LOFAR Imaging Cookbook Documentation

Release 23.2

Edited by Sarrvesh Sridhar

Jun 18, 2019

CONTENTS:

CHAPTER

ONE

INTRODUCTION TO LOFAR COMPUTING FACILITIES[1]

1.1 The LOFAR cluster layout

The correlated data coming from the Cobalt[2] correlator are stored on a cluster of machines called *CEP4* cluster. *CEP4* has been added to the LOFAR offline system at the beginning of 2017. Till December 2016, another cluster of computing machines was used to store and process the data (CEP2). Nowadays, *CEP4* is normally used by the Radio Observatory to process the data through the initial stages of the data reduction (flagging and averaging of the visibilities), while another CEP facility is currently used by both the commissioners and LOFAR users to manually play with the data and understand which strategy to use for the calibration and the imaging: the new commissioning cluster *CEP3*. *CEP4* facility provides technologies that were not available on CEP2, especially with respect to resource management.

In the following sections we will focus on discussing the architecture of CEP facilities as well as their usage policies.

1.2 The LOFAR phase 4 cluster - CEP4

The LOFAR CEP4 cluster is composed by 50 compute nodes (**cpu01-50**), 4 GPU nodes (**gpu01-04**), 18 storage nodes (**data01-18**), 50 compute nodes (**cpu01-50**), 2 meta-data nodes (**meta01-02**), 2 head nodes (**head01-head02**) and 1 management node (**mgmt01**). A detailed description of all the packages available on the new cluster and on its network interface can be found on the wiki. Processed data products will usually made available to the user via the Long-Term Archive, but may also be copied to the *CEP3* cluster upon request for further analysis by the user in the original proposal. Due to the intensive nature of the standard data pipelines and the need for these compute resources to be allocated and scheduled by Radio Observatory staff, access to the resources on CEP4 will be strictly limited, with a few exceptions, to the Radio Observatory. In the following, a short description of the computing characteristics/performances of the new cluster is given.

The Lofar Phase 4 cluster consists of:

- 50 compute nodes (called cpu01..cpu50)
- 4 GPU nodes (gpu01..gpu04)
- 18 storage nodes (data01..data18)
- 2 meta-data nodes (meta01..meta02)
- 2 head nodes (head01..head02)
- 1 management node (mgmt01)

[1] This chapter is maintained by M. Iacobelli.
[2] It is located in Groningen, The Netherlands.

Each node is reachable as XXXX.cep4.control.lofar. Users are only allowed on head01 and head02. Each compute node consists of:

- CPU: Intel Xeon E5-2680v3 2.5 GHz (12 cores, HyperThreading disabled)
- Memory: 256GB @ 2133 MHz
- Disk: 2x 300GB 10Krpm SAS RAID
- Network: 2x 1GbE, 2x 10GbE, 1x FDR InfiniBand

Each GPU node consists of:

- CPU: Intel Xeon E5-2630v3 2.4 GHz (8 cores, HyperThreading disabled)
- Memory: 320GB @ 1866 MHz
- Disk: 2x 300GB 10Krpm SAS RAID + 2x 6TB 7.2Krpm SAS RAID
- Network: 2x 1GbE, 2x 10GbE, 1x FDR InifiniBand

Each head node consists of:

- CPU: Intel Xeon E5-2603v3 1.6 GHz (6 cores, HyperThreading disabled)
- Memory: 128GB at 1600 MHz
- Disk: 2x 300GB 10Krpm SAS RAID + 2x 6TB 7.2Krpm SAS RAID
- Network: 2x 1GbE, 2x 10GbE, 1x FDR InifiniBand

The other nodes are not accessible (storage, meta-data, and management nodes).

Storage: The storage and meta-data nodes provide about 2PB LustreFS global filesystem through the InfiniBand network to all nodes in the data partition, thus implying that all nodes see the same data.

Processing: CEP4 will use a SLURM batch scheduling system to schedule and run all observation and processing jobs on the cluster.

NOTE: It is emphasised again that CEP4 is not meant for commissioning work. For that, commissioners can use CEP3 (see access policies here). It is emphasized that the disks on CEP3 are not intended for long term storage of results. As it is impossible for the Observatory to micro-manage the disk space, commissioners should be aware that disk deletions could happen with very little warning.

1.3 CEP3

The CEP3 cluster is available since November 2014 and allows running science processing close to the CEP4 facility as well as for commissioning work. An overview of the hardware and software as well as of major policies and procedures is available on the LOFAR wiki. The cluster consists of 24 equivalent servers with the following specifications:

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

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

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

USAGE POLICY: Observing, CEP4 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 Operations & Support (SOS). Access for Users will be granted only at the discretion of the Science Operations & Support. Users should conform to the access, resource allocation and data deletion policies issued by the SOS at all times.

To have access to CEP3 a formal request must be submitted to sos@astron.nl or be explicitly given in a proposal for LOFAR observing time. When submitting a request users should clearly include the following information:

- A brief explanation of why access to CEP3 is required (e.g., you do not have access to suitable computing resources elsewhere)
- Project to be worked on (i.e. commissioning, cycle or archived data (post) processing)
- Description of the kind of processing for which CEP3 access is requested
- List of collaborators (if any) who should also have access
- General description of the data to be processed (e.g. data size)
- Estimated processing time required

Users awarded with access to CEP3 will be able to access the cluster for a limited period of time (2 months by default). The awarded period starts:

- from the moment the user's data is copied from CEP4 (after Radio Observatory pipeline processing) to CEP3
- following the timeline communicated to the user via the SOS notification and available on the wiki.

At the beginning of a Cycle, users requesting CEP3 processing time in their observing proposals can derive this timeline by checking the observing schedule, which is available here. Access timelines related to observing programs involving observations spread in time will be discussed between the PI and Science Support. In general, info about CEP3 access of users are detailed on the LOFAR wiki. After the granted period on CEP3 has expired, all users data products generated on the cluster will be automatically and promptly removed, to enable new users to have enough disk space to perform their data reduction.

Extensions to the default 8-weeks period will be granted only in exceptional circumstances and only if properly justified through a formal request to be sent to sos@astron.nl no later than 1 week before the expiration of your access privileges. Monitoring of node usage during allocated time will be performed and the evaluation of extension requests will be based on such statistics.

1.4 Logging on to CEP3

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

To access CEP3, begin by logging on to **portal.lofar.eu**[3]

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

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

where *<user name>* is likely to be the user's surname. Type in your default password[4]. Once you are on the portal, you can then log in to the front end (you are requested to use **lhdhead.offline.lofar**)[5] as

```
ssh -Y lhdhead.offline.lofar
```

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

```
ypasswd
```

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

In order to log on to (for example) **lof019**[6], start a Slurm job from the head node **lhdhead.offline.lofar**. Any job will do, but we advice starting an interactive bash shell

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

This should give you a prompt (or more than one if you have more nodes on your job). Once you have the Slurm job running and log using ssh-keys enabled (see the section on *generating SSH keys*), you are allowed standard SSH access with X-forwarding to the reserved nodes from the head node starting from a new terminal screen on the head node **lhdhead.offline.lofar**

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

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

For more information on the cluster architecture/properties, see:

- LOFAR cluster page on the wiki
- CEP3 page on the wiki

1.5 Setting up your working environment

1.5.1 Login scripts

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

- Delete any potential **.profile** or **.cshrc** file that might be in your **$HOME**.
- Copy over the appropriate **cshrc** or **bashrc** depending on your (t)CSH or BASH login shells.

  ```
  ln -s /opt/cep/login/cshrc $HOME/.cshrc
  ```

or

```
ln -s /opt/cep/login/bashrc $HOME/.profile
```

- Exit and log in again. You should now see a welcome message.

[4] Your default password will be communicated to you at the moment of the creation of your Lofar account by Teun Grit.
[5] You may have to log out of and log in again to the portal first.
[6] The Radio Observatory will assign you with a suitable *lof* node to work on.

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.

The Lofar Login Environment provides a basic environment to run all system-installed packages and tools (like python). Non-system packages have been installed in **/opt/cep** and the software environment can be setup in a flexible way using the Modules Software Environment Management software. This provides a flexible way to load and unload specific packages or versions of packages. Some of the useful *module* command to manipulate the software environment are listed below:

Table 1.1: Example module commands to manipulate the software environment

Command	Description	Example
module avail	List all available package and versions	
module load <package>	Load a specific package	'module load lofar'.
module load <package>/<version>	Load a specific version of the package	module load casa/4.2.1
module list	List all loaded packages	
module unload <package>	Unload a package	module unload lofar
module purge	Unload all loaded packages	
module help	Display help information	

Note that loading a particular software package using **module load** will also load other packages that the specified package depends on. For example, ""

module load lofar

will load dependent packages like casacore, casarest, and python-casacore.

Also note that the **lofar** module loads the latest stable version of the LOFAR software which is released twice a year. To load an older stable release of LOFAR software, you should load **lofar/<version>**. If you wish to use the latest "daily build" which contains pre-release software that are under active development, you can do so by running **module load lofim**. The pre-release software is built on CEP3 every day and so you load the pre-release software from a specific day by running **module load lofim/<day>**.

Some of the commonly used packages are

- aips
- AOFlagger
- CASA
- Sagecal
- Dysco
- Generic pipeline
- Prefactor
- Factor
- PyBDSF
- DAL
- DS9
- Karma
- Duchamp

- LoSoTo
- LSMTool
- RMextract
- RMSynthesis
- PyRMSynth
- Wsclean

Detailed information about how to activate and use the above packages can be found on the LOFAR wiki.

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

```
casa lofar
```

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

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

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

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

You can now **cd** into it and use it as your working space. You can copy in here the data provided by the Radio Observatory by e.g. typing:

```
> scp -r <user name>@lhdhead.offline.lofar:/data/<user>/<LOFAR dataset> .
```

where **<LOFAR dataset>** has the syntax **LXXXXX**[8].

1.5.2 Generation of SSH keys

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

The following steps will allow you to generate SSH keys on a Linux or OS X machine. From the front end node **lhdhead.offline.lofar** set up passwordless access to the **lof** nodes (if you have a job running to provide you access) via SSH:

- create the directory $HOME/.ssh if it does not already exist.
- The following command allows you to generate the SSH keys

  ```
  ssh-keygen -t rsa
  ```

- The above command creates a set of public and private keys in **$HOME/.ssh**. Copy the public key to **authorized_keys** as

  ```
  cp ~/.ssh/id_rsa.pub ~/.ssh/authorized_keys
  ```

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

- Note that **$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.

See the LOFAR wiki for additional information and instructions for a Windows machine.

1.5.3 Disable SSH Host Key Checking

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

```
$ ssh lof019
The authenticity of host 'lce019 (10.176.0.19)' can't be established.
RSA key fingerprint is 73:27:96:cd:f5:04:b7:c3:57:47:49:97:8b:87:8b:15.
Are you sure you want to continue connecting (yes/no)?
```

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

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

1.5.4 Copying data from and to CEP3 cluster

Data transfers from CEP4 to CEP3 should always be coordinated with the Radio Observatory[7]. Data retrieval from LTA locations as well as from/to other computing facilities is also possible as detailed in the LOFAR wiki. Although access to the CEP3 systems can only be done through the **lhdhead.offline.lofar** head node, data transfers to the outside world can be done directly. *Because data will be transferred via the LOFAR portal, care should be taken not to flood the available network bandwidth with the public internet. Thus we recommend to limit the bandwidth to not disturb the portal access of other users. Note that the portal capacity is 120MB/s.*

[7] sos[at]astron[dot]nl

CHAPTER

TWO

DATA INSPECTION[1]

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. 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.

- The quality of raw measurement sets can be inspected using data quality inspection plots that are generated by the Radio Observatory. These plots are available online for up to three weeks after the time of observation. After that period, they are compressed and stored offline. Users can access them by creating a ticket through the JIRA Helpdesk.
- 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.

- Python-casacore (formerly "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. These allow you to customize what is plotted, and can be significantly faster than CASA for plotting large datasets. An extensive description of the python-casacore utilities is available online.
- Visibilities can be plotted relatively rapidly by making use of the combination of python-casacore and the plotting package PGPLOT, both of which work quickly. The script **uvplot.py**[2] can be used to plot visibility data and can be significantly faster than the PLOTMS task in CASA.
- CASA is the python-based next generation replacement for AIPS/AIPS++ and can be used to display the data. Be aware that trying to inspect the raw visibilities with CASA will produce a "**segmentation fault**" error. To avoid this, you should make a copy of the dataset with DPPP (see the first example parset in Sect.~ref{theparsetfile}). Moreover, when opening with CASA a MeasurementSet observed between May and October 2011, you will get an **Unrecognized mount type** error[3]. CASA has multiple functionalities that allows one to inspect different data products.
  - **casabrowser** can be used to inspect the content of a Measurement Set.
  - **casaplotms** can be used to visibility data against different parameters.
  - **casaviewer** can be used to inspect images.
  - For a detailed information and examples of how to use CASA, we direct users to the CASA Guides.

[1] This chapter is maintained by A. Shulevski and Valentina Vacca.

[2] The script was written by George Heald and is available through the LOFAR GitHub repository.

[3] 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 as *taql 'update <ms name>/ANTENNA set MOUNT="X-Y" '*

- 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. The script **plot_Ateam_elevation.py**, which is available as part of the LOFAR repository, can be used to inspect the contribution of these A-team sources to the observed visibility data.
- For manipulating measurement sets, 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 detailed documentation of TaQL can be found here and here.

2.1 Analyzing the data quality with aoqplot

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

Basically, the tool allows you to plot standard deviations and RFI percentages and some other quantities, over time, frequency, baselines and the time-frequency domain of the full observation, i.e., over all sub-bands. Since an observation can be several terabytes of data, the performance of standard tools to perform such analysis is an issue, and the **aoqplot** was designed to overcome this problem. Fig.~ref{aoqplot-window} 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.

2.1.1 Usage

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

```
$ aoqplot yourobservation.gvds
```

The gvds file is given by the observatory and describes the observation, in particular the locations of the measurement sets. The **aoqplot** tool will now connect to all the nodes with an ssh connection, start a bash-session on the node and start a client there that connects back to your node and sends 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 lof001**, you should be able to start **aoremoteclient** within bash without a **module load lofar**. The easiest way of doing this is by putting **lofar** in **$HOME/.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
```

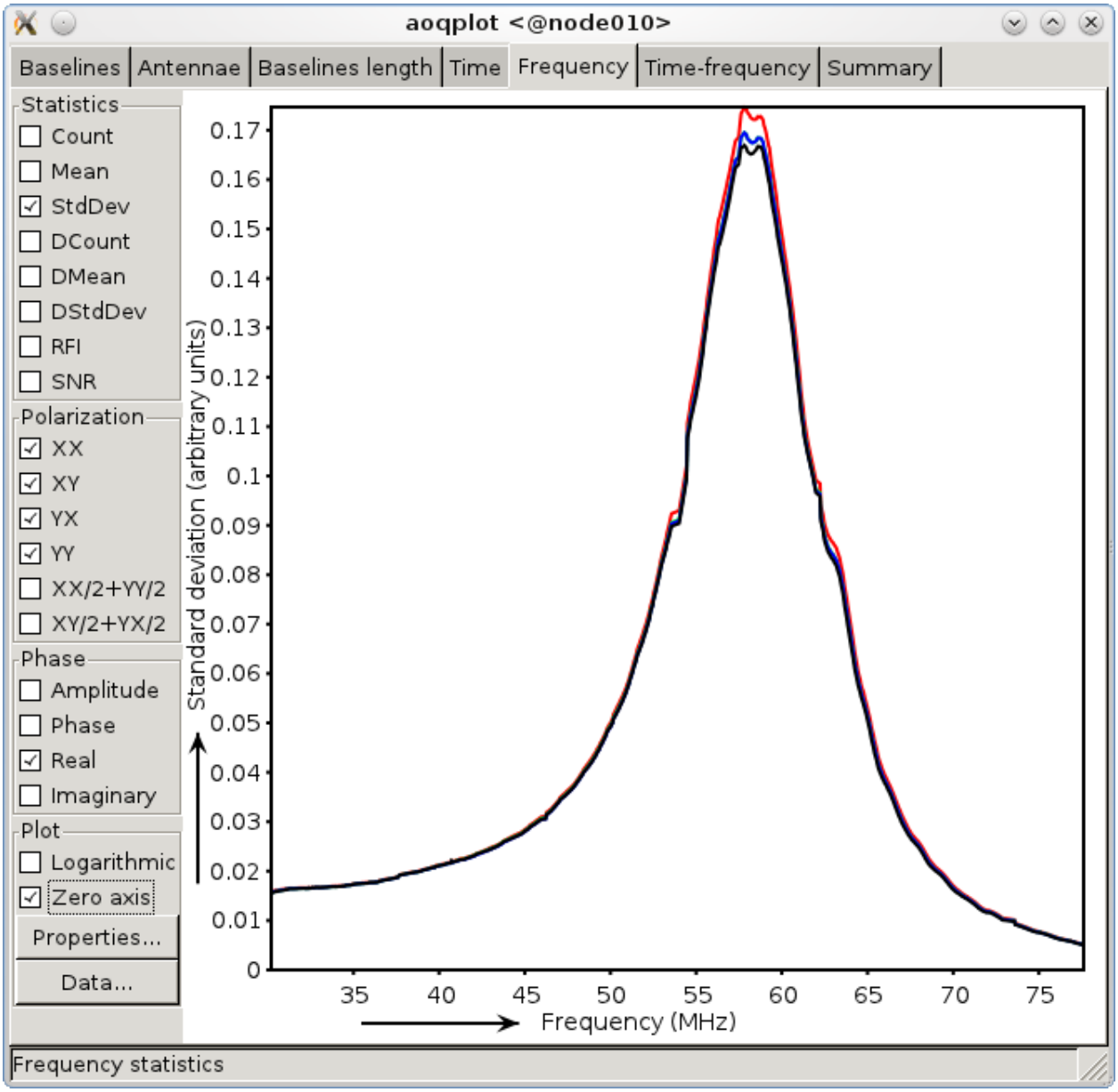


Fig. 2.1: The **aoqplot** window showing the standard deviation of the data over frequency of a full observation.

2.1.2 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.~ref{aoqplot-window}). 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.

2.1.3 Background information

The **aoqplot** tool works together with DPPP. Recent versions (>21 December 2011) of DPPP will add so-called quality statistic tables to a measurement set. These tables circumvent having to read the entire DATA column of a measurement set to get the basic statistics. The way they are stored is described in the quality statistics proposal written by Andr'e Offringa. Because the statistics plotting tool require these tables, you can not directly plot statistics of measurement sets that are averaged by an older DPPP, or have not been averaged at all.

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

respectively, are calculated as:

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

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

If you select "amplitude" in the **aoqplot** user interface, the actual plotted quantity is:

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

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

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

```
aoquality collect SB000.MS
```

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

2.2 Additional information: manual flagging in CASA

While manual flagging will not be practical once the pipeline is completed, during early stages it may be useful to remove remaining RFI in order to test the calibration or imaging routines. Flagging tasks in CASA include **FLAGDATA** and **FLAGCMD** for command-line based flagging. The task **PLOTMS** offers GUI-based flagging. **PLOTXY** can also be used for manual flagging, but users should be aware that it is being deprecated in favor of **PLOTMS** and may not be available in future releases of CASA. Once the CASA **PLOTMS** has loaded and data is visible, click the **Mark Region** button, highlight data that you wish to flag, click the **Flag** button, and **Quit** once you are finished.

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

Observations at 'low' elevation (below $\sim 30^\circ$ for Cygnus~A, and below $\sim 40^\circ$ for 3C196) are sufficiently noisy that they are of limited use. These bad time ranges need to be identified and removed. This could be done through DPPP, but also by manual flagging in CASA or using the CASA **SPLIT**[4] task or the python script **split_ms_by_time.py**[5]. 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.

2.3 The Drawer

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

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

[5] The script is available through the LOFAR GitHub repository.

observation (one can generally generate an all sky plot in less than a few minutes). The concept behind the Drawer was already known and used in AIPS (task FRMAP).

As the fringes "produced" by each individual baseline are rotating on the sky, each source modulates the visibility, depending on its distance from the phase center (far away sources give a higher fringe rate). The Drawer performs an FFT of the visibility of each given baseline in a particular timeslot, along the time axis, and finds the dominant frequency. From that value, and from the "speed" of the given baseline in the uv plane, it solves a simple equation and derives a line on the sky. Per baseline, it reflects all the possible places where the source producing the given detected modulation could be. The lines of all the baselines/timeslots are then gridded onto an image. The pixel values do not reflect the flux of the sources, but the log of the occurrence of fringe finding. The current version of the software does not deal with data chunks yet, i.e. it first reads the whole MS and puts the visibilities into memory. Therefore it performs quicker on averaged datasets containing a few channels.

2.3.1 Examples

Once you have initialized your work environment, you can access drawMS as

```
> drawMS -h

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

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

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

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

   --ColName=COLNAME   Name of the column to work on. Default is DATA. For
                       example: --ColName=CORRECTED_DATA
   --uvrange=UVRANGE   UV range (in meters, not in lambda!). Default is
                       0,10000000. For example: --uvrange=100,1000
   --wmax=WMAX         Maximum W distance. Default is 10000000.
   --timerange=TIMERANGE
                       Time selection range, in fraction of total observing
                       time. For example, --timerange=0.1,0.2 will select the
                       second 10% of the observation. Default is 0,1.
   --AntList=ANTLIST   List of antennas to compute the lines for. Default is
                       all. For example: --AntList=0,1,2 will plot 0-n, 1-n,
                       2-n
   --FillFactor=FILLFACTOR
                       The probability of a baseline/timeslot to be
                       processed. Default is 1.0. Useful when large dataset
                       are to be drawn. For example --FillFactor=0.1 will
                       result in a random selection of 10% of the data

