBS expects information about the antenna field layout to be present in the MS. The data writer should take care of including this, but in some cases it can fail due to problems during the observation. The program `makebeamtables` can be used to manually add the required information to an existing MS. Documentation is available on the LOFAR Wiki[63].

7.16.1 Common problems

In the following some error messages are reported together with the solution we have found to fix them.

- `Station <name> is not a LOFAR station or the additional information needed to compute the station beam is missing.`

  Solution: Run `makebeamtables` on all subbands to add the information required to compute the station beam model.

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

  Solution: Check your source catalog file. You can run `makesourcedb`[64] locally on your catalog file to get a more detailed error message:

  ```
  makesourcedb in=<catalog> out="test.sky" format="<"
  ```

- `[FAIL] 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.

[63]`http://www.lofar.org/operations/doku.php?id=engineering:software:tools:makebeamtables`
[64]`http://www.lofar.org/operations/doku.php?id=engineering:software:tools:makesourcedb`

8 LoSoTo: LOFAR Solution Tool[65]

The LOFAR Solution Tool (LoSoTo) is a Python package which handles LOFAR solutions in a variety of ways. The data files used by LoSoTo are not in the standard parmdb format used by BBS/NDPPP (e.g. the "instrument" table). LoSoTo uses instead an innovative data file, called H5parm, which is based on the HDF5 standard[66].

WARNING: LoSoTo is still in a beta version! Please report bugs to fdg@hs.uni-hamburg.de and drafferty@hs.uni-hamburg.de. LoSoTo will be soon integrated in the LOFAR environment but up to that moment to use it on cep3 users have to:

```
 source /home/fdg/scripts/losoto/tools/lofarinit.[c]sh
```

8.1 H5parm

H5parm is simply a list of rules which specify how data are stored inside the tables of an HDF5 compliant file. We can say that H5parm relates to HDF5 in the same way that parmdb relates to MeasurementSet. The major advantage of using HDF5 is that it is an opensource project developed by a large community of people. It has therefore a very easy-to-use Python interface (the `pytables` module) and it has better performance than competitors.

8.1.1 HDF5 format

There are three different types of nodes used in H5parm:

Array: all elements are of the same type.

CArray: like Arrays, but here the data are stored in chunks, which allows easy access to slices of huge arrays, without loading all data in memory. These arrays can be much larger than the physically available memory, as long as there is enough disk space.

Tables: each row has the same fields/columns, but the type of the columns can be different within each other. It is a database-like structure.

The use of tables to create a database-like structure was investigated and found to be not satisfactory in terms of performance. Therefore LoSoTo is now based on CArrays organized in a hierarchical fashion which provides enough flexibility but preserves performance.

8.1.2 Characteristics of the H5parm

H5parm is organized in a hierarchical way, where solutions of multiple datasets can be stored in the same H5parm (e.g. the calibrator and the target field solutions of the same observation) into different *solution-sets* (solset). Each solset can be thought as a container for a logically related group of solutions. Although its definition is arbitrary, usually there is one solset for each beam and for each scan. Each solset can have a custom name or by default it is called sol### (where ### is an increasing integer starting from 000).

[65]This section is maintained by Francesco de Gasperin (`fdg@hs.uni-hamburg.de`).
[66]`http://www.hdfgroup.org/HDF5/`

Each solset contains an arbitrary number of *solution-tables* (soltab) plus a couple of Tables with some information on antenna locations and pointing directions. Soltabs also can have an arbitrary name. If no name is provided, then it is by default set to the solution-type name (amplitudes, phases, clock, tec...) plus again an increasing integer (e.g. amplitudes000, phase000...). Since soltab names are arbitrary the proper solution-type is stated in the *parmdb_type* attribute of the soltab node. Supported values are: amplitude, phase, scalarphase, rotation, clock, tec, tecscreen and phase_offset.

Soltabs are also just containers; inside each soltab there are several CArrays which are the real data holders. Typically there are a number of 1-dimensional CArrays storing the *axes* values (see Table 3) and two *n*-dimensional (where *n* is the number of axes) CArrays, "values" and "weights", which contain the solution values and the relative weights.

Soltabs can have an arbitrary number of axes of whatever type. Here we list some examples:

amplitudes : time, freq, pol, dir, ant

phases : time, freq, pol, dir, ant

clock : time, ant

tec : time, ant, dir

foobar : foo, bar...

Theoretically the values/weights arrays can be only partially populated, leaving NaNs (with 0 weight) in the gaps. This allows to have e.g. different time resolution in the core stations and in the remote stations (obviously this ends up in an increment of the data size). Moreover, solution intervals do not have to be equally spaced along any axis (e.g. when one has solutions on frequencies that are not uniformly distributed across the band). The attribute *axes* of the "values" CArrays states the axes names and, more important, their order.

Axis name	Format	Example
time (s)	float64	[4.867e+09, 4.868e+09, 4.869e+09]
freq (Hz)	float64	[120e6,122e6,130e6...]
ant	string (16 char)	[CS001LBA]
pol	string (2 char)	[?XX?, ?XY?, ?RR?, ?RL?]
dir	string (16 char)	[?3C196?,?pointing?]
val	float64	[34.543,5345.423,123.3213]
weight (0 = flagged)	float32 [from 0 to 1]	[0,1,0.9,0.7,1,0]

Table 3: Default names and formats for axes values.

8.1.3 Example of H5parm content

Here is an example of the content of an H5parm file having a single solset (sol000) containing a single soltab (amplitude000).

```
# this is the solset
/sol000 (Group) ''

# this is the antenna Table
/sol000/antenna (Table(36,), shuffle, lzo(5)) 'Antenna names and positions'
```

```
  description := {
  "name": StringCol(itemsize=16, shape=(), dflt='', pos=0),
  "position": Float32Col(shape=(3,), dflt=0.0, pos=1)}
  byteorder := 'little'
  chunkshape := (2340,)

# this is the source Table
/sol000/source (Table(1,), shuffle, lzo(5)) 'Source names and directions'
  description := {
  "name": StringCol(itemsize=16, shape=(), dflt='', pos=0),
  "dir": Float32Col(shape=(2,), dflt=0.0, pos=1)}
  byteorder := 'little'
  chunkshape := (2730,)

# this is the soltab
/sol000/amplitude000 (Group) 'amplitude'

# this is the antenna axis, with all antenna names
/sol000/amplitude000/ant (CArray(36,), shuffle, lzo(5)) ''
  atom := StringAtom(itemsize=8, shape=(), dflt='')
  maindim := 0
  flavor := 'numpy'
  byteorder := 'irrelevant'
  chunkshape := (36,)

# direction axis, with all directions
/sol000/amplitude000/dir (CArray(2,), shuffle, lzo(5)) ''
  atom := StringAtom(itemsize=8, shape=(), dflt='')
  maindim := 0
  flavor := 'numpy'
  byteorder := 'irrelevant'
  chunkshape := (2,)

# frequency axis, with all the frequency values
/sol000/amplitude000/freq (CArray(5,), shuffle, lzo(5)) ''
  atom := Float64Atom(shape=(), dflt=0.0)
  maindim := 0
  flavor := 'numpy'
  byteorder := 'little'
  chunkshape := (5,)

# polarization axis
/sol000/amplitude000/pol (CArray(2,), shuffle, lzo(5)) ''
  atom := StringAtom(itemsize=2, shape=(), dflt='')
  maindim := 0
  flavor := 'numpy'
  byteorder := 'irrelevant'
  chunkshape := (2,)

# time axis
```

```
/sol000/amplitude000/time (CArray(4314,), shuffle, lzo(5)) ''
  atom := Float64Atom(shape=(), dflt=0.0)
  maindim := 0
  flavor := 'numpy'
  byteorder := 'little'
  chunkshape := (4314,)

# this is the CArray with the solutions, note that its shape is the product of all axes
/sol000/amplitude000/val (CArray(2, 2, 36, 5, 4314), shuffle, lzo(5)) ''
  atom := Float64Atom(shape=(), dflt=0.0)
  maindim := 0
  flavor := 'numpy'
  byteorder := 'little'
  chunkshape := (1, 1, 10, 2, 1079)

# weight CArray, same shape of the "val" array
/sol000/amplitude000/weight (CArray(2, 2, 36, 5, 4314), shuffle, lzo(5)) ''
  atom := Float64Atom(shape=(), dflt=0.0)
  maindim := 0
  flavor := 'numpy'
  byteorder := 'little'
  chunkshape := (1, 1, 10, 2, 1079)
```

8.1.4 H5parm benchmarks

For a typical single-SB parmdb of 37 MB the relative H5parm is around 5 MB large. A typical H5parm for an 8 hrs observation using 244 SBs is $\sim$ 3 GB (LBA) and $\sim$ 5 GB (HBA). Reading times between compressed and non-compressed H5parms are comparable within a factor of 2 (compressed is slower). Compared to parmdb the reading time of the python implementation of H5parm (mid-compression) is a factor of a few (2 to 10) faster.

This is a benchmark example:

```
INFO: H5parm filename = L99289-cal_SB081.h5
INFO: parmdb filename = L99289-cal_SB081.MS/instrument/
INFO: ### Read all frequencies for a pol/dir/station
INFO: PARMDB -- 1.9 s.
INFO: H5parm -- 0.28 s.
INFO: ### Read all times for a pol/dir/station
INFO: PARMDB -- 1.85 s.
INFO: H5parm -- 0.28 s.
INFO: ### Read all rotations for 1 station (slice in time)
INFO: PARMDB -- 1.94 s.
INFO: H5parm -- 0.3 s.
INFO: ### Read all rotations for all station (slice in time)
INFO: PARMDB -- 8.05 s.
INFO: H5parm -- 0.26 s.
INFO: ### Read all rotations for remote stations (slice in ant)
INFO: PARMDB -- 3.81 s.
INFO: H5parm -- 1.65 s.
```

```
INFO: ### Read all rotations for a dir/station and write them back
INFO: PARMDB -- 2.01 s.
INFO: H5parm -- 0.47 s.
INFO: ### Read and tabulate the whole file
INFO: parmdb -- 0.67 s.
INFO: H5parm -- 0.02 s.
```

8.2 LoSoTo

LoSoTo is made by several components. It has some tools used mostly to transform parmdb to H5parm and back (see Sec. 8.2.1). A separate program (`losoto.py`) is instead used to perform operations on the specified H5parm. `losoto.py` receives its commands by reading a parset file that has the same syntax of BBS/NDPPP parsets (see Sec.8.2.3).

8.2.1 Tools

There are currently four tools shipped with LoSoTo:

`parmdb_benchmark.py` provide a comparison between parmdb and H5parm for reading/writing

`parmdb_collector.py` fetches parmdb tables from the cluster using a gds file

`H5parm_importer.py` creates an h5parm file from an instrument table (parmdb) or a globaldb created by hand or with `parmdb_collector.py`

`H5parm_merge.py` copy a solset from an H5parm files into another one

`H5parm_exporter.py` export an H5parm to a pre-existing parmdb

The usage of these tools is described in Sec. 8.3.

8.2.2 Operations

These are the operations that LoSoTo can perform:

ABS : takes the absolute value of the solutions (probably most meaningful for amplitudes).

CLIP : clip all solutions n times above and $1/n$ times below the median value (only for amplitudes).

CLOCKTEC : perform clock/tec separation (code maintained by Maaijke Mevius).

FLAG : iteratively remove a general trend from the solutions and then perform noisy region detection and outlier rejection. For phases this is done in real/imaginary space, for amplitude in log space.

FLAGEXTEND : flag a point if surrounded by enough other flags in a chosen N-dimensional space

INTERP : interpolate solutions along whatever (even multiple) axis. Typically one can interpolate in time and/or frequency. This operation can also simply rescale the solutions to match the median of the calibrator solution on a specific axis.

NORM : normalize solutions of an axis to have a chosen average value.

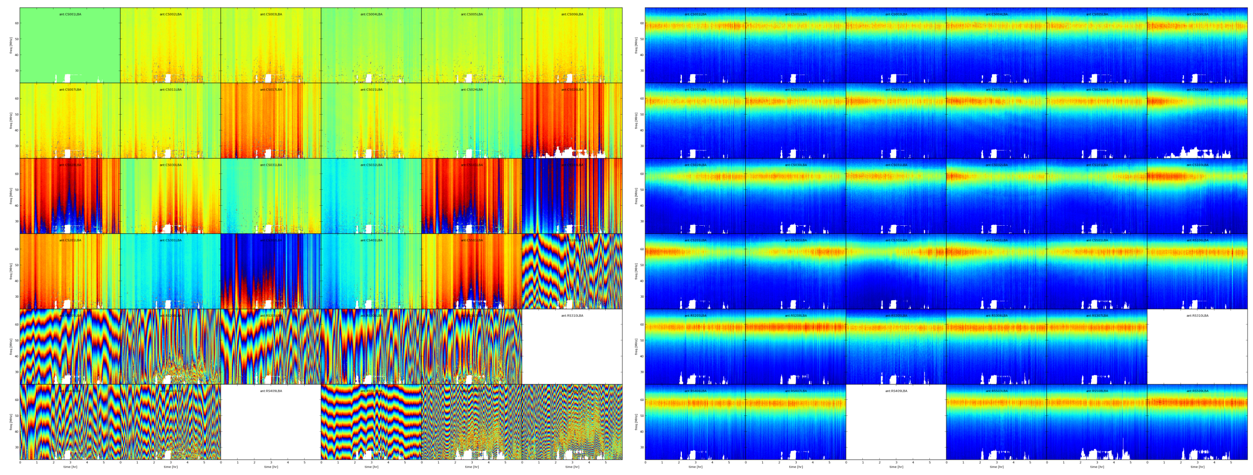


Figure 23: Example of phase (left and amplitude (right) solutions after the FLAG operation. Image made with the PLOT operation. White pixels are flagged data, every plot is an antenna, X-axis is time and Y-axis is frequency.

PLOT : plot solutions in 1D/2D plots.

PLOTTECSCREEN : plot TEC screen (code maintained by David Rafferty).

RESET : reset the solution values to 1 for all kind of solutions but for phases which are set to 0.

REWEIGHT : manually set weights to a specific values (can be used to hand-flag data, e.g. a bad antenna/timerange).

SMOOTH : smooth solutions using a multidimensional running median. The n-dimensional surface generated by multiple axis (e.g. time and freq) can be smoothed in one operation using a different FWHM for each axis.

TECFIT : fit TEC values per direction and station to phase solutions (code maintained by David Rafferty).

TECSCREEN : fit TEC screens to TEC values (code maintained by David Rafferty).

EXAMPLE : this is just an example operation aimed to help developing of new operations.

Beside these operations which require the activation through a LoSoTo parset file (see Sec. 8.2.3), one can call `losoto.py` with the "-i" option passing an H5parm as argument to obtain some information on it. Information on a specific subset of solsets can be obtained with "-i -f solset_name(s)". If "-i" is combined with "-v" (verbose), LoSoTo will take a bit more time and outputs also the percentage of flagged data and the values of all axes will be written in a file (e.g. file.h5-axes_values.txt). Using the "-d" options instead one can delete a chosen soltab (e.g. "losoto.py -d sol000/phase000 file.h5").

```
$ losoto.py -i -v single.h5
WARNING: Axes values saved in single.h5-axes_values.txt

Summary of single.h5

Solution set 'sol000':
======================
Directions: 3C196
            pointing

Stations: CS001LBA   CS002LBA   CS003LBA   CS004LBA
          CS005LBA   CS006LBA   CS007LBA   CS011LBA
          CS017LBA   CS021LBA   CS024LBA   CS026LBA
          CS028LBA   CS030LBA   CS031LBA   CS032LBA
          CS101LBA   CS103LBA   CS201LBA   CS301LBA
          CS302LBA   CS401LBA   CS501LBA   RS106LBA
          RS205LBA   RS208LBA   RS305LBA   RS306LBA
          RS307LBA   RS310LBA   RS406LBA   RS407LBA
          RS409LBA   RS503LBA   RS508LBA   RS509LBA

Solution table 'amplitude000' (type: amplitude): 2 pols, 2 dirs, 36 ants, 5 freqs, 4314
Flagged data 1.131%

    History:
    2014-12-17 13:29:25: CREATE (by H5parm_importer.py from
                         lof011:/home/fdg/scripts/losoto/examples/single.globaldb)

Solution table 'rotation000' (type: rotation): 2 dirs, 36 ants, 5 freqs, 4314 times
Flagged data 1.131%

    History:
    2014-12-17 13:29:25: CREATE (by H5parm_importer.py from
                         lof011:/home/fdg/scripts/losoto/examples/single.globaldb)

Solution table 'phase000' (type: phase): 2 pols, 2 dirs, 36 ants, 5 freqs, 4314 times
Flagged data 1.131%

    History:
    2014-12-17 13:29:25: CREATE (by H5parm_importer.py from
                         lof011:/home/fdg/scripts/losoto/examples/single.globaldb)
```

8.2.3 LoSoTo parset

This is an example parset for the interpolation in amplitude:

```
LoSoTo.Steps = [interp]
LoSoTo.Solset = [sol000]
LoSoTo.Soltab = [sol000/amplitude000]
LoSoTo.SolType = [amplitude]
LoSoTo.ant = []
LoSoTo.pol = [XX,YY]
LoSoTo.dir = []

LoSoTo.Steps.interp.Operation = INTERP
LoSoTo.Steps.interp.InterpAxes = [freq, time]
LoSoTo.Steps.interp.InterpMethod = nearest
LoSoTo.Steps.interp.MedAxes = []
LoSoTo.Steps.interp.Rescale = F
LoSoTo.Steps.interp.CalSoltab = cal000/amplitude000
LoSoTo.Steps.interp.CalDir = 3C295
```

In the first part of the parset "global" values are defined. These are values named LoSoTo.val_name. In Table 4 the reader can find all the possible global values.

Var Name	Format	Example	Comment
LoSoTo.Steps	list of steps	[flag,plot,smoothPhases,plot2]	sequence of steps names in
LoSoTo.Solset	list of solset names	[sol000, sol001]	restrict to these solsets
LoSoTo.Soltab	list of soltabs: "solset/soltab"	[sol000/amplitude000]	restrict to these soltabs
LoSoTo.SolType	list of solution types	[phase]	restrict to soltab of this solu
LoSoTo.ant	antenna names	[CS001_HBA]	restrict to these antennas
LoSoTo.pol	polarizations	[XX, YY]	restrict to these polarizatio
LoSoTo.dir[a]	directions	[pointing, 3C196]	restrict to these pointing di
LoSoTo.freq	frequencies	[30076599.12109375]	restrict to these frequencies
LoSoTo.time	times	[123456789.1234]	restrict to these times
LoSoTo.Ncpu[b]	integer	10	number of processes to spa

[a] it is important to notice that the default direction (e.g. those related to BBS solving for anything that is not "directional": Gain, CommonRotationAngle, CommonScalarPhase...) has the direction named "pointing". [b] only some operations are multiprocess (see Table 5).

Table 4: Definition of global variables in LoSoTo parset.

For every stepname mentioned in the global "steps" variable the user can specify step-specific parameters using the syntax: LoSoTo.Steps.stepname.val_name. At least one of these options must always be present, which is the "Operation" option that specifies which kind of operation is performed by that step among those listed in Sec. 8.2.2. All the global variables (except from the "steps" one) are also usable inside a step to change (override) the selection criteria for that specific step. Selection can be a string (interpreted as a regular expression), a list of values (exact match) or can have a min/max value which is activated using the axisName.minmax sintax (e.g. LoSoTo.freq.minmax = [30e6,1e9] to select data from 30 MHz to 1 GHz). A list of step-specific parameters is given in Table 5.

Var Name	Format	Default	Comment
ABS			
CLIP			
Axes	list of axes names	[time]	Axis name to take medians over, may be multiple. Iteration will be on all other axes.
ClipLevel	float	5	Factor above/below median at which to clip. The cut is done at >ClipLevel and at <1/ClipLevel.
CLOCKTEC			
FlagBadChannels	bool	True	detect and remove bad channel before fitting.
FlagCut	float	1.5	
Chi2cut	float	30000.	
CombinePol	bool	False	Find a combined polarization solution.
FitOffset	bool	False	
FLAG (multiprocess)			
Axis	single axis name	time	Axis to do statistics for flagging.
MaxCycles	int	5	Max number of flagging cycles.
MaxRms	float	5.	Rms to clip outliers.

MaxRmsNoise	float	5.	Rms to clip noisy data region.
Window	int	60	Window used to remove trends in timestamps (e.g. seconds/Hertz).
Order	0|1|2	1	Order of the function fitted during trend removal.
MaxGap	int	300	Maximum gaps allowed before fitting 2 trends in timestamps (e.g. seconds/Hertz), hardly used for LOFAR.
Reaplce	bool	False	Replace bad values with the interpolated ones, instead of flagging them.
PreFlagZeros	bool	False	Flag zeros/ones (bad solutions in BBS). They should be flagged at import time.
EXTENDFLAG (multiprocess)			
Axes	list of axis names	[freq,time]	Axes used to find close flags.
Percent	float	50	percent of flagged data around the point to flag it.
Size	int	11	Size of the window (diameter, per axis), better if odd.
Cyles	int	3	Number of independent cycles of flag expansion.
INTERP			
CalSoltab	soltab name	”	The calibrator solution table (e.g. ’cal000/amplitude000’).
CalDir	dir name	”	Use a specific dir (e.g. 3C295) from CalSoltab instead that the same of the target.
InterpAxes	list of axes names	[time, freq]	The axes along which to interpolate, can be multiple.
InterpMethod	nearest | linear | cubic	linear	Type of interpolation method.
Rescale	bool	False	Just rescale to the median value of CalSoltab, do not interpolate.
MedAxis	axis name	”	Rescale to the median of this axis.
NORM			
NormVal	float	1.	The value to normalize the mean.
NormAxis	axis name	time	The axis to normalize.
PLOT			
Axes	list of axes names	[]	1- or 2-element array which says the coordinates for the axes for a 2 for 3D plot respectively.
MinMax	[float, float]	[0,0]	Force a min/max value for the dependent variable (0 means automatic).
TableAxis	axis name	”	Axis to plot on a page - e.g. ’ant’ to get all antenna’s on one file.
ColorAxis	axis name	”	Axis to plot in different colours - e.g. ’pol’ to get correlations with different colors.
ShadeAxis	axis name	”	Axis to plot is different shades (alpha) - e.g. freq for a small range to compare subband to subband solutions on one plot.
Log	XYZ	”	use Log=’XYZ’ to set which axes to put in Log (e.g. Log=’XZ’ to have X and Z axis in log).

PlotFlag	bool	False	Plot also flagged data (in red)?
Unwrap	bool	False	Unwrap the selected data (1D only).
Reference	antenna name	”	Antenna name for referencing phases.
Prefix	string	”	Give a prefix to saved plots.
Add	table name	”	Tables to ”add” (e.g. ’sol000/tec000’) to solutions, it works only for tec and clock to be added to phases.
RESET			
REWEIGHT			
WeightVal	float (0 to 1)	1.	Set weights to this values.
FlagBad	bool	False	Re-flag bad values.
SMOOTH			
Axes	list of axes names	[freq, time]	Axis name on which to smooth, may be multiple.
FWHM	list of float	[10, 5]	FWHM, one for each axis.
Mode	runningmedian | mean | median	runningmedian	Mean/median set all the solutions to the mean/median.
TECFIT			
Algorithm	algorithm name	sourcediff	The algorithm to use in TEC fitting.
MinBands	int	4	Minimum number of bands a source must have to be used.
MaxStations	int	26	Maximum number of stations to use.
OutSoltab	soltab name	ion000/tec000	the output solution table.
TECSCREEN			
Height	float	200e3	The height in meters of the screen.
Order	int	15	The maximum order of the KL decomposition.
OutSoltab	soltab name	ion000/tecscreen000	the output solution table.

Table 5: Definition of step-specific variables in LoSoTo parset.

8.3 Usage

This is a possible sequence of commands to run LoSoTo on a typical observation:

1. Collect the parmdb of calibrator and target:

```
parmdb_collector.py -v -d "target.gds" -c "clusterdesc" -g globaldb_tgt
parmdb_collector.py -v -d "calibrator.gds" -c "clusterdesc" -g globaldb_cal
```

   where “[target | calibrator].gds” is the gds file (made with combinevds) of all the SB you want to use. You need to run the collector once for the calibrator and once for the target. “Clusterdesc” is a cluster description file as the one used for BBS (not stand-alone).

   One can create the globaldb also by hand. Just create a directory, copy all the instrument (parmdb) tables you need calling them: instrument-1, instrument-2... Then copy from one of the MS (they are all the same) the ANTENNA, FIELD and sky tables. This directory is now a valid globaldb.

2. Convert the set of parmdbs into an h5parm:

```
H5parm_importer.py -v tgt.h5 globaldb_tgt
H5parm_importer.py -v cal.h5 globaldb_cal
```

This command converts ALL the instrument tables (parmdb) inside the globaldb directories in a single solset inside tgt.h5.

One can then merge the two h5parms in a single file (this is needed if you want to interpolate/rescale/copy solutions in time/freq from the cal to the tgt):

```
H5parm_merge.py -v cal.h5:sol000 tgt.h5:cal000
```

An easier approach is to directly append the second globaldb to the h5parm file of the first (note the same name for the h5parm):

```
H5parm_importer.py -v tgt.h5 globaldb_tgt
H5parm_importer.py -v tgt.h5 globaldb_cal
```

One can create a h5parm also from a single SB:

```
H5parm_importer.py -v LXXXXX_3c295.h5 LXXXXX_3c295.MS
```

This command converts the "instrument" (parmdb) table inside the MS in a solset inside LXXXXX_3c295.h
Note that given the definition of globaldb above, a single-SB measurementset is a perfect valid one.

3. Run LoSoTo using e.g. the parset given in Sec. 8.2.3:

```
losoto.py -v tgt.h5 losoto-interp.parset
```

4. Convert back the h5parm into parmdb:

```
H5parm_exporter.py -v -c tgt.h5 globaldb_tgt
```

5. Redistribute back the parmdb tables into globaldb_tgt that are now updated (inspect with parmdbplot), there's no automatic tool for that yet.

8.4 Developing in LoSoTo

LoSoTo is much more than a stand alone program, the user can use LoSoTo to play easily with solutions and to experiment. The code is freely available and is already the result of the collaborative effort of several people. If interested in developing your own operation, please have a look at: `https://github.com/revoltek/losoto/`.

In the "tools" directory the user can find all the tools described in Sec. 8.2.1 plus some other program. All these programs are stand-alone. `losoto.py` is the main program which calls all the operations (one per file) present in the "operation" directory. It relays on the `h5parm.py` library which deals with reading and writing from an H5parm file and the `operations_lib.py` library which has some functions common to several operations.

An example operation one can use to start coding its own, is present in the "operation" directory under the name "`example.py`". That is the first point to start when interested in writing a new operation. The most important thing shown in the example operation is the use of the H5parm library to quickly fetch an write back solutions in the H5parm file.

8.5 Clock/TEC separation

In LoSoTo an algorithm is implemented with which it is possible to separate the BBS phase solutions into a instrumental delay (clock) and an ionospheric component (TEC, a measure of the differential ionospheric electron content). This is done using the difference in frequency dependence of both effects: The phase shift due to a delay error goes as ν, whereas ionospheric refraction gives to first order an phase shift proportional to $1/\nu$. Accordingly, one needs to have solutions over a large enough bandwidth to be able to do the separation [67].

There are three situations for which Clock/TEC separation could be useful. The first is if one needs to transfer from a calibrator not only the amplitudes, but also the phase solutions, eg. to be able to combine more sub-bands before doing a phase self-calibration on the target field. Since the ionospheric refraction is a direction dependent effect, in most cases it does not make sense to transfer the ionospheric phases. It is then possible to only apply the clock solutions from the calibrator to the target field. However, one should take note that the delays between stations drift and therefore this method is only useful if calibrator data was taken simultaneously with the target field.

A more experimental case for Clock/TEC separation is the fit of a TECscreen that can be applied during imaging with the AWimager, to correct for the direction dependent ionospheric effects. In this case, it is good to remove the direction independent instrumental effects as good as possible and only use the fitted TEC as input for the TECscreen. This has only been tested in very limited cases.

Finally, the TEC solutions of the clock/TEC separation give insight in the general ionospheric conditions of your observation. This could be of relevance if one wants eg. to estimate the noise due to remaining ionospheric errors.

It is important (especially for LBA) to correct for differential Faraday rotation before attempting to do a clock/TEC separation on the diagonal phases. The most straightforward way to do this, is to solve in BBS for diagonal gains and a common rotation angle.

The Clock/TEC fit as it is implemented in LoSoTo, returns per timeslot and station two arrays, one with clock errors (in s) and one with differential TEC solutions (in TEC-units). Furthermore a constant (in time) phase offset per station can be estimated. The remaining phase errors (eg. due to cable reflections) are of second order.

It is possible to write the clock solutions to the instrument tables and thus correct for them in BBS.

[67]The exact bandwidth requirements have not been tested yet, but it has been shown that on good S/N (calibrator) HBA data, it is possible to do the clock/TEC separation with 15 solutions, evenly distributed over 60 MHz.

9 SAGECAL[68]

This chapter is a step by step description of the SAGECAL method for self-calibration of LOFAR data. The mathematical framework of the algorithm can be found in Yatawatta et al. 2009 and Kazemi et al. 2011. The contents are applicable to the latest version (0.3.8) of SAGECAL and older versions are obsolete. The latest source code can be obtained from http://sagecal.sf.net.

9.1 Introduction

The acronym SAGECAL stands for Space Alternating Generalized Expectation Maximization Calibration. The Expectation Maximization is used as a solution to maximum likelihood estimation to reduce the computational cost and speed of convergence. The commonly used Least Squared calibration method involves the inversion of a matrix corresponding to the full set of unknown parameters; where the convergence to a local minimum impels a slow speed of convergence and significant computational cost. The SAGE algorithm, on the other hand, allows to compute a direct estimation of subsets unknown parameters, providing faster convergence and/or reduced computational costs. If K is the number of sources and N are the stations, the computational cost scales as $\mathscr{O}((KN)^2)$ for the Least Squared, while is $\mathscr{O}(KN^2)$ for the Expectation Maximization.

BBS uses the Least Squared algorithm to solve the Measurement Equation. The maximum number of directions for which solve using directional gains with BBS is currently limited to 5 or 6 directions; this is a consequence of the limited computing power available in CEP2 and CEP3. The typical LOFAR Field customarily requires to solve for a number of directions generally higher than 5 or 6, this is due to the wide field of view (FOV), variable beam pattern and ionospheric errors. SAGECAL has been tested to solve to a maximum of 300 directions (in the GPU EoR cluster not in CEP2 or CEP3) and 512 stations. The version installed in the CEP clusters is not optimized so that other users can also use the same node while SAGECAL is running; e.g. a total of about 50 directions have been successfully tested in CEP1 cluster.

9.2 Using SAGECAL

9.2.1 Data preparation

Before running SAGECAL, you need to take a few precautions. First of all, the data format is the Measurement Set (MS), so no extra conversions are necessary. Averaging in time is not essential, but it is definitely recommended in order to work with smaller datasets (you can average down to e.g. 10 seconds). If data have been already demixed, this step is not needed. The MS can have more than one channel. Also it is possible to calibrate more than one MS together in SAGECAL. This might be useful to handle situations where the data is very noisy. An interesting note about SAGECAL: If the conventional demixing could not be performed on your data (e.g. if the A-team was too close to the target source) you will be able to successfully remove the A-team sources using this new algorithm by working on averaged data(!)
Another procedure you will need apply to your data before SAGECAL is to calibrate them in the standard way with BBS, solving for the four G Jones elements as well as for the element beam. After BBS, you will need to flag the outliers using DPPP. You are now ready to start the algorithm.

All the programs related to the SAGECAL can be found in **/opt/cep/sagecal/bin/** or see **/home/sarod/bin** for the latest build.

[68]The authors of this chapter are Emanuela Orrú (`e.orru[at]astro[dot]rug[dot]nl`) and Sarod Yatawatta (`yatawatta[at]astron[dot]nl`).

9.2.2 Model

Make an image of your MS (using casapy or awimager). You can use Duchamp to create a mask for the image. To create a sky model, you can adopt buildsky for point sources and shapelet_gui in case of extended emission (see Chapter 14) . Note that you are free to use any other source finder (like PyBDSM - see (Chapter 11)) if you feel more confident with it. A new functionality has been implemented to transform the the BBS model into the model format used by SAGECAL (Chapter 14. The important is that in the sky model any source name starting with 'S' indicates shapelet, 'D' a disk, 'R' a ring, 'G' a Gaussian, while others keys are point sources. In the following, we report a few useful sky models examples:

```
## name h m s d m s I Q U V spectral_index RM extent_X(rad) extent_Y(rad)
## pos_angle(rad) freq0
P1C1 0 12 42.996 85 43 21.514 0.030498 0 0 0 -5.713060 0 0 0 0 115039062.0
P5C1 1 18 5.864 85 58 39.755 0.041839 0 0 0 -6.672879 0 0 0 0 115039062.0

# A Gaussian mjor,minor 0.1375,0.0917 deg diameter, pa 43.4772 deg
G0  5 34 31.75 22 00 52.86 100 0 0 0 0.00 0 0.0012  0.0008 -2.329615801 130.0e6

# A Disk radius=0.041 deg
D01 23 23 25.67 58 48 58 80 0 0 0 0 0 0.000715 0.000715 0 130e6

# A Ring radius=0.031 deg
R01 23 23 25.416 58 48 57 70 0 0 0 0 0 0.00052 0.00052 0 130e6

# A shapelet ('S3C61MD.fits.modes' file must be in the current directory)
S3C61MD 2 22 49.796414 86 18 55.913266 0.135 0 0 0 -6.6 0 1 1 0.0 115000000.0
```

Note that it is also possible to have sources with 3rd order spectra (with -F 1 option). Here is such an example:

```
##  name h m s d m s I Q U V spectral_index0 spectral_index1 spectral_index2
## RM extent_X(rad) extent_Y(rad) pos_angle(rad) freq0
PJ1C1 18 53 33.616 86 10 19.559 0.008594 0 0 0 -5.649676 -2.0 -60.0 0 0 0 0 \
      152391463.2
```

But all the sources should have either 1st order or 3rd order spectra, mixing is not allowed.

The number of directions you want to solve for are described in the cluster file. In here, one or more sources corresponding to the patch of the skymodel you need to correct for are combined together as the following example:

```
## cluster_id chunk_size source1 source2 ...
0 1 P0C1 P0C2
1 3 P11C2 P11C1 P13C1
2 1 P2C1 P2C2 P2C3
```

Note: comments starting with a '#' are allowed for both sky model and cluster files.

9.2.3 SAGECAL

SAGECAL will solve for all directions described in the cluster file and will subtract the sources listed in the sky model. Note that if you are not interested in the residual data, putting negative values for `cluster_id` will not subtract the corresponding sources from data. Run SAGECAL as follows:

```
sagecal -d my.MS -s my_skymodel -c my_clustering -t 120 -p my_solutions
```

This will read the data from the DATA column of the MS and write the calibrated data to the CORRECTED_DATA column. If these columns are not present, you have to create them first. If you need to calibrate more than one MS together, first create a text file with all the MS names, line by line. Then run

```
sagecal -f MS_names.txt -s my_skymodel -c my_clustering -t 120 -p my_solutions
```

Running sagecal -h will provide the additional options, some of them are:

```
-F sky model format: 0: LSM, 1: LSM with 3 order spectra : default 0
-I input column (DATA/CORRECTED_DATA) : default DATA
-O ouput column (DATA/CORRECTED_DATA) : default CORRECTED_DATA
-e max EM iterations : default 3
-g max iterations  (within single EM) : default 2
-l max LBFGS iterations : default 10
-m LBFGS memory size : default 7
-n no of worker threads : default 6
-t tile size : default 120
-x exclude baselines length (lambda) lower than this in calibration : default 0

Advanced options:
-k cluster_id : correct residuals with solution of this cluster : default -99999
-j 0,1,2... 0 : OSaccel, 1 no OSaccel, 2: OSRLM, 3: RLM:  4: RTR, 5: RRTR: default 0
```

Use a solution interval (e.g. -t 120) that is big enough to get a decent solution and not too big to make the parameters vary too much (about 20 minutes per solution is a reasonable value).
In case of both bright and faint sources in the model, you might need to use different solution intervals for different clusters. In order to do that you need to define the values of the second column of the cluster file in such a way that the cluster with the longest solution interval is 1. While the cluster with shorter solution interval will be equal to n, where n is the number of times the longer solution interval is divided. This means that if -t 120 is used to select 120 timeslots, cluster 0 will find a solution using the full 120 timeslots, while cluster 1 will solve for every 120/3=40 timeslots. The option -k will allow to correct the residuals using the solutions calculated for a specific direction which is defined by the cluster_id. The -k option is analogue to the correct step in BBS. See the simulations section later in this chapter for more detailed use of this.

