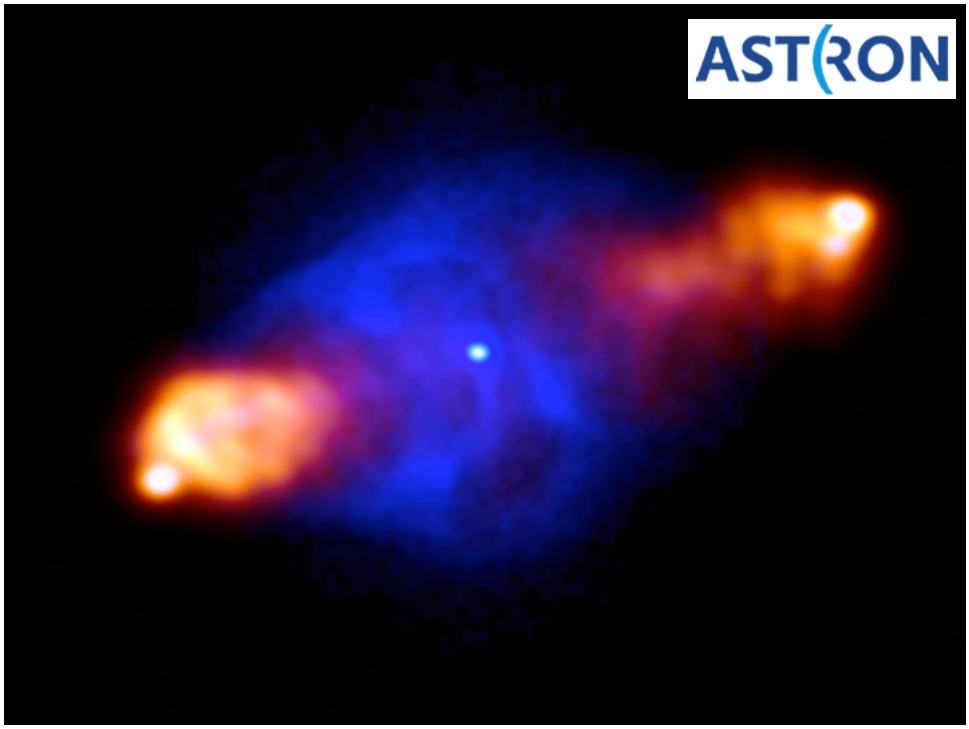
THE LOFAR IMAGING COOKBOOK:
Manual data reduction with the imaging pipeline

Version 14.1

April 22, 2014



Edited by Roberto F. Pizzo

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 The LOFAR wiki[1] contains much more information on each stage of the data reduction, but might be out of date in many places.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 R. F. Pizzo, 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: Roberto F. Pizzo[4]

Authors: Roberto F. 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, on behalf of the Lofar commissioning team.

[1] http://www.lofar.org/operations/doku.php?id=software:standard_imaging_pipeline
[2] https://proxy.lofar.eu/redmine
[3] http://lofar.org/operations/doku.php?id=maintenance:lofar_issue_tracker
[4] pizzo[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

2014-04-22
- corrections to the BBS chapter (Chapter 6);
- corrections to the Selfcal chapter (Chapter 11).

2014-03-28
- major update of the BBS chapter with inclusion of documentation on new functionality (Chapter 6);
- description of the Selfcal tool (Chapter 11);
- introduction of a chapter describing the LoSoTo (LOFAR Solution Tool) software (Chapter 7);
- introduction of a brief discussion on the Clock-TEC separation procedure (Sect. 7.4);
- description of the CEP3 processing cluster (Sect. 1.1.2).

2012-08-30
- addition of a practical reduction example for the field of 3C295 (Sect. 13.2);
- addition of an appendix describing the new Demixing approach (Appendix B);
- general updates of NDPPP, BBS, AWimager, Source detection, SAGECAL, and Shapelets chapters.

2012-10-25
- Included documentation on CEP1 `/staging3` area (Sect. 1.7);
- included documentation on script plot_Ateam_elevation.py in Sect. 2.6;
- modified Chapter 3 to include an updated description of the Standard Imaging Pipeline;
- updated NDPPP sections (Chapter 5) with info on
  - summing an arbitrary selection of stations into a single 'superstation';
  - support for RFIstrategies;
  - how to use the parset keyword 'ignoretarget' while demixing;
- removed 'demixing' Chapter and included description of the alghoritm within the NDPPP Chapter (Sect 5.1.10);
- included plots of Global Bandpass for HBA LOW, MID, and HIGH in Sect. 6.12.2;

- included info on how to manually add the beam tables to a MeasurementSet in Sect. 6.15;
- updated Chapter 8. The only supported version for the cluster file format is the old hybrid version;
- described script to transform a BBS skymodel into a sagecal model (Chapter 12)
- removed Chapter 'General troubleshooting' because not relevant anymore.

Contents

1 Getting Started[5]

1.1 The LOFAR cluster layout

1.1.1 CEP1

The correlated data coming from the BlueGene/P (BG/P)[6] are stored on a cluster of machines called CEP2 cluster. 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 CEP1 is used by the commissioners to manually play with the data and understand which strategy to use for the calibration and the imaging. An extensive description of CEP2 is given in Sect. 1.8, while in this section we will focus on discussing the architecture of CEP1. Note that CEP1 will soon be replaced by a ew commissioning cluster, CEP3 (see Sect.cep3).
The LOFAR CEP1 cluster is composed by 72 compute nodes (`lce001-072`) and 24 storage nodes (lse001-024). The storage nodes have 4 partitions of 5.1 TB each. Each partition holds a single XFS filesystem. In the following, a few more details on this cluster components are given.

- **Frontend:** it has 2 Intel Xeon L5420 quad core processors, 16 GB internal memory, 2 GbE interfaces and 2 TB disks. There are two identical frontends: `lfe001` and `lfe002`. They are used to build the software and regulate the workload on the compute nodes.
- **Processing units:** the 72 compute elements have 2 Intel Xeon L5420 quad core processors, 16 GB internal memory, 2 GbE interfaces and 1 TB disks. They can be accessed by secure shell.
- **Storage units:** the 24 storage nodes are HP DL180G5 boxes, having 2 Intel Xeon L5420 quad core processors, 16 GB internal memory, 6 GbE network interfaces and a total of 24 TB disk space. They have a XFS filesystem.

NOTE: *CEP2 is not meant for commissioning work. For that, commissioners can use CEP1 and in particular the compute nodes from lce001 to lce059, with prior coordination with the Radio Observatory (sciencesupport[at]astron[dot]nl). The nodes lce060 to lce072 are used for system and software tests. Due to the limited disk space, commissioners are strongly urged to administer their own disk space, and minimize their disk usage by deleting intermediate data reduction products and moving their final data to other systems themselves (the Observatory can provide advise in moving and archiving data). It is emphasized that the disks in CEP1 are not intended for long term storage or results, intermediate products, scripts or other files. As it is impossible for the Observatory to micro-manage the disk space, commissioners should be aware that disk deletions could happen with very short warning.*

1.1.2 CEP3

CEP1 will be replaced soon by CEP3. The CEP3 cluster will be available from middle of April 2014 and will allow for running science processing close to the CEP2 facility. The cluster consists of 24 equivalent servers with the following specifications:

- DELL PowerEdge R720 Rack server

[5] This chapter is maintained by R. F. Pizzo, `pizzo[at]astron[dot]nl`.
[6] It is located in Groningen, The Netherlands.

- 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 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. One server is reserved for observatory use and one server is reserved for LOFAR system and software development.

The LOFAR Imaging Cookbook will be updated with some more information about how to use CEP3 one the commissioning of the new cluster will be completed.

NOTE: *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. The User's data products generated on CEP3 nodes will be automatically removed on a regular basis, to enable new users to have enough disk space to perform their data reduction.

1.2 Logging on to CEP1

As mentioned above, normal users have access only to CEP1 and in here they can experience the quality of the data and understand the best approach to use for the calibration and imaging of the visibilities.

- To access CEP1, 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.

[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`).

- You can now log in to the front end (you are requested to use `lfe002`)[9]:

  ```
  > ssh -Y lfe002
  ```

  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).

  An interesting tool you might want to explore from the front end is `cexec`, which allows you to run the same command on multiple nodes. E.g., to show all users on lce019 - lce023:

  ```
  > cexec lce:18-22 users
  --------- lce019---------
  heald rol
  --------- lce020---------
  rol
  --------- lce021---------
  rol
  --------- lce022---------
  rafferty rafferty rol
  --------- lce023---------
  asgekar asgekar
  ```

  As this output points out, the numbering of the nodes needed by the `cexec` command is shifted with respect to their id number; e.g. to check the range of lce nodes between 001 and 009, you need to type `cexec lce:  0-8 'command'`. The very first time you run the `cexec` command on a series of lce nodes, you may need to log in to each of the nodes separately to store your SSH key (see Sect 1.3.2). However, you can avoid this by using the procedure described in Sect. 1.3.3.

- Log on to (for example) `lce019`[10]

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

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

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

  ```
  > tcsh
  ```

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

Processing can now take place.

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

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

[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 `lce` node to work on

1.3 Setting up your working environment

1.3.1 Login scripts

Log in to the front end cluster and from your $HOME directory follow the approaches reported below.

Under the (t)csh shell:

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

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

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

```
> use LofIm
```

This software is newly built every day. Sometimes, e.g. when the build of the current day is unsuccessful, you need a specific "build of the day". For that, use

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

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

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

```
> use Casa
```

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

```
> use Pythonlibs
```

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

```
> use Karma
```

To run a few scripts needed during the data reduction (see Sect. 14.2) without having to copy them into your working directory, just type:

```
> use Tools
```

A few of these scripts will also need the Pydal libraries libraries to be initialized:

```
> use LUS
```

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 LofIm
```

the login scripts will initialize Casa and LofIm for you at login time.

More information on this topic can be found on the LOFAR wiki [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. 6) 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 `lfe002` set up passwordless access to the `lce` nodes via SSH:

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

  Press enter and again when prompted for a password.

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

  `$HOME/.ssh/authorized_keys` must be available on all the machines you need to access with this key; since your home directory is automatically mounted on all the cluster nodes, they should already be accessible—you can copy it to other, external, systems if required.

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

1.3.3 Disable SSH Host Key Checking

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

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

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

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

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

1.4 Setting up a personal database for BBS

A personal database is required to use BBS. This database needs to be created once and re-created whenever the BBS SQL code changes. Such changes are infrequent.

First, log on to the front end node of the cluster (`lfe002`). If you are creating the database for the first time, issue the following commands. If you are re-creating the database, please skip to the next block of commands.

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

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

Now, download `bbs-sql.tgz`[13], extract it, and execute it using `psql`.

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

The messages that appear on the screen should all be informational, optionally prefixed by NOTICE, and can be ignored. (Should you encounter a message prefixed by ERROR, this is a genuine error and should be reported via the LOFAR issue tracker). Afterwards, the `bbs-sql` directory can be removed.

1.4.1 Common errors

- If you get this error :

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

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

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

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

- If you get the error "Language p̈lpgsql—" does not exist", you have to run the CREATELANG command on your database (see section 1.4)

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

1.5 Setting up a working space

At the current stage of LOFAR commissioning, the data recorded during the observations are stored on the CEP2 compute cluster (see Sect. 1.8), and a few sub bands are copied to CEP1 by the science support personnel to allow the users to understand the best data reduction strategy to follow to successffuly calibrate and image the visibilities. The copy is done to specific compute nodes, which are selected on the basis of their available free disk space. The location of the copied data will be communicated to you by the science support personnel, together with the name of the lce node you have been assigned to work on. 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>@lceXXX:/data/scratch/pipeline/<LOFAR dataset> .
```

where `<LOFAR dataset>` has the syntax `LXXXXX`[14].

1.6 Running a command on multiple nodes

As already mentioned before, the `cexec` program can be used from the front end to run a command on various nodes at the same time. For example:

```
> cexec lce: ls /data/scratch
```

executes the `ls` command on all lce nodes. It will show the machine name and the matching files.

On top of `cexec`, the `cexecms` command has been developed. It also runs a command on all lce machines, but it has the ability to insert the file name in the command. Running the command without arguments gives some help info.

For example:

```
> cexecms -c lce: "echo ms=<FN> $HOSTNAME" "/data1/L2011_25121/*"
```

results in the output

```
--------- lce019 ---------
No file matching /data1/L2011_25121/* found
--------- lce022 ---------
```

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

```
ms=/data1/L2011_25121/L25121_SB216_uv.MS lfe002
ms=/data1/L2011_25121/L25121_SB217_uv.MS lfe002
ms=/data1/L2011_25121/L25121_SB218_uv.MS lfe002
ms=/data1/L2011_25121/L25121_SB219_uv.MS lfe002
ms=/data1/L2011_25121/L25121_SB220_uv.MS lfe002
ms=/data1/L2011_25121/L25121_SB221_uv.MS lfe002
ms=/data1/L2011_25121/L25121_SB222_uv.MS lfe002
ms=/data1/L2011_25121/L25121_SB223_uv.MS lfe002
ms=/data1/L2011_25121/L25121_SB224_uv.MS lfe002
ms=/data1/L2011_25121/L25121_SB225_uv.MS lfe002
ms=/data1/L2011_25121/L25121_SB226_uv.MS lfe002
ms=/data1/L2011_25121/L25121_SB227_uv.MS lfe002
--------- lce024 ---------
ms=/data1/L2011_25121/L25121_SB240_uv.MS lfe002
ms=/data1/L2011_25121/L25121_SB241_uv.MS lfe002
ms=/data1/L2011_25121/L25121_SB242_uv.MS lfe002
ms=/data1/L2011_25121/L25121_SB243_uv.MS lfe002
```

1.7 The staging areas

We have recently added a GlusterFS parallel filesystem to the CEP1 installation, based upon the existing lse storage nodes. This enables CEP1 users on lce compute nodes to have a large storage volume attached to these nodes, that can be seen as a temporary archive. Each lce node has this large disk of 91 TB mounted as /staging3[15]. Your benefit is the fact that you don't need to know on which lse node your data lives. For more details, please see also http://www.gluster.org/. There are several flavors of parallel file systems. What most of them have in common is complexity and a corresponding administrative burden. GlusterFS on the contrary has a simple approach. It forms a layer over the existing local file systems. All directory entries are physically available on all nodes, but all files are randomly spread over the 20 lse nodes underneath. This works well, but has a severe drawback: MeasurementSets are spread over all nodes, since one MS consists of 68 individual files. As a direct consequence, if one of the lse nodes fails, your MS's will likely be corrupted since they will be missing important components. Our advice therefore is: ONLY STORE TAR BALLS OF MSs on this file system. In the rare case that one of the lse nodes breaks forever, only a few SB's of your observation will be lost, and not the entire dataset.

Details on how to proceed are reported in the following:

- An example of an MS on a lce node:

  ```
  lce003> tar cvf /staging3/<mydirectory>/L20567_SB20.tar
   /data/scratch/asgekar/L2010_20567/L20567_SB020-uv.MS.dppp
  ```

- Copy your tar file from /staging3 to your current lce node:

  ```
  lce002> cd /data/scratch/<mydir>; tar xvf /staging3/<mydirectory>/L51130.tar
  ```

- Start your processing locally on lce0xx

[15] Three more staging areas (/staging1, /staging2, and /staging4) have been created, but access is restricted to the Observatory personnel only.

Please do not process data using a NFS mounted /staging3 disc, but copy your data to the local machine first. We would like to emphasize that the /staging3 filesystem is not a long term archive, but it should only be used to temporary archive the data you are planning to work on in the very near future.

1.8 The LOFAR phase 2 cluster - CEP2

The Lofar phase 2 cluster (CEP2) has been adopted to store the observations and process them through the Lofar Imaging Pipeline. The idea is that as soon as an observation is performed, some processing is immediately done on this cluster (as RFI removal and averaging). Afterwords, a few sub bands of that specific observation will be copied to the CEP1 cluster, where the users will have the chance to process them manually in order to find out which parameters suit the data reduction the best. Once these parameters are known, they will be used to process the original data on the CEP2 cluster through BBS and the imager. 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). In the following, a short description of the computing characteristics/performances of the new cluster is given.

The Lofar Phase 2 cluster consists of

- 100 locus compute nodes with 21TB each (called locusxxx)
- 2 lhn head nodes (called lhnxxx; lhn001 is the main head node for users)
- 4 router nodes
- Infiniband interconnect using Ethernet protocol

Each locusxxx compute nodes consists of:

- 64GB memory
- 24 CPU's (6 quad cores @2.1GHz)
- 21TB diskspace, XFS filesystem
- 40 Gbps Infiniband network interface

Each head node (lhn001 & lhn002) consists of:

- 64GB memory
- 16 CPU's @ 2.3GHz
- 512 GB Solid State Disc
- 6TB RAID5 disks

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.

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.

2 Data Inspection[16]

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[17]. 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
```

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

[17] 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

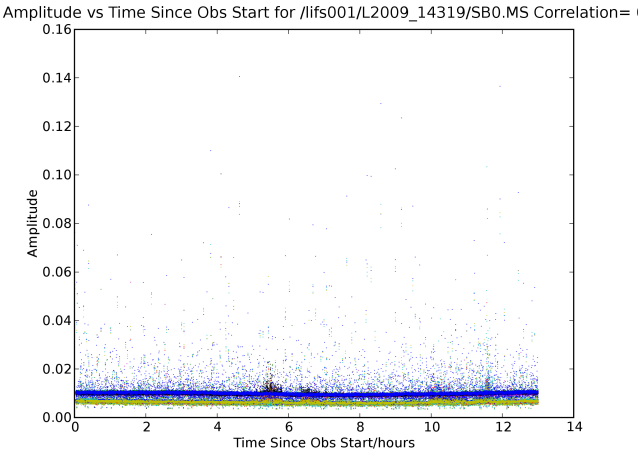
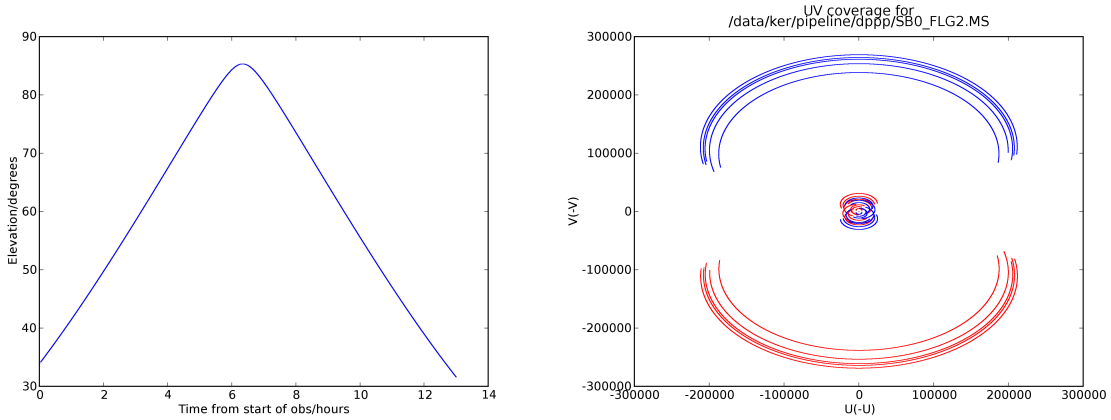


Figure 1: Plotted here using a pyrap script is the amplitude vs time for the SB0.MS 3C196 observation. The high level of RFI is instantly apparent.



(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[18]. It plots either amplitude or phase against time, frequency, or channel number.

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

```
> use LUS
> use LofIm
```

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

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

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

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

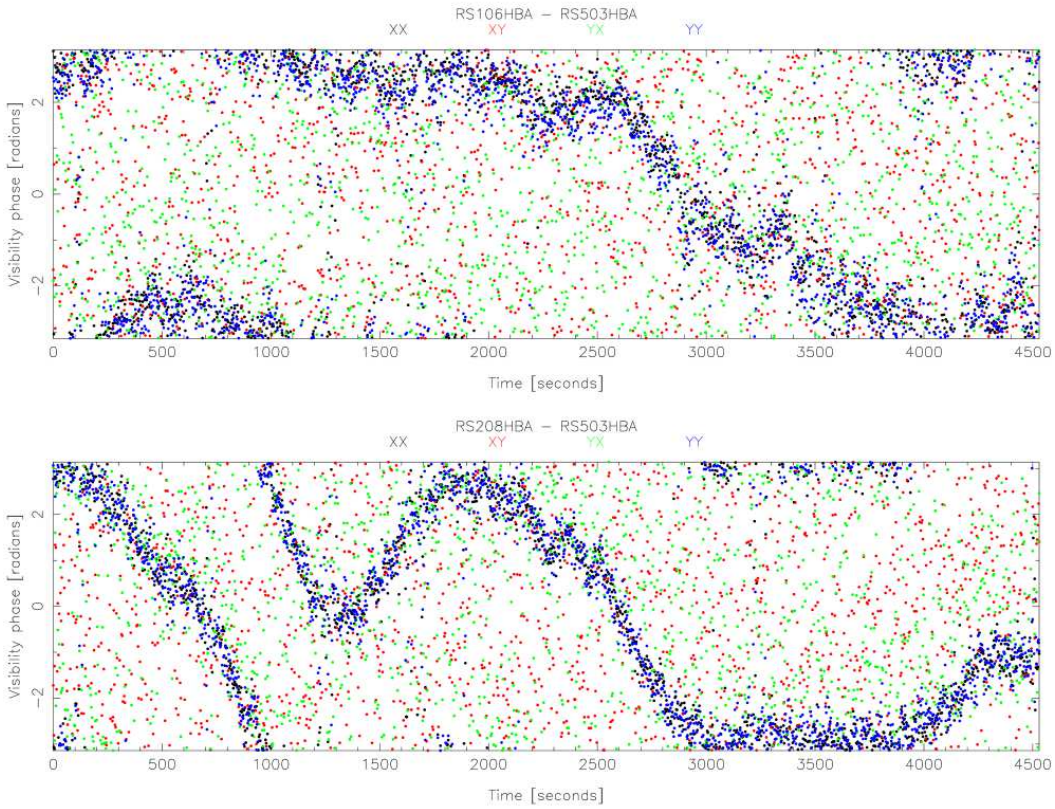


Figure 3: An example of using `uvplot.py`. The command used to generate this plot was `/home/heald/bin/uvplot.py -i /data/scratch/pipeline/L2009_15697/SB64.MS -t 0,1500 -y phase -n 1,2 -x time -d output.ps/cps` . The point size has been increased afterward using `/home/heald/bin/embiggen.csh`.

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

```
/home/heald/bin/uvplot.py -i /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.

Troubleshooting. If you receive the following error

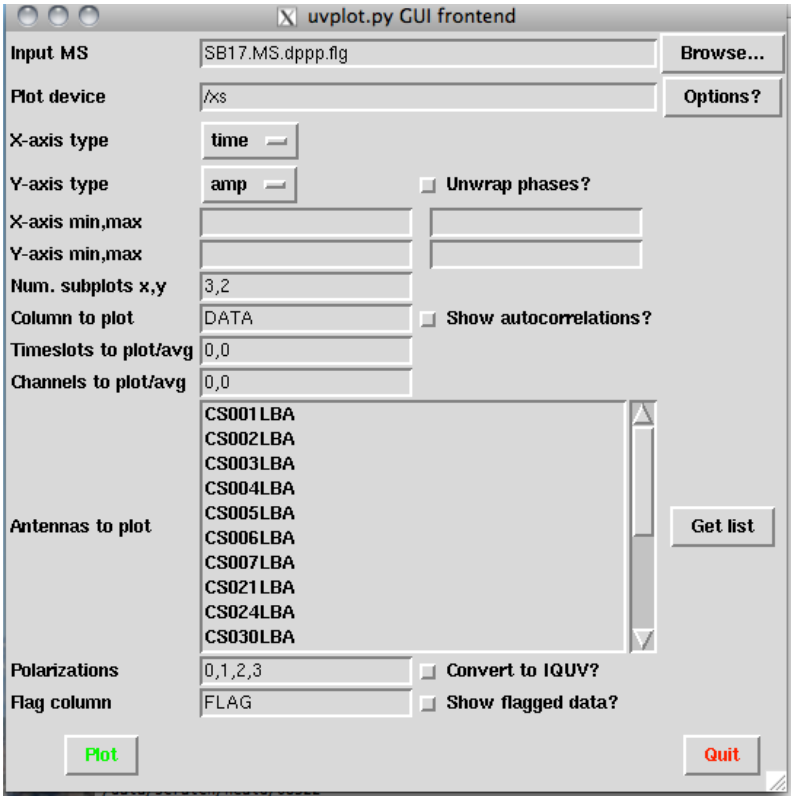


Figure 4: The GUI interface to the plotting program `uvplot.py`.

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

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

2.4 CASA

CASA[19] is the python-based next generation replacement for AIPS/AIPS++. It is however a beta version, and caution must be exercised in its use, as it is still lacking in functionality and has bugs present. CASA can be used to display the data[20]. Be aware that trying to inspect the raw visibilities with CASA will produce a "`segmentation fault`" error. To avoid this, you should make a copy of the dataset with NDPPP (see the first example parset in Sect. 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" '
```

[19] http://casa.nrao.edu/

[20] 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.

2.4.1 Casaviewer

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

2.4.5 CASA bugs

- Trying to inspect the raw data with `casaviewer` or `plotxy` is impossible. It is advised to first `touch` the raw data through NDPPP (see Sect. 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.
- Many of the plotting options advertised in the help files do not exist yet - another good reason to become familiar with Pyrap!

2.5 The Drawer[22]

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 *u*v 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 LofIm
> use Pythonlibs
```

The help file of drawMS is self contained:

```
/home/tasse/drawMS/drawMS -h

Options:
  --version             show program's version number and exit
  -h, --help            show this help message and exit

  * Necessary options:
```

[22]This sub section was kindly provided by Cyril Tasse (cyril[dot]tasse[at]obspm[dot]fr)

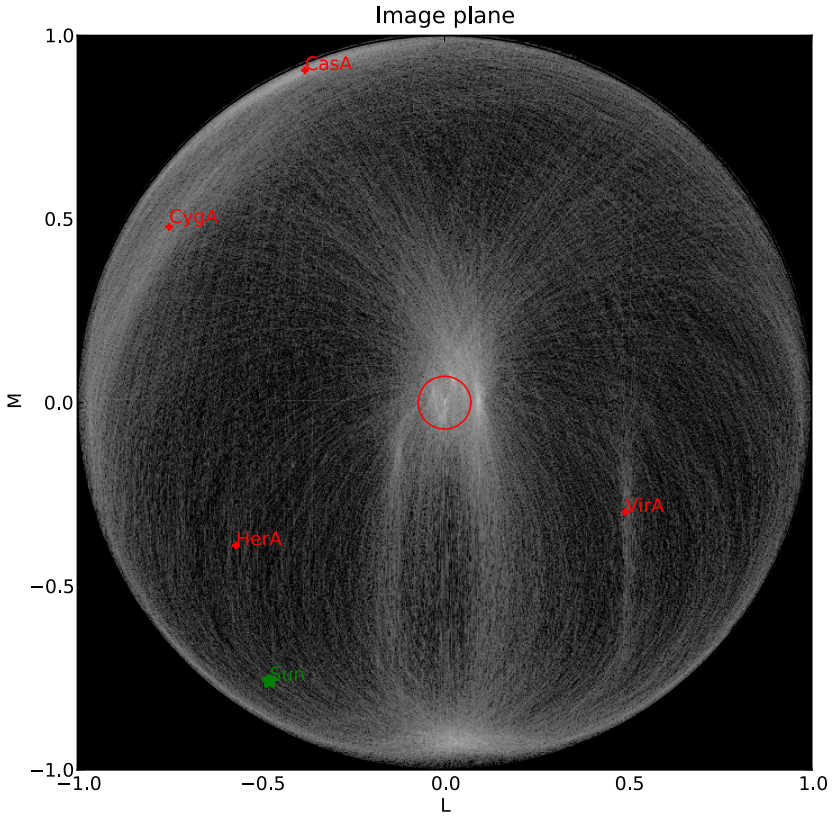


Figure 5: drawMS is a simple algorithm that allows to quickly recover spatial information on the sources that have the brightest apparent flux. In this example, drawMS is run on the raw data of an observation of the Bootes field. One can clearly see the contribution from CasA, CygA, and TauA, while there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

```
  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
  --FillFactor=FILLFACTOR
                        The probability of a baseline/timeslot to be
                        processed. Default is 1.0. Useful when large dataset
```