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

   --timestep=TIMESTEP
                       Time step between the different time chunks of which
```

(continues on next page)

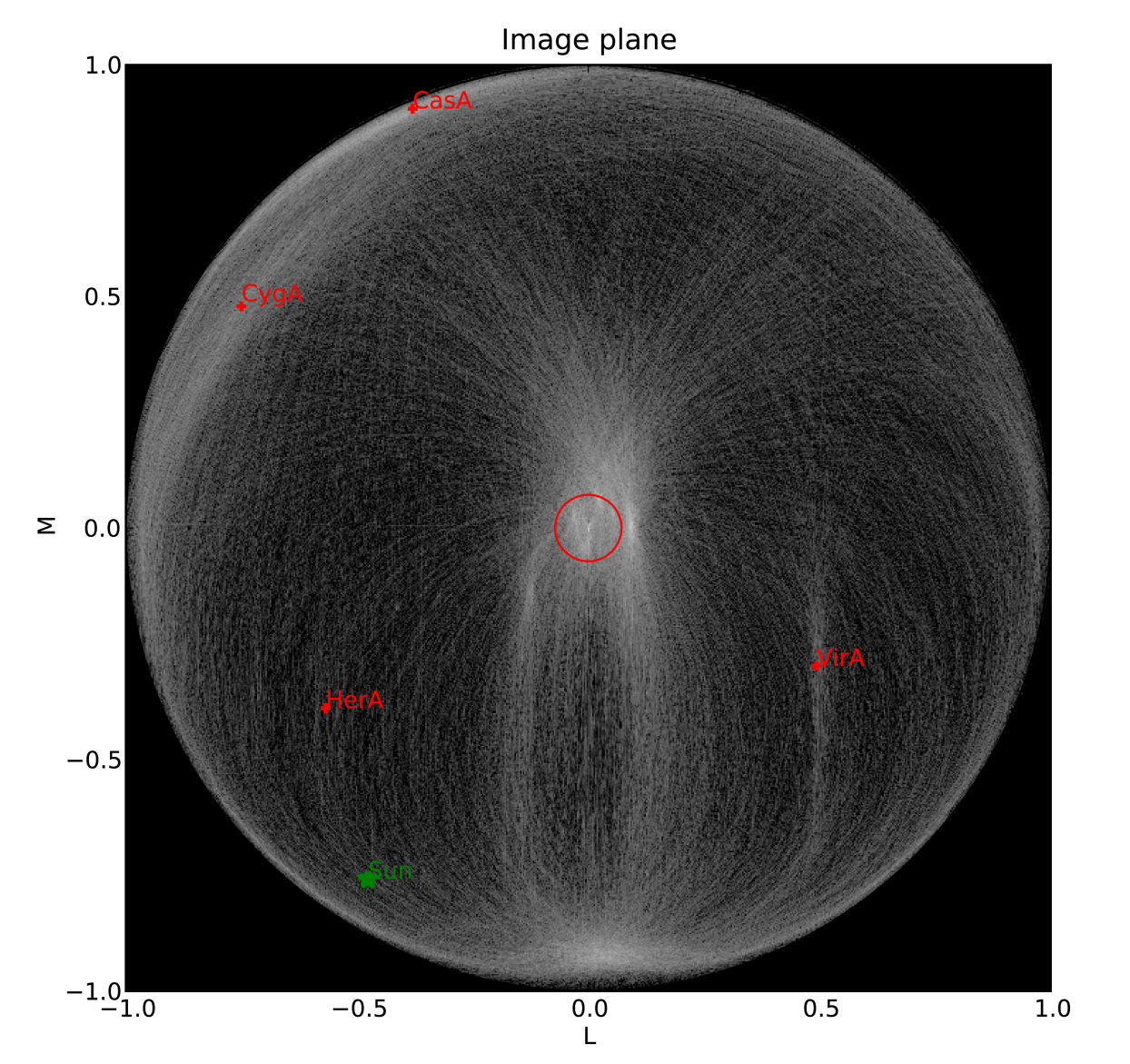


Fig. 2.2: DrawMS is a simple algorithm that allows to quickly recover spatial information on the sources that have the brightest apparent flux. In this example, drawMS is run on the raw data of an observation of the Bootes field. One can clearly see the contribution from CasA, CygA, and TauA, while there is no direct contribution from the Sun.

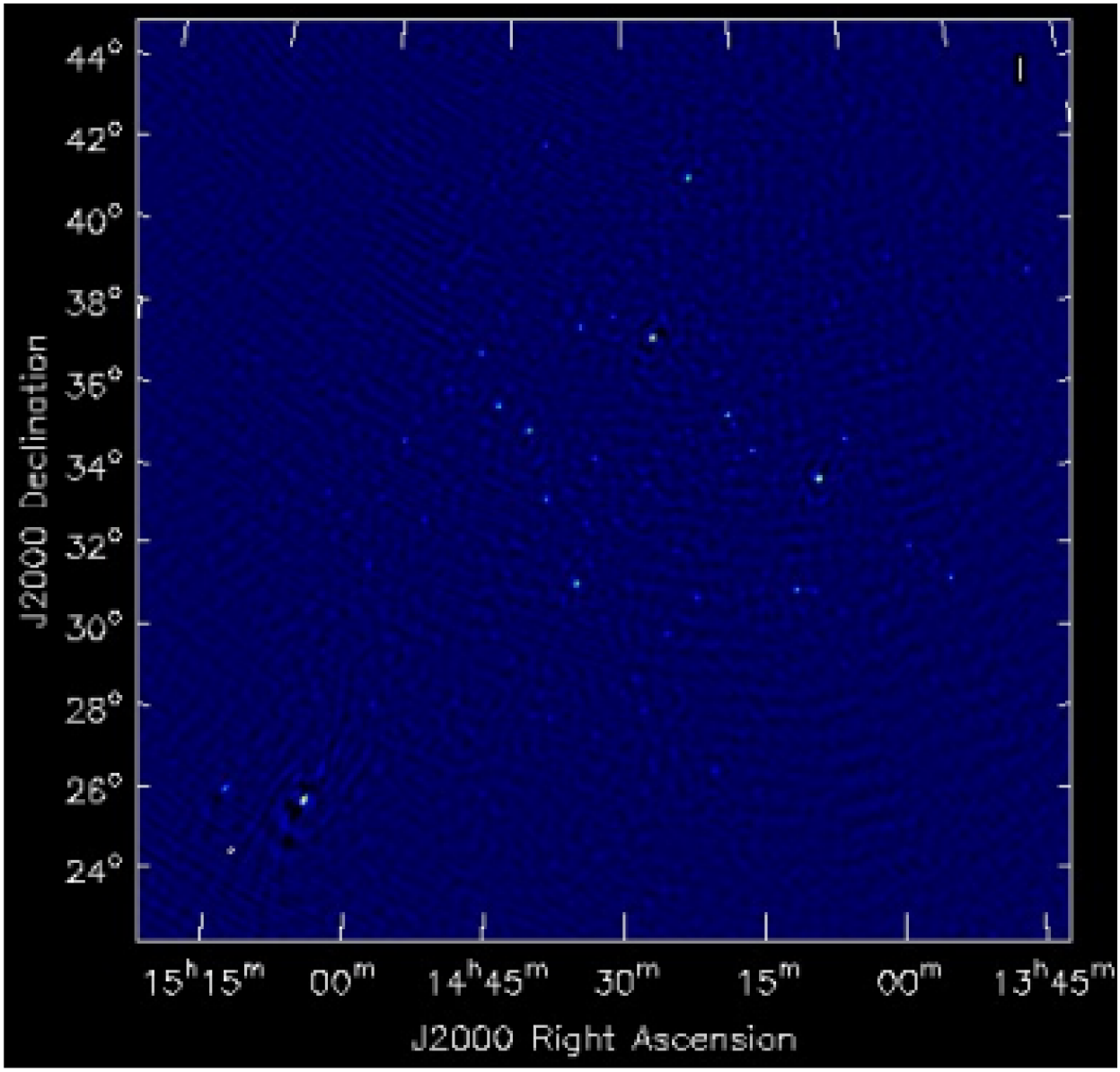


Fig. 2.3: A wide field of view image of the Bootes field which is computationally expensive to generate.

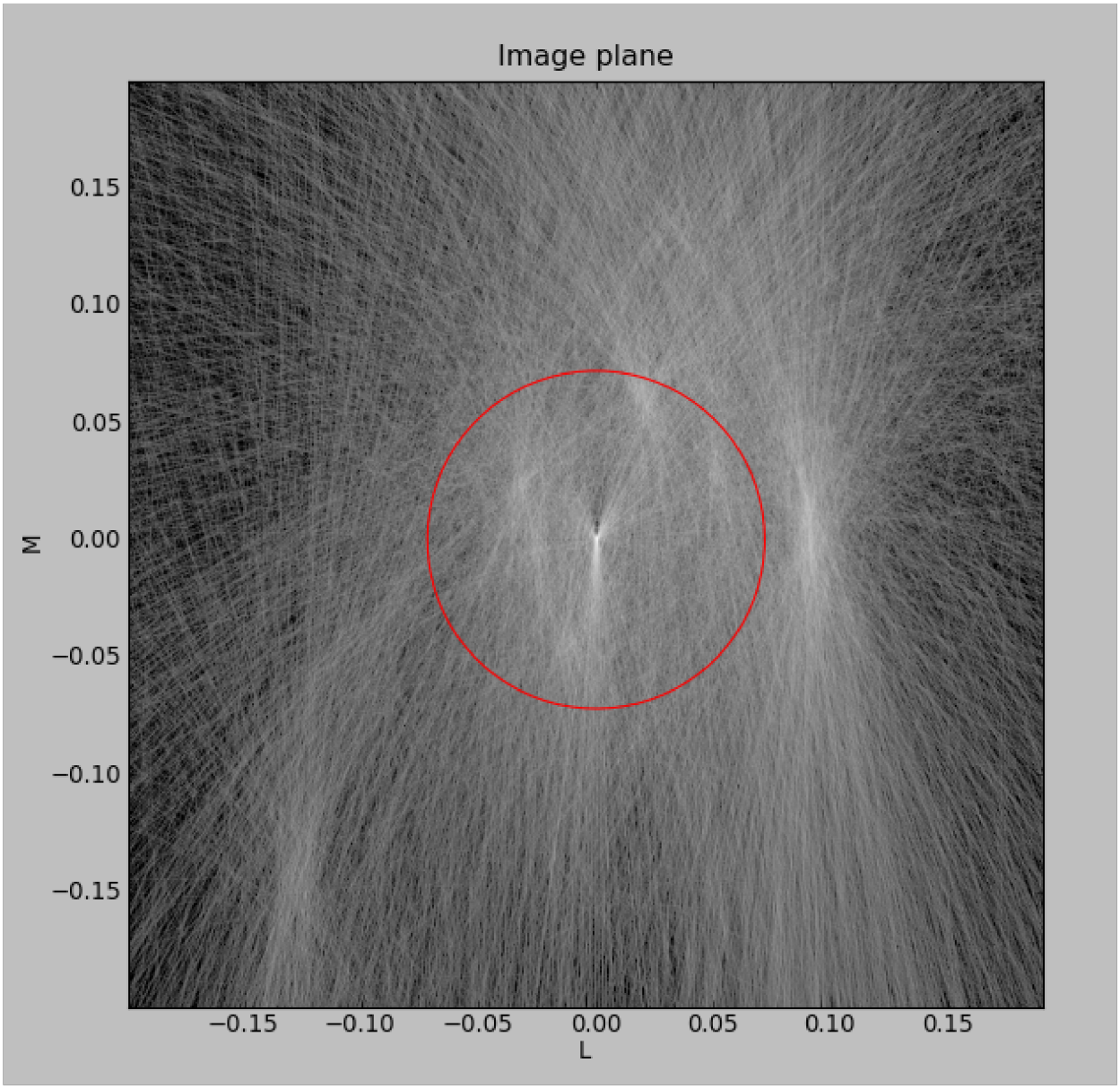


Fig. 2.4: In contrast to the image in Fig. 2.3, the output of drawMS takes a few minutes to generate. Line plots show the overdensities corresponding to real sources in the image.

(continued from previous page)

```
                        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 shown in Fig. 2.3, on the raw data:

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

The following command lets you make a movie of the line plots like the one shown in Fig. 2.3

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

CHAPTER

THREE

AOFLAGGER[1]

The frequencies covered by LOFAR are considerably affected by RFI, both in the low and the high band. Without proper flagging of the RFI, the RFI affects the image quality or can make calibration fail. This chapter describes how to analyze and flag LOFAR observations using the AOFlagger.

In most cases, one would normally use the AOFlagger step in NDPPP with the default settings to flag the data. In this chapter we look at what the flagger does and how to alter its behaviour.

3.1 Tutorial

The tutorial data can be downloaded from the LOFAR LTA. Please refer to the chapter on practical examples for details.

The AOFlagger program comes with a tool called **rfigui** that is part of the **lofar** module on the CEP3 cluster. If you haven't done so, load the module:

```
module load lofar
```

We will use the rfigui to display the measurement set. The rfigui will not write to the measurement set, so you do not need to copy the measurement set yet. Use the following command to open the measurement set in rfigui:

```
rfigui L74759_SAP000_SB001_uv.MS
```

3.1.1 Browsing baselines

A window should pop-up which can be used to select how to open the measurement set. Note that the visualized column can be selected here, which is useful to analyze data in a different column (e.g. CORRECTED_DATA). Since this is raw data, it will only have a DATA ("observed") column, so leave the settings as they are and press "Open".

Another window will pop-up which allows you to select two antennas, a band and a field. Since LOFAR measurement sets never have multiple bands or fields, they will always have only one option. With the two antenna selection boxes, you can select which cross-correlation to visualize. Select antenna CS002HBA0 in both selection frames and press "Load". It will take some while before the visibilities are loaded.

You should now see a dynamic spectrum (time on the x-axis, frequency on the y-axis) of auto-correlation CS002HBA0 x CS002HBA0. Note several things:

- When you move the mouse over the dynamic spectrum, the status bar will provide some information of the visibilities at the location of the mouse.

[1] This author of this chapter is Andre Offringa.

- At the bottom of the plot is a purple line. The rfigui uses purple to indicate visibilities that are flagged. In this case, it is flagged because in raw LOFAR data, channel 0 contains invalid data due to how the poly-phase filter works. The purple overlay can be turned off with the "Original flags" toggle button in the toolbox.
- Note the sharp patterns that recur in a horizontal way, such as at 119.05 and 119.18 MHz. These are common, and are caused by normal transmitters. Use the mouse to find which features I mean.
- Around 13.30 h, vertical features are visible. These can be caused by broadband RFI such as lightning, or by a malfunctioning instrument.
- Finally, there are some big wavy features visible. These are intermodulations in the receivers, and are only present in auto-correlations (and sometimes in the cross-baseline of a dual split station, e.g. CS002HBA0 x CS002HBA1).

Auto-correlations are much more sensitive to RFI. Because the auto-correlations are normally not used in imaging, in fact this RFI situation is not representative. Let's go to a cross-correlation: press the "Forward" button. This will load the correlations for the baseline of this antenna by the next antenna, in this case CS002HBA0 x CS002HBA1. Notice that the RFI situation is much better, and the dominating features are the consistent transmitters at 119.05 MHz and 119.18 MHz. Press "Forward" a few more times and notice the changes.

Let's go to a remote station: go to the "Go" (newer version: "Browse") menu and select "Go to…". Load baseline CS004HBA0 x RS106HBA. Note the smooth vertical features in the background. These are fringes caused by observed sources. Therefore, a flagging strategy should not flag these.

3.1.2 Plotting & flagging

Power spectra and time-power plots can be helpful to analyze the magnitude of RFI. In menu "Plot", select "Power spectrum". A pop-up window appears with the power over frequency over the selected region in the main window. The RFI is clearly visible as spikes in the spectrum. Note also that the bandpass shape is curving down at 119.23 MHz. One normally does not want to include these edge channels because of these features, and they are therefore cut-off in NDPPP (see the NDPPP tutorial).

Now select "Plot time scatter" in the "Plot" menu, and notice how the RFI looks like in this plot. In this plot, different colours show the different polarizations. Also try the "mean spectrum" and "power vs time" plots, and notice how they differ.

Normally, the time-frequency plot shows the Stokes I power of the visibilities. However, internally AOFlagger has loaded complex correlations for all four polarizations. Select "Data" → "Keep real data". Fringes are more pronounced in the real data. When you press one of the "Keep…" options, the rfigui will no longer keep the other parts in memory. This can be confusing: for example, try selecting "Data" → "Keep imaginary data" and understand the error message. To see the imaginary data, first reload the baseline with "Go" → "Go to…" (new version: refresh button). Try analyzing the different parts of this correlation by using the other "Data" → "Keep…" buttons and reloading the baseline when necessary.

Before continuing, close the plot window and reload the baseline.

3.1.3 Testing a flagging strategy

Select "Actions" → "Execute strategy". A progress window will pop-up, which you can close once the flagging is done. Notice that yellow flags have appeared in the time-frequency plot. The default flagging strategy has indeed flagged all visible RFI. Additionally, it has flagged a bit of the edge channels (e.g. in the top-right), because of the curvature in the pass-band at the edges. This is normally not a problem, since these channels will be cut off with NDPPP.

Try toggling the yellow flags off and on with the "Alternative flags" button on the toolbar. Before continuing, make sure that the yellow flags are shown. Plot the power spectrum and time scatter plots as before. The power spectrum will now have two lines: the original one, and one that shows the spectrum after the flagged visibilities have been

taken off. The time scatter plot will only show unflagged samples. Indeed, the RFI samples have been removed, and a clear difference between XX/YY and XY/YX is visible. If you hide the yellow flags and replot the scatter plot, the flagged samples will be shown again.

Clear the flags by reloading the baseline.

3.1.4 Editing the flagging strategy

Press "Action" → "Edit strategy". A window will pop-up that shows a tree that represents the flagging script. Every row is an action, and certain actions can have sub-actions[2].

Notice that there are two "SumThreshold" actions: one is in the "iterate 2 times" loop, while the other is below the loop. When you press one of the SumThreshold actions, a settings box will appear. Set the "Base sensitivity" to 0.7 and press "Apply" for both SumThreshold actions. Now, rerun the strategy with "Actions" → "Execute strategy" and notice the difference.

Try playing with the settings below, and rerun the flagger each time. Do not forget to press "Apply" after changing the settings of an action. If you make modifications you don't like, you can press the "Default" button in the edit startegy window to reset the strategy.

- The strategy normally slowly converges by iterating a couple of times. Only in the last SumThreshold action, the maximum sensitivity is used. The "Iterate 2 times" action performs this slow convergence. In the "Iterate 2 times" action, try changing the number of iterations to 1 and see if this is enough for this baseline.
- Normally, each polarization is searched for RFI. Change the "For each polarization" action so that only Stokes Q is searched for RFI.
- In the "Statistical flagging" action, play with the "Minimum time ratio" and "Minimum frequency ratio". For example, is a "Minimum time ratio" of 60% a good setting?
- Try setting the threshold of the "Time selection" action to 2.5.

This subband is a rather good subbands. While for LOFAR many subbands are like this, there are also subbands which are much worse. Typically you want to try the strategy on some good and bad subband, as well as on some small on some long baselines. You can chose to flag different subbands with a different strategy, but in general it is easier to have one generic strategy. It is also generally better to flag slightly too much than too little. Finally, it is not necessary to consider the flagging speed.

Once you have made a modified strategy, you can save it with the "Save" button in the "Edit strategy window". When saving a strategy, give the filename an extension of ".rfis". These strategies can be used by the AOFlagger step inside NDPPP, to flag and average the data at the same time, as well as that they can be used by the "**aoflagger**" command line program. It is normally faster to do this with NDPPP.

3.1.5 Running the aoflagger command line program

We are now going to flag the measurement set without running NDPPP. To do this, we need write access to the measurement set. Therefore, copy the measurement set mentioned above to your scratch directory. Measurement sets that are created by the LOFAR correlator have a special "raw" format. To make the flag table writeable by aoflagger, it is necessary to store the flags with a different format. This can be done with the following command:

```
> makeFLAGwritable L74759_SAP000_SB001_uv.MS
Successful read/write open of default-locked table L74759_SAP000_SB001_uv.MS:
  23 columns, 5337090 rows
Created new FLAG column; copying old values ...
FLAG column now stored with SSM to make it writable
```

[2] A description of the steps in the pipeline is given in this paper.

The measurement set can now be flagged with the aoflagger command. To get a list of commands, run the program without parameters:

```
> aoflagger
AOFlagger 2.7.1 (2015-07-01) command line application
This program will execute ...
```

Two settings need to be changed: i) for large raw sets, it is best to use the indirect reading mode; ii) since we have made a modified strategy, we want to specify our new strategy. If you are using aoflagger 2.7, you can immediately specify the .rfis file that you have created in the gui. At the point of writing, you will need to use a file which has been prepared by me, because there is an issue with different versions of the gui and the command line aoflagger.

Combined all together, run aoflagger like this:

```
> aoflagger -indirect-read -strategy ~offringa/tutorials/aoflagger-tutorial.rfis␣
↪L74759_SAP000_SB001_uv.MS
AOFlagger 2.7.1 (2015-07-01) command line application
...
0% : +-+-+-+-+-+-Initializing...
...
```

Please note that the current working directory will be used as a temporary storage location, and a lot of temporary data will be written there. Once the flagger is done, you can open the measurement set in the rfigui, and the found flags will be shown with purple.

3.2 Documentation

The manual for AOFlagger can be found on the site http://aoflagger.sourceforge.net/. The properties and performance of the AOFlagger have been described in the following papers:

- Post-correlation radio frequency interference classification methods.
- A morphological algorithm for improving radio-frequency interference detection.

CHAPTER

FOUR

THE DEFAULT PRE-PROCESSING PIPELINE (DPPP)[1]

DPPP (default pre-processing pipeline) is the initial processing routine for LOFAR data. It is capable of flagging, averaging, gain calibration, and various other operations described below (see following sections for more details). It is possible to define an arbitrary operation in Python (or C++), which could be used for trying out new operations. The list of functions that can be carried out using DPPP are

- Flagging (automatic or manual) including the AOFlagger
- Average in time and/or frequency
- Phase shift to another phase center
- Count flags and writing the counts into a table for plotting purposes.
- Combine subbands into a single MeasurementSet
- Demix and subtract A-team sources outside the field of view
- Add stations to form a superstation (for long baselines)
- Filter out stations, baselines, channels, times
- Predict a sky model
- Apply or de-apply the LOFAR beam model
- Do gain calibration
- Apply calibration solutions
- An arbitrary step defined in user code (C++ or Python)

Each of these operations works on the output of the previous one in a streaming way (thus without intermediate I/O to disk). The operations can be combined in any way. The same type of operation can be used multiple times with probably different parameters. The operations are defined in a so-called **parset** file, a text file containing key-value pairs defining the operations (steps) and their parameters.

The ability to execute an arbitrary DPPP step is implemented by means of dynamically loadable shared libraries or by a script written in Python. Especially the latter is a nice way to add experimental operations to DPPP. It is described in more on the LOFAR wiki.

The input to DPPP is any (regularly shaped) MeasurementSet (MS). Regularly shaped means that all time slots in the MS must contain the same baselines and channels. Furthermore, the MS should contain only one spectral window; for a multiband MS this can be achieved by specifying which spectral window to use. The data in the given column are piped through the steps defined in the **parset** file 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 write the averaged data to a new MeasurementSet. If time slots are missing, flagged data containing zeroes are inserted to make the data nicely contiguous for later processing.

[1] This section is maintained by Ger van Diepen and Tammo Jan Dijkema.

The output can be a new MeasurementSet, but it is also possible to update the flags or data in the input. When doing operations changing the meta data (e.g., averaging or phase-shifting), it is not possible to update the input MeasurementSet. Any step can be followed by an output step creating (or updating) a MeasurementSet, whereafter the data can be processed further, possibly creating another 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 python-casacore's **msconcat** command, because **msconcat** keeps the individual spectral windows, while DPPP combines them into one.

Detailed information on DPPP can be found here. For specific questions regarding this software, you can contact the software developers, Tammo Jan Dijkema or Ger van Diepen.

4.1 How to run DPPP

The proper environment has to be defined to be able to run LOFAR programs such as DPPP. Normally this is done by the command:

```
module load lofar
```

Some parameters have to be given to DPPP telling the operations to perform. Usually they are defined in a **parset** file (see section on *the parset file* for details). The DPPP task can be run like:

```
DPPP some.parset
```

It is also possible to specify parset keys on the command line, which will be added to the keys given in the (optional) parset file or override them. For example:

```
DPPP msin=in.MS steps=[] msout=out.MS
```

does not use a parset file and specifies all keys on the command line. Note this run will copy the given MeasurementSet, while flagging NaN values. The command:

```
DPPP some.parset msout.overwrite=true
```

uses a parset, but adds (or overrides) the key **msout.overwrite**. If no arguments are given, DPPP will try to use the **DPPP.parset** or **NDPPP.parset** file. If not found, DPPP will print some help info.

An example **NDPPP.parset** file can be found in the LOFAR GitHub Cookbook repository: https://github.com/lofar-astron/LOFAR-Cookbook/tree/master/Parset

DPPP will print outputs to the screen or a logger, including some brief statistics like the percentage of data being flagged for each antenna and frequency channel.

The next sections give some example DPPP runs. They only show a subset of the available parameters. DPPP is described in more detail on the LOFAR operations wiki which also contains several examples. It also contains a description of all its parameters.

4.1.1 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 cannot be used (yet) in 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.

4.1.2 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.

4.1.3 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. Note that it might be better to flag the channels instead of removing them. In that way no gaps are introduced when concatenating neighbouring subbands.
2. Two operations are done: flagging followed by averaging. The parameters for these steps are specified thereafter using the step names defined in the **steps** parameter. Note that arbitrary step names can be used, but for clarity they should be meaningful.
3. A step is defined by various parameters. Their names have to be prefixed with the step name to connect them to the step. In this way it is also possible to have multiple steps of the same type in a single DPPP run. Usually the type of step has to be defined, unless a step name is used representing a type. In this case the aoflagger is used, an advanced data flagger developed by Andr'e Offringa. This flagger works best for large time windows, so it tries to collect as many data in memory as possible. However, to avoid memory problems this step will not use more than 25% of the available memory.
4. The next step is averaging the data, 60 channels and 5 time slots to a single data point. Averaging is done in a weighted way. The new weight is the sum of the original weights.

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

4.1.4 Update flags using the preflagger

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

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

No output MS is given meaning that the input MS will be updated. Note that the {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.

4.1.5 Remove baselines and/or channels

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

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 option (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 before, even if done in another program like **msselect**). In this way it can be used to 'normalize' a MeasurementSet.

4.1.6 Combining stations into a superstation

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

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

This example adds stations CS002 till CS006 to form a new station CSNew. The example shows that the parameter **stations** 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 are calculated by summing the autocorrelations of the input stations. By setting parameter **sumauto** to false, they are calculated from the crosscorrelations of the input stations.

4.1.7 Update flags for NaNs

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

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

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

4.1.8 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.

4.1.9 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 the chapter on data inspection.

There are two ways to do the demixing:

- The old demixer will demix in the same way for the entire observation without taking temporal variations into account. One can define:
  - The baselines to use.
  - The (A-team) sources to solve for and to subtract.
  - If the target has to be ignored, solved or deprojected.
  - Possibly different time and frequency averaging factors to use for demix and subtract.