You are now ready to image the residual data. Successively, run “restore” (see Sect. 14) on the residual image before updating the sky model and starting another loop of SAGECAL. SAGECAL cycles can be done till you are satisfied by the end product.

9.2.4 Robustness

Many have experienced flux loss of weaker background sources after running any form of directional calibration. There is a new algorithm in SAGECAL that minimizes this. To enable this, use -j 2 option while running SAGECAL. The theory can be found in Kazemi and Yatawatta, 2013. Also for best speed and robustness, it is recommended to use -j 5. Also for number of stations greater than 64, -j 5 option is faster and less memory consuming.

9.3 Simulations

After running SAGECAL, you get solutions for all the directions used in the calibration. Using these solutions, it is possible to correct the data for each direction and make images of that part of the sky. This will in theory enable to image the full field of view with correct solutions applied to each direction. This is a brief description on how to do it. We will use an example to illustrate this. Assume you have run SAGECAL as below:

```
sagecal -d my.MS -s my_skymodel -c my_clustering -t 120 -p my_solutions
```

Now you have the my_solutions file that contains solutions for all the directions given in my_clustering file. The -a 1 option or -a 2 option enables simulation mode in SAGECAL. If -a 1 is given, the sky model given by my_skymodel and my_clustering is simulated (with gains taken from solutions if -p my_solutions is given) and written to the output data. If -a 2 is given, the model given by my_skymodel and my_clustering is simulated (with the gains if -p my_solutions is given), and then added to the input data (-I option) and written to the output data (-O option).

There is one additional thing you can do while doing a simulation: correction for any particular direction. By using -k cluster_id , you can select any cluster number from the my_clustering file to use as the solutions that are used to correct the data. If the cluster_id specified is not present in the my_clustering file, no corrections will be applied.

Simulating the full sky model back to the data might be too much in some cases. It is also possible to simulate only a subset of clusters back to the data. This is done by specifying a list of clusters to ignore during the simulation, using -z ignore_list option. Here, ignore_list is a text file with the cluster numbers (one per line) to ignore. Note also that even while clusters are ignored, their solutions can still be used to correct the data (so these options are independent from each other).

Thus, by cleverly using the -a, -k and -z options, you can get an image that is fully corrected (using SAGECAL solutions) for all directional errors and moreover, simulations take only a fraction of the time taken for calibration.

9.4 Distributed Calibration

It is possible to use SAGECAL to calibrate a large number of subbands, distributed across many computers. This is done by using OpenMPI and SAGECAL together and see sagecal-mpi -h for further information.

References

S. Yatawatta et al., “Radio interferometric calibration using the SAGE algorithm,” *IEEE DSP*, Marco Island, FL, Jan. 2009.

S. Kazemi and S. Yatawatta et al., “Radio interferometric calibration using the SAGE algorithm,” *MNRAS, 414-2*, 2011.

S. Kazemi and S. Yatawatta, “Robust radio interferometric calibration using the T-distribution,” *MN-RAS*, 2013.

10 The AW imager[69]

10.1 Introduction

In this section we will describe the necessary steps needed to perform successful imaging of LOFAR data using the AWimager[70].

Note that the AWimager is still in the development phase, therefore this documentation is very dynamic and it is currently meant to provide the basic instructions on how to use the code.

In particular, a new version of the AWImager is scheduled for mid 2015. This new version will have more features, like multi-frequency clean. The options for AWImager2 are different (more intuitive) than that of AWImager. It will be documented in the next release of the Cookbook.

10.2 Background

The AWimager is specially adapted to image wide fields of view, and imaging data produced by non-coplanar arrays, where the W term in the measured visibilities is not negligible. Furthermore, AWimager corrects for direction dependent effects (LOFAR beam and ionosphere) varying in time and frequency. The used algorithm is A-projection.

The algorithm is implemented using some CASA libraries.

10.3 Usage

To run AWimager, you first need to setup your environment using:

```
use Lofar
```

Before running AWimager, it is necessary to calibrate the dataset and correct the visibilities towards the phase center of the observation. This can now be done by not specifying any direction in the *correct* step of BBS.

```
Step.correct.Model.Sources = []
```

AWimager can run in a parallel fashion. The number of processing cores (`n`) to be used during imaging can be specified:

```
export OMP_NUM_THREADS=n
```

If not specified, all cores will be used.
AWimager is quite memory hungry, so the number of cores should be limited in case it fails due a `'bad alloc'` error.

[69]This Chapter is maintained by Bas van der Tol (`tol[at]astron[dot]nl`).
[70]Cyril Tasse, Bas van der Tol, Joris van Zwieten, Ger van Diepen, Sanjay Bhatnagar 2013, *Applying full polarization A-Projection to very wide field of view instruments: An imager for LOFAR*. A&A, 553, A105

10.4 Output files

AWimager creates several image output files. Note that in the following list `<image>` is the image name given using the `image` parameter.

- `<image>.model` is the uncorrected dirty image.
- `<image>.model.corr` is the dirty image corrected for the average primary beam.
- `<image>.restored` and `<image>.restored.corr` are the restored images.
- `<image>.residual` and `<image>.residual.corr` are the residual images.
- `<image>.psf` is the point spread function.
- `<image>0.avgpb` is the average primary beam.

Furthermore, a few other files might be created for AWimager's internal use.

10.5 Parameters

An extensive list of the parameters that can be used by the AWimager can be obtained by typing:

```
awimager -h
```

Eventually, to run the imager, you can type:

```
awimager ms=test.MS image=test.img weight=natural wprojplanes=64 npix=512
cellsize=60arcsec data=CORRECTED_DATA padding=1. niter=2000
timewindow=300 stokes=IQUV threshold=0.1Jy operation=csclean
```

which is one command spread over multiple lines.
It is also possible to specify these parameters in a parset and run it like:

```
awimager parsetname
```

Many parameters can be set for the AWimager. Several of them are currently being tested by the commissioners. The most important parameters are listed below.

10.5.1 Data selection

These parameters specify the input data.

- `ms`
  The name of the input MeasurementSet.
- `data`
  The name of the data column to use in the MeasurementSet. The default is DATA.
- `antenna`
  Baseline selection following the CASA baseline selection syntax

- `wmax`
  Ignore baselines whose w-value exceeds wmax (in meters).
- `uvdist`
  Ignore baselines whose length (in wavelengths) exceed uvdist.
- `select`
  Only use data matching this TaQL selection string. For example, `sumsqr(UVW[:2])<1e8` selects baselines with length $<$10km.

10.5.2 Image properties

These parameters define the properties of the output image.

- `image`
  The name of the image.
- `npix`
  The number of pixels in the RA and DEC direction
- `cellsize`
  The size of each pixel. An unit can be given at the end. E.g. `30arcsec`
- `padding`
  The padding factor to use when imaging. It can be used to get rid of effects at the edges of the image. If, say, 1.5 is given, the number of pixels used internally is 50% more.
- `stokes`
  The Stokes parameters for which an image is made. If A-projection is used, it must be IQUV.

10.5.3 weighting

These parameters select the weighting scheme to be used.

- `weight`
  Weighting scheme (uniform, superuniform, natural, briggs (robust), briggsabs, or radial)
- `robust`
  Robust weighting parameter.

10.5.4 Operation

This parameter selects the operation to be performed by awimager.

- `operation`
  The operation to be performed by the AWimager.
  - csclean = make an image and clean it (using Cotton-Schwab).
  - multiscale = use multiscale cleaning
  - image = make dirty image only.
  - predict = fill the data column in the MeasurementSet by predicting the data from the image.
  - empty = make an empty image. This can be useful if only image coordinate info is needed.

10.5.5 Deconvolution

These parameters control the deconvolution algorithm. Only those parameters that are applicable to the selected operation will be used.

- `niter`
  The number of clean iterations to be done. The default is 1000.
- `gain`
  The loop gain for cleaning. The default is 0.1.
- `threshold`
  The flux level at which to stop cleaning. The default is 0Jy.
- `uservector`
  Comma separated list of scales (in pixels) to be used by multiscale deconvolution

10.5.6 Gridding

These parameters control the AW-projection algorithm.

- `wprojplanes`
  The number of W projection planes to use.
- `maxsupport`
  The maximum of W convolution functions. The default is 1024.
- `oversample`
  The oversampling to use for the convolution functions. The default is 8.
- `timewindow`
  The width of the time window (in sec) where the AW-term is assumed to be constant. Default is 300 sec. The wider the window, the faster the imager will be.
- `splitbeam`
  Evaluate station beam and element beam separately? The default is true. AWimager will work much faster if the correction for the station and element beam can be applied separately. This should only be done if the element beam is the same for all stations used.

For more details, the user can refer to the Busy Wednesday comissioning reports[71] (specifically those from September 28 and October 26 2011).

[71] `http://www.lofar.org/operations/doku.php?id=commissioning:busy_wednesdays`

11 Source detection and sky model manipulation: PyBDSM and LSMTool[72]

11.1 Source detection: PyBDSM

11.1.1 Introduction

PyBDSM (**Py**thon **B**lob **D**etection and **S**ource **M**easurement) is a Python source-finding software package written by Niruj Mohan, Alexander Usov, and David Rafferty. PyBDSM can process FITS and CASA images and can output source lists in a variety of formats, including `makesourcedb` (BBS), FITS and ASCII formats. It can be used interactively in a casapy-like shell or in Python scripts. The full PyBDSM manual is located at `http://tinyurl.com/PyBDSM-doc`.

11.1.2 Recent Changes

Changes to PyBDSM since the last version of the cookbook include:

- New output option (`bbs_patches = 'mask'`) to allow patches in an output BBS sky model to be defined using a mask image (set with the `bbs_patches_mask` option).
- Many minor bug fixes (use `help changelog` for details).

11.1.3 Setup

The latest version of PyBDSM is installed on the CEP clusters. To initialize your environment for PyBDSM, run:

```
> use Lofar
```

After initialization, the interactive PyBDSM shell can be started with the command `pybdsm` and PyBDSM can be imported into Python scripts with the command `from lofar import bdsm`.

11.1.4 Usage

The following describes how to run an analysis using the PyBDSM interactive interface. For details on using PyBDSM directly in Python scripts, see Section 11.1.6.

After initialization (see above), the PyBDSM interactive shell is started from the prompt with the command `pybdsm`. Upon startup, the version number and a brief overview of the available commands and tasks are shown:

```
> pybdsm
PyBDSM version 1.8.3 (LOFAR revision 30100)
========================================================================
PyBDSM commands
  inp task ............ : Set current task and list parameters
  par = val ........... : Set a parameter (par = '' sets it to default)
```

[72]This section is maintained by David Rafferty (`drafferty[at]hs[dot]uni-hamburg[dot]de`).

```
                        Autocomplete (with TAB) works for par and val
  go .................. : Run the current task
  default ............. : Set current task parameters to default values
  tput ................ : Save parameter values
  tget ................ : Load parameter values
PyBDSM tasks
  process_image ....... : Process an image: find sources, etc.
  show_fit ............ : Show the results of a fit
  write_catalog ....... : Write out list of sources to a file
  export_image ........ : Write residual/model/rms/mean image to a file
PyBDSM help
  help command/task ... : Get help on a command or task
                          (e.g., help process_image)
  help 'par' .......... : Get help on a parameter (e.g., help 'rms_box')
  help changelog ...... : See list of recent changes
________________________________________________________________________________
```

A standard analysis is performed using the process_image task. This task reads in the input image, calculates background rms and mean images, finds islands of emission, fits Gaussians to the islands, and groups the Gaussians into sources. Use inp process_image to list the parameters:

```
BDSM [1]: inp process_image
--------> inp(process_image)
PROCESS_IMAGE: Find and measure sources in an image.
================================================================================
filename ................ '': Input image file name
adaptive_rms_box ..... False : Use adaptive rms_box when determining rms and
                               mean maps
advanced_opts ........ False : Show advanced options
atrous_do ............ False : Decompose Gaussian residual image into multiple
                               scales
beam .................. None : FWHM of restoring beam. Specify as (maj, min, pos
                               ang E of N) in degrees. E.g., beam = (0.06, 0.02,
                               13.3). None => get from header
flagging_opts ........ False : Show options for Gaussian flagging
frequency ............. None : Frequency in Hz of input image. E.g., frequency =
                               74e6. None => get from header. For more than one
                               channel, use the frequency_sp parameter.
interactive .......... False : Use interactive mode
mean_map .......... 'default': Background mean map: 'default' => calc whether to
                               use or not, 'zero' => 0, 'const' => clipped mean,
                               'map' => use 2-D map.
multichan_opts ....... False : Show options for multi-channel images
output_opts .......... False : Show output options
polarisation_do ...... False : Find polarisation properties
psf_vary_do .......... False : Calculate PSF variation across image
rms_box ............... None : Box size, step size for rms/mean map calculation.
                               Specify as (box, step) in pixels. E.g., rms_box =
                               (40, 10) => box of 40x40 pixels, step of 10
                               pixels. None => calculate inside program
rms_map ............... None : Background rms map: True => use 2-D rms map;
```

```
                                False => use constant rms; None => calculate
                                inside program
shapelet_do ......... False : Decompose islands into shapelets
spectralindex_do ..... False : Calculate spectral indices (for multi-channel
                                image)
thresh ............... None : Type of thresholding: None => calculate inside
                                program, 'fdr' => use false detection rate
                                algorithm, 'hard' => use sigma clipping
thresh_isl ............ 3.0 : Threshold for the island boundary in number of
                                sigma above the mean. Determines extent of
                                island used for fitting
thresh_pix ............ 5.0 : Source detection threshold: threshold for the
                                island peak in number of sigma above the mean. If
                                false detection rate thresholding is used, this
                                value is ignored and thresh_pix is calculated
                                inside the program
```

Standard autocompletion of command, task, and parameter names and values is available with the <TAB> key. As in casapy, inp prints the parameter names and their current values, as well as a short description of each parameter. Full information about a parameter is available with the help command (e.g., help 'mean_map'). Once parameters are set, the analysis can be run with the command go. The progress of the fit is printed to the screen, and a log file (named as the input image name with ".pybdsm.log" appended) with additional information about the fit will be saved in the current directory.

Once processing has finished, the results of the fit may be inspected using the show_fit task:

```
BDSM [1]: inp show_fit
--------> inp(show_fit)
SHOW_FIT: Show results of fit.
================================================================================
broadcast ........... False : Broadcast Gaussian and source IDs and coordinates
                                to SAMP hub when a Gaussian is clicked?
ch0_flagged ......... False : Show the ch0 image with flagged Gaussians (if
                                any) overplotted
ch0_image ............ True : Show the ch0 image. This is the image used for
                                source detection
ch0_islands .......... True : Show the ch0 image with islands and Gaussians (if
                                any) overplotted
gmodel_image ......... True : Show the Gaussian model image
gresid_image ......... True : Show the Gaussian residual image
mean_image ........... True : Show the background mean image
pi_image ............ False : Show the polarized intensity image
psf_major ........... False : Show the PSF major axis variation
psf_minor ........... False : Show the PSF minor axis variation
psf_pa .............. False : Show the PSF position angle variation
pyramid_srcs ........ False : Plot the wavelet pyramidal sources
rms_image ............ True : Show the background rms image
smodel_image ........ False : Show the shapelet model image
source_seds ......... False : Plot the source SEDs and best-fit spectral
                                indices (if image was processed with
                               spectralindex_do = True). Sources may be chosen
                               by ID with the 'c' key or, if ch0_islands = True,
                               by picking a source with the mouse
sresid_image ......... False : Show the shapelet residual image
```

Internally derived images (e.g, the Gaussian model image) can be exported to FITS files using the export_image task:

```
BDSM [1]: inp export_image
--------> inp(export_image)
EXPORT_IMAGE: Write one or more images to a file.
================================================================================
outfile .............. None : Output file name. None => file is named
                              automatically; 'SAMP' => send to SAMP hub (e.g.,
                              to TOPCAT, ds9, or Aladin)
clobber ............. False : Overwrite existing file?
img_format ........... 'fits': Format of output image: 'fits' or 'casa'
img_type ....... 'gaus_resid': Type of image to export: 'gaus_resid',
                              'shap_resid', 'rms', 'mean', 'gaus_model',
                              'shap_model', 'ch0', 'pi', 'psf_major',
                              'psf_minor', 'psf_pa', 'psf_ratio',
                              'psf_ratio_aper', 'island_mask'
mask_dilation ........... 0 : Number of iterations to use for island-mask
                              dilation. 0 => no dilation
pad_image ........... False : Pad image (with zeros) to original size
```

Lastly, the positions, fluxes, etc. of the fitted Gaussians may be written in a number of formats to a file using the write_catalog task:

```
BDSM [1]: inp write_catalog
--------> inp(write_catalog)
WRITE_CATALOG: Write the Gaussian, source, or shapelet list to a file.
================================================================================
outfile .............. None : Output file name. None => file is named
                              automatically; 'SAMP' => send to SAMP hub (e.g.,
                              to TOPCAT, ds9, or Aladin)
bbs_patches .......... None : For BBS format, type of patch to use: None => no
                              patches. 'single' => all Gaussians in one patch.
                              'gaussian' => each Gaussian gets its own patch.
                              'source' => all Gaussians belonging to a single
                              source are grouped into one patch. 'mask' => use
                              mask file specified by bbs_patches_mask
bbs_patches_mask ...... None : Name of the mask file (of same size as input
                              image) that defines the patches if bbs_patches =
                              'mask'
catalog_type ......... 'srl': Type of catalog to write:  'gaul' - Gaussian
                              list, 'srl' - source list (formed by grouping
                              Gaussians), 'shap' - shapelet list (FITS format
                              only)
clobber ............. False : Overwrite existing file?
correct_proj ......... True : Correct source parameters for image projection
```

```
                                  (BBS format only)?
format .............. 'fits': Format of output catalog: 'bbs', 'ds9', 'fits',
                                  'star', 'kvis', 'ascii', 'csv', 'casabox', or
                                  'sagecal'
incl_chan ........... False : Include flux densities from each channel (if
                                  any)?
incl_empty .......... False : Include islands without any valid Gaussians
                                  (source list only)?
srcroot ............... None : Root name for entries in the output catalog (BBS
                                  format only). None => use image file name
```

The information included in the output varies depending on the format used: with the ASCII and FITS formats, all Gaussian or source parameters are included; with the other formats, only a subset of parameters are included. See the PyBDSM manual (`http://tinyurl.com/PyBDSM-doc`) for details. Additionally, in the BBS output file, sources with sizes smaller than the beam are denoted as point sources.

Upon the successful completion of any task, all parameters are saved to the file "pybdsm.last" in the current directory and may be reloaded using the command `tget`. The current parameters may also be saved to "pybdsm.last" at any time using `tput`. If a file name is given to the `tput` or `tget` commands (e.g., `tput '3C196.sav'`), the parameters are saved to or loaded from the given file.

11.1.5 Examples

This section gives examples of using PyBDSM on the following: an image that contains only fairly simple sources and no strong artifacts, an image with strong artifacts around bright sources, and an image with complex diffuse emission. It is recommended that interactive mode (enabled with `interactive=True`) be used for initial runs on a new image, as this allows the user to check the background mean and rms images and the islands found by PyBDSM before proceeding to fitting. Also, if a very large image is being fit, it is often helpful to run on a smaller (but still representative) portion of the image (defined using the `trim_box` parameter) to verify that the chosen parameters are appropriate before fitting the entire image.

11.1.5.1 Simple Example A simple example of using PyBDSM on a LOFAR image (an HBA image of 3C61.1) is shown below. In this case, default values are used for all parameters. Generally, the default values work well on images that contain relatively simple sources with no strong artifacts.

```
BDSM [1]: inp process_image
BDSM [2]: filename = 'sb48.fits'
BDSM [3]: go
--------> go()
--> Opened 'sb48.fits'
Image size .............................. : (256, 256) pixels
Number of channels ...................... : 1
Beam shape (major, minor, pos angle) .... : (0.002916, 0.002654, -173.36) degrees
Frequency of averaged image ............. : 146.497 MHz
Blank pixels in the image ............... : 0 (0.0%)
Flux from sum of (non-blank) pixels ..... : 29.565 Jy
Derived rms_box (box size, step size) ... : (61, 20) pixels
--> Variation in rms image significant
--> Using 2D map for background rms
--> Variation in mean image significant
--> Using 2D map for background mean
Min/max values of background rms map .... : (0.05358, 0.25376) Jy/beam
Min/max values of background mean map ... : (-0.03656, 0.06190) Jy/beam
--> Expected 5-sigma-clipped false detection rate < fdr_ratio
--> Using sigma-clipping thresholding
Number of islands found ................. : 4
Fitting islands with Gaussians .......... : [====] 4/4
Total number of Gaussians fit to image .. : 12
Total flux in model ..................... : 27.336 Jy
Number of sources formed from Gaussians   : 6

BDSM [4]: show_fit
--------> show_fit()
========================================================================
NOTE -- With the mouse pointer in plot window:
  Press "i" ........ : Get integrated fluxes and mean rms values
                       for the visible portion of the image
  Press "m" ........ : Change min and max scaling values
  Press "0" ........ : Reset scaling to default
  Click Gaussian ... : Print Gaussian and source IDs (zoom_rect mode,
                       toggled with the "zoom" button and indicated in
                       the lower right corner, must be off)
------------------------------------------------------------------------
```

The figure made by `show_fit` is shown in Figure 24. 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 TOPCAT or BBS, as follows:

```
BDSM [5]: write_catalog
--------> write_catalog()
--> Wrote FITS file 'sb48.pybdsm.srl.fits'
```

The output Gaussian or source list contains source positions, fluxes, etc. BBS patches are also supported.

11.1.5.2 Image with Artifacts Occasionally, an analysis 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 25. It is clear that a number of spurious sources are being detected. Simply raising the threshold for island detection (using the `thresh_pix` 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 63 pixels when run with the default parameters, resulting in an rms map that is over-smoothed. Therefore, one should set `rms_box=(20,10)` so that the rms map is computed using a box of 20 pixels in size with a step size

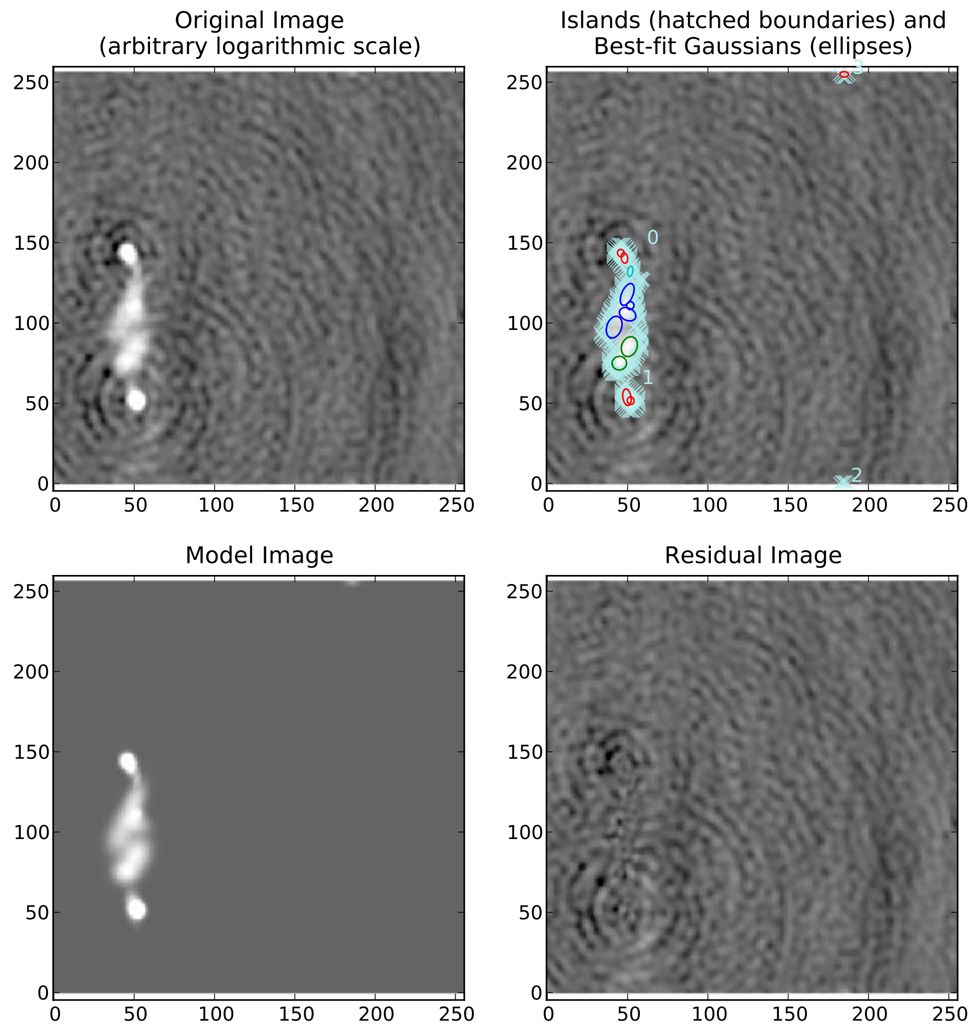


Figure 24: Output of `show_fit`, showing the original image with and without sources, the model image, and the residual (original minus model) image. Boundaries of the islands of emission found by PyBDSM are shown in light blue. The fitted Gaussians are shown for each island as ellipses (the sizes of which correspond to the FWHMs of the Gaussians). Gaussians that have been grouped together into a source are shown with the same color. For example, the two red Gaussians of island #1 have been grouped together into one source, and the nine Gaussians of island #0 have been grouped into 4 separate sources. The user can obtain information about a Gaussian by clicking on it. Additionally, with the mouse inside the plot window, the display scaling can be modified by pressing the "m" key, and information about the image flux, model flux, and rms can be obtained by pressing the "i" key.

of 10 pixels (i.e., the box is moved across the image in 10-pixel steps). See Figure 26 for a summary of the results of this call.

11.1.5.3 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, affecting the island determination (this can be checked during a run by setting `interactive=True`). Setting `rms_map=False` and `mean_map='const'` or `'zero'` will force PyBDSM to use a constant mean and rms value across the whole image. Additionally, setting `atrous_do=True` will fit Gaussians of various scales to the residual image to recover extended emission missed in the standard fitting. Depending on the source structure, the `thresh_isl` and `thresh_pix` 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 27.

11.1.6 Usage in Python scripts

PyBDSM may also be used non-interactively in Python scripts (for example, to automate source detection in a large number of images for which the optimal analysis parameters are known). To use PyBDSM in a Python script, import it by calling `from lofar import bdsm` inside your script. Processing may then be done using `bdsm.process_image()` as follows:

```
img = bdsm.process_image(filename, <args>)
```

where `filename` is the name of the image (in FITS or CASA format) or PyBDSM parameter save file and `<args>` is a comma-separated list of arguments defined as in the interactive environment (e.g., `beam = (0.033, 0.033, 0.0), rms_map=False`). 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). The same tasks used in the interactive PyBDSM shell are available for examining the fit and writing out the source list, residual image, etc. These tasks are methods of the Image object returned by `bdsm.process_image()` and are described below:

`img.show_fit()` 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.export_image()` Write an internally derived image (e.g., the model image) to a FITS file.

`img.write_catalog()` This method writes the Gaussian or source list to a file.

The input parameters to each of these tasks are the same as those available in the interactive shell. See the PyBDSM documentation (`http://tinyurl.com/PyBDSM-doc`) for more details and scripting examples.

11.2 Sky model manipulation: LSMTool

11.2.1 Introduction

LSMTool is a Python package which allows for the manipulation of sky models in the `makesourcedb` format (used by BBS). Such models include those output by PyBDSM, those made by `gsm.py`, and CASA clean-component models (after running `casapy2bbs.py`).

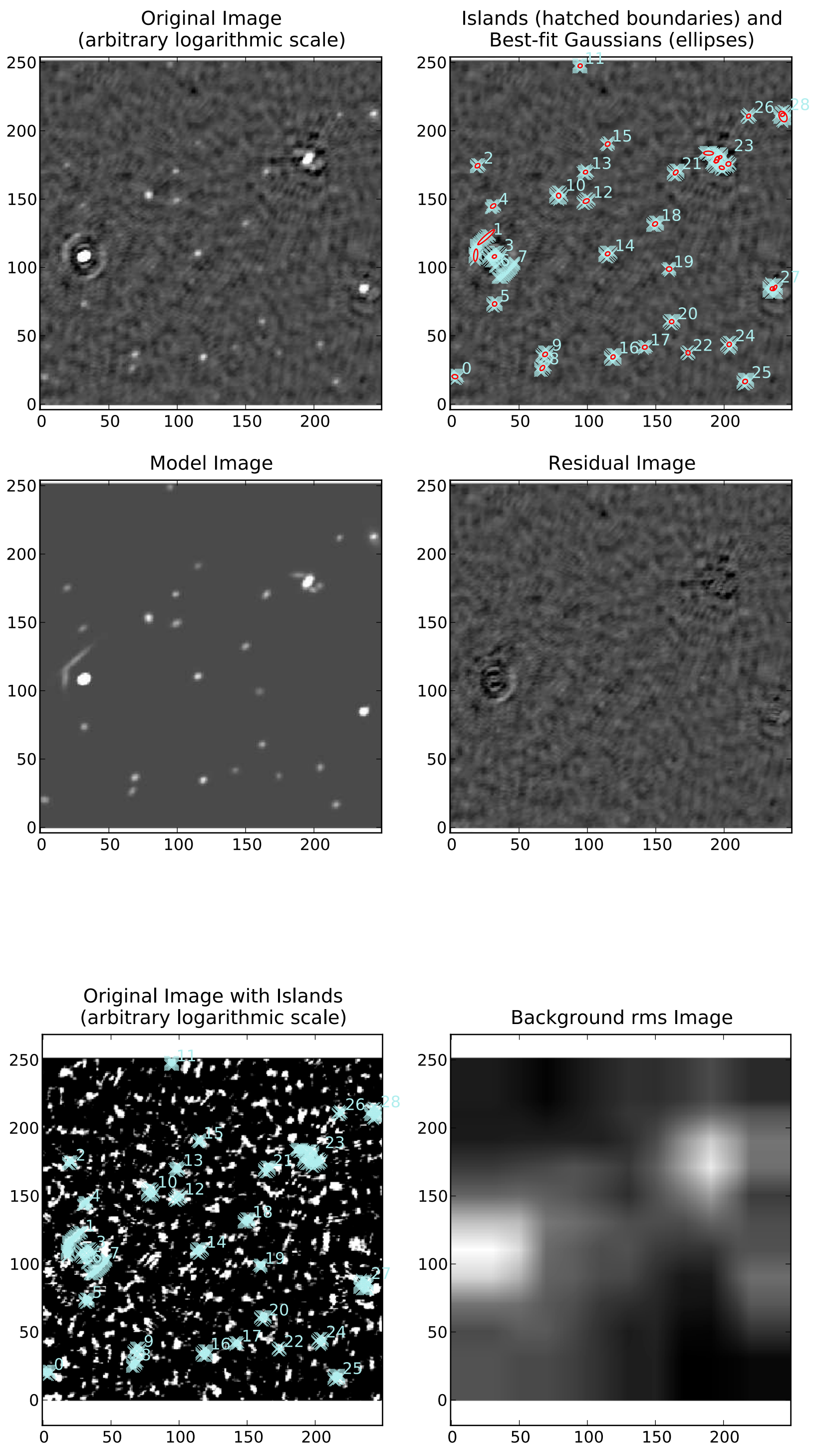


Figure 25: Example fit with default parameters of an image with strong artifacts around bright sources. A number of artifacts near the bright sources are picked up as sources. The background rms map for the same region (produced using `show_fit`) is shown in the lower panel: the rms varies fairly slowly across the image, whereas ideally it would increase more strongly near the bright sources (reflecting the increased rms in those regions due to the artifacts).

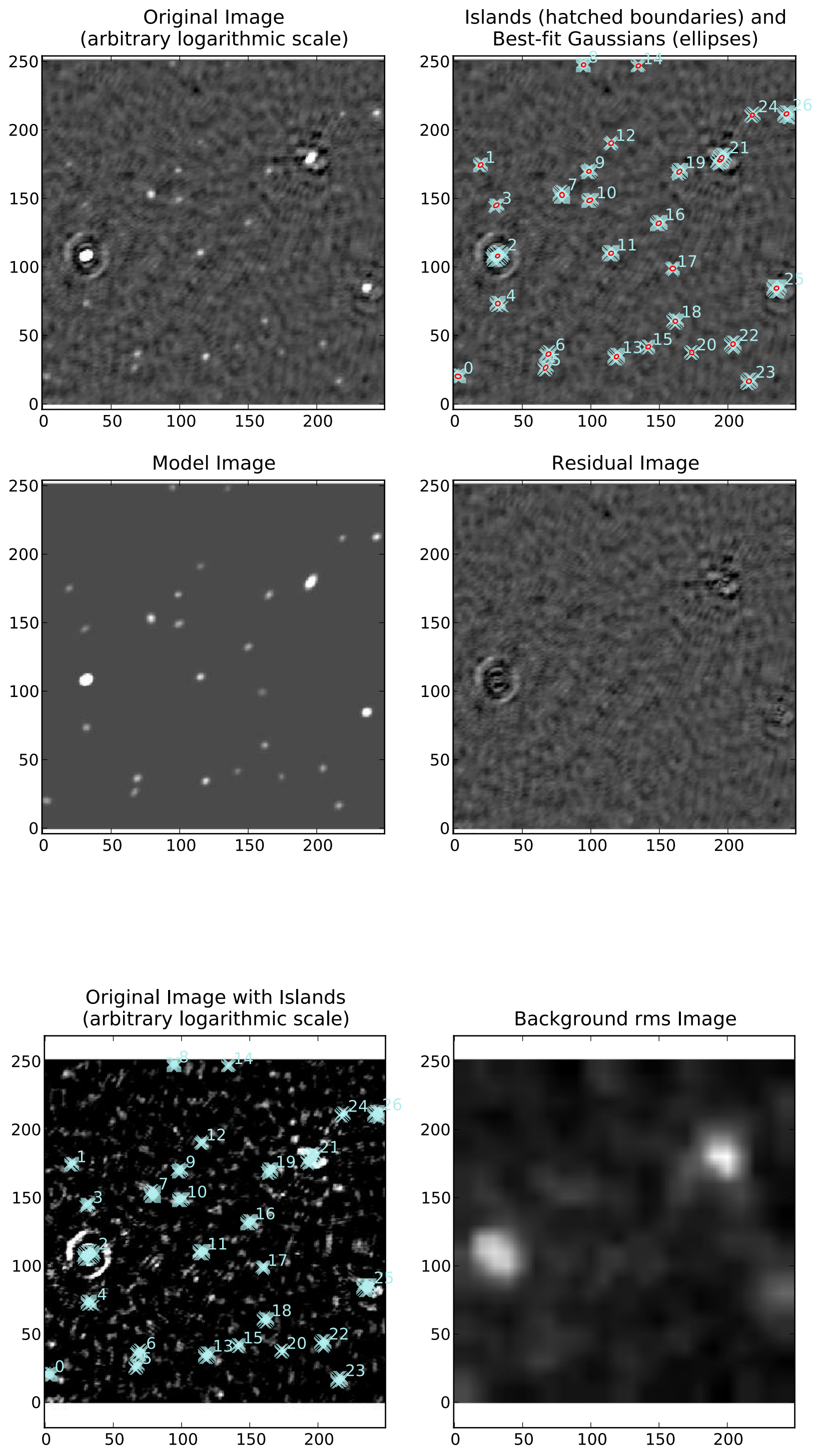


Figure 26: The same as Figure 25, but with `rms_box=(20,10)`. The rms map now varies on scales similar to that of the regions affected by the artifacts, and both bright and faint sources are recovered properly.