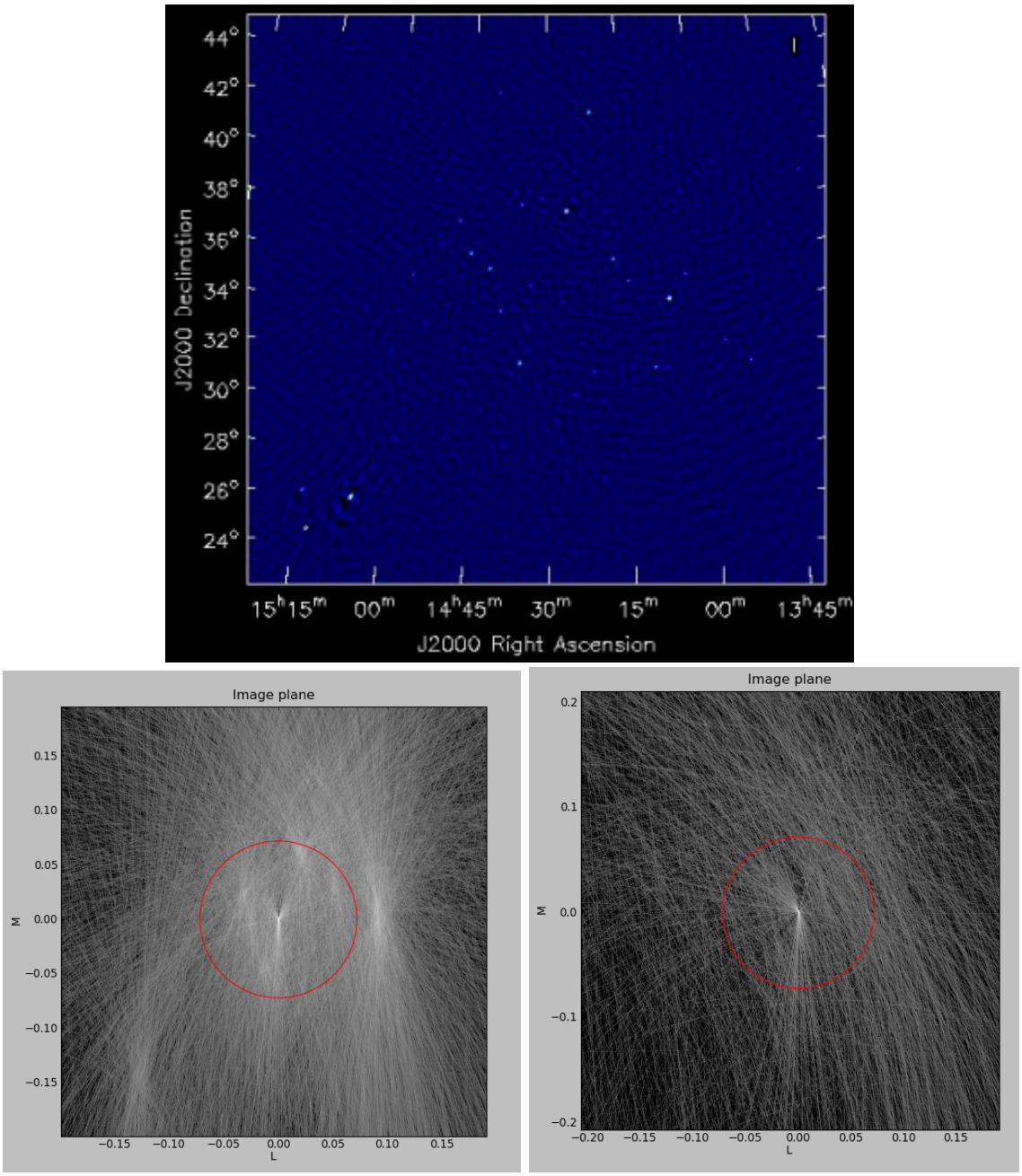


Figure 6: *Top panel*: a very wide field image of the Bootes field (a few SBs), which took a long time to be imaged because of the large field of view. *Bottom panel*: drawMS outputs were generated in less than 5 minutes. Line plots show the overdensities corresponding to real sources in the image. We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

```
                    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:

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

and for the second part of the run:

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

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

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

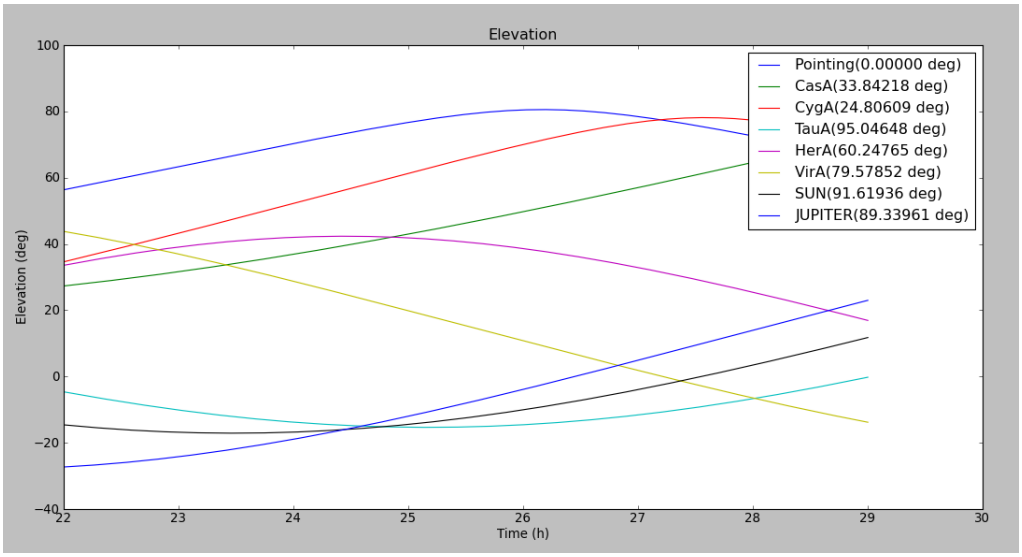


Figure 7: The elevation and angular distance of the A-team from the target field centered on B1835+62. The plot was obtained with `plot_Ateam_elevation.py`.

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`, available in `/opt/cep/tools/cookbook` on CEP1. 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` [23] 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 LofIm; use Casacore

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

[23] It can be found in `/opt/cep/tools/cookbook`

```
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[24]

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:

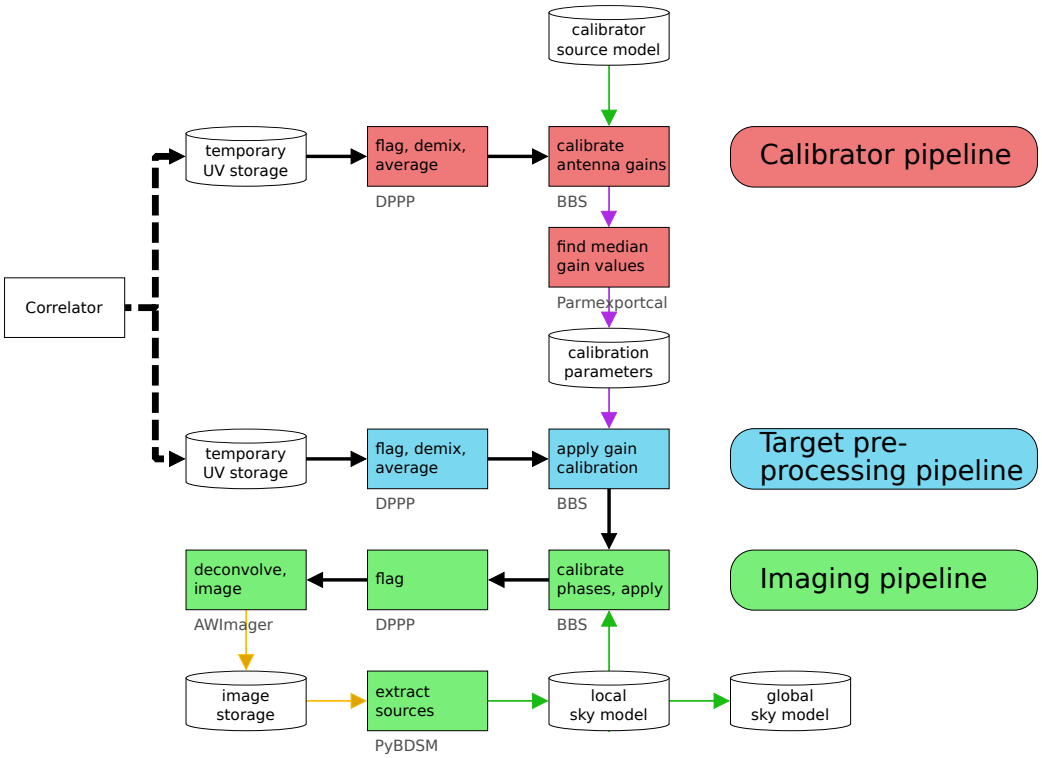


Figure 8: A diagram of the Standard Imaging Pipeline.

- 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 NDPPP - 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. 6) - 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. 9). AWImager is able to perform

[24]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. 10) 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-during processes

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

4 The AOFlagger[25]

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 NDPPP (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.

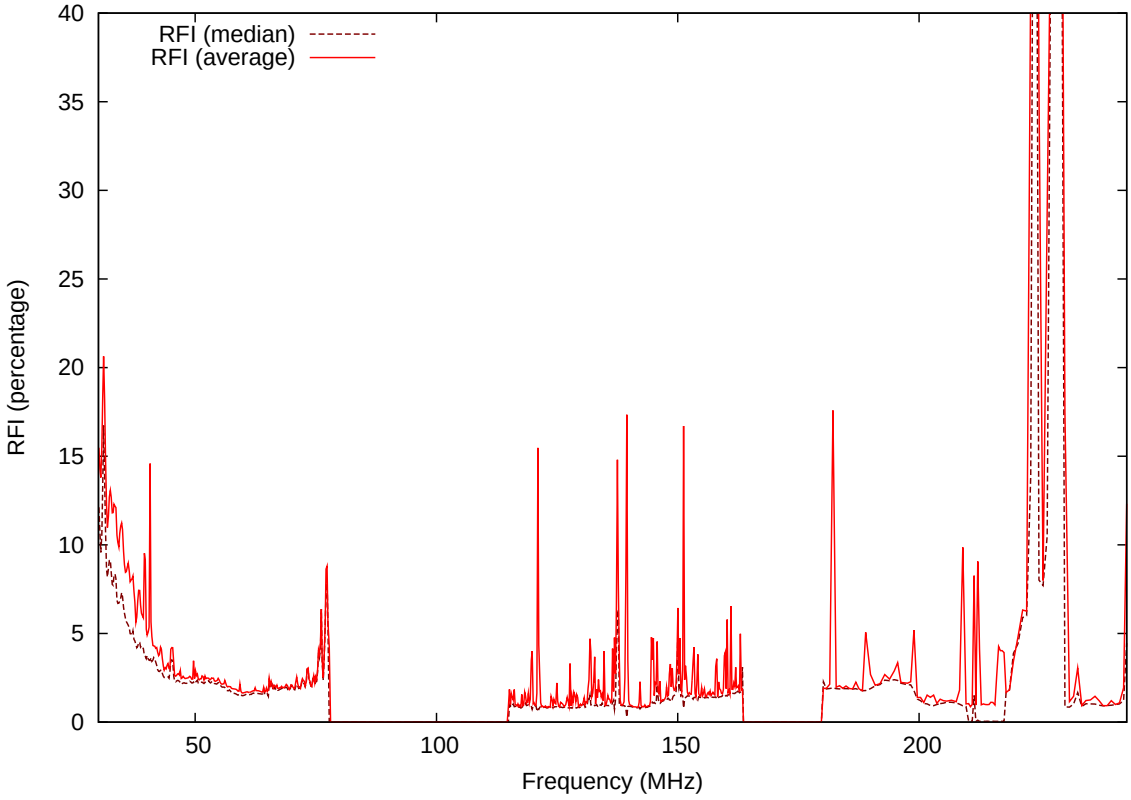
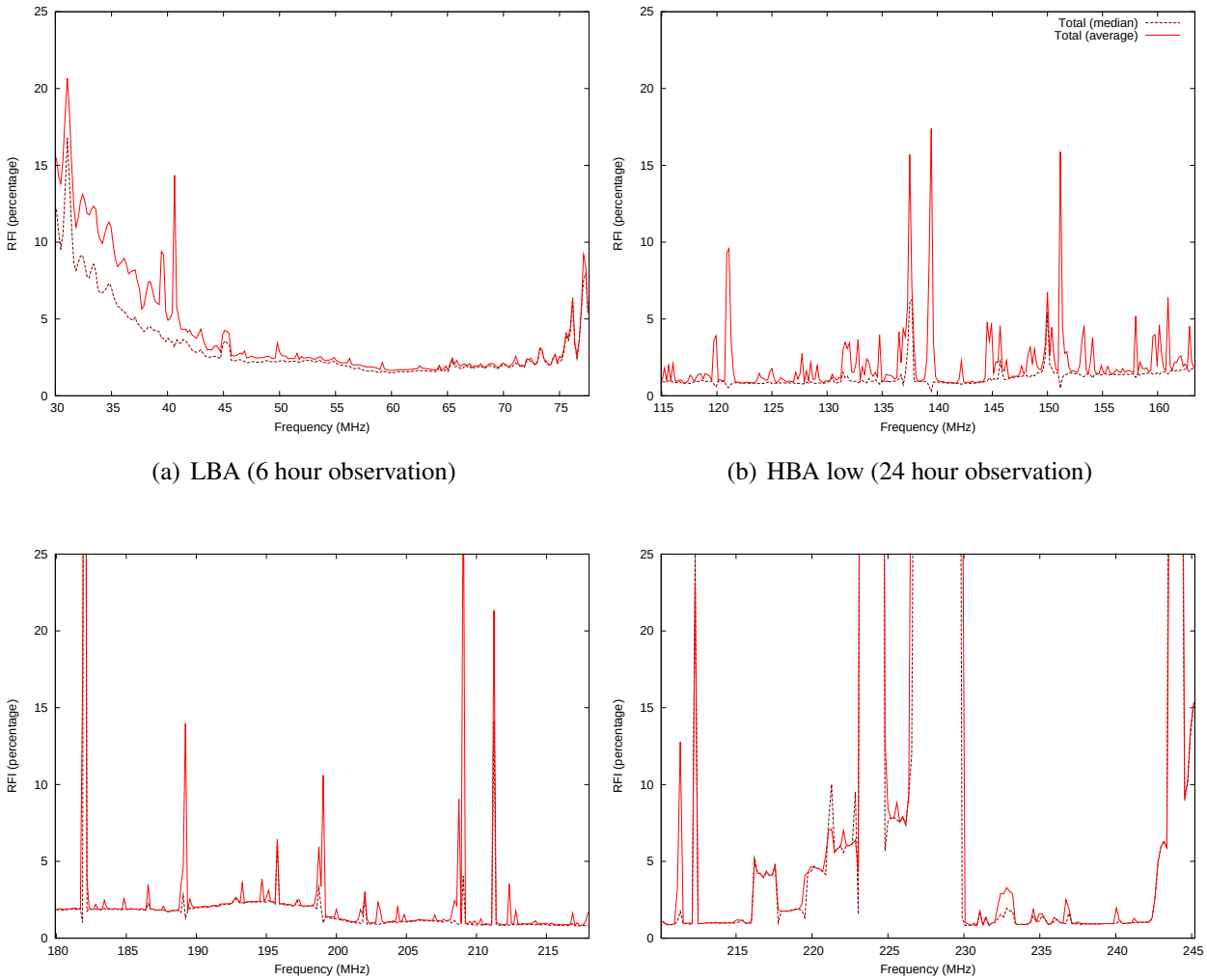


Figure 9: The RFI distribution in most of the observable frequencies of LOFAR. It combines data from four observations, whose RFI spectra have been given in more detail in figure 10. In most of the frequencies, the RFI situation is benign and cause only a few percent of data loss. Towards 30 MHz and 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 NDPPP 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 NDPPP (see Chapter 5). If one wants to use the default flagging settings, it is recommended that NDPPP 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 NDPPP, please see Chapter 5. In the following sections, the details regarding

[25]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.

the AOFlagger as 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 NDPPP to average the data. When averaging a measurement set that has been flagged by the AOFlagger, your NDPPP 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,
```

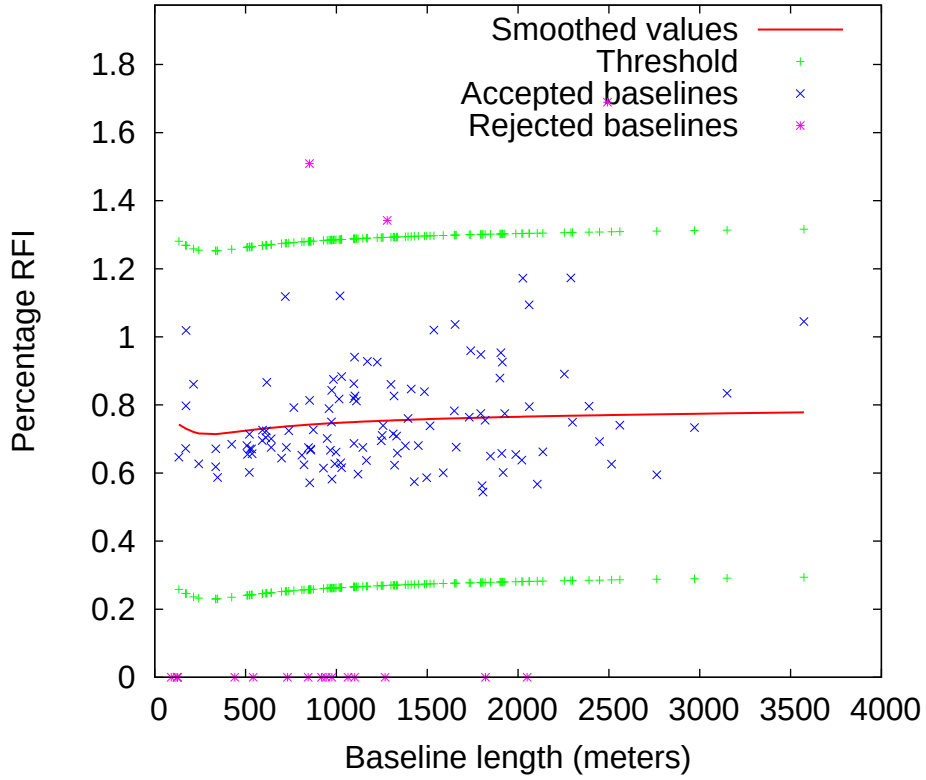


Figure 11: The percentage of RFI vs baseline length for sub band 127 of the observation `L2010_08567`. The bad baselines (in purple) will be reported in the output of RFI Console.

```
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 `NDPPP.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 New Default Pre-Processing Pipeline (NDPPP)[26]

NDPPP (new 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.

NDPPP 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

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 input to NDPPP 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.

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 NDPPP combines them into one.

Detailed information on NDPPP can be found here[27]. For specific questions regarding this software, you can contact the software developer, Ger van Diepen (diepen[at]astron[dot]nl).

5.1 Various ways to use NDPPP

Several operations can be done using NDPPP 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.

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

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

5.1.1 Basic usage

To run NDPPP, 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 `NDPPP.parset` file into your working directory by typing:

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

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

```
> use LofIm
> NDPPP
```

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

```
> NDPPP some.parset
```

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:

```
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 NDPPP 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 NDPPP 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 NDPPP 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
tt 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 NDPPP 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 NDPPP 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 NDPPP.

```
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 NDPPP.

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

NDPPP will always test the input data column for NaNs. However, if no steps are specified, NDPPP 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.

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.

Till recently, demixing was applied to the data by using a script developed by Bas van der Tol. Lately, the algorithm has been included within NDPPP and can be invoked through the demixer step. With respect to the scripted version, the demixer makes it possible to:

- specify a source model for the target and for other strong sources in the field to get a better gain solution for the sources to subtract. All source models must reside in a so-called SourceDB table;
- calculate the gain solutions jointly for all source models;
- use different averaging factors for demix and subtract.

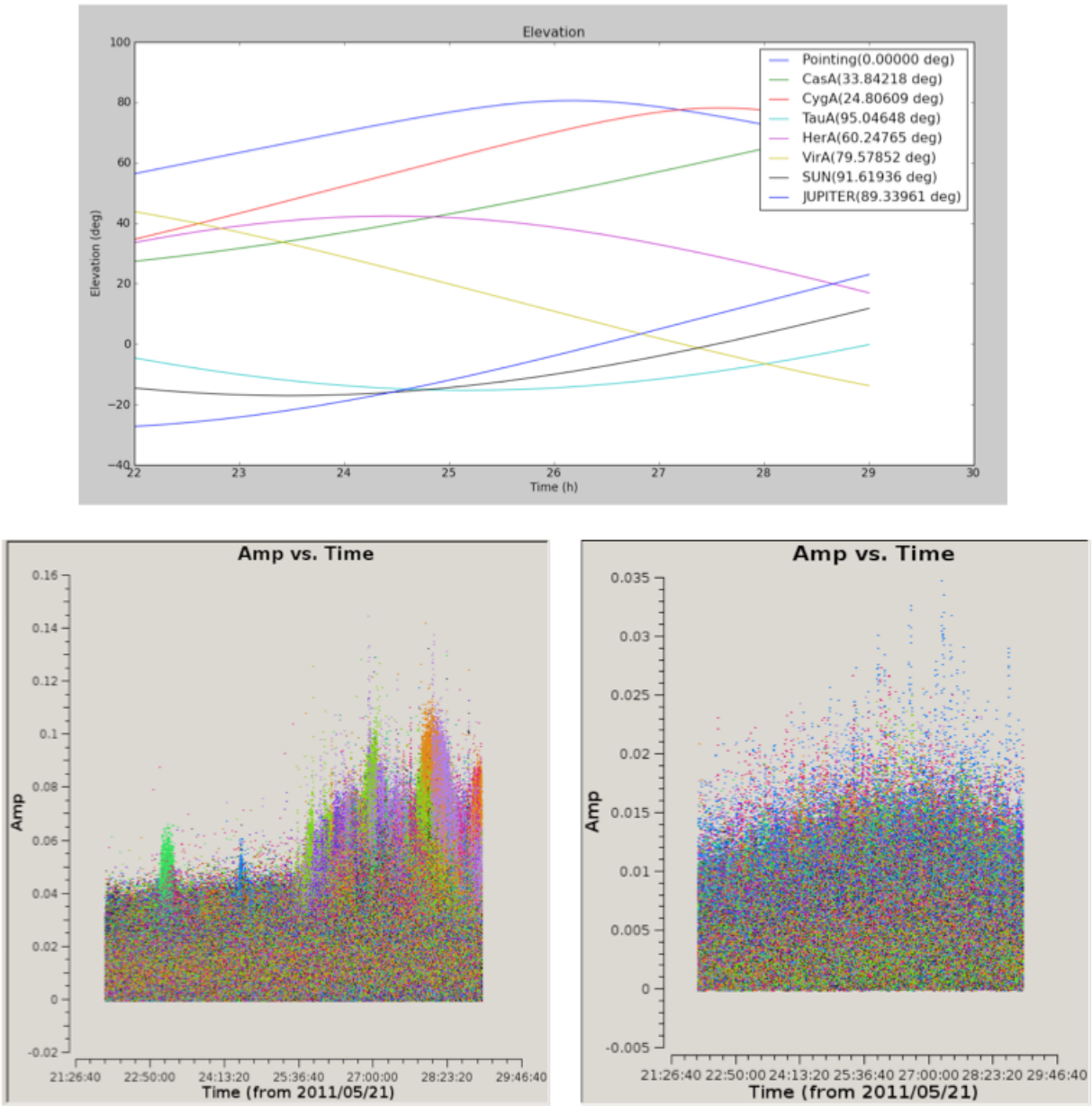


Figure 12: *Top:*the elevation and angular distance of the A-team from the target field centered on B1835+62. This plot shows that during this observation CygA and CasA are very high in elevation and pretty close to our target (24 and 33 degrees, respectively). *Bottom left:* target visibilities before 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.

Below, an example 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 subbands 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.

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

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

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

steps = [preflag,flag1,count,avg1,flag2,avg2,count]

preflag.type=preflagger
# This step will flag the autocorrelations.
# Note that they are not flagged by default by NDPPP.
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 NDPPP command. It defaults to NDPPP.parset.

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

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

5.2.1 Input / output parameters

A description of the input/output parameters of NDPPP 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, see Sect. 1.5). Output data should *not* be written back to the storage disks. Also, do not write output data to /home/<user name>, as space is very limited on this disk.

You can let NDPPP create a so-called VDS file, which tells other data processing programs (notably, BBS and mwimager) where the data live. You need a so-called cluster description file for this. These can be found in /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 NDPPP can be summarized as follows.

- If one correlation is flagged, all correlations will be flagged.
- The msin step flags data containing NaNs or infinite numbers.
- A PreFlagger step can be used to flag (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 NDPPP itself iterates over the baselines. Default strategy files exist for LBA and HBA observations (named LBAdefault and HBAdefault).

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 NDPPP can be summarized as follows.

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

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

- The UVW coordinates are also averaged (not recalculated).
- It fills the new column LOFAR_FULL_RES_FLAG with the flags at the original resolution for the channels selected from the input MS. It can be used by BBS to deal with bandwidth and time smearing.
- Averaging in frequency requires that the average factor fits integrally. E.g. one cannot average every 5 channels when having 256 channels.
- When averaging in time, dummy time slots will be inserted for the ones missing at the end. In that way the output MeasurementSet is still regular in time.
- An averaged point can be flagged if too few unflagged input points were available

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, NDPPP uses a new baseline selection syntax, which is common also to CASA. Its properties are reported in the following.

- Whitespace can be given at will.
- The selection is given as a list of groups separated by semicolons. A group can be preceded by an exclamation mark meaning exclusion. It can be used to exclude part of a previous group. If desired, part of that excluded group can be selected again in a subsequent group.
  Note that the OR relation is used for ordinary groups, while AND is used for excluded groups. For example:

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

  means

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

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

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 uparrow for negation. For example:
     `[a-zA-Z0-9]` a single letter or digit.
     `[abcde]` one of these 5 letters.
     `[^abcde]` not one of these letters.
  4. curly braces indicate a choice of strings (separated by commas). For example:
     `'*{h,cc}'` for any name ending in `h` or `cc`
     As shown in the last example a pattern has to be enclosed in single or double quotes if it contains a comma (or possibly other special characters).

- An antenna name can be given as an extended regular expression if enclosed in slashes[28]. 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`

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

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.

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

```
!CS*
```
all baselines not containing 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 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.

5.5 Analyzing the data quality with `aoqplot`

Once you have succesfully run NDPPP 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.

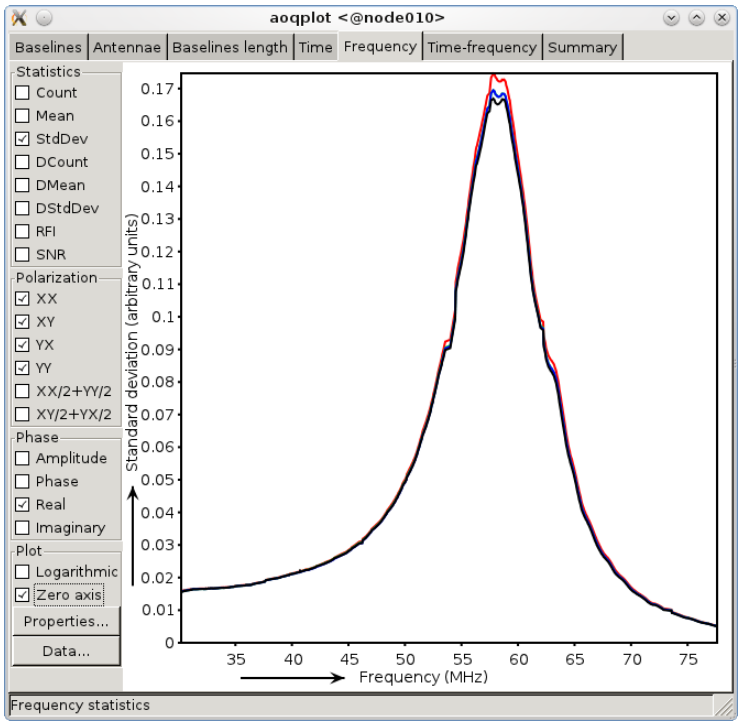


Figure 13: The `aoqplot` window showing the standard deviation of the data over frequency of a full observation.

5.5.1 Usage

If your environment allows (see below), one can get the statistic plots of an observation of a NDPPPed 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

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
```

5.5.2 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.

5.5.3 Background information

The `aoqplot` tool works together with NDPPP. Recent versions (>21 December 2011) of NDPPP 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[29]. Because the statistics plotting tool require these tables, you can not directly plot statistics of measurement sets that are averaged by an older NDPPP, 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:

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

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

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 NDPPP 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.

[29] offringa[at]astro[dot]rug[dot]nl

5.6 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. The tasks `flagdata` or `plotxy` may be useful for this. Once the CASA Plotter has loaded and data is visible, click the `Mark Region` button, highlight data that you wish to flag, click the `Flag` button, and `Quit` once you are finished.

Observations at 'low' elevation (below $\sim 30^\circ$ for Cygnus A, and below $\sim 40^\circ$ for 3C196) are sufficiently noisy that they are of limited use. These bad time ranges need to be identified and removed. This could be done through NDPPP, but also by manual flagging in CASA or using the CASA SPLIT task or the python script `split_ms_by_time.py` (Section 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.

6 Calibration with BBS[30]

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.

6.1 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 6.9 we will treat the global solve.

As input, BBS requires an observation (one or more subbands / measurement sets), a source catalog (see Section 6.3), a table of initial model parameters (see Section 6.5), and a configuration file (also called *parset*, see Section 6.7) 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.

6.2 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.
- `<parset>`, which defines the reduction (see Section 6.7).

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

- <source catalog>[31], which defines a list of sources that can be used for calibration (see Section 6.3).
- 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.

6.3 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 6.4 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.

For more information about the recognized fields, default values, and units, please refer to the makesourcedb documentation on the LOFAR wiki[32].

[31] 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.

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

6.3.1 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. We are working on this (issue #5365).

6.3.2 Spectral index

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

$$\log_{10}(S) = \log_{10}(S_0) + c_0 \log_{10}\left(\frac{\nu}{\nu_0}\right) + c_1\left[\log_{10}\left(\frac{\nu}{\nu_0}\right)\right]^2 + \ldots + c_n\left[\log_{10}\left(\frac{\nu}{\nu_0}\right)\right]^{n+1} \quad (4)$$

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 $[c_0, \ldots, c_n]$[33].

6.3.3 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:

$$Q(\lambda) = p\, I(\lambda)\, \cos(2\chi(\lambda)) \quad (5)$$
$$U(\lambda) = p\, I(\lambda)\, \sin(2\chi(\lambda)) \quad (6)$$
$$\chi(\lambda) = \chi_0 + RM\, \lambda^2 \quad (7)$$

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*.

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:

$$\chi_0 = \frac{1}{2}\tan^{-1}\left(\frac{U}{Q}\right) - \lambda_0^2\, RM \quad (8)$$

[33] 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.

$$p = \frac{\sqrt{(Q^2+U^2)}}{I(\lambda_0)} \tag{9}$$

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.

6.4 GSM[34]

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 6.3) 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 4 in Section 6.3.2.

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 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

[34]This section was contributed by Bart Scheers.

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:

$$\rho_{i,\star} \equiv \sqrt{\frac{(\alpha_i \cos\delta_i - \alpha^\star \cos\delta^\star)^2}{\sigma^2_{\alpha_i} + \sigma^2_{\alpha^\star}} + \frac{(\delta_i - \delta^\star)^2}{\sigma^2_{\delta_i} + \sigma^2_{\delta^\star}}}, \tag{10}$$

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 4 in Section 6.3. 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.

6.5 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`[35].

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:

```
> 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.

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

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.

```
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
```

6.6 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.

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 6.7.5	direction dependent
`Beam`	The LOFAR beam model. See 6.6.1	direction dependent
`DirectionalGain`	Directional gain	direction dependent
`CommonScalarPhase`	Scalar Phase	direction independent
`CommonRotation`	Rotation	direction independent
`TEC`	TEC (ionosphere), see 6.7.5	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 two most commonly used effects, Gain and DirectionalGain, have only one option: `Phasors`. When set to `True` (or `T`), the gains are expressed like $Ae^{i\phi}$, 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 6.6.1.