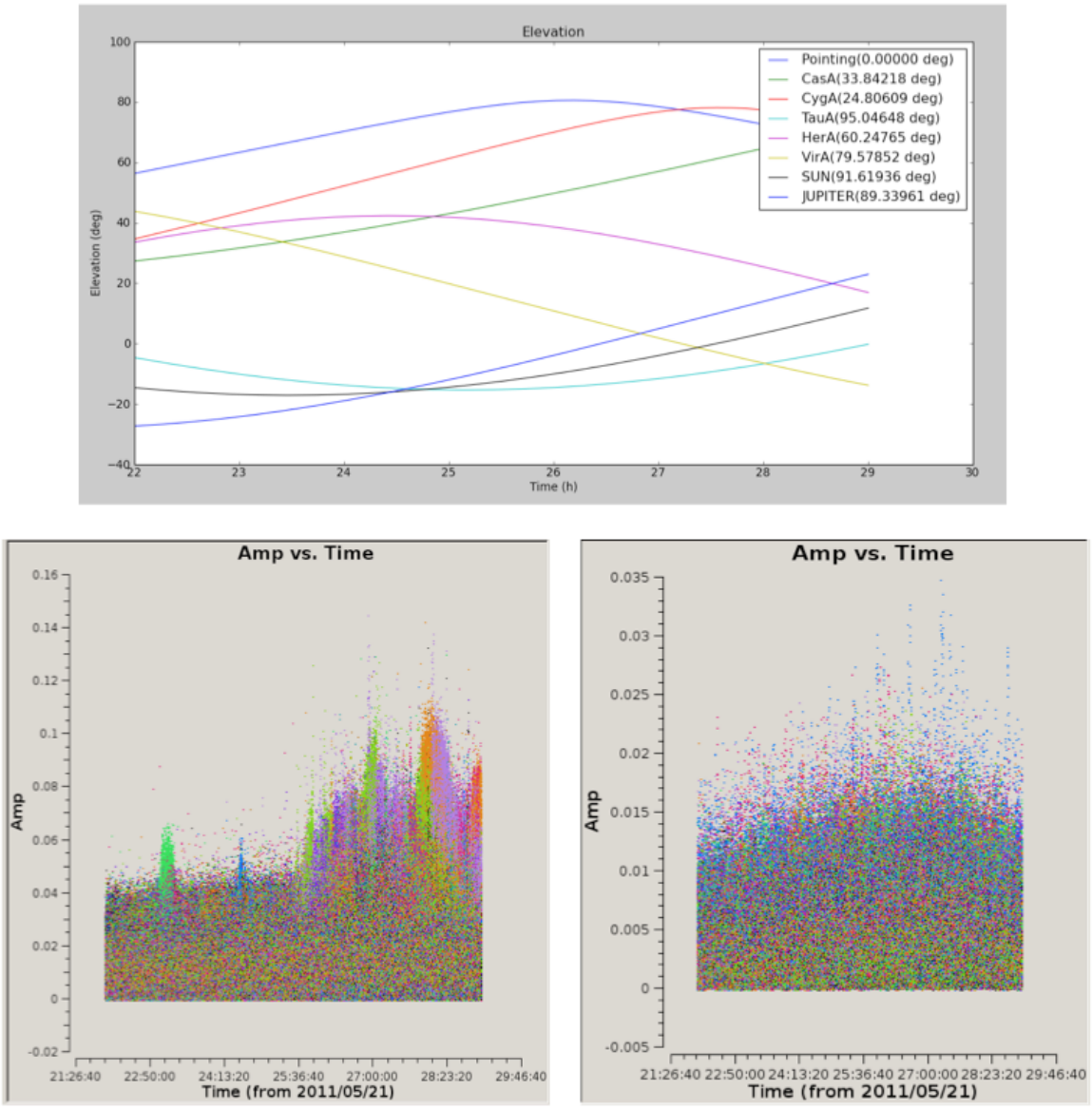


Fig. 4.1: **Top:** the elevation and angular distance of the A-team from the target field centered on B1835+62. This plot shows that during this observation CygA and CasA are very high in elevation and pretty close to our target (24 and 33 degrees, respectively). **Bottom left:** target visibilities 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-team sources is gone. This is particularly evident in the second part of the observation.

- Recently a new (smart)demixing scheme (designed by Reinout van Weeren) has been added to DPPP. Basically it works the same as the old demixer, but for each time window it estimates the data by evaluating a rough model of the A-team sources and target. Using those data it tests which sources have to be demixed, which baselines should be used, and if the target has to be ignored, solved, or deprojected. The LOFAR beam is taken into account in estimate, solve, and subtract. In this scheme one can specify:
  - A detailed model of the A-team sources to be used in the solve/subtract.
  - A rough model of the A-team sources to be used in the estimation. If not given, the detailed model is used.
  - A model of the target field which can be obtained using e.g. gsm.py.
  - The baselines to be used in the demixer. Note that the estimation step might exclude baselines for a given time window.
  - Various threshold and ratio values to test which sources, etc. to use.

Below, an example old demixer parset is given

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

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

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

```
demix.ignoretarget=true
```

Examples of demixing performance on real data are given in Fig. 4.1 shown above.

4.1.10 Combine measurement sets

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

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

The first line shows that a wildcarded MS name can be given, so all MeasurementSets with a name matching the pattern will be used. The data of all subbands are combined into a single subband and the meta frequency info will be updated accordingly. The second line means that the data in the CORRECTED_DATA column will be used and written as the DATA column in the output MS. The third line means that only the cross-correlations of the core and remote stations are selected and written into the output MS. Note this is different from flagging the baselines as shown in the preflagger example. Input selection means that non-matching baselines are fully omitted, while the preflagger

only flags baselines. Note that no further operations are needed, thus no steps are given. However, it is perfectly possible to include any other step. In this case one could use **count**.

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

4.2 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.

A description of all the parameters that can be used in DPPP can be found on the LOFAR wiki.

4.2.1 Input parameters (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. An example measurement set name is **L32667_SAP000_SB010_uv.MS**.

4.2.2 Output parameters (msout)

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

If you are working on CEP3, note that data should be written to **/data/scratch/<username>** (which may need creating initially). Output data should **not** be written back to the storage disks. Also, do not write output data to **/home/<username>**, as space is very limited on this disk.

You can let **DPPP** create a so-called VDS file, which tells other data processing programs (notably, **BBS** and **mwimager**) where the data live. You need a so-called cluster description file for this. These can be found in the LOFAR Cookbook GitHub repository. For the curious, the cluster description is a simple ASCII file that should be straightforward to understand).

4.2.3 Flagging

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

- If one correlation is flagged, all correlations will be flagged.
- The msin step flags data containing NaNs or infinite numbers.
- A **PreFlagger** step can be used to flag (or unflag) on time, baseline, elevation, azimuth, simple uv-distance, channel, frequency, amplitude, phase, real, and imaginary. Multiple values (or ranges) can be given for one or more of those keywords. A keyword matches if the data matches one of the values. The results of all given keywords are AND-ed. For example, only data matching given channels and baselines are flagged. Keywords can be grouped in a set making it a single (super) keyword. Such sets can be OR-ed or AND-ed. It makes it possible to flag, for example, channel 1-4 for baseline A and channel 34-36 for baseline B. Below it is explained in a bit more detail.
- A **UVWFlagger** step can be used to flag on UVW coordinates in meters and/or wavelengths. It is possible to base the UVW coordinates on a given phase center. If no phase center is given, the UVW coordinates in the input MS are used.
- **A MADFlagger step can be used to flag on the amplitudes of the data.**
  - It flags based on the median of the absolute difference of the amplitudes and the median of the amplitudes. It uses a running median with a box of the given size (number of channels and time slots). It is a rather expensive flagging method with usually good results.
  - It is possible to specify which correlations to use in the MADFlagger. Flagging on XX only, can save a factor 4 in performance.
- **An AOFlagger step can be used to flag using the AOFlagger.**
  - Usually it is faster than using **rficonsole** itself, because it does not reorder the data. Instead it flags in a user-defined time window. It is possible to specify a time window overlap to reduce possible edge effects. The larger the time window, the better the flagging results. It is possible to specify the time window by means of the amount of memory to be used.
  - The flagging strategy can be given in an rficonsole strategy file. Such a file should not contain a 'baseline iteration' command, because DPPP itself iterates over the baselines. Default strategy files exist for LBA and HBA observations (named LBAdefault and HBAdefault).
  - Note that the **AOFlagger** flags more data if there is a large percentage of zero data in the time window. This might happen if zero data is inserted by DPPP for missing time slots in a MeasurementSet or for missing subbands when concatenating the MeasurementSets of multiple subbands.
  - By default the QUALITY subtables containing flagging statistics are written. They can be inspected using **aoqplot**.

4.2.4 Averaging

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

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

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

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

4.3 Flag statistics

Several steps shows statistics during output about flagged data points.

- A flagger step shows the percentage of visibilities flagged by that step. It shows:
  - The percentages per baseline and per station.
  - The percentages per channel.
  - 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.
- An AOFlagger, PreFlagger, and UVWFlagger 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.

CHAPTER

FIVE

GAIN CALIBRATION WITH DPPP[1]

The most common gain calibration procedures can be performed with DPPP. Formerly BBS was the standard calibration tool, but for the supported calibration scenarios DPPP is at least 10 times faster. For calibration scenarios that are not (yet) supported in DPPP, it may be necessary to resort to BBS, see the chapter on BBS.

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

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

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

5.1 Calibration variants

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

Shape	Calibration type	Free parameters
$\mathbf{G}_p = \begin{pmatrix} A_{xx}^{(p)} e^{\phi_{xx}^{(p)}} & A_{xy}^{(p)} e^{\phi_{xy}^{(p)}} \\ A_{yx}^{(p)} e^{\phi_{yx}^{(p)}} & A_{yy}^{(p)} e^{\phi_{yy}^{(p)}} \end{pmatrix}$	fulljones	8
$\mathbf{G}_p = \begin{pmatrix} A_{xx}^{(p)} e^{\phi_{xx}^{(p)}} & 0 \\ 0 & A_{yy}^{(p)} e^{\phi_{yy}^{(p)}} \end{pmatrix}$	diagonal	4
$\mathbf{G}_p = \begin{pmatrix} e^{\phi_{xx}^{(p)}} & 0 \\ 0 & e^{\phi_{yy}^{(p)}} \end{pmatrix}$	phaseonly	2
$\mathbf{G}_p = \begin{pmatrix} e^{\phi^{(p)}} & 0 \\ 0 & e^{\phi^{(p)}} \end{pmatrix}$	scalarphase	1

5.2 Making a skymodel

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

[1] This section is maintained by Tammo Jan Dijkema.

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

The part **format='<'** is necessary to convince makesourcedb that the format is given by the first line in the file. You can use the program **showsourcedb** to verify the output sourcedb.

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

5.3 Calibration

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

```
msin=L123.MS
msout=
steps=[gaincal]
gaincal.sourcedb=L123.MS/sky
gaincal.parmdb=L123.MS/instrument
gaincal.caltype=phaseonly
gaincal.solint=2
gaincal.nchan=0
```

The part **solint=2** specifies that we only want one solution for every two time intervals. This can improve the signal to noise ratio - but one should have a physical argument that tells that the solutions do not change within the solution interval. The part **nchan=0** tells that one solution is computed that is assumed constant over the entire band. Setting it to e.g. **nchan=1** will compute a separate solution for each channel (again, at the cost of signal to noise).

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

The solution table can (and should!) be inspected with **parmdbplot.py**, see Section~ref{bbs:inspect-solutions}. Note that this calibration step does not yet change the visibilities. To perform complex operations on the solutions, like smoothing them, LoSoTo can be used, see Chapter~ref{losoto}.

5.4 Applying solutions

To apply the calibration solutions in DPPP, the step **applycal** can be used. The following parset applies the solutions that were obtained by gaincal. Note that currently. the solution table is only written at the very end of DPPP, so that it is not possible to solve and apply the solutions in the same run of DPPP.

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

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

5.5 Transferring solutions and the beam

When transferring solutions from a calibrator to a target, the sensitivity of the beam across the sky needs to be taken into account: the instrument does not have the same sensitivity at the position of the calibrator field as at the position of the target field. You can compensate for this by using a model for the LOFAR beam. Effectively, then instead of equation~ref{eq:dpppcal} the following equation is solved for $\mathbf{G}_p$:

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

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

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

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

5.6 Applying the beam

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

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

CHAPTER

SIX

LOSOTO: LOFAR SOLUTION TOOL[1]

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

LoSoTo can be accessed on CEP3 by

```
module load losoto
```

6.1 H5parm

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

6.1.1 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.

6.1.2 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

[1] This chapter is maintained by Francesco de Gasperin.

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. If no name is provided, then it is by default set to the solution-type name (amplitudes, phases, clock, tec...) plus again an increasing integer (e.g. amplitudes000, phase000...). Since soltab names are arbitrary the proper solution-type is stated in the **parmdb_type** attribute of the soltab node. Supported values are: amplitude, phase, scalarphase, rotation, clock, tec, tecscreen and phase_offset.

Soltabs are also just containers; inside each soltab there are several CArrays which are the real data holders. Typically there are a number of 1-dimensional CArrays storing the **axes** values (see Table 6.1) and two n-dimensional (where n is the number of axes) CArrays, "values" and "weights", which contain the solution values and the relative weights.

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

- **amplitudes**: time, freq, pol, dir, ant
- **phases**: time, freq, pol, dir, ant
- **clock**: time, ant, [pol]
- **tec**: time, ant, dir, [pol]
- **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.

Table 6.1: Default names and formats for axes values.

Axis name	Format	Example
time (s)	float64	[4.867e+09, 4.868e+09, 4.869e+09]
freq (Hz)	float64	[120e6,122e6,130e6...]
ant	string	[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 or 1]	[0,1,0.9,0.7,1,0]

6.1.3 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,)
```

(continues on next page)

(continued from previous page)

```
# 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
```

(continues on next page)

(continued from previous page)

```
 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)
```

6.1.4 H5parm benchmarks

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

This is a benchmark example:

```
INFO: H5parm filename = L99289-cal_SB081.h5
INFO: parmdb filename = L99289-cal_SB081.MS/instrument/
INFO: ### Read all frequencies for a pol/dir/station
INFO: PARMDB -- 1.9 s.
INFO: H5parm -- 0.28 s.
INFO: ### Read all times for a pol/dir/station
INFO: PARMDB -- 1.85 s.
INFO: H5parm -- 0.28 s.
INFO: ### Read all rotations for 1 station (slice in time)
INFO: PARMDB -- 1.94 s.
INFO: H5parm -- 0.3 s.
INFO: ### Read all rotations for all station (slice in time)
INFO: PARMDB -- 8.05 s.
INFO: H5parm -- 0.26 s.
INFO: ### Read all rotations for remote stations (slice in ant)
INFO: PARMDB -- 3.81 s.
INFO: H5parm -- 1.65 s.
INFO: ### Read all rotations for a dir/station and write them back
INFO: PARMDB -- 2.01 s.
INFO: H5parm -- 0.47 s.
INFO: ### Read and tabulate the whole file
INFO: parmdb -- 0.67 s.
INFO: H5parm -- 0.02 s.
```

6.2 LoSoTo

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

6.2.1 Tools

There are currently four tools shipped with LoSoTo:

- **parmdb_benchmark.py** provides a comparison between parmdb and H5parm for reading/writing
- **parmdb_collector.py** fetches parmdb tables from the cluster using a gds file
- **H5parm_importer.py** creates an h5parm file from an instrument table (parmdb) or a globaldb created by hand or with **parmdb_collector.py**
- **H5parm_merge.py** copies a solset from an H5parm files into another one
- **H5parm_exporter.py** exports an H5parm to a pre-existing parmdb

Details on how to use these tools are described below under section *Usage*.

6.2.2 Operations

These are the operations that LoSoTo can perform:

- **ABS**: takes the absolute value of the solutions (probably most meaningful for amplitudes).
- **CLIP**: clip all solutions n times above and $1/n$ times below the median value (only for amplitudes).
- **CLOCKTEC**: perform clock/tec separation (code maintained by Maaijke Mevius).
- **DUPLICATE**: duplicate a solution table.
- **FARADAY**: extract Faraday rotation solutions from the difference between RR and LL phase solutions.
- **FLAG**: iteratively remove a general trend from the solutions and then perform noisy region detection and outlier rejection. For phases this is done in real/imaginary space, for amplitude in log space.
- **FLAGEXTEND**: flag a point if surrounded by enough other flags in a chosen N-dimensional space
- **INTERP**: interpolate solutions along whatever (even multiple) axis. Typically one can interpolate in time and/or frequency. This operation can also simply rescale the solutions to match the median of the calibrator solution on a specific axis.
- **NORM**: normalize solutions of an axis to have a chosen average value.
- **PLOT**: plot solutions in 1D/2D plots.
- **PLOTTECSCREEN**: plot TEC screen (code maintained by David Rafferty).
- **RESET**: reset the solution values to 1 for all kind of solutions but for phases which are set to 0.
- **RESIDUAL**: subtract a table from another (e.g. remove TEC from phases).
- **REWEIGHT**: manually set weights to a specific values (can be used to hand-flag data, e.g. a bad antenna/timerange).
- **SMOOTH**: smooth solutions using a multidimensional running median. The n-dimensional surface generated by multiple axis (e.g. time and freq) can be smoothed in one operation using a different FWHM for each axis.
- **TECFIT**: fit TEC values per direction and station to phase solutions (code maintained by David Rafferty).
- **TECSCREEN**: fit TEC screens to TEC values (code maintained by David Rafferty).
- **EXAMPLE**: this is just an example operation aimed to help developing of new operations.

Beside these operations which require the activation through a *LoSoTo parset* file, one can call **losoto.py** with the "-i" option passing an H5parm as argument to obtain some information on it. Information on a specific subset of solsets can be obtained with "-i -f solset_name(s)". If "-i" is combined with "-v" (verbose), LoSoTo will take a bit more

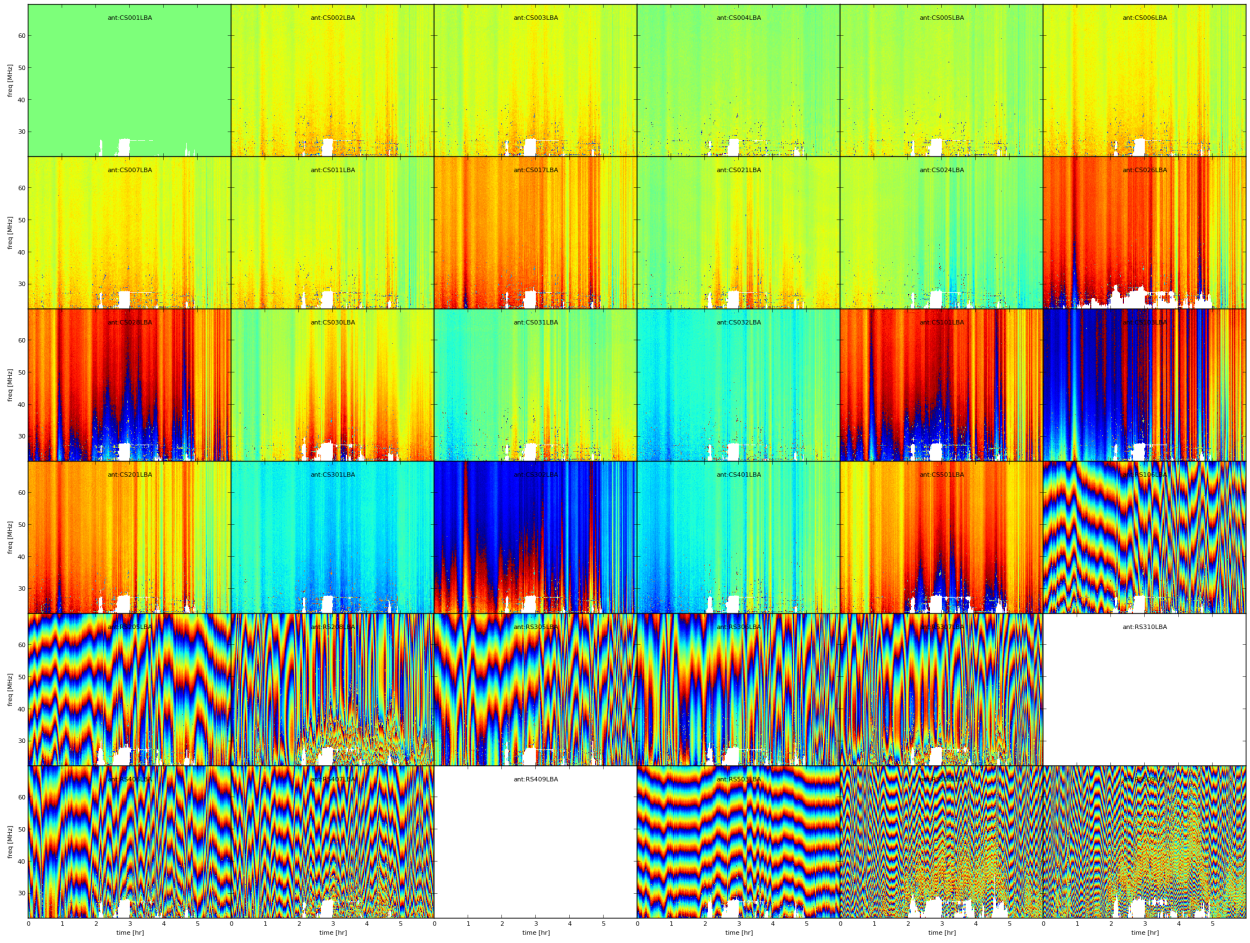


Fig. 6.1: Example phase solutions plotted using PLOT operation after the FLAG operation. White pixels are flagged data, every plot is an antenna, X-axis is time and Y-axis is frequency.

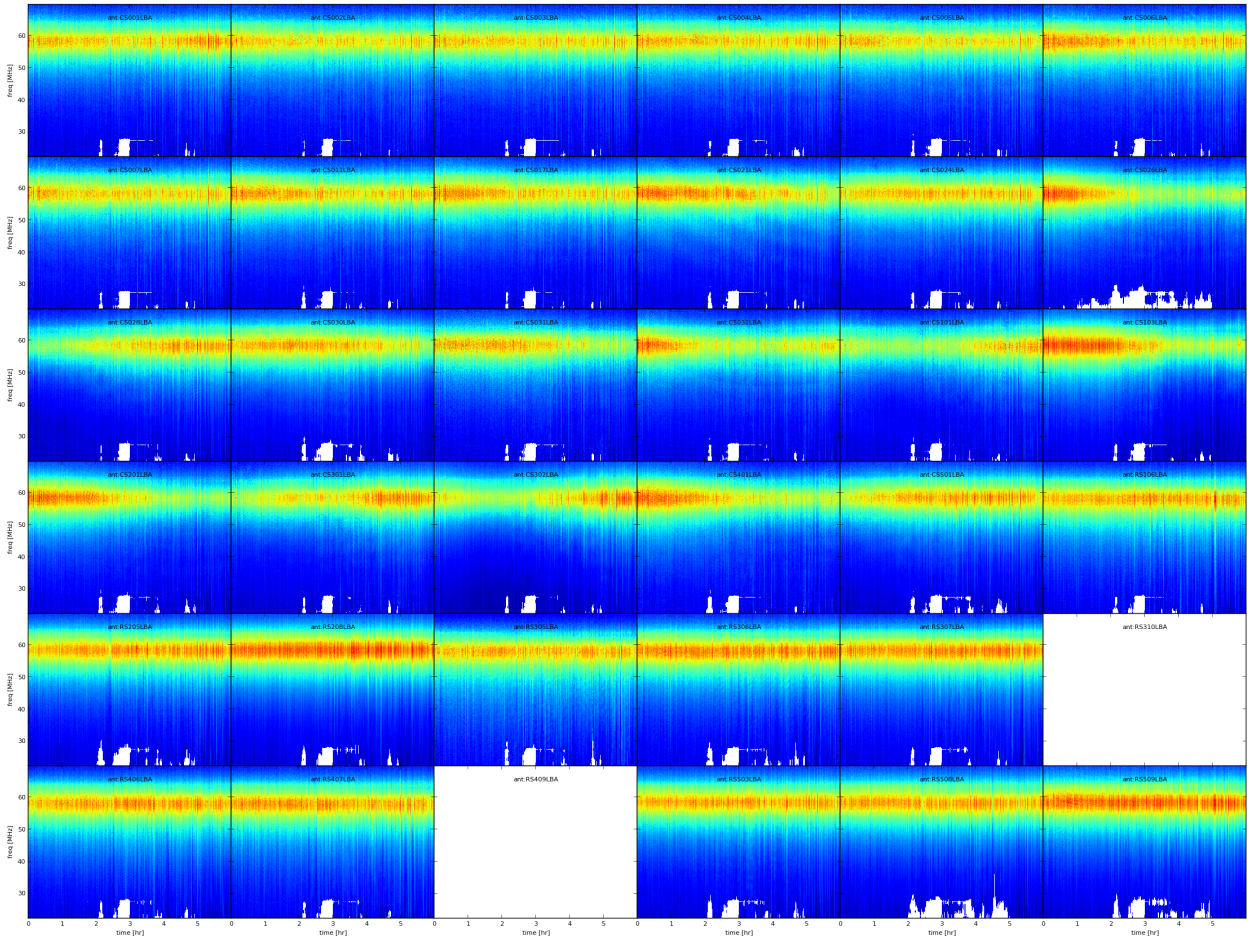


Fig. 6.2: Example amplitude solutions plotted using PLOT operation after the FLAG operation. White pixels are flagged data, every plot is an antenna, X-axis is time and Y-axis is frequency.

time and outputs also the percentage of flagged data and the values of all axes will be written in a file (e.g. file.h5-axes_values.txt). Using the "-d" options instead one can delete a chosen soltab (e.g. "losoto.py -d sol000/phase000 file.h5").

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

Summary of single.h5

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

Directions: 3C196
            pointing

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

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

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

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

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

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

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

6.2.3 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 = []
```

(continues on next page)

(continued from previous page)

```
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 6.2, the reader can find all the possible global values.

Table 6.2: Definition of global variables in LoSoTo parset.

Variable name	Format	Example	Comment
LoSoTo.Steps	list of steps	[flag, plot, smoothPhases, plot2]	sequence of steps names in order
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	antenna names	[CS001_HBA]	restrict to these antenna
LoSoTo.pol	polarization	[XX, YY]	restrict to these polarizations
LoSoTo.dir[3]	directions	[pointing, 3C196]	restrict to these pointing directions
LoSoTo.freq	frequencies	[30076599.12109375]	restrict to these frequencies
LoSoTo.time	times	[123456789.1234]	restrict to these times
LoSoTo.Ncpu[4]	integer	10	number of processes of spawn

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 under section *Operations*. All the global variables (except from the "steps" one) are also usable inside a step to change (override) the selection criteria for that specific step. Selection can be a string (interpreted as a regular expression), a list of values (exact match) or can have a min/max/step value which is activated using the axisName.minmax sintax (e.g. LoSoTo.freq.minmax = [30e6,1e9,5] to select 1 point every 5 from 30 MHz to 1 GHz). A list of step-specific parameters is given in https://github.com/revoltek/losoto.

6.3 Usage

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

- Collect the parmdb of calibrator and target:

[3] 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, and so on) has the direction named "pointing".

[4] Only some operations have been parallelized.

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

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

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

- Convert the set of parmdbs into an h5parm

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

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

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

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

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

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

One can create a h5parm also from a single SB

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

This command converts the "instrument" (parmdb) table inside the MS in a solset inside LXXXXX_3c295.h5. Note that given the definition of globaldb above, a single-SB measurement set is a perfect valid one.

- Run LoSoTo using e.g. the parset explained in *LoSoTo parset*

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

- Convert back the h5parm into parmdb:

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

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

6.4 Developing in LoSoTo

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

In the "tools" directory the user can find all the tools described in section *Tools* plus some other program. All these programs are stand-alone. **losoto.py** is the main program which calls all the operations (one per file) present in the

"operation" directory. It relays on the **h5parm.py** library which deals with reading and writing from an H5parm file and the **operations_lib.py** library which has some functions common to several operations.

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

6.5 Clock/TEC separation

In LoSoTo an algorithm is implemented with which it is possible to separate the BBS phase solutions into a instrumental delay (clock) and an ionospheric component (TEC, a measure of the differential ionospheric electron content). This is done using the difference in frequency dependence of both effects: The phase shift due to a delay error goes as ν, whereas ionospheric refraction gives to first order an phase shift proportional to $1/\nu$. Accordingly, one needs to have solutions over a large enough bandwidth to be able to do the separation[2].

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

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

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

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

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

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

[2] 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.

CHAPTER

SEVEN

THE WSCLEAN IMAGER[1]

WSClean is a command-line imager that is tailered for wide-field imaging. It supports several deconvolution methods, including multi-scale and multi-frequency deconvolution and an "RM-synthesis"-deconvolution mode. It also supports correcting images for the LOFAR beam.

To use WSClean on one of the CEP clusters, first include the WSClean environment:

```
module load Wsclean
```

A list of command line options can be acquired by running **wsclean** without parameters:

```
$ wsclean
WSClean version 2.4 (2017-05-28)
This software package is released under the GPL version 3.
...
```

An extensive manual for WSClean is available on-line, at wsclean homepage. It includes a chapter about LOFAR beam correction. WSClean has also been described in the following article: http://arxiv.org/abs/1407.1943.

[1] The author of this chapter is Andre Offringa.

CHAPTER

EIGHT

SOURCE DETECTION AND SKY MODEL MANIPULATION: PYBDSF AND LSMTOOL[1]

8.1 Source detection: PyBDSF

8.1.1 Introduction

PyBDSF (**Py**thon **B**lob **D**etection and **S**ource **F**inder) is a Python source-finding software package written by Niruj Mohan, Alexander Usov, and David Rafferty. PyBDSF can process FITS and CASA images and can output source lists in a variety of formats, including **makesourcedb** (BBS), FITS and ASCII formats. It can be used interactively in a casapy-like shell or in Python scripts. The full PyBDSF manual can be found here.

8.1.2 Setup

The latest version of PyBDSF is installed on the CEP3 cluster. To initialize your environment for PyBDSF, run:

```
module load pybdsf
```

After initialization, the interactive PyBDSF shell can be started with the command **pybdsf** and PyBDSF can be imported into Python scripts with the command **import bdsf**.

8.1.3 Tutorials

This section gives examples of using PyBDSF 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 PyBDSF 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.

8.1.4 Simple Example

A simple example of using PyBDSF 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.

[1] This section is maintained by David Rafferty).