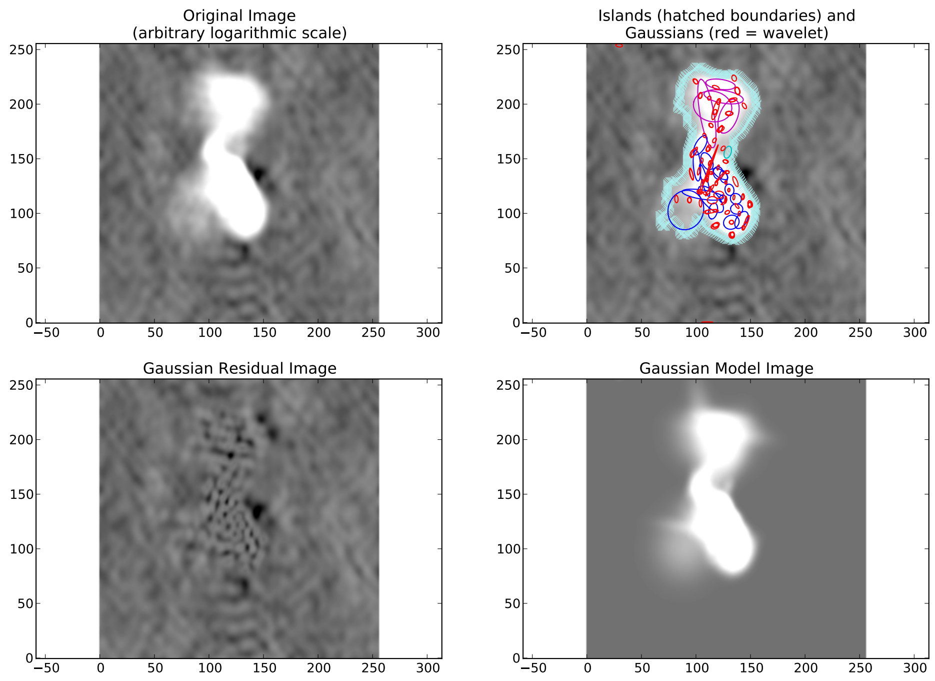

Original Image
(arbitrary logarithmic scale)
Islands (hatched boundaries) and
Gaussians (red = wavelet)
Gaussian Residual Image
Gaussian Model Image

Figure 27: Example fit of an image of Hydra A with `rms_map=False`, `mean_map='zero'`, and `atrous_do=True`. The values of `thresh_isl` and `thresh_pix` were adjusted before fitting (by setting `interactive=True`) to obtain an island that enclosed all significant emission.

11.2.2 Setup

To initialize your environment for LSMTool, users on CEP2 and CEP3 should run the following commands:

```
use LofIm
source ~rafferty/init_lsmtool
```

11.3 Basic Usage

The command-line version of LSMTool can be run as follows:

```
Usage: lsmtool <skymodel> <parset> [<beam MS>]
Options:
  --version   show program's version number and exit
  -h, --help  show this help message and exit
  -q          Quiet
  -v          Verbose
```

The parset specifies the operations to perform and their parameters. These are described in the next sections.

11.4 Operations

These are the operations that LSMTool can perform:

SELECT : Select sources by source or patch properties

REMOVE : Remove sources by source or patch properties

TRANSFER : Transfer a patch scheme from one sky model to another

GROUP : Group sources into patches

UNGROUP : Remove patches

MOVE : Move a source or patch position

MERGE : Merge two or more patches into one

CONCATENATE : Concatenate two sky models

ADD : Add a source

SETPATCHPOSITIONS : Calculate and set patch positions

PLOT : Plot a simple representation of the sky model

COMPARE : Compare source fluxes and positions of two sky models

11.5 The Parset File

As with NDPPP and LoSoTo, LSMTool can be run with a parset that specifies the operations to perform and their parameters.

Below is an example parset that filters on the flux, adds a source, and then groups the sources into patches:

```
LSMTool.Steps = [selectbright, addsrc, grp, setpos]

# Select only sources above 1 mJy
LSMTool.Steps.selectbright.Operation = SELECT
LSMTool.Steps.selectbright.FilterExpression = I > 1.0 mJy

# Add a source
LSMTool.Steps.addsrc.Operation = ADD
LSMTool.Steps.addsrc.Name = new_source
LSMTool.Steps.addsrc.Type = POINT
LSMTool.Steps.addsrc.Ra = 277.4232
LSMTool.Steps.addsrc.Dec = 48.3689
LSMTool.Steps.addsrc.I = 0.69

# Group using tessellation to a target flux of 50 Jy
LSMTool.Steps.grp.Operation = GROUP
LSMTool.Steps.grp.Algorithm = tessellate
LSMTool.Steps.grp.TargetFlux = 50.0 Jy
LSMTool.Steps.grp.Method = mid

# Set the patch positions to their midpoint and write final skymodel
LSMTool.Steps.setpos.Operation = SETPATCHPOSITIONS
LSMTool.Steps.setpos.Method = mid
LSMTool.Steps.setpos.OutFile = grouped.sky
```

In the first line of this parset the step names are defined. Steps are applied sequentially, in the same order defined in the list of steps. A list of step-specific parameters is given in Table 6.

Var Name	Format	Example	Comment
Operation	string	SELECT	An operation among those defined in Sec. 11.4
OutFile	string	out_sky_model.sky	Name of output file
SELECT and REMOVE			
FilterExpression	string	I > 10.0 Jy	Filter for selection
Aggregate	bool	False	Filter by aggregated patch property
ApplyBeam	bool	True	If true, apparent fluxes will be used
TRANSFER			
PatchFile	string	sky_model_with_patches.sky	File with patches that will be transferred
GROUP			
Algorithm	string	tessellate	One of tessellate, cluster, single, every, or a CASA mask filename
TargetFlux	string	10.0 Jy	Target total flux of patches (tessellate only)
NumClusters	int	100	Number of clusters (cluster only)
Threshold	float	0.1	Island threshold from 0 to 1 (threshold only)
FWHM	string	60 arcsec	FWHM of Gaussian for smoothing before thresholding (threshold only)
ApplyBeam	bool	True	If true, apparent fluxes will be used
UNGROUP			
MOVE			
Name	string	src1	Name of source or patch to move.
Position	list of floats	[12.3, 23.4]	RA and Dec in degrees to move to
Shift	list of floats	[0.001, 0.0]	RA and Dec in degrees to shift by
MERGE			
Patches	list of strings	[bin1, bin2, bin3]	Patch names to merge
Name	string	merged_patch	Name of new merged patch
SETPATCHPOSITIONS			
Method	string	mid	Set patch positions to mid, mean, or wmean positions
CONCATENATE			
Skymodel2	string	in_sky_model2.sky	Name of second sky model to concatenate
MatchBy	string	position	Identify duplicates by position or name
Radius	string	30 arcsec	Radius within which matches are identified
Keep	string	all	If two sources match, keep: all, from1, or from2
InheritPatches	bool	False	Matches inherit patches from parent sky model
ADD			
Name	string	src1	Name of source; required
Type	string	POINT	Type; required
Patch	string	new_patch	Patch name; required if sky model has patches
RA	float or string	12:45:30.4	RA; required
Dec	float or string	+76.45.02.48	Dec; required
I	float	0.69	Flux in Jy; required
AnyValidColumnName		value	Any valid column name can be specified
PLOT			
LabelBy	string	patch	Label points by: source or patch
COMPARE			
OutDir	string	comparison_plots/	Output directory for plots
SkyModel2	string	in_sky_model2.sky	Name of second sky model
Radius	string	10 arcsec	Radius within which matches are identified
LabelBy	string	patch	Label plot points by source or patch
ExcludeMultiple	bool	True	Exclude sources with multiple matches
IgnoreSpec	float	-0.7	Ignore any source in SkyModel2 with this spectral index

Table 6: Definition of variables in the LSMTool parset.

11.6 Interactive use and scripting

LSMTool can also be used interactively (in IPython, for example) or in Python scripts without the need for a parset. To use LSMTool in a Python script or interpreter, import it as follows:

```
>>> import lsmtool
```

A sky model can then be loaded with, e.g.:

```
>>> LSM = lsmtool.load('skymodel.sky')
```

All of the operations described in Section 11.4 are available as methods of the resulting sky model object (with the same name as the corresponding operation). For example, the following commands which duplicate the steps done in the example parset given in Section 11.5:

```
>>> LSM.select('I > 1.0 mJy')
>>> LSM.add({'Name':'new_source', 'Type':'POINT', 'Ra':277.4232, 'Dec':48.3689,
            'I':0.69})
>>> LSM.group(algorithm='tesselate', targetFlux='10.0 Jy')
>>> LSM.setPatchPositions(method='mid')
```

In many cases, the methods accept parameters with the same names as those used in a parset (see the source-code documentation for details). The sky model can then written to a new file with:

```
>>> LSM.write('grouped.sky')
```

Additionally, sky models can be written out as ds9 region files and kvis annotation files (as well as all the formats supported by the astropy.table package, such at VOTable, HDF5, and FITS):

```
>>> LSM.write('outskymodel.reg', format='ds9')
>>> LSM.write('outskymodel.ann', format='kvis')
>>> LSM.write('outskymodel.fits', format='fits')
>>> LSM.write('outskymodel.hdf5', format='hdf5')
>>> LSM.write('outskymodel.vo', format='votable')
```

In addition to the operations described above, a number of other methods are available:

LSM.copy() : Return a copy of the sky model object

LSM.info() : Print information about the sky model

LSM.more() : Print the sky model to the screen, using more-like controls

LSM.broadcast() : Send the sky model to other applications using SAMP

LSM.getColNames() : Returns a list of the column names in the sky model

LSM.getColValues() : Returns a numpy array of column values

LSM.getRowIndex() : Returns the row index or indices for a source or patch

LSM.getRowValues() : Returns a table or row for a source or patch

LSM.getPatchPositions() : Returns patch RA and Dec values

LSM.getDefaultValues() : Returns column default values

LSM.getPatchSizes() : Returns an array of patch sizes

LSM.getDistance() : Returns an array of angular distances to given position

LSM.setColValues() : Sets column values

LSM.setRowValues() : Sets row values

LSM.setDefaultValues() : Sets default column values

For details on these methods, please see the source-code documentation.

12 Automated Self-Calibration[73]

The standard technique of self-calibration (iterative updates of both sky model and direction-independent instrumental gains) is available in the form of a python script that automatically utilizes `BBS`, `DPPP`, `awimager`, and `PyBDSM` in a pre-defined loop algorithm. Further development of the python script into an operational form that can be run by the Radio Observatory is underway. It is assumed that before attempting to perform self-calibration, the LOFAR data set has been flux calibrated (e.g. by the Radio Observatory using the Standard Imaging Pipeline).

This section provides a brief guide to using `selfcal.py` on a LOFAR dataset, and details the internal procedures. The script is written to be completed without user interaction and in a consistent manner; therefore, there are very few options that can be set at runtime. Additional details are available at:

http://www.lofar.org/operations/doku.php?id=commissioning:selfcal

12.1 Overview

The goal of the self-calibration (selfcal) process is to improve the quality of the final image, through an iterative process. At each iteration, a calibration step is performed[74] to update the instrumental gain table. An image of the full field of view is then created at a particular angular resolution, and the source finder is used to update the sky model that is then used for the next calibration round. In early iterations, the data are imaged at low angular resolution. In each iteration, the image resolution is improved such that successive models have an increasing level of detail, and the calibration of the remote stations progressively improves. The iterative process is continued until the image resolution obtained is the best resolution possible with the data (taking into account the observing frequency and longest available baseline). In figure 29, the highlighted segments of the pipeline correspond to the ones used within the selfcal process (including the loop itself).

12.2 Availability

Work is in progress to implement the selfcal procedure within the Standard Imaging Pipeline. At present, the stand-alone selfcal version is available on CEP2, CEP3 and on Flits (ASTRON Astronomy Group cluster). On CEP2 and CEP3, the stand-alone `selfcal.py` is included in the daily build. On Flits, the most recent version of the stand-alone `selfcal.py` is maintained by N. Vilchez.

To utilize the stand-alone `selfcal.py` at your own facility, the LOFAR trunk must be installed (`selfcal.py` is included in the LOFAR trunk available through `svn`, not in releases yet). To obtain the LOFAR trunk software, LOFAR `svn` access is required. Installation details are available at:

http://www.lofar.org/wiki/doku.php?id=engineering:software:lofar-cmake

[73]This section is maintained by Nicolas Vilchez (`vilchez[at]astron[dot]nl`) and George Heald (`heald[at]astron[dot]nl`).

[74]In the current version, the automatic script performs phase calibration only.

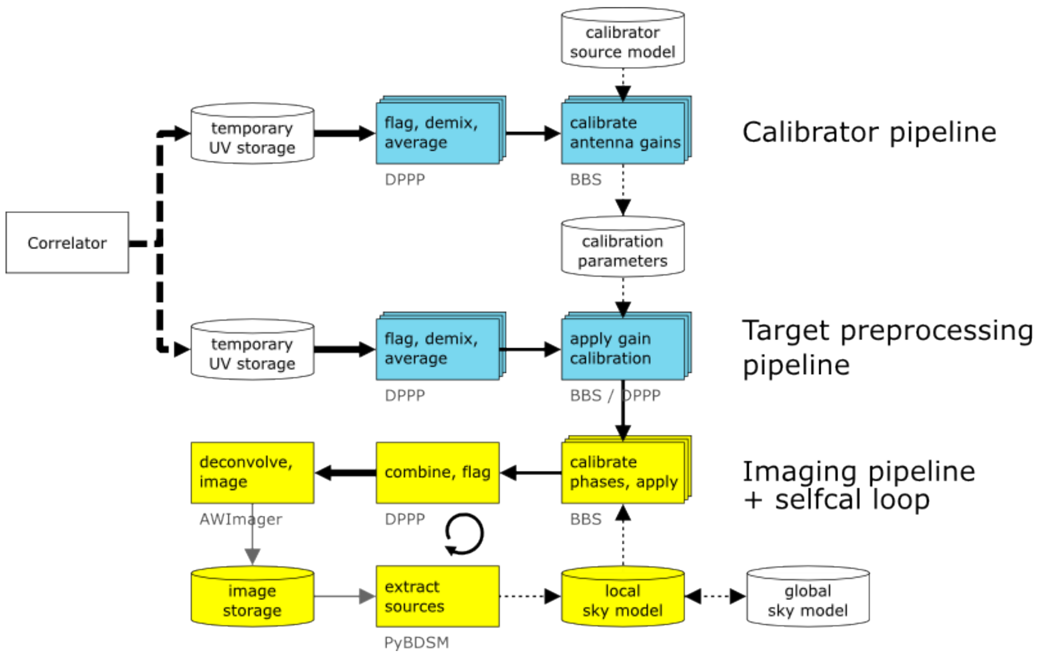


Figure 28: Illustration of the selfcal process in the Standard Imaging Pipeline.

12.3 Selfcal: the stand-alone version

12.3.1 Usage

On CEP2 and CEP3 the latest version of `selfcal.py` is installed, but you need to setup your environment by typing[75]:

```
> use LofIm
```

in order to make use of the daily build from the LOFAR trunk. Alternatively, you may add `LofIm` to the `.mypackages` file in your home directory.

Note that the `LofIm` package conflicts with the `LUS` package (especially for `awimager`). To use `selfcal.py`, you must disable `LUS` from your environment (i.e. erase `LUS` from the `.mypackages` file in your `$HOME` directory).

The self-calibration procedure is initiated by typing `selfcal.py` at the command prompt. To access usage information, type `selfcal.py -h` and the following will appear:

```
Usage:  selfcal.py PATH/parsetFile
OR
Usage:  selfcal.py --obsDir= --outputDir= [Options:  --skyModel=
--nbCycle= --outerfovclean= --VLSSuse= --annulusRadius= --startResolution=
--endResolution= --resolutionVector= --mask= --UVmin= --startingFactor= --FOV=
--nofPixelPerBeam= --robust= --maskDilation=]
```

and a detailed list of parameters.

Only “obsDir” and “outputDir” (which specify the input and the output directories) are required for a successful `selfcal.py` run. All other parameters have sensible default values as determined through

[75]On Flits, `selfcal.py` is directly installed, no use command is required.

commissioning.

The `selfcal.py` parameters are:

- **Required Parameters:**

  1. `obsDir` (string): This is the observation directory which contains data (or frequency merged data generated by `mergeSB.py`; see § 12.3.2)
     Data must contain the same number of subbands, have the same frequency range and the same pointing centre. Checks are done internally. The `obsDir` must contain ONLY data, no additional files. Due to this, if the `outputDir` is in the `obsDir` tree, the code will crash. The full path must be provided.
  2. `outputDir` (string): This is the output directory where the selfcal process will be executed. If it doesn't exist, it will be created automatically.
     The full path must be provided.

- **Optional Parameters:**

  1. `skyModel` (string, default:'none'): This option allows the user to provide a skymodel instead of using the `VLSSuse` option. The skymodel must be BBS compatible. The full path of the file must be provided.
     NB: If the `skyModel` option is used, the `VLSSuse` parameter must be not set.
     (a) Case 1: `VLSSuse=yes`: GSM sky model (VLSS) is used for initial calibration
     (b) Case 2: `VLSSuse=no`: low resolution (90 arcsec) skymodel is used for initial calibration (generated with first-pass image from the input data)
     (c) Case 3: `skyModel` is specified (and `VLSSuse=no`): your own skymodel is used for initial calibration
  2. `nbCycle` (int, default:10): This is the number of selfcal cycles to perform. It must be between 5 and 20. The selfcal process will start at 15 times the best possible resolution and decrease the resolution at each cycle in linear steps.
  3. `outerfovclean` (string, default:'no'): accepts 'yes' or 'no'. If this option is activated, a preprocessing cleaning to subtract sources from outside the field of view will be applied before the selfcal process.
     A first low resolution image will be generated (using core stations only) to obtain a sky model. This preliminary image has 30arcsec pixel size and a larger field of view (fov=`FOV`+2×`annulusRadius` degrees). A skymodel is extracted and separated into an 'annulus' skymodel (containing sources with angular distance >`FOV`/2 deg from the target) and a 'centre' skymodel (containing sources with angular distance <`FOV`/2 deg from the target). The 'annulus' skymodel is then subtracted from the visibilities. This cleaning is an iterative process and will continue until there are less than 10% of sources from the original annulus skymodel (typically 2-3 iterations are needed).
  4. `VLSSuse` (string, default:yes): accepts 'yes' or 'no': Use the VLSS-based skymodel for the first iteration of selfcal (as generated by `gsm.py`).
     If the user does not want to use VLSS skymodel (`VLSSuse=no`), the code will create a preliminary image (see the `outerfovclean` parameter described above), and use the 'center' skymodel as the input skymodel.
  5. `annulusRadius` (float, default:1): In degrees; if `outerfovclean=yes`, you can select the radius of the outer annulus from which sources will be subtracted. By default, the value is 1 deg, i.e. the field of view will be $5+2\times1=7$ deg, and all sources which are less than 1 degree of the edge of the field of view will be subtracted.

6. `startResolution` (float, default:15$\times$ the best resolution available from the data): The resolution to use in the first selfcal cycle. If not set, the starting resolution is automatically computed in arcseconds, using
   Starting Resolution $= 15 \times \lambda/\max(\text{baseline})$.
   NB: the `startResolution` option conflicts with `resolutionVector` and `startingFactor` parameters.
   NB: if `startResolution` is provided, `endResolution` must also be provided.
7. `endResolution` (float, default:best resolution available in the data): The resolution to use in the final selfcal cycle. If not set, the best resolution available is calculated using:
   Best Resolution $= \lambda/\max(\text{baseline})$
   NB: the `endResolution` option conflicts with `resolutionVector` parameter, but not with `startingFactor`
   NB: if `endResolution` is provided, `startResolution` does not need to be provided.
8. `resolutionVector` (list of float: e.g. `resolutionVector=x1, x2, x3` (do not use parentheses), default:[none]): Select the resolution for each cycle. Format: list of float in arcseconds.
   NB: the `resolutionVector` option conflicts with `startResolution`, `endResolution` and `startingFactor` parameters.
9. `mask` (string, default:'yes'): accepts 'yes' or 'no': Use clean masks when imaging.
10. `maskDilation` (integer, default:0): If mask is used (generated a from PyBDSM-extracted sky model), it is possible to dilate it. 0 means no dilation, 1 means extend the mask regions by 1 pixel in all directions, etc.
    For more details see the PyBDSM `export_image` documentation.
11. `UVmin` (float, default:0 or 0.1 in klambda): Set the `awimager` parameter `UVmin` for each imaging run. If unset, `UVmin=0.1` will be used for observations with declinations below 35 degrees, otherwise `UVmin=0`. It is recommended to use `UVmin=0` if the user is interested in imaging extended or diffuse emission.
12. `startingFactor` (int, default:15): The resolution of the initial selfcal cycle is set by `startingFactor` $\times$ Best Resolution, or `startingFactor` $\times$ `endResolution`.
    NB: the `startingFactor` option conflicts with `startResolution` and `resolutionVector` parameters, but not with `endResolution`
13. `FOV` (float, default:5): Select the image size (in degrees).
14. `nofPixelPerBeam` (float, default:4): Number of pixels per synthesized beam.
15. `robust` (float, default:none): If the `robust` parameter is specified, it will be kept constant for all selfcal cycles. If unset, the robust parameter varies iteratively from 1 to -2. Must be in the interval: [2;-2]

- **Future Parameters:**

  1. `catalogType` (string, default:'none'): accepts 'GSM' or 'PYBDSM': `catalogType` is used together with the `peeling` and `DDC` options. It is required to be able to select in the initial catalog the brightest source (or sources) to peel.
  2. `peeling` (string, default:'no'): accepts 'yes' or 'no': If `peeling=yes`, the brightest source (or sources) in the initial catalog will be peeled.
     NB: the `peeling` and `DDC` options conflict. Do not use both at the same time.
  3. `DDC` (string, default:'no'): accepts 'yes' or 'no': Same as `peeling` option, except that the subtraction is not applied, just the Direction Dependent Calibration.

4. `nofDirections` (int, default:1): Number of directions (i.e. brightest sources) required to peel or to apply the Direction Dependent Calibration.

The output directory will be automatically created if it does not exist. If the output directory already exists, it is advisable to delete the output of any previous runs to avoid conflicts. Note that if the selfcal process crashes, it is not possible to resume from the point where the crash occurred; a new run (starting from scratch) would be required.

12.3.2 Required Data Format

To process visibility data with the `selfcal.py` script, flux-calibrated data must be written in the `CORRECTED_DATA` column of your Measurement Sets. Selfcal currently only performs phase calibration. Flux calibration is assumed to have been completed (e.g. by the Observatory using the Standard Imaging Pipeline) before running `selfcal.py`; additional amplitude calibration is not performed (yet) by the self-calibration procedure.

The self-calibration routine assumes that the input visibility data have been grouped in sets of contiguous subbands, where the sets typically are made of 10-20 subbands. To facilitate the grouping of subbands and to ensure that the input visibility data are formatted in the way expected by `selfcal.py`, a helper function is provided and should be used before running self-calibration.

The script, called `mergeSB.py`, accepts intermediate or final visibility (MS) data from the Radio Observatory pipeline. These datasets are named following a specific convention. Therefore the names of the MS visibility datasets follow a specific pattern:

- Intermediate data: `L123456_SAP123_SB123_uv.MS.dppp`
- Final data: `L123456_SB123_uv.dppp.MS`

These visibility datasets are equivalent and may be used interchangeably for `mergeSB.py` and `selfcal.py`.

If the names of your visibility datasets are different, but you still feel that they are prepared properly for use in the self-calibration routine, then you will have to manually change the name of the Measurement Sets to conform with one of the two naming conventions provided above. In addition, please be aware that the Measurement Sets will be grouped alphabetically (using the `ls` order); therefore be sure that the subband numbers are given as `001`, `002`, etc.

For the input to `selfcal.py`, the names of the frequency merged Measurement Sets do not need to follow any convention (except that the chronological order of a set of snapshots must somehow be reflected in the alphabetical order of the MS names). The procedures in `selfcal.py` will always work on the `CORRECTED_DATA` column, which is created by `mergeSB.py`. To avoid confusion it is highly recommended to use the two scripts together, unless you want to work on only one subband.

To combine subbands, type `mergeSB.py` at the command prompt. If you provide no arguments (or type `mergeSB.py -h`) then the required parameters are listed:

```
MISSING Parameters
Usage:  mergeSB.py --obsDir= --outputDir= --column2Merge= Optional:
--nofChannelPerSubband= --checkNames]
```

If `nofChannel2Merge` is not provided, the default behavior is to merge the data to a frequency resolution corresponding to 1 channel per subband. The default value for `checkNames` is yes.

- **Required Parameters:**

    1. `obsDir` (string): This parameter specifies a directory containing only a set of single-subband Measurement Sets (MS) that should be merged (per time chunk). This directory should contain the Measurement Sets for all available time chunks, and all subbands per time chunk, that the user wants to merge. For example if you want to use `selfcal.py` on a dataset that consists of 10 time chunks and one group of 5 contiguous subbands, then the `obsDir` must contain $10 \times 5 = 50$ MSs. It is often the case that one or more Measurement Sets are missing. This is acceptable, because `DPPP` (which is used to merge the subbands) will fill missing frequency values with flagged data, corresponding to missing Measurement Sets. This is true as long as the first and the last subband are present for all time chunks.
    The path given for `--obsDir` must be a full path.
    2. `outputDir` (string): This parameter specifies the output directory for merged Measurement Sets. In this directory, two subdirectories will be created. The first one, named `Merged-Data`, will contain the merged time chunks. This subdirectory should be used afterwards as the `--obsDir` parameter for `selfcal.py`. The second subdirectory, named `DPPP_Parset`, will contain the parsets that are used for merging the subbands. The user can inspect these parsets to find out how the subbands have been merged by `mergeSB.py`.
    The path given for `--outputDir` must be a full path.
    3. `column2Merge` (string): This is the name of the column to merge. Usually `CORRECTED_DATA` is used (`awimager` will only use the `CORRECTED_DATA` column).
    Usually, this should be set to either `DATA` or `CORRECTED_DATA`.

- **Optional parameters:**

    1. `nofChannelPerSubband` (int, default:1): This is the number of channels per subband required after the subband concatenation. By default, `mergeSB.py` averages to 1 channel per subband.
    2. `checkNames` (string, default:'yes'): [OPTION NOT ACTIVATED AT THE MOMENT] `mergeSB.py` checks the names of the MSs (to verify that they are either intermediate or final pipeline data products, see documentation). If `checkNames` is set to 'no', a warning appears if the input MS names do not respect the expected nomenclature, but `mergeSB.py` will continue without exiting.

12.3.3 Selfcal implementation details

The self-calibration process consists of a set of individual tasks. These are run in a sequence which is looped, and at each cycle (iteration) the image resolution is increased to improve the sky model, allowing a better phase calibration to be performed at the next step.

The individual tasks that are combined within a single iteration are:

1. Calibration with a skymodel using `calibrate-stand-alone` (BBS)
2. Flagging using `DPPP`
3. Imaging using `awimager`
4. Sky model extraction using the `PyBDSM` source finder.[76]

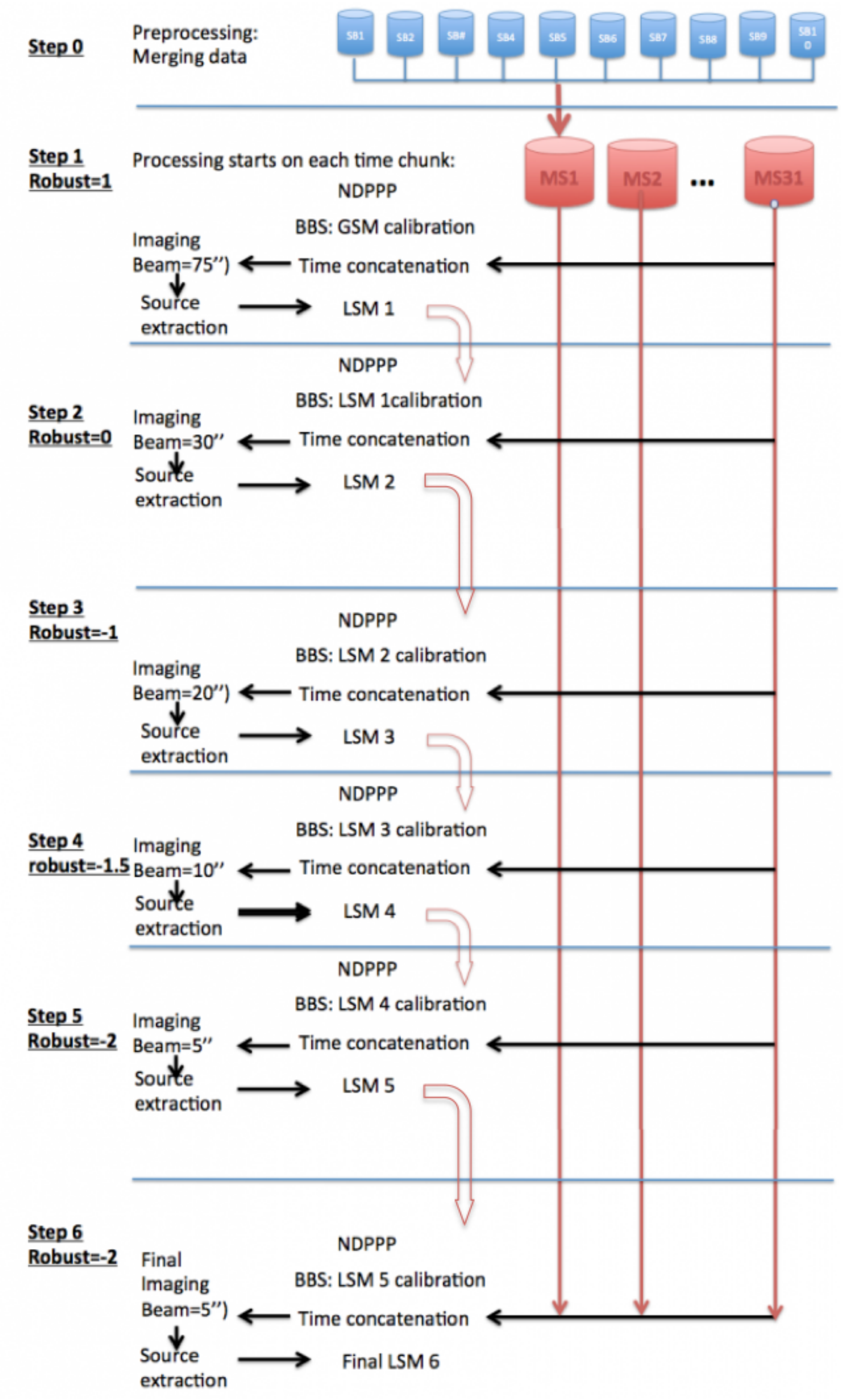


Figure 29: Illustration of the iterations performed by the selfcal process.

The diagram in Figure 29 shows an example of the self-calibration process. Within a cycle, each time chunk is calibrated separately using `BBS`, then each of them is flagged using `DPPP`. It is necessary to calibrate and flag them separately because it is not possible (at the moment) to calibrate non-continuous time-concatenated MSs using BBS. After this step (which is parallelized on 8 cores), the MSs are concatenated in time using the `msconcat` command from the `pyrap.tables` python module.

[76]The PyBDSM manual is available at `http://tinyurl.com/PyBDSM-doc` (html version) or `ftp://ftp.strw.leidenuniv.nl/pub/rafferty/PyBDSM/PyBDSM-1.8_manual.pdf` (pdf version).

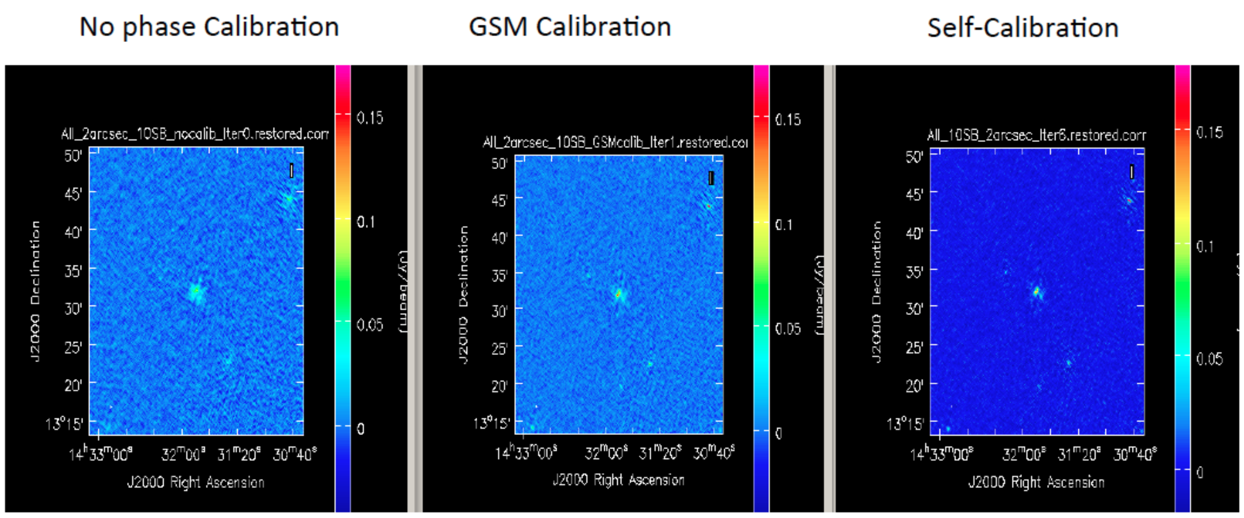


Figure 30: Results of selfcal process with an example HBA dataset at approximately 140 MHz. There are clear improvements in the image noise after performing self-calibration. The noise levels in the three images are 30, 10, and 2 mJy/beam for the cases of: no phase calibration (*left panel*), standard GSM-based phase calibration (*middle*), and self-calibration (*right panel*). The thermal noise is estimated to be approximately 0.7 mJy/beam for this dataset.

Next, `awimager` images the time concatenated MS with specific parameters. To finish, a sky model is extracted using `pybdsm` on the newly created image.

The algorithm is designed to improve the angular resolution in each cycle. This means that `selfcal` must include longer baselines at each iteration. Moreover, the robust parameter is also changed in each iteration to give additional weight to longer baselines. The default behavior is to start at `robust=1` (nearly natural weighting) at low resolution (15 times the best resolution) and end at `robust=-2` (uniform weighting) when including all available baselines. As described in § 12.3, the progression of imaging parameters from one iteration to the next can be modified by the user.

12.3.4 Selfcal examples

In Figure 30, an example of the self-calibration process is shown for HBA-low. The example dataset had the following properties:

- 31 time chunks
- 10 contiguous subbands
- Best beam resolution $= 5''$

An example of using self-calibration on an example LBA dataset is shown in Figure 31. This LBA dataset contains 10 subbands and the total duration of the observation was 4.5 h. Ionospheric effects become very strong on baselines longer than about 20 km (in this particular example), so the direction independent strategy does not improve the data quality for angular resolution less than about $50''$ in this particular case. If this example is representative, then `selfcal` can be expected to improve calibration on baselines out to approximately 20 km, depending on ionospheric conditions at the time. Further improvements will require direction dependent calibration. At present, with both HBA and LBA datasets, the selfcal process typically manages to obtain a final image noise about 2-3 times the theoretical thermal noise.

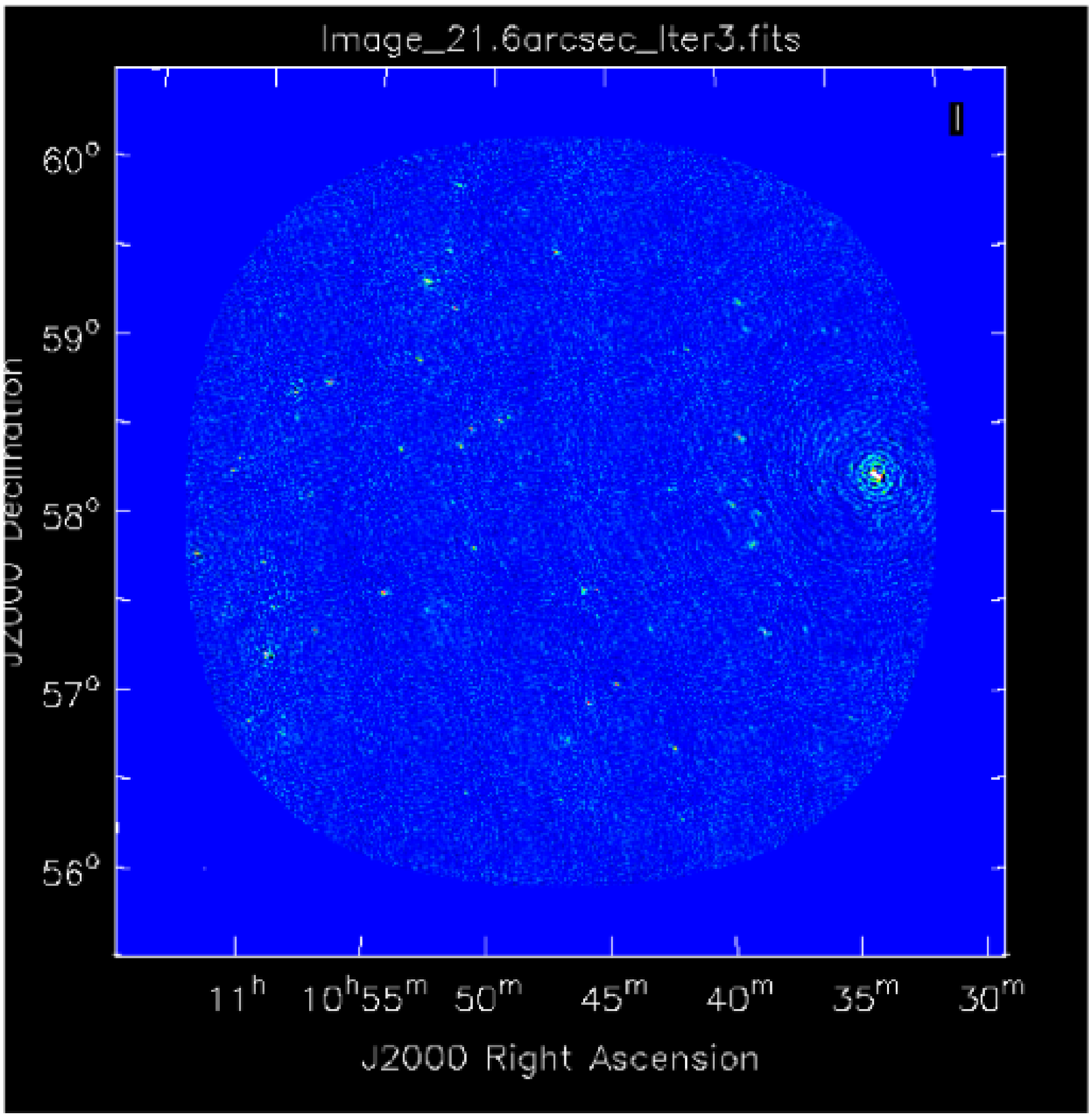


Figure 31: Results of selfcal process with LBA (∼60 MHz). The noise levels in the image is 22 mJy/beam. The thermal noise is estimated to be approximately 11 mJy/beam for this dataset.