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
```

6.6.1 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 NDPPP). 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.

6.7 Example reductions

The sequence of operations that BBS will perform on the data are defined in a configuration or *parset* file[36]. The BBS documentation on the LOFAR wiki documents all the options (see the LOFAR wiki[37]), and it is highly recommended that you obtain a hardcopy of this for future reference.

[36] Examples of these files can be found in `/globaldata/COOKBOOK/Parset`.

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

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.

6.7.1 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 6.10.

An example parset file[38] 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
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
```

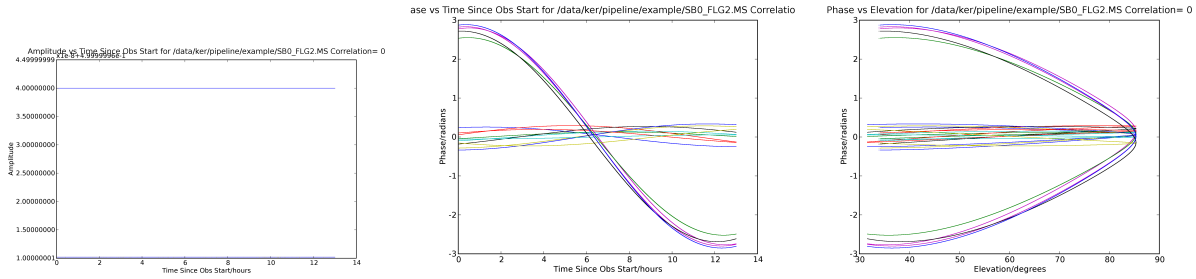
[38] This example can be found in `/globaldata/COOKBOOK/Parset/simulation.parset`.

```
# 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.

6.7.2 Gain calibration (direction independent)

The following parset[39] describes a direction independent gain calibration. The meaning of each parameter is the same as in Section 6.7.1 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.

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

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

```
# 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[40].

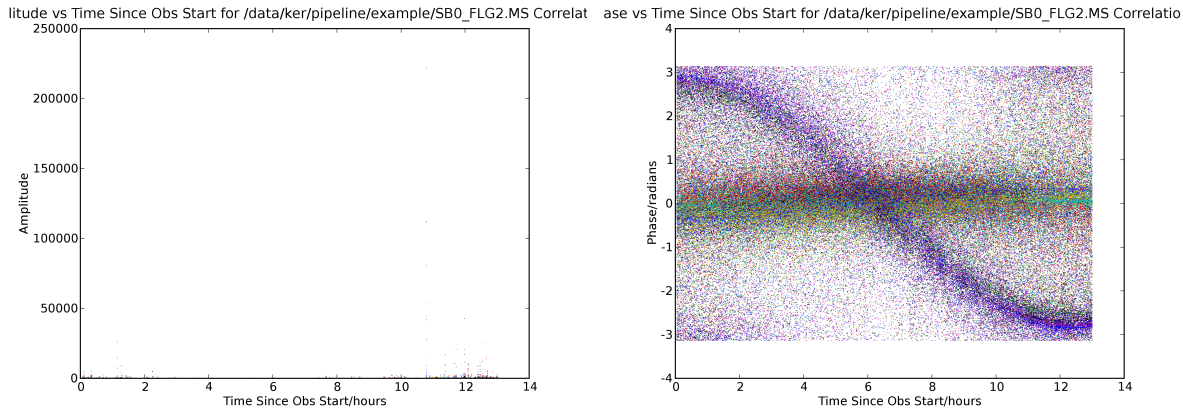
6.7.3 Gain calibration (direction independent, phase or amplitude only)

The parset file to perform a (direction independent) phase or amplitude calibration differs from the parset file for regular gain calibration (see Section 6.7.2) 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

[40] 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.



(a) Amp. vs. time.

(b) Phase vs. time.

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

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.

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[41].

```
############################################################
# 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 = *&
```

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

```
# 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
# 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
```

6.7.4 Gain calibration (direction dependent) with source subtraction

This section has been adapted from a document written by Annalisa Bonafede[42]. In the following, we report the parset[43] 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
#
```

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

[43] This example can be found in `/globaldata/COOKBOOK/Parset/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[44]:

```
############################################################
# 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
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.

6.7.5 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

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

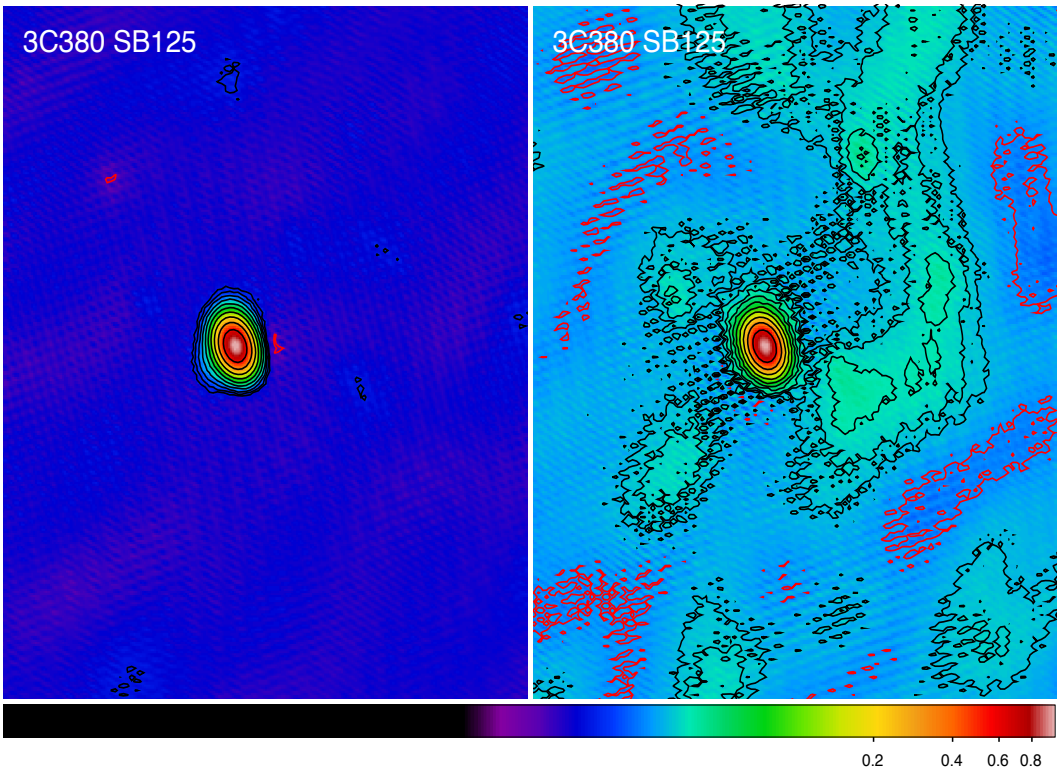


Figure 16: The radio source 3C380 imaged by LOFAR at 135 MHz (observation ID L2010_08567). *Left panel*: Self calibration has been applied with directional-gain correction and subtraction of Cyg A and Cas A. *Right panel*: Self calibration has been applied without correction. The lowest contour level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

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 6.9), 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`[45] can be used to plot differential TEC solutions per baseline, when run on an MS containing TEC solutions.

[45] `/opt/cep/tools/cookbook/Solution_Plotter.py`

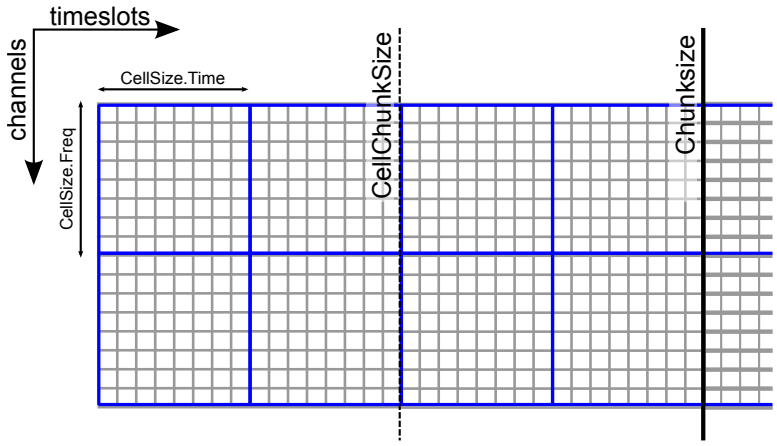


Figure 17: The different solve domains and chunk parameters for BBS. In this example, `CellSize.Time=8`, `CellSize.Freq=8`, `ChunkSize=32`, `CellChunkSize=2` (do not use these values yourself, they are probably not good for actual use).

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

You can instruct `calibrate-stand-alone`[46] to run with multithreading with the parameter `-t` or `--numthreads`:

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

6.9 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.

6.9.1 Setting up your environment

Before using the distributed version of BBS, you will have to set up a personal database (see Section 1.4). 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 (see also Section 1.5). This has to be done only once.

The distributed version of BBS requires a file that describes the compute cluster (an example of such a file is `cep1.clusterdesc`[47]) and a configuration or *parset*[48] 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 6.2).

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

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

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

6.9.2 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 6.7).
- `<source catalog>`, which defines a list of sources that can be used for calibration (see Section 6.3).
- `<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.

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[49] 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.

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

6.9.3 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:

- 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]
  ```

6.10 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.[50]

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, so for example it does not make a lot of sense if the FFT'ed image is larger that $\sim$
- `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/*"
```

[50] On CEP1, catalog files with up to ca. 10,000 CCs can be processed until the nodes run out of memory.

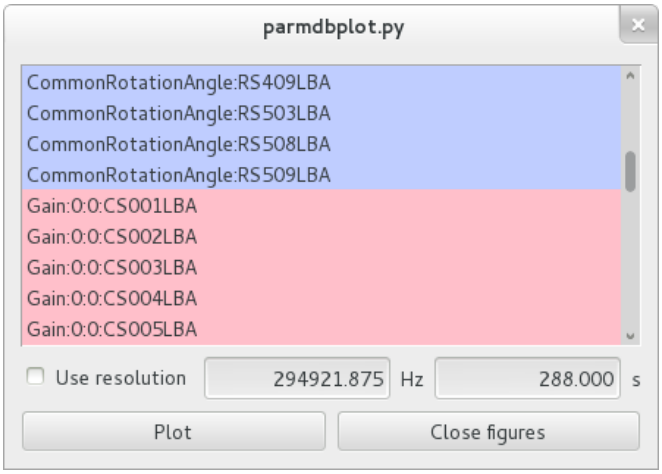


Figure 18: The main window of `parmdbplot`.

6.11 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.

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.

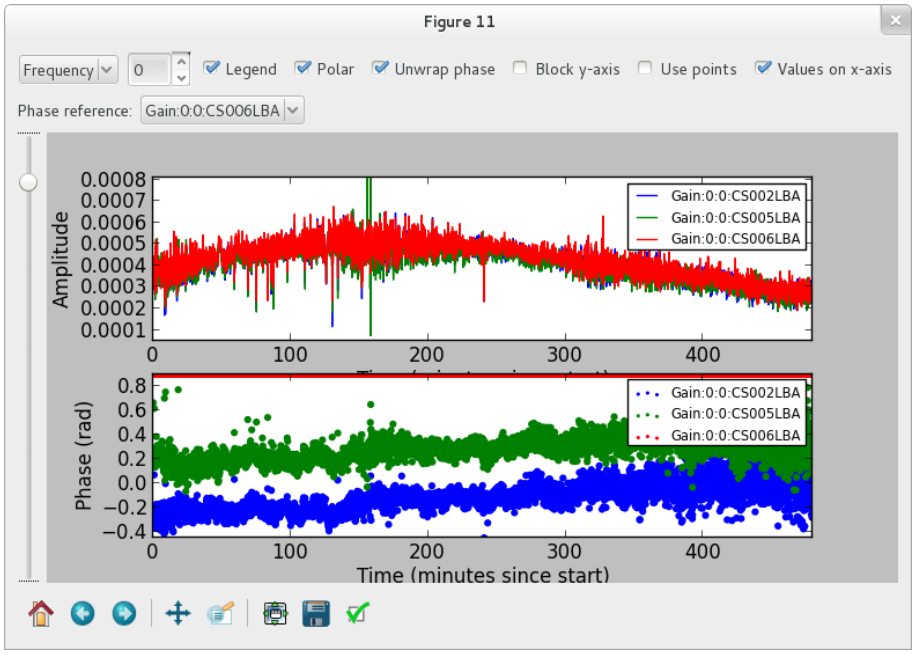


Figure 19: The plot window of `parmdbplot`.

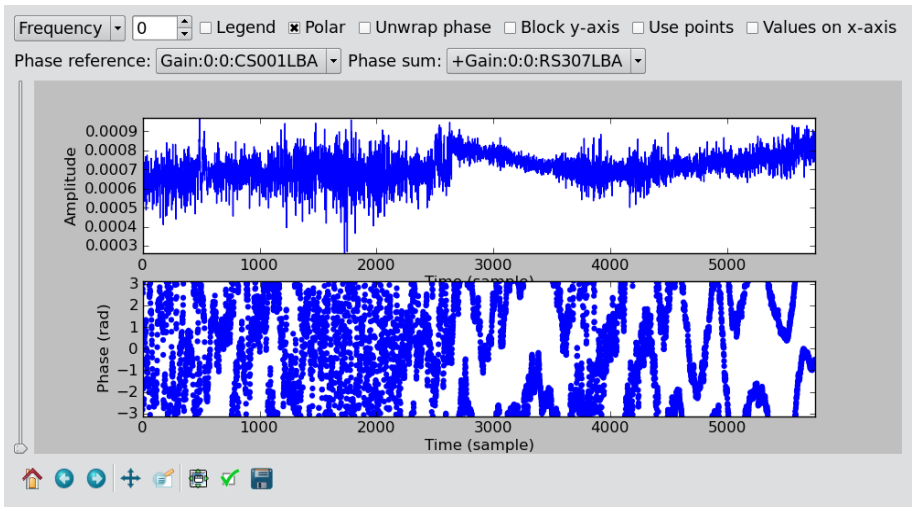


Figure 20: The plot window of `parmdbplot` with a phase sum

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.

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` script by George Heald:

```
>/home/heald/bin/solplot.py -q *.MS/instrument/
```

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

6.12 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).

6.12.1 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.

6.12.2 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.

6.13 Gain transfer from a calibrator to the target source[51]

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

[51] George Heald (`heald[at]astron[dot]nl`) contributed to the writing of this section.

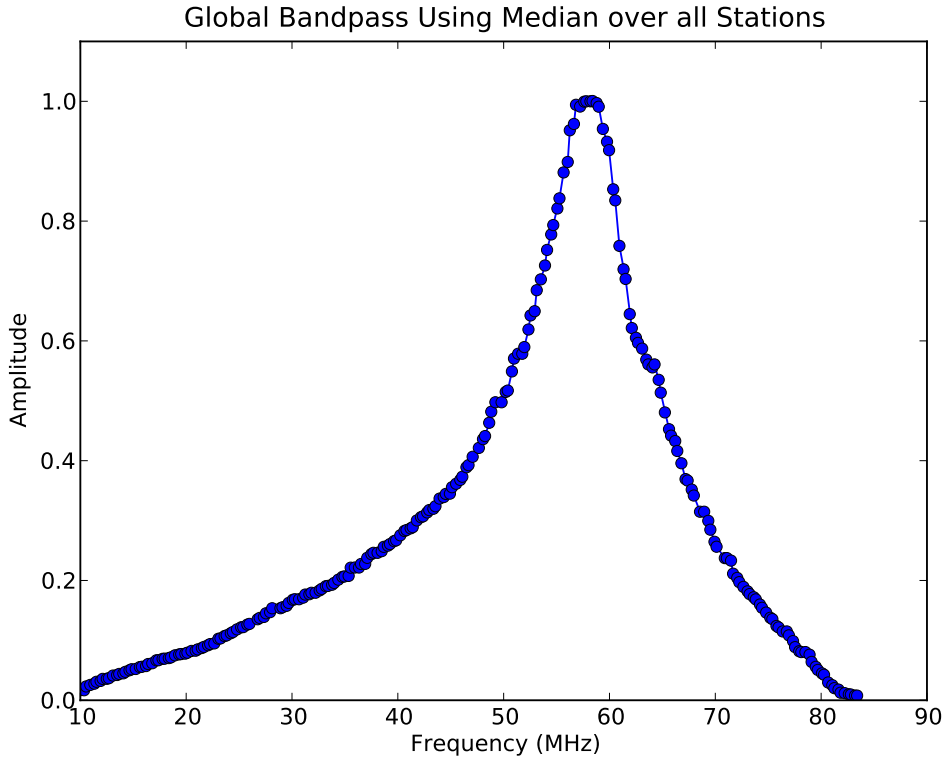
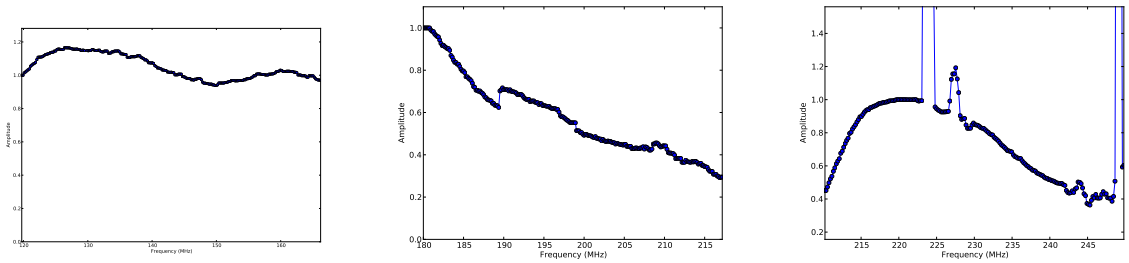


Figure 21: The global bandpass in LBA between 10–85 MHz.



(a) HBA-low (b) HBA-mid (c) HBA-high

Figure 22: The global bandpass for the HBA.

not suggested for long target observations, because the calibrator gains will only remain valid for a relatively short time (the validity is not trivial to quantify, but is probably less than an hour in all circumstances, and quite possibly significantly less).

- 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.

6.13.1 The "traditional" approach

When doing calibration transfer in the normal way, i.e. the calibrator and target are observed at different times, the gain solutions can be transferred using `parmdbm`. Full documentation for that program is available on the LOFAR wiki[52]. The procedure is as follows:

1. When calibrating the calibrator observation through BBS, it is important to obtain *time independent* solutions. Moreover, the frequencies should match the target subbands. In order to achieve time independence you should use these settings in the BBS parset[53]:

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

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

   ```
   > 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
   ```

3. Apply the gain solutions to target field. 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.

6.13.2 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:

[52] http://www.lofar.org/operations/doku.php?id=engineering:software:tools:parmdbm
[53] Note that these settings would be dangerous for a long observation!

- 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.
- 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. 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 = []
   ```

3. 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 NDPPP.

6.14 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 NDPPPP (see Section 5).

The script `CallSolFlag.py`[54] 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>
```

[54] This script can be found in `/opt/cep/tools/cookbook` and can be invoked directly after having initialized the `Tools` packages (see Sect. 1.3).

6.15 Troubleshooting

- Bugs should be reported using the LOFAR issue tracker[55].
- 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.
- 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[56].

6.15.1 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`[57] 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 as described in Section 1.4.

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

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

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

7 LoSoTo: LOFAR Solution Tool[58]

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. LoSoTo uses instead an innovative data file, called H5parm, which is based on the HDF5 standard[59].

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 cep1 users have to:

```
source /cep1home/fdg/scripts/losoto/tools/lofarinit.csh
```

7.1 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 `tables` module) and it has better performance than competitors.

7.1.1 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.

7.1.2 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).

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.

[58] This section is maintained by Francesco de Gasperin (`fdg[at]hs[dot]uni-hamburg[dot]de`).
[59] http://www.hdfgroup.org/HDF5/

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, and tecscreen.

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:

aplitudes : 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.

7.1.3 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 shapes
/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)
```

7.1.4 H5parm benchmarks

For a parmdb of 37 MB the relative H5parm with no compression is about 89 MB. Using a maximum compression, the H5parm ends up being 18 MB large. 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: ### Read all frequencies for a pol/dir/station
INFO: PARMDB -- 1.0 s.
INFO: H5parm -- 0.35 s.
INFO: ### Read all times for a pol/dir/station
INFO: PARMDB -- 0.98 s.
INFO: H5parm -- 0.34 s.
INFO: ### Read all rotations for 1 station (slice in time)
INFO: PARMDB -- 0.99 s.
INFO: H5parm -- 0.53 s.
INFO: ### Read all rotations for all station (slice in time)
INFO: PARMDB -- 12.2 s.
INFO: H5parm -- 4.66 s.
INFO: ### Read all rotations for remote stations (slice in ant)
INFO: PARMDB -- 4.87 s.
INFO: H5parm -- 1.91 s.
INFO: ### Read all rotations for a dir/station and write them back
INFO: PARMDB -- 1.33 s.
INFO: H5parm -- 1.08 s.
```

```
INFO: ### Read and tabulate the whole file
INFO: parmdb -- 0.94 s.
INFO: H5parm -- 0.02 s.
```

7.2 LoSoTo

LoSoTo is made by several components. It has some tools used mostly to transform parmdb to H5parm and back (see Sec. 7.2.1). 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.7.2.3).

7.2.1 Tools

There are currently four tools shipped with LoSoTo:

`parmdb_collector.py` fetches parmdb tables from the cluster

`H5parm_importer.py` creates an h5parm file from an instrument table (parmdb) or a globaldb created 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. 7.3.

7.2.2 Operations

These are the operations that LoSoTo can perform:

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

PLOT : plot solutions in 1D/2D plots or plot TEC screens.

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.

CLIP : clip all solutions a certain factor above/below the median value.

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

FLAG : iteratively remove a general trend from the solutions and then perform an outlier rejection. This operation is still to be implemented.

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

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.

CLOCKTEC : perform clock/tec separation.

TECFIT : fit TEC values per direction and station to phase solutions

TECSCREEN : fit TEC screens to TEC values

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

Beside these operations which require activation through the LoSoTo parset (see Sec. 7.2.3), one can call `losoto.py` with the "-i" option and 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)".

```
$ losoto.py -i single.h5

Summary of single.h5

Solution set 'sol000':
======================

Directions: 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': 2 pols, 2 dirs, 36 ants, 1 freq, 4314 times

Solution table 'rotation000': 2 dirs, 36 ants, 1 freq, 4314 times

Solution table 'phase000': 2 pols, 2 dirs, 36 ants, 1 freq, 4314 times
```

7.2.3 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,plot_again]	sequence of steps names
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 solution type
LoSoTo.ant	list of antenna names	[CS001_HBA]	restrict to these antennas
LoSoTo.pol	list of polarizations	[XX, YY]	restrict to these polarizations
LoSoTo.dir[a]	list of directions	[pointing, 3C196]	restrict to these pointing directions

[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...) have the direction: "pointing".

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. 7.2.2. All the global variables (except from the "steps" one) are also usable inside a step to change the selection criteria for that specific step. A list of step-specific parameters is given in Table 5.

7.3 Usage

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

1. Collect the parmdb of calibrator and target:

```
~fdg/scripts/losoto/tools/parmdb_collector.py -v -d "target.gds"
 -c "clusterdesc" -g globaldb_tgt
~fdg/scripts/losoto/tools/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).

2. Convert the set of parmdbs into an h5parm:

```
~fdg/scripts/losoto/tools/H5parm_importer.py -v tgt.h5 globaldb_tgt
~fdg/scripts/losoto/tools/H5parm_importer.py -v cal.h5 globaldb_cal
```

3. 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):

```
~fdg/scripts/losoto/tools/H5parm_merge.py -v cal.h5:sol000 tgt:cal000
```

4. Run LoSoTo using e.g. the parset given in Sec. 7.2.3:

Var Name	Format	Example	Comment
Operations	string	RESET	An operation among those defined in Sec. 7.2.2
RESET			
Weight	bool	0 | 1	True: reset also the weights[a]
PLOT			
PlotType	1D | 2D | TECScreen	1D	Type of plot
Axes	list of axes names	[time]	For 1D plots is one axis name, two axes for 2D plots
MinMax	[float, float]	[0,100]	Force a min/max value for the dependent variable
Prefix	string	images/test_	Give a prefix to all the plots
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
ABS			
CLIP			
Axes	list of axes names	[freq, time]	Axis name to take medians over, may be multiple
ClipLevel	float	5	factor above/below median at which to clip
FLAG (TBI)			
NORM			
NormVal	float	1.	the value to normalize the mean
NormAxis	axis name	time	the axis to normalize
INTERP			
CalSoltab	soltab name	cal001/amplitude000	The calibrator solution table
CalDir	dir name	3C196	Use a specific dir 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	0 | 1	Just rescale to the median value of CalSoltab, do not interpolate
MedAxis	axis name	time	rescale to the median of this axis
CLOCKTEC (TBI)			
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

[a] note that weights are currently not propagated back to parmdb

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

```
~fdg/scripts/losoto/losoto.py -v tgt.h5 losoto-interp.parset
```

5. Convert back the h5parm into parmdb:

```
~fdg/scripts/losoto/tools/H5parm_exporter.py -v -c tgt.h5 globaldb_tgt
```

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

7.4 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 [60].

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

[60] 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.

combine more subbands before doing a phase selfcalibration 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.

8 SAGECAL[61]

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 (0.2.5) of SAGECAL and older versions are obsolete.

8.1 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 $\mathcal{O}((KN)^2)$ for the Least Squared, while is $\mathcal{O}(KN^2)$ 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 CEP1 and CEP2. 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 direction (in the GPU EoR cluster not in CEP1 or CEP2). 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.

8.2 Using SAGECAL

8.2.1 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 NDPPP. You are now ready to start the algorithm.

All the programs related to the SAGECAL can be found in **/opt/cep/sagecal/bin/**.

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

8.2.2 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 12) . Note that you are free to use any other source finder (like PyBDSM - see (Chapter 10)) 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 12. 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.

8.2.3 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 follow:

```
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:

```
-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
-o robust rho, robust matrix inversion during correction: default 1e-09
-j 0,1,2... 0 : OSaccel, 1 no OSaccel, 2: OSRLM, 3: RLM: default 0
-L robust nu, lower bound: default 2
-H robust nu, upper bound: default 30
-R randomize iterations: default 1
```

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.

You are now ready to image the residual data. Successively, run "restore" (see Sect. 12) 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.

8.2.4 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.

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,” *MNRAS*, 2013.

9 The AW imager[62]

9.1 Introduction

In this section we will describe the necessary steps needed to perform successful imaging of LOFAR data using the AWimager. Note that the AWimager is still in the development phase, therefore this documentation is very dynamic and it is meant to provide the basic instructions on how to use the code.

9.2 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. In the future it might become available in the CASA software.

9.3 Usage

To run AWimager, you first need to setup your environment using:

```
use LofIm
```

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.

9.4 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.

[62] This Chapter is maintained by Bas van der Tol (`tol[at]astron[dot]nl`).

- <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.

9.5 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.

9.5.1 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.

9.5.2 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.

9.5.3 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.

9.5.4 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.

9.5.5 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

9.5.6 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[63] (specifically those from September 28 and October 26 2011).

[63] http://www.lofar.org/operations/doku.php?id=commissioning:busy_wednesdays

10 Source detection: PyBDSM[64]

10.1 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 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.

10.2 Recent Changes

Changes to PyBDSM since the last version of the cookbook include:

- New output options: Enabled output of images in CASA format in the `export_image` task (`img_format = 'casa'`). Added an option to the `export_image` task to export an island-mask image, with ones where there is emission and zeros elsewhere (`image_type = 'island_mask'`). Features in the island mask may be optionally dilated by specifying the number of dilation iterations with the `mask_dilation` parameter. Added an option to write a CASA region file to the `write_catalog` task (`format = 'casabox'`). Added an option to write a CSV catalog to the `write_catalog` task (`format = 'csv'`). Added option (`bbs_patches = 'mask'`) to allow patches in an output BBS sky model to be defined using a mask image and enabled SAGECAL sky-model output in the `write_catalog` task.
- Improved wavelet fitting: Added option so that wavelet fitting can be done to the sum of images on the remaining wavelet scales, improving the signal for fitting (controlled with the `atrous_sum` option). Added option so that user can choose whether to include new islands found only in the wavelet images in the final fit or not (controlled with the `atrous_orig_isl` option).
- Renamed `blank_zeros` to `blank_limit`. The `blank_limit` option now specifies a limit below which pixels are blanked.
- Fixed many minor bugs (use `help changelog` for details).

10.3 Setup

The latest version of PyBDSM is installed on the CEP I and CEP II clusters. To initialize your environment for PyBDSM, run:

```
> use LofIm
```

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`.

[64] This section is maintained by David Rafferty (`rafferty[at]strw[dot]leidenuniv[dot]nl`).

10.4 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 10.6.

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 1.8.1 (LOFAR revision 28191)
================================================================================
PyBDSM commands
  inp task ............ : Set current task and list parameters
  par = val ........... : Set a parameter (par = '' sets it to default)
                          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.

10.5 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.

10.5.1 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 23. 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.

10.5.2 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 24. 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 of 10 pixels (i.e., the box is moved across the image in 10-pixel steps). See Figure 25 for a summary of the results of this call.

10.5.3 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 26.