```
BDSF [1]: inp process_image
BDSF [2]: filename = 'sb48.fits'
BDSF [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

BDSF [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 Fig. 8.1.

In the plot window, one can zoom in, save the plot to a file, etc. The list of best-fit Gaussians found by PyBDSF 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.pybdsf.srl.fits'
```

The output Gaussian or source list contains source positions, fluxes, etc. BBS patches are also supported.

8.1.5 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 PyBDSF from fitting Gaussians to such artifacts. An example of running PyBDSF with the default parameters on such an image is shown in Fig. 8.2. 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

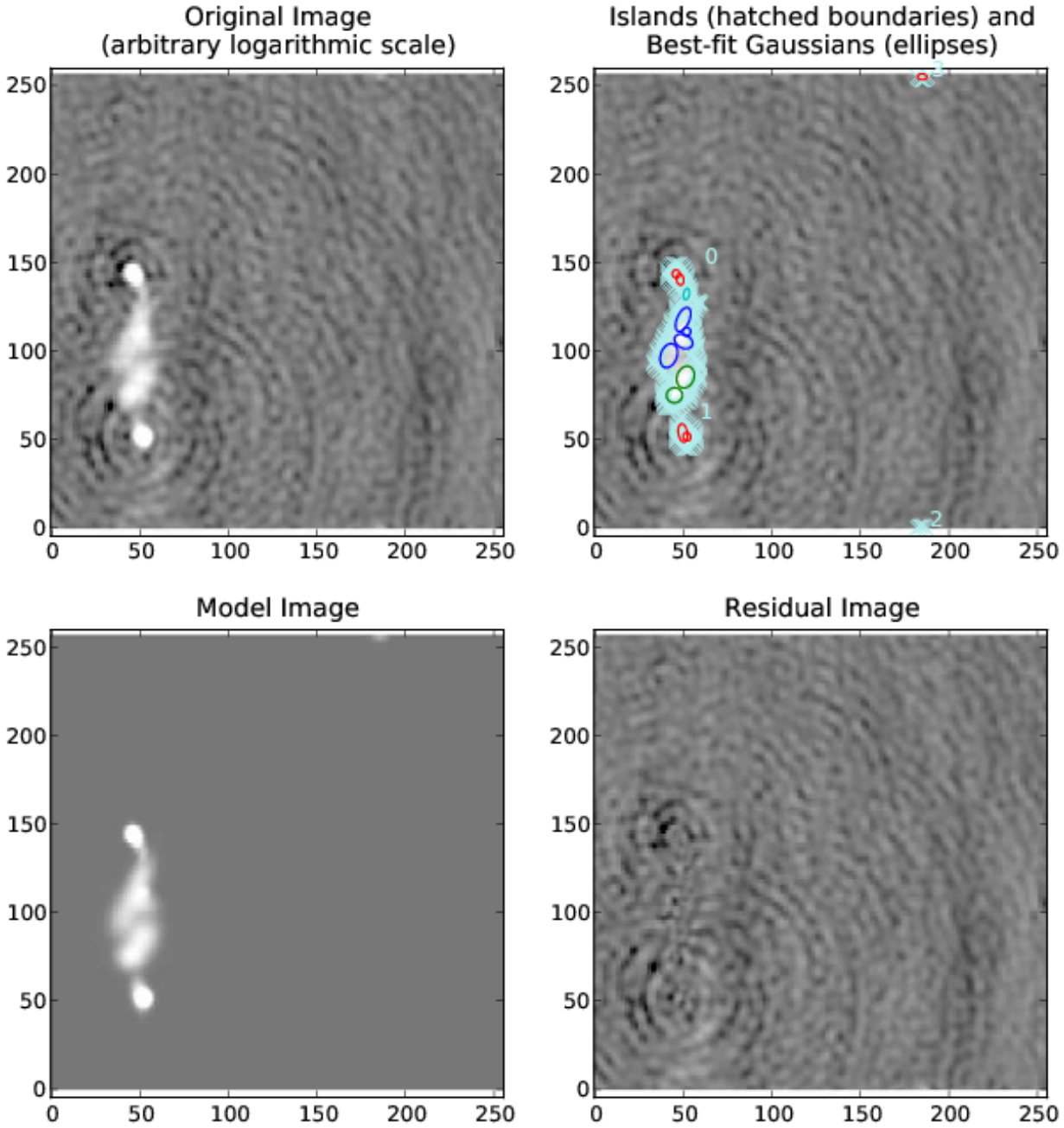


Fig. 8.1: Output of **show_fit**, showing the original image with and without sources, the model image, and the residual (original minus model) image. Boundaries of the islands of emission found by PyBDSF are shown in light blue. The fitted Gaussians are shown for each island as ellipses (the sizes of 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.

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 PyBDSF 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 Fig. 8.4 for a summary of the results of this call.

8.1.6 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 PyBDSF 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 PyBDSF finds and fits islands of emission properly. An example analysis of an image with significant extended emission is shown in Fig. 8.6.

8.1.7 Usage in python scripts

PyBDSF 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 PyBDSF 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 PyBDSF 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, PyBDSF 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 PyBDSF shell are available for examining the fit and writing out the source list, residual image, etc. These tasks are methods of the Image object returned by **bdsm.process_image()** and are described below:

- **img.show_fit()**: This method shows a quick summary of the fit by plotting the input image with the islands and Gaussians found, along with the model and residual images.
- **img.export_image()**: Write an internally derived image (e.g., the model image) to a FITS file.
- **img.write_catalog()**: This method writes the Gaussian or source list to a file.

The input parameters to each of these tasks are the same as those available in the interactive shell. For more details and scripting examples, see PyBDSF documentation.

8.2 Sky model manipulation: LSMTool

8.2.1 Introduction

LSMTool is a Python package which allows for the manipulation of sky models in the **makesourcedb** format (used by BBS and DPPP). Such models include those output by PyBDSF, those made by **gsm.py**, and CASA clean-component models (after conversion with **casapy2bbs.py**). The full LSMTool manual is located at http://www.astron.nl/citt/lsmtool.

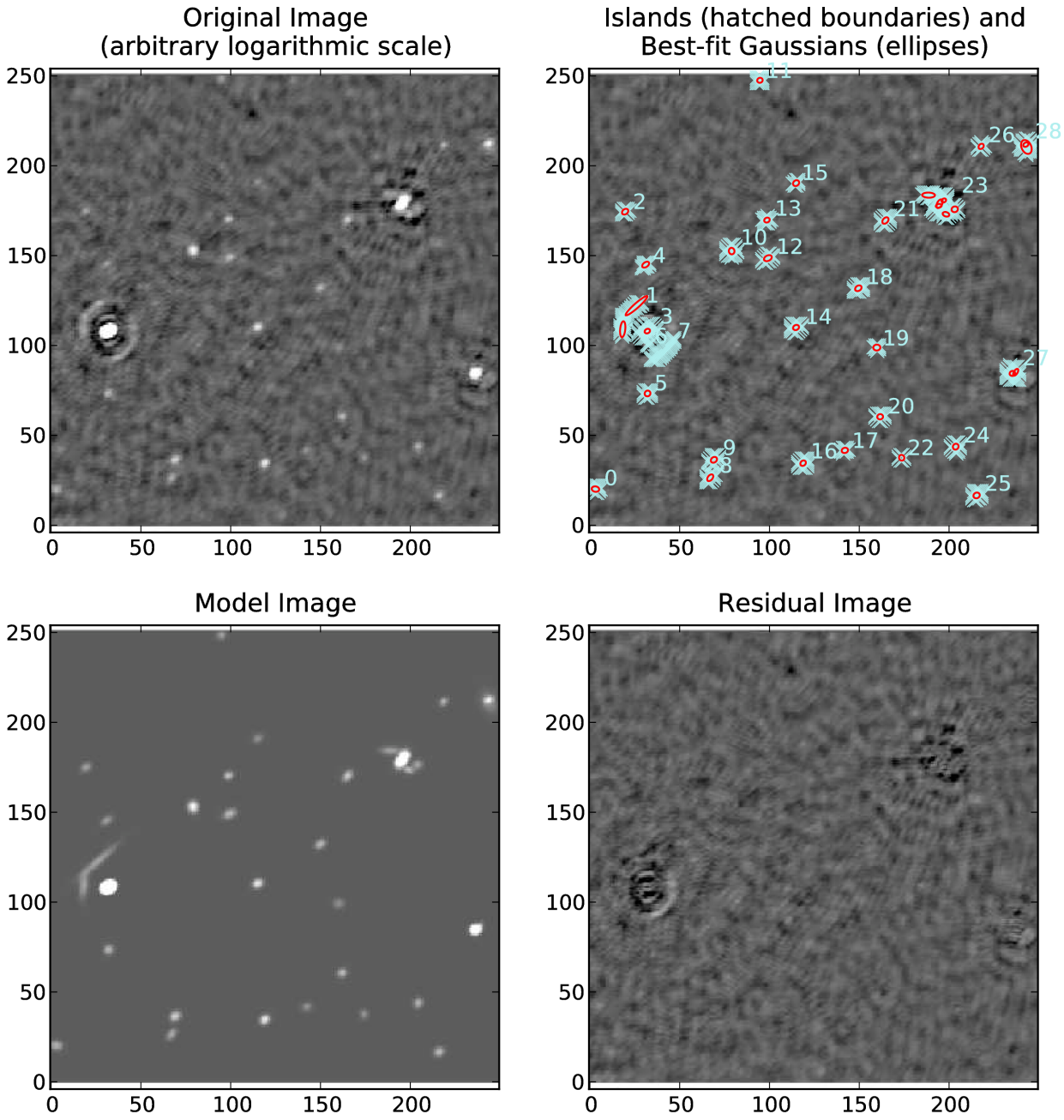


Fig. 8.2: Example fit with default parameters of an image with strong artifacts around bright sources. A number of artifacts near the bright sources are picked up as sources.

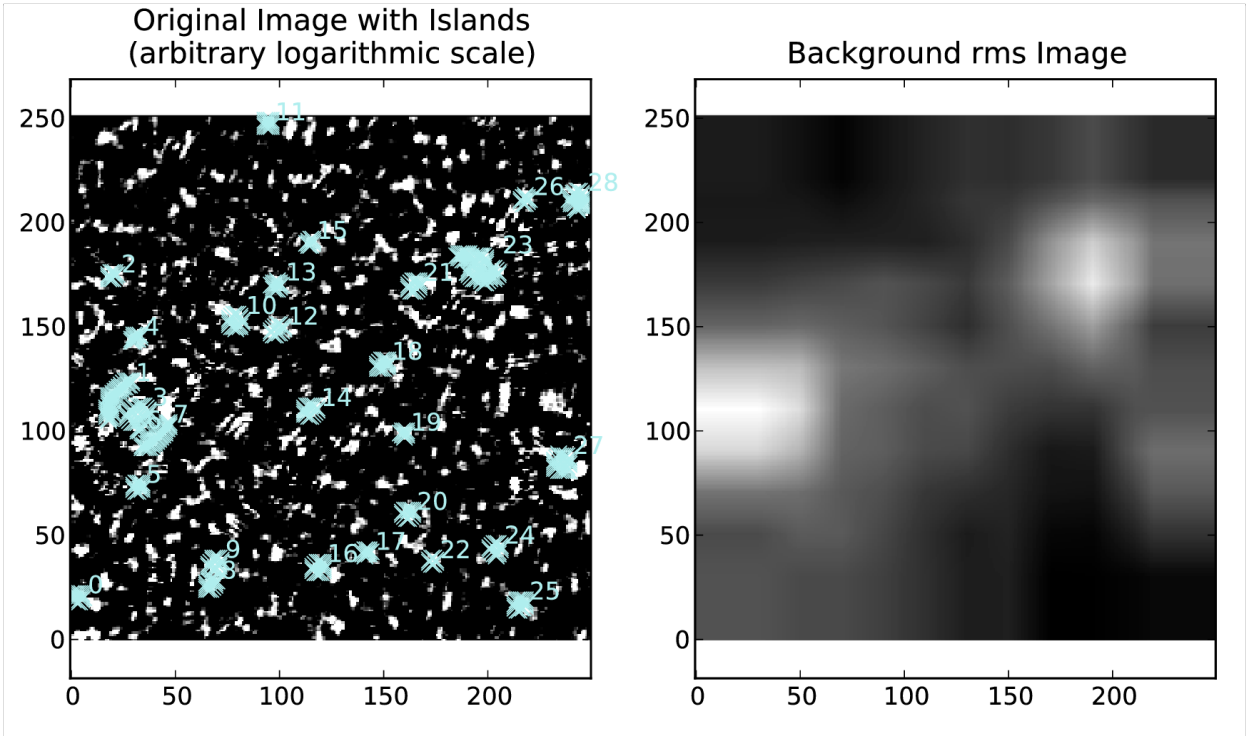


Fig. 8.3: The background rms map for the same region (produced using **show_fit**) as in Fig. 8.2: the rms varies fairly slowly across the image, whereas ideally it would increase more strongly near the bright sources (reflecting the increased rms in those regions due to the artifacts).

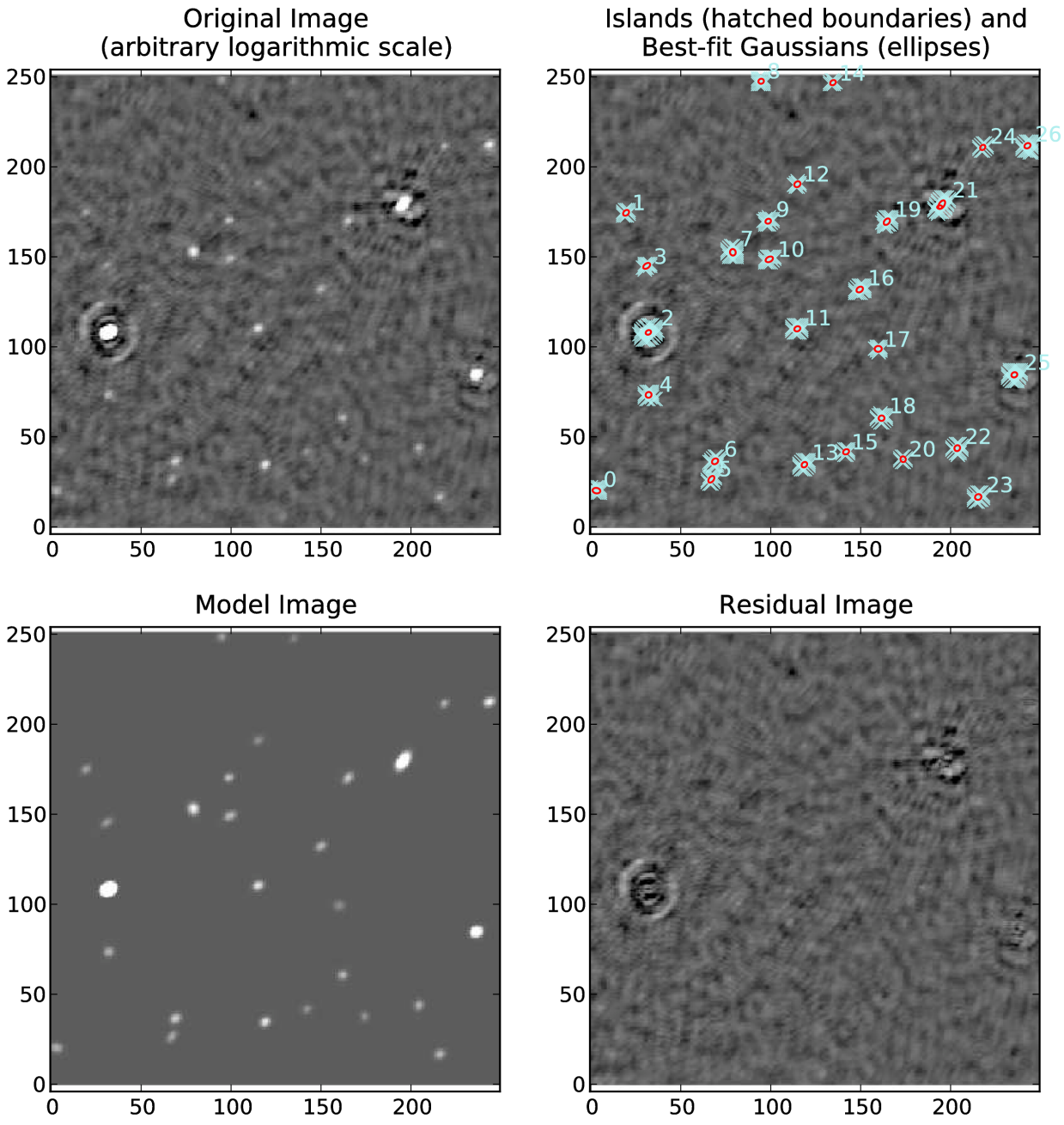


Fig. 8.4: Same as Fig. 8.2 but with **rms_box=(20,10)**.

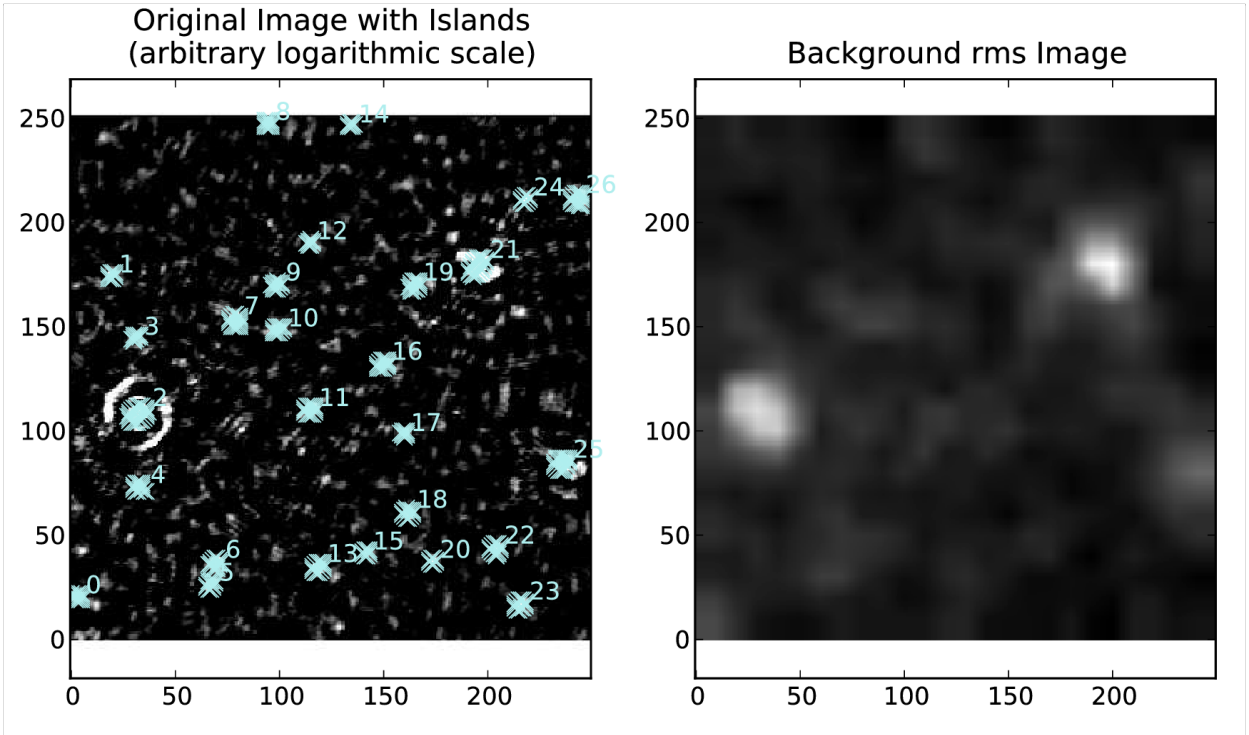


Fig. 8.5: The rms map, corresponding to the fit shown in Fig. 8.4 varies on scales similar to that of the regions affected by the artifacts, and both bright and faint sources are recovered properly.

8.2.2 Setup

To initialize your environment for LSMTool, users on the CEP3 cluster should run the following command:

```
module load lsmtool
```

8.2.3 Tutorials

This section gives examples of using LSMTool to select sources and group them into patches.

Filter and Group a Sky Model

As with many of the LOFAR tools (e.g., DPPP), LSMTool can be run with a parset that specifies the operations to perform and their parameters. Below is an example parset that filters on the Stokes I flux density (selecting only those sources with flux densities above 1 mJy), adds a source to the sky model, and then groups the sources into patches (so that each patch has a total flux density of around 50 Jy):

```
LSMTool.Steps = [selectbright, addsrc, grp, setpos]

 # Select only sources above 1 mJy
 LSMTool.Steps.selectbright.Operation = SELECT
 LSMTool.Steps.selectbright.FilterExpression = I > 1.0 mJy

 # Add a source
```

(continues on next page)

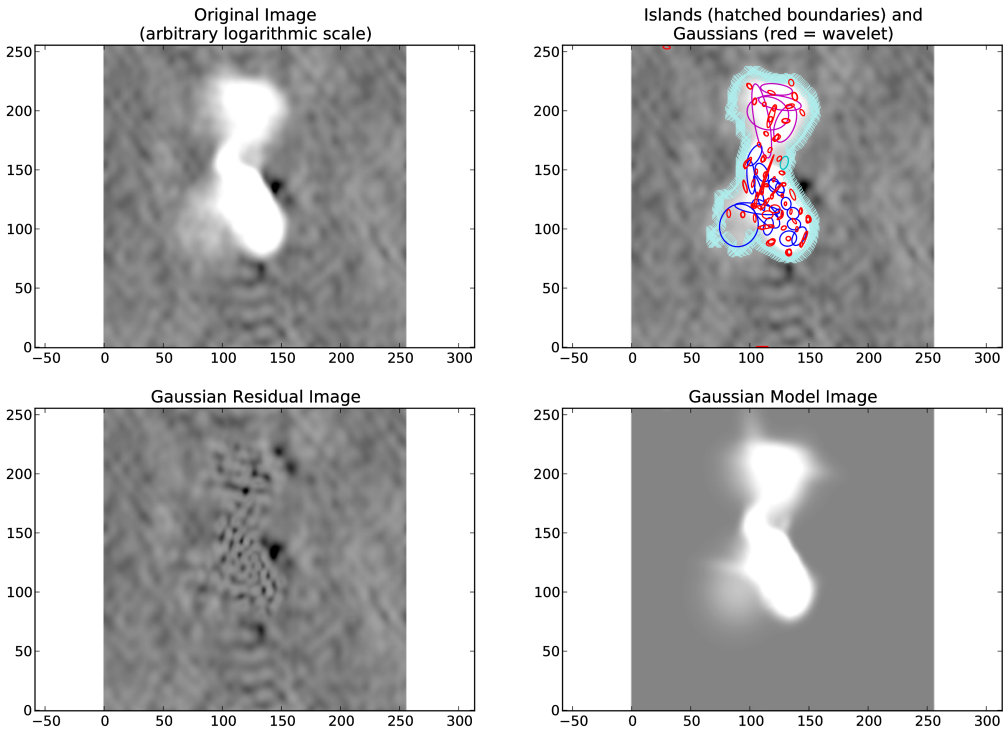


Fig. 8.6: Example fit of an image of Hydra A with **rms_map=False**, **mean_map='zero'**, and **atrous_do=True**. The values of **thresh_isl** and **thresh_pix** were adjusted before fitting (by setting **interactive=True**) to obtain an island that enclosed all significant emission.

(continued from previous page)