13 Faraday Rotation Measure Synthesis[77]

Firstly described by Burn (1966) and then extended and implemented by Brentjens & de Bruyn (2005) Faraday Rotation Measure (RM)-synthesis is a powerful detection technique. for interpreting polarized emission data in order to separate the contributions of different sources lying on the same line of sight. In this chapter, we provide a brief overview of the principles and the available tools suitable for imaging multiple polarized structures along the line of sight.

13.1 Overview

Synchrotron emission radiated by relativistic electrons accelerated by magnetic fields is linearly polarized, the (intrinsic) plane of polarization being orthogonal to the component of the magnetic field in the plane of the sky. Moreover the polarization angle (χ) can be rotated when propagating through a magnetised plasma, the amount of rotation being a function of wavelength (λ). This effect is called Faraday rotation and the amount of rotation is described by the Faraday depth (ϕ) as:

$$\phi = 0.81 \int_{source}^{observer} n_e \vec{B} d\vec{s} \tag{13}$$

, with n_e being the electron density measured in cm^{-3}, $\vec{B}$ the magnetic field measured in μG and the path-length $\vec{s}$ is expressed in parsecs.

Synchrotron radiation can be probed by polarimetric radio observations. In the simplest case of emission and rotation regions being separated (and beam depolarization can be neglected), the amount of rotation of the polarisation plane and the ϕ are characterised by:

$$\chi = \chi_0 + \lambda^2 \phi \tag{14}$$

and the amount of rotation ϕ can be found as the slope of a polarization angle versus wavelength squared plot. In such a case the Faraday depth is equal to the rotation measure (*RM*) at all wavelengths.

However polarization angle often does not depend linearly on λ^2, thus making a linear fit inappropriate. RM-Synthesis is based on the fact that the complex polarized intensity (P) can be calculated as a Fourier transform of the Faraday dispersion function (F) in Faraday space with the coordinate ϕ which is the Faraday depth, i.e.:

$$P(\lambda) = \int_{-\infty}^{\infty} F(\phi) e^{2i\phi\lambda^2} d\phi \tag{15}$$

. Performing the inverse Fourier transform of P one obtains F which is the polarized intensity emerging from a region with Faraday depth ϕ.

Therefore RM-Synthesis is a Fourier transformation of Stokes U and Q maps, performing a derotation of Faraday rotated emission using a set of assumed ϕ values. Unfortunately, the inversion of equation 15 is limited in practice by the fact that P can be measured only for $\lambda^2 > 0$ and in only in a finite spectral band (Brown et al. 2009). The lower bound λ_{min} limits the detection of objects which are extended in Faraday space, while the upper bound λ_{max} suppresses the detection of small-scale structures of the object in Faraday space. Moreover the lack of negative λ^2 affects the reconstruction of the intrinsic polarization angle. Finally, a poor sampling within the observed bandwidth prevent the reconstruction of objects at large Faraday depths. As a result, the resolution in ϕ-space depends on λ^2 coverage and resolution.

Note that the reconstructed Faraday dispersion function is the convolution of the actual Faraday dispersion function with the "RM Spread Function" (RMSF), corresponding to a point spread function

[77]This chapter is maintained by M. Iacobelli, `iacobelli[at]astron[dot]nl`.

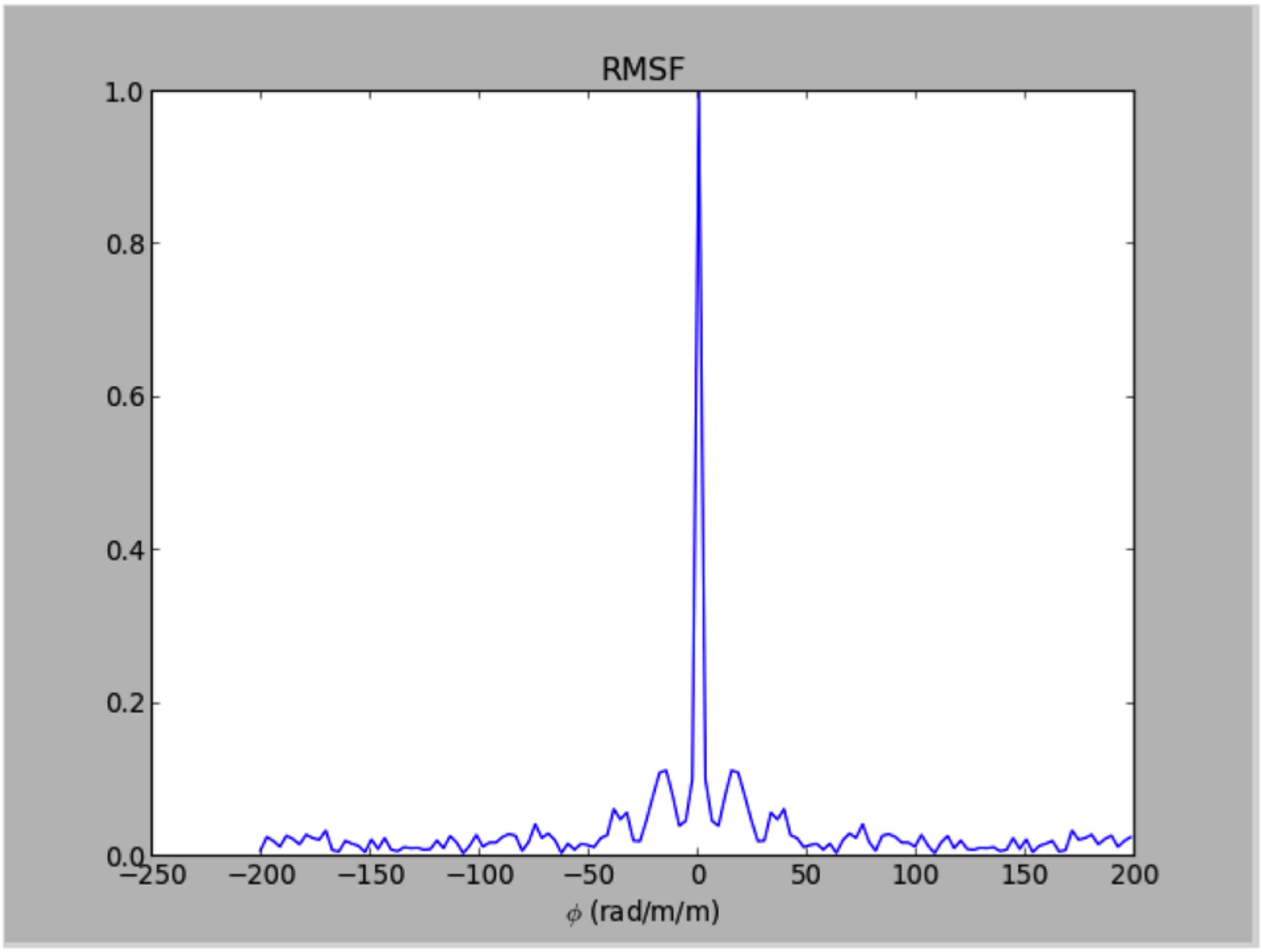


Figure 32: Rotation measure spread function (RMSF) of a test observation with smooth λ^2 coverage (bad channels were flagged). In this example the resolution in Faraday depth space is: $RMSF_{FWHM}$ = 3.00 rad m^{-2} and sidelobes are not prominent.

(PSF) in ϕ-space. Thus, the FWHM of the RMSF determines the precision with which one can determine the ϕ at the peak of a Faraday dispersion function (see e.g. Fig. 32). In case of a complex and/or poor reconstructed Faraday dispersion function, RMSF sidelobes can make the interpretation difficult thus requiring a deconvolution operation (Heald 2009).

To summarise, RM-Synthesis can disentangle emission and rotation layers. However in case of mixed emitting and Faraday-rotating region an incomplete λ^2 coverage will result in loss of sensitivity with poor or no detection. To be applied, RM-Synthesis requires wide-band polarimetry with the λ^2 coverage and resolution determining its basic parameters: resolution in ϕ, sensitivity to extended ϕ-structures, the highest ϕ value detectable. Inputs of RM-Synthesis are Stokes U and Q maps obtained from carefully calibrated Measurement Sets (MS). Main outputs of the RM-Synthesis are:

- the computed RMSF (users should check it carefully).
- data cubes mapping polarised intensity / polarisation angle as a function of spatial coordinates, the third dimension being the Faraday depth.

Therefore, RM-Synthesis frames show the distributions of polarized intensity / polarisation angle in the plane of the sky at various Faraday depths. The output cubes can be displayed by using a FITS viewer (e.g. `DS9,kvis`).

13.1.1 Tools

Currently, two different python implementation of RM-synthesis are available[78]:

- RMSynthesis (in CEP3 activate with `use RMSynthesis`). This Python script perform a basic (no RMclean) RM-synthesis, provided a FITS cube with Stokes Q images, a FITS cube with Stokes U images, and a text file containing the frequencies of these images. To check options just execute `rmsynthesis` in your terminal.

- Pyrmsynth (in CEP3 activate with `use RMSynthesis` and execute `rm_synthesis.py`. This script performs RM-synthesis, either simply by Fourier transformation (to produce a dirty image) or using the RMCLEAN method as described by Heald (2009). The software reads in a parameter file. The code works on sets of FITS files containing images from a single sub-band, or some other subset of the observed frequencies. As a default, the code assumes all Stokes parameters to be saved in one FITS file. There is an additional option that allows for the handling of separately save Q and U FITS files. User defined frequency weights can be included by providing a text file in which each line contains a weight to be applied to each frequency. The name of this file <u>must</u> be "weight.txt". Additionally output files are: a plot of the RMSF and a Stokes V cube. Also available is library `rm_tools` (`use import rm_tools` within a python script). To check options execute `rm_synthesis.py --help` in your terminal.

[78]They are installed on the CEP3 cluster (see Chapter 1) and can also be downloaded at RMSynthesis and Pyrmsynth

14 Sky Model Construction Using Shapelets[79]

In this chapter, we give a tutorial overview of sky model construction using shapelets and other source types suitable for self calibration. Note that shapelets decomposition should be used in the case we are dealing with extended sources.

14.1 Introduction

In this tutorial, we present construction of accurate and efficient sky models for calibration of LOFAR data, using shapelets. However, we do not present any theoretical material on shapelets and their strengths and weaknesses for use in self calibration. We refer the reader to Yatawatta (2010; 2011) for a more mathematical presentation on these subjects.

We always work with FITS images for our model construction. Therefore, it is assumed that you already have an image of the sky that is being observed, which is good enough to create a sky model from. You can obtain a FITS image of the sky that is being observed in many ways. For example, you can use images made by other instruments (at a probably different frequency/resolution) and one such source is Sky View[80]. You can also do a rough calibration of the data and make a preliminary image of the sky. And if you are hardcore, you can also manipulate an empty FITS file to create the shape that you want to model (we shall discuss this later).

The FITS file contains more information than that is shown as the image. Since we are dealing with images made with radio interferometers, almost all images have been deconvolved (e.g. by CLEAN). The Point Spread Function (PSF) plays an important role in deconvolution. Most FITS files have information about the approximate PSF that we will be using a lot. This information is stored in the header of the FITS file with the keywords `BMAJ`,`BMIN`, and `BPA`. The `BMAJ` and `BMIN` keywords give the PSF width as the major and minor axes of a Gaussian. The `BPA` keyword gives the position angle (or the rotation) of the Gaussian. We will learn how to manipulate these keywords (or add them if your FITS file is without them) later.

Throughout this tutorial, we will calibrate an observation of Virgo-A around 50 MHz. In Fig. 33, we have shown an image of Virgo-A made by the VLA at 74 MHz. The red circle on top right corner shows the PSF for this image. Although the frequency and resolution does not match the LOFAR observation, we will be using this image to build a sky model.

Looking closer at Fig. 33, we see that there is bright compact structure at the center and weak diffuse structure surrounding it. You should always keep in mind the golden rule in source modeling: A point source is best modeled by a point source and nothing else. Almost always, you will have images with both compact structure (best modeled by point sources) and extended structure (best modeled using shapelets). In our example, we need to model the central compact structure as point sources and the remainder as shapelets. See Yatawatta (2010) for a theoretical explanation.

14.2 Software Overview

There are several steps needed in building a good sky model. You can skip some steps depending on particular requirements (and if you can use other software to do the same). We give a general overview of various tools used in different stages of sky model construction. All the software is installed in `/opt/cep/sagecal/bin` in the CEP clusters.

[79]The author of this chapter is Sarod Yatawatta (yatawatta[at]astron[dot]nl).
[80]`http://skyview.gsfc.nasa.gov/cgi-bin/skvadvanced.pl`

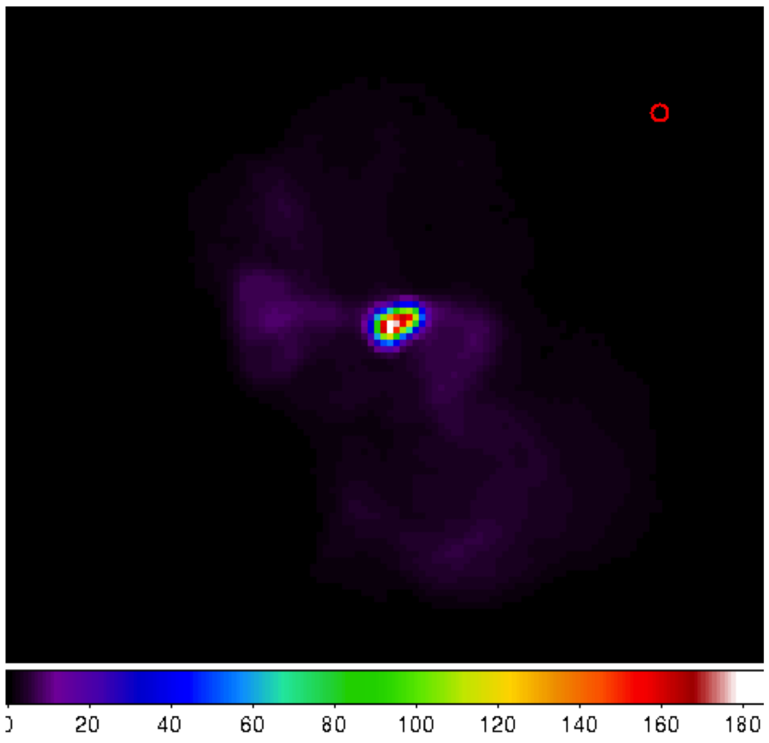


Figure 33: Virgo-A image made by the VLA at 74 MHz. The red circle on top right corner is the PSF.

14.2.1 modkey

The program `modkey` is used to modify keywords in FITS files. For example, if you want to modify the `BMAJ` keyword in the `example.fits` FITS file

```
modkey -f example.fits -k BMAJ -d 0.1
```

will set the value of `BMAJ` to `0.1`. If this key does not exist, it will be created. Try using

```
modkey -h
```

for more usage examples.

14.2.2 fitscopy

Most FITS files will be too large to work with. The sources that you want to model will be only in small areas of the large FITS file. The program `fitscopy` will create a smaller FITS file by selecting a smaller rectangle from the larger FITS file. For example, if you want to select the area given by the pixels $[x0,y0]$ bottom left hand corner and $[x1,y1]$ top right hand corner of the file `large.fits`

```
fitscopy large.fits small.fits x0 y0 x1 y1
```

will do the trick.

14.2.3 ds9 and kvis

We use both `ds9` and `kvis` to display FITS file as well as display regions (`ds9`) and annotations (`kvis`).

14.2.4 Duchamp

The source extraction program `Duchamp` is written by Matthew Whiting[81]. We will only be using `Duchamp` to create a mask file for a given FITS image. A mask is a FITS file with the same size as the original image, but with zeros everywhere except at the selected pixels. Here is a simple configuration file for creating a mask for `example.fits` FITS file

```
##########################################
imageFile example.fits
logFile         logfile.txt
outFile         results.txt
spectraFile     spectra.ps
minPix          5
flagATrous      0
snrRecon        10.
snrCut          5.
threshold 0.030
minChannels     3
flagBaseline    0
flagKarma 1
karmaFile duchamp.ann
flagnegative 0
flagMaps 0
flagOutputMask 1
flagMaskWithObjectNum 1
flagXOutput 0
############################################
```

The threshold for pixel selection is given by the `threshold` parameter which is 0.03 in the above example. After creating the configuration file, and saving it as `myconf.txt`, you can run `Duchamp` as

```
Duchamp -p myconf.txt
```

This will create a mask file called `example.MASK.fits` which we will be using at later stages.

NOTE: Only versions later than 1.1.9 produce the right output.

14.2.5 buildsky

We mentioned before that whenever we have compact structure, it is best modeled by using point sources. The program `buildsky` creates a model with only point sources for a given image. However, we must have a mask file. So if we have `example.fits` image and `example.MASK.fits` mask file, the simplest way of using this is

```
buildsky -f example.fits -m example.MASK.fits
```

This will create a file called `example.fits.sky.txt` that can be used as input for BBS. It also creates a ds9 region file called `example.fits.ds9.reg` that you can use to check your sky model.

You can see other options by typing

```
buildsky -h
```

[81]`http://www.atnf.csiro.au/people/Matthew.Whiting/Duchamp/`

14.2.6 restore

We use `restore` to restore a sky model onto a FITS file. The sky model can be specified in two different ways. It can directly read a BBS sky model like:

```
# Name, Type, Ra, Dec, I, Q, U, V, MajorAxis, MinorAxis, Orientation,
# ReferenceFrequency, SpectralIndex= with '[]'
# NOTE: no default values taken, for point sources
#  major,minor,orientation has to be all zero
# Example:
# note: bmaj,bmin, Gaussian radius in degrees, bpa also in degrees
Gtest1, GAUSSIAN, 18:59:16.309, -22.46.26.616, 100, 100, 100, 100,
0.222, 0.111, 100, 150e6, [-1.0]
Ptest2, POINT, 18:59:20.309, -22.53.16.616, 100, 100, 100, 100, 0,
0, 0, 140e6, [-2.100]
```

and also it can read an LSM sky model like (see chapter on SAGECAL for more information)

```
## this is an LSM text (hms/dms) file
## fields are (where h:m:s is RA, d:m:s is Dec):
## name h m s d m s I Q U V spectral_index RM
##    extent_X(rad) extent_Y(rad) pos_angle(rad) freq0
P1C1 1 35 29.128 84 21 51.699 0.061585 0 0 0 0 0 0 0 0 1000000.0
```

using `-o 0` for BBS and `-o 1` or `-o 1` for LSM. Note that `buildsky` will now (version 0.0.6) only produce LSM with 3rd order spectra. Spectral indices use natural logarithm, $\exp(ln(I_0) + p1 * ln(f/f_0) + p2 * ln(f/f_0)^2 + \ldots)$ so if you have a model with common logarithms like $10^{(log(J_0)+q1*log(f/f_0)+q2*log(f/f_0)^2+\ldots)}$ then, conversion is $I_0 = J_0$, $p1 = q1$, $p2 = q2/ln(10)$, $p3 = q3/(ln(10)^2)$ and so on.

As you can see, both above sky models are the same. In addition, the LSM sky model can be used to represent Gaussians (name starting with `G`), disks (name starting with `D`) and rings (name starting with `R`).

Once you have such a sky model (text file `sky.txt`), and a FITS file called `example.fits`, you can do many things:

```
restore -f example.fits -i sky.txt
```

will replace the FITS file with the sky model, so the original image will be overwritten;

```
restore -f example.fits -i sky.txt  -a
```

will add the sky model to the image; and

```
restore -f example.fits -i sky.txt  -s
```

will subtract the sky model from the FITS file.

You can also use solutions obtained by `SAGECal` when you restore a sky model:

```
restore -f example.fits -i sky.txt -c sagecal_cluster.txt -l sagecal_sky.txt
```

will use the solution file `sagecal_sky.txt` and the cluster file `sagecal_cluster.txt` while restoring the sky model. New solution files created by `SAGECal` has 3 additional lines at the beginning. Newer versions (0.0.10) of restore will properly handle this.

As before, you can see more options by typing

```
restore -h
```

14.2.7 shapelet_gui

The GUI used in decomposing FITS file to shapelets is called `shapelet_gui`. Once you run this program you will be seeing the GUI as in Fig. 34.

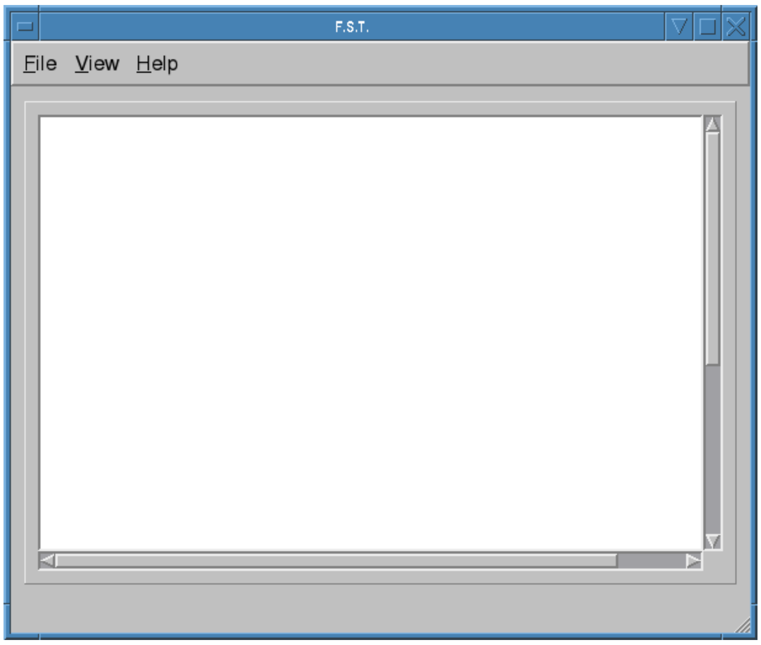
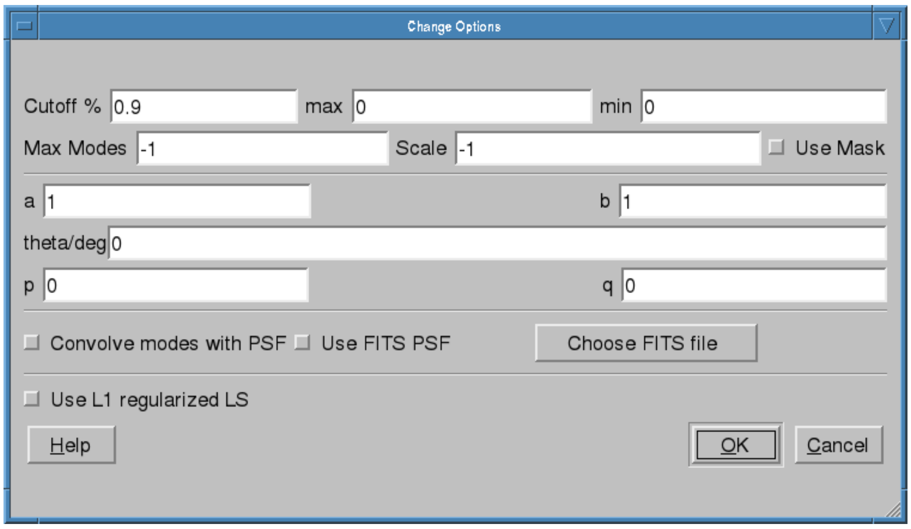


Figure 34: The `shapelet_gui` initial screen.

The essential parameters can be changed by using `View->Change Options` menu item. Once you select this, you will see the dialog as in Fig. 35. We will go through the options in Fig. 35 one by one.

- **Cutoff** This parameter is used to select the rectangle of pixels where most of the flux in the image is concentrated. A cutoff of 0.9 will select all the pixels above 0.1 of the peak flux. By using cutoff of 1.0, the whole image is selected.
- **max** If this value is not 0, pixels above this value will be truncated to this value.
- **min** If this value is not 0, pixels below this value will be truncated to 0.
- **Max Modes** The maximum number of shapelet basis functions used. If you enter 100 here, a 10×10 array of shapelet modes will be used. Use a small number here to save memory. The default value of -1 makes the program determine this automatically.
- **Scale** This is the scale (or β) of the shapelet basis. The default value of -1 makes the program determine this automatically.



Cutoff % 0.9 max 0 min 0
Max Modes -1 Scale -1 ☐ Use Mask
a 1 b 1
theta/deg 0
p 0 q 0
☐ Convolve modes with PSF ☐ Use FITS PSF
☐ Use L1 regularized LS

Figure 35: The options dialog for shapelet decomposition.

- **Use Mask** Instead of using a cutoff, we can also use a mask to select the pixels for shapelet modeling. The mask can be created using `Duchamp`. If this option is enabled, for the image `example.fits` FITS file, you must have the `example.MASK.fits` mask file in the same location. Note: make sure that `flagMaskWithObjectNum 0` is used for the input for Duchamp.

- **a, b, theta** These parameters are used in linear transforms. It is possible to scale and rotate your image before you do a shapelet decomposition. This is not yet implemented in BBS.

- **p, q** Normally, the center of the shapelet basis is selected to be the center of the FITS file. However, you can give any arbitrary location of your FITS file as the center by changing **p** and **q**. These have to be in pixels.

- **Convolve modes with PSF** As we mentioned before, almost all images will have a PSF. If the PSF is larger than the pixel size, it is useful to enable this option. The PSF is obtained by using the `BMAJ`,`BMIN`,`BPA` keywords of the FITS file.

- **Use FITS PSF** It is also possible to give another FITS file as the PSF. This generally has to be much smaller than the image.

- **Use L1 regularized LS** Instead of using normal L2 minimization to find the shapelet decomposition, you can also use L1 regularization. The difference in results is negligible in most cases.

It is advised to always enable **Use Mask** and **Convolve modes with PSF** options to get best performance. You can also get more information on all these options by clicking the **Help** button.

Finally, after fine tuning your options, you can select `File->Open` to select your FITS file and it will produce an output like Fig. 36. If you are not satisfied with the result, you can go back and `View->Change Options` to re-tune your parameters. Once you have done that, you can decompose the same FITS file by selecting `View->Decompose` from the menu.

Apart from displaying the output, each time you decompose a FITS file, `shapelet_gui` will produce several files. Most importantly, for your input `example.fits` image, it will produce `example.fits.modes` text file that can be used in BBS. Here is an extract of one such file:

```
23 23 27.273176 58 49 1.217289
```

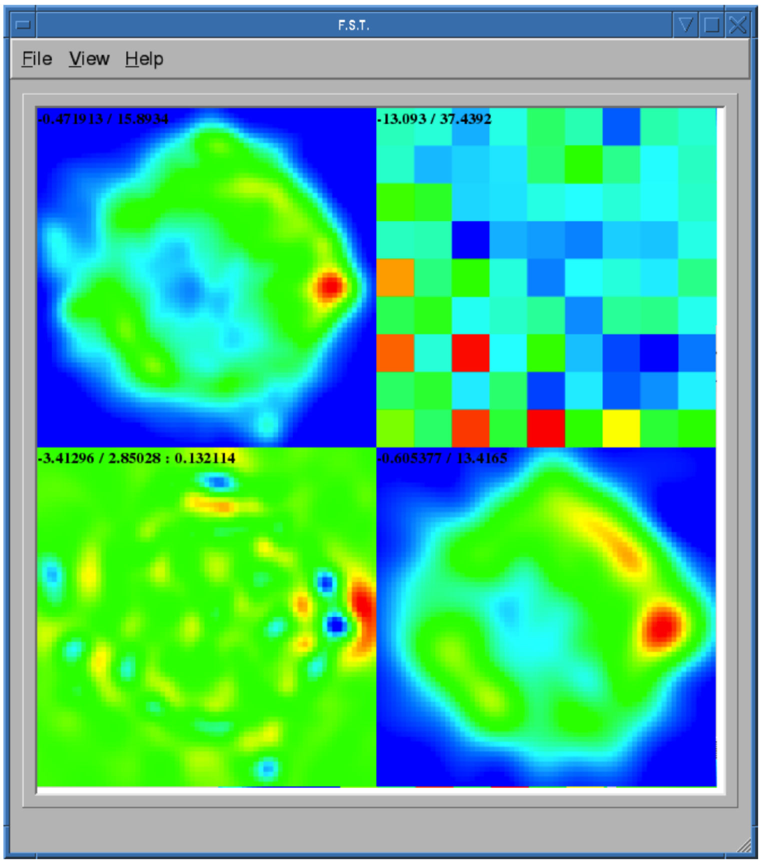


Figure 36: Output of shapelet modelling: (top left) original image (top right) shapelet modes (bottom left) residual image (bottom right) shapelet model.

```
9 1.255970e-03
0 1.864041e+01
1 5.311269e+00
2 3.354807e+01
3 7.081891e+00
4 3.743916e+01
5 1.209364e+01
6 2.458361e+01
7 7.033823e+00
8 8.411157e+00
-- many more rows --
# BBS format:
## NAME shapelet  23:23:27.273176 58.49.1.217289 1.0 thisfile.fits.modes
```

The thing to note from the above listing is the last line. It shows you exactly how to enter this into BBS. You have to create a text file such as

```
#
FORMAT = Name Type RA Dec I IShapelet

Ex1 shapelet  23:23:27.273176 58.49.1.217289 1.0 example.fits.modes
```

where we have copied the last line, changing the source name to whatever we like (in this case `Ex1`) and changing the last field to `example.fits.modes`.

14.2.8 convert_skymodel.py

This script converts sky models in BBS format to LSM format and vice versa.

```
Usage: convert_skymodel.py [options]

Options:
  -h, --help            show this help message and exit
  -i INFILE, --infile=INFILE
                        Input sky model
  -o OUTFILE, --outfile=OUTFILE
                        Output sky model (overwritten!)
  -b, --bbstolsm        BBS to LSM
  -l, --lsmtobbs        LSM to BBS
```

14.3 Step by Step Example

In this section, we will use most of the programs described before to calibrate a LOFAR observation of Virgo-A. We will use Fig. 33 (FITS file `vira-cen.fits`) to build the initial sky model.

14.3.1 Initial point source model

As we mentioned in section 14.1, the central compact part in Fig. 33 is best modeled using point sources. Therefore, we create the following as input to `Duchamp`

```
imageFile vira-cen.fits
logFile   logfile.txt
outFile   results.txt
spectraFile spectra.ps
minPix    5
flagATrous  0
snrRecon  10.
snrCut    5.
threshold 10.010
minChannels 3
flagBaseline    0
flagKarma 1
karmaFile duchamp.ann
flagnegative 0
flagMaps 0
flagOutputMask 1
flagMaskWithObjectNum 1
flagXOutput 0
```

After running `Duchamp` with this input file, we select only the bright compact center (that is the reason for using 10.01 as threshold) as seen on Fig. 37. Now we run `buildsky` to build the sky model for this as

```
buildsky -f vira-cen.fits -m vira-cen.MASK.fits
```

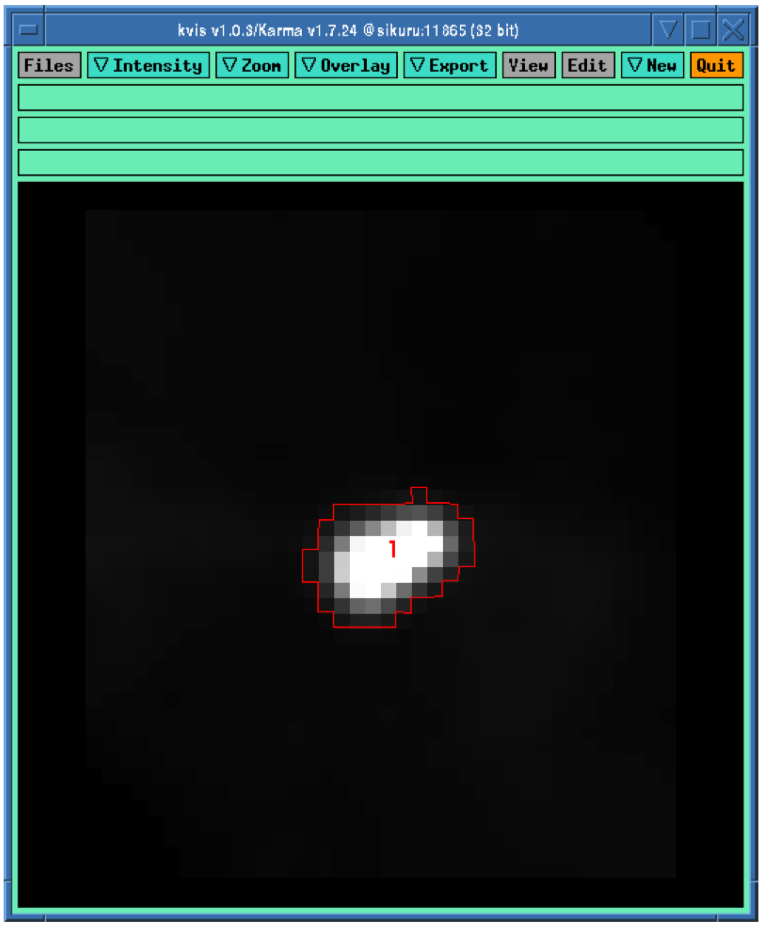


Figure 37: Compact center indicated by the red curve.

This will create the first part of the sky model for BBS (file `vira-cen.fits.sky.txt`):

```
# (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.
P1C1, POINT, 12:30:45.93, +12.23.48.07, 172.155091, 0.0, 0.0, 0.0
P1C2, POINT, 12:30:47.39, +12.23.51.92, 141.518663, 0.0, 0.0, 0.0
P1C3, POINT, 12:30:47.34, +12.23.31.64, 173.054910, 0.0, 0.0, 0.0
P1C4, POINT, 12:30:48.90, +12.23.40.67, 177.304557, 0.0, 0.0, 0.0
P1C5, POINT, 12:30:48.75, +12.23.21.23, 155.029319, 0.0, 0.0, 0.0
```

Using ds9 we can also see our sky model as in Fig. 38.