10.6 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

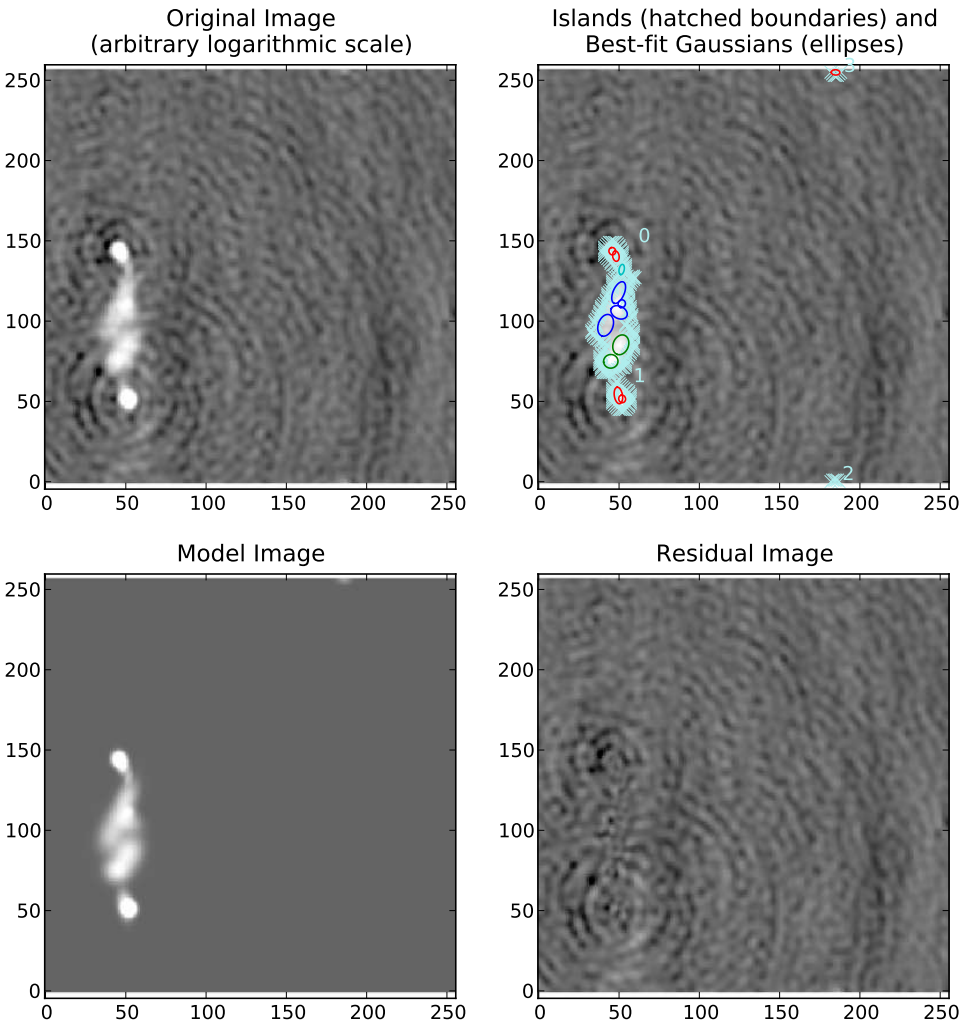


Figure 23: Output of `show_fit`, showing the original image with and without sources, the model image, and the residual (original minus model) image. Boundaries of the islands of emission found by PyBDSM are shown in light blue. The fitted Gaussians are shown for each island as ellipses (the sizes of 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.

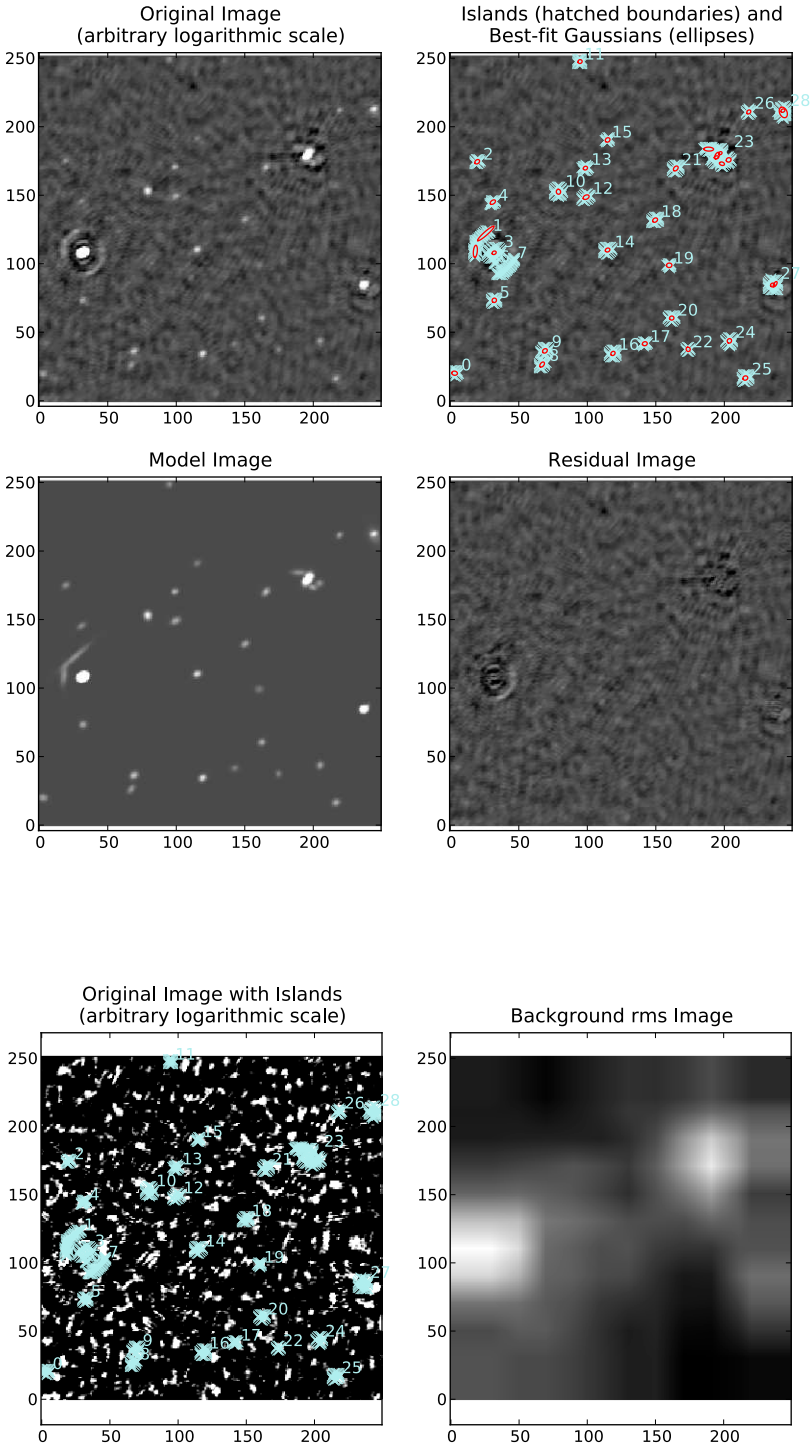


Figure 24: Example fit with default parameters of an image with strong artifacts around bright sources. A number of artifacts near the bright sources are picked up as sources. The background rms map for the same region (produced using `show_fit`) is shown in the lower panel: the rms varies fairly slowly 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).

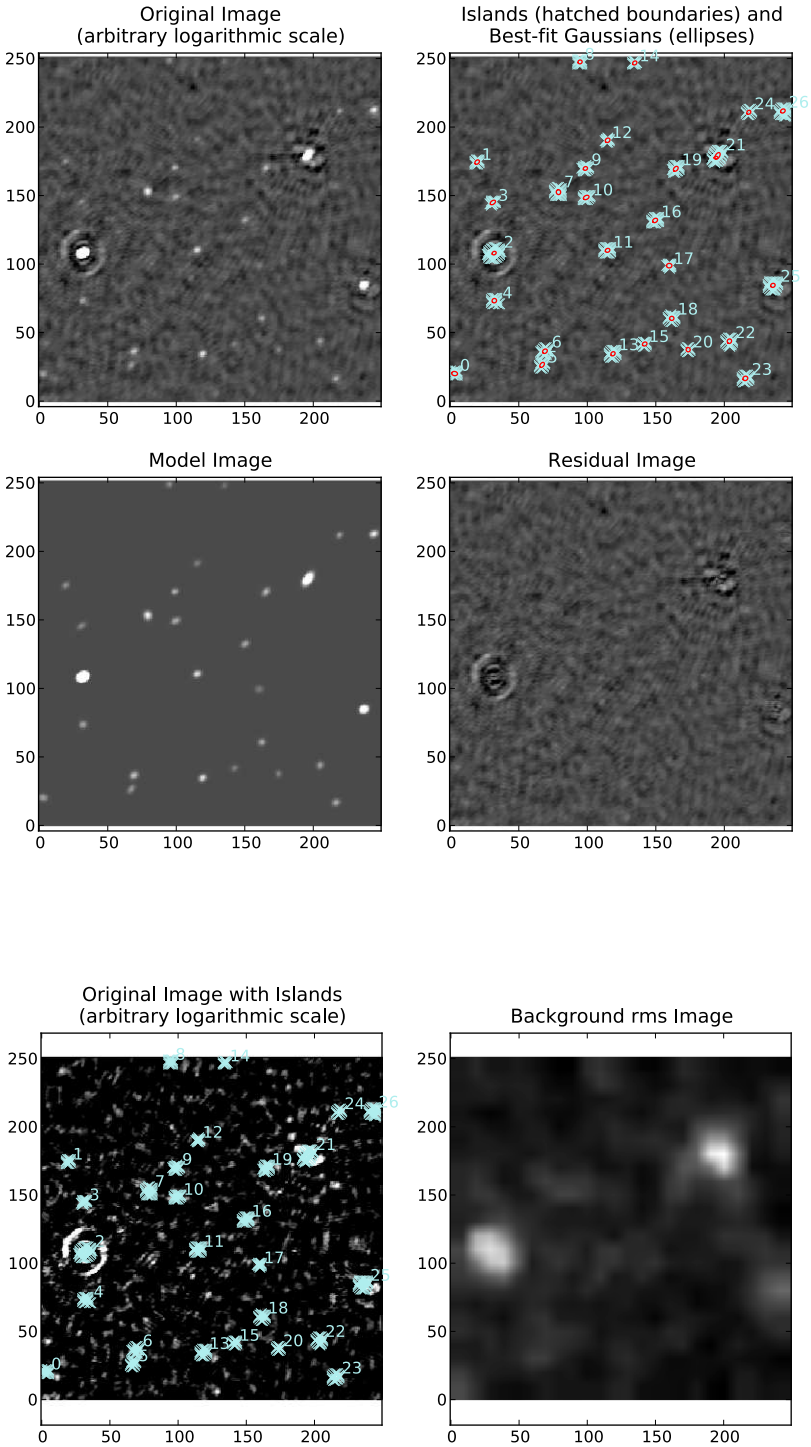


Figure 25: The same as Figure 24, but with `rms_box=(20,10)`. The rms map now varies on scales similar to that of the regions affected by the artifacts, and both bright and faint sources are recovered properly.

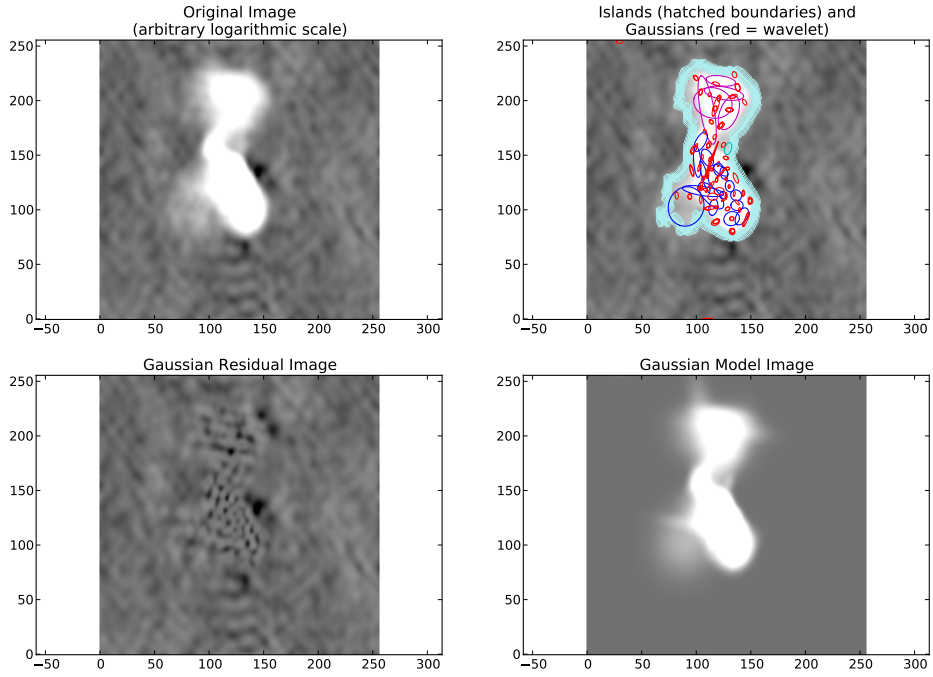


Figure 26: Example fit of an image of Hydra A with `rms_map=False`, `mean_map='zero'`, and `atrous_do=True`. The values of `thresh_isl` and `thresh_pix` were adjusted before fitting (by setting `interactive=True`) to obtain an island that enclosed all significant emission.

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 Automated Self-Calibration[65]

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`, `NDPPP`, `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

11.1 Overview

The goal of the self-calibration process is to improve the quality of the final image, through an iterative process. At each iteration, a calibration step is performed[66] 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 28, the highlighted segments of the pipeline correspond to the ones used within the selfcal process (including the loop itself).

11.2 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 CEP1, CEP2 and on Flits (ASTRON Astronomy group cluster).

11.3 Selfcal: Stand alone version

11.3.1 Usage

`selfcal` in currently in the `LofIm` package[67]. The self-calibration procedure is initiated by typing `selfcal.py` at the command prompt. If you provide no options, a description of the needed parameters is provided:

```
MISSING Parameters
Usage:  selfcal.py --obsDir= --outputDir= --nbCycle=
```

[65] This section is maintained by Nicolas Vilchez (`vilchez[at]astron[dot]nl`) and George Heald (`heald[at]astron[dot]nl`).

[66] In the current version, the automatic script performs phase calibration only.

[67] Note that the `LofIm` package conflicts with the `LUS` package (especially for `AWimager`). To use selfcal, you must disable `LUS` from your environment (i.e erase `LUS` from the `.mypackages` file in your `$HOME` directory)

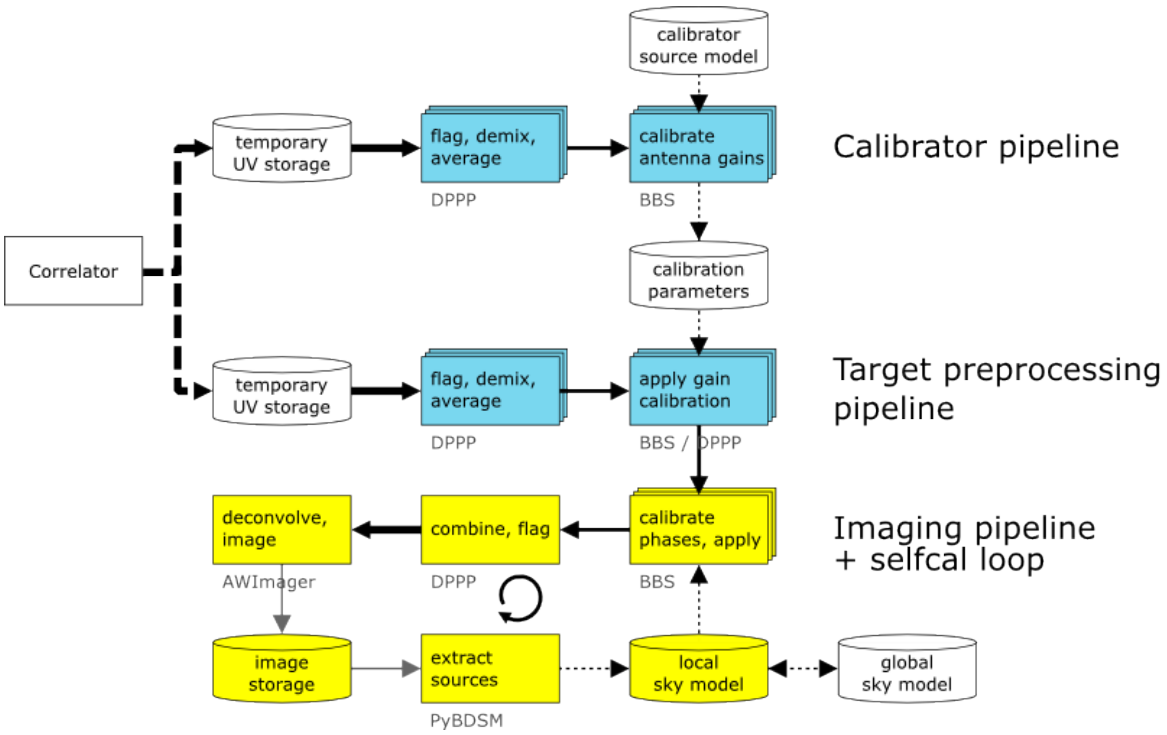


Figure 27: Illustration of the selfcal process in the Standard Imaging Pipeline.

The input parameters are:

- `--ObsDir`: This parameter specifies the full path of the input visibility data (Measurement Sets). Note that the full path must be provided. This directory should contain Measurement Sets corresponding to one or more individual snapshots, each with a number of bands (groups of 10-20 contiguous subbands). For further details of the assumed input data structure, please see Sect. 11.3.2.
- `--outputDir`: This parameter specifies the full path of the output data (images, catalogs, etc). Note that the full path must be provided.
- `--nbCycle`: This parameters allows you to specify the number of iterations to perform in the self-calibration loop. It can be a number in the range $5-20$. Generally speaking, requesting a large number of iterations can be useful to avoid divergence of the process, but comes at the cost of increased time to complete the procedure.

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.

11.3.2 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 then the required parameters are listed:

```
MISSING Parameters
Usage:  mergeSB.py --obsDir= --outputDir= --column2Merge=
```

The required parameters are:

- `--obsDir`: 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 $10 \times 5 = 50$ MS. It is often the case that one or more Measurement Sets are missing. This is acceptable, because NDPPP (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 subbands for each time chunk are present for all time chunks. The path given for `--obsDir` must be a full path.
- `--outputDir`: 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 `NDPPP_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.

- `--column2Merge`: With this parameter you can specify the data column that you want to merge. Usually, it is `DATA` or `CORRECTED_DATA`.

11.3.3 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 `NDPPP`
3. Imaging using `AWimager`
4. Sky model extraction using the `PyBDSM` source finder.[68]

The diagram in Figure 28 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 `NDPPP`. It is necessary to calibrate and flag them separately because it is not possible (at the moment) to calibrate non continuous time concatenated MS 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. 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. It starts at `robust=1` (nearly natural weighting) at low resolution (15 times the best resolution) and ends at `robust=-2` (uniform weighting) when including all available baselines.

11.3.4 Selfcal examples

In Figure 29, an example of the self-calibrated process is shown for HBA-low. The example dataset had the following properties:

- 31 time chunks
- 10 contiguous subbands
- Best beam resolution = 5arcsec

An example of using self-calibration on an example LBA dataset is shown in Figure 30. This LBA dataset contains 10 subbands and the total duration of the observation is 4.5 hr. 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

[68] 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).

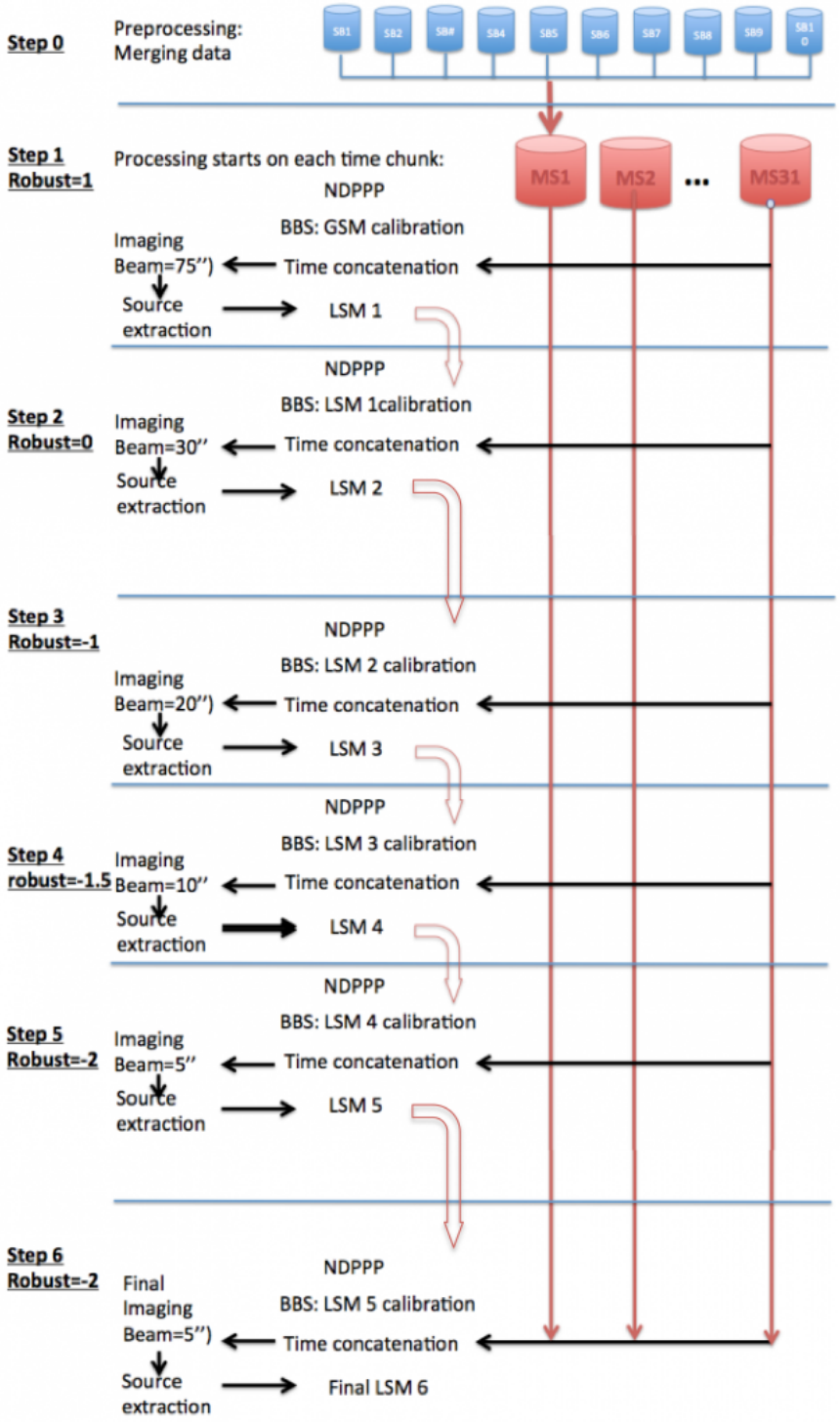


Figure 28: Illustration of the selfcal process in the global pipeline.

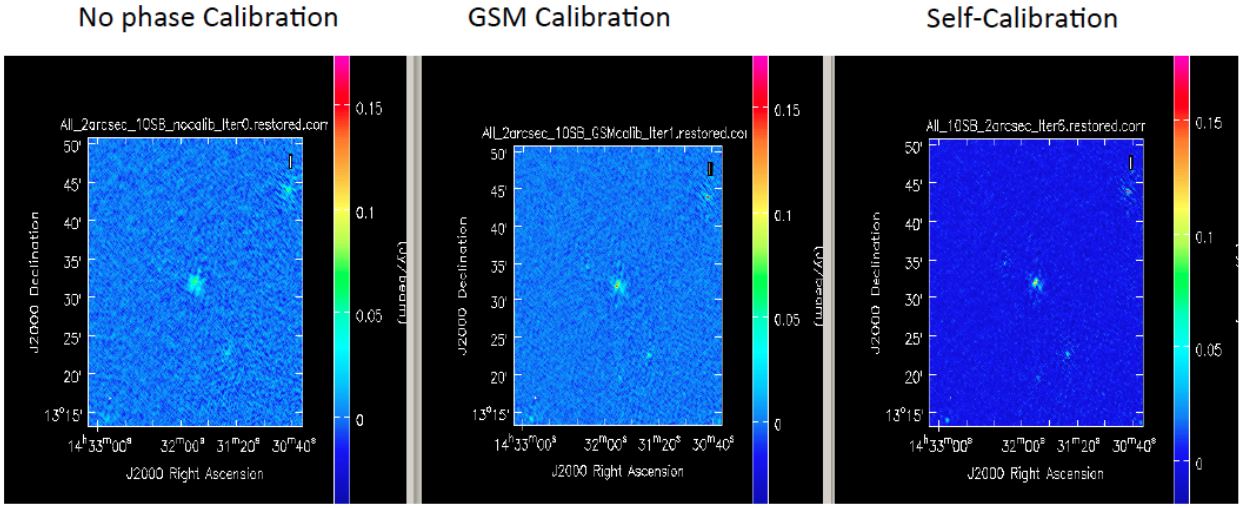


Figure 29: Results of selfcal process with an example HBA dataset at approximately 140 MHz. There are clear improvements in the image noise after performing self-calibration. The noise levels in the three images are 30, 10, and 2 mJy/beam for the cases of: no phase calibration (*left panel*), standard 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.

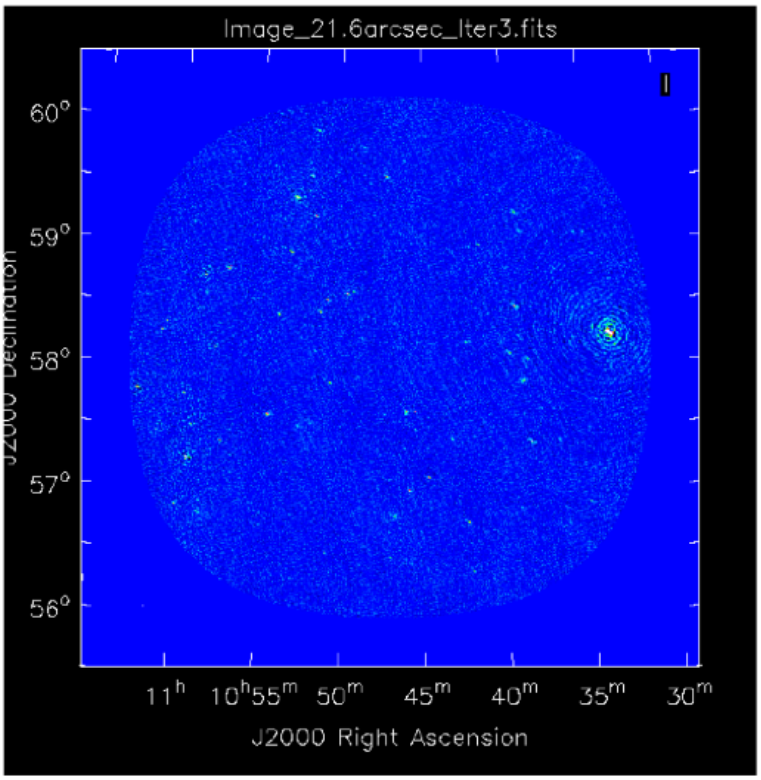


Figure 30: Results of selfcal process with LBA (∼60 MHz). The noise level in the image is 22 mJy/beam. The thermal noise is estimated to be approximately 11 mJy/beam for this dataset.

time. Further improvements will require direction dependent calibration. It does however appear that `selfcal` manages to produce images with final noise levels only about 2-3 times the nominal thermal noise, in both HBA and LBA.

12 Sky Model Construction Using Shapelets[69]

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.

12.1 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[70]. 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. 31, 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. 31, 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.

12.2 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.

[69] The author of this chapter is Sarod Yatawatta (yatawatta[at]astron[dot]nl).

[70] `http://skyview.gsfc.nasa.gov/cgi-bin/skvadvanced.pl`

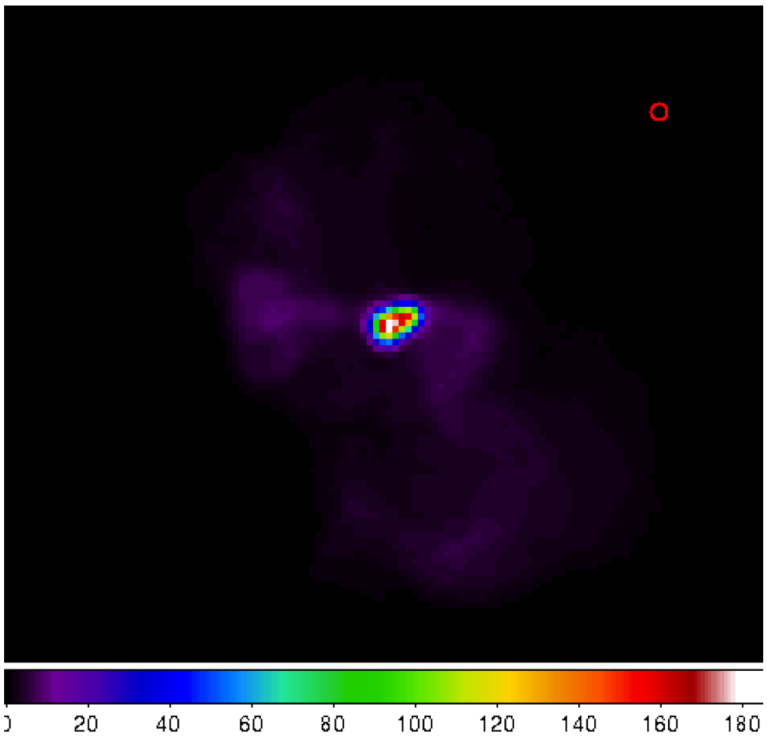


Figure 31: Virgo-A image made by the VLA at 74 MHz. The red circle on top right corner is the PSF.

12.2.1 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.

12.2.2 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.

12.2.3 ds9 and kvis

We use both `ds9` and `kvis` to display FITS file as well as display regions (`ds9`) and annotations (`kvis`).

12.2.4 Duchamp

The source extraction program `Duchamp` is written by Matthew Whiting[71]. 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.