```
LSMTool.Steps.addsrc.Operation = ADD
LSMTool.Steps.addsrc.Name = new_source
LSMTool.Steps.addsrc.Type = POINT
LSMTool.Steps.addsrc.Ra = 277.4232
LSMTool.Steps.addsrc.Dec = 48.3689
LSMTool.Steps.addsrc.I = 0.69

# Group using tessellation to a target flux of 50 Jy
LSMTool.Steps.grp.Operation = GROUP
LSMTool.Steps.grp.Algorithm = tessellate
LSMTool.Steps.grp.TargetFlux = 50.0 Jy
LSMTool.Steps.grp.Method = mid

# Set the patch positions to their midpoint and write final skymodel
LSMTool.Steps.setpos.Operation = SETPATCHPOSITIONS
LSMTool.Steps.setpos.Method = mid
LSMTool.Steps.setpos.OutFile = grouped.sky
```

In the first line of this parset the step names are defined. Steps are applied sequentially, in the same order defined in the list of steps. See the full manual for a list of steps and their allowed parameters.

LSMTool can also be used interactively (in IPython, for example) or in Python scripts without the need for a parset, which is generally more convenient. To use LSMTool in a Python script or interpreter, import it as follows:

```
>>> import lsmtool
```

A sky model can then be loaded with, e.g.:

```
>>> s = lsmtool.load('skymodel.sky')
```

The following commands duplicate the steps done using the parset above:

```
# Select only sources above 1 mJy
 >>> s.select('I > 1.0 mJy')

 # Add a source
 >>> s.add({'Name':'new_source', 'Type':'POINT', 'Ra':277.4232, 'Dec':48.3689, 'I':0.
↪69})

 # Group using tessellation to a target flux of 50 Jy
 >>> s.group(algorithm='tesselate', targetFlux='10.0 Jy')

 # Set the patch positions to their midpoint and write final skymodel
 >>> s.setPatchPositions(method='mid')
 >>> s.write('grouped.sky')
```

Select sources within a given region

In this example, LSMTool is used to select only those sources within a radius of 1 degree of a given position (RA = 123.2123 deg, Dec = +34.3212 deg):

```
# Get the distance of each source to the given position (here specified as
# RA, Dec in degrees; one can also specify the postion in makesourcedb format)
>>> dist = s.getDistance(123.2123, 34.3212)
```

(continues on next page)

(continued from previous page)

```
# Remove sources with distances of more than 1 degree
>>> s.remove(dist > 1.0)
```

Find Sources below a Given Size in a Clean-component Sky Model

It can be difficult to identify sources in a clean-component sky model (such as one made by **casapy2bbs.py**), as there is no explicit connection between the components. LSMTool provides a grouping algorithm (named **threshold**) that can be used to find sources in such a sky model. For example, the following lines will identify sources and select only those smaller than 3 arcmin:

```
# Use thresholding to group clean components into patches by smoothing the
# clean-component model with a Gaussian of 60 arcsec FWHM
>>> s.group('threshold', FWHM='60.0 arcsec')

# Get the sizes (in arcmin) of the patches, weighted by the clean-component
# fluxes
>>> sizes = s.getPatchSizes(units='arcmin', weight=True)

# Select sources with sizes below 3 arcmin (the "aggregate" parameter means
# to select on the patch sizes instead of the individual source sizes)
>>> s.select(sizes < 3.0, aggregate=True)
```

CHAPTER

NINE

FACTOR: FACET CALIBRATION FOR LOFAR[1]

Factor is a Python package that allows for the production of wide-field HBA LOFAR images using the facet calibration scheme described in van Weeren et al. (2016) to correct for direction-dependent effects (DDEs). To initialize your environment for Factor, users on CEP3 should run the following command:

```
source ~rafferty/init_factor
```

If you want to install Factor yourself, follow the instructions in the **README.md** file on the Factor GitHub repository. Detailed documentation for Factor can be found here.

9.1 Data preparation: Pre-facet calibration pipeline[2]

The input data to Factor must have the average amplitude scale set and average clock offsets removed. Furthermore, the LOFAR beam towards the phase center should be removed. The data should be concatenated in frequency to bands of about 2 MHz bandwidth (so about 10–12~subbands). All bands (= input measurement sets) need to have the same number of frequency channelsfootnote{Factor does lots of averaging by different amounts to keep the data-size and computing time within limits. If the input files have different numbers of channels then finding valid averaging steps for all files gets problematic.}. Also the number of channels should have many divisors to make averaging to different scales easy. The data should then undergo direction-independent, phase-only self calibration, and the resulting solutions must be provided to Factor.

The pre-facet calibration pipelines are intended to prepare the observed data so that they can be used in Factor. The pipelines are defined as parsets for the genericpipeline, that first calibrate the calibrator, then transfer the gain amplitudes, clock delays and phase offsets to the target data, do a direction-independent phase calibration of the target, and finally image and subtract sources.

Please have a look at the documentation for the genericpipeline here. You should be reasonably familiar with setting up and running a genericpipeline before running this pipeline parset.

9.1.1 Download and set-up

The pipeline parsets and associated scripts can be found on the Prefactor GitHub repository. These pipelines require as input data that has been pre-processed by the ASTRON averaging pipeline. They can work both with observations that had a second beam on a calibrator and with interleaved observations in which the calibrator was observed just before and after the target. No demixing is done, but A-Team clipping is performed for the target data[3].

To run the genericpipeline you need a correctly set up configuration file for the genericpipeline. Instruction for setting up the configuration file can be found online. In addition to the usual settings you need to ensure that the plugin scripts

[1] This chapter is maintained by David Rafferty.
[2] This section was written by Andreas Horneffer with a lot of help from Tim Shimwell.
[3] Versions of the pipeline (parset) which also do demixing are being planned.

are found, for that you need to add the pre-facet calibration directory to the **recipe_directories** in the pipeline parset (so that the **plugins** directory is a subdirectory of one of the **recipe_directories**).

9.1.2 Overview and examples

The most recent overview of the pre-facet pipeline functionality as well as several use cases and troubleshooting information can be found on the pre-facet pipeline GitHub wiki.

After the pipeline was run successfully, the resulting measurement sets are moved to the specified directory. The instrument table of the direction-independent, phase-only calibration are inside the measurement sets with the names **instrument_directioninpendent**. These measurement sets are the input to the Initial-Subtract pipeline, which should be run next to do the imaging and source subtraction. Once the Initial-Subtract pipeline has finished, the data are ready to be processed by Factor.

9.2 Factor tutorials

This section includes examples of setting up and using Factor. Setting up a Factor run mainly involves creating a parset that defines the reduction parameters. For many of these parameters, the default values will work well, and hence their values do not need to be explicitly specified in the parset. Therefore, only the parameters relevant to each example are described below. A detailed description of all parameters can be found on the online.

9.2.1 Quick-start

Below are the basic steps in a typical Factor run on a single machine (e.g., a single CEP3 node).

- Collect the input bands and Initial-Subtract sky models (produced as described in sub-section ref{factor:pre-facet-pipeline}) in a single directory. The MS names must end in ".MS" or ".ms":

```
[/data/factor_input]$ ls
L99090_SB041_uv.dppp_122298A79t_119g.pre-cal.ms
L99090_SB041_uv.dppp_122298A79t_119g.pre-cal.wsclean_low2-model.merge
L99090_SB041_uv.dppp_122298A79t_121g.pre-cal.ms
L99090_SB041_uv.dppp_122298A79t_121g.pre-cal.wsclean_low2-model.merge
L99090_SB041_uv.dppp_122298A79t_123g.pre-cal.ms
L99090_SB041_uv.dppp_122298A79t_123g.pre-cal.wsclean_low2-model.merge
L99090_SB041_uv.dppp_122298A79t_125g.pre-cal.ms
L99090_SB041_uv.dppp_122298A79t_125g.pre-cal.wsclean_low2-model.merge
```

- Edit the Factor parset (see online documentation for a detailed description) to fit your reduction and computing resources. For example, the parset for a single machine could be:

```
[/data]$ more factor.parset
[global]
dir_working = /data/factor_output
dir_ms = /data/factor_input

[directions]
flux_min_jy = 0.3
size_max_arcmin = 1.0
separation_max_arcmin = 9.0
max_num = 40
```

- Run the reduction:

```
[/data]$ runfactor factor.parset
```

9.2.2 Selecting the DDE calibrators and facets

Before facet calibration can be done, suitable DDE calibrators must be selected. Generally, a suitable calibrator is a bright, compact source (or a group of such sources)[4]. The DDE calibrators can be provided to Factor in a file (see online documentation for details) or can be selected internally. This example shows how to set the parameters needed for internal selection.

Three parameters in the **[directions]** section of the parset are most important for the internal calibrator selection: **flux_min_Jy**, **size_max_arcmin**, and **separation_max_arcmin**. These parameters set the minimum flux density (in the highest-frequency band) and maximum size of the calibrators and the maximum allowed separation that two sources may have to be grouped together as a single calibrator (see online documentation for details). The following are good values to use as a starting point:

```
[directions]
flux_min_Jy = 0.3
size_max_arcmin = 1.0
separation_max_arcmin = 9.0
```

It is recommended that you set **interactive = True** under **[global]** while experimenting with the calibrator selection, as Factor will select the calibrators and then pause to allow you to verify them. Now, run Factor (with **runfactor factor.parset**) and once it asks for verification, open one of the ***high2**} images generated during the Initial Subtract pipeline in ds9 or casaviewer. Then load the region files located in the **dir_working/regions/**} directory (where **dir_working** is the working directory defined in your Factor parset). Fig. 9.1 shows an example of such an image with the regions overlaid.

Check that none of the calibrator image regions are overlapping significantly. If they are, try adjusting the above parameters to either group the calibrators together (so that they form a single calibrator) or remove one of them. Also check that no facet is larger than a degree or so. If such large facets are present, try adjusting the above parameters to obtain more calibrators (e.g., by lowering **flux_min_Jy**). If the large facets are on the outer edge of the faceted region, you should also consider reducing **faceting_radius_deg** to limit their size.

Once the calibrator and facet selection is acceptable, Factor will make a directions file named **factor_directions.txt** in **dir_working**. This file can now be edited by hand if desired (e.g., to specify a sky model or clean mask for a calibrator).

9.2.3 Imaging a target source

If you are interested only in a single target source, you may want to process just the brightest sources in the field and those nearest the target. Fig. 9.2 shows the layout of an example field, with the target source (a galaxy cluster) indicated by the blue circle in facet_patch_468. Note that the faceting radius has been reduced substantially to speed up processing (since facets outside of this radius are small patches that run much more quickly than full facets). A few facets (e.g., facet_patch_566 and facet_patch_273) contain bright sources with artifacts that affect the target. Additionally, although they contain fainter sources, artifacts from the neighboring facets (e.g., facet_patch_424) can also affect the target significantly. Therefore, a good strategy is to process these facets before processing the target. Note that the lowest possible noise would be obtained by processing the entire field but would require much longer to complete.

We can limit the processing to the sources above by moving them to the top of the directions file (named **dir_working/factor_directions.txt** if generated automatically) and setting **ndir_process** under **[directions]** in the

[4] Ideally calibrators should be selected that provide roughly uniform coverage of the field with separations less than the typical distance over which the direction-dependent effects vary strongly (this distance depends on the severity of the ionospheric conditions, but is usually less than a degree or so). However, the achievable uniformity of coverage will vary from field to field, depending on the specific distribution of sources.

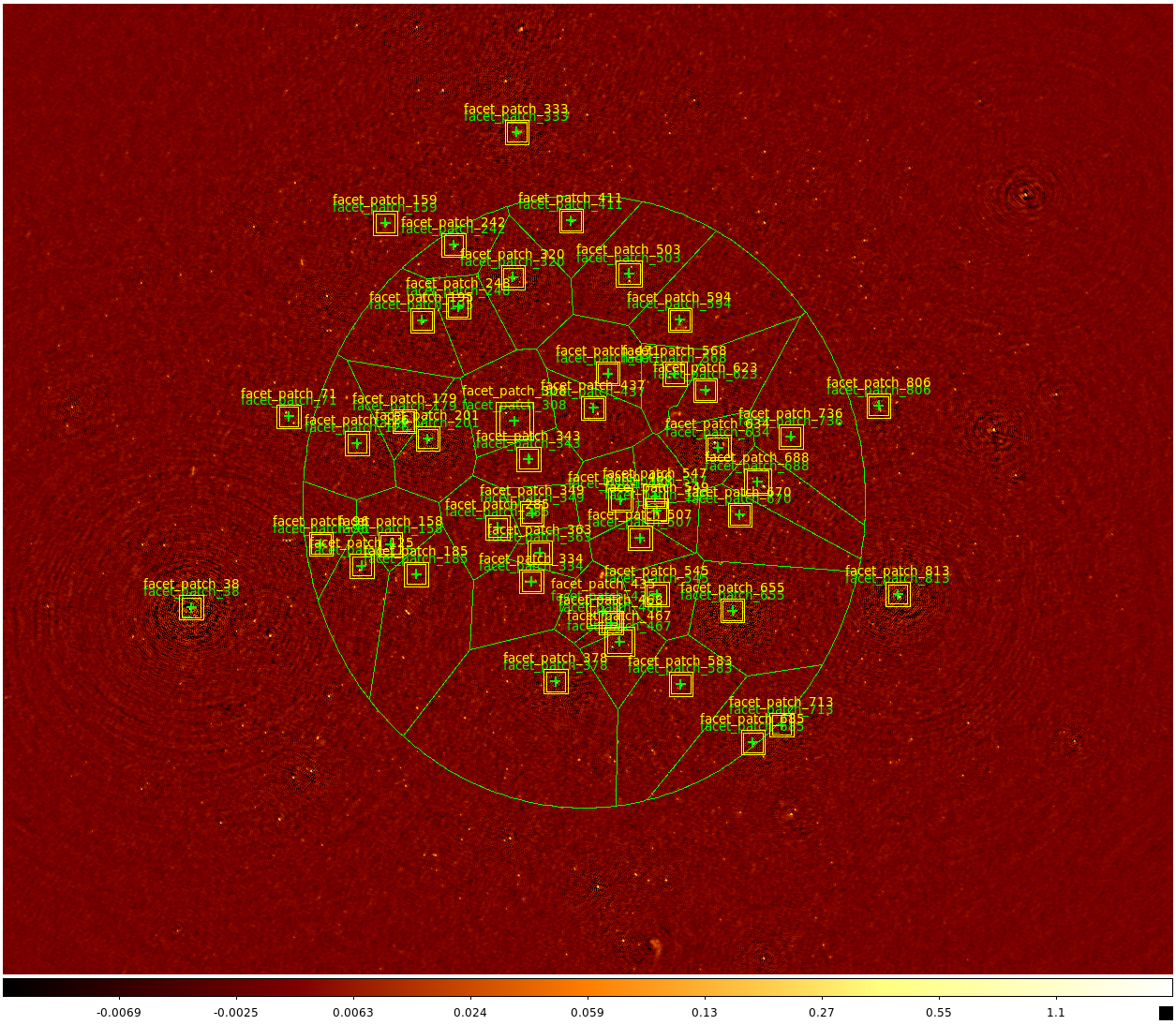


Fig. 9.1: Example of internally generated calibrators and facets. Yellow boxes show the regions to be imaged during self calibration (the inner box shows the region over which sources are added back to the empty datasets) and green regions show the facets. Each direction is labeled twice: once for the calibrator region and once for the facet region. Note the elliptical region that encompasses the faceted area. This region is controlled by the **faceting_radius_deg** parameter under [directions] in the parset: inside this radius (adjusted for the mean primary beam shape), full facets are used; outside this radius, only small patches are used that are much faster to process. This radius can be adjusted to limit the size of the outer facets and the region to be imaged.

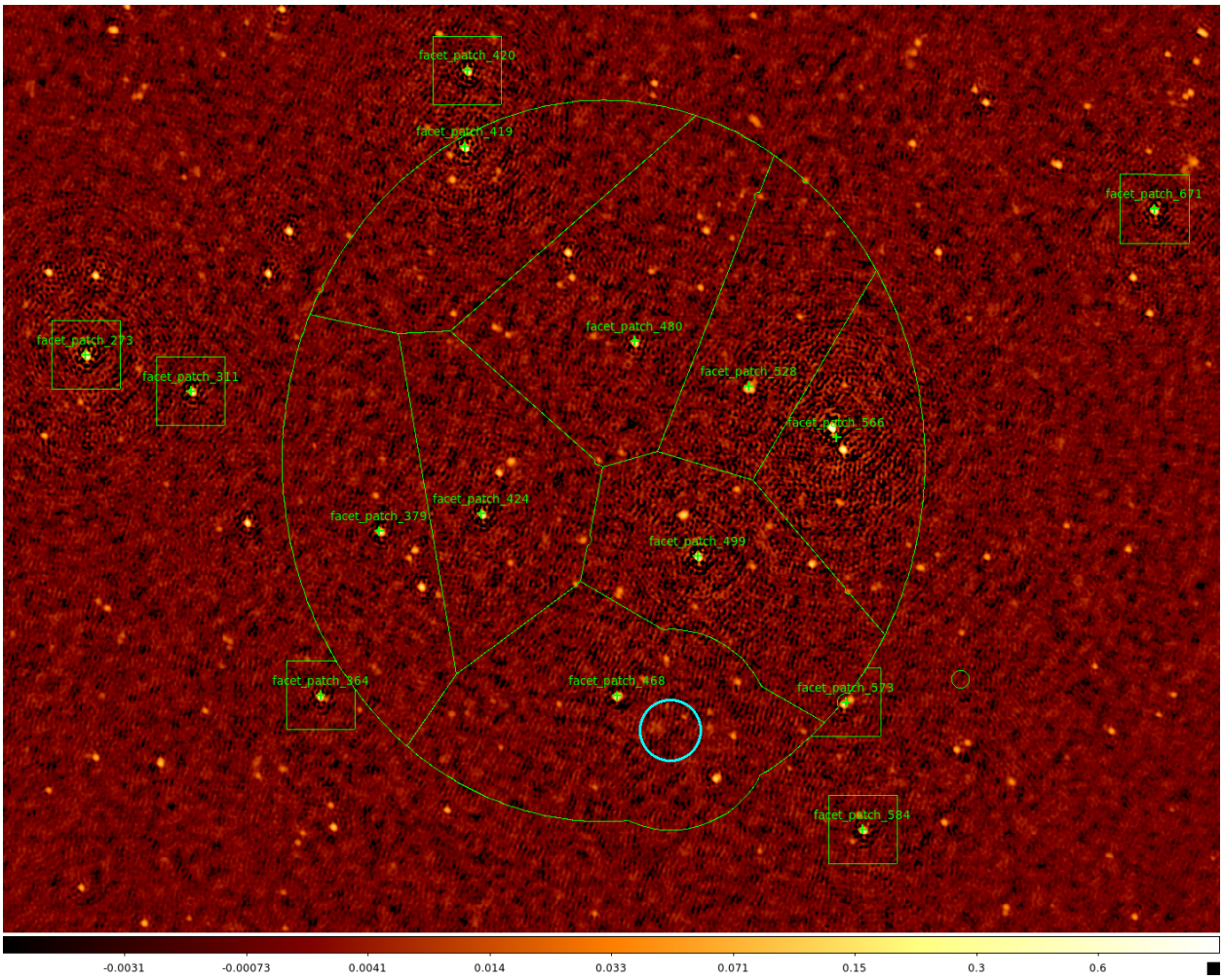


Fig. 9.2: Example of the field layout for a case in which an image is desired only for a single target source (indicated by the blue circle). A region of avoidance was also specified (with a larger radius than the blue circle) that resulted in the curved bulges to the boundary to the north and south of the target.

parset to the number of sources we wish to be processed (including the facet with the target source). Note that you should typically have many more calibrators in the directions file than directions to be processed, as the facets shapes and sizes are determined from all calibrators together and too few can result in very large facets. An example directions file is shown below (note that only the first two columns are shown). In this case, the target facet (facet_patch_468) will be the seventh direction to be processed, so we set **ndir_process = 7** under **[directions]** in the parset. Lastly, set **image_target_only = True** under **[imaging]** so that only the target facet is imaged.

```
# name position ...

facet_patch_566 14h29m38.5962s,44d59m14.0061s ...
facet_patch_499 14h31m36.5221s,44d41m39.6252s ...
facet_patch_424 14h34m38.5876s,44d48m00.415ss ...
facet_patch_379 14h36m04.2603s,44d45m16.2451s ...
facet_patch_273 14h40m14.6989s,45d10m53.6821s ...
facet_patch_573 14h29m33.1473s,44d19m43.2515s ...
facet_patch_468 14h32m44.3536s,44d20m50.7736s ...
......
facet_patch_550 14h30m03.9315s,46d51m44.6965s ...
facet_patch_480 14h32m28.9389s,45d13m49.3407s ...
facet_patch_778 14h20m03.7803s,44d09m40.3293s ...
```

Additionally, if the target source is diffuse, Factor may not be able to adjust the facet boundaries properly to ensure that it lies entirely within a single facet. To ensure that no facet boundary passes through the target source, you can define the position and the size of an avoidance region under the **[directions]** section of the parset. The **target_ra**, **target_dec**, and **target_radius_arcmin** parameters determine this avoidance region. An example of such a region for the target above is given below and the resulting facets are shown in Fig. 9.1. Note that the boundaries of the target facet (facet_patch_468) are curved to avoid this region.