14.3.2 Initial shapelet model

Next, we need to model the extended structure inf Fig. 33. However, before we do this we have to subtract our point source model from this figure. We use `restore` to do this

```
restore -f vira-cen.fits -i vira-cen.fits.sky.txt -s
```

which gives us the new image as in Fig. 39. Note that the bright central part in Fig. 39 is almost subtracted. It is not completely gone, and some parts of it is negative. Nevertheless, this is all right for

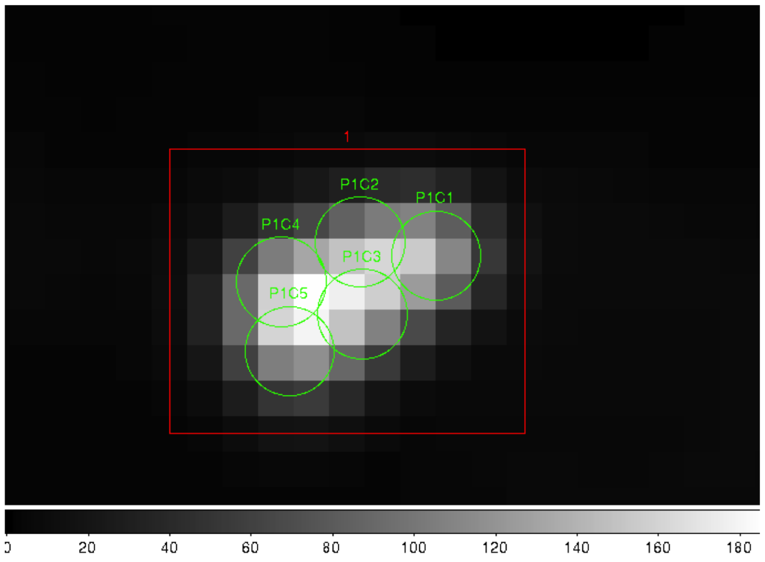


Figure 38: Compact center modeled by two point sources (green circles).

now because we are only building an approximate sky model. Now we need to create another mask for this image for the diffused structure. We use to following file for `Duchamp`.

```
imageFile vira-cen.fits
logFile   logfile.txt
outFile   results.txt
spectraFile spectra.ps
minPix    5
flagATrous  0
snrRecon  10.
snrCut    5.
threshold 1.010
minChannels 3
flagBaseline    0
flagKarma 1
karmaFile duchamp.ann
flagnegative 0
flagMaps 0
flagOutputMask 1
flagMaskWithObjectNum 0
flagXOutput 0
```

Note that we have used a lower threshold (1.01) this time, compared to the previous value. Once running `Duchamp`, we get the mask as indicated by Fig. 40.

Now we are ready to build the shapelet model. We first change some parameters using `View->Change Options`. We set **Cutoff** to 1.0, **Max Modes** to 200, and the center **p** to 75 and **q** to 74 to move the origin of the shapelets a bit. Furthermore, we enable `Use Mask` and `Convolve Modes with PSF` options. Then we use `File->Open` to select `vira-cen.fits` as input. After a few seconds, we get the result as in Fig. 41.

We can easily create an input to BBS for this shapelet model as follows:

```
#
```

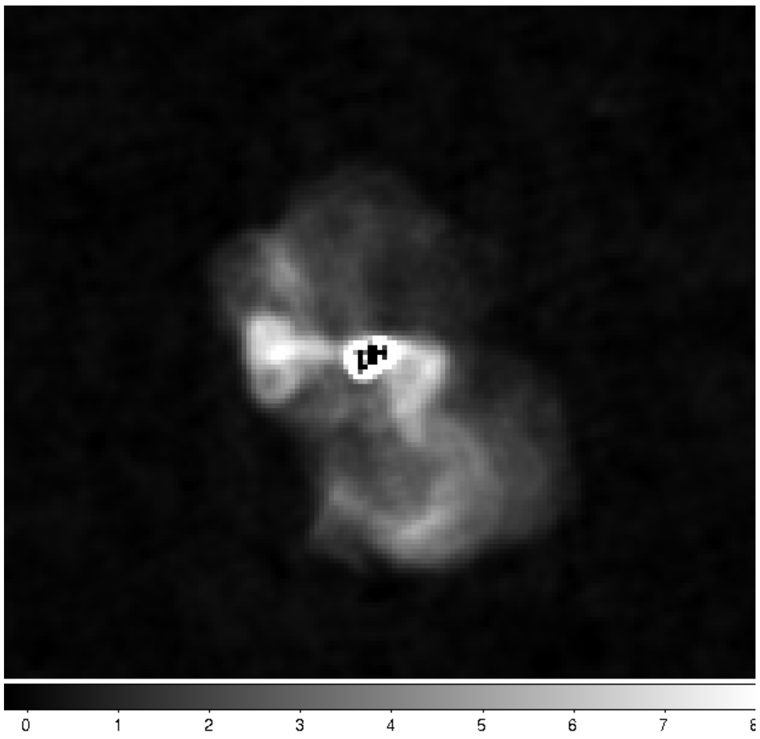


Figure 39: Diffused structure after subtracting the center.

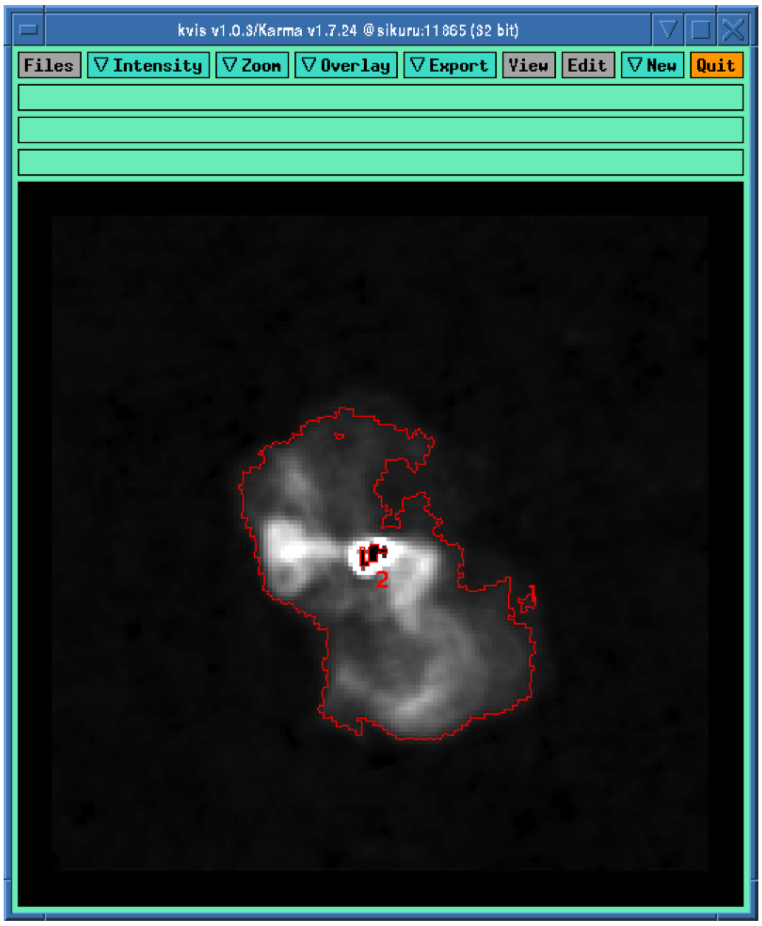


Figure 40: Mask for the diffused structure.

```
FORMAT = Name Type RA Dec I IShapelet
```

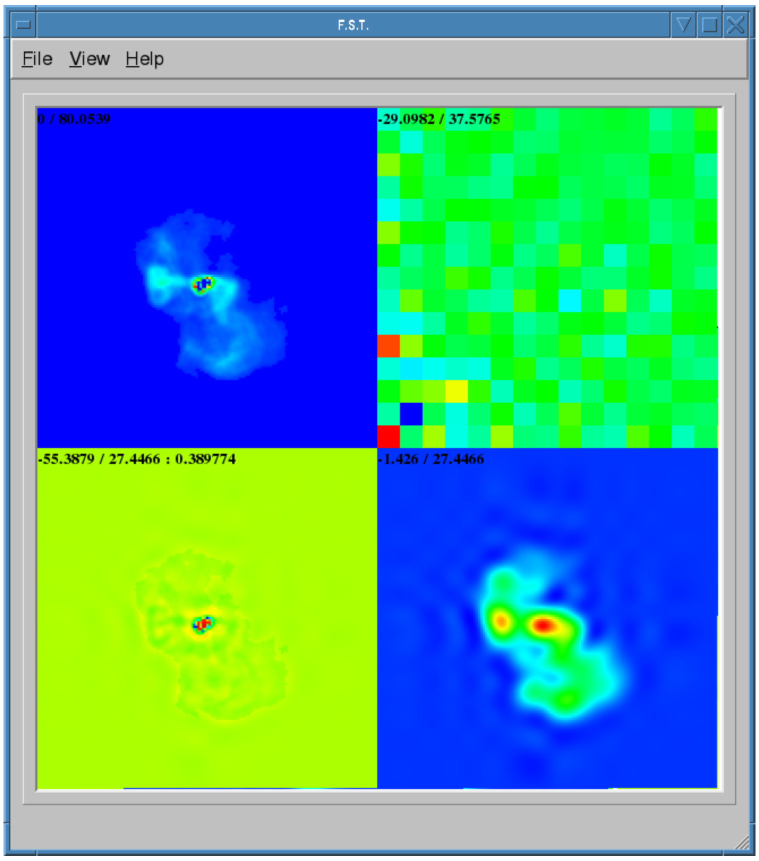


Figure 41: Shapelet model of the diffused structure.

```
VirAD shapelet   12:30:48.317433 12.23.27.999947 1.0  vira-cen.fits.modes
```

14.3.3 Using both shapelets and point sources together

Here is the complete sky model using both point sources and shapelets:

```
# (Name, Type, Patch, Ra, Dec, I, Q, U, V, ReferenceFrequency='60e6',
   SpectralIndex='[0.0]', Ishapelet) = format
# The above line defines the field order and is required.
, , CENTER, 12:30:45.00, +12.23.48.00
P1C1, POINT, CENTER, 12:30:45.93, +12.23.48.07, 172.155091, 0.0, 0.0, 0.0
P1C2, POINT, CENTER, 12:30:47.39, +12.23.51.92, 141.518663, 0.0, 0.0, 0.0
P1C3, POINT, CENTER, 12:30:47.34, +12.23.31.64, 173.054910, 0.0, 0.0, 0.0
P1C4, POINT, CENTER, 12:30:48.90, +12.23.40.67, 177.304557, 0.0, 0.0, 0.0
P1C5, POINT, CENTER, 12:30:48.75, +12.23.21.23, 155.029319, 0.0, 0.0, 0.0
VirAD, shapelet, CENTER, 12:30:48.317433, 12.23.27.999947, 1.0, , , ,
vira-cen.fits.modes
```

Note that the above model gives `CENTER` as the patch direction.

14.3.4 Simulation

Once we have the point source and shapelet sky models, we can run BBS. After this is done, you are free to do whatever you like with these sky models.

First and foremost, it is advised to do a simulation with your sky model and the measurement set that you need to calibrate to make sure your sky model is correct. Moreover, this is also useful to check if there are any errors in flux scales. For a point source, there cannot be any error in flux. However, for an extended source, the flux will be slightly lower than your model in the image. This is because the Fourier transform preserves the integral of flux and not the peak value. So, it is urged to do a simulation first before doing any calibration. We have shown the simulated image in Fig. 42.

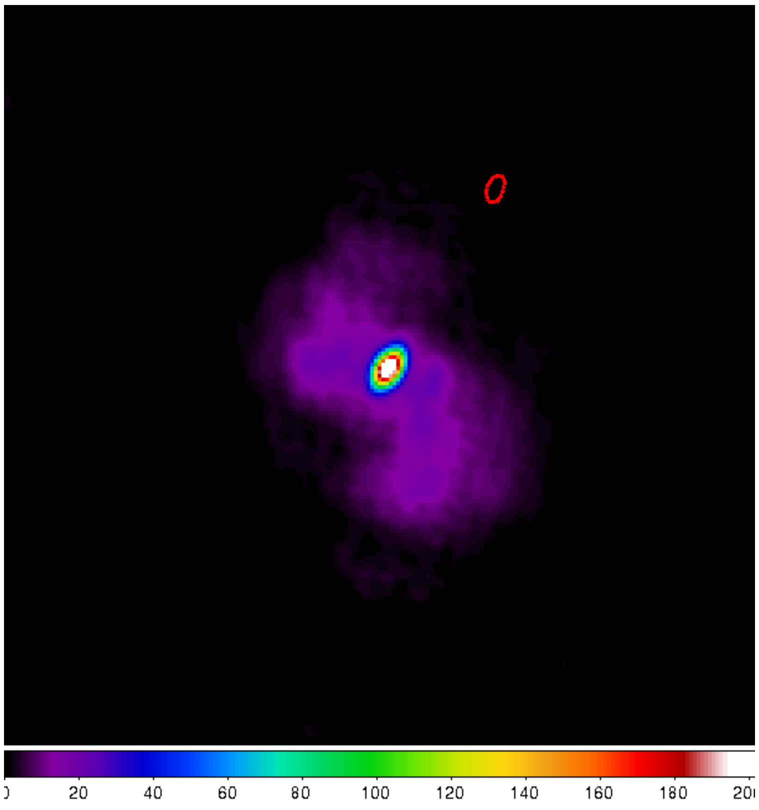


Figure 42: Simulated image of Virgo-A. The red ellipse is the PSF.

By looking at Fig. 42, we do not see any major discrepancy in our sky model (although we have lower resolution) so we go ahead with calibration.

14.3.5 Calibration

You can use the normal calibration procedure you adopt with any other LOFAR observation here. So we will not go into details. We have shown the image made after calibration in Fig. 43.

NOTE: It is advised to use uniform weights to compare the calibrated image to the model image.

Using Fig. 43, we can repeat our sky model construction to get a better result. This of course depends on your science requirements.

14.3.6 Residual

A better way to check the accuracy of your sky model is to subtract this model from the calibrated data and make an image of the residual. In Fig. 44, we have shown the residual for two subbands of 1.5 hour duration at 55 MHz. We clearly see an off center source (about 2 Jy) on top right hand corner.

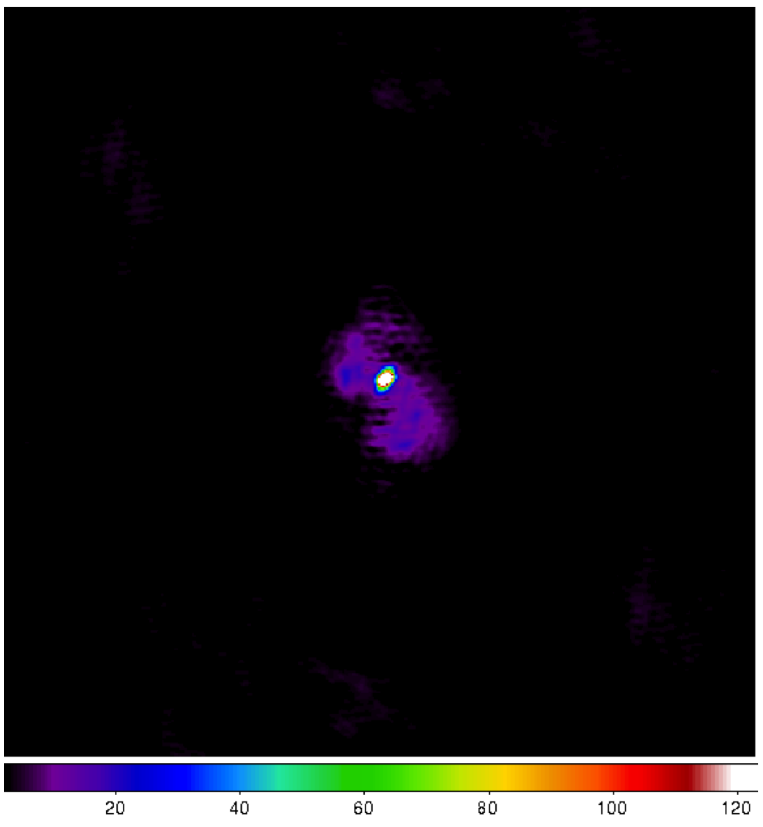


Figure 43: Calibrated image of Virgo-A (uniform weights).

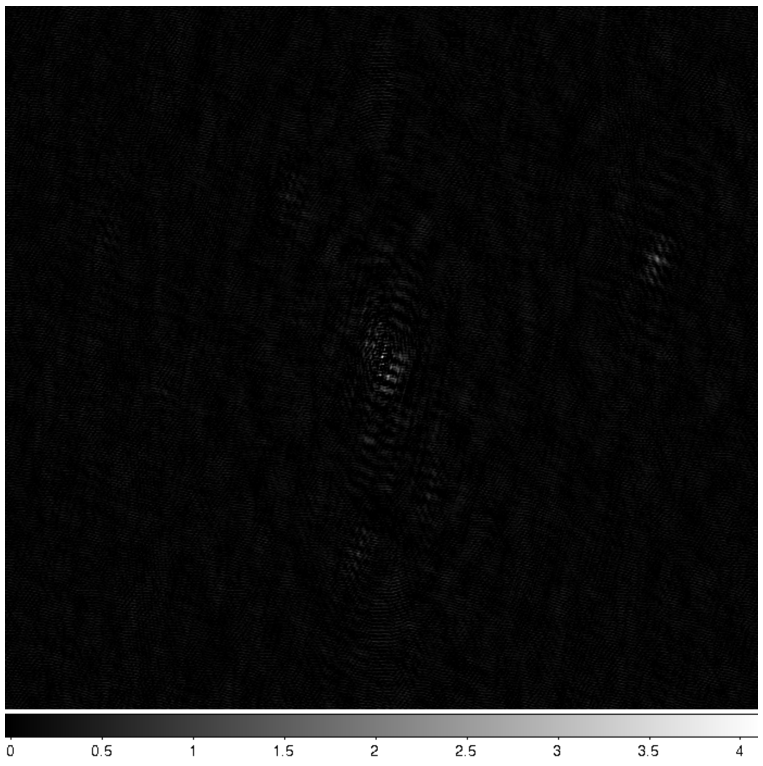


Figure 44: Residual image of Virgo-A. An off center source is present on top right hand corner.

14.3.7 Recalibration

Once you have the residual image, you can also include to off center sources and update the sky model to re-calibrate the data.

14.4 Conclusions

We have given only a brief overview of the software and techniques in extended source modeling using shapelets. There are many points that we have not covered in this tutorial. However, we hope you (the user) will experiment and explore all available possibilities. Questions/Comments/Bug reports can be sent to `yatawatta[at]astron[dot]nl`.

References

S. Yatawatta, “Fundamental limitations of pixel based image deconvolution in radio astronomy,” *in proc. IEEE Sensor Array and Multichannel Signal Processing Workshop (SAM)*, Jerusalem, Israel, pp. 69–72, 2010.

S. Yatawatta, “Radio astronomical image deconvolution using prolate spheroidal wave functions,” *IEEE International Conference on Image Processing (ICIP) 2011*, Brussels, Belgium, Sep. 2011.

S. Yatawatta, “Shapelets and Related Techniques in Radio-Astronomical Imaging,” *URSI GA*, Istanbul, Turkey, Aug. 2011.

15 Practical examples[82]

In this Chapter, examples of how to inspect and analyze LOFAR data are given. The aim of these exercises is for the User to become familiar with the software used to process LOFAR data and to be able to apply this knowledge to other data sets. Please note that each LOFAR data set is different and special care should be taken when directly applying the methods given in these exercises to other LOFAR data sets.

It is assumed here that the User has already gone through the "Getting Started" procedure described in Chapter 1. Note, where NDPPP is used in this chapter, the reader should read DPPP.

15.1 3C 295 – A bright source at the centre of the field

In this exercise, the user will calibrate LBA and HBA datasets for 3C 295. By the end of the exercise the User should be able to

- inspect raw LOFAR data,
- automatically and manually flag data with NDPPP, including demix the LBA data,
- calibrate the data with BBS,
- produce maps with AWimager,
- create a sky model from the data, and,
- subtract bright sources using BBS

Log into one of the compute nodes above, and make a new directory, for example,

```
> ssh -Y lof019
> cd /data/scratch/<username>/
> mkdir tutorial/
> mkdir tutorial/3c295/
> cd tutorial/3c295
```

You will be using the LOFAR software tools. To initialise these use

```
> use LofIm
```

15.2 HBA

The raw data set for this exercise can be found in

```
/globaldata/COOKBOOK/Tutorial/3c295/data/L74759/
```

The unique LOFAR observation number is L74759 and there are two sub-bands, SB000 and SB001. The data set is in Measurement Set (MS) format and the filenames are respectively

[82]The author of this Chapter is Wendy WIlliams (wwilliams[at]strw[dot]leidenuniv[dot]nl). Contribution was also given by many commissioners: Alicia Berciano Alba, Valentina Vacca, Poppy Martin, Maciej Ceglowski, and Carmen Toribio.

```
L74759_SAP000_SB000_uv.MS
L74759_SAP000_SB001_uv.MS
```

for the two sub-bands.

To copy it to your current working directory use:

```
cp -r /globaldata/COOKBOOK/Tutorial/3c295/data/L74759/L74759_SAP000_SB00[01]_uv.MS .
```

15.2.1 Inspecting the raw data

It is always useful to find out what the details of the observation are (frequency, integration time, number of stations) before starting on the data reduction. This is done using the command,

```
> msoverview in=L74759_SAP000_SB000_uv.MS verbose=T
msoverview: Version 20110407GvD
================================================================================
           MeasurementSet Name:  /data2/wwilliams/tutorial/3c295/work/hba/L74759_SAP000_SB000_uv.MS      MS Version 2
================================================================================
           This is a raw LOFAR MS (stored with LofarStMan)

   Observer: unknown     Project: 2012LOFAROBS
Observation: LOFAR
Antenna-set: HBA_DUAL_INNER

Data records: 5337090       Total integration time = 3599 seconds Observed from   12-Nov-2012/12:47:00.0   to   12-Nov-2012/13:46:59.0 (UTC)


Fields: 1
  ID   Code Name                RA               Decl           Epoch        nRows
  0         BEAM_0              14:11:20.500000 +52.12.10.00000 J2000       5337090

Spectral Windows:  (1 unique spectral windows and 1 unique polarization setups)
  SpwID  Name   #Chans   FrameCh1(MHz)     ChanWid(kHz)  TotBW(kHz)  CtrFreq(MHz)  Corrs
  0      SB-0      64   TOPO     118.849         3.052       195.3    118.9453   XX  XY  YX  YY

Antennas: 54:
  ID   Name  Station   Diam.    Long.         Lat.                Offset from array center (m)             ITRF Geocentric coordinates (m)
                                                                     East         North     Elevation               x               y               z
  0    CS001HBA0LOFAR     31.3 m   +006.52.07.1  +52.43.34.7         -0.0006       -0.1627  6364571.5626  3826896.235000   460979.455000  5064658.203000
  1    CS001HBA1LOFAR     31.3 m   +006.52.02.2  +52.43.31.8         -0.0013       -0.1634  6364571.8531  3826979.384000   460897.597000  5064603.189000
  2    CS002HBA0LOFAR     31.3 m   +006.52.07.6  +52.43.46.8         -0.0006       -0.1598  6364569.5445  3826600.961000   460953.402000  5064881.136000
  3    CS002HBA1LOFAR     31.3 m   +006.52.08.0  +52.43.48.2         -0.0005       -0.1595  6364569.4094  3826565.594000   460958.110000  5064907.258000
  ...
  ...
  ...
  42   CS501HBA0LOFAR     31.3 m   +006.51.57.9  +52.44.29.9         -0.0019       -0.1495  6364565.1209  3825568.820000   460647.620000  5065683.028000
  43   CS501HBA1LOFAR     31.3 m   +006.51.59.7  +52.44.25.8         -0.0017       -0.1504  6364565.4869  3825663.508000   460692.658000  5065607.883000
  44   RS106HBALOFAR     31.3 m   +006.59.05.6  +52.41.21.6         0.0598       -0.1945  6364586.2658  3829205.598000   469142.533000  5062181.002000
  45   RS205HBALOFAR     31.3 m   +006.53.50.8  +52.40.17.6         0.0143       -0.2098  6364593.2751  3831479.670000   463487.529000  5060989.903000
  ...
  ...
  ...
  52   RS508HBALOFAR     31.3 m   +006.57.13.3  +53.03.21.7         0.0436        0.1214  6364441.3265  3797136.484000   463114.447000  5086651.286000
  53   RS509HBALOFAR     31.3 m   +006.47.04.7  +53.13.30.1        -0.0443        0.2670  6364384.0353  3783537.525000   450130.064000  5097866.146000

  The MS is fully regular, thus suitable for BBS
   nrows=5337090   ntimes=3594   nbands=1   nbaselines=1485 (54 autocorr)
```

From this, you should see that HBA_DUAL mode was used, i.e. the core stations are split in two HBA sub-fields, giving a total of 54 stations. The observation was $\sim$ 1 hour (3599 seconds) with 3594 timestamps so the time resolution is $\sim$ 1 s. There are 64 spectral channels and the frequency is 118.849 MHz for SB000 and 119.044 MHz for SB001, and the total bandwidht for each subband is $\sim$ 0.2 MHz.

15.2.2 Flagging and data compression

The data set that we are using is uncompressed and unflagged; the total size of each MS is 11 Gb. The data flagging and compression are carried out using NDPPP (see Chapter 5 for details). Typically, initial RFI flagging and averaging will be done by the averaging pipeline run by the Radio Observatory. Note that the limitation on the compression in frequency is set by the size of the field you wish to image and the amount of bandwidth smearing at the edges of the field. The time averaging is limited not only by the amount of time smearing you will allow but also by the changes in the ionosphere.

In this example we are flagging the data using the aoflagger algorithm within NDPPP. Here we will compress the sub-band to 4 channels in frequency and 5 s in time. The parset file for the flagging and compression should be copied to your working directory,

```
> cp /globaldata/COOKBOOK/Tutorial/3c295/parsets/hba/NDPPP_HBA_preprocess.parset .
> cat NDPPP_HBA_preprocess.parset
msin = L74759_SAP000_SB000_uv.MS
msin.autoweight=TRUE
msin.datacolumn=DATA

msout = L74759_SAP000_SB000_uv.MS.avg.dppp
msout.datacolumn=DATA

steps=[preflagger0,preflagger1,aoflagger,averager]

preflagger0.chan=[0,1,62,63]
preflagger0.type=preflagger

preflagger1.corrtype=auto
preflagger1.type=preflagger

aoflagger.autocorr=F
aoflagger.timewindow=0
aoflagger.type=aoflagger

averager.freqstep=16      # compresses from 64 to 4 channels
averager.timestep=5      # compresses 5 time-slots, i.e. 5s
averager.type=averager
```

This parset file will take the data set, flag and compress and then make a new copy in your working area. If necessary, edit the msin and msout fields to point at your working directory, using your favourite editor (e.g. vim, nano, nedit). To run NDPPP use,

```
> NDPPP NDPPP_HBA_preprocess.parset > log.ndppp.flagavg 2>&1 &
```

The 2>&1 pipes both the stdout and stderr to the log file and the & runs the task in the background so you can continue working on the terminal.

Depending on the use of the cluster, it will take about $\sim 5 - 10$ minutes to flag and average the data. The progress bar reports the stage of the initial NDPPP steps, not the entire NDPPP run, so it will keep running several minutes after the progress bar reaches 100%. Note that it is normal to get the following warning:

```
log4cplus:WARN Property configuration file "parmdbm.log_prop" not found.
log4cplus:WARN Using basic logging configuration.
```

You can inspect the output log file by using,

```
> cat log.ndppp.flagavg
```

The log file lists the input and output parameters, the level of flagging at each step and the total amount of data flagged. You will see that the total data flagged for each of the flagging steps is 4.7%, 3.4% and 2.7% respectively.

Edit the msin and msout fields of the parset to do the same for the second sub-band.

The flagged and compressed data set should now be in your working directory and each MS should have a total size of 87 Mb, which is much more manageable than before. You can use msoverview to look at a summary of this data set using.

```
> msoverview in=L74759_SAP000_SB000_uv.MS.avg.dppp verbose=True
```

15.2.3 Post-compression data inspection and flagging

We will use the CASA task plotms to inspect the data. Only limited information for using plotms is given here, the User is directed to the CASA cookbook[83] for full details,

```
> use Casa
> casaplotms
```

Figure 45 shows the Amp. vs. time and Amp. vs. UV distance (wavelengths) for SB000.

By inspection of the amplitude vs. uv-distance plots, antenna's CS302HBA0 (blue) and CS302HBA1 (purple) have clearly low amplitudes compared to the other baselines. We will leave them for now and see how they perform after calibration.

15.2.4 Calibration with BBS

Black Board Self-calibration is a highly flexible calibration software developed for LOFAR that is capable of carrying out direction dependent gains (solutions for different parts of the image) and the time-dependent corrections for the LOFAR beam. In this example, we will go through the process of a single direction solution with the beam correction.

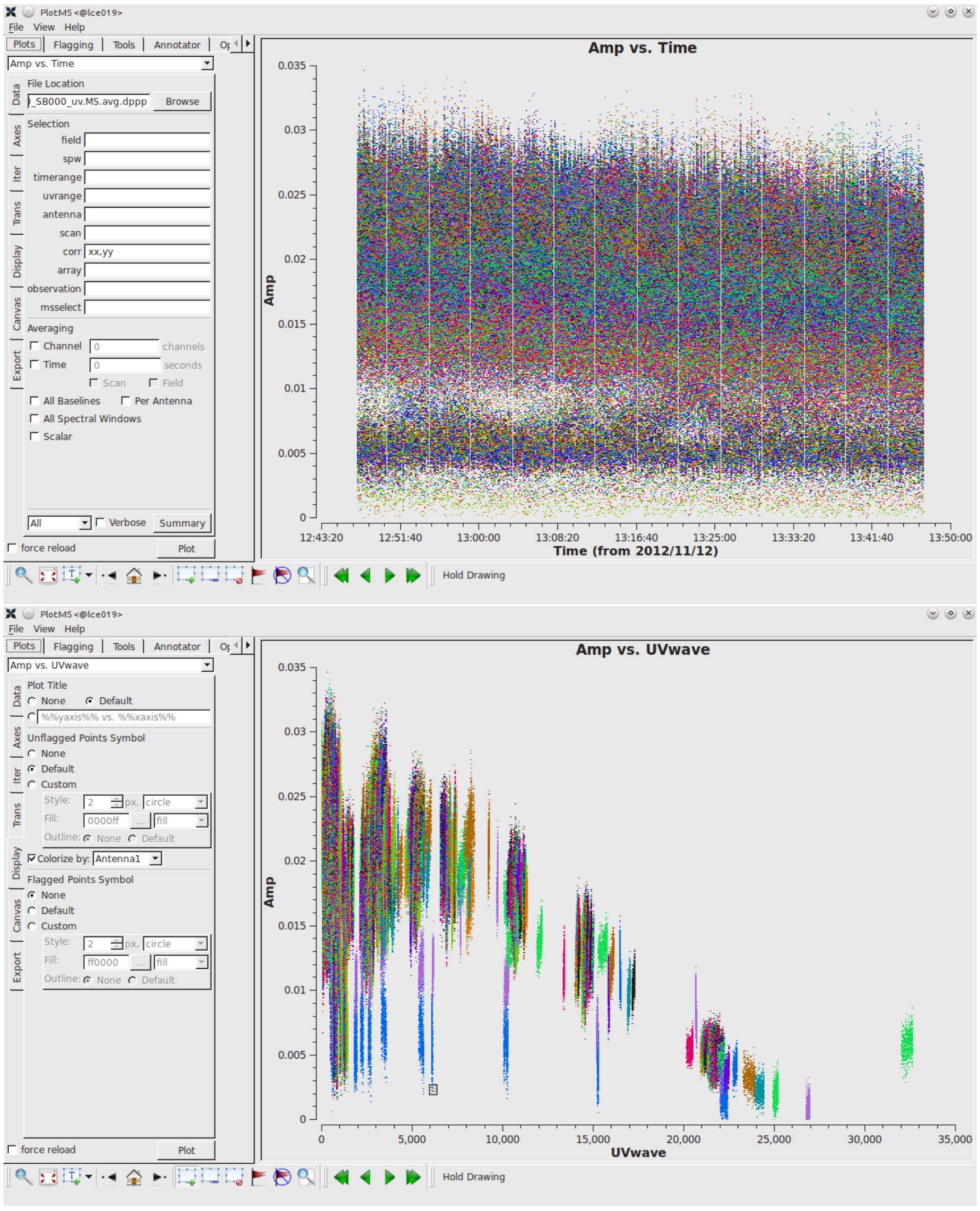
Here we will use the stand-alone version of BBS[84] to calibrate single sub-bands. The stand-alone version can be run using the following command

```
> calibrate-stand-alone <MS> <parset> <source catalog>
```

So before we can run the calibration, we need an initial sky model for correcting the data and a parset file to direct the calibration. Since 3C 295 is a well-known calibrator source, we already have a good model for it. The sky model consists of two point sources and can be copied to your working area using,

```
> cp /globaldata/COOKBOOK/Tutorial/3c295/models/3C295TWO.skymodel .
> cat 3C295TWO.skymodel
# (Name, Type, Patch, Ra, Dec, I, ReferenceFrequency='150.e6', SpectralIndex) = format

, , 3c295, 14:11:20.64, +52.12.09.30
3c295A, POINT, 3c295, 14:11:20.49, +52.12.10.70, 48.8815, , [-0.582, -0.298, 0.583, -0.363]
3c295B, POINT, 3c295, 14:11:20.79, +52.12.07.90, 48.8815, , [-0.582, -0.298, 0.583, -0.363]
```

[83]http://casa.nrao.edu/Doc/Cookbook/casa_cookbook.pdf
[84]BBS is described in Chapter 7



File Location: SB000_uv.MS.avg.dppp
Selection: corr xx,yy
Averaging: Channel 0 channels; Time 0 seconds
Plot Title: [✓ Default] %%yaxis%% vs. %%xaxis%%
Unflagged Points Symbol: [✓ Default]
Style: 2 px, circle; Fill: 0000ff fill; Outline: [✓ None]
Colorize by: Antenna1
Flagged Points Symbol: [✓ None]
Style: 2 px, circle; Fill: ff0000 fill; Outline: [✓ None]

Figure 45: SB000. Top: Plotting the visibility amplitude against time. Bottom: Plotting the visibility amplitude against UV distance in wavelengths. (The colour scheme ‘Antenna1’ is used here.)

Here you can see that the two point-source components on 3C 295 have been grouped in a single 'patch'. Note that there are other sources visible within the field of view, but 3C 295 should be sufficiently bright to dominate the field.

ASIDE: Usually one makes an initial sky model based on what we think the sky looks like at the frequency and resolution that we are interested in. This means constructing a model from good image we have at a different frequency/resolution, or in the case of self-calibration, the image we have just made (see for example Chapter 10). Alternatively, a sky model can be created using the gsm.py tool. This tool extracts sources in a cone of a given radius around a given position on the sky from the Global Sky Model or GSM. The GSM contains all the sources from the VLSS, NVSS, and WENSS survey catalogs. See Sections 6.4 and 6.5 for more information about the GSM and gsm.py.

Running gsm.py without any arguments will show you the correct usage (help).

```
> gsm.py

Insufficient arguments given; run as:

   /opt/cep/LofIm/daily/Tue/lofar_build/install/gnu_opt/bin/gsm.py outfile RA
DEC radius [vlssFluxCutoff [assocTheta]] to select using a cone

   outfile         path-name of the output file
                   It will be overwritten if already existing
   RA              cone center Right Ascension (J2000, degrees)
   DEC             cone center Declination     (J2000, degrees)
   radius          cone radius                 (degrees)
   vlssFluxCutoff  minimum flux (Jy) of VLSS sources to use
                   default = 4
   assocTheta      uncertainty in matching     (degrees)
                   default = 0.00278  (10 arcsec)
```

So now we can construct the command to make a model for the 3C 295 field:

```
> gsm.py 3c295_field.model 212.835495 52.202770 3.0
Sky model stored in source table: 3c295_field.model
```

For now, we will return to using the simple two point source model of 3C 295.

The parset file for BBS can be found at,

```
> cp /globaldata/COOKBOOK/Tutorial/3c295/parsets/hba/bbs.parset  .
> cat bbs.parset
Strategy.ChunkSize = 500
Strategy.Steps = [solve, correct]

Step.solve.Operation = SOLVE
Step.solve.Model.Sources = [3c295]
Step.solve.Model.Gain.Enable = T
Step.solve.Model.Beam.Enable = T
Step.solve.Model.Beam.UseChannelFreq = 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 = 50
Step.solve.Solve.Options.MaxIter = 500
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 = [3c295]
Step.correct.Model.Gain.Enable = T
Step.correct.Model.Beam.Enable = T
Step.correct.Model.Beam.UseChannelFreq = T
Step.correct.Output.Column = CORRECTED_DATA
```

This is a very simple parset file that solves and corrects the data. To run BBS, use the following command,

```
> calibrate-stand-alone -f L74759_SAP000_SB000_uv.MS.avg.dppp \
  bbs.parset 3C295TWO.skymodel > log.bbs.gaincal_sb000 2>&1 &
```

Note that using the redirect "> log.bbs.gaincal_sb000" command allows you to save and inspect the output of BBS and using the "&" at the end runs BBS in the background allowing you to continue with other tasks, e.g. you can simultaneously run a similar command for the second sub-band. The calibration process should be completed in about 10 minutes. There is currently no progress bar incorporated into calibrate-stand-alone. However, you can (somewhat primitively!) follow the progress by checking the log file, in particular by inspecting how many of the data chunks have been processed. Do a 'grep Time log.bbs.gaincal_sb000'. Alternatively, using the 'top' command will allow you to see when the process is complete.