12.2.5 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
```

[71] `http://www.atnf.csiro.au/people/Matthew.Whiting/Duchamp/`

12.2.6 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` for LSM. Note that `buildsky` will now (version 0.0.5) only produce LSM with 3rd order spectra.

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.

As before, you can see more options by typing

```
restore -h
```

12.2.7 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. 32.

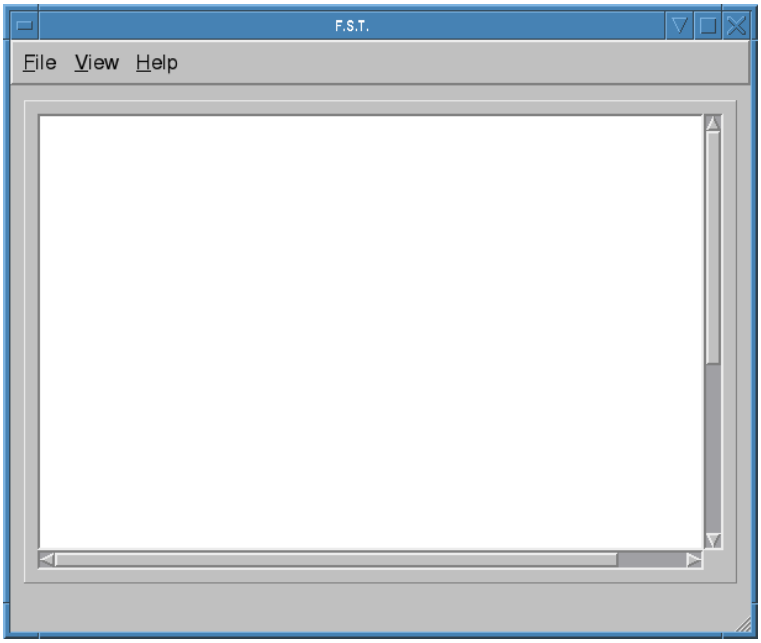


Figure 32: The `shapelet_gui` initial screen.

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. 33. We will go through the options in Fig. 33 one by one.

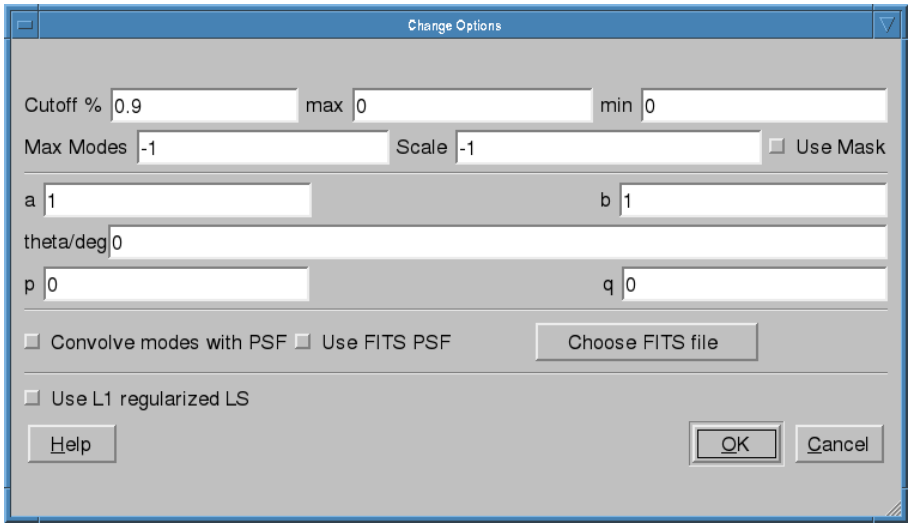


Figure 33: The options dialog for shapelet decomposition.

- **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.
- **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. 34. 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
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 --
```

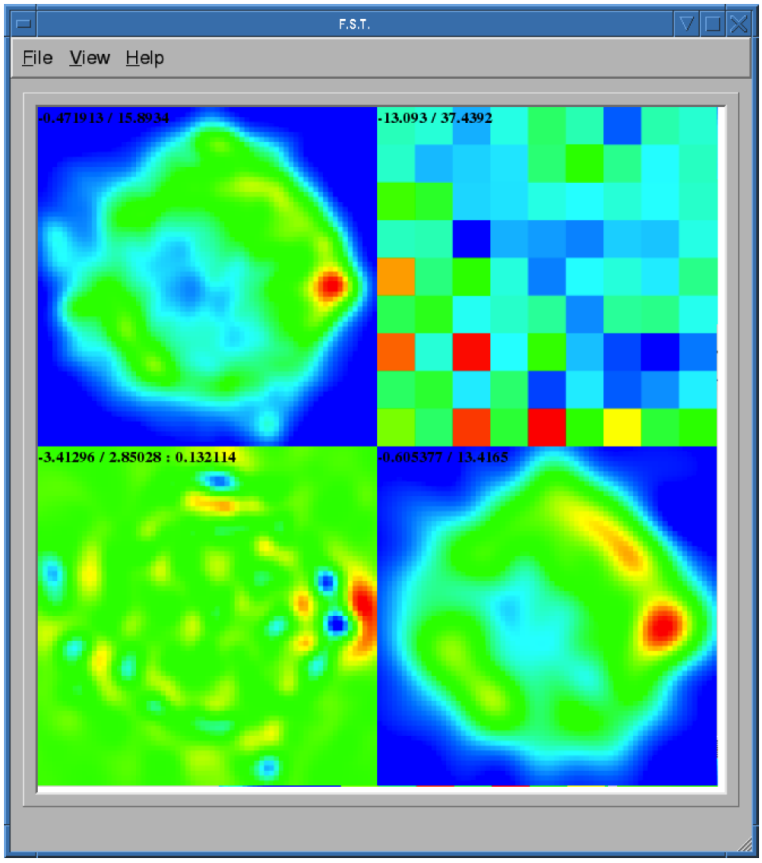


Figure 34: Output of shapelet modelling: (top left) original image (top right) shapelet modes (bottom left) residual image (bottom right) shapelet model.

```
# 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`.

12.2.8 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
```

12.3 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. 31 (FITS file `vira-cen.fits`) to build the initial sky model.

12.3.1 Initial point source model

As we mentioned in section 12.1, the central compact part in Fig. 31 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. 35. Now we run `buildsky` to build the sky model for this as

```
buildsky -f vira-cen.fits -m vira-cen.MASK.fits
```

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
```

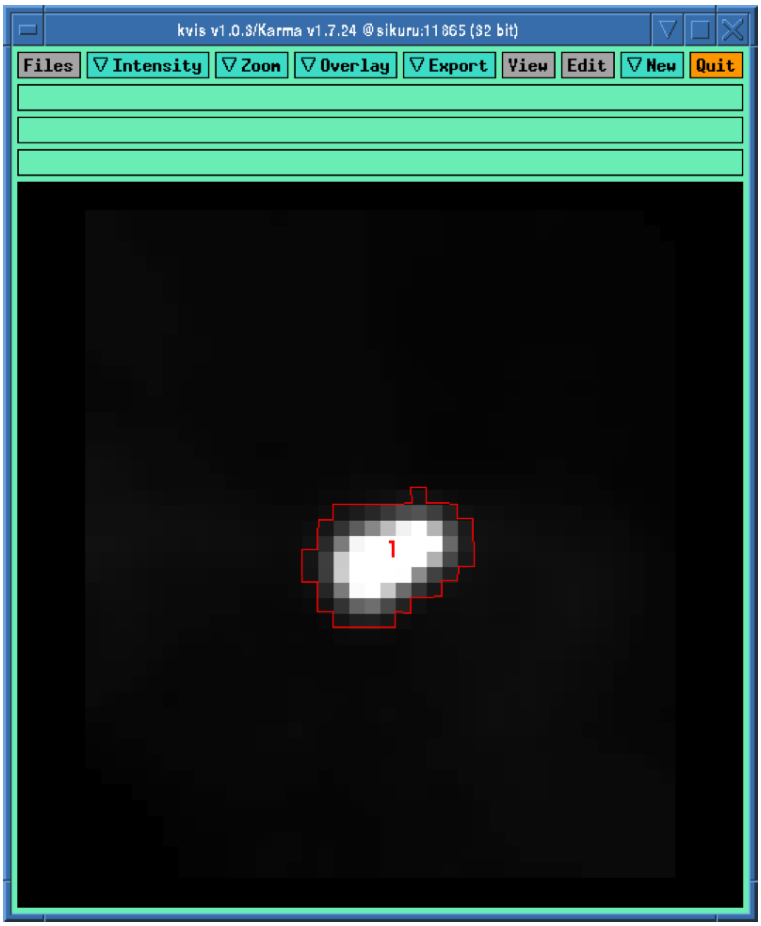


Figure 35: Compact center indicated by the red curve.

```
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. 36.

12.3.2 Initial shapelet model

Next, we need to model the extended structure inf Fig. 31. 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. 37. Note that the bright central part in Fig. 37 is almost subtracted. It is not completely gone, and some parts of it is negative. Nevertheless, this is all right for 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
```

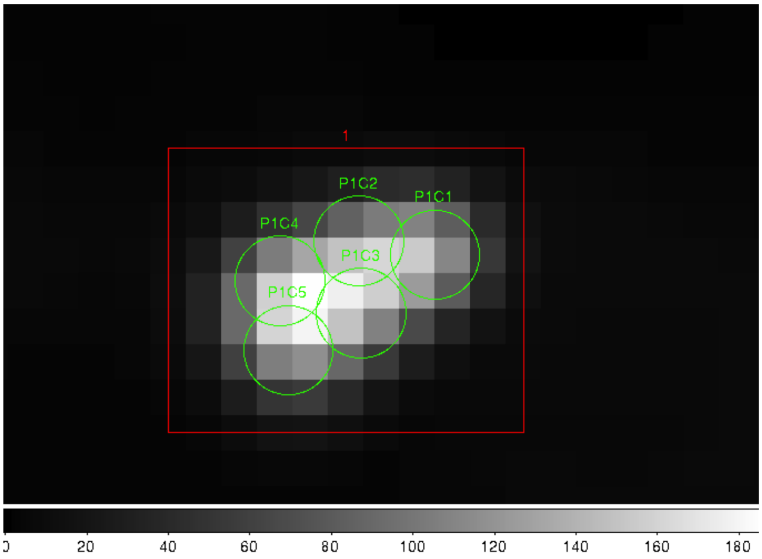


Figure 36: Compact center modeled by two point sources (green circles).

```
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. 38.

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. 39.

We can easily create an input to BBS for this shapelet model as follows:

```
#
FORMAT = Name Type RA Dec I IShapelet

VirAD shapelet   12:30:48.317433 12.23.27.999947 1.0  vira-cen.fits.modes
```

12.3.3 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',
```

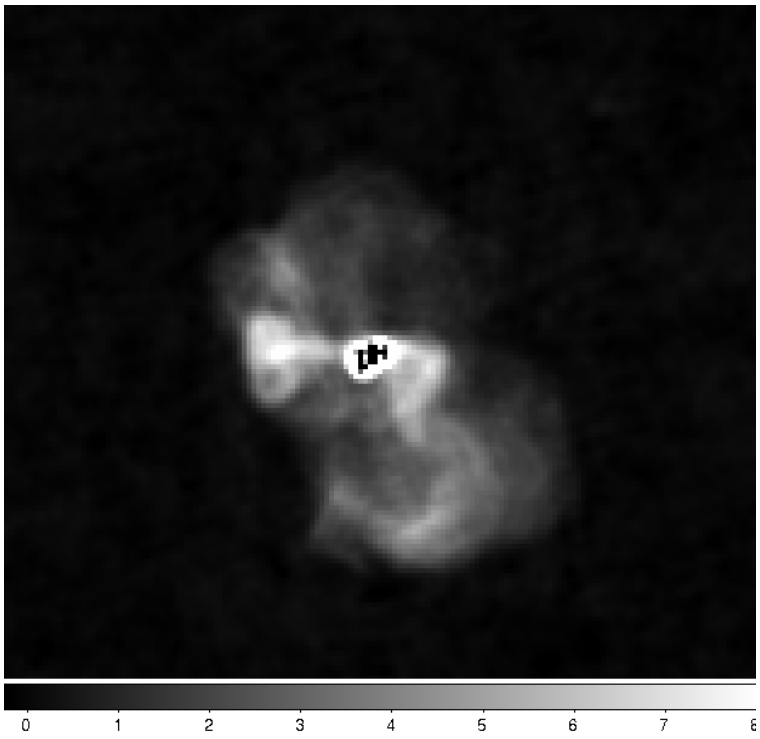


Figure 37: Diffused structure after subtracting the center.

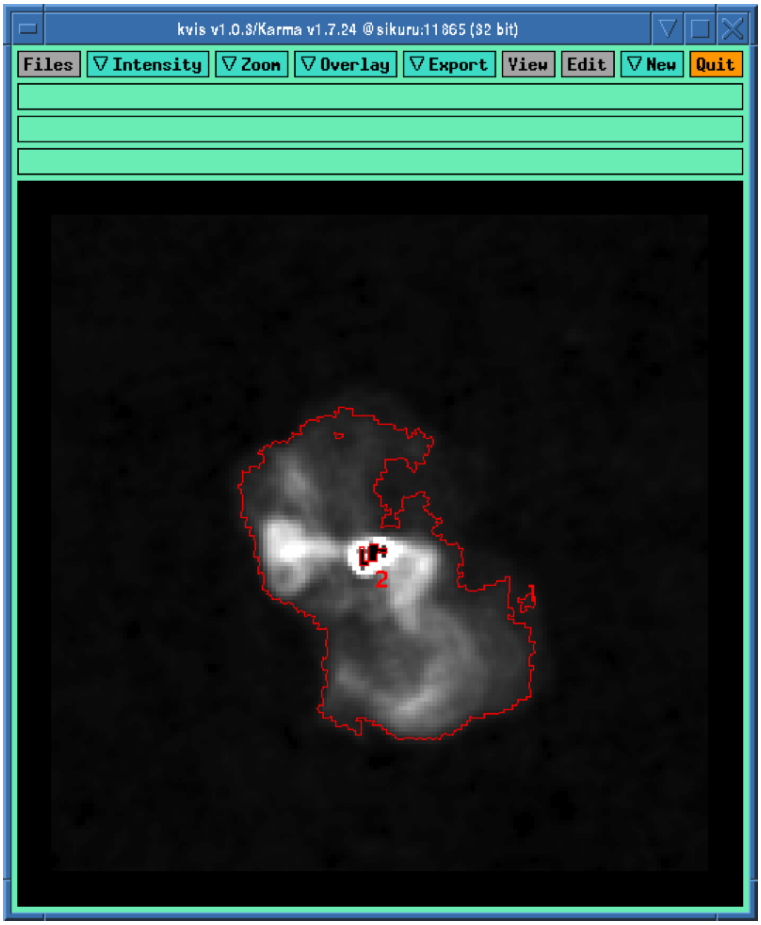


Figure 38: Mask for the diffused structure.

```
SpectralIndex='[0.0]', Ishapelet) = format
```

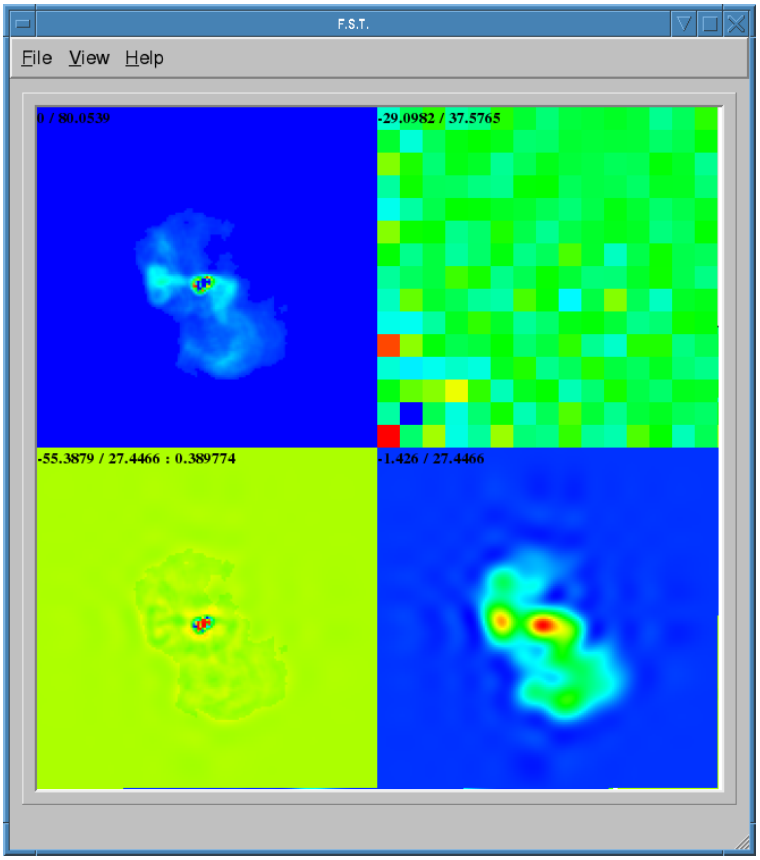


Figure 39: Shapelet model of the diffused structure.

```
# 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.

12.3.4 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. 40.

By looking at Fig. 40, we do not see any major discrepancy in our sky model (although we have lower resolution) so we go ahead with calibration.

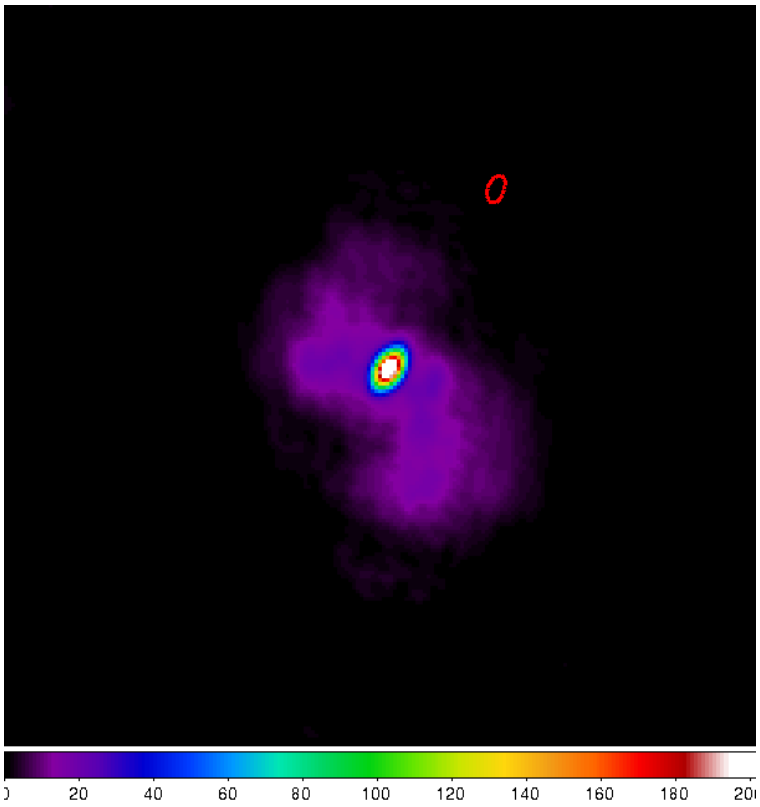


Figure 40: Simulated image of Virgo-A. The red ellipse is the PSF.

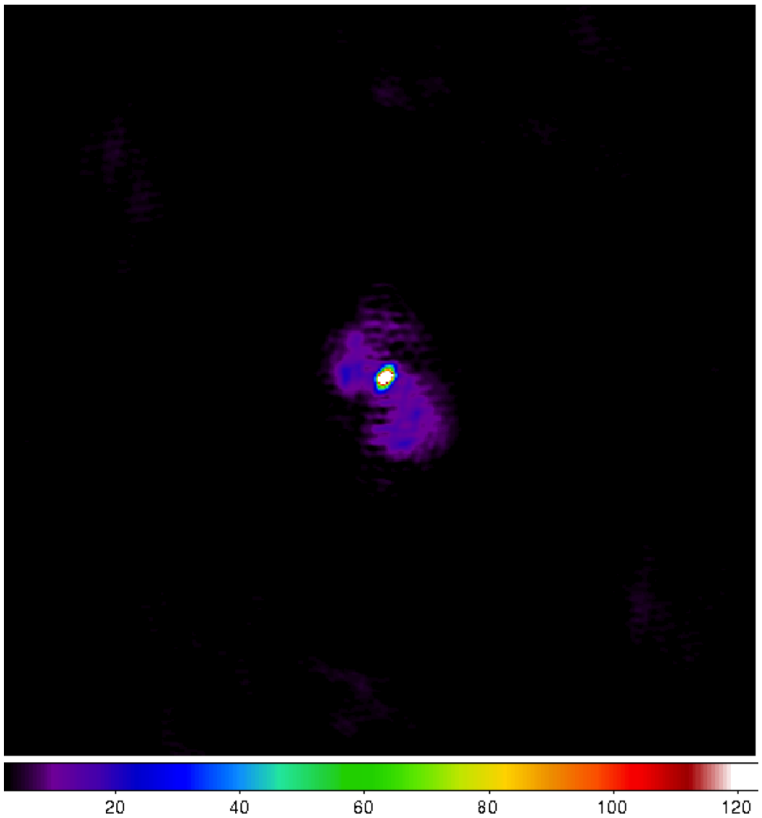


Figure 41: Calibrated image of Virgo-A (uniform weights).

12.3.5 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. 41.

NOTE: It is advised to use uniform weights to compare the calibrated image to the model image.

Using Fig. 41, we can repeat our sky model construction to get a better result. This of course depends on your science requirements.

12.3.6 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. 42, 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.

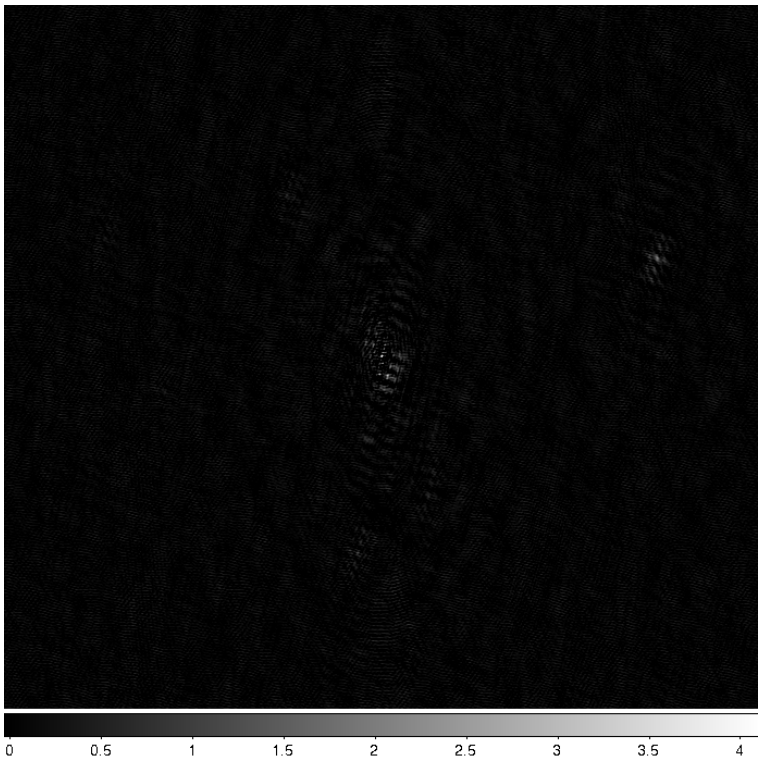


Figure 42: Residual image of Virgo-A. An off center source is present on top right hand corner.

12.3.7 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.

12.4 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.

13 Practical examples[72]

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.

13.1 Cygnus A: a bright source at the centre of the map

In this exercise, the User will calibrate an HBA dataset for Cygnus A. By the end of the exercise the User should be able to

- inspect raw LOFAR data,
- automatically and manually flag data within NDPPP,
- calibrate the data with BBS,
- produce maps with CASA,
- create a sky model from the data, and,
- iterate using the skills that have been learnt to self-calibrate the data to make a final image.

The raw data set for this exercise can be found at

`/data/scratch/tutorials/cyga/L24921_SB005_uv.MS`

on the compute nodes lce055, lce056, lce057. The unique LOFAR observation number is `L24921` and the sub-band that will be used is `SB005` The data set is in Measurement Set (MS) format.

13.1.1 Inspecting raw data

Log into one of the compute nodes above, and make a new directory, for example (you can substitute `$USERNAME` by your username),

```
> ssh lce055 -Y
> cd /data/scratch/$USERNAME/
> mkdir TUTORIAL/
> cd TUTORIAL/
> pwd
/data/scratch/$USERNAME/TUTORIAL
```

You will be using the LOFAR software tools. To initialize these, use

[72]The authors of this Chapter are John McKean (`mckean[at]astron[dot]nl`) and 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.

```
>  use LofIm
```

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=/data/scratch/tutorials/cyga/L24921_SB005_uv.MS verbose=T
```

From here, you should see that 27 stations were used for this observation, the frequency is 133 MHz and that there are 64 spectral channels. 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.

13.1.2 Flagging and data compression

The data set that we are using is uncompressed and unflagged; the total size of the data set is 11 Gb. Typically, LOFAR data will be flagged and compressed (in time and frequency) by the Radio Observatory, but in this exercise we shall do this process ourselves.

The data flagging and compression are carried out using NDPPP (see Chapter 5 for details). Typically, the Radio Observatory will compress the data to 3 and 10 s in time for the LBA and HBA data, respectively, and to 16 spectral channels. Since we are only interested in the bright source at the centre of the field we will compress the sub-band to 1 channel in frequency and 15 s in time. Note that the limitation on the compression in time is set by the changes in the ionosphere, which we will return to later in the reduction process.

The parset file for the flagging and compression[73] should be copied to your working directory,

```
> cp /home/mckean/TUTORIALS/CYGA/NDPPP.1.parset .
> more NDPPP.1.parset
```

```
msin = /data/scratch/tutorials/cyga/L24921_SB005_uv.MS
msin.startchan = 1
msin.nchan = 62
msin.datacolumn = DATA

msout = "L24921_SB005_uv.MS"
msout.datacolumn = DATA

steps = [preflag,flag1,count,avg1,flag2,avg2,count]

preflag.type=preflagger
preflag.corrtype=auto # flags the autocorrelations

flag1.type=madflagger
flag1.threshold=4
flag1.freqwindow=31
flag1.timewindow=5
flag1.correlations=[0,3] # flags the XX and YY polarizations

avg1.type = squash
avg1.freqstep = 64
avg1.timestep = 1        # compresses to 1 channel
```

[73] Note that in this example we are flagging the data using the madflagger algorithm. Good results could also be obtained by using the aoflagger algorithm as standalone (Sect. 4) or within NDPPP (Sect. 5)

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

avg2.type = squash
avg2.timestep = 5        # compresses 5 time-slots i.e. 15 s
```

This parset file will take the data set, flag and compress and then make a new copy in your working area. The windows in frequency and time that are used to generate the statistics needed to find outliers have been determined from extensive testing and on average, should give good results. Edit the msin and msout fields to point at your working directory. To run NDPPP, use

```
> NDPPP NDPPP.1.parset > ndppp.txt
```

Depending on the use of the cluster, it will take about ~ 10 minutes to compress and flag the data (see the percentage progress bar). You can inspect the output log file by using

```
> more ndppp.txt
```

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 autocorrelations have been flagged (those baselines with 100% flagging) and that the total data flagged for this data set is 22.7%; this is actually quite large for a LOFAR dataset. As we will see later, several hours of data were flagged by the correlator in some baselines.

The flagged and compressed data set should now be in your working directory and have a total size of 113 Mb, which is much more manageable than before. You can look at a summary of this data set using

```
> msoverview in=L24921_SB005_uv.MS
```

```
msoverview: Version 20110407GvD
================================================================================
           MeasurementSet Name:  L24921_SB005_uv.MS       MS Version 2
================================================================================
 Observer: unknown     Project: MSSS
Observation: LOFAR
Antenna-set: HBA_ZERO

Data records: 996786       Total integration time = 39610 seconds
   Observed from   02-Apr-2011/01:00:00.0   to   02-Apr-2011/12:00:10.0 (UTC)

Fields: 1
  ID   Code Name          RA             Decl           Epoch
  0          BEAM_0       19:59:28.2900 +40.44.02.0000 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      1 TOPO  133.398438  189.208984  189.208984  133.398438  XX  XY  YX  YY
```

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 L24921_SB005_uv.MS L24921_SB005_uv.MS.copy
```

13.1.3 Post-compression inspection and flagging

A good way to visualize the data is to use the CASA task plotms. Only limited information for using plotms is given here, the user is directed to the CASA cookbook[74] for full details,

```
> use Casa
> casaplotms
```

A GUI will open that will allow you to plot the various properties of a data set (Amp vs. Time, Phase vs. Time, Amp vs. UV Distance, etc.). The tabs on the left can be used to load the data set and select which parts of the data are plotted (data), change the axes that are being plotted (axes), and change the colours of the data, for clarity (display).

It is often useful to only plot the "corr= xx, yy". Different antenna pairs (i.e. baselines) can be plotted using "antenna=21&22; 5&6".

In the data tab select "browse" and choose the Measurement Set. In the display tab select "colorize by: Antenna 2". Press plot. You will see a plot of all of the visibility amplitudes as a function of time (see Fig. 43), with each colour representing a different baseline.

It is also useful to look at the visibility amplitude as a function of uv distance. This can be done by pressing the axes tab and selecting "UVDist_L" in the X-axis tab.

Try selecting different correlations, colours and axes to plot against each other to become familiar with plotms.

Return to the plot of visibility amplitudes as a function of time (see Figure 43; top left). We see that one of the baselines shows a large amplitude. From a plot of the visibility amplitude as a function of uv distance we see that this baseline has an unusually large amplitude with respect to the other baselines. It may be possible to correct this during the calibration process, but lets identify and flag this baseline as an example.

At the bottom of the plotms window are some action buttons. Select the box with the green mark (see Fig. 43; top right) and mark a box around the visibilities with the large amplitudes. Select the action button with the magnifying glass; the one on the right. This will display the information about these visibilities to your xterm.