```
[directions]
target_ra = 14h31m59.7s
target_dec = +44d15m48s
target_radius_arcmin = 15
```

9.2.4 Peeling bright sources

In many observations there will be one or more strong sources that lie far outside the FWHM of the primary beam but nevertheless cause significant artifacts inside it. However, because of strong variations in the beam with frequency at these locations, self calibration does not generally work well. Therefore, Factor includes the option to peel these outlier sources. A high-resolution sky model (in makesourcedb format) is needed for this peeling. Factor includes a number of such sky models for well-known calibrators (see online documentation for a list). If an appropriate sky model is not available inside Factor, you must obtain one and specify it in the directions file. To enable peeling, the **outlier_source** entry in the directions file must be set to **True** for the source.

It is also possible that a strong source (such as 3C295) will lie inside a facet. Such a source can be peeled to avoid dynamic-range limitations. Peeling will be done if a source exceeds **peel_flux_Jy** (set under the **[calibration]** section of the parset) and a sky model is available. After peeling, the facet is then imaged as normal.

9.2.5 Checking the results of a run

You can check the progress of a run with the checkfactor script by supplying the Factor parset as follows:

```
[/data]$ checkfactor factor.parset
```

A plot showing the field layout will be displayed (see Fig. 9.3), with facets colored by their status. Left-click on a facet to see a summary of its state, and press "c" to see a plot of the self calibration images and their clean masks. An

example of such a plot is shown in Fig. 9.4. For a successful self calibration, you should see a gradual improvement from **image02** to **image42**.

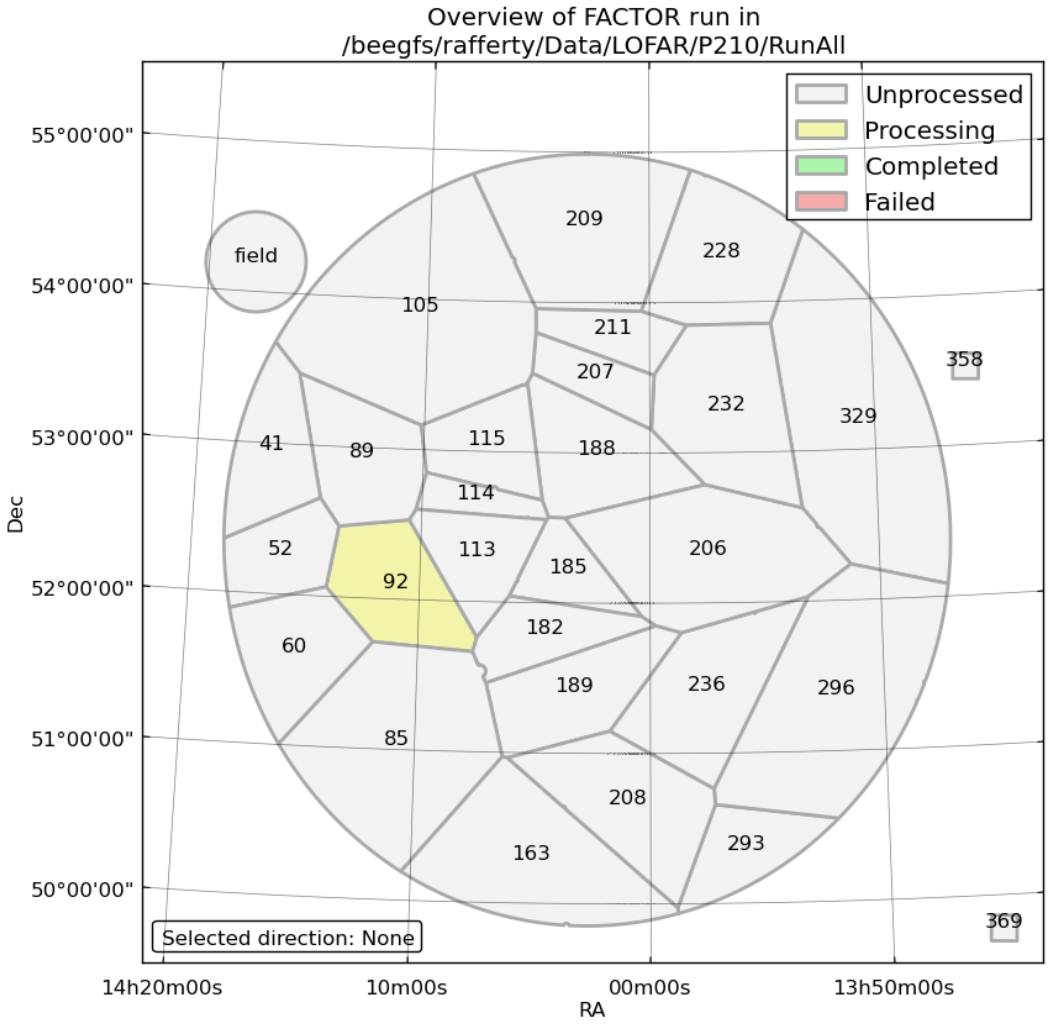


Fig. 9.3: Example **checkfactor** plot.

If self calibration has completed for a direction, select it and press "t" to see plots of the differential TEC and scalar phase solutions and "g" to see plots of the slow gain solutions. These plots can be useful in identifying periods of bad data or other problems.

9.2.6 Dealing with failed self calibration

Self calibration may occasionally fail to converge for a direction, causing to Factor to skip the subtraction of the corresponding facet model. An example in which self calibration failed is shown in Fig. 9.5. Note how the image noise does not improve much in this case during the self calibration process. While Factor can continue with the processing of other directions and apply the calibration solutions from the nearest successful facet to the failed one (depending on the value of the **exit_on_selfcal_failure** option under **[calibration]** in the parset; see online documentation for

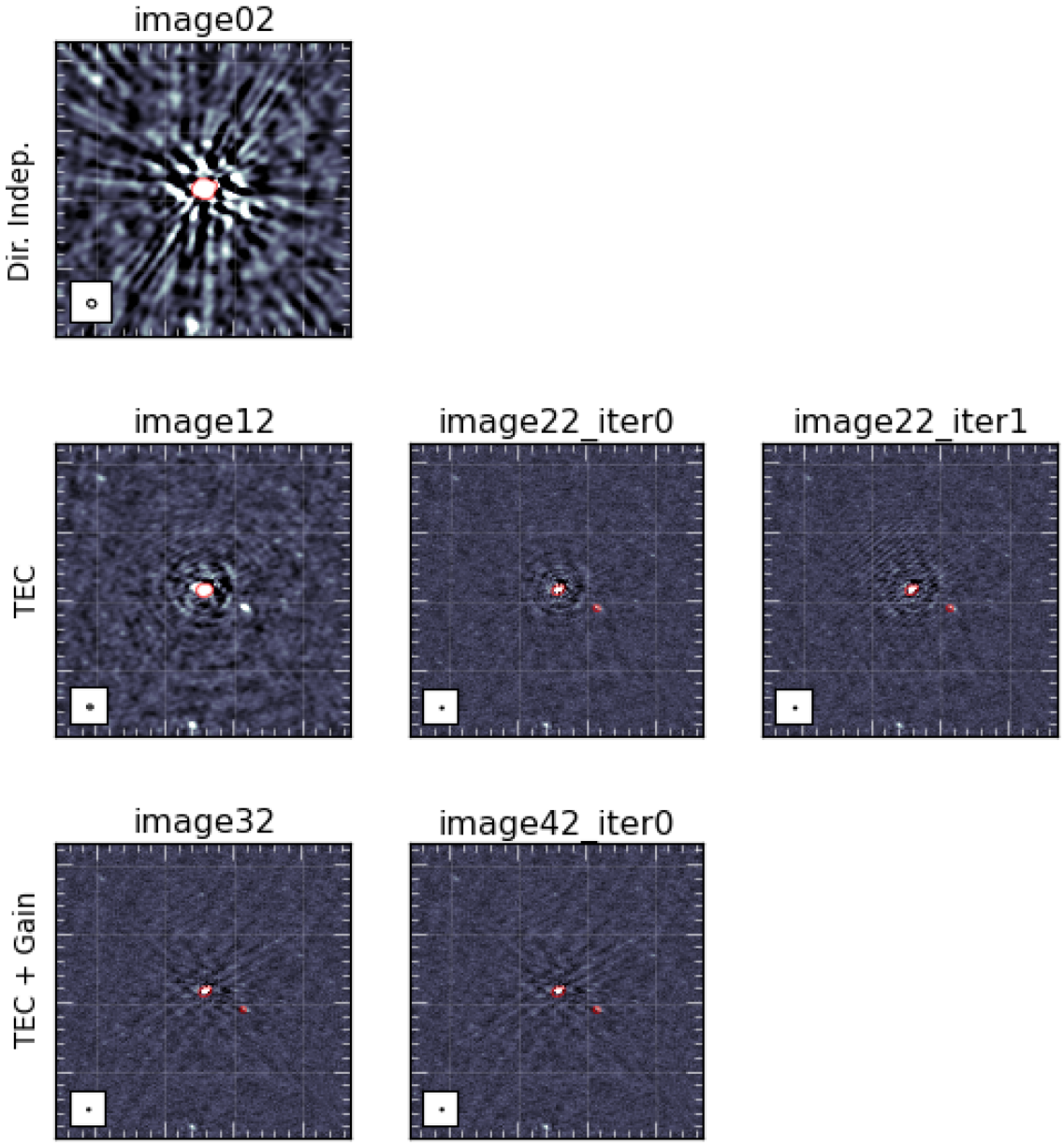


Fig. 9.4: Example of self calibration images for a successful run. The self calibration process starts with **image02** (the image obtained after applying only the direction-independent solutions from pre-Factor) and proceeds to **image12**, **image22**, etc. In some cases, Factor will iterate a step until no more improvement is seen. These steps are indicated by the "iter" suffix.

details), it is generally better to stop Factor and try to correct the problem. In these cases, turning on multi-resolution self calibration or supplying a custom clean mask may improve the results.

Multi-resolution self calibration involves imaging at decreasing resolutions, from 20 arcsec to full resolution ($\approx$ 5 arcsec). This process can stabilize clean in the presence of calibration errors. An example of such a situation is shown in Fig. 9.5. Multi-resolution self calibration can be enabled by setting **multires_selfcal = True** under **[calibration]** in the parset.

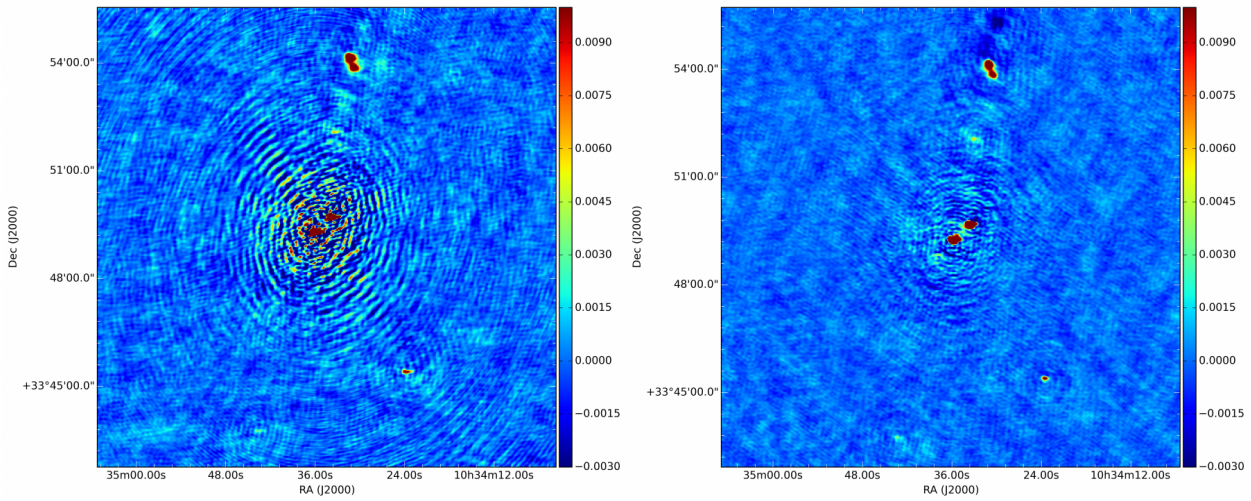


Fig. 9.5: Example of self calibration images at the same stage of reduction for an unsuccessful run (left) in which an incorrect source structure is present (the bright sources both have "L"-shaped morphologies) and the successful run (right) in which multi-resolution self calibration was used.

In some cases, usually when the calibrator has a complex structure, it may be necessary to supply a clean mask made by hand. Such a clean mask can be made by opening one of the self calibration images (e.g., ***image22.image**; see online documentation for details of the **facetselfcal** output) in casaviewer and drawing a region that encloses the source emission but excludes any artifacts. Save the region in CASA format to a file and specify the full path to this file in the **region_selfcal** column of the directions file (named **dir_working/factor_directions.txt** if generated automatically) for the appropriate direction. Note that the supplied mask must include all significant emission within the self calibration region.

Lastly, it is possible that the DDE calibrator is too faint to produce good self calibration results. In this case, the slow-gain solutions are generally noisy and the images produced after the slow-gain calibration (e.g., ***image32.image**) are worse than those before ((e.g., ***image22.image**). To fix this problem, either process other, brighter facets first (to lower the overall noise) or skip self calibration of the problematic facet.

To restart self calibration for the failed direction, you will need to reset it. See the section on *resuming and resetting a run* below for instructions on how to reset a direction.

9.2.7 Imaging at the end of a run

By default, once self calibration has been completed, Factor will image all facets at full resolution (1.5 arcsec cellsize, robust of -0.25, and no taper). To image the facets with different parameters, set one or more of the following facet parameters in the parset under **[imaging]**: **facet_cellsize_arcsec**, **facet_robust**, **facet_taper_arcsec**, and **facet_min_uv_lambda**. These parameters can be lists, with one set of images for each entry. For example, the following settings will make two images per facet, one at full resolution (1.5 arcsec cellsize, robust of -0.25, and no taper) and one at lower resolution (15 arcsec cellsize, robust of -0.5, and a 45 arcsec taper):

```
[imaging]
facet_cellsize_arcsec = [1.5, 15.0]
facet_robust = [-0.25, -0.5]
facet_taper_arcsec = [0.0, 45.0]
```

This run will produce two output directories: **dir_working/results/facetimage/** (with the full-resolution images) and **dir_working/results/facetimage_c15.0r-0.5t45.0u0.0/** (with the lower-resolution images).

9.2.8 Resuming and resetting a run

If a Factor run is interrupted, it can be resumed by simply running it again with the same command. Occasionally, however, you will want to change some settings (e.g., to supply a clean mask for self calibration) or something will go wrong that prevents resumption. In these cases, you can reset the processing for a direction with the "-r" flag to **runfactor**. For example, to reset direction facet_patch_350, the following command should be used:

```
runfactor factor.parset -r facet_patch_350
```

9.2.9 Finding the final output products

Usually, the final, primary-beam-corrected image of the field or target will be located in **dir_working/results/fieldmosaic/field** and will be called ***correct_mosaic.pbcor.fits**. A full description of the primary output products generated by Factor is available online.

Additionally, it may be desirable to image a facet by hand using the calibrated data (e.g., to ensure that any diffuse emission is properly cleaned). To do so, one can use **archivefactor** (see *archiving a factor run*) to archive the calibrated data for a facet in a convenient form for use in CASA or WSClean. Note that these data have not been corrected for any primary beam attenuation but have had the beam effects at the field phase center removed.

9.2.10 Archiving a Factor run

A Factor run can be archived at any point with **archivefactor**. For example, the following will produce a minimal archive (including images and plots, but no visibility data):

```
archivefactor factor.parset dir_output
```

To include the calibrated visibility data for one of more facets in the archive, add the "-d" flag followed by the names of the facets:

```
archivefactor factor.parset dir_output -d facet_patch_468,facet_patch_350
```

Lastly, to make an archive from which a Factor run can later be resumed, use the "-r" flag:

```
archivefactor factor.parset dir_output -r
```

To resume from such an archive, use **unarchivefactor**. See the online documentation for details.

CHAPTER

TEN

USEFUL RESOURCES

10.1 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.

Essential pages on the wiki are:

- Main imaging wiki page
- DPPP
- BBS

10.2 Useful analysis scripts

A compilation of some practical python scripts is available at the LOFAR-Contributions GitHub repository.

The scripts provided are[1] :

- **average.py**: averages images from multiple sub bands together
- **average_weights.py**: averages images weighting them by the inverse of their variance.
- **CallSolFlag.py**: flags calibrated data
- **closure.py**: prints closure phase vs time/elevation for selected antennas
- **coordinates_mode.py**: routines to work with astronomical coordinates
- **plot_flags.py**: plots "images" of frequency versus time on a baseline-by-baseline basis, with the pixel values equal to the visibility amplitudes
- **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.
- **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.

[1] If you have other scripts that could be useful for other commissioners, please contact Sarrvesh Sridhar.

- **msHistory.py**: prints information from the HISTORY table of a Measurement Set. Useful for obtaining a quick listing of the parset values used in e.g. DPPP.
- **plotElevation.py**: given a Measurement Set, plots the elevation of the target source as a function of time
- **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.
- **embiggen.csh**: increases the size of plotted points in postscript files. Useful when producing ps output from e.g. uvplot.py.
- **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.
- **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.
- **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.

In addition to the scripts described above, the GitHub repository also contains a collection of other scripts that are no longer being maintained or deprecated.

10.3 Contact points

Some key contact points are listed below:

- **LOFAR Imaging Cookbook** - Sarrvesh Sridhar
- **DPPP** - Ger van Diepen, Tammo Jan Dijkema, and David Rafferty
- **AOFlagger** - Andre Offringa
- **BBS** - Tammo Jan Dijkema, Vishambhar Nath Pandey
- **AWImager** - Tammo Jan Dijkema, and Bas van der Tol
- **Python-casacore/TaQL/Casacore** - Ger van Diepen and Tammo Jan Dijkema
- **SAGECAL, Shapelets** - Sarod Yatawatta
- **PyBDSM, LSMTool** - David Rafferty

CHAPTER

ELEVEN

CALIBRATION WITH BBS[1]

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

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

11.1 Overview

The sky as observed by LOFAR is the "true" sky distorted by characteristics of the instrument (station beam, global bandpass, clock drift, and so on) 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 *global parameter estimation* we will treat the global solve.

As input, BBS requires an observation (one or more subbands / measurement sets), a *source catalog*, a table of initial model parameters (see section on *instrument tables*), and a configuration file (also called *parset*, see section *example reductions*) 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.

[1] This section is maintained by Tammo Jan Dijkema). It was originally written by Joris van Zwieten and Reinout van Weeren.

11.2 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 *example reductions*).
- **<source catalog>**, which defines a list of sources that can be used for calibration (see section *source catalog*).
- The **-f** option, which overwrites stale information (the instrument table and sky model) 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.

11.3 Source catalog

The source catalog / sky model 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 sky model 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 in a GitHub repository.

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 *GSM* 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 sourcedb 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.

11.3.1 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**: it's defined relative to the north of some image and not relative to 'true' north. This is only relevant if the Gaussian is not symmetric, i.e. the major axis and minor axis differ, and you are transferring a Gaussian model component to another pointing over a large angular distance.

11.3.2 Spectral index

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

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

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 $[c_0, \ldots, c_n]$.

11.3.3 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:

$$Q(\lambda) = p\ I(\lambda)\ \cos(2\chi(\lambda)) \tag{11.1}$$

$$U(\lambda) = p\ I(\lambda)\ \sin(2\chi(\lambda)) \tag{11.2}$$

$$\chi(\lambda) = \chi_0 + RM\ \lambda^2 \tag{11.3}$$

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:

$$\chi_0 = \frac{1}{2}\tan^{-1}(\frac{U}{Q}) - \lambda_0^2\ RM \tag{11.4}$$

$$p = \frac{\sqrt{(Q^2 + U^2)}}{I(\lambda_0)} \tag{11.5}$$

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.

11.4 GSM

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 *Source catalog*) 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 in section *Spectral index*.

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 11.1. The arguments RA and DEC can be copy-pasted from the output of 'msoverview' to select the pointing of your observation. A sensible value for **radius** is 5 degrees.

Table 11.1: The parameters and criteria that are used for creating the initial Global Sky Model.

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 source in outfile. Defaults to 4 Jy.
assocTheta	degrees	Search radius centered on VLSS source for which counterparts are searched. Defaults to 10 arcsec taking into account the VLSS resolution of 80 arcsec.

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 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:

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

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 the equation in section *Source catalog*. 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 **store-spectraplots**. When true and not performance driven, this will plot all the spectra of the sources in the catalog file, named by their VLSS name.

11.4.1 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.

11.4.2 Generating a higher resolution skymodel

The skymodel generated using **gsm.py** is limited in angular resolution due to the $80''$ resolution of the VLSS. Users in need of a higher resolution skymodel can generate a model of the sky using data from The GMRT Sky Survey - Alternative Data Release (TGSS-ADR). Skymodels generated using TGSS-ADR have an angular resolution of $20''$. Information about the survey and the data products provided by TGSS-ADR are described in detail in Intema et al (2017).

You can generate a skymodel from TGSS-ADR as

```
# 3C295 = (212.83,52.21)
ra=212.83
dec=52.21
radius=5
out=skymodel.txt
wget -O $out 'http://tgssadr.strw.leidenuniv.nl/cgi-bin/gsmv2.cgi?coord=${ra},${dec}\&
↪radius=${radius}\&unit=deg\&deconv=y
```

The above set of commands will generate the file **skymodel.txt** which contains all TGSS-ADR sources within a 5 degree radius from the specified coordinate (212.83,52.21). Note that the coordinates are in degrees and in the J2000 epoch.

11.4.3 Instrument tables

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 parmdb. To view the contents of a parmdb, use the tool **parmdbplot.py** (see section *Inspecting the solutions*).

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

Before even starting the actual BBS, there need to be values in the parmdb, because they are used as starting values by BBS[2]. 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.

Please note that if you create your own parmdb, you will almost always want to include the default **adddef** commands listed below to set the correct defaults for **Gain**, **DirectionalGain**. Otherwise, estimating these parameters will not work correctly. The default parameters are automatically created if you use **calibrate-stand-alone** with the option **-f** or **–replace-parmdb**. To set default values in a parmdb manually, use the following commands in parmdbm.

```
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
```

11.5 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 11.2, 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.

Table 11.2: 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).

Effect	Description	Jones matrix
ScalarPhase	A phase that is equal for both dipoles	$\begin{pmatrix} e^{\alpha_p} & 0 \\ 0 & e^{\alpha_p} \end{pmatrix}$
Rotation[18]	Faraday Rotation without frequency dependency	$\begin{pmatrix} \cos(\alpha_p) & -\sin(\alpha_p) \\ \sin(\alpha_p) & \cos(\alpha_p) \end{pmatrix}$
FaradayRotation[19]	Faraday Rotation, $\rho_p = \alpha_p \cdot (\text{channel wavelength})^2$	$\begin{pmatrix} \cos(\rho_p) & -\sin(\rho_p) \\ \sin(\rho_p) & \cos(\rho_p) \end{pmatrix}$
Directional TEC	TEC (ionosphere), $\phi_p = \frac{-8.44797245 \cdot 10^9 \alpha_p}{\text{channel frequency}}$	$\begin{pmatrix} e^{\phi_p} & 0 \\ 0 & e^{\phi_p} \end{pmatrix}$
Beam[20]	The LOFAR beam model	See section on Beam model
DirectionalGain	Directional gain	$\begin{pmatrix} \alpha{:}0{:}0{:}p & \alpha{:}0{:}1{:}p \\ \alpha{:}1{:}0{:}p & \alpha{:}1{:}1{:}p \end{pmatrix}$
CommonScalarPhase	Scalar phase	$\begin{pmatrix} e^{\alpha} & 0 \\ 0 & e^{\alpha} \end{pmatrix}$
CommonRotation	Rotation	$\begin{pmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{pmatrix}$
TEC	TEC (ionosphere), $\phi = \frac{-8.44797245 \cdot 10^9 \alpha}{\text{channel frequency}}$	$\begin{pmatrix} e^{\phi} & 0 \\ 0 & e^{\phi} \end{pmatrix}$
Gain	Gain	$\begin{pmatrix} \alpha{:}0{:}0 & \alpha{:}0{:}1 \\ \alpha{:}1{:}0 & \alpha{:}1{:}1 \end{pmatrix}$
Clock	Clock, $\phi = \alpha \cdot 2\pi \cdot$ channel frequency	$\begin{pmatrix} e^{\phi} & 0 \\ 0 & e^{\phi} \end{pmatrix}$

[2] For calibration with DPPP this is not necessary.
[18] The effect **Rotation** is stored in the instrument table as **RotationAngle**.
[19] The effect **FaradayRotation** is stored in the instrument table as **RotationMeasure**.
[20] The effect **CommonRotation** is stored in the instrument table as **CommonRotationAngle**.

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

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

For configuration of the beam model that is used, see section *Beam model*.

11.5.1 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 analogue tile beam former.

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

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

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

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

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

11.6 Solver

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

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

11.7 Example reductions

The sequence of operations that BBS will perform on the data are defined in a configuration or **parset** file[3]. The BBS documentation on the LOFAR wiki documents all the options (see the LOFAR wiki), and it is highly recommended that you obtain a hard-copy of this for future reference.

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

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

11.7.1 Simulation

Given a source catalog and (optionally) a table of model parameters, BBS can be used to compute simulated *uv*-data (without noise)[4]. 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 *Pre-computed visibilities*.

An example parset file[5] to simulate {it uv}-data for all the sources in the source catalog is shown below.

```
# simulation.parset
Strategy.InputColumn = DATA
Strategy.TimeRange = []
Strategy.Baselines = *&
Strategy.ChunkSize = 100
Strategy.Steps = [predict]
Step.predict.Operation = PREDICT
Step.predict.Model.Sources = []
Step.predict.Output.Column = MODEL_DATA
```

11.7.2 Gain calibration (direction-independent)

The following parset[6] describes a direction independent gain calibration. The meaning of each parameter is the same as in section *Simulation* 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**.

```
# uv-plane-cal.parset
Strategy.InputColumn = DATA
Strategy.TimeRange = []
Strategy.Baselines = *&
Strategy.ChunkSize = 100
Strategy.Steps = [solve, correct]
```

(continues on next page)

[3] Examples of these files can be found at *https://github.com/lofar-astron/LOFAR-Cookbook <https://github.com/lofar-astron/LOFAR-Cookbook>*.

[4] Use DPPP's **predict** step for this, it is much more efficient.

[5] This example can be found at https://github.com/lofar-astron/LOFAR-Cookbook.

[6] This example can be found at https://github.com/lofar-astron/LOFAR-Cookbook.

(continued from previous page)

```
Step.solve.Operation = SOLVE
Step.solve.Model.Sources = [3C196]
Step.solve.Model.Gain.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.Model.Beam.Enable = True  <-uncomment if you want to apply the beam

Step.correct.Operation = CORRECT
Step.correct.Model.Gain.Enable = T
Step.correct.Output.Column = CORRECTED_DATA
#Step.correct.Model.Beam.Enable = True <-uncomment if you want to apply the beam
```

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[7].

11.7.3 Gain calibration (direction-dependent) with source subtraction

This section has been adapted from a document written by Annalisa Bonafede. In the following, we report the parset[8] file and skymodel used for the subtraction of Cas A and Cyg A from the observation of the radio source 3C380.

The parset includes the following steps:

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

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

Strategy.ChunkSize = 100
Strategy.Steps = [solve, subtract, correct]

Step.solve.Operation = SOLVE
Step.solve.Model.DirectionalGain.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.subtract.Operation = SUBTRACT
```

(continues on next page)

[7] 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.

[8] This example can be found at https://github.com/lofar-astron/LOFAR-Cookbook.

(continued from previous page)

```
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[9]

```
############################################################
# 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 Fig. 11.1, compared to the map obtained without subtraction.

11.8 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 Fig. 11.2). 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 Fig. 11.2, 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.

11.8.1 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.

[9] This source catalog is available at https://github.com/lofar-astron/LOFAR-Cookbook.