Once complete it is useful to look at the calibrated data with parmdbplot.py:

```
> parmdbplot.py  L74759_SAP000_SB000_uv.MS.avg.dppp/instrument/
```

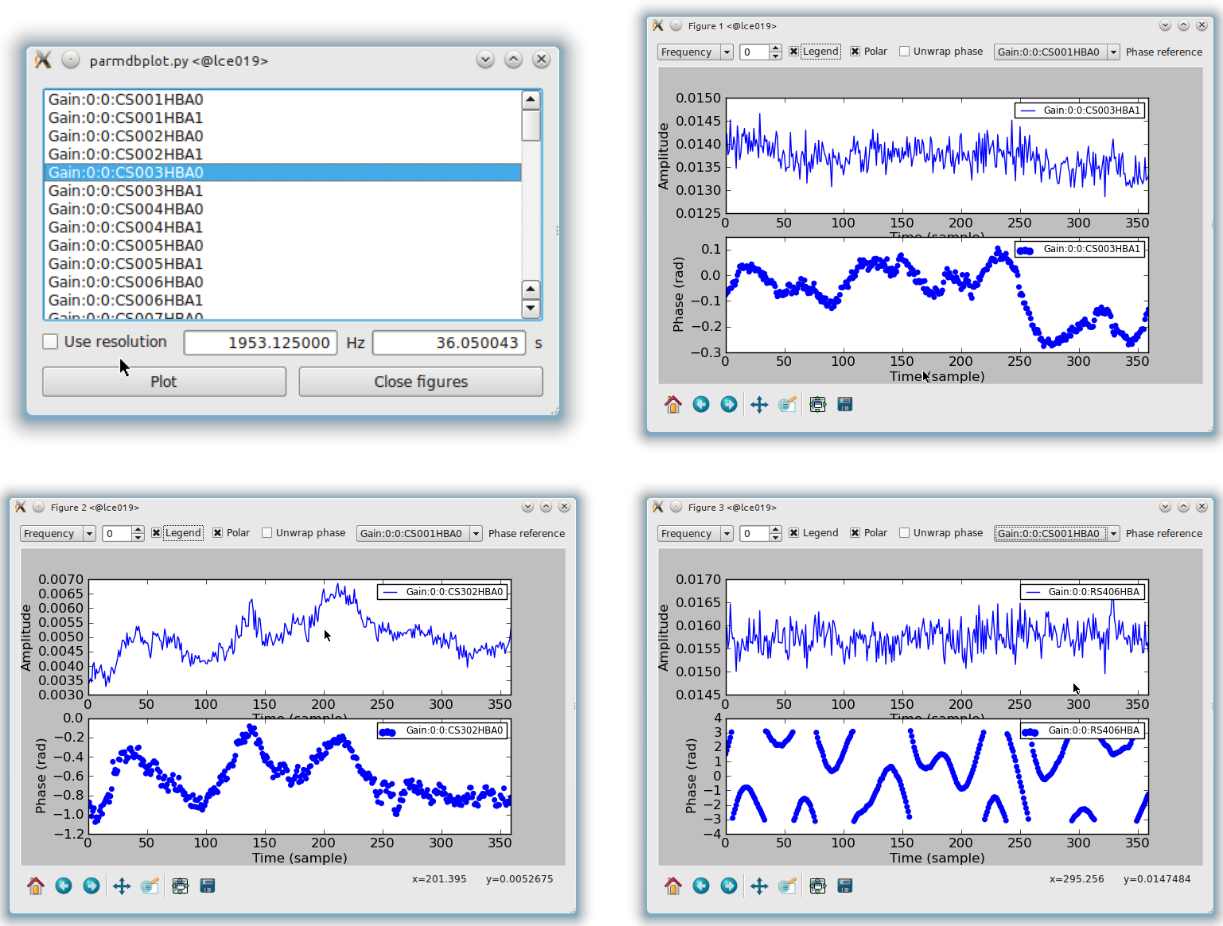
It is useful to de-select the 'use resolution' option as this will plot all of the solutions that we solved for. After de-selecting the 'use resolution' option select a few stations and look at the solutions. Figure 46 shows some solution plots for SB000.

ASIDE: While parmdbplot.py is very useful and diverse, sometimes you want a quick look at all the solutions. In python you can use `lofar.parmdb` to read and plot the solutions. This example script plots all the phase and amplitude solutions in a single image (see Fig. 47):

```
> python /globaldata/COOKBOOK/Tutorial/3c295/scripts/plot_solutions_all_stations.py -p -a \
L74759_SAP000_SB000_uv.MS.avg.dppp/instrument/ hba_sb000_gains
> display hba_sb000_gains_amp.png
> display hba_sb000_gains_phase.png
```

Or use George Heald's solplot.py (see Chapter 6: inspecting the solutions).

We can also inspect the corrected data with casaplotms. Go to the axes tab and plot the amplitude against time for the corrected data by selecting "Data Column: corrected" and plot only the XX

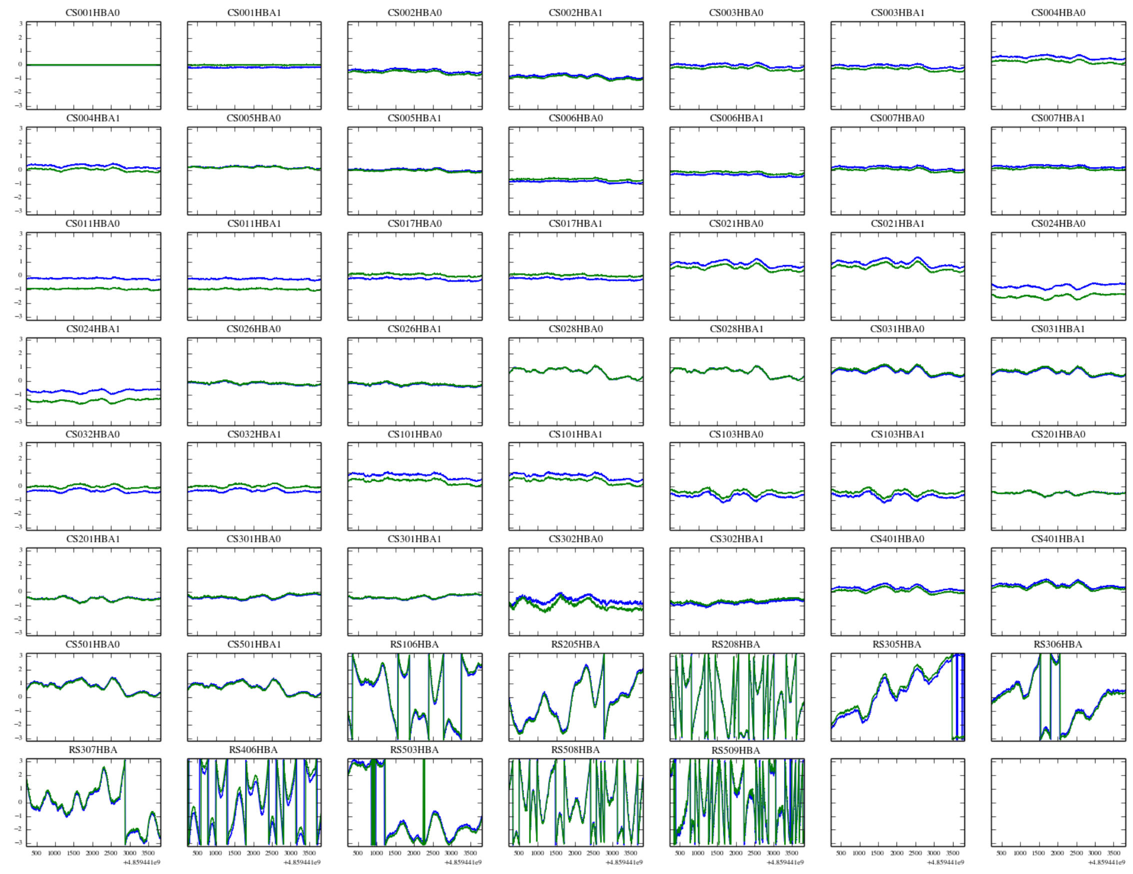

parmdbplot.py <@lce019>

```
Gain:0:0:CS001HBA0
Gain:0:0:CS001HBA1
Gain:0:0:CS002HBA0
Gain:0:0:CS002HBA1
Gain:0:0:CS003HBA0
Gain:0:0:CS003HBA1
Gain:0:0:CS004HBA0
Gain:0:0:CS004HBA1
Gain:0:0:CS005HBA0
Gain:0:0:CS005HBA1
Gain:0:0:CS006HBA0
Gain:0:0:CS006HBA1
Gain:0:0:CS007HBA0
```

☐ Use resolution 1953.125000 Hz 36.050043 s

Plot | Close figures

Figure 46: Top left: The parmdb window. Use this to select the stations for which you want to inspect the solutions, and to change the resolution that is used to display the solutions. Top right: The solutions for CS003HBA1. Bottom left: The solutions for CS302HBA0. Bottom right: The solutions for RS406HBA.



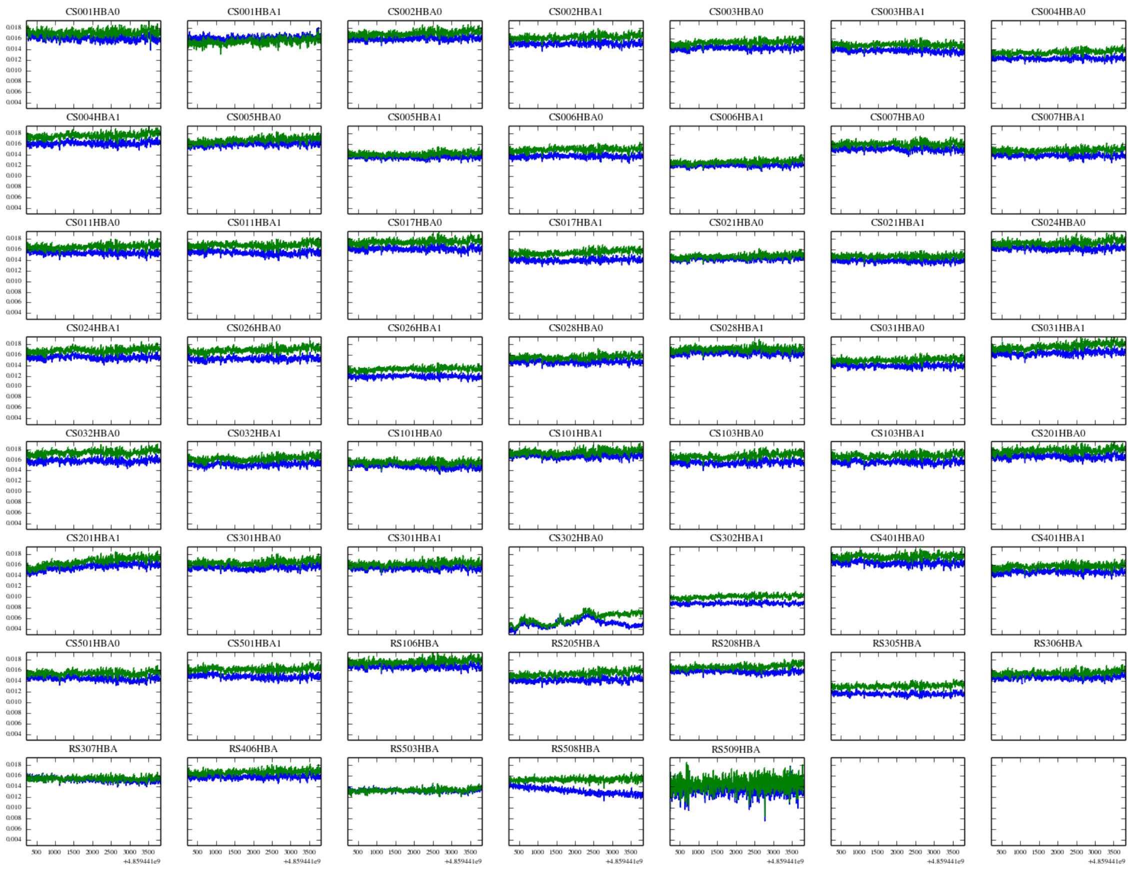
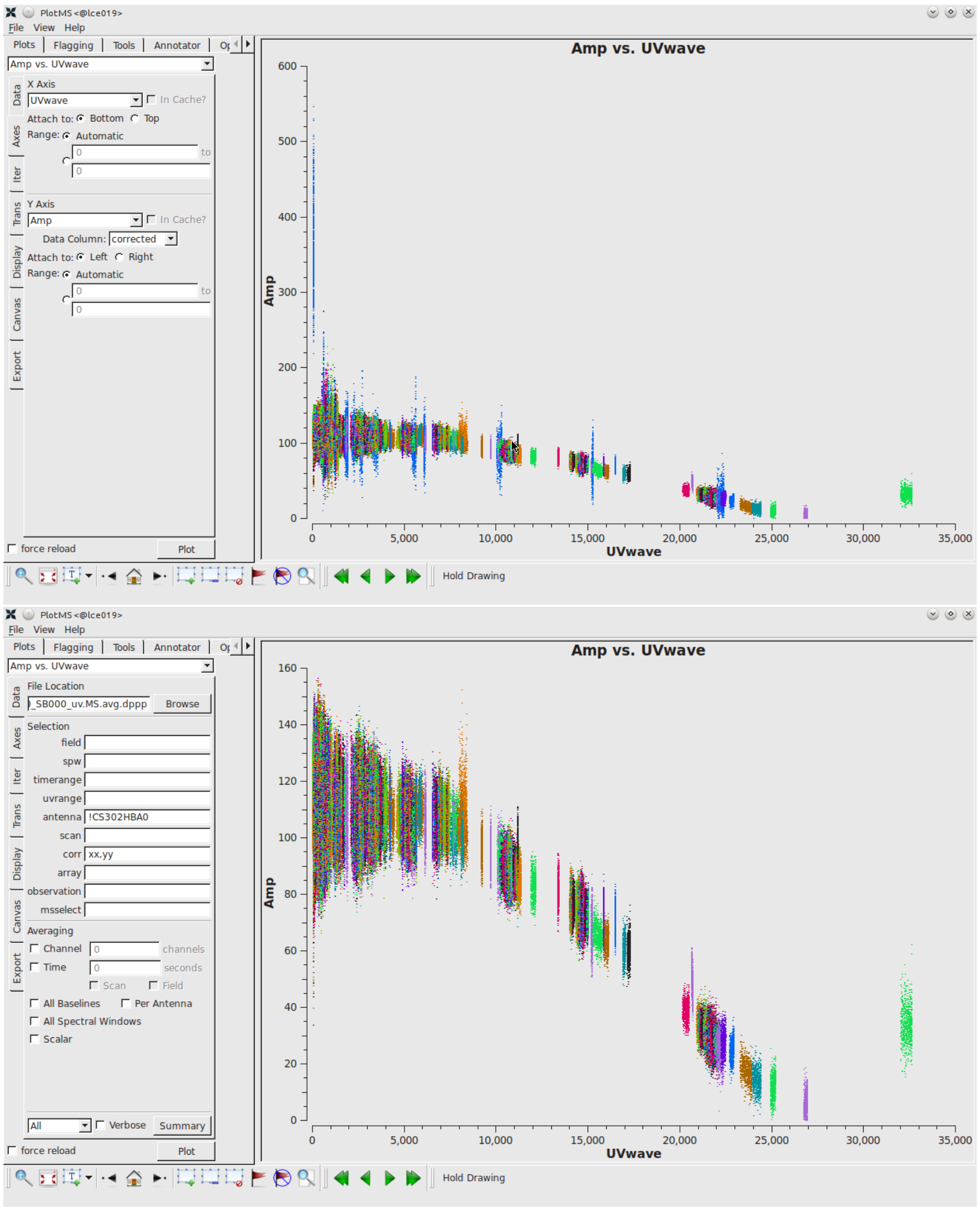


Figure 47: Top: Phase solutions for all stations for SB000 (polarizations in different colours). Bottom: Amplitude solutions for all stations for SB000.

and YY correlations. Figures 48 and 49 show these plots for SB000 and SB001 respectively. For SB000, it is clear that the solutions for CS302HBA are still very noisy. For both sub-bands, baselines RS508HBA&RS509HBA (visible in orange) and RS208HBA&RS509HBA (in green) look bad and for SB001, CS302HBA0&CS302HBA1 (in blue) also looks bad.



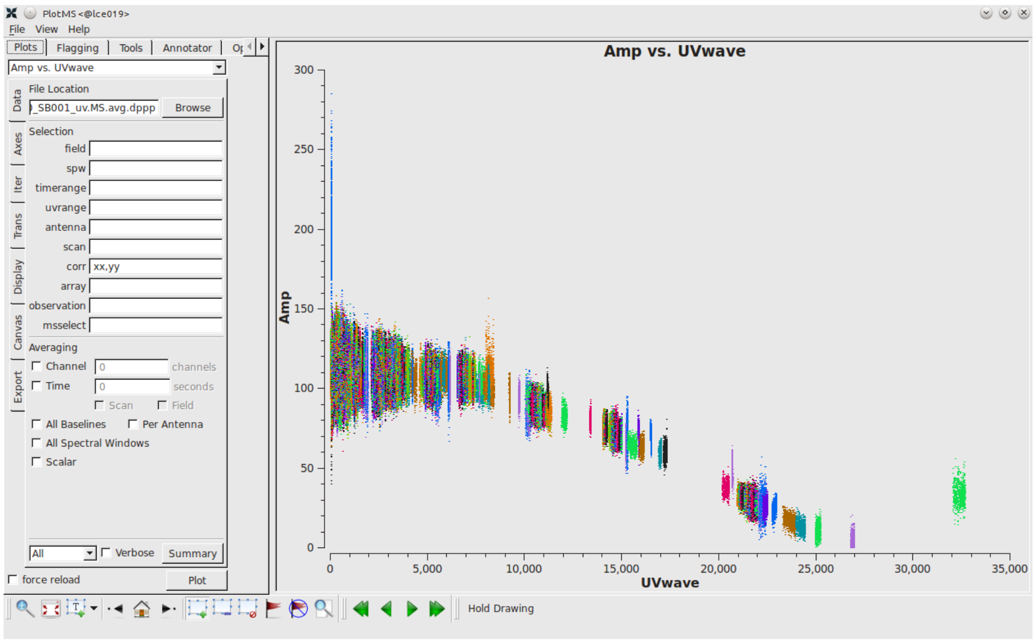
Amp vs. UVwave
X Axis: UVwave
Attach to: (●) Bottom ( ) Top
Range: (●) Automatic
Y Axis: Amp
Data Column: corrected
Attach to: (●) Left ( ) Right
Range: (●) Automatic

Amp vs. UVwave
File Location: _SB000_uv.MS.avg.dppp
antenna: !CS302HBA0
corr: xx,yy

Figure 48: SB000. Top: Plotting the visibility amplitude against UV distance. Bottom: Excluding antenna CS302HBA.

We will flag all of these bad baselines now with NDPPP. Leaving the msout field blank means NDPPP will update the flags in the input measurement set. The NDPPP parsets are:

```
> cp /globaldata/COOKBOOK/Tutorial/3c295/parsets/hba/NDPPP.flag.sb000.parset .
> cat NDPPP.flag.sb000.parset
msin = L74759_SAP000_SB000_uv.MS.avg.dppp
msin.startchan = 0
msin.nchan = 1
msin.datacolumn = DATA
```


Amp vs. UVwave

File Location: _SB001_uv.MS.avg.dppp

corr: xx,yy

Figure 49: SB001. Plotting the visibility amplitude against UV distance.

```
msout =
steps = [flag]
flag.type=preflagger
flag.baseline=[[RS508HBA&RS509HBA], [RS208HBA&RS509HBA], [CS302HBA*]]

> NDPPP NDPPP.flag.sb000.parset > ndppp.flag0.txt &
```

and likewise for the other subband:

```
> cp /globaldata/COOKBOOK/Tutorial/3c295/parsets/hba/NDPPP.flag.sb001.parset .
> cat NDPPP.flag.sb001.parset
msin = L74759_SAP000_SB001_uv.MS.avg.dppp
msin.startchan = 0
msin.nchan = 1
msin.datacolumn =
msout.datacolumn = DATA
steps = [flag]
flag.type=preflagger
flag.baseline=[[RS508HBA&RS509HBA], [RS208HBA&RS509HBA], [CS302HBA*]]

> NDPPP NDPPP.flag.sb001.parset > ndppp.flag1.txt &
```

Next we will re-do the calibration (it's probably a good idea after removing some bad data):

```
> calibrate-stand-alone -f L74759_SAP000_SB000_uv.MS.avg.dppp.flag bbs.parset \
  3C295TWO.skymodel > log.bbs.gaincal2_sb000 &
```

The amplitude against time for the flagged corrected data is plotted in 50.

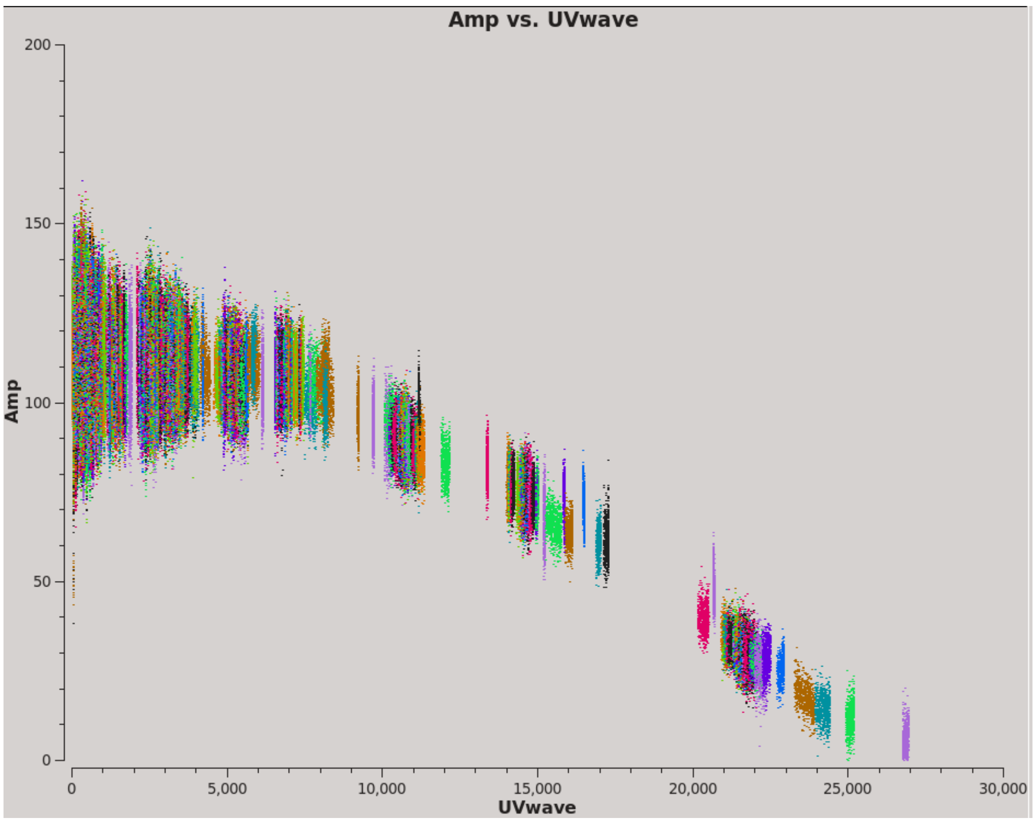


Figure 50: Plotting the visibility amplitude against UV distance for SB000 after flagging.

15.2.5 Imaging

Here we will use the AWimager[85] to do the deconvolution. While 3C 295 is the dominant source at the centre of the field we can actually image the large field and find other sources exploiting the wide-field imaging techniques built into the AWimager. The list of parameters along with a brief description of each can be shown with

```
> awimager -h
```

Note that the first time you run it, you are likely to get a "Cannot read table of Observatories". Should this happen, you can resolve the issue by correcting your ".casarc" file to look like this (there should be no space at the end on the line)

```
> cat  ˜/.casarc
measures.directory: /opt/cep/casacore/data
```

We will use a parameter file for awimager. At 120 MHz the LOFAR (NL Remote) field of view is 4.5 deg and the resolution should be around 8″ First, to make a dirty image, set niter to 0:

```
> cp /globaldata/COOKBOOK/Tutorial/3c295/parsets/hba/awimger.sb000.parms .
> cat awimger.sb000.parms
ms=L74759_SAP000_SB000_uv.MS.avg.dppp
image=L74759_SAP000_SB000.dirty.img
data=CORRECTED_DATA
weight=briggs
```

[85] see Chapter 8

```
robust=0
npix=4096
cellsize=5.5arcsec
padding=1.2
stokes=I
operation=mfclark
wmax=20000
UVmin=0.08
UVmax=18
niter=0
```

If you have run the imager before, it is advisable to remove any existing images:

```
> rm -rf L74759_SAP000_SB000.dirty.img*
```

To run awimager call

```
> awimager awimger.sb000.parms > log.awim.sb000 2>&1
```

this will take a few minutes to run. Note that in order to make a dirty image set “operation=mfclark” and “niter=0” so that it does 0 iterations of cleaning[86]. The last few lines of the output will look like (timestamps have been removed for clarity)

```
ImageSkyModel::makeApproxPSFs   bmaj: 30.3401", bmin: 23.855", bpa: 96.0219 deg
MFCleanImageSkyModel::solve     Final maximum residual = 99.094
MFCleanImageSkyModel::solve     Model 0: max, min residuals = 99.094, -16.3156 clean flux 0
imager::clean() Threshhold not reached yet.
imager::clean() Fitted beam used in restoration: 30.3401 by 23.855 (arcsec) at pa 96.0219 (deg)
Final normalisation
clean     162.27 real      401.19 user       8.74 system
awimager normally ended
```

AWImager produces a lot of images as output, including

```
L74759_SAP000_SB000.dirty.img.model          # uncorrected model image
L74759_SAP000_SB000.dirty.img.model.corr     # corrected model image
L74759_SAP000_SB000.dirty.img.restored       # restored (residual + convolved model) image
L74759_SAP000_SB000.dirty.img.residual       # residual image
L74759_SAP000_SB000.dirty.img.residual.corr  # corrected residual image
L74759_SAP000_SB000.dirty.img.psf            # point spread function
```

The image we wish to look at now is the “restored” image. This can be done with casaviewer:

```
> casaviewer L74759_SAP000_SB000.dirty.img.restored
```

the dirty image should look like a single strong point source convolved with the psf (See Fig. 51).

Now we will do some cleaning. If you look at George Heald’s beta noise calculator for LOFAR[87], with 21 core and 9 remote split HBA stations, we should expect a noise of a few mJy for an hour’s observation at 120 MHz. Initially though, we will just clean over the entire image down to a relatively high threshold of 0.1 Jy.

[86]setting “operation=image” does not work to make a dirty image with this version of the imager.
[87]http://www.astron.nl/~heald/test/sens.php

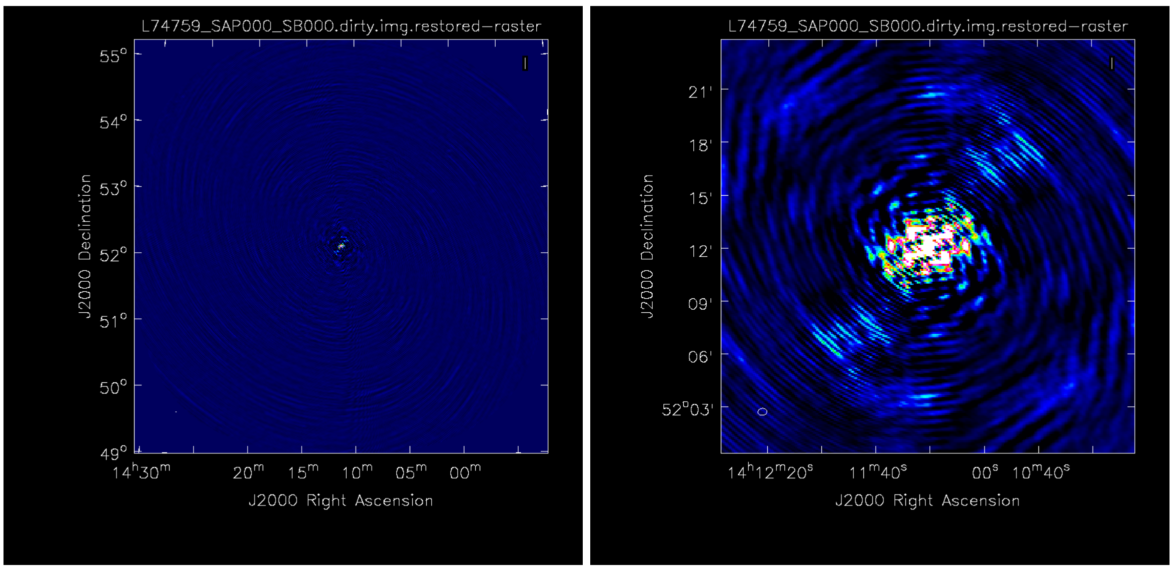

L74759_SAP000_SB000.dirty.img.restored−raster
J2000 Declination
J2000 Right Ascension
L74759_SAP000_SB000.dirty.img.restored−raster
J2000 Declination
J2000 Right Ascension

Figure 51: Left: The dirty image for 3C 295 (SB000). Right: Zoom in 4x. The data range is set to [-2, 20].

```
> cp /globaldata/COOKBOOK/Tutorial/3c295/parsets/hba/awimger.sb000.clean.parms .
> cat awimger.sb000.clean.parms
ms=L74759_SAP000_SB000_uv.MS.avg.dppp
image=L74759_SAP000_SB000.img
data=CORRECTED_DATA
weight=briggs
robust=0
npix=4096
cellsize=5.5arcsec
padding=1.2
stokes=I
operation=mfclark
wmax=20000
UVmin=0.08
UVmax=18
niter=1000

> rm -rf L74759_SAP000_SB000.clean.img*
> awimager awimger.sb000.clean.parms > log.clean.awim.sb000 2>&1
```

AWimager will run for $\sim$ 20 minutes and will produce a lot of output to the log file for each major cycle.

The output from AWimager consists of several images:

```
L74759_SAP000_SB000.img.model          # uncorrected dirty image
L74759_SAP000_SB000.img.residual       # residual image
L74759_SAP000_SB000.img.psf            # point spread function
L74759_SAP000_SB000.img.restored       # restored image
L74759_SAP000_SB000.img.restored.corr  # corrected restored image
L74759_SAP000_SB000.img.model.corr     # corrected model image
L74759_SAP000_SB000.img.residual.corr  # corrected residual image
```

Figure 52 shows the cleaned corrected image ("restored.corr"). One can see 3C 295 at the centre of the field and even though only 3C 295 was in our calibration model there are clearly about thirty other sources visible in the field.

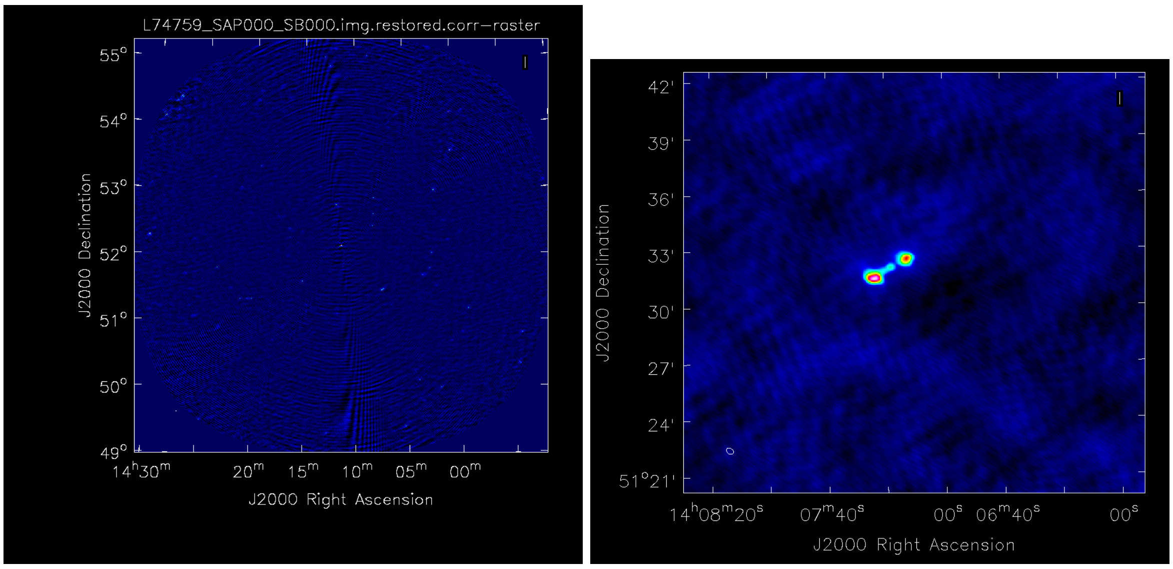


Figure 52: Left: The cleaned image for 3C 295 (SB000). Right: A zoom-in of the source in the lower right corner. The data range is set to [-2, 20].

15.2.6 Combining Measurement Sets

It is often useful to combine calibrated Measurement Sets for separate sub-bands into a single Measurement Set, both to allow faster processing in subsequent BBS runs and also to allow a single image of the combined data to be made.

```
> cp /globaldata/COOKBOOK/Tutorial/3c295/parsets/hba/NDPPP.combineMS.parset .
> cat NDPPP.combineMS.parset
msin = L74759_SAP000_SB*_uv.MS.avg.dppp
msin.datacolumn = DATA
msout = 3C295_BAND0.MS

steps = []
```

The wild card in line one of this very simple parset means that all sub-bands of the observation will be combined. Line two means that the DATA column from the input MSs will be written to the DATA column in the output. Here we only have two subbands to combine but the method works for many. If you do msoverview now you will see that there are 8 spectral channels. At this point you need to redo the calibration because we have copied the DATA column.

```
> calibrate-stand-alone -f 3C295_BAND0.MS bbs.parset
  3C295TWO.skymodel > log.bbs.gaincal 2>&1 &
```

You may wish to change "Solve.CellSize.Freq = 0" to 1 or 2 to solve for every channel or to combine every 2 channels. 3C 295 is bright enough that there is enough signal to do this.

Now we can make a single image of the combined data set:

```
> cp /globaldata/COOKBOOK/Tutorial/3c295/parsets/hba/awimger.band0.clean.parms .
> cat awimger.band0.clean.parms
ms=3C295_BAND0.MS
image=3C295_BAND0.img
data=CORRECTED_DATA
weight=briggs
robust=0
npix=4096
cellsize=5.5arcsec
padding=1.2
stokes=I
operation=mfclark
wmax=20000
UVmin=0.08
UVmax=18
niter=5000
threshold=0.1Jy
```

```
> rm -rf 3C295_BAND0.img*
> awimager awimger.band0.clean.parms > log.clean.awim.band0 2>&1
```

15.2.7 Subtraction of 3C 295

3C 295 is the dominant source at the center of the field. In order to image the rest of the field we will subtract it using BBS. In this specific case this is easy because we already have a very good model for this calibrator source so we will use that instead of making a model from the image.

We require a parset that includes a subtract step for the source 3C 295. We have already done a solve step to obtain our gain solutions so we will simply subtract the source using those solutions:

```
> cp /globaldata/COOKBOOK/Tutorial/3c295/parsets/hba/bbs_subtract3c295.parset .
> cat bbs_subtract3c295.parset
Strategy.ChunkSize = 500
Strategy.InputColumn = DATA
Strategy.TimeRange = []
Strategy.Baselines = *&
Strategy.Steps = [subtract]

Step.subtract.Operation = SUBTRACT
Step.subtract.Model.Sources = [3c295]
Step.subtract.Model.Beam.Enable = T
Step.subtract.Model.Beam.UseChannelFreq = T
Step.subtract.Model.Gain.Enable = T
Step.subtract.Model.Cache.Enable = T
Step.subtract.Output.Column = 3C295_SUBTRACTED
```

```
> calibrate-stand-alone 3C295_BAND0.MS  bbs_subtract3c295.parset > log.bbs.gaincalsubtract 2>&1 &
```

The Model.Gain is enabled in the subtract step to make sure we subtract 3C 295 with the appropriate solutions. Note the different call to calibrate-stand-alone. We no longer use the '-f' option as this expands to '–replace-parmdb', '–replace-sourcedb' which will cause your gain solutions in the instrument table to be overwritten. Moreover, we do not need to supply a skymodel as this is already contained in the sourcedb "3C295_BAND0.MS.sub/sky" from our last calibration. This should take ~ 10 minutes to run.