```
PlotMS::locate+ Scan=0 Field=BEAM_0(0)
Time=2011/04/02/09:51:21.8 BL=RS106HBA-RS205HBA(20-21)
Spw=0 Chan=0 Freq=0.133411 Corr=XX X=2361.09 Y=19.1714
```

This is only an example of a single visibility; this action will print the information for all of the visibilities in the box. In this case, we see that the problematic baseline is RS106HBA-RS205HBA or antenna 20&21. Lets select only that baseline by inserting "antenna=20&21" in the data tab (see Fig. 43; bottom left). To flag this baseline, select all of the visibilities using the box with the green mark action button and then flag using the flag action button. Select "antenna= " in the data tab and replot to see the data set with the flags applied. This manual method is most useful for making small flags to a small data set. Alternatively, we may have 244 sub-bands of data that we want to apply flags to which would take quite a long time to do manually. These types of flags can also be done using NDPPP. So, lets flag this problematic baseline with NDPPP. First copy the parset file to your working directory:

```
> cp /home/mckean/TUTORIALS/CYGA/NDPPP.flag.parset .
> more NDPPP.flag.parset
```

[74] http://casa.nrao.edu/Doc/Cookbook/casa_cookbook.pdf

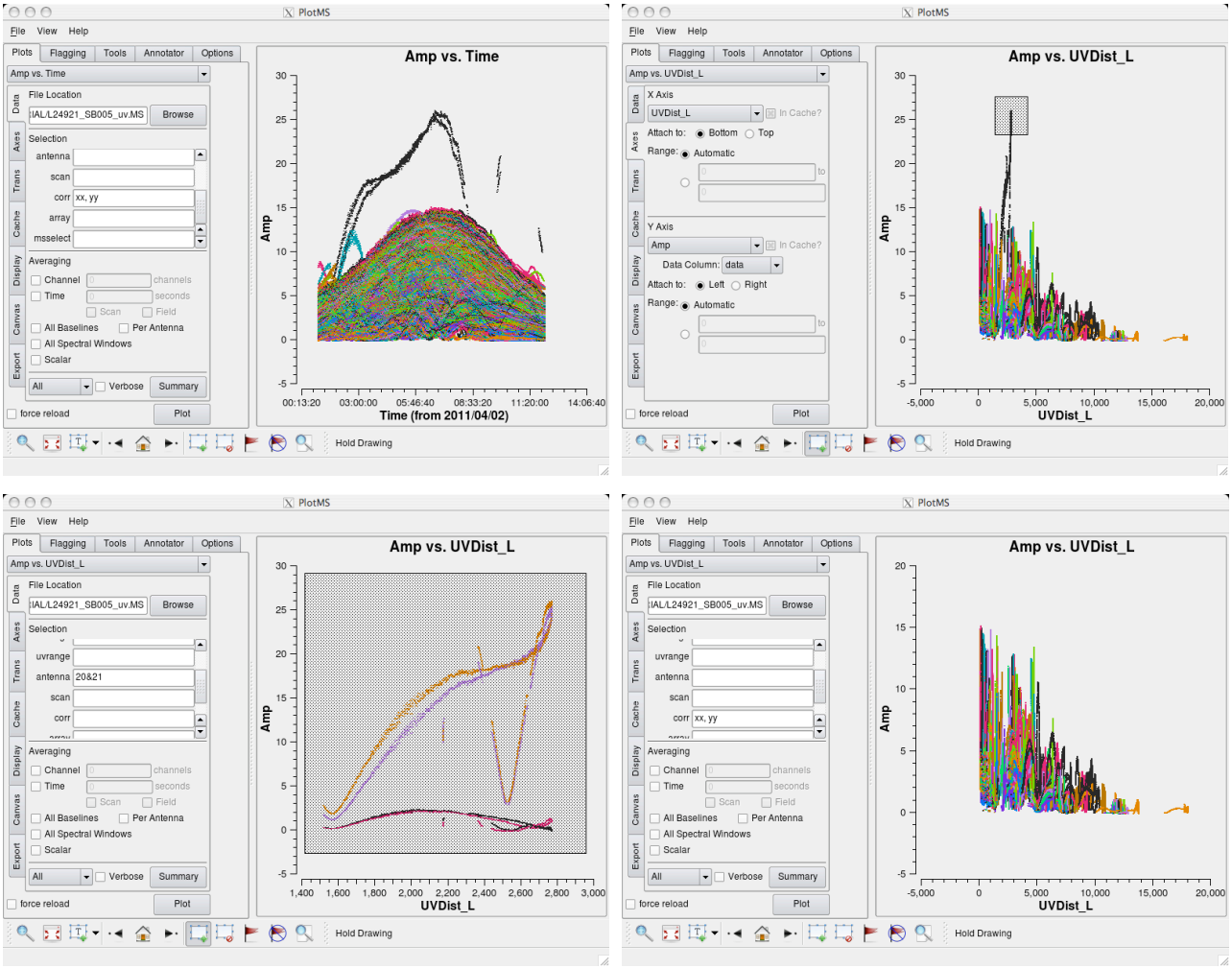


Figure 43: Top left: Plotting the visibility amplitude against time. Top right: Plotting the visibility amplitude against UV distance and selecting the baseline with an unusually large amplitude. Bottom left: Selecting only the visibilities for the baseline 20&21 and flagging. Bottom right: The data set without the flagged baseline 20&21.

```
msin = L24921_SB005_uv.MS.copy        # A copy of our compressed data set
msin.startchan = 0
msin.nchan = 1
msin.datacolumn = DATA

msout = "L24921_SB005_uv_flag.MS"
msout.datacolumn = DATA

steps = [flag1,count]

flag1.type=preflagger
flag1.baseline=RS106HBA&RS205HBA

> NDPPP NDPPP.flag.parset > ndppp.flag.txt
> more ndppp.flag.txt
```

From the output you should see that the baseline RS106HBA&RS205HBA is 100% flagged. You can also verify this by plotting the data set within plotms.

13.1.4 Calibration within BBS

Black Board Self-calibration is a highly flexible calibration software being 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 simple process of a single direction solution without the beam correction. This is possible because Cygnus A dominates the uv data and is at the pointing centre for the entire observation.

BBS has two modes: a stand-alone mode for solving only one subband, and a 'global solve' mode for finding parameters over multiple subbands, which may be stored on different computers. We will give an example of the usage of both modes.

Stand-alone mode: `calibrate-stand-alone`

Before we can run the calibration process, we need a parset file to direct the calibration and an initial sky model for correcting the data. The sky model is based on an image of Cygnus A taken with the VLA with $\sim$3 arcsec resolution at 327 MHz. As there is frequency dependent structure in the source, further iterations of self-calibration will be needed to improve on this initial sky model. The sky model should be copied to your working area using

```
> cp /home/mckean/TUTORIALS/CYGA/cyga.start.mod .
> more cyga.start.mod
```

This model consists of 935 clean components.

The parset file can be found at

```
> cp /home/mckean/TUTORIALS/CYGA/bbs.parset .
> more bbs.parset
```

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

Step.solve.Operation = SOLVE
Step.solve.Model.Sources = []
Step.solve.Model.Gain.Enable = T
Step.solve.Model.Cache.Enable = T
Step.solve.Solve.Parms = ["Gain:0:0:*","Gain:1:1:*"]
Step.solve.Solve.CellSize.Freq = 0
Step.solve.Solve.CellSize.Time = 1
Step.solve.Solve.CellChunkSize = 10
Step.solve.Solve.Options.MaxIter = 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
Step.correct.Model.Sources = []
Step.correct.Model.Gain.Enable = 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 (on one line),

```
> calibrate-stand-alone -f L24921_SB005_uv_flag.MS bbs.parset
    cyga.start.mod > test.txt &
```

Using the "> test.txt &" command allows you to save and inspect the output of BBS. In particular, should be able to see the number of data chunks that have been processed.

Multiple nodes mode: `calibrate`

The multiple-node mode of BBS needs a number of files that are used for parallel processing of multiple sub-bands. Even though we have only one sub-band in this example, we still show how to do this (you can skip this section if you used `calibrate-stand-alone` above.

```
> makevds /home/mckean/clusterdesc/cep1.clusterdesc
                  /data/scratch/mckean/TUTORIALS/L24921_SB005_uv_flag.MS
```

Note that this command is all one line. It is necessay to write out the whole path of where the data set is.

This will create a new file `L24921_SB005_uv_flag.MS.vds`

```
> more L24921_SB005_uv_flag.MS.vds
```

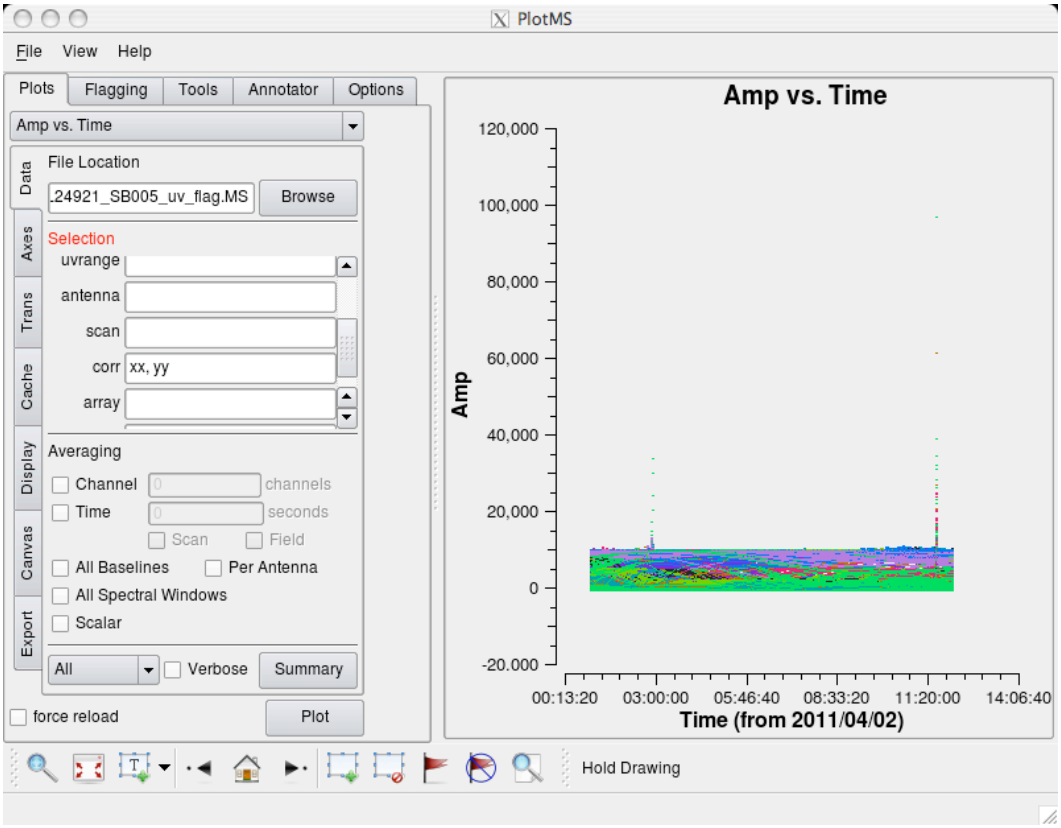
```
Name       = /data/scratch/mckean/TUTORIAL/L24921_SB005_uv_flag.MS
FileName   = /data/scratch/mckean/TUTORIAL/L24921_SB005_uv_flag.MS
FileSys    = lce006:/data
ClusterDesc= /home/mckean/clusterdesc/cep1.clusterdesc
StartTime  = 2011/04/02/01:00:00.000
EndTime    = 2011/04/02/12:00:09.985
StepTime   = 15.0209
NChan      = [1]
StartFreqs = [133325195.312]
EndFreqs   = [133496093.75]
Extra.CorrNames=[XX,XY,YX,YY]
Extra.DataCubeShape=[4,1,996786]
Extra.DataFileIsRegular=1
Extra.DataFileName=/data/scratch/mckean/TUTORIAL/L24921_SB005_uv_flag.MS/
                      table.f3_TSM0
Extra.DataTileShape=[4,1,32768]
Extra.FieldDirectionDec=[+040.44.02.000000]
Extra.FieldDirectionRa=[19:59:28.290000]
Extra.FieldDirectionType=J2000
Extra.StationNames=[CS001HBA0,CS002HBA0,CS003HBA0,CS004HBA0,CS005HBA0,
CS006HBA0,CS007HBA0,CS017HBA0,CS021HBA0,CS024HBA0,CS026HBA0,CS030HBA0,
CS032HBA0,CS101HBA0,CS103HBA0,CS201HBA0,CS301HBA0,CS302HBA0,CS401HBA0,
CS501HBA0,RS106HBA,RS205HBA,RS208HBA,RS306HBA,RS307HBA,RS406HBA,RS503HBA]
```

This file simply summarizes where the data are and what the important parameters are.

```
> combinevds bbs.gds L24921_SB005_uv_flag.MS.vds
> more bbs.gds
```

```
Name       = bbs.gds
ClusterDesc= /home/mckean/clusterdesc/cep1.clusterdesc
StartTime  = 2011/04/02/01:00:00.000
EndTime    = 2011/04/02/12:00:09.985
StepTime   = 15.0209
```

```
NChan      = [1]
StartFreqs = [133325195.312]
```



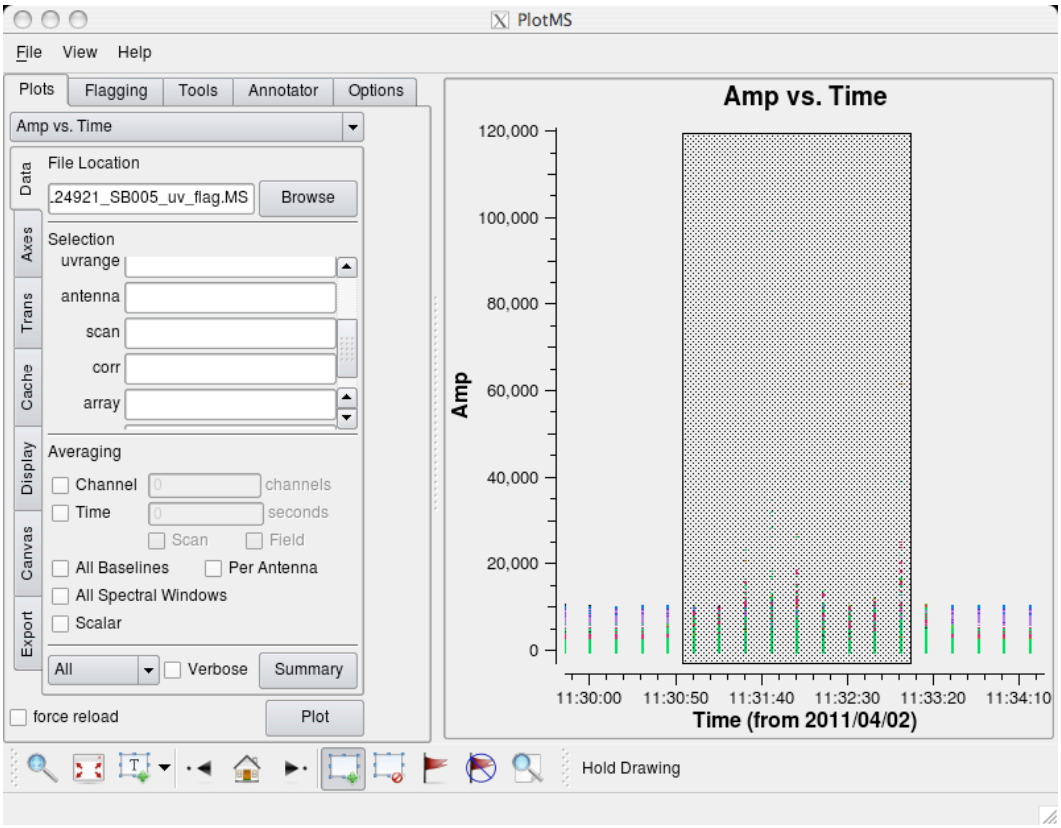


Figure 44: Top: Plotting the visibility amplitude against time we see some time stamps have high amplitudes. Bottom: A zoom in of the bad time stamps, which are then flagged within plotms.

```
EndFreqs    = [133496093.75]
Extra.FieldDirectionDec=[+040.44.02.000000]
Extra.FieldDirectionRa=[19:59:28.290000]
Extra.FieldDirectionType=J2000
NParts = 1
Part0.Name        = /data/scratch/mckean/TUTORIAL/L24921_SB005_uv_flag.MS.vds
Part0.FileName    = /data/scratch/mckean/TUTORIAL/L24921_SB005_uv_flag.MS
Part0.FileSys     = lce006:/data
Part0.ClusterDesc= /home/mckean/clusterdesc/cep1.clusterdesc
Part0.StartTime   = 2011/04/02/01:00:00.000
Part0.EndTime     = 2011/04/02/12:00:09.985
Part0.StepTime    = 15.0209
Part0.NChan       = [1]
Part0.StartFreqs  = [133325195.312]
Part0.EndFreqs    = [133496093.75]
```

This command combines all of the vds files into one. Note that the locations of the data will be different; they will point to your working directory.

```
> calibrate -f --key test --cluster-desc
/home/mckean/clusterdesc/cep1.clusterdesc  --db ldb001
--db-user postgres bbs.gds bbs.parset cyga.start.mod
/data/scratch/mckean/TUTORIAL/ > test.txt &
```

The calibration process should be completed in about 15 mins. Once it is complete it is useful to look at the calibrated data,

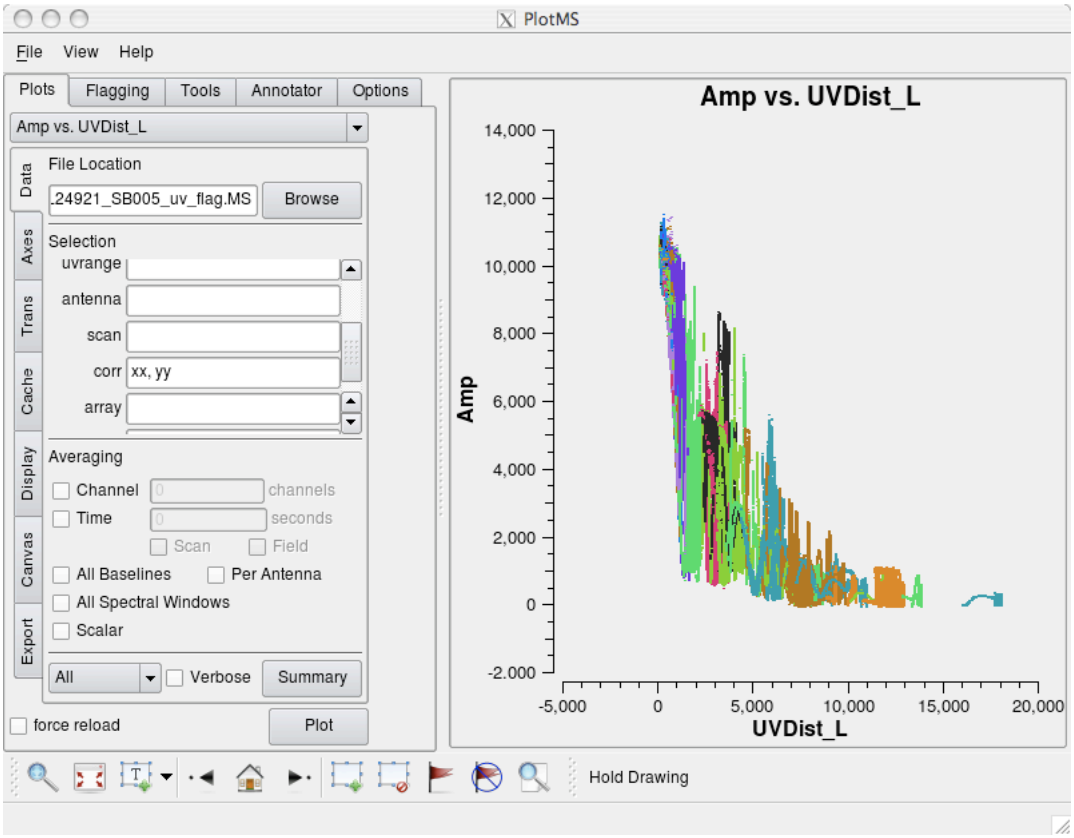


Figure 45: The visibility amplitudes as a function of uv distance for the data after correction in BBS.

```
> casaplotms
```

Go to the axes tab and plot the amplitude against time for the corrected data. You will see that there are a few time stamps where the solutions are poor, leading to visibilities with large amplitudes (see Figure 44). Use the magnifying glass (the one on the left) to draw a box around the area with poor data. As it is only a couple of minutes of data that are corrupted, lets just flag all the affected time stamps. This is done by selecting the data and flagging using the flag action button. It is also useful to look at the visibility amplitude as a function of uv distance to confirm that the data look good i.e. no amplitude spikes (see Figure 45).

It is also useful to inspect the amplitude and phase solutions for each station. This is done using

```
> parmdbplot.py L24921_SB005_uv_flag.MS/instrument/
```

A small window will appear (see Figure 46) that lists all of the stations, for both Gain:0:0 and 1:1, i.e. for the two polarizations that we just solved for in BBS. It is useful to de-select the "use resolution" option or set the time to 1s, since this will plot all of the solutions that we solved for. First select station Gain:0:0:CS003 and hit plot. A new window showing the amplitude (top) and phase (bottom) solutions will appear (see Figure 46). First you will notice that the amplitude rises with time, peaks and then falls again. This is an important difference between LOFAR and other radio interferometers, where the amplitudes tend to be constant, or only slightly varying over time. The change in the amplitude arises from the projected area of the station changing as the source rises and then sets during the observation. You should also be able to see the phases change in a sensible way i.e., you can trace the change in phase over time. This is harder to predict since it depends on the state of the ionosphere during the observation, but you should see similar phases for the core stations, particularly those in the Superterp.

Try plotting other core stations to see similar amplitude and phase corrections. Note that you can also plot multiple stations together by selecting those that you are interested in before plotting.

Next, let's plot remote station RS406 (see Figure 46). We see that for a limited time the solutions peak sharply. This suggests that there are possibly bad data at those times. Due to the large amplitudes, it is difficult to see the quality of the solutions in general. Lets zoom in to the solutions with lower amplitudes using the zoom slider in the left part of the window. We see that, in general, the amplitude solutions look similar to the case of the core station, i.e. rising to a peak and then falling, but the solutions are much noisier and not as smooth as the core station example. This is because our model does not fit the data sufficiently well. Also note how fast the phases are varying. They are changing much faster than for the core station, but we can still trace these changes with the 15 s averaging time for the visibilities.

We will carry out self-calibration to improve the model for the source. We should expect to see the amplitude solutions tend towards something similar to what the core stations have, i.e. smoothly varying.

13.1.5 Cleaning and making a new model

We must de-convolve the data to make science quality images, but also so that we can construct a better model for the data for self-calibration. There are a number of imagers available that can be used, but in this example, we will use the imager that is part of CASA. Since for Cygnus A we plan to image only the centre of the observed field, we can use a simple imager. If you want to image the full field of view, then imaging that incorporates wide-field imaging techniques (e.g. widefield in CASA) must be used.

```
> casapy
> help clean
```

The task clean will be used and the help file that is part of CASA describes in detail the parameters that can be set. For this example, some basic parameters for the image size (imsize), the cell size (cell) and the weighting are used, but feel free to experiment.

```
> inp clean
```

```
CASA <1>: inp clean
---------> inp(clean)
#  clean :: Invert and deconvolve images with selected algorithm
vis                 = 'L24921_SB005_uv_flag.MS'
imagename           =    'cyga1'
outlierfile         =          ''
field               =          ''
spw                 =          ''
selectdata          =       False
mode                =       'mfs'
     nterms         =           1
     reffreq        =          ''
gridmode            =          ''
niter               =        3000
gain                =        0.01
```

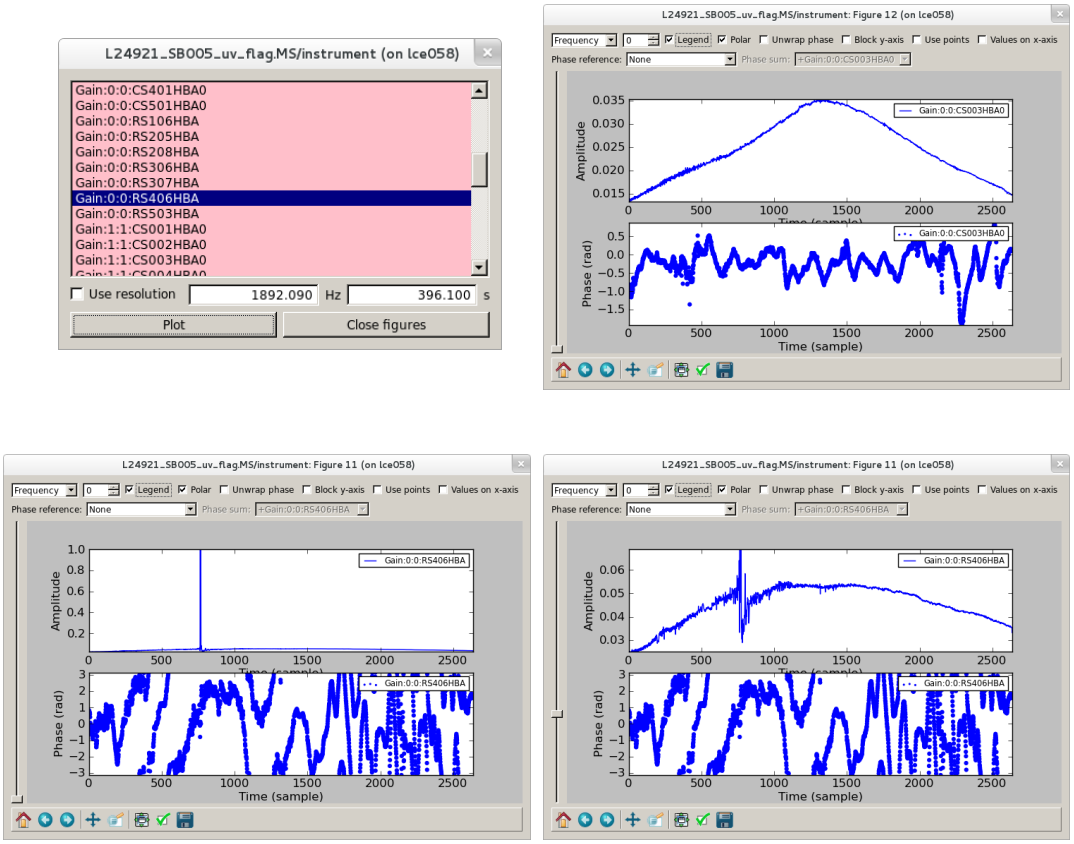


Figure 46: Top left: The parmdb window. Use this to select the stations whose solutions you want to inspect, and to change the resolution that is used to display the solutions. Top right: The solutions for CS003. Bottom left: The solutions for RS208. Bottom right: A zoom in for the solutions of RS208.

```
threshold           =    '0.0mJy'
psfmode             =     'clark'
imagermode          =          ''
multiscale          =          []
interactive         =       False
mask                = 'cyga.mask'
imsize              = [2048, 2048]
cell                = ['1.0arcsec', '1.0arcsec']
phasecenter         =          ''
restfreq            =          ''
stokes              =         'I'
weighting           =    'briggs'
     robust         =       -0.75
     npixels        =           0
uvtaper             =       False
modelimage          =          ''
restoringbeam       =        ['']
pbcor               =       False
minpb               =         0.2
calready            =        True
async               =       False
```

With this set-up clean will operate without any interaction from the user "interactive = False". As Cygnus A is such a bright source and there are likely to be negative side-lobes in the image due to the limited dynamic range, it is important to use clean boxes to limit the area that clean will use for the de-convolution. A specially prepared mask file should be copied to your working directory,

```
> cp -rf /home/mckean/TUTORIALS/CYGA/cyga.mask .
```

If you wish to create your own mask file, this is best done using the interactive cleaning. To start the de-convolution use,

```
CASA <2>: go clean
```

The current set-up has only 3000 iterations that will clean 0.01 of the peak from the image. Therefore clean will have to be run several times to complete the process. In total, the clean component flux in the image should be around 10500 Jy. This information is given in the CASA logger window. You can look at the images with the viewer,

```
CASA <3>: go viewer
```

and selecting the files from the pop-up window. In Figure 47 are the residual and cleaned images from our first calibration loop.

The output from clean is 5 images

```
cyga1.image     # the de-convolved image
cyga1.residual  # the residual image
cyga1.model     # the model image
cyga1.flux      # the total flux image
cyga1.psf       # the psf image
```

The model image will contain the several thousand clean components that make up our new model for Cygnus A. This model file can be convert to a new bbs sky file using (outside of CASA),

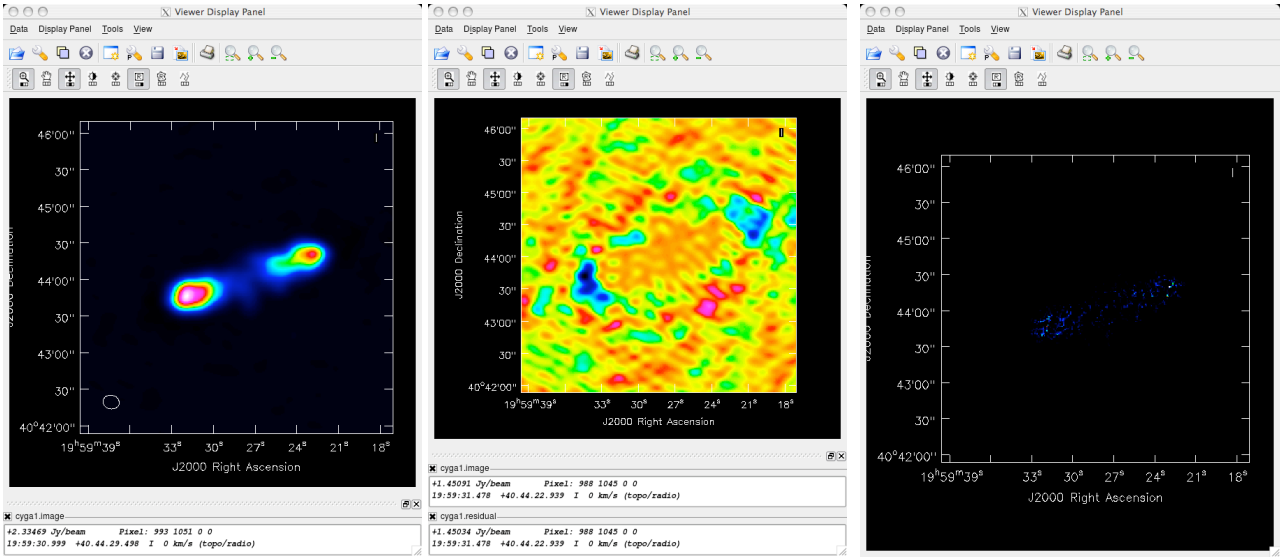


Figure 47: The clean image (left), the residual (middle) and the model (right) image for Cygnus A.

```
> casapy2bbs.py cyga1.model  cyga1.model.bbs
info: total number of CLEAN components: 1024
info: total flux in CLEAN components: 10501.65 Jy
info: clipping at: 10501.65 Jy
info: number of CLEAN components selected: 1024 (100.00%)
info: flux in selected CLEAN components: 10501.65 Jy (100.00%)
> more cyga1.model.bbs
```

You wil see the new bbs sky file containing the positions and flux densities of the clean components in bbs format.

13.1.6 Manual self-calibration

Using the skills that you have now developed, you should be able to self-calibrate the data with this new model to make a better model for the source.