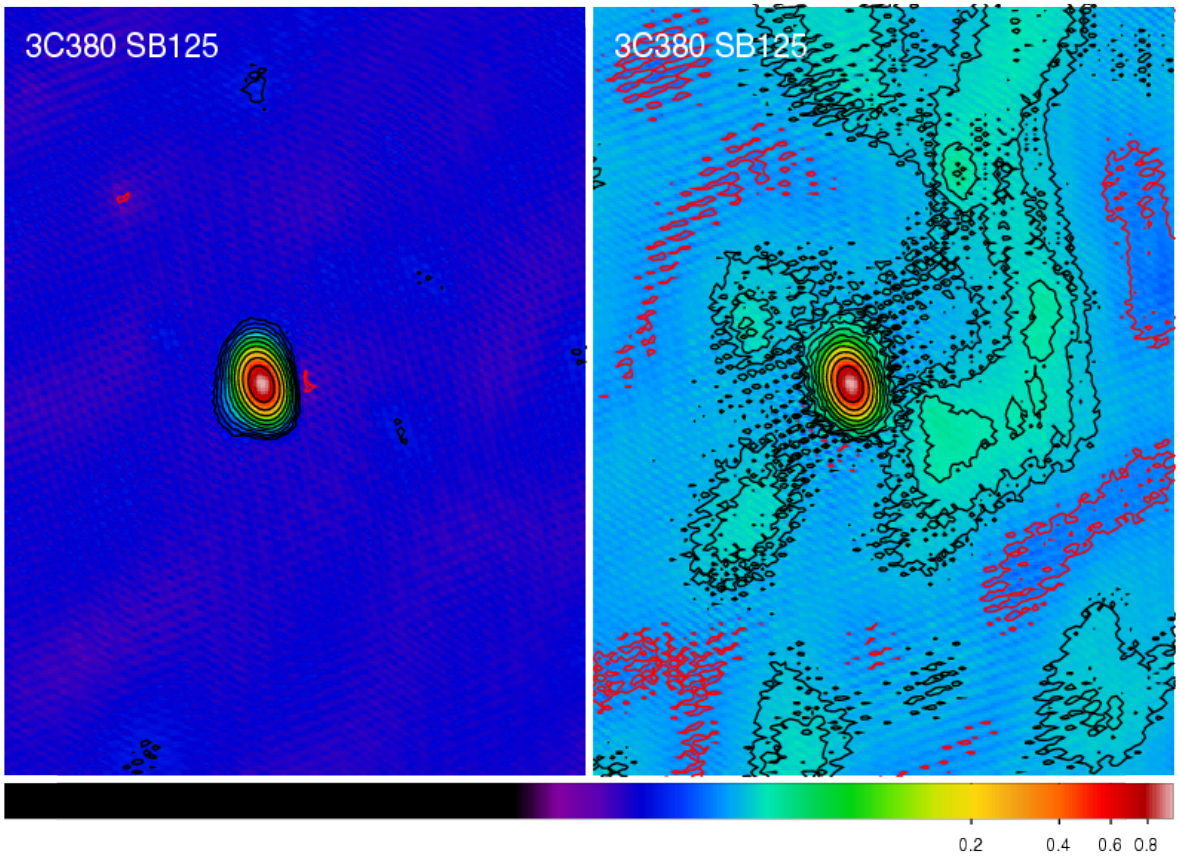


Fig. 11.1: The radio source 3C380 imaged by LOFAR at 135~MHz (observation ID **L2010_08567**). **Left panel**: Self calibration has been applied with directional-gain correction and subtraction of CygA and CasA. **Right panel**: Self calibration has been applied without correction. The lowest contour 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.

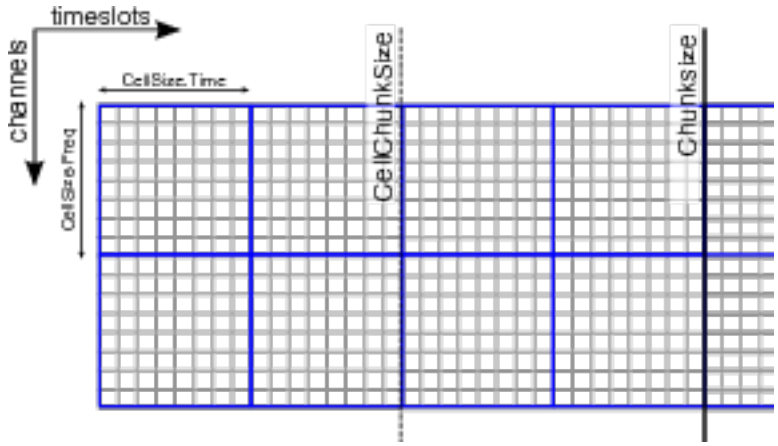


Fig. 11.2: The different solve domains and chunk parameters for BBS. In this example, **CellSize.Time=8**, **CellSize.Freq=8**, **ChunkSize=32**, **CellChunkSize=2** (do not use these values yourself, they are probably not good for actual use).

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 × nThreads** should give you some speedup.

You can instruct **calibrate-stand-alone**[10] to run with multithreading with the parameter **-t** or **–numthreads**

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

11.9 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.

11.9.1 Setting up your environment

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

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

The distributed version of BBS requires a file that describes the compute cluster (an example of such a file is **cep1.clusterdesc**[11]) and a configuration or *parset*[12] 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.

[10] Currently, it is not possible to combine multithreading with a global solve or with the **calibrate** script.
[11] You can copy this file from https://github.com/lofar-astron/LOFAR-Cookbook.
[12] Examples can be found in https://github.com/lofar-astron/LOFAR-Cookbook.

11.9.2 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/<>'/<>'.
- **<parset>**, which defines the reduction (see section *Example reductions*).
- **<source catalog>**, which defines a list of sources that can be used for calibration (see section *Source catalog*).
- **<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 *Logging on to CEP3*).
- 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[13] 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.

11.9.3 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

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

```
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]
```

11.10 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.[14]

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.

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

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.
- **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/*"
```

11.11 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 **lofar** environment (see *Setting up your working environment*). 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 Fig. 11.3). 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 $\text{frequency} \times \text{time}$. 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 Fig. 11.4 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

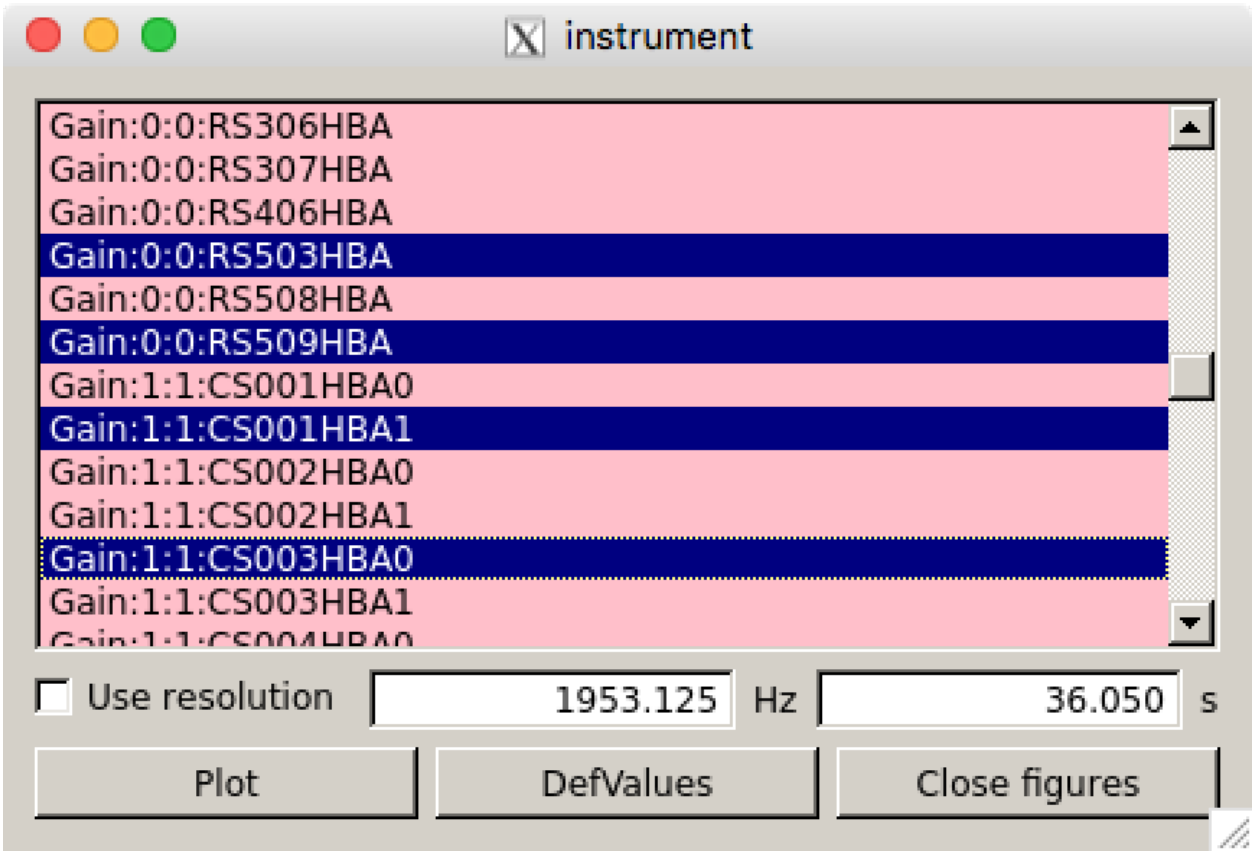


Fig. 11.3: The main window of **parmdbplot**.

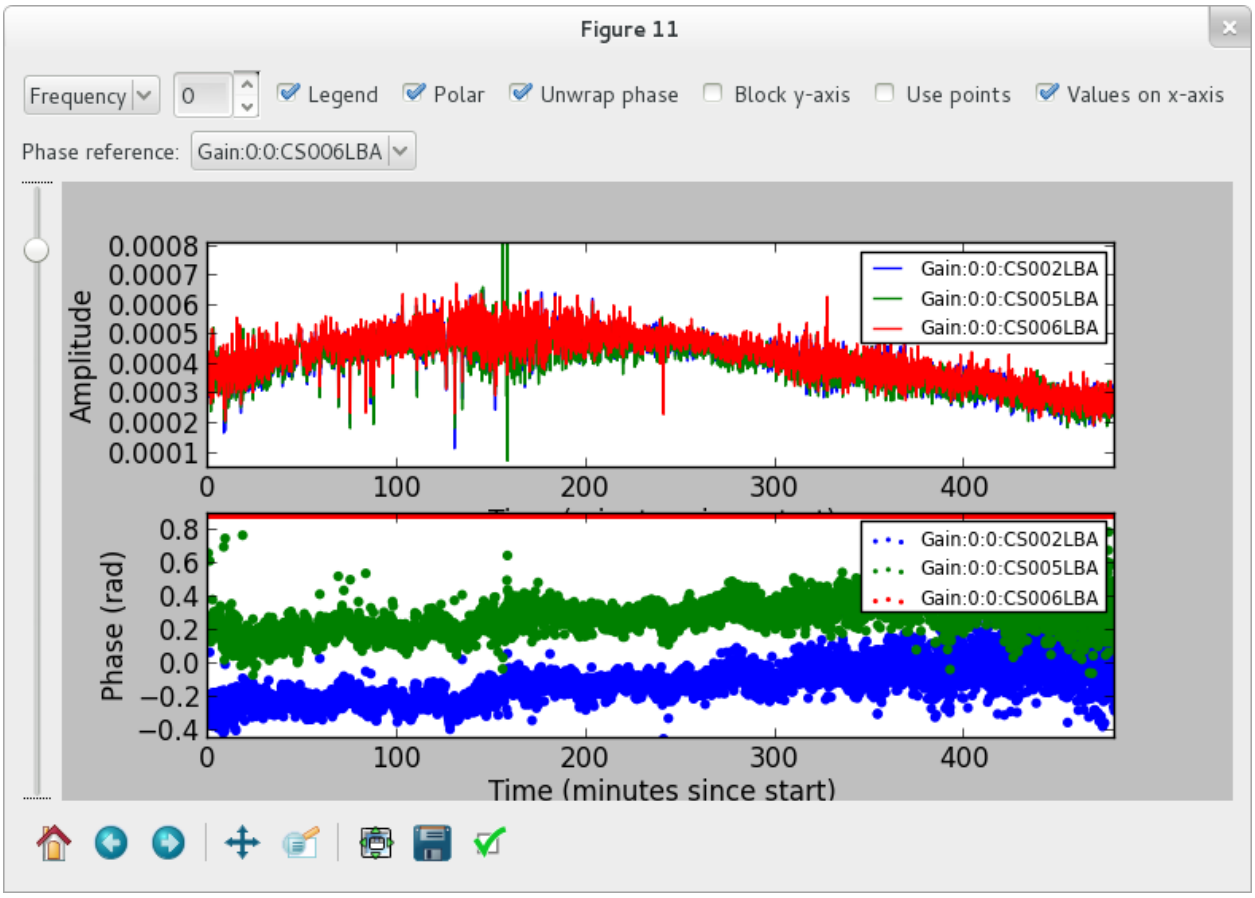


Fig. 11.4: The plot window of **parmdbplot**.

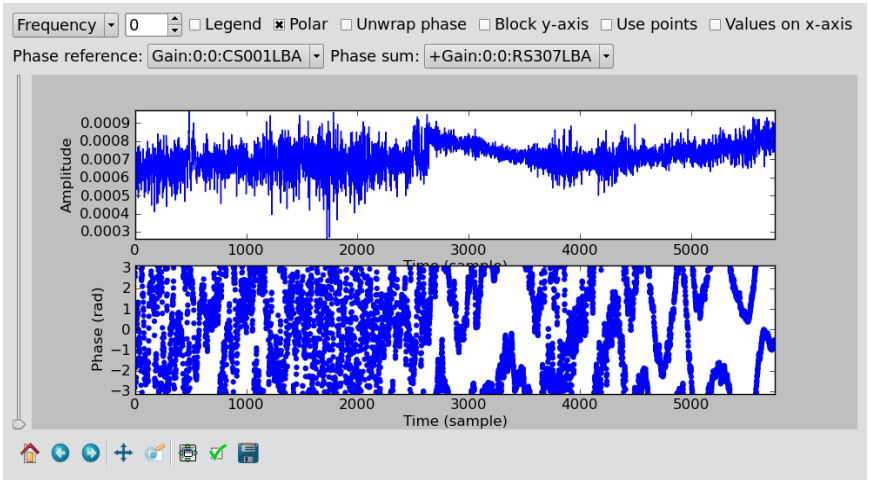


Fig. 11.5: The plot window of **parmdbplot** with a phase sum.

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.

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. 11.5). 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**[15] script by George Heald:

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

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

11.12 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).

11.12.1 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 model 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.

Fig. 11.6 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.

11.12.2 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

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

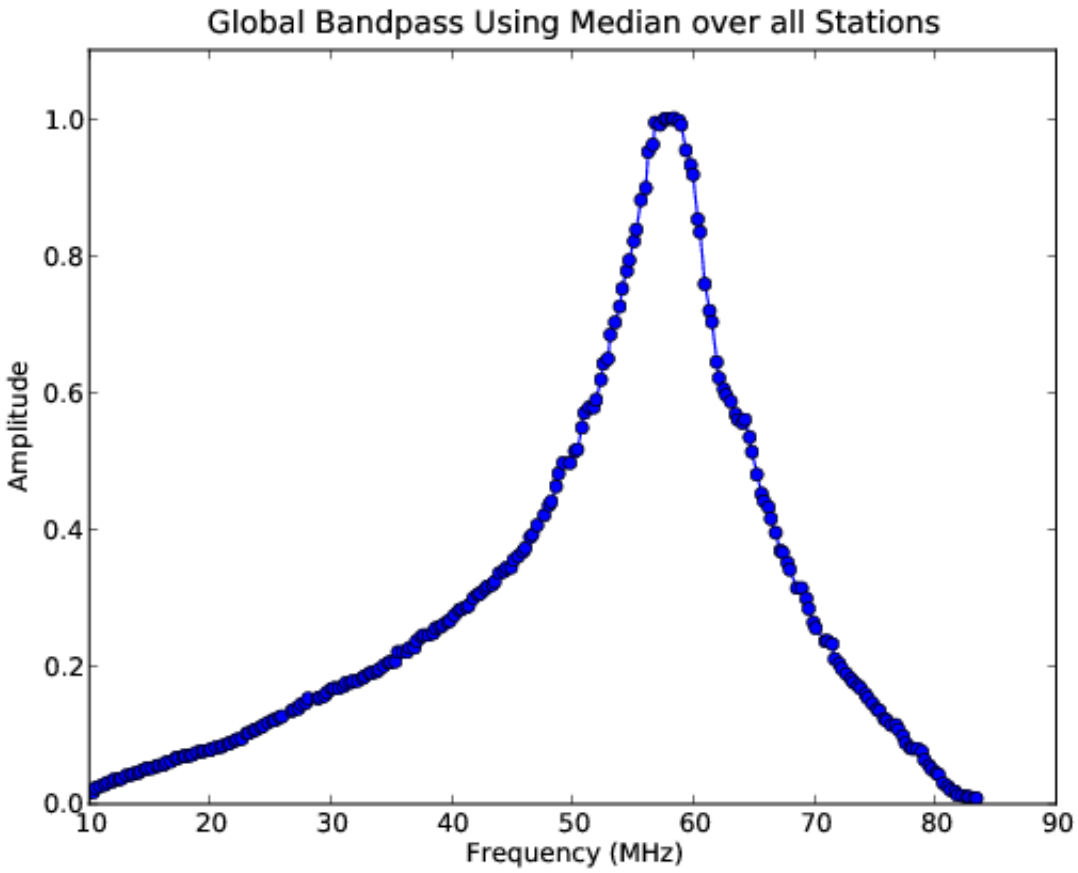
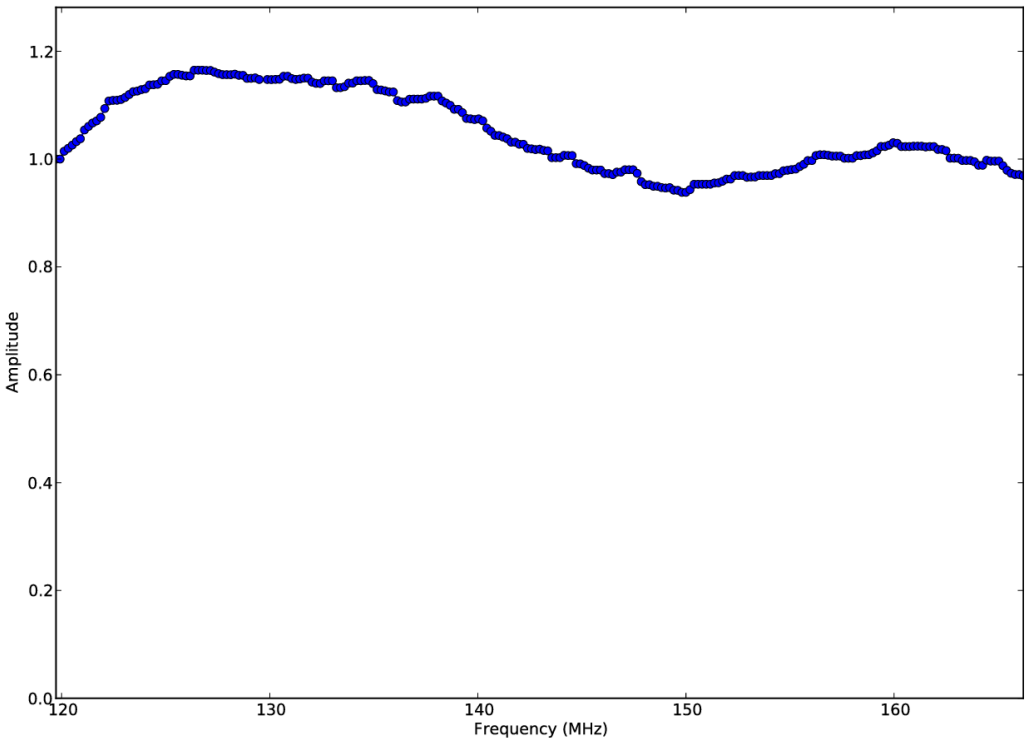
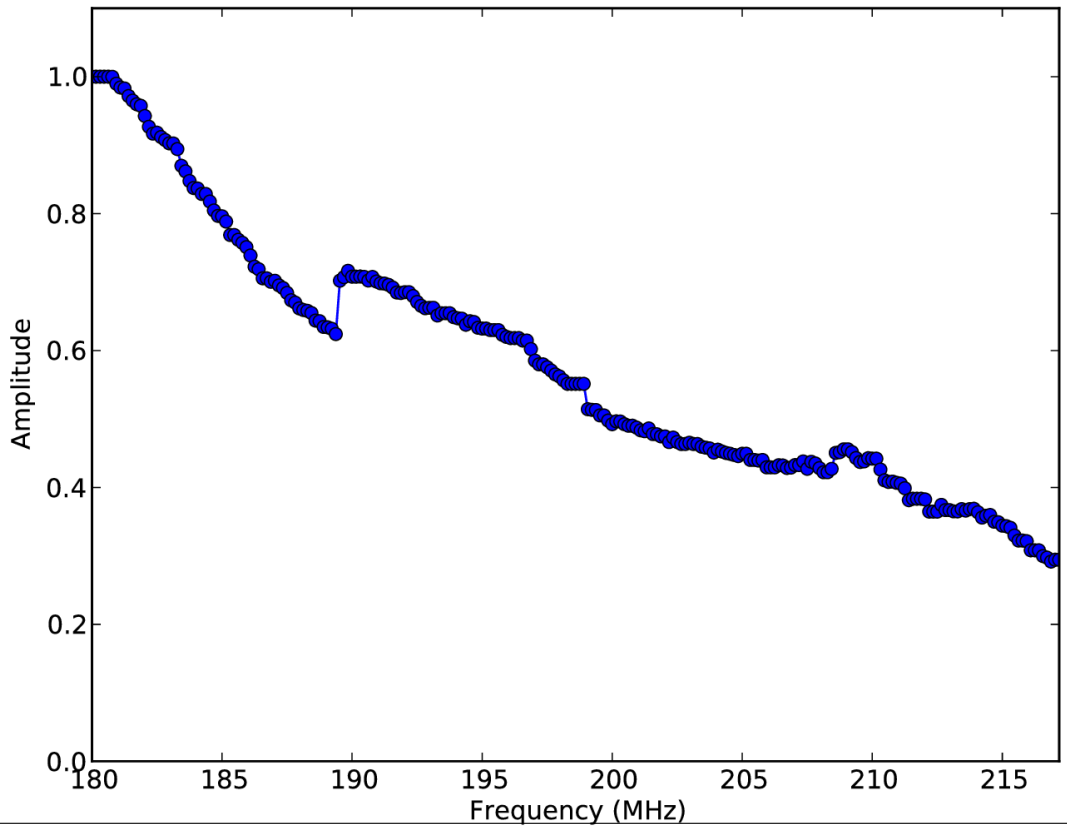


Fig. 11.6: The global bandpass in LBA between 10–85 MHz.


Amplitude
Frequency (MHz)
0.0
0.2
0.4
0.6
0.8
1.0
1.2
120
130
140
150
160


Amplitude
Frequency (MHz)
0.0
0.2
0.4
0.6
0.8
1.0
180
185
190
195
200
205
210
215

demixing was done. A two-point-source model was used for calibration. The resulting bandpass is shown in Fig. 11.7. Note that several frequency intervals in the HBA-high observation were affected by severe RFI.

.. _gain transfer:

11.13 Gain transfer from a calibrator to the target source

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

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

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

11.13.1 The "traditional" approach

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

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

In order to achieve time independence you should use these settings in the BBS parset[16]

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

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

```
> 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
```

[16] Note that these settings would be dangerous for a long observation!

An alternative scheme to make the solutions of the calibrator observation time independent is to have a smaller **CellSize.Time**, and afterwards take the median[17]. This way, time cells where calibration failed do not affect the solutions. To follow this scheme, calibrate the calibrator as you would do normally, for example with **CellSize.Time=5**. To make the solutions time independent, use the tool **parmexportcal.py** (see **parmexportcal –help** or its documentation. For example, if you have calibrated the calibrator and stored the solution in **cal.parmdb**, you can take the median amplitudes as follows

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

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

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

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

11.13.2 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:

- 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:

- Calibrate the calibrator using BBS. The time and frequency resolution of the solutions can be whatever is needed for the best results. Ensure that the beam is enabled.

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

- (Optional) Perform some corrections to the calibrator solutions, for example flagging, smoothing or interpolation. This is best done in LoSoTo.
- 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 = []
  ```

[17] This is the calibration scheme that the LOFAR pipelines follow.

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

11.14 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 or DPPP.

11.15 Troubleshooting

- Bugs should be reported using the LOFAR issue tracker.
- 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 **:math:‘<‘key name:math:‘>‘*.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.