We will now copy the subtracted data to a new measurement set:

```
> cp /globaldata/COOKBOOK/Tutorial/3c295/parsets/hba/NDPPP.copysub.parset .
> cat NDPPP.copysub.parset
msin = 3C295_BAND0.MS
msin.datacolumn = 3C295_SUBTRACTED
msout = 3C295_BAND0.MS.sub
msout.datacolumn = DATA
steps = []

> NDPPP NDPPP.copysub.parset
```

Now, we still need to apply the gain solutions to the rest of the field and do a beam correction before imaging the CORRECTED_DATA. Remember that what is in the DATA column of this measurement set is the uncorrected data with 3C 295 subtracted out.

```
> cp /globaldata/COOKBOOK/Tutorial/3c295/parsets/hba/bbs_correct3c295.parset
> cat bbs_correct3c295.parset
Strategy.ChunkSize = 500
Strategy.InputColumn = DATA
Strategy.TimeRange = []
Strategy.Baselines = *&
Strategy.Steps = [correct]

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

> calibrate-stand-alone --parmdb 3C295_BAND0.MS/instrument 3C295_BAND0.MS.sub
bbs_correct3c295.parset > log.bbs.gaincalcorrect 2>&1 &
```

Here we need to specify where the solution table is (still in the old measurement set) and we do not need to give a sourcedb as we are only applying the solutions for the centre of the field.

The 3C 295-subtracted visibility amplitudes are plotted against time in 53.

We can make an image of the subtracted data

```
> cp /globaldata/COOKBOOK/Tutorial/3c295/parsets/hba/awimger.sub.clean.parms .
> cat awimger.sub.clean.parms
ms=3C295_BAND0.MS.sub
image=3C295_BAND0.sub.img
data=CORRECTED_DATA
weight=briggs
robust=0
npix=4096
cellsize=5.5arcsec
padding=1.2
stokes=I
operation=mfclark
```

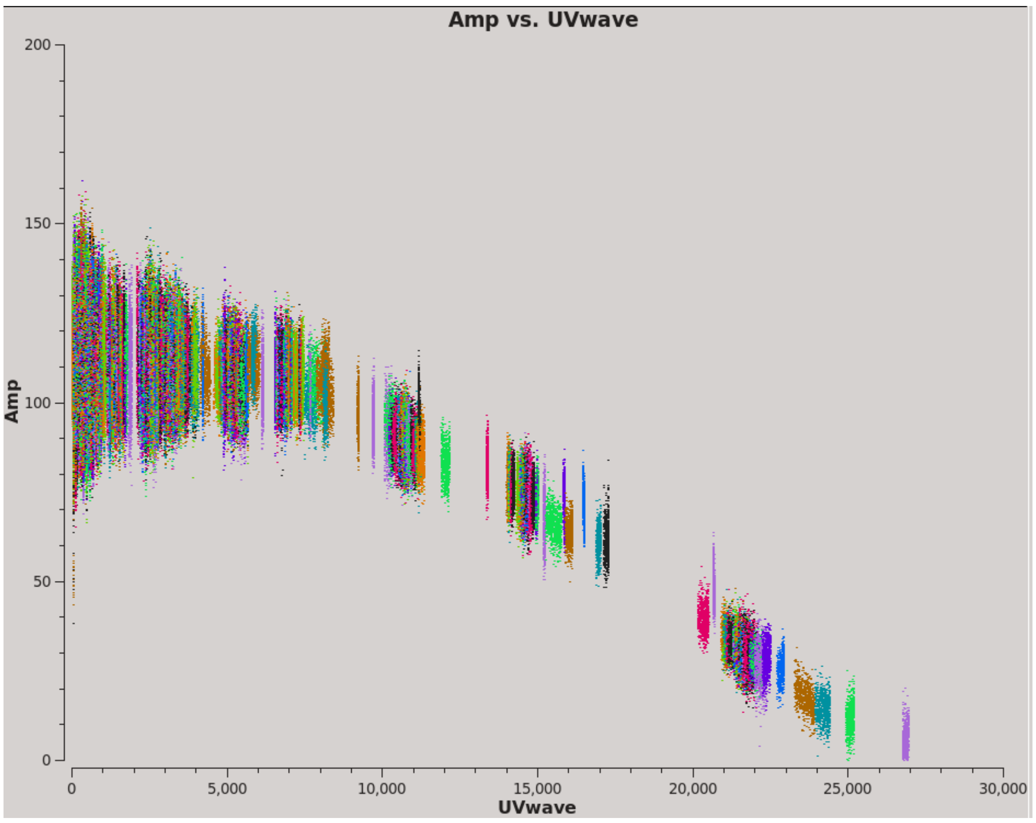


Figure 53: Plotting the visibility amplitude against UV distance after flagging.

```
wmax=20000
UVmin=0.08
UVmax=18
niter=1000
threshold=30mJy

> rm -rf 3C295_BAND0.sub.img*
> awimager awimger.sub.clean.parms > log.clean.awim.sub.band0 2>&1
```

15.3 LBA

The raw LBA data set for this exercise can be found in,

```
/globaldata/COOKBOOK/Tutorial/3c295/data/L74762/
```

The unique LOFAR observation number is L74762 and there are two sub-bands, SB000 and SB001. The data set is in Measurement Set (MS) format and the filenames are respectively

```
L74762_SAP000_SB000_uv.MS
L74762_SAP000_SB001_uv.MS
```

for the two sub-bands.

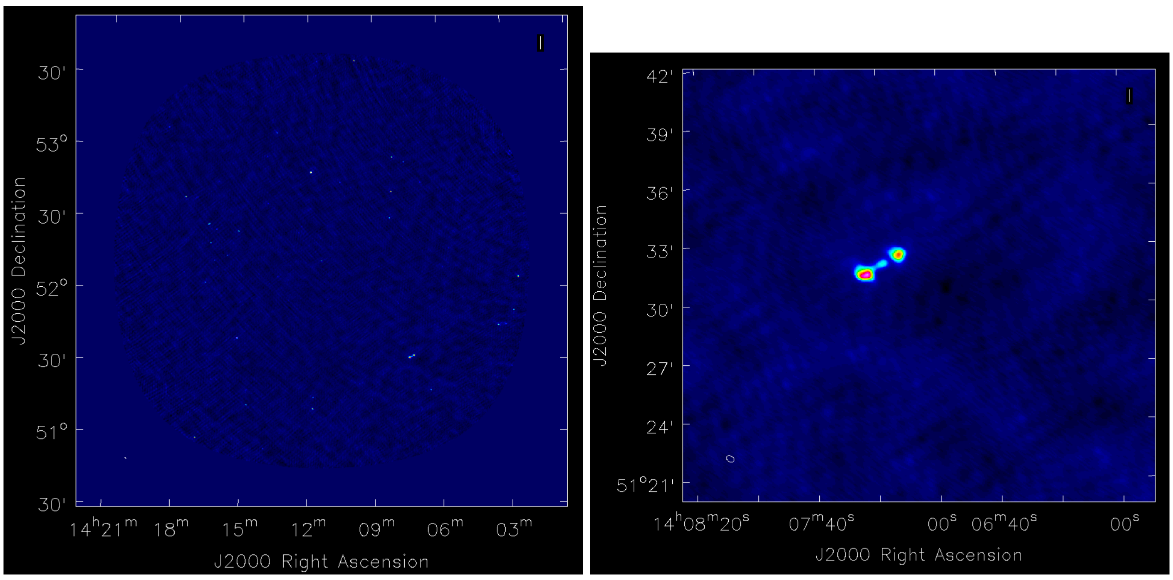


Figure 54: Left: The cleaned image for 3C 295 (SB000) after 3C 295 has been subtracted. Right: A zoom-in of the source in the lower right corner. The data range is set to [-0.1, 1].

15.3.1 Inspecting the raw data

Use msoverview again to find out the details of the observation (frequency, integration time, number of stations):

```
> msoverview in=L74762_SAP000_SB000_uv.MS verbose=T

msoverview: Version 20110407GvD
================================================================================
           MeasurementSet Name:  L74762_SAP000_SB000_uv.MS      MS Version 2
================================================================================
           This is a raw LOFAR MS (stored with LofarStMan)

   Observer: unknown     Project: 2012LOFAROBS
Observation: LOFAR
Antenna-set: LBA_OUTER

Data records: 1897632       Total integration time = 3597.99 seconds
   Observed from   12-Nov-2012/14:06:00.5   to   12-Nov-2012/15:05:58.5 (UTC)

Fields: 1
  ID   Code Name               RA            Decl           RefType
  0         BEAM_0             14:11:20.5167 +52.12.09.9276 J2000
   (nVis = Total number of time/baseline visibilities per field)

Spectral Windows:  (1 unique spectral windows and 1 unique polarization setups)
  SpwID  #Chans Frame Ch1(MHz)    ChanWid(kHz)TotBW(kHz)  Ref(MHz)    Corrs
  0          64 TOPO  59.4741821  3.05175781  195.3125    59.5703125  XX  XY  YX  YY

Antennas: 32:
  ID   Name  Station   Diam.    Long.         Lat.
  0    CS001LBALOFAR     86.0 m   +006.52.03.5  +52.43.34.0
  1    CS002LBALOFAR     86.0 m   +006.52.11.4  +52.43.47.4
  ...
  ...
  ...
  30   RS508LBALOFAR     86.0 m   +006.57.11.4  +53.03.19.2
  31   RS509LBALOFAR     86.0 m   +006.47.07.0  +53.13.28.2

The MS is fully regular, thus suitable for BBS
   nrows=1897632   ntimes=3594   nbands=1   nbaselines=528 (32 autocorr)
```

From this, you should see that 32 stations were used for this observation, that the observation was ~ 1 hour that there are 64 spectral channels and the frequency is 59.474 MHz for SB000 and 59.669 MHz for SB001. This gives a useful first look at the data, but we will take a closer look after the data have been converted from the raw correlator visibilities to a proper Measurement Set.

15.3.2 Flagging and demixing

As with the HBA, the data set is uncompressed and unflagged; the total size of each MS is 3.9 Gb. The data flagging and compression are carried out using NDPPP (see Chapter 5 for details). We will compress the sub-band to 1 channel in frequency and 10 s in time. Note that the limitation on the compression in time is set by the changes in the ionosphere. For LBA data it is almost always necessary to demix the data to remove the bright radio sources from the data. Demixing is described in detail in Chapter 6 and has been implemented in NDPPP. Usually this will be performed by the Radio Observatory but we include it here so you can learn how to do it.

To see which A-team sources need to be demixed use the plot_Ateam_elevation python script,

```
> plot_Ateam_elevation.py L74762_SAP000_SB000_uv.MS
```

the output of which is shown in Fig.55. From this we can see that CygA and CasA are over 40 deg elevation for the duration of the observation and should be demixed. They are also both about 60 deg away from the pointing centre (the distances of the A-team sources from the pointing centre are indicated in the legend).

The parset file for the flagging[88] and demixing should be copied to your working directory. Note that the demixing outputs the compressed data.

A sky model containing the sources to be demixed is also required

```
> cp -r /globaldata/COOKBOOK/Tutorial/3c295/models/Ateam_LBA_CC.sky .


> cp /globaldata/COOKBOOK/Tutorial/3c295/parsets/lba/NDPPP_LBA_preprocess.parset .
> cat NDPPP_LBA_preprocess.parset
msin = L74762_SAP000_SB000_uv.MS
msin.autoweight=TRUE
msin.datacolumn=DATA

msout = L74762_SAP000_SB000_uv.MS.dem.dppp
msout.datacolumn=DATA
```

[88]using the aoflagger algorithm.

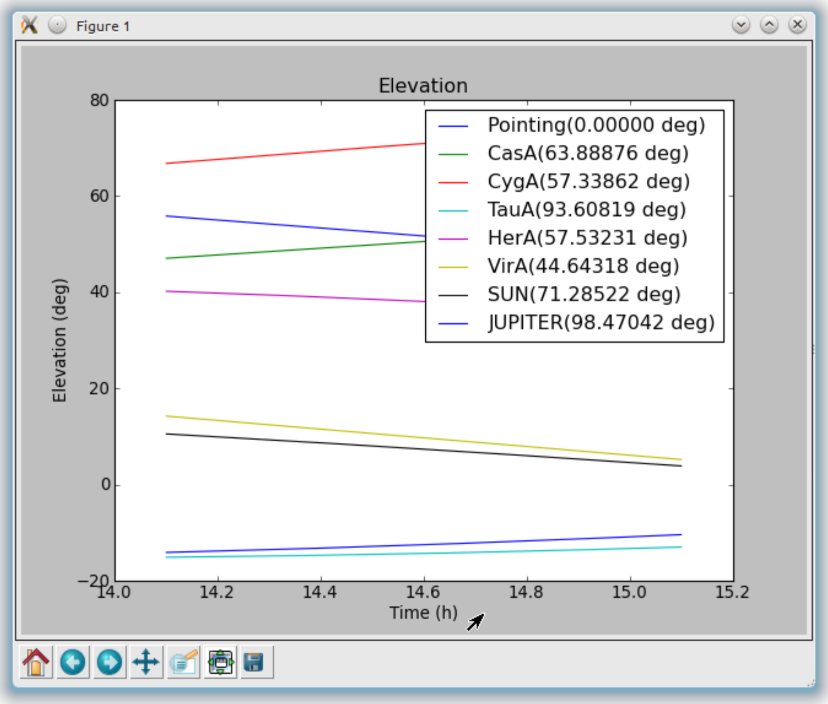


Figure 55: A-team elevation for the LBA observation.

```
steps=[preflagger0,preflagger1,aoflagger,demixer]

preflagger0.chan=[0,1,62,63]
preflagger0.type=preflagger

preflagger1.corrtype=auto
preflagger1.type=preflagger

aoflagger.autocorr=F
aoflagger.count.save=FALSE
aoflagger.keepstatistics=T
aoflagger.memorymax=0
aoflagger.memoryperc=0
aoflagger.overlapmax=0
aoflagger.overlapperc=-1
aoflagger.pedantic=F
aoflagger.pulsar=F
aoflagger.timewindow=0
aoflagger.type=aoflagger

demixer.freqstep=64     # compresses to 1 channel
demixer.timestep=10     # compresses 10 time-slots, i.e. 10s
demixer.skymodel=Ateam_LBA_CC.sky
demixer.subtractsources=[CasA, CygA]  # which sources to demix
demixer.type=demixer

> NDPPP NDPPP_LBA_preprocess.parset > log.ndppp.demix 2>&1 &
```

Depending on the use of the cluster, it will take about $\sim 10-12$ minutes to demix and flag the data (see the percentage progress bar). Inspecting the log file, you will see that the total data flagged for each of the flagging steps is 4.7%, 5.6% and 1.2% respectively.

Edit the msin and msout fields of the parset to do the same for the second sub-band.

The flagged and demixed data set should now be in your working directory and each MS should have a total size of 32 Mb, which is much more manageable than before. You can use msoverview to look at a summary of this data set using.

```
> msoverview in=L74762_SAP000_SB000_uv.MS.dem.dppp verbose=True
```

Some of the tasks that are used will make changes to the MS file, so let's make a copy of the compressed data set for safety,

```
> cp -rf L74762_SAP000_SB000_uv.MS.dem.dppp L74762_SAP000_SB000_uv.MS.dem.dppp.copy
```

15.3.3 Post-compression data inspection and flagging

We will use the CASA task plotms to inspect the data. Figure 56 shows the Amp. vs Time and Amp. vs UV distance (wavelengths) for SB000. We can see that there are a few short baselines with large fluctuating amplitudes. We will do some limited flagging for now.

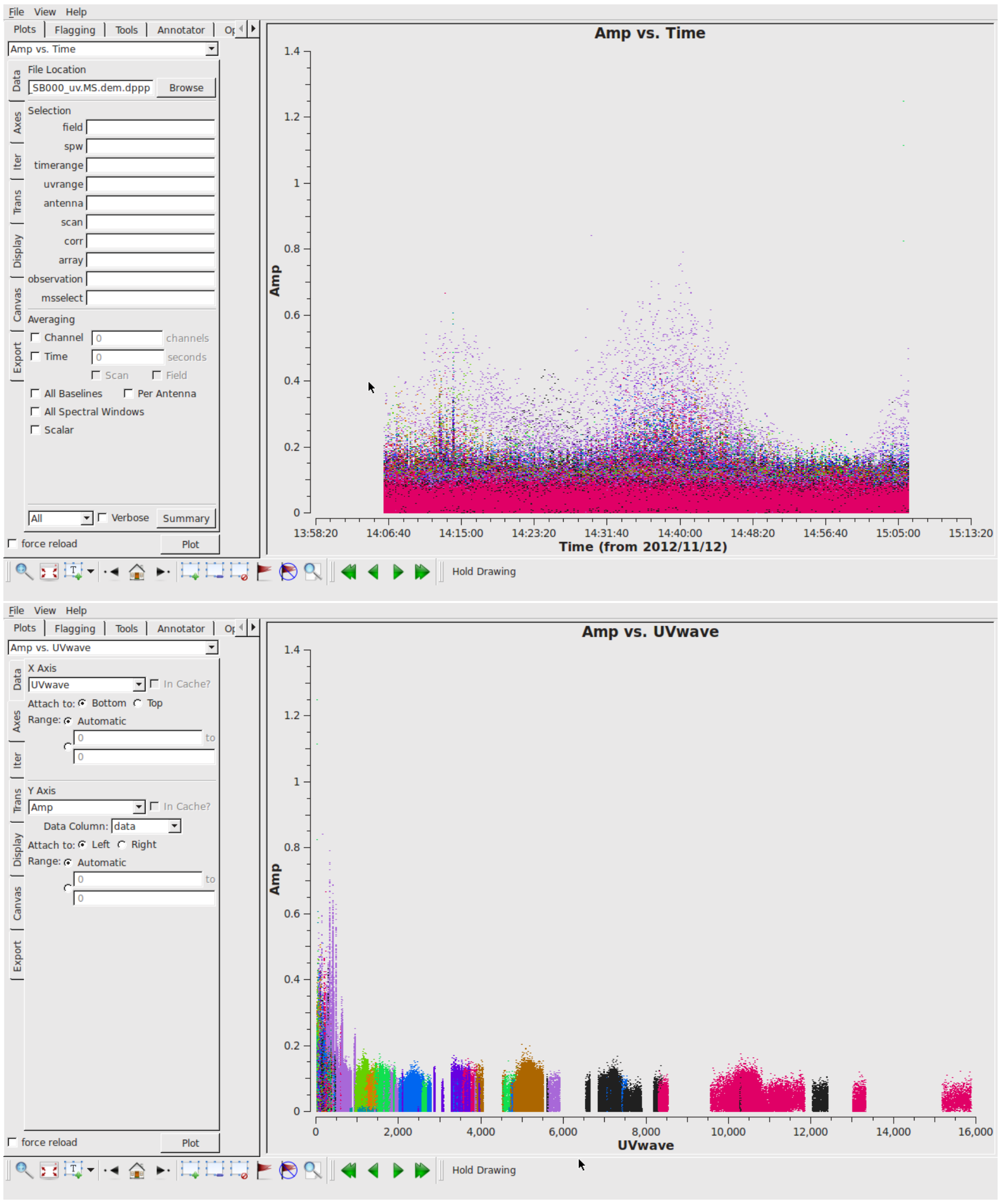
Plotting amplitude against baseline we see that the amplitudes for all baselines to RS305LBA are too low. Also CS302LBA seems to have unusually large amplitudes on most of its baselines. We will also clip the higher amplitudes (above 0.3).

```
> cp /globaldata/COOKBOOK/Tutorial/3c295/parsets/lba/NDPPP_LBA_flag1.parset .
> cat NDPPP_HBA_preprocess.parset
msin = L74762_SAP000_SB000_uv.MS.dem.dppp
msin.datacolumn=DATA

msout =

steps=[flag1, clip]

flag1.type=preflagger
flag1.baseline = [ [CS302LBA], [RS305LBA] ]

clip.type=preflagger
clip.amplmax=0.3
```

15.3.4 Combining Measurement Sets

Once you have done the initial flagging and demixing for the second subband, you can combine the two subbands

```
> cp /globaldata/COOKBOOK/Tutorial/3c295/parsets/lba/NDPPP.combineMS.parset .
> cat NDPPP.combineMS.parset
```



File Location: SB000_uv.MS.dem.dppp

X Axis: UVwave
Attach to: [✓ Bottom] Top
Range: [✓ Automatic]

Y Axis: Amp
Data Column: data
Attach to: [✓ Left] Right
Range: [✓ Automatic]

Figure 56: SB000. Top: Plotting the visibility amplitude against time. Bottom: Plotting the visibility amplitude against UV distance in wavelengths. Colourise by ‘Antenna2’ to obtain these colours.

```
msin = L74762_SAP000_SB00[01]_uv.MS.dem.dppp
msin.datacolumn = DATA
msout = 3C295_LBA_BAND0.MS

steps = []
```

15.3.5 Calibration with BBS

Here we will use the stand-alone version of BBS[89] to calibrate single sub-bands. The stand-alone version can be run using the following command

```
> calibrate-stand-alone -f <MS> <parset> <source catalog>
```

We use the same sky model as before:

```
> cp /globaldata/COOKBOOK/Tutorial/3c295/models/3C295TWO.skymodel .
> cat 3C295TWO.skymodel
# (Name, Type, Patch, Ra, Dec, I, ReferenceFrequency='150.e6', SpectralIndex) = format

, , 3c295, 14:11:20.64, +52.12.09.30
3c295A, POINT, 3c295, 14:11:20.49, +52.12.10.70, 48.8815, , [-0.582, -0.298, 0.583, -0.363]
3c295B, POINT, 3c295, 14:11:20.79, +52.12.07.90, 48.8815, , [-0.582, -0.298, 0.583, -0.363]
```

The parset file can be found at,

```
> cp /globaldata/COOKBOOK/Tutorial/3c295/parsets/lba/bbs.parset  .
> cat bbs.parset
Strategy.ChunkSize = 100
Strategy.Steps = [solve, correct]

Step.solve.Operation = SOLVE
Step.solve.Model.Sources = [3c295]
Step.solve.Model.Gain.Enable = T
Step.solve.Model.Beam.Enable = T
Step.solve.Model.Beam.UseChannelFreq = 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 = 4
Step.solve.Solve.CellChunkSize = 10
Step.solve.Solve.Options.MaxIter = 1000
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
```

[89]BBS is described in Chapter 7

```
Step.correct.Model.Sources = [3c295]
Step.correct.Model.Gain.Enable = T
Step.correct.Model.Beam.Enable = T
Step.correct.Model.Beam.UseChannelFreq = T
Step.correct.Output.Column = CORRECTED_DATA
```

This is a very simple parset file that solves and corrects the data. To run BBS, use the following command:

```
> calibrate-stand-alone -f 3C295_LBA_BAND0.MS bbs.parset 3C295TWO.skymodel \\
  > log.bbs.solve 2>&1  &
```

The calibration process should be completed in about 10 minutes. You can simultaneously run a similar command for the second sub-band.

When BBS is complete we can look at the calibrated data with parmdbplot.py. After de-selecting the "use resolution" option select a few stations and look at the solutions. Figures 57 and 58 shows some solution plots for SB000. It is clear that there are a few spikes in the solutions.

```
> python /globaldata/COOKBOOK/Tutorial/3c295/scripts/plot_solutions_all_stations.py -p -a \
3C295_LBA_BAND0.MS/instrument/ lba_sb000_gains
> display lba_sb000_gains_amp.png
> display lba_sb000_gains_phase.png
```

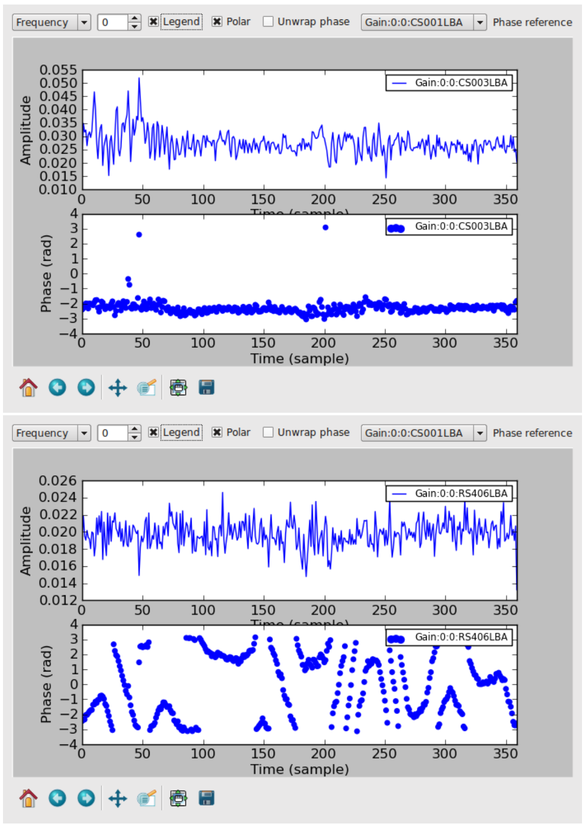


Figure 57: Top: The solutions for CS003LBA. Bottom: The solutions for RS406LBA.

Once again, we can inspect the corrected data with casaplotms. Figure 59 shows the corrected Amp. vs. Time plots for both sub-bands. From this we see again that there are some scans with bad

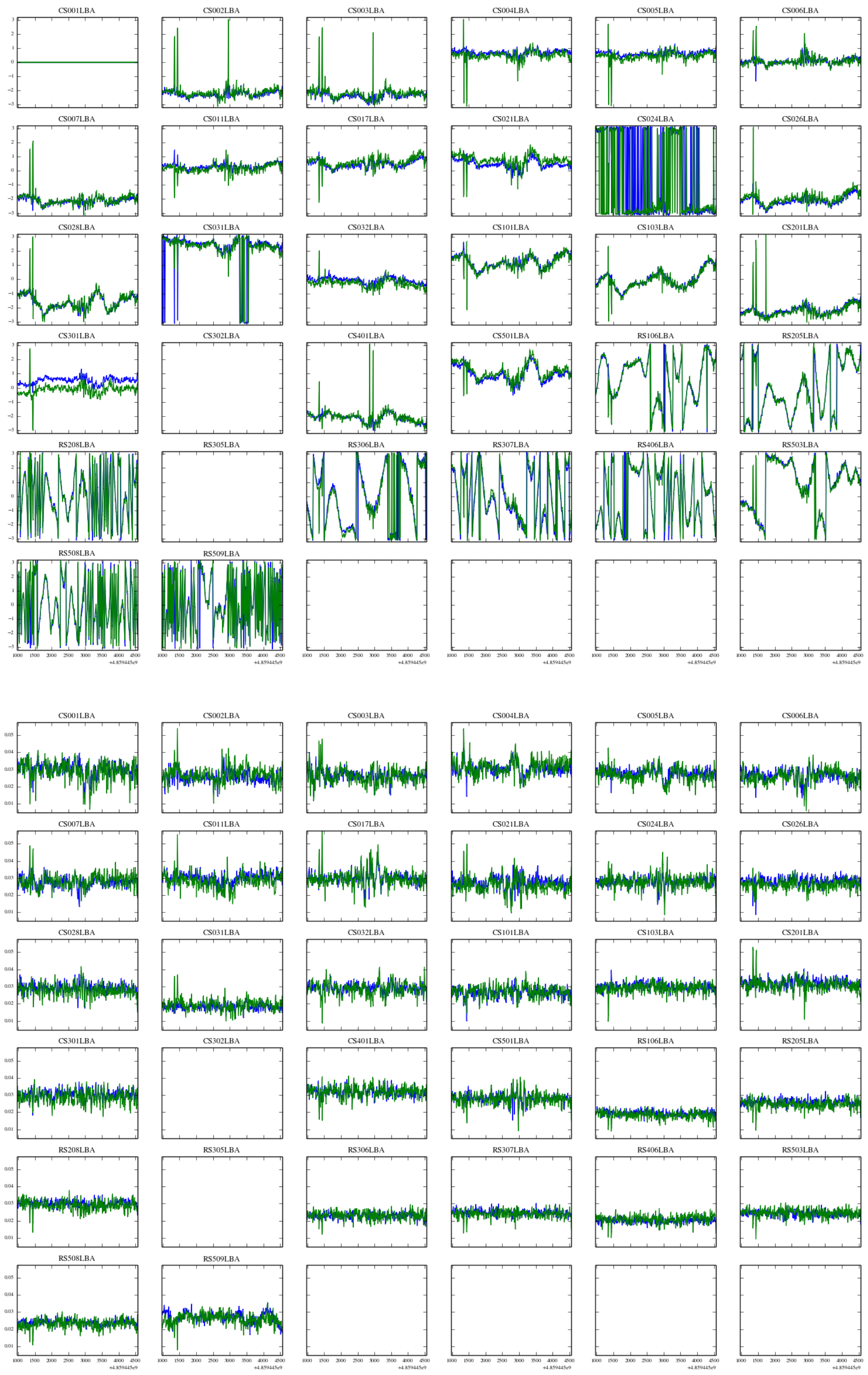


Figure 58: Top: Phase solutions for all stations for SB000 (polarizations in different colours). Bottom: Amplitude solutions for all stations for SB000.

solutions and there is a lot of scatter overall to high amplitudes. Now we will do some basic flagging with NDPPP using the aoflagger on the CORRECTED_DATA. Figure 60 shows the Amp. vs UV distance plots for both sub-bands after flagging.

```
> cp /globaldata/COOKBOOK/Tutorial/3c295/parsets/lba/NDPPP_LBA_flag.parset .
> cat NDPPP_HBA_preprocess.parset
msin = 3C295_LBA_BAND0.MS
msin.datacolumn=CORRECTED_DATA

msout =

steps=[aoflagger]

aoflagger.autocorr=F
aoflagger.timewindow=0
aoflagger.type=aoflagger

> NDPPP NDPPP_LBA_flag.parset > log.ndppp.flag 2>&1 &
```

15.3.6 Imaging

Here we will also use the AWimager[90] to do the deconvolution.

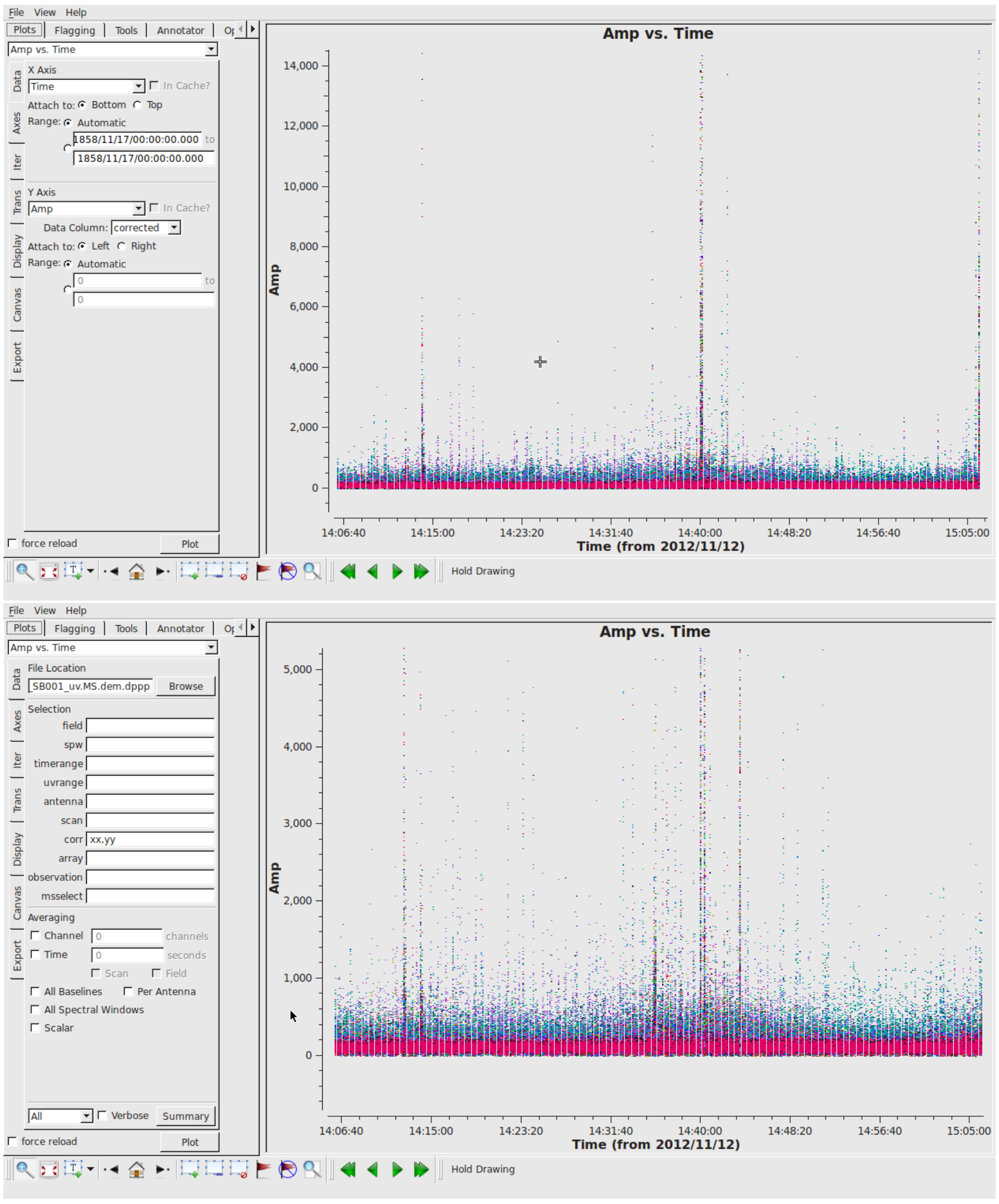
We will use a parameter file for awimager. At 60 MHz the LOFAR (NL Remote) field of view is 4.5 deg and the resolution should be around 8″ First, to make a dirty image:

```
> cp /globaldata/COOKBOOK/Tutorial/3c295/parsets/lba/awimger.b0.parms .
> cat awimger.b0.parms
ms=3C295_LBA_BAND0.MS
image=3C295_LBA_BAND0.dirty.image
data=CORRECTED_DATA
padding=1.2
cellsize=6arcsec
npix=2048
stokes=I
weight=briggs
robust=0.
operation=mfclark
niter=0

> awimager awimger.b0.parms
```

And then a cleaned image:

```
> cp /globaldata/COOKBOOK/Tutorial/3c295/parsets/lba/awimger.b0.clean.parms .
> cat awimger.b0.clean.parms
ms=3C295_LBA_BAND0.MS
```

[90]see Chapter 9



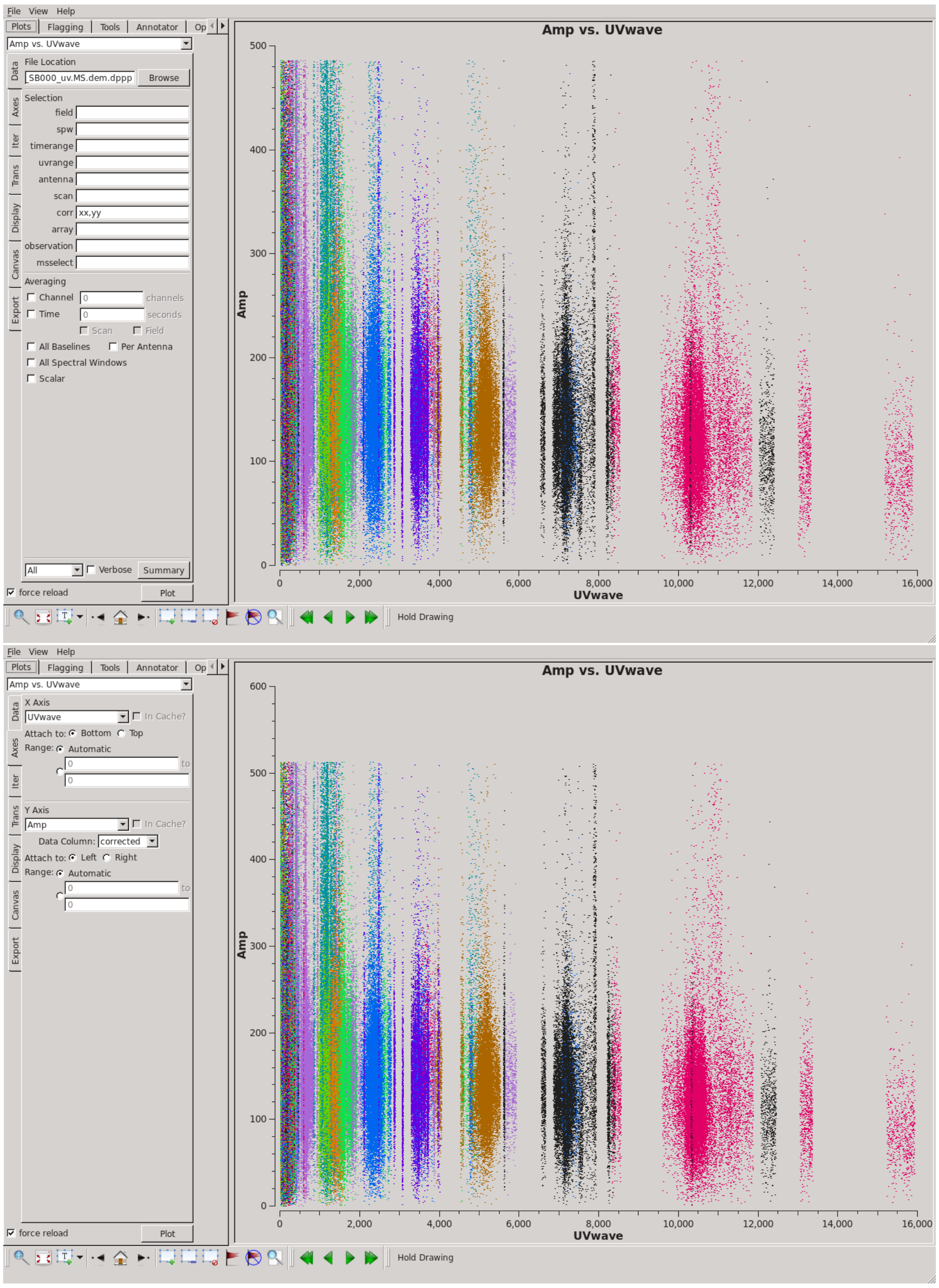
X Axis: Time
Attach to: Bottom
Range: Automatic
1858/11/17/00:00:00.000 to 1858/11/17/00:00:00.000

Y Axis: Amp
Data Column: corrected
Attach to: Left
Range: Automatic

File Location: SB001_uv.MS.dem.dppp

Selection
corr: xx,yy

Figure 59: Top: Plotting the visibility amplitude against time for SB000. Bottom: SB001.