```
> calibrate-stand-alone -f L24921_SB005_uv_flag.MS bbs.parset cyga1.model.bbs > test.txt
```

13.2 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 lce019
> 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
```

13.2.1 HBA

The raw data set for this exercise can be found in

```
/data/scratch/williams/tutorial/3c295/L74759/
```

on compute node lce019. 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

```
L74759_SAP000_SB000_uv.MS
L74759_SAP000_SB001_uv.MS
```

for the two sub-bands.

To copy it to your current working directory use:

```
scp -r lce019:/data/scratch/williams/tutorial/3c295/L74759/L74759*MS .
```

13.2.1.1 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:  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 = 3597.99 seconds
   Observed from   12-Nov-2012/12:47:00.5   to   12-Nov-2012/13:46:58.5 (UTC)

Fields: 1
  ID   Code Name                RA            Decl           RefType
  0         BEAM_0              14:11:20.5000 +52.12.10.0000 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  118.849182  3.05175781  195.3125    118.945312  XX  XY
YX  YY

Antennas: 54:
  ID   Name  Station   Diam.    Long.         Lat.
  0    CS001HBA0LOFAR     31.3 m   +006.52.07.1  +52.43.34.7
  1    CS001HBA1LOFAR     31.3 m   +006.52.02.2  +52.43.31.8
  2    CS002HBA0LOFAR     31.3 m   +006.52.07.6  +52.43.46.8
  3    CS002HBA1LOFAR     31.3 m   +006.52.08.0  +52.43.48.2
  ...
  ...
  ...
  42   CS501HBA0LOFAR     31.3 m   +006.51.57.9  +52.44.29.9
  43   CS501HBA1LOFAR     31.3 m   +006.51.59.7  +52.44.25.8
  44   RS106HBALOFAR     31.3 m   +006.59.05.6  +52.41.21.6
  ...
  ...
  ...
  53   RS509HBALOFAR     31.3 m   +006.47.04.7  +53.13.30.1

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, there are 64 spectral channels and the frequency is 118.849 MHz for SB000 and 119.044 MHz for SB001.

13.2.1.2 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, the Radio Observatory will compress the data to 3 and 10 s in time for the LBA and HBA data, respectively, and to 16 spectral channels. 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.

The parset file for the flagging[75] and compression should be copied to your working directory,

```
> cp /home/williams/tutorial/3c295/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.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

averager.freqstep=64     # compresses to 1 channel
averager.timestep=10     # compresses 10 time-slots, i.e. 10s
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 > ndppp1.txt
```

Depending on the use of the cluster, it will take about $\sim 5-10$ minutes to compress and flag the data. The progress bar reports the progress 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.
```

[75] Note that in this example we are flagging the data using the aoflagger algorithm.

You can inspect the output log file by using

```
> less ndppp1.txt
```

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
```

Some of the tasks that are used will make changes to the MS file, so lets make a copy of the compressed data set for safety,

```
> cp -rf L74759_SAP000_SB000_uv.MS.avg.dppp
   L74759_SAP000_SB000_uv.MS.avg.dppp.copy
```

13.2.1.3 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[76] for full details,

```
> use Casa
> casaplotms
```

Figure 48 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.

13.2.1.4 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[77] to calibrate single sub-bands. The stand-alone version can be run using the following command

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

[76] http://casa.nrao.edu/Doc/Cookbook/casa_cookbook.pdf

[77] BBS is described in Chapter 6

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 /home/williams/tutorial/3c295/3C295TWO.model .
> cat 3C295TWO.model
```

```
# (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]
```

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. 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.

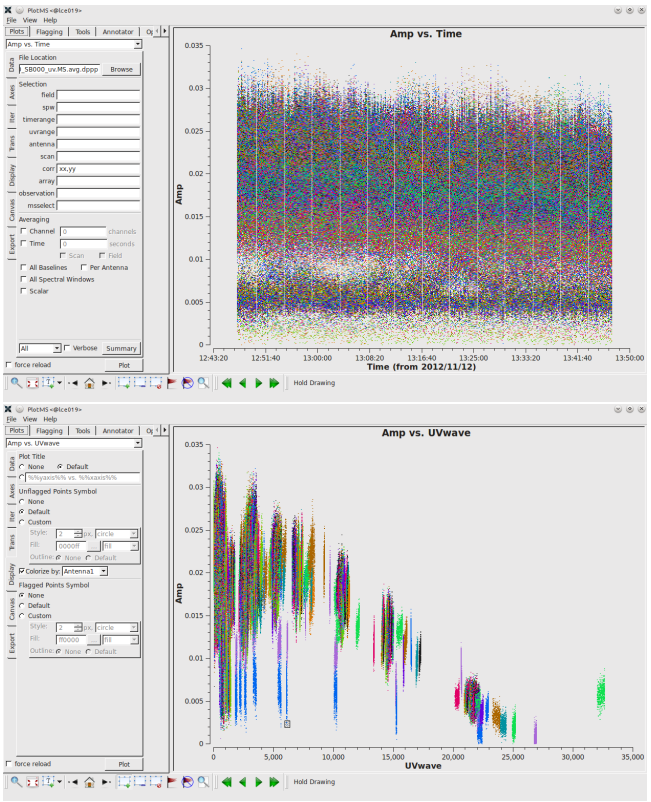


Figure 48: SB000. Top: Plotting the visibility amplitude against time. Bottom: Plotting the visibility amplitude against UV distance in wavelengths. (The colour scheme 'Antenna1' is used here.)

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 /home/williams/tutorial/3c295/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.Cache.Enable = T
Step.solve.Solve.Parms = ["Gain:0:0:*","Gain:1:1:*"]
Step.solve.Solve.CellSize.Freq = 0
Step.solve.Solve.CellSize.Time = 1
Step.solve.Solve.CellChunkSize = 10
Step.solve.Solve.Options.MaxIter = 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
Step.correct.Model.Sources = [3c295]
Step.correct.Model.Gain.Enable = T
Step.correct.Model.Beam.Enable = 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.model > bbs_sb000.txt &
```

Note that using the redirect "> bbs_sb000.txt" 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. Note that currently there is 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. 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 keep the 'use resolution' option unchecked as this will plot all of the solutions that we solved for. With the 'use resolution' option deselected select a few stations and look at the solutions. Figure 49 shows some solution plots for SB000.

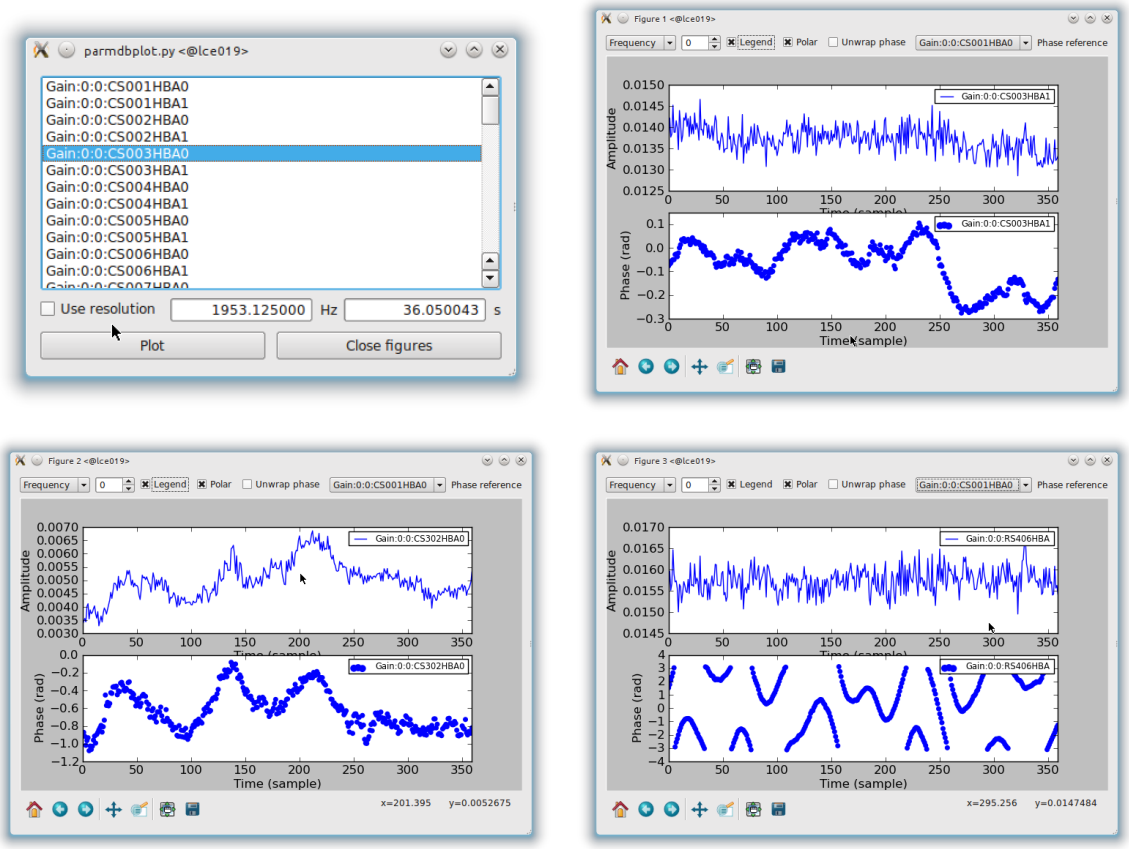


Figure 49: Top left: The parmdb window. Use this to select the stations whose solutions you want to inspect, and to change the resolution that is used to display the solutions. Top right: The solutions for CS003HBA1. Bottom left: The solutions for CS302HBA0. Bottom right: The solutions for RS406HBA.

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 and

YY correlations. Figures 50 and 51 show these plots for SB000 and SB001 respectively. For SB000, it is clear that the solutions for CS302HBA0 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.

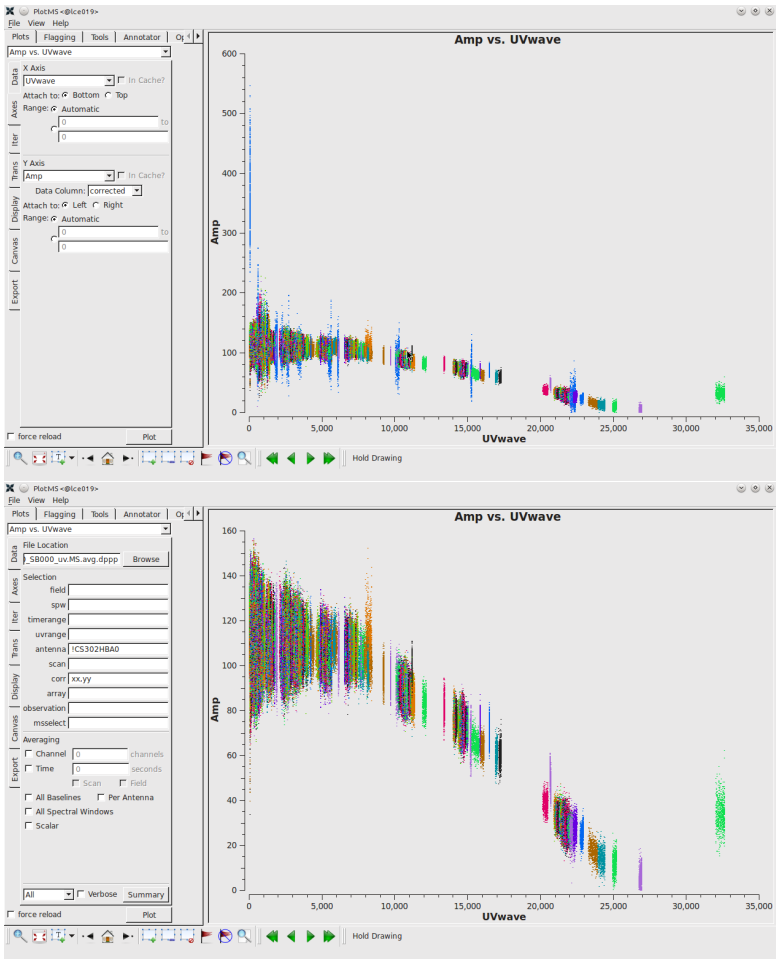


Figure 50: SB000. Top: Plotting the visibility amplitude against UV distance. Bottom: Excluding antenna CS302HBA0.

We will flag all of these bad baselines now with NDPPP. Since NDPPP can only write to the DATA column we will write a new dataset and re-do the calibration. The NDPPP parsets are:

```
> cp /home/williams/tutorial/3c295/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
msout = L74759_SAP000_SB000_uv.MS.avg.dppp.flag
msout.datacolumn = DATA
steps = [flag]
flag.type=preflagger
flag.baseline=RS508HBA&RS509HBA; RS208HBA&RS509HBA; CS302HBA0

> NDPPP NDPPP.flag.sb000.parset > ndppp.flag0.txt &
```

and likewise for the other subband:

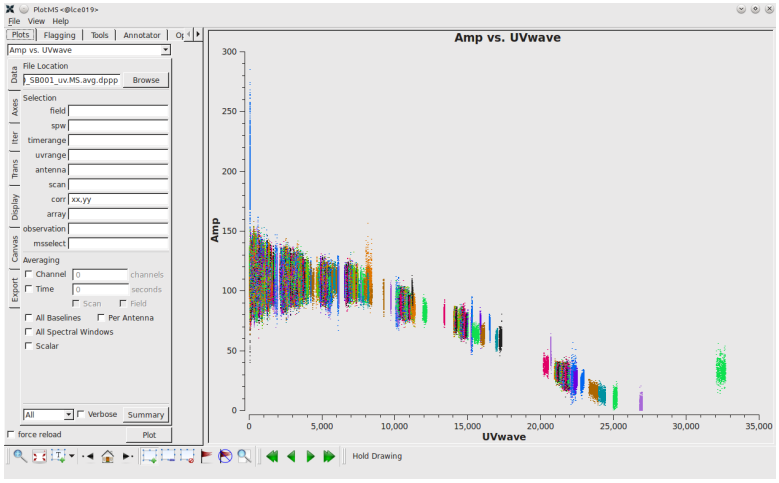


Figure 51: SB001. Plotting the visibility amplitude against UV distance.

```
> cp /home/williams/tutorial/3c295/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 = DATA
msout = L74759_SAP000_SB001_uv.MS.avg.dppp.flag
msout.datacolumn = DATA
steps = [flag]
flag.type=preflagger
flag.baseline=RS508HBA&RS509HBA; RS208HBA&RS509HBA; CS302HBA0&CS302HBA1

> NDPPP NDPPP.flag.sb001.parset > ndppp.flag1.txt &
```

We can re-do the calibration with:

```
> calibrate-stand-alone -f L74759_SAP000_SB000_uv.MS.avg.dppp.flag bbs.parset \
  3C295TWO.model > bbs_sb000.txt &
```

The amplitude against time for the flagged corrected data is plotted in 52.

13.2.1.5 Imaging

Here we will use the AWimager[78] 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
```

At the time of this writing, in order to use the fast new version of the AWimager, you need to source it from Cyril Tasse's home[79]

[78] see Chapter 9

[79] In future this will be part of the daily build

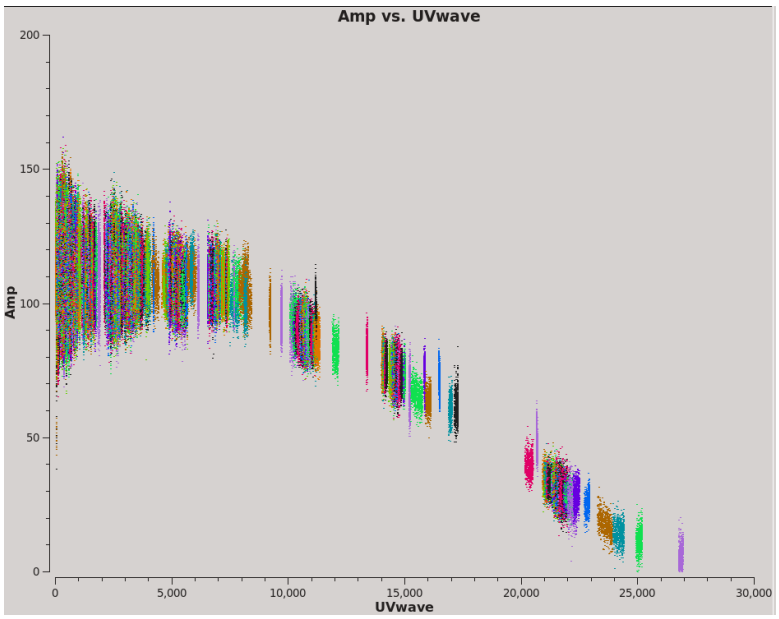


Figure 52: Plotting the visibility amplitude against UV distance for SB000 after flagging.

```
> source ~tasse/jaws22/init.sh
```

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 /home/williams/tutorial/3c295/awimger.sb000.parms .
> cat awimger.sb000.parms
```

```
ms=L74759_SAP000_SB000_uv.MS.avg.dppp.flag
image=L74759_SAP000_SB000.dirty.img
data=CORRECTED_DATA
gain=0.1
cellsize=3arcsec
npix=4096
stokes=I
weight=briggs
robust=0
operation=mfclark
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
```

this will take 2 – 3 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[80]. The last few lines of the output will look like (timestamps have been removed for clarity)

```
---------------------------> finalizeToVis
[4096, 4096] [4096, 4096]
FCleanImageSkyModel::solve     Final maximum residual = 92.3824
MFCleanImageSkyModel::solve     Model 0: max, min residuals = 92.3824, -14.071
clean flux 0
imager::clean() Threshhold not reached yet.
imager::clean() Fitted beam used in restoration: 25.434 by 13.9604 (arcsec) at
pa 51.1185 (deg)
Final normalisation
clean      133.3 real      376.42 user        7.89 system
awimager normally ended
```

AWImager produces a lot of images as output, including

```
L74759_SAP000_SB000.dirty.img.model          # uncorrected dirty image
L74759_SAP000_SB000.dirty.img.residual       # residual image
L74759_SAP000_SB000.dirty.img.psf            # point spread function
L74759_SAP000_SB000.dirty.img.residual.corr  # corrected residual image
```

The image we wish to look at now is the “residual.corr” image. It may seem strange to look at the residual but this is because no iterations of cleaning has been done so there will be nothing in the restored image. This can be done with casaviewer:

```
> casaviewer L74759_SAP000_SB000.dirty.img.residual.corr
```

the dirty image should look like a single strong point source convolved with the psf (See Fig. 53).

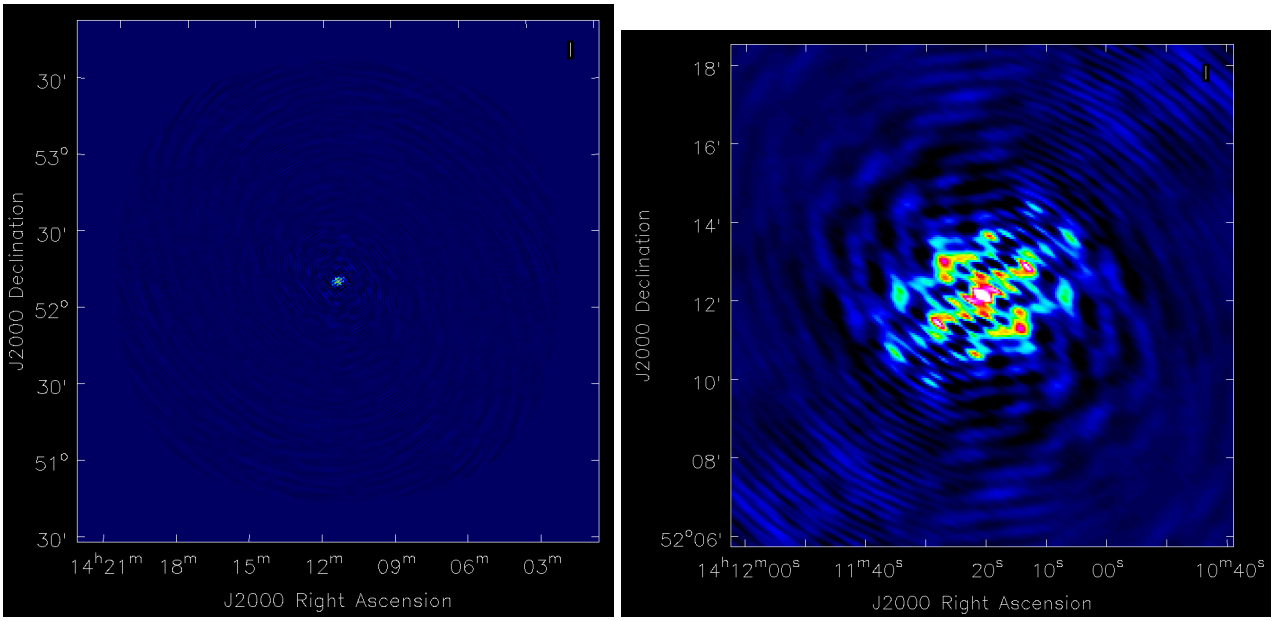


Figure 53: Left: The dirty image for 3C 295 (SB000). Right: Zoom in 4x. The data range is set to [-0.1, 1].

[80] setting “operation=image” does not work to make a dirty image with this version of the imager.

Now we will do some cleaning. If you look at George Heald's beta noise calculator for LOFAR[81], 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.

```
> cp /home/williams/tutorial/3c295/awimger.sb000.clean.parms .
> cat awimger.sb000.clean.parms
```

```
ms=L74759_SAP000_SB000_uv.MS.avg.dppp.flag
image=L74759_SAP000_SB000.clean.img
data=CORRECTED_DATA
padding=1.
cyclefactor=1.5
gain=0.1
cellsize=3arcsec
npix=4096
stokes=I
weight=briggs
robust=0
operation=mfclark
niter=10000
threshold=0.1Jy
```

```
> rm -rf L74759_SAP000_SB000.clean.img*
> awimager awimger.sb000.clean.parms
```

AWimager will run for $\sim$ 20 minutes and will produce a lot of output to the screen for each major cycle.

```
---------------------------> finalizeToVis
[4096, 4096] [4096, 4096]
imager::clean() Successfully deconvolved image
imager::clean() Fitted beam used in restoration: 25.434 by 13.9604 (arcsec) at
pa 51.1184 (deg)
Final normalisation
... restored image too ...
clean        1761 real       8755.3 user         87.3 system
awimager normally ended
54 real        14373 user 2937.9000000000000909 system
awimager normally ended
\end{lstlisting


The output from AWimager consists of several images:
\begin{verbatim}
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
\end{verbatim}
Figure \ref{fig:tutorial_hba_images_clean} shows the cleaned corrected image (``restor
```

[81] http://www.astron.nl/~heald/test/sens.php

```
\begin{figure}[htp]
 \centering
\includegraphics[width=0.475\textwidth]{tutorial_hba_clean_sb000.png}
\includegraphics[width=0.505\textwidth]{tutorial_hba_clean_sb000_zoom1.png}
\caption{Left: The cleaned image for 3C\,295 (SB000). Right: A zoom-in of the source i
\label{fig:tutorial_hba_images_clean}
\end{figure}

% The model image will contain the several thousand clean components that make up our
%
%
% \begin{verbatim}
% > casapy2bbs.py cyga1.model cyga1.model.bbs
% info: total number of CLEAN components: 1024
% info: total flux in CLEAN components: 10501.65 Jy
% info: clipping at: 10501.65 Jy
% info: number of CLEAN components selected: 1024 (100.00%)
% info: flux in selected CLEAN components: 10501.65 Jy (100.00%)
% > more cyga1.model.bbs
% \end{verbatim}
%
% You wil see the new bbs sky file containing the positions and flux densities of the

\paragraph{Subtraction of 3C\,295}\mbox{}\\

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.

We make a copy of the measurement set before carrying out the
subtraction.
\begin{verbatim}
 cp -r L74759_SAP000_SB000_uv.MS.avg.dppp.flag
 L74759_SAP000_SB000_uv.MS.avg.dppp.flag.sub
\end{verbatim}


We require a parset that includes a subtract step for the source 3C\,295.
%\begin{verbatim}
%> calibrate-stand-alone -f L74759_SAP000_SB001_uv.MS.avg.dppp.flag
%bbs_subtraction_SB001.parset 3C295TWO.model > bbs subtraction_SB001.txt
%\end{verbatim}
%where the parset includes a subtract step for the source 3c295, which consists
%of two point sources grouped in one patch called ``3c295'':
\begin{verbatim}
> cp /home/williams/tutorial/3c295/bbs_subtract3c295.parset .
> cat bbs_subtract3c295.parset
\end{verbatim}
\begin{lstlisting}
Strategy.ChunkSize = 100
Strategy.InputColumn = DATA
Strategy.TimeRange = []
Strategy.Baselines = *&
Strategy.Steps = [solve,subtract,correct]

Step.subtract.Operation = SUBTRACT
Step.subtract.Model.Sources = [3c295]
Step.subtract.Model.Beam.Enable = T
Step.subtract.Model.Gain.Enable = T
Step.subtract.Model.Cache.Enable = T
```

```
Step.solve.Operation = SOLVE
Step.solve.Model.Sources = [3c295]
Step.solve.Model.Gain.Enable = T
Step.solve.Model.Beam.Enable = T
Step.solve.Model.Cache.Enable = T
Step.solve.Solve.Parms = ["Gain:0:0:*","Gain:1:1:*"]
Step.solve.Solve.CellSize.Freq = 0
Step.solve.Solve.CellSize.Time = 1
Step.solve.Solve.CellChunkSize = 10
Step.solve.Solve.Options.MaxIter = 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
Step.correct.Model.Sources = []
Step.correct.Model.Gain.Enable = T
Step.correct.Model.Beam.Enable = T
Step.correct.Output.Column = CORRECTED_DATA

> calibrate-stand-alone -f L74759_SAP000_SB000_uv.MS.avg.dppp.flag.sub
  bbs_subtract3c295.parset 3C295TWO.model > bbs_subtract_sb000.txt &
```

This should take $\sim$ 10 minutes to run.

The 3C 295-subtracted visibility amplitudes are plotted against time in Figure 54.

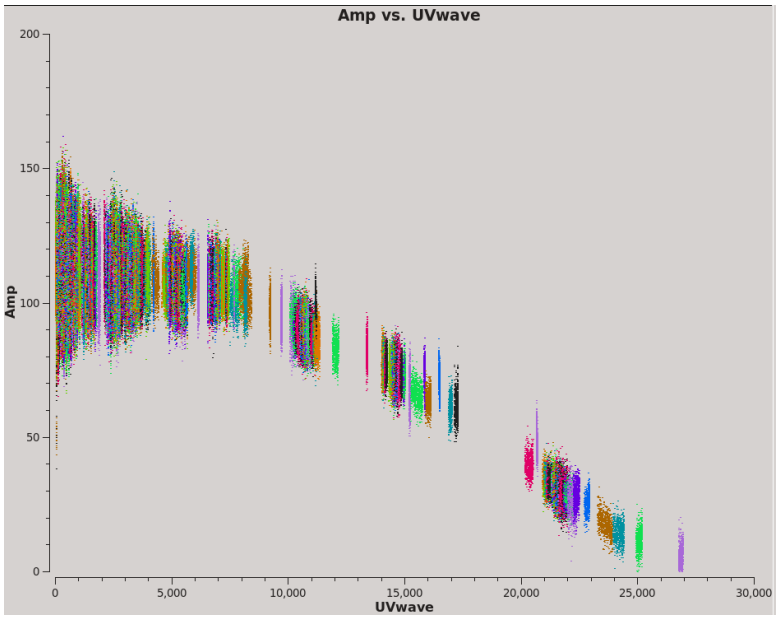


Figure 54: Plotting the visibility amplitude against UV distance after flagging.

We can make an image of the subtracted data

```
> cp /home/williams/tutorial/3c295/awimger.sb000.sub.clean.parms .
> cat awimger.sb000.sub.clean.parms
```

```
ms=L74759_SAP000_SB000_uv.MS.avg.dppp.flag.sub
image=L74759_SAP000_SB000.sub.clean.img
data=CORRECTED_DATA
padding=1.
cyclefactor=1.5
gain=0.1
cellsize=3arcsec
npix=4096
stokes=I
weight=briggs
robust=0
operation=mfclark
niter=20000
threshold=30mJy

> rm -rf L74759_SAP000_SB000.sub.clean.img*
> awimager awimger.sb000.sub.clean.parms
```

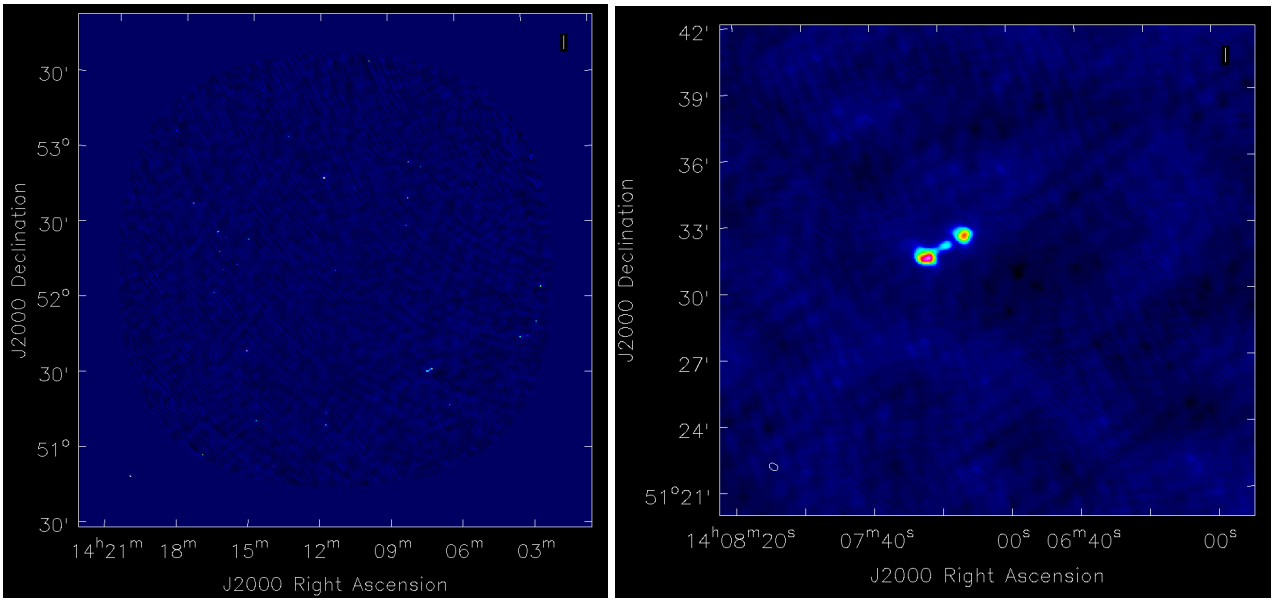


Figure 55: Left: The cleaned image for 3C 295 (SB000) after 3C 295 has been subtracted. Right: A zoom-in of the source in the lower right corner. The data range is set to [-0.1, 1].

13.2.1.6 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 /home/williams/tutorial/3c295/NDPPP.combineMS.parset .
> cat NDPPP.combineMS.parset
msin = L74759_SAP000_SB*_uv.MS.avg.dppp.flagi.sub
msin.datacolumn = CORRECTED_DATA
msout = L74759_SAP000_SBcomb_uv.MS.avg.dppp.flag.sub

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 CORRECTED DATA column from the input MSs will be written to the DATA column in the output. Here we only have two but the method should provide a simple way of combining multiple sub-bands. Note that this method will collapse all channels. To preserve individual spectral windows in your data then 'msconcat' within CASA can be used.

Using the skills that you have now developed, you should be able to image the concatenated data.

13.2.2 LBA

The raw LBA data set for this exercise can be found in,

`/data/scratch/williams/tutorial/3c295/L74762/`

on compute node lce018. 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.

13.2.2.1 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.

13.2.2.2 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 5 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,

```
python /opt/cep/tools/cookbook/plot_Ateam_elevation.py L74762_SAP000_SB000_uv.MS
```

the output of which is shown in Fig.56. 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[82] 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 /home/williams/tutorial/3c295/Ateam_LBA_CC.sky .

> cp /home/williams/tutorial/3c295/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
```

[82] using the aoflagger algorithm.

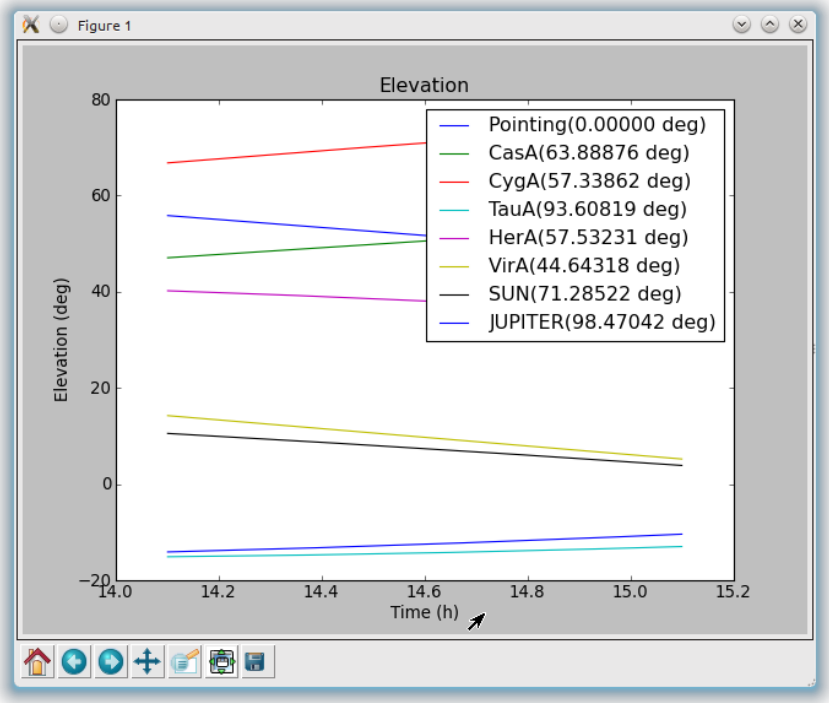


Figure 56: A-team elevation for the LBA observation.

```
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 > ndppp1.txt
```

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
```

13.2.2.3 Post-compression data inspection and flagging

We will use the CASA task plotms to inspect the data. Figure 57 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 leave these as is for now.

13.2.2.4 Calibration with BBS

Here we will use the stand-alone version of BBS[83] 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 /home/williams/tutorial/3c295/3C295TWO.model .
> cat 3C295TWO.model
```

```
# (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 /home/williams/tutorial/3c295/bbs.parset  .
> cat bbs.parset
```

[83] BBS is described in Chapter 6

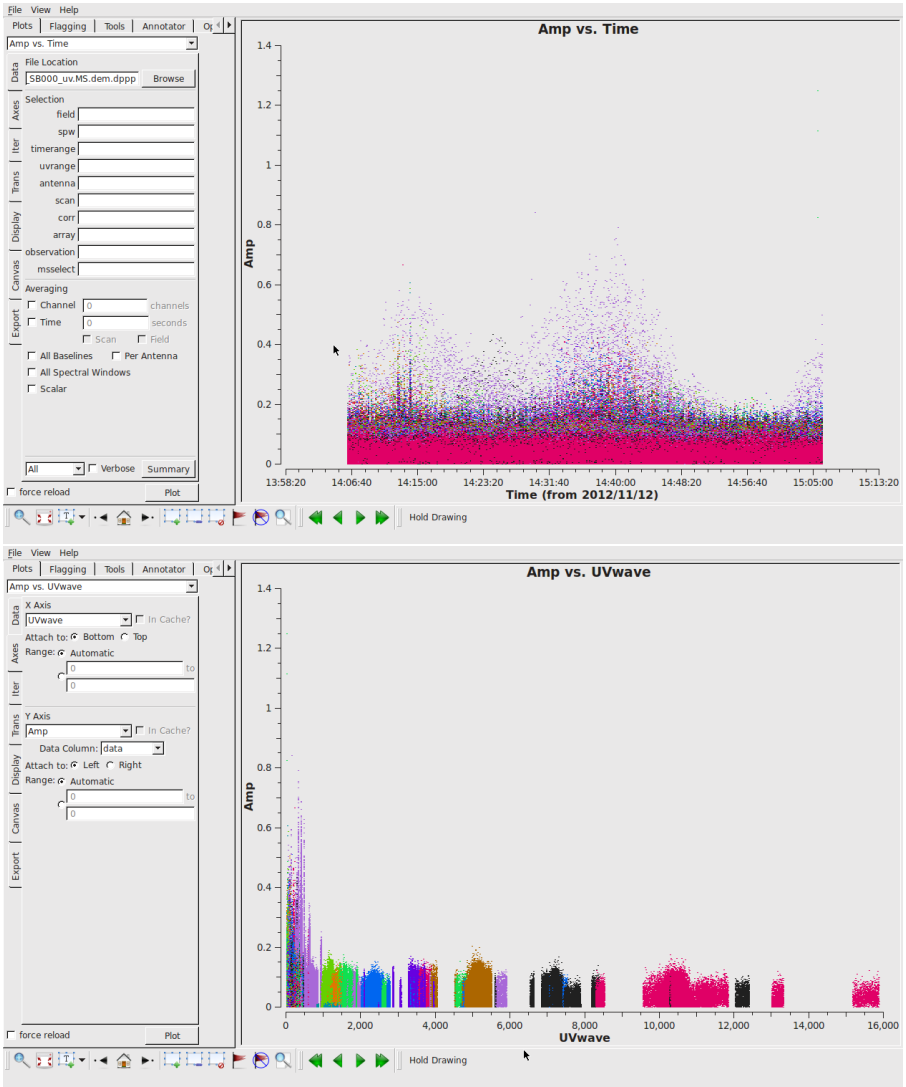


Figure 57: SB000. Top: Plotting the visibility amplitude against time. Bottom: Plotting the visibility amplitude against UV distance in wavelengths. Colourise by 'Antenna2' to obtain these colours.

```
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.Cache.Enable = T
Step.solve.Solve.Parms = ["Gain:0:0:*","Gain:1:1:*"]
Step.solve.Solve.CellSize.Freq = 0
Step.solve.Solve.CellSize.Time = 1
Step.solve.Solve.CellChunkSize = 10
Step.solve.Solve.Options.MaxIter = 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
Step.correct.Model.Sources = [3c295]
Step.correct.Model.Gain.Enable = T
Step.correct.Model.Beam.Enable = 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 L74762_SAP000_SB000_uv.MS.dem.dppp bbs.parset
3C295TWO.model
  > bbs_sb000.txt &
```

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. Figure 58 shows some solution plots for SB000. It is clear that there are a few spikes in the solutions.

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 solutions and there is a lot of scatter overall to high amplitudes. Here we will flag these time stamps manually in plotms and also clip all amplitudes above ~ 500. Figure 60 shows the Amp. vs UV distance plots for both sub-bands after flagging.

We can re-do the calibration with:

```
> calibrate-stand-alone -f L74762_SAP000_SB000_uv.MS.dem.dppp bbs.parset
   3C295TWO.model
> bbs_sb000.txt &
```

13.2.2.5 Imaging

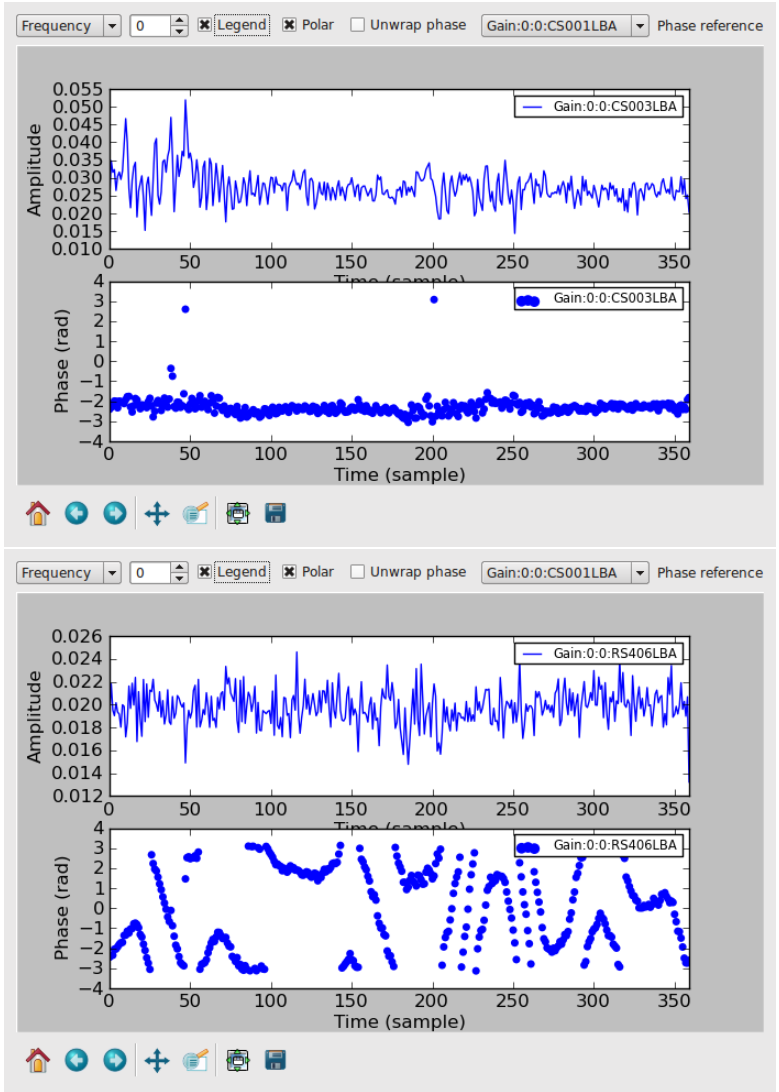


Figure 58: Top: The solutions for CS003LBA. Bottom: The solutions for RS406LBA.

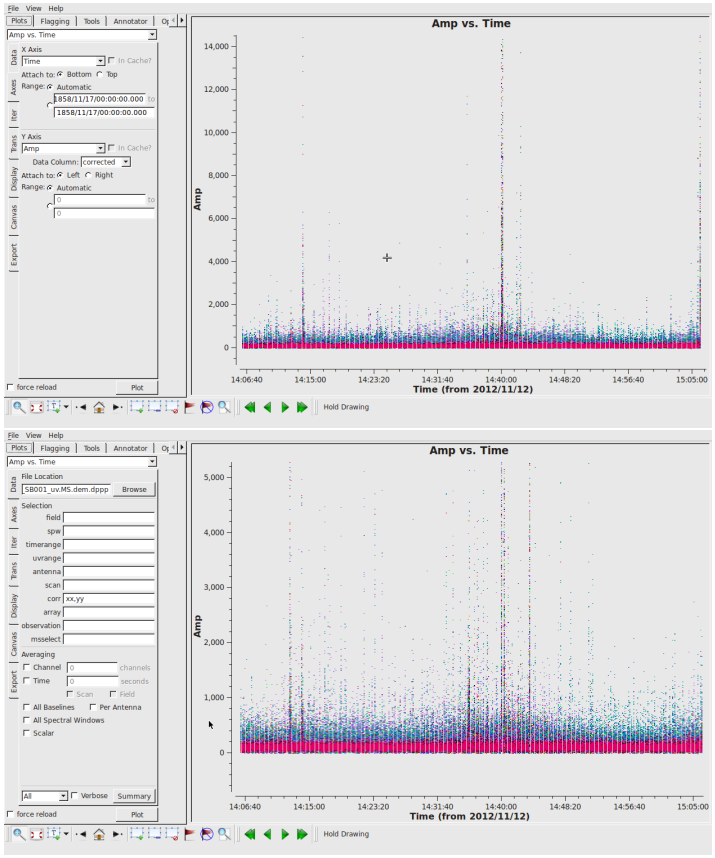


Figure 59: Top: Plotting the visibility amplitude against time for SB000. Bottom: SB001.

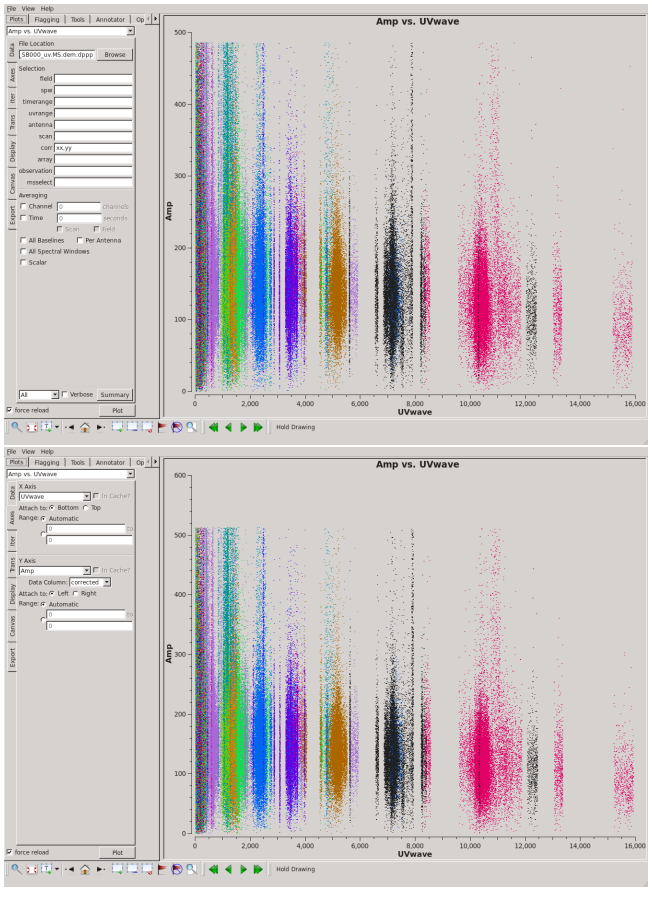


Figure 60: Top: Plotting the visibility amplitude against UV distance for SB000 after flagging. Bottom: SB001.

Here we will also use the AWimager[84] 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 /home/williams/tutorial/3c295/awimger.sb000.parms .
> cat awimger.sb000.parms
```

```
ms=L74762_SAP000_SB000_uv.MS.dem.dppp
image=L74762_SAP000_SB000.dirty.img
data=CORRECTED_DATA
padding=1.
oversample=5
cyclefactor=1.5
gain=0.1
cellsize=6arcsec
npix=2048
stokes=I
weight=briggs
robust=0
wprojplanes=256
wmax=15000
operation=image
niter=0
```

```
> awimager awimger.sb000.parms
```

```
> cp /home/williams/tutorial/3c295/awimger.sb000.clean.parms .
> cat awimger.sb000.clean.parms
```

```
ms=L74759_SAP000_SB000_uv.MS.avg.dppp.flag
image=L74759_SAP000_SB000.clean.img
data=CORRECTED_DATA
padding=1.
cyclefactor=1.5
gain=0.1
cellsize=3arcsec
npix=4096
stokes=I
weight=briggs
robust=0
operation=mfclark
niter=10000
threshold=0.1Jy
```

```
> awimager awimger.sb000.clean.parms
```

Figure 61 shows the dirty image and Fig. 62 shows the cleaned corrected image.

[84] see Chapter 9

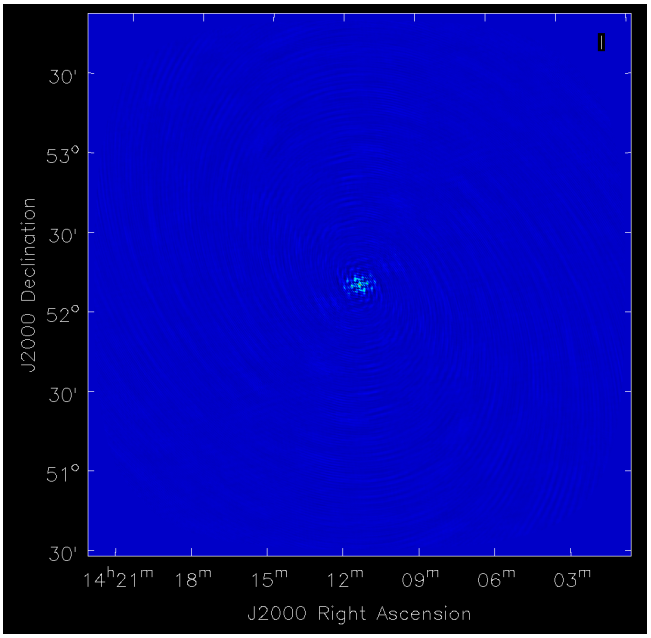


Figure 61: The dirty LBA image for 3C 295 (SB000).

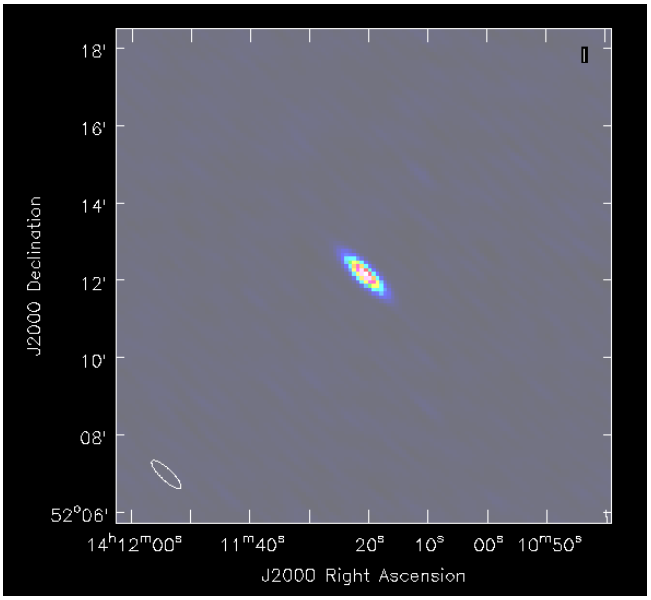


Figure 62: The cleaned LBA image for 3C 295.

14 Useful resources[85]

14.1 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

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

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

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

Busy Thursdays: http://www.lofar.org/operations/doku.php?id=commissioning:busy_wednesdays

14.2 Useful analysis scripts

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

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

You can copy them directly from `/opt/cep/tools/cookbook`. To use them without having to copy them in your working directory, just type:

```
> use Tools
```

For a few of these scripts, you will need to initialize the PyDAL libraries:

```
> use LUS
```

The scripts provided are[86]:

- 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

[85] This Chapter is maintained by R. F. Pizzo, pizzo[at]astron[dot]nl

[86] If you have other scripts that could be useful for other commissioners, please contact Roberto F. Pizzo at pizzo[at]astron[dot]nl

- coordinates_mode.py: routines to work with astronomical coordinates
- plot.py: inspect gain solutions
- solfetch.py: modules required for solflag.py
- solflag.py: carries out solution-based flagging
- solplot.py: modules required for solflag.py
- uvcoverage.py: plots the uv coverage for a Measurement
- plot_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. NDPPP.
- plotElevation.py: given a Measurement Set, plots the elevation of the target source as a function of time
- split_ms_by_time.py: extracts part of a Measurement Set (selected by timerange) and writes out to a new Measurement Set. Optionally excludes selected antennas.
- uvcov.py: plots uv coverage for one or more Measurement Sets. If all Measurement Sets are for the same source at the same time (in other words are different subbands of the same observation), then use the '-s' option to save a lot of time. Do NOT use that option if the input Measurement Sets are not coincident in time.
- uvplot.py: plots data from a Measurement Set in several combinations, in a per-baseline fashion. Not as flexible as casaplotms, but should be faster.

- uvrms.py: performs RM Synthesis on the data in a Measurement Set.
- 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.

14.3 Contact points

Some key contact points are listed below:

- LOFAR Imaging Cookbook - Roberto Francesco Pizzo (pizzo[at]astron[dot]nl)
- NDPPP - Ger van Diepen (diepen[at]astron[dot]nl), Vishambhar Nath Pandey (pandey[at]astron[dot]nl), Adriaan Renting (renting[at]astron[dot]nl), and David Rafferty (rafferty[at]strw[dot]leidenuniv[dot]nl)
- AOFlagger - André Offringa (andre[dot]offringa[at]anu[dot]edu[dot]au)
- BBS - Joris van Zwieten (zwieten[at]astron[dot]nl), Tammo Jan Dijkema (dijkema[at]astron[dot]nl), Vishambhar Nath Pandey (pandey[at]astron[dot]nl)
- AWImager - Ger van Diepen (diepen[at]astron[dot]nl), Cyril Tasse (cyril[dot]tasse[at]obspm[dot]fr), and Bas van der Tol (vdtol[at]strw[dot]leidenuniv[dot]nl), Joris van Zwieten (zwieten[at]astorn[dot]nl)

- Distributed Pipeline Running - John Swinbank (swinbank[at]transientskp[dot]org), Marcel Loose (loose[at]astron[dot]nl), Roberto F. Pizzo (pizzo[at]astro[dot]nl)
- Pyrap/CASA - Ger van Diepen (diepen[at]astron[dot]nl)
- SAGECAL, Shapelets - Sarod Yatawatta (yatawatta[at]astron[dot]nl)
- PyBDSM - David Rafferty (rafferty[at]strw[dot]leidenuniv[dot]nl)

14.4 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[87].

[87] http://www.lofar.org/operations/doku.php?id=commissioning:busy_wednesdays

15 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 Roberto Francesco Pizzo (pizzo[at]astron[dot]nl) using the LOFAR issue tracker[88].

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

[88] 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,

A GNU screen

For some long running processes, as the flagging (NDPPP, see Sect. 5), the calibration (BBS, see Sect. 6) or creating a large image, it can be handy to have a session running on one of the cluster nodes, and then leave that running the background overnight. This can be done by putting the task in the background and use a command like `disown`. Sometimes, however, it is more convenient to have a program running the in the foreground. In such a case, it is difficult to log out without killing the process. GNU screen to the rescue.

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

Have a first look at screen using the help option:

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

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

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

```
> screen
```

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

```
> sleep 30
```

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

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

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

```
screen -r
```

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

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

In case you 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 NDPPP, BBS and/or mwimager can work nicely.

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

```
escape ^Jj
```

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

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

There are many other options; see for example http://www.emacswiki.org/emacs/GnuScreen. 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[89]

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*[90] 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

```
~montes/xmm/simulation
```

All the dependencies of the software are installed in the LOFAR cluster CEP1.

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
`--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

[89] The author of this appendix is Valentina Vacca (vvacca[at]ira[dot]inaf[dot]it).

[90] This software has been developed by Jose Sabater (`jsm[at]roe[dot]ac[dot]uk`).

	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[91]. 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 ~montes/xmm/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 \
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
```

[91] 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.

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 $\sim$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 $S_A/S_T \geq$ `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  ~montes/xmm/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[92] 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 6), 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]

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.

[92] This procedure has been developed by Reinout van Weeren (`rvanweeren[at]cfa[dot]harvard[dot]edu`).

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[93] 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

```
/opt/cep/tools/cookbook/Ateamclipper.py
```

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



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
```

[93] An algorithm based on a subtraction procedure is currently under development.

```
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.

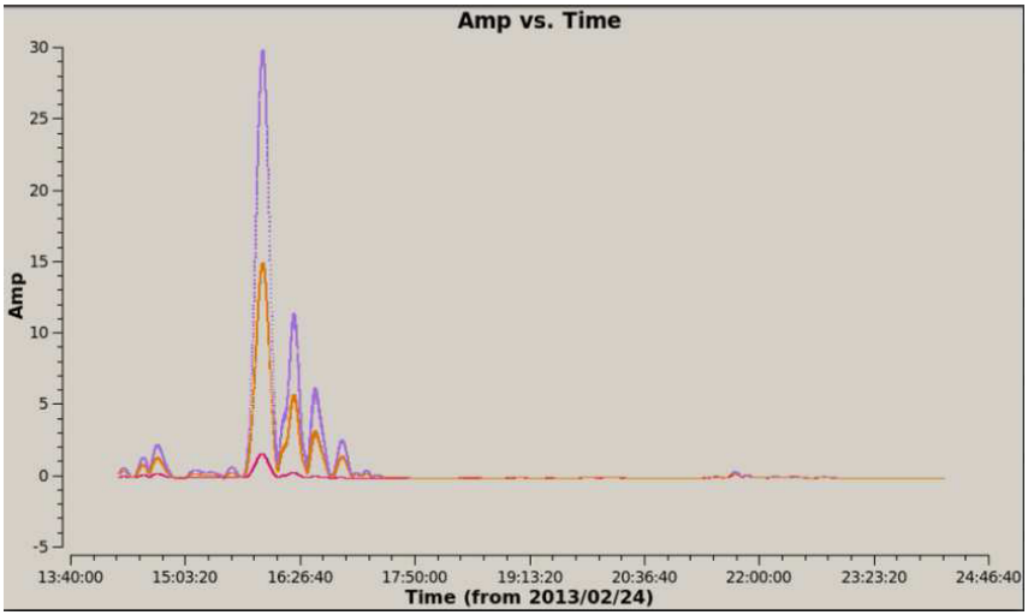


Figure 63: Example of amplitude of A-team sources versus time for a single baseline of an HBA observation. The A-team sources affect the observation just for a fraction of the total observing time.

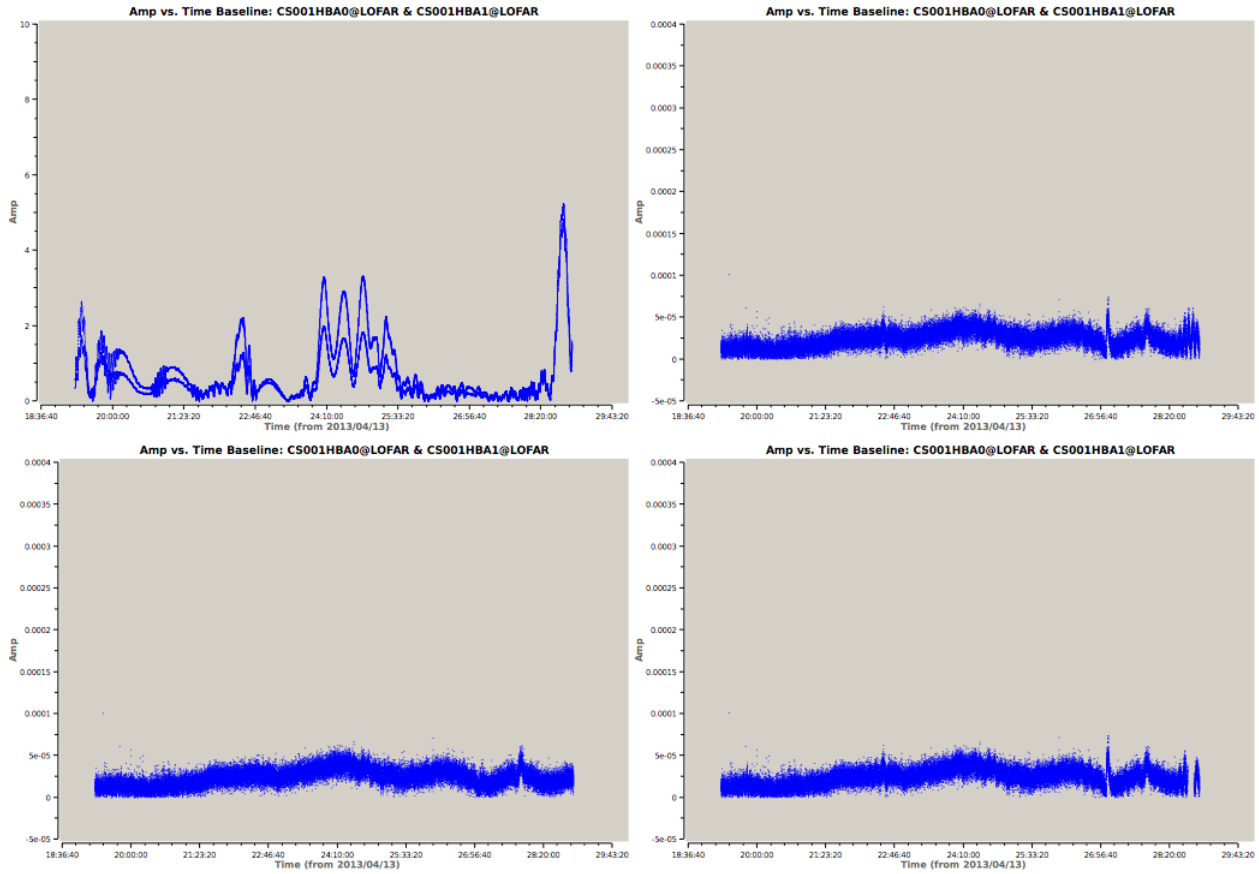


Figure 64: *Top left:* Simulation of the A-team sources VirA, CygA, CasA, and TauA during an HBA observation of 3C 299 (PI: Dr. Chiara Ferrari). *Top right:* Amplitude versus time from the raw data set. *Bottom left:* Amplitude versus time after the standard demixing subtraction of VirA, CygA, CasA, 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.