Amp vs. UVwave

File Location
SB000_uv.MS.dem.dppp

Selection
field
spw
timerange
uvrange
antenna
scan
corr xx,yy
array
observation
msselect

Averaging
☐ Channel 0 channels
☐ Time 0 seconds
☐ Scan ☐ Field
☐ All Baselines ☐ Per Antenna
☐ All Spectral Windows
☐ Scalar

All ☐ Verbose

☑ force reload

X Axis
UVwave ☐ In Cache?
Attach to: [✓ Bottom] Top
Range: [✓ Automatic]
0 to
0

Y Axis
Amp ☐ In Cache?
Data Column: corrected
Attach to: [✓ Left] Right
Range: [✓ Automatic]
0 to
0

☑ force reload

Figure 60: Top: Plotting the visibility amplitude against UV distance for SB000 after flagging. Bottom: SB001.

```
image=3C295_LBA_BAND0.dirty.image
data=CORRECTED_DATA
padding=1.2
cellsize=6arcsec
npix=2048
stokes=I
weight=briggs
robust=0.
operation=mfclark
niter=0
niter=500
threshold=0.5Jy

> awimager awimger.b0.clean.parms
```

Figure 61 shows the dirty image and Fig. 62 shows the cleaned corrected image.

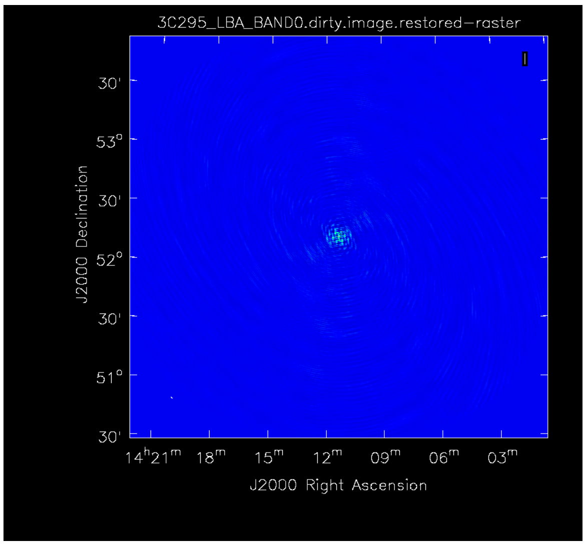


Figure 61: The dirty LBA image for 3C 295 (SB000).

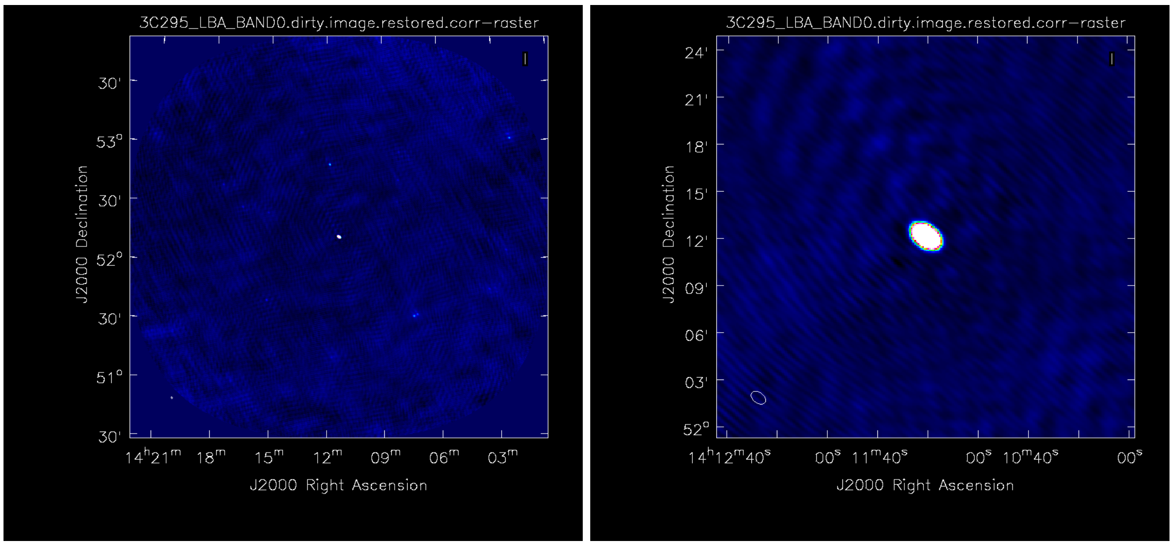


Figure 62: The cleaned LBA image for 3C 295. The scale is set to [-1,10].

16 Useful resources[91]

16.1 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`
Essential pages on the wiki are:

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

DPPP: `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`

16.2 Useful analysis scripts

A compilation of some practical python scripts is available at the LOFAR-Contributions GitHub repository:

`https://github.com/lofar-astron/LOFAR-Contributions`

The scripts provided are[92]:

- 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
- 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

[91]This Chapter is maintained by R. F. Pizzo, `pizzo[at]astron[dot]nl`
[92]If you have other scripts that could be useful for other commissioners, please contact Roberto A. Shulevski at `shulevski[at]astron[dot]nl`

- uvcoverage.py: plots the uv coverage for a Measurement
- plot_flags.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. DPPP.
- 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.
- fixlofaruvw.py: corrects the faulty UVW column header. Use this on all data sets recorded before 20/03/2011 to get the astrometry correct. This script changes the MEASINFO.Ref label in the UVW column to J2000.
- plot_Ateam_elevation.py: it makes plots of the elevation and angular distance of the Ateam and other sources (Sun, Jupiter) given a Measurement Set.

- do_demixing.py: applies the demixing routine from Bas vdTol to the data to get rid of the A-team sources. The instructions are at the top of the file.
- CutBeamFromSkyModel.py: given a skymodel, it produces two sub-skymodels, the first containing all the components within a particular radius from a given coordinate, the second all the rest.
- modskymodel.py: it can shift skymodels by a given angular amount. It can manipulate skymodels also in other ways, like masking them and updating their spectral index values.
- listr_v2.py: it is a clone of the old AIPS matrix listing of data files. For the data or corrected-data column, it lists amplitudes (or phases) averaged by baseline over a specified time interval. It does also cross-hands and identifies the antennas.
- fromsky.py: it converts a BBS skymodel file into the MODEL_DATA column of a visibility dataset.
- flagnancorrected.py: it searches CORRECTED_DATA column for NaN and flags them.
- flagnandata.py: it searches DATA column for NaN and flags them.
- Solution_Plotter.py: it plots amplitude, phase solutions per antenna and the differential TEC on a baseline.
- skymodel_to_ds9reg.py: it plots the output of gsm.py with ds9.

16.3 Contact points

Some key contact points are listed below:

- LOFAR Imaging Cookbook - Aleksandar Shulevski (shulevski[at]astron[dot]nl)
- DPPP - Ger van Diepen (diepen[at]astron[dot]nl), Tammo Jan Dijkema (dijkema[at]astron[dot]nl, and David Rafferty (rafferty[at]strw[dot]leidenuniv[dot]nl)
- AOFlagger - André Offringa (offringa[at]astron[dot]nl)
- BBS - Tammo Jan Dijkema (dijkema[at]astron[dot]nl), Vishambhar Nath Pandey (pandey[at]astron[dot]nl)
- AWImager - Tammo Jan Dijkema (dijkema[at]astron[dot]nl), and Bas van der Tol (tol[at]astron[dot]lnl)
- Selfcal - Nicolas Vilchez - (vilchez[at]astron[dot]nl) and George Heald (heald[at]astron[dot]nl)
- Pyrap/CASA - Ger van Diepen (diepen[at]astron[dot]nl)
- SAGECAL, Shapelets - Sarod Yatawatta (yatawatta[at]astron[dot]nl)
- PyBDSM, LSMTool - David Rafferty (rafferty[at]strw[dot]leidenuniv[dot]nl)
- RM Synthesis - Marco Iacobelli (iacobelli[at]astron[dot]nl)

16.4 Commissioning reports

The development of software for LOFAR data reduction is favor by the commissioning activities, which push forward the improvement of the instrument. All the reports of the tests performed by the commissioners on the different aspects of the LOFAR reduction softwares are available on the LOFAR Wiki[93].

[93] http://www.lofar.org/operations/doku.php?id=commissioning:busy_wednesdays

17 Acknowledgments

To this cookbook many commissioners and software developers have contributed. As the routines, the hardware, and the software needed for LOFAR develop very quickly, what is reported in this manual might be sometimes incorrect. We try to keep it as up to date as possible, but we surely need your feedback to improve its quality. Please send comments and suggestions of improvements to Aleksandar Shulevski (shulevski[at]astron[dot]nl) using the LOFAR issue tracker[94].

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
- Version 1.2: Annalisa Bonafede
- Version 2.0: Emanuela Orru' & Fabien Batejat
- Version 2.1: Roberto Francesco Pizzo
- Version 2.2: Roberto Francesco Pizzo
- Version 2.3: Roberto Francesco Pizzo
- Version 3.0: Roberto Francesco Pizzo
- Version 4.0: Roberto Francesco Pizzo
- Version 5.0: Roberto Francesco Pizzo
- Version 5.1: Roberto Francesco Pizzo
- Version 6.0: Roberto Francesco Pizzo, Laura Birzan, Ger van Diepen, Sven Duscha, John McKean, André Offringa, David Rafferty, Cyril Tasse, Bas van der Tol, Reinout van Weeren, and Joris van Zwieten
- Version 7.0: Roberto Francesco Pizzo, Laura Birzan, Ger van Diepen, Sven Duscha, John McKean, André Offringa, David Rafferty, Cyril Tasse, Bas van der Tol, Reinout van Weeren, Sarod Yatawatta, and Joris van Zwieten
- Version 8.0: Roberto Francesco Pizzo, Laura Birzan, Ger van Diepen, Sven Duscha, John McKean, André Offringa, David Rafferty, Aleksandar Shulevski, Cyril Tasse, Bas van der Tol, Reinout van Weeren, Sarod Yatawatta , and Joris van Zwieten
- Version 9.0: Roberto Francesco Pizzo, Laura Birzan, Ger van Diepen, Sven Duscha, Geaorge Heald, John McKean, André Offringa, David Rafferty, Cyril Tasse, Bas van der Tol, Reinout van Weeren, Sarod Yatawatta , and Joris van Zwieten
- Version 10.0 - 12.0: Roberto Francesco Pizzo, Laura Birzan, Ger van Diepen, Sven Duscha, George Heald, John McKean, André Offringa, Emanuela Orrú, David Rafferty, Cyril Tasse, Bas van der Tol, Reinout van Weeren, Sarod Yatawatta , and Joris van Zwieten

[94]https://proxy.lofar.eu/redmine

- Version 13.0: Roberto Francesco Pizzo, Ger van Diepen, Tammo Jan Dijkema, John McKean, André Offringa, Emanuela Orrú, David Rafferty, Cyril Tasse, Bas van der Tol, Valentina Vacca, Reinout van Weeren, Wendy Williams, Sarod Yatawatta , and Joris van Zwieten
- Version 14.0: Roberto Francesco Pizzo, Ger van Diepen, Tammo Jan Dijkema, G. Heald, Francesco de Gasperin, John McKean, Maaijke Mevius, André Offringa, Emanuela Orrú, David Rafferty, Cyril Tasse, Bas van der Tol, Valentina Vacca, Nicolas Vilchez, Reinout van Weeren, Wendy Williams and Sarod Yatawatta,
- Version 15.0: Roberto F. Pizzo, Ger van Diepen, Tammo Jan Dijkema, G. Heald, Francesco de Gasperin, M. Iacobelli, John McKean, Maaijke Mevius, André Offringa, Emanuela Orrú, David Rafferty, Cyril Tasse, Bas van der Tol, Valentina Vacca, Nicolas Vilchez, Reinout van Weeren, Wendy Williams and Sarod Yatawatta
- Version 16.0: Roberto F. Pizzo, Ger van Diepen, Tammo Jan Dijkema, G. Heald, Francesco de Gasperin, M. Iacobelli, John McKean, Maaijke Mevius, André Offringa, Emanuela Orrú, David Rafferty, Cyril Tasse, Bas van der Tol, Valentina Vacca, Nicolas Vilchez, Reinout van Weeren, Wendy Williams and Sarod Yatawatta
- Version 17.0: Roberto F. Pizzo, Ger van Diepen, Tammo Jan Dijkema, G. Heald, Francesco de Gasperin, M. Iacobelli, John McKean, Maaijke Mevius, André Offringa, Emanuela Orrú, David Rafferty, Cyril Tasse, Bas van der Tol, Valentina Vacca, Nicolas Vilchez, Reinout van Weeren, Wendy Williams and Sarod Yatawatta
- Version 18.0: Roberto F. Pizzo, Ger van Diepen, Tammo Jan Dijkema, G. Heald, Francesco de Gasperin, M. Iacobelli, John McKean, Maaijke Mevius, André Offringa, Emanuela Orrú, David Rafferty, Cyril Tasse, Bas van der Tol, T. J. Dijkema, Valentina Vacca, Nicolas Vilchez, Reinout van Weeren, Wendy Williams and Sarod Yatawatta

A GNU screen

For some long running processes, as the flagging (DPPP, see Sect. 5), the calibration (BBS, see Sect. 7) 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 accidentally 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 DPPP, 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. In particular, if you want to dump out what is in screen's buffer (not only what you see on the screen, but also whatever is in its scrollback buffer), you can:

- press Control-a
- type ":hardcopy -h output.txt" (output.txt can be whatever you want)

Then you can open `output.txt` with a text editor to have a look at what screen had in its memory. To make screen remember more lines, you can edit `/.screenrc` so that the line containing "defscrollback" says something like

```
defscrollback 10000
```

B LOFAR simulation software and new Demixing approach[95]

The A-team or other bright sources can strongly affect the observation of nearby targets. To understand which is the best strategy to adopt in order to deal with these strong sources, it is useful to make the user aware in advance about their effect on the observed target when the observations will be performed. In this context a *LOFAR simulation software*[96] has been developed. The software package is composed of two Python scripts, one to simulate a mock data set (`simulate.py`) and the other to compare simulations and observations (`compare.py`). They can be found at

```
 /opt/cep/tools/simulation/
```

All the dependencies of the software are installed in the LOFAR cluster CEP3. To source the scripts type

```
> use Lofar
```

and

```
> use Cookbook
```

By using these scripts, the user can simulate two mock Measurement Sets containing respectively the target and A-team sources, and compare them in order to understand which percentage of the data will be affected by the bright sources.

B.1 Simulating a Measurement Set

The simulation of the data is performed through the script `simulate.py`. The script includes three main blocks generating commands and parsets that the user can run in order to:

1. create an empty mock Measurement Set with the same starting time, ending time, and pointing as the observations, by using `makems`;
2. update the calibration tables in the Measurement Set with the information to compute the beam of the instrument, by using `makebeamtables`;
3. simulate the effect of a model of sources on the data, by using BBS.

If necessary, commands and parsets can be modified by the user before to be run.

When launching the script the user can use the following options:

`--ra=RA`	right ascension of the field
`--dec=DEC`	declination of the field
`--time=TIME`	start date and time of the observation in the format YYYY/MM/DD/hh:mm:ss.s
`--n-time=N_TIME`	duration of the observation in integer multiples of 10 s (the default value is 1800 corresponding to 5 h)
`--name=NAME`	name of the simulated field and prefix of the output files

[95]The author of this appendix is Valentina Vacca (vvacca[at]mpa-garching[dot]mpg[dot]de).

[96]This software has been developed by Jose Sabater (`jsm[at]roe[dot]ac[dot]uk`).

`--source=SOURCE1,SOURCE2,...`	name of the source(s) to simulate
`--sky-model=SKY_MODEL`	skymodel of the source(s) to simulate
`--path=PATH`	path to store the parsets and Measurement Sets of the simulation
`--overwrite`	overwrite existing parsets and Measurement Sets
`--lba`	simulate a LBA observation (the default is HBA)
`--antenna-conf=ANTENNA_CONF`	antenna configuration (the default antenna configurations are HBA_DUAL_INNER for HBA and LBA_OUTER for LBA)
`--h` or `--help`	shows the usage of the script

The format for time, right ascension, and declination is the same as used in the parset. In particular, we note that declination uses dots instead than colons. At the moment it is possible to insert in the simulation only the sources 3C53 and 3C237, and the A-team sources CygA, CasA, CasA4, TauA, VirA, VirA4, HydraA[97]. The default location of the files produced when launching the script is

```
/data/scratch/<user_name>/simulation/
```

The present version of the software creates the working directory `simulation/` and the parsets, while the commands are only displayed on the screen and have to be executed by the user. In the future release the commands will be directly executed by the script and other antenna configurations will be allowed.

B.1.1 Example

As an example, we run `simulate.py` to produce a mock observation in the direction of 3C 299 ($RA = 14h21m05.88s$ and $DEC = 41°44'49.490''$), with a total observational time ~ 2 h starting at 04:16:02.5 in date 14 April 2013 and we will investigate the effect of Cygnus A on the target source:

```
python /opt/cep/tools/simulation/simulate.py --ra 14:21:05.88 \
--dec 41.44.49.490 --time 2013/04/14/04:16:02.5 --n-time 720 \
--name 3C299 --source  CygAGG --sky-model \
/globaldata/COOKBOOK/Models/Ateam_LBA_CC.skymodel --overwrite
```

The script creates the folder `simulation/` in the default directory `/data/scratch/<user_name>/` containing the parsets

```
makems_3C299_HBA.parset
predict_CygAGG.parset
```

and gives as an output on the screen the following commands that the user has to run to produce the mock Measurement Set

```
cd /data/scratch/<user_name>/simulation/;
makems /data/scratch/<user_name>/simulation/makems_3C299_HBA.parset;
makebeamtables antennaset=HBA_DUAL_INNER ms=20130414_3C299_HBA.MS \
```

[97]Models for these sources are available e.g. in `/globaldata/COOKBOOK/Models/Ateam_LBA_CC.skymodel`. Note: when CasA and VirA are used the simulation is very slow.

```
overwrite=True
cp -r /data/scratch/<user_name>/simulation/20130414_3C299_HBA.MS \
/data/scratch/<user_name>/simulation/20130414_3C299_HBA_CygAGG.MS
cd /data/scratch/<user_name>/simulation/;
(date; calibrate-stand-alone -f \
/data/scratch/<user_name>/simulation/20130414_3C299_HBA_CygAGG.MS \
predict_CygAGG.parset \
/globaldata/COOKBOOK/Models/Ateam_LBA_CC.skymodel; \
date) | tee log_3C299_CygAGG.txt
```

which is one long command split over multiple lines (splitting indicated by \).
This output contains all the instructions the user has to follow to produce the mock Measurement Set:

1. move to the working directory `simulation/`
2. run `makems` with the parset `makems_3C299_HBA.parset` to create the mock Measurement Set `20131304_3C299_HBA.MS`
3. run `makebeamtables` on the mock Measurement Set to compute the station beam model
4. change the name of the Measurement Set from `20131304_3C299_HBA.MS` to `20131304_3C299_HBA_CygAGG.MS`
5. move again to the working directory `simulation/` if you are not there anymore
6. run BBS on the Measurement Set by using the parset `predict_CygAGG.parset` to predict the effect of the A-team source(s) on the target. At the moment no default sky model is present, therefore the user has to specify one. The output on the screen of BBS plus the `date` before and after BBS was running are redirected to the file `log_3C299_CygAGG.txt`. With these parameters BBS takes ~5 m to run.

B.2 Comparing observations and simulations

Once the simulation has been produced, the user can investigate the effect of the bright sources on the target field with the script `compare.py`. The code calculates the percentage of data points affected by the A-team for the XX and YY polarization products, according to two possible criteria:

1. the user fixes a flux `threshold` and all the data points with flux larger than this threshold are considered affected. In this case the user needs only a mock Measurement Set containing the A-team sources or other bright sources that can affect the data;
2. the user fixes a `level` and compares the flux of the bright source(s) (S_A) with the flux of a central source in the target field (S_T). If $S_\mathrm{A}/S_\mathrm{T} \geq$ `level`, the corresponding visibility will be considered affected by the bright source(s). In this case the user needs a mock Measurement Set containing the target source (alternatively, if the observations have been already performed, the observed data can be used) and a mock Measurement Set containing the A-team sources.

When launching the script on the mock Measurement Set containing A-team/bright sources that can affect the observation of a target source of interest, the user can choose among the following options:

`--s` or `--source=SOURCE.MS`	simulated Measurement Set of the central source

	(alternatively the real observation can be used)
`--t` or `--threshold=T1, T2, ...`	threshold(s) (the default value is 5 Jy). A different threshold for each bright source can be specified
`--l` or `--level=L1, L2, ...`	ratio(s) between the flux of an A-team/bright source S_A and of the target source S_T. A different level for each A-team/bright source can be specified
`-f` or `--stat-function=STAT_FUNCTION`	statistical function used for the threshold/level. Possible options are: max, min, median, std, mean (the default function is median)
`--all-correlations=CORRELATIONS`	correlations considered (not working yet)
`-p` or `--plot`	produce plots
`--h` or `--help`	shows the usage of the program

As an output a percentage of the amount of data above the threshold or above the level selected by the user (and therefore affected by A-team/bright sources) at different times is given for all channels and for XX and YY correlation products. This percentage is calculated according the statistical function selected among the options.

Due to a bug in the Python module `argparse`, if the options `--t` or `--l` are used, they can not be followed by the name of the Measurement Set containing the A-team sources. In this case the name of the Measurement Set has to be placed before these options, just after `compare.py`. Nevertheless, they can be followed by any other option.

At the moment the code does not allows the simultaneous comparison of multiple bright sources. This option will be available in a future release.

B.2.1 Example

As an example, we run `compare.py` on the simulation produced with `simulation.py` at the step before by using the option `--t`:

```
python  /opt/cep/tools/simulation/compare.py \
/data/scratch/vacca/LSS/simulation/20130414_3C299_HBA_CygAGG.MS \
--t 4 -f "median"
```

A combination of the options `--t` and `--l` is also possible.

The script gives as an output on the screen:

```
Successful readonly open of default-locked table \
/data/scratch/<user_name>/simulation/20130414_3C299_HBA_CygAGG.MS: \
25 columns, 1274400 rows
=================================
Flux density threshold: 4.000000
Freq:  0 XX:   2.98% YY:   2.86%
Freq:  1 XX:   2.81% YY:   2.81%
Freq:  2 XX:   0.00% YY:   0.00%
Freq:  3 XX:   0.00% YY:   0.00%
Freq:  4 XX:   0.12% YY:   0.06%
```

```
Freq:  5 XX:   0.00% YY:   0.00%
Freq:  6 XX:   0.12% YY:   0.12%
Freq:  7 XX:   0.00% YY:   0.00%
Freq:  8 XX:   0.06% YY:   0.06%
Freq:  9 XX:   0.12% YY:   0.06%
Freq: 10 XX:   0.88% YY:   0.64%
Freq: 11 XX:   0.94% YY:   0.82%
Freq: 12 XX:   1.46% YY:   1.34%
Freq: 13 XX:   1.29% YY:   1.29%
```

The script takes a few seconds to run. During the observation, 10% of the data will be affected by Cygnus A above a threshold of 4 Jy.

B.3 New demixing algorithm

When the A-team or other bright sources are present for the most of the observational time, the best strategy to eliminate them from the visibilities is to apply the standard demixing procedure that consists on the subtraction of the A-team sources from the entire observation (see Sect. 5.1.10). On the contrary, when long HBA observations are performed, A-team sources can be visible just for a fraction of the observation (see for example Fig. 63) and therefore they can just partially affect the data set. In this case their subtraction from the all observation can be dangerous and it is better to adopt the alternative strategy[98] described in the following.

B.3.1 Predict

When observations are already available the user does not need to simulate a mock observation but only to simulate the effect of A-team sources on the target. In this case, the first step consists in simulating the A-team sources (VirA, CygA, CasA, TauA) during the observations of interest. Through the stand-alone version of BBS (described in Chapter 7), the data set can be filled with a sky model containing the A-team sources (VirA, CygA, CasA, TaurA):

```
calibrate-stand-alone -f dataset.MS predict.parset skymodel.skymodel
```

where

1. `data set.MS` is the observed data set;
2. `predict.parset` is a parset aiming at simulating data according to a given sky model;
3. `skymodel.skymodel` is the file containing the model of the A-team sources.

An example of the parset to be used is

```
Strategy.InputColumn = DATA
Strategy.ChunkSize = 300
Strategy.UseSolver = F
Strategy.Steps = [predict4]
```

[98]This procedure has been developed by Reinout van Weeren (`rvanweeren[at]cfa[dot]harvard[dot]edu`).

```
Step.predict4.Model.Sources = [VirA_4_patch,CygAGG,CasA_4_patch,TauAGG]
Step.predict4.Model.Cache.Enable    = T
Step.predict4.Model.Gain.Enable     = F
Step.predict4.Operation             = PREDICT
Step.predict4.Output.Column         = MODEL_DATA
Step.predict4.Model.Beam.Enable     = True
```

and an example of a skymodel for LBA observations can be found at

```
/globaldata/COOKBOOK/Models/Ateam_LBA_CC.skymodel
```

The simulation should require about 1 h (for a 10 h long observation, 4 frequency channels and a time step of 5 s). As a result the data set will contain in the MODEL_DATA column the information about A-team sources visible during the observations.

B.3.2 A-team clipper

The second step consists in using a Python algorithm that takes in input the observed data set and flags[99] the A-team signal baseline by baseline above a threshold chosen according to the observing frequency. Different thresholds are possible for LBA and HBA data: default values are considered in the script but the clip levels can be adjusted by the user. The default threshold for LBA observations is 50 Jy. This value is required to remove the A-team contribution but at these low frequencies this implies the flag of most of the data. Therefore, at the moment for LBA observations at this frequency it is strongly recommended to use the standard demixing procedures. For HBA observations the algorithm is highly efficient for A-team sources far away ($\geq$20–30°) from the target. The default threshold is 5 Jy but values in the range 2–10 Jy can be suitable for different data sets, being 10 Jy the safest value in order not to flag the target source. The user can choose the proper value by comparing the amplitude of the observed target and the contribution from A-team sources. The script can be found at the LOFAR-Contributions GitHub repository:

```
https://github.com/lofar-astron/LOFAR-Contributions
```

Once the user copies it in his home directory it can be launched as follows

```
python Ateamclipper.py dataset.MS
```

The scripts takes few tens of seconds to run and an example of output is

```
Successful read/write open of default-locked table \
L123834_SB417_uv.dppp.MS.newtest: 27 columns, 12303270 rows \
Successful readonly open of default-locked table \
L123834_SB417_uv.dppp.MS.newtest/SPECTRAL_WINDOW: 14 columns, 1 rows
------------------------------
SB Frequency [MHz] 134.765625
% input XX flagged 4.12201796758
% input YY flagged 4.12201796758

Cliplevel used [Jy] 4.0
```

[99]An algorithm based on a subtraction procedure is currently under development.

```
Doing polarization,chan 0 0
Doing polarization,chan 0 1
Doing polarization,chan 0 2
Doing polarization,chan 0 3
Doing polarization,chan 1 0
Doing polarization,chan 1 1
Doing polarization,chan 1 2
Doing polarization,chan 1 3
Doing polarization,chan 2 0
Doing polarization,chan 2 1
Doing polarization,chan 2 2
Doing polarization,chan 2 3
Doing polarization,chan 3 0
Doing polarization,chan 3 1
Doing polarization,chan 3 2
Doing polarization,chan 3 3

% output XX flagged 6.54876711638
% output YY flagged 6.54876711638
```

The output gives information about the frequency of the data set and about the consequent clip level applied. The percentage of flags applied to the observation before (`input XX and YY flagged`) and after (`output XX and YY flagged`) the clipper is given as well. If the percentage of flags due to the algorithm is large ($\geq 20\%$) it is strongly suggested to apply the standard demixing procedure.

In Fig. 64 an example of the result from this algorithm is shown. In the top left panel, the amplitude versus time of the A-team sources VirA, CygA, CasA, and TauA, simulated as described in Sect. B.3.1, during an HBA observation of 3C 299 (PI: Dr. Chiara Ferrari) is shown for a single baseline. Amplitudes larger than 4 Jy are present at the end of the observations. In the top right panel the amplitude versus time from the same baseline is shown for the raw observed data set. In the real observation, a pattern can be identified when the simulated A-team sources are above 4 Jy. In the bottom left panel the amplitude versus time is shown after the standard demixing subtraction of VirA, CygA, CasA, and TauA, while in the bottom right panel after their flag with the new demixing algorithm with a cut threshold of 4 Jy.

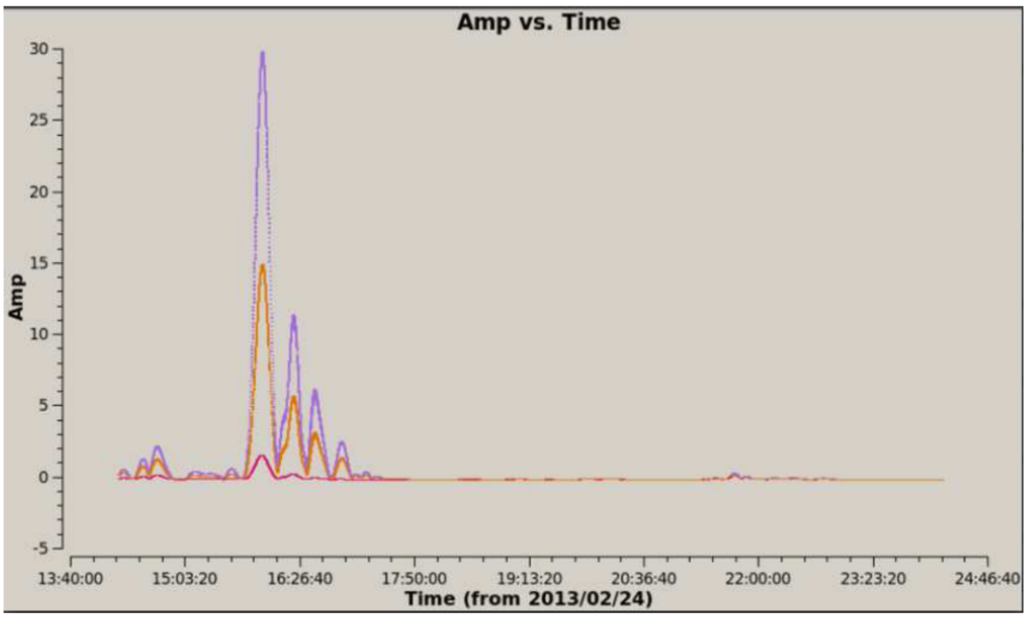


Figure 63: Example of amplitude of A-team sources versus time for a single baseline of an HBA observation. The A-team sources affect the observation just for a fraction of the total observing time.

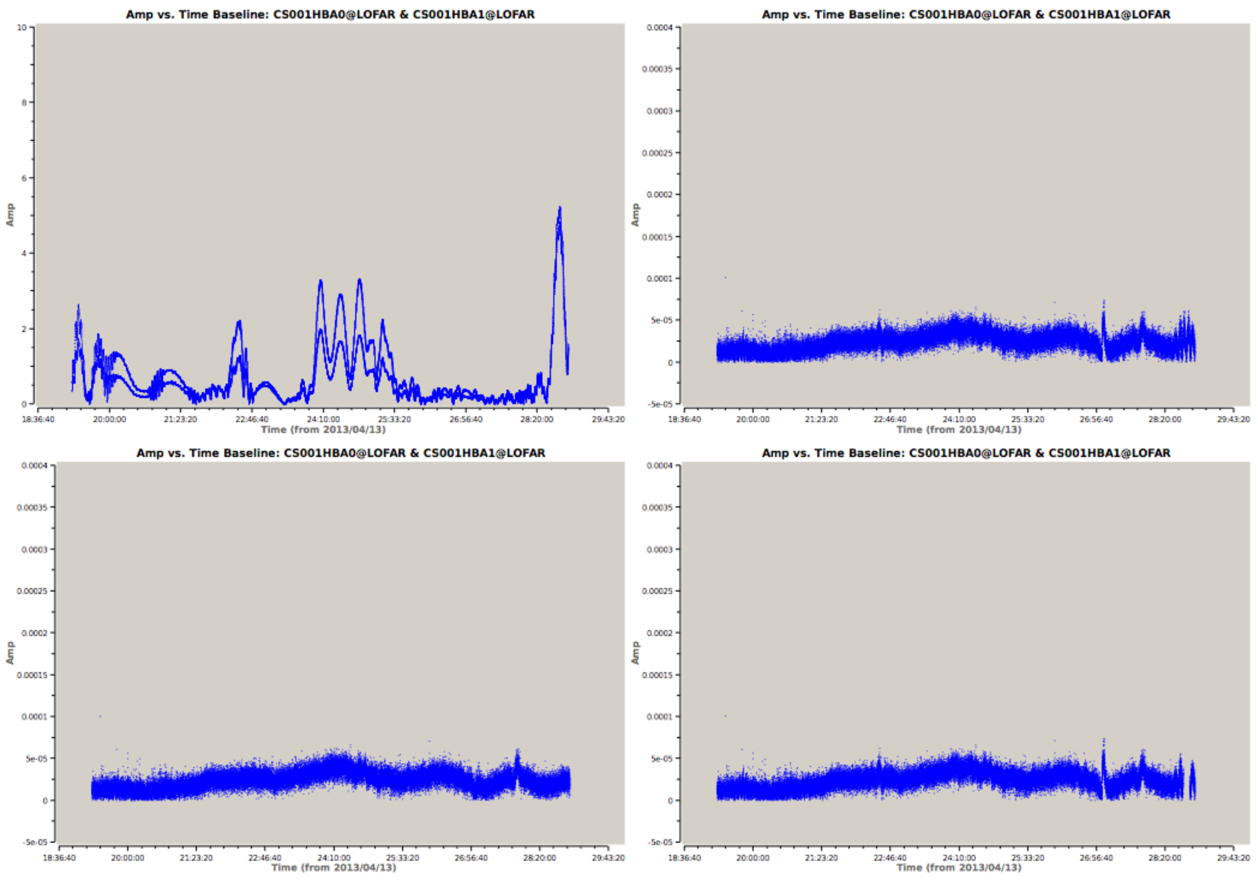


Figure 64: *Top left:* Simulation of the A-team sources VirA, CygA, CasA, and TauA during an HBA observation of 3C 299 (PI: Dr. Chiara Ferrari). *Top right:* Amplitude versus time from the raw data set. *Bottom left:* Amplitude versus time after the standard demixing subtraction of VirA, CygA, CasA, and TauA. *Bottom right:* Amplitude versus time after the flag of VirA, CygA, CasA, and TauA with the new demixing algorithm with a cut threshold of 4 Jy. The sources affecting the data is likely Cygnus A.