11.15.1 Common problems

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

1. Station is not a LOFAR station:

```
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.

2. setupsourcedb failed:

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

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

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

3. Database failure:

```
[FAIL] error: clean database for key default failed
```

Solution: Check if you can reach the database server on **ldb002** and make sure that you created your personal database correctly.

CHAPTER

TWELVE

SKY MODEL CONSTRUCTION USING SHAPELETS[1]

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. 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. ref{shp:viravla}, 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. 12.1, 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.

[1] The author of this chapter is Sarod Yatawatta.

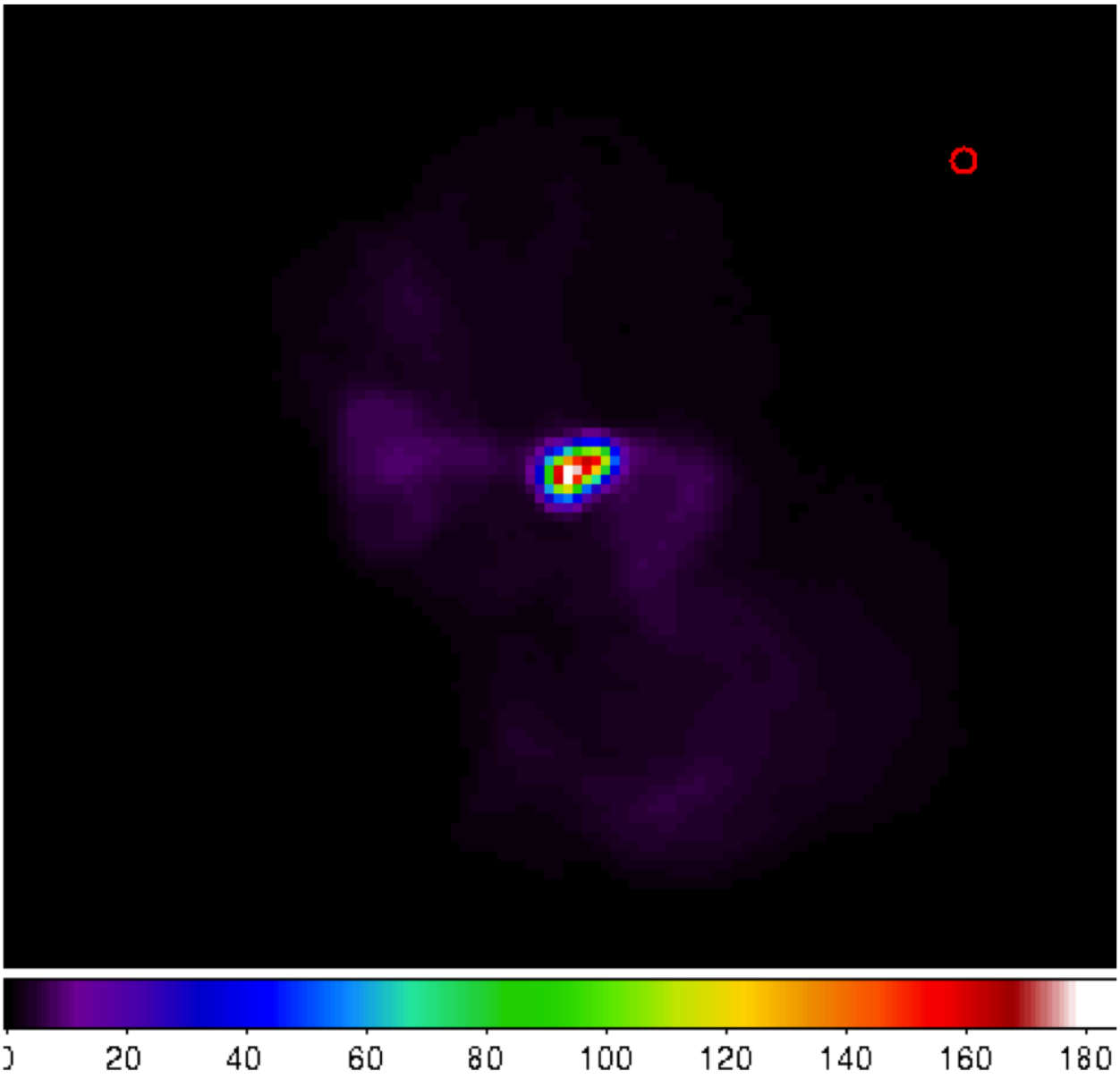


Fig. 12.1: Virgo-A image made by the VLA at 74 MHz. The red circle on top right corner is the PSF.

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/shapelet/bin** in the CEP clusters.

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. 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
```

(continues on next page)

(continued from previous page)

```
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
```

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** or **-o 1** for LSM. Note that **buildsky** will now (version 0.0.6) only produce LSM with 3rd order spectra. Spectral indices use natural logarithm, $\exp(\ln(I_0) + p1 * \ln(f/f_0) + p2 * \ln(f/f_0)^2 + \ldots)$ so if you have a model with common logarithms like

$$10^{(\log(J_0)+q1*\log(f/f_0)+q2*\log(f/f_0)^2+\ldots)}$$

then, conversion is $I_0 = J_0$, $p1 = q1$, $p2 = q2/\ln(10)$, $p3 = q3/(\ln(10)^2)$ and so on.

As you can see, both above sky models are the same. In addition, the LSM sky model can be used to represent Gaussians (name starting with **G**), disks (name starting with **D**) and rings (name starting with **R**).

Once you have such a sky model (text file **sky.txt**), and a FITS file called **example.fits**, you can do many things

```
restore -f example.fits -i sky.txt
```

will replace the FITS file with the sky model, so the original image will be overwritten;

```
restore -f example.fits -i sky.txt  -a
```

will add the sky model to the image; and

```
restore -f example.fits -i sky.txt  -s
```

will subtract the sky model from the FITS file.

You can also use solutions obtained by **SAGECal** when you restore a sky model:

```
restore -f example.fits -i sky.txt -c sagecal_cluster.txt -l sagecal_sky.txt
```

will use the solution file **sagecal_sky.txt** and the cluster file **sagecal_cluster.txt** while restoring the sky model. New solution files created by **SAGECal** has 3 additional lines at the beginning. Newer versions (0.0.10) of restore will properly handle this.

As before, you can see more options by typing

```
restore -h
```

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. 12.2.

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. 12.3.

We will go through the options in Fig. ref{shp:gui1} one by one.

- **Cutoff** This parameter is used to select the rectangle of pixels where most of the flux in the image is concentrated. A cutoff of 0.9 will select all the pixels above 0.1 of the peak flux. By using cutoff of 1.0, the whole image is selected.
- **max** If this value is not 0, pixels above this value will be truncated to this value.

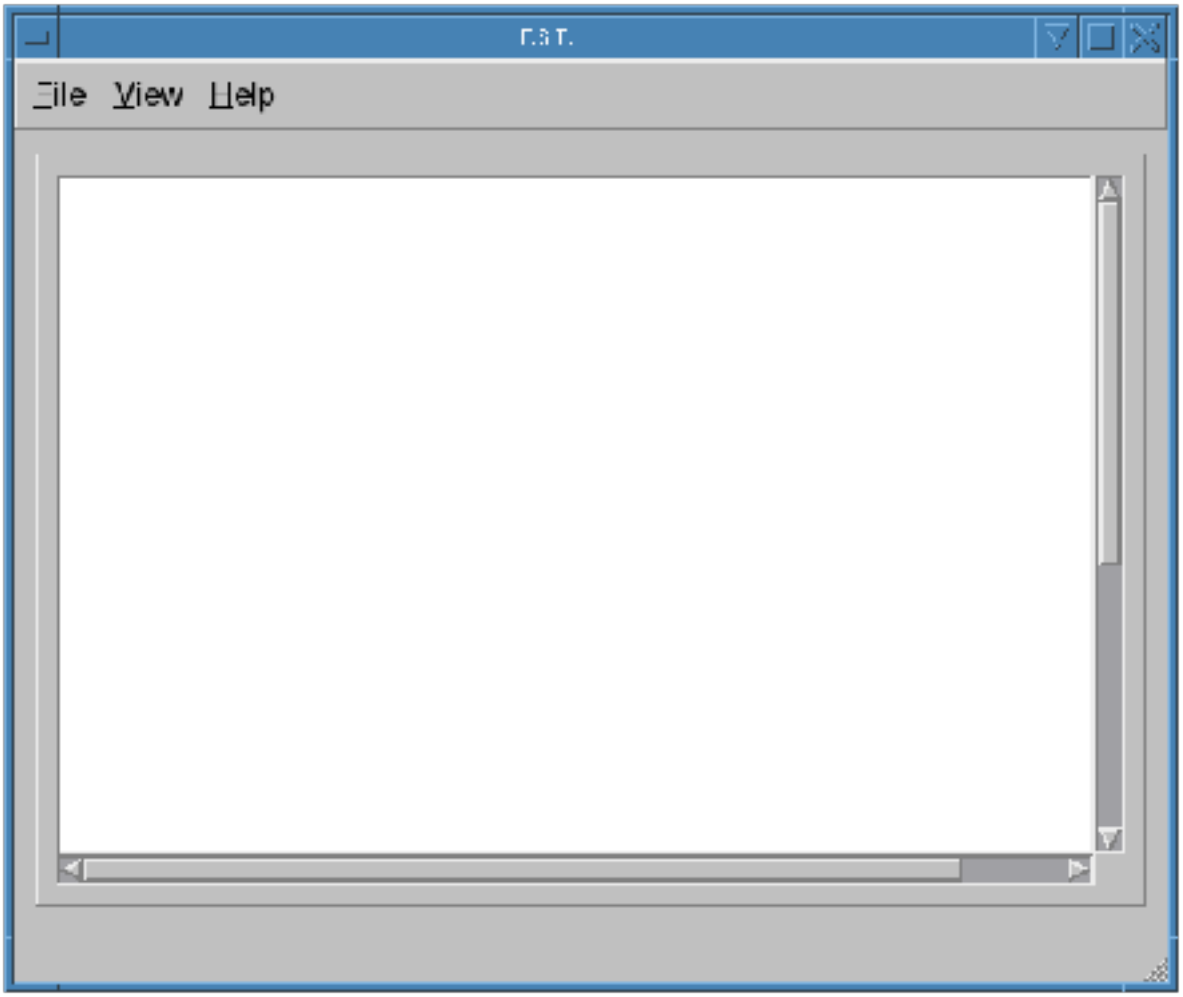


Fig. 12.2: The **shapelet_gui** initial screen.

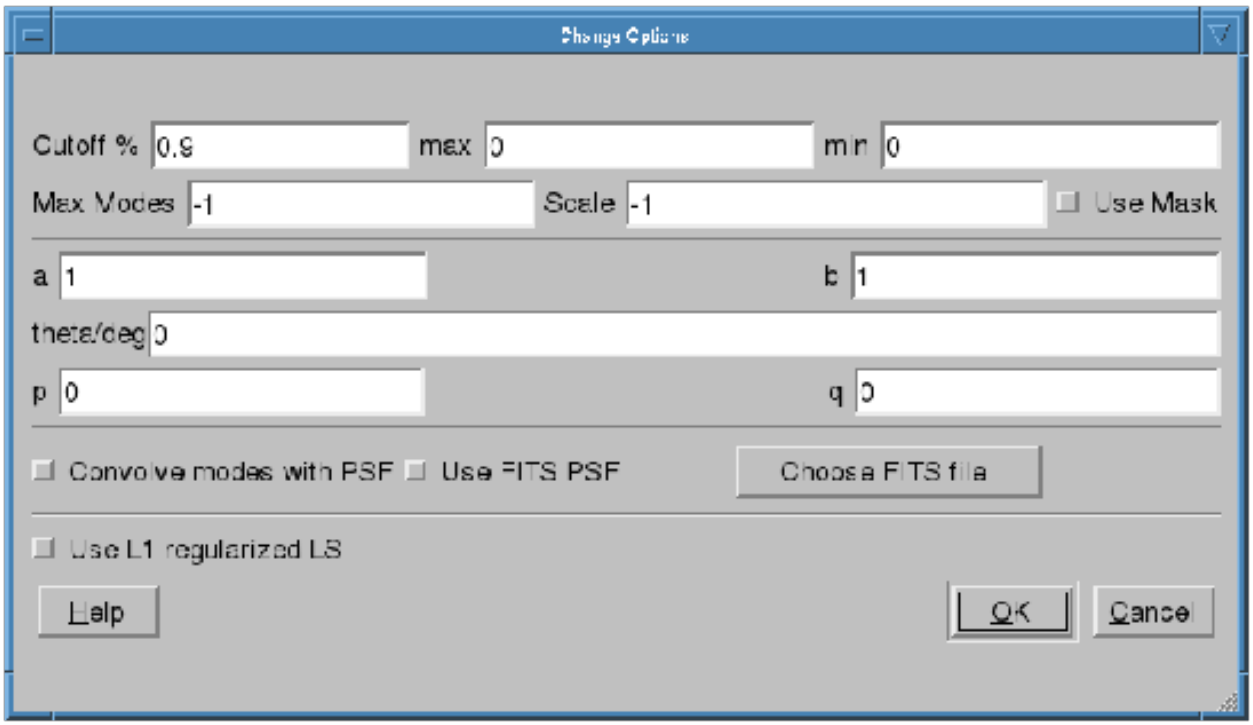


Fig. 12.3: The options dialog for shapelet decomposition.

- **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. 12.4. 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 --
# 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
```

12.2.9 create_clusters.py

This script creates a cluster file that can be used by SAGECal, given an input sky model.

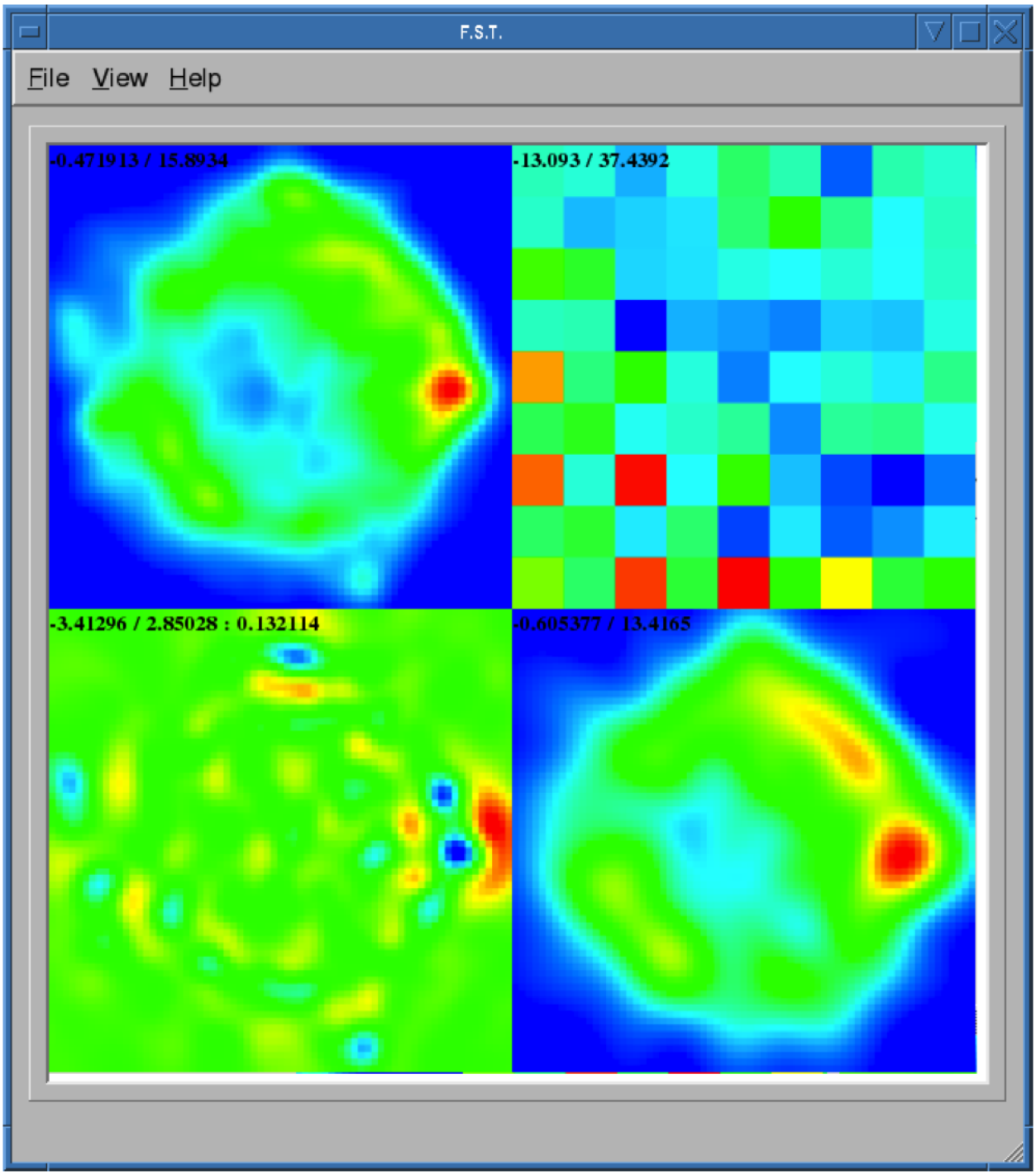


Fig. 12.4: Output of shapelet modelling: (top left) original image (top right) shapelet modes (bottom left) residual image (bottom right) shapelet model.

```
Usage: create_clusters.py [options]

Options:
  -h, --help            show this help message and exit
  -s SKYMODEL, --skymodel=SKYMODEL
                        Input sky model
  -c CLUSTERS, --clusters=CLUSTERS
                        Number of clusters
  -o OUTFILE, --outfile=OUTFILE
                        Output cluster file
  -i ITERATIONS, --iterations=ITERATIONS
                        Number of iterations
```

The sky model has to be in LSM format, **-c** option gives the number of clusters to create. It uses weighted K-means clustering algorithm, and the number of iterations for this is given by **-i**, usually about 10 iterations is enough for convergence. This and many other scripts can be downloaded from **sagecal.sf.net**.

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. 12.1 (FITS file **vira-cen.fits**) to build the initial sky model.

12.3.1 Initial point source model

As we mentioned in the *Introduction*, the central compact part in Fig. 12.1 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. 12.1.

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**):

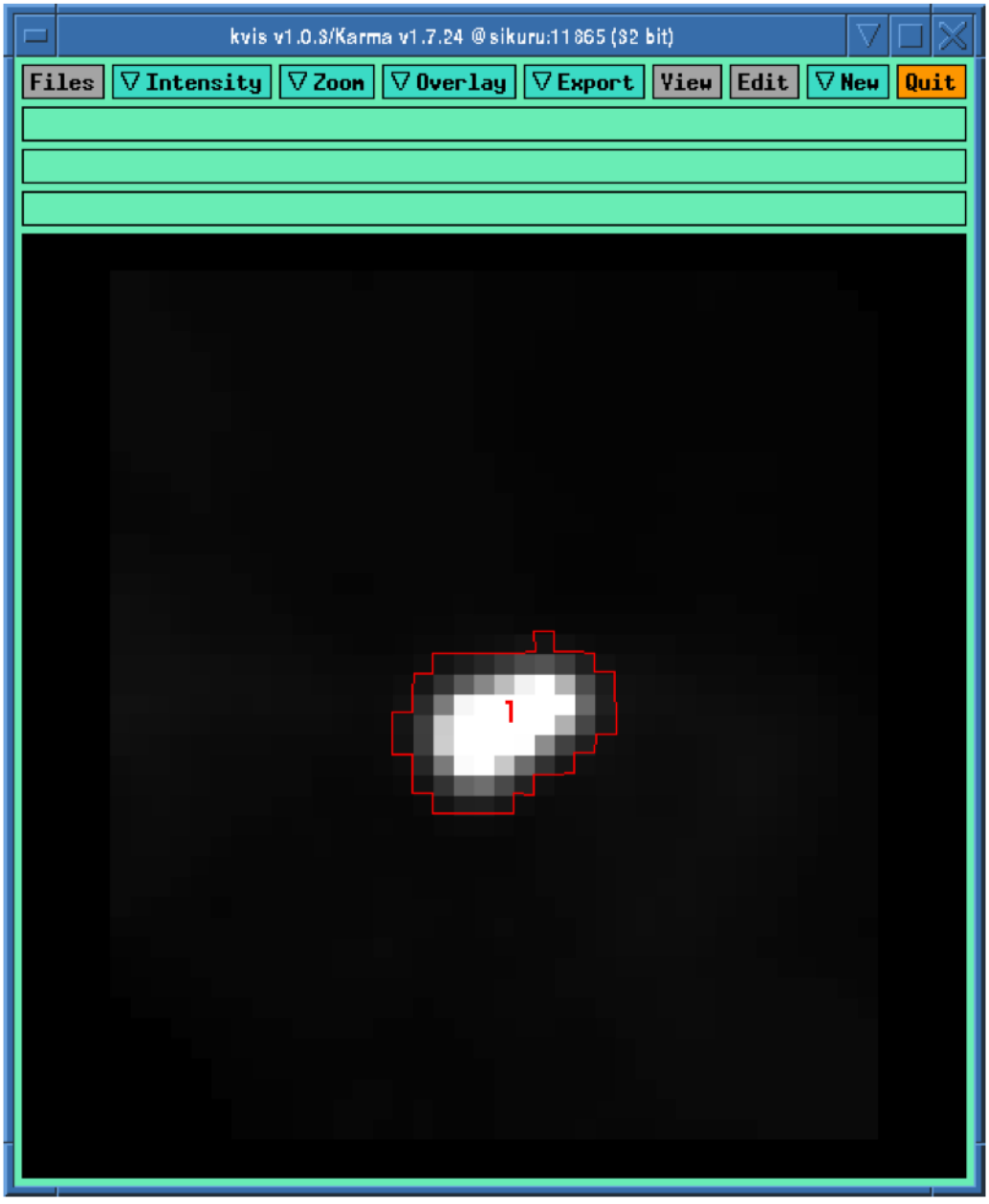


Fig. 12.5: Compact center indicated by the red curve.

```
   # (Name, Type, Ra, Dec, I, Q, U, V,
ReferenceFrequency='60e6', SpectralIndexDegree='0',
SpectralIndex:0='0.0', MajorAxis, MinorAxis, Orientation) = format
 # The above line defines the field order and is required.
 P1C1, POINT, 12:30:45.93, +12.23.48.07, 172.155091, 0.0, 0.0, 0.0
 P1C2, POINT, 12:30:47.39, +12.23.51.92, 141.518663, 0.0, 0.0, 0.0
 P1C3, POINT, 12:30:47.34, +12.23.31.64, 173.054910, 0.0, 0.0, 0.0
 P1C4, POINT, 12:30:48.90, +12.23.40.67, 177.304557, 0.0, 0.0, 0.0
 P1C5, POINT, 12:30:48.75, +12.23.21.23, 155.029319, 0.0, 0.0, 0.0
```

Using **ds9** we can also see our sky model as in Fig. 12.6.

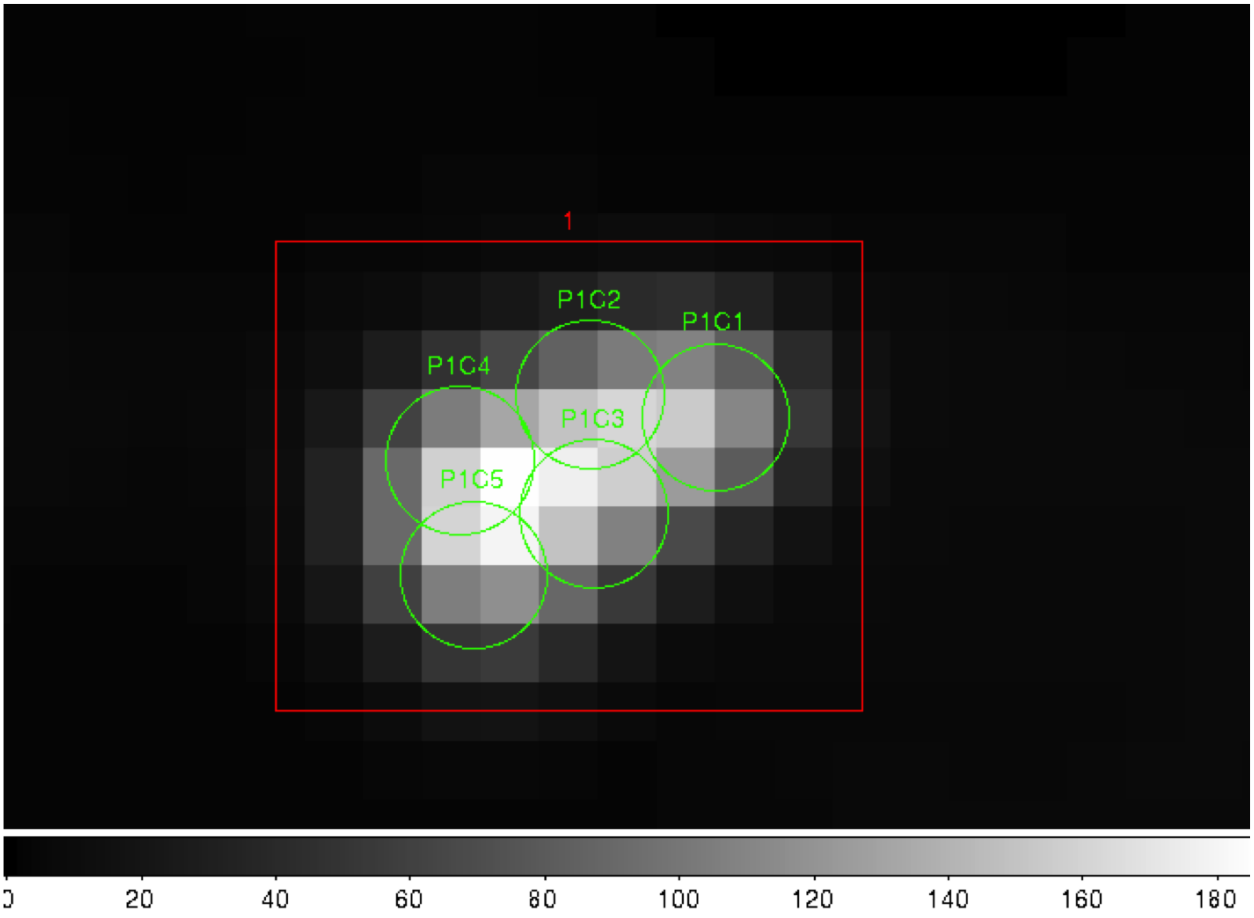


Fig. 12.6: Compact center modeled by two point sources (green circles).

12.3.2 Initial shaplet model

Next, we need to model the extended structure in Fig. 12.1. 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. 12.7.

Note that the bright central part in Fig. 12.7 is almost subtracted. It is not completely gone, and some parts of it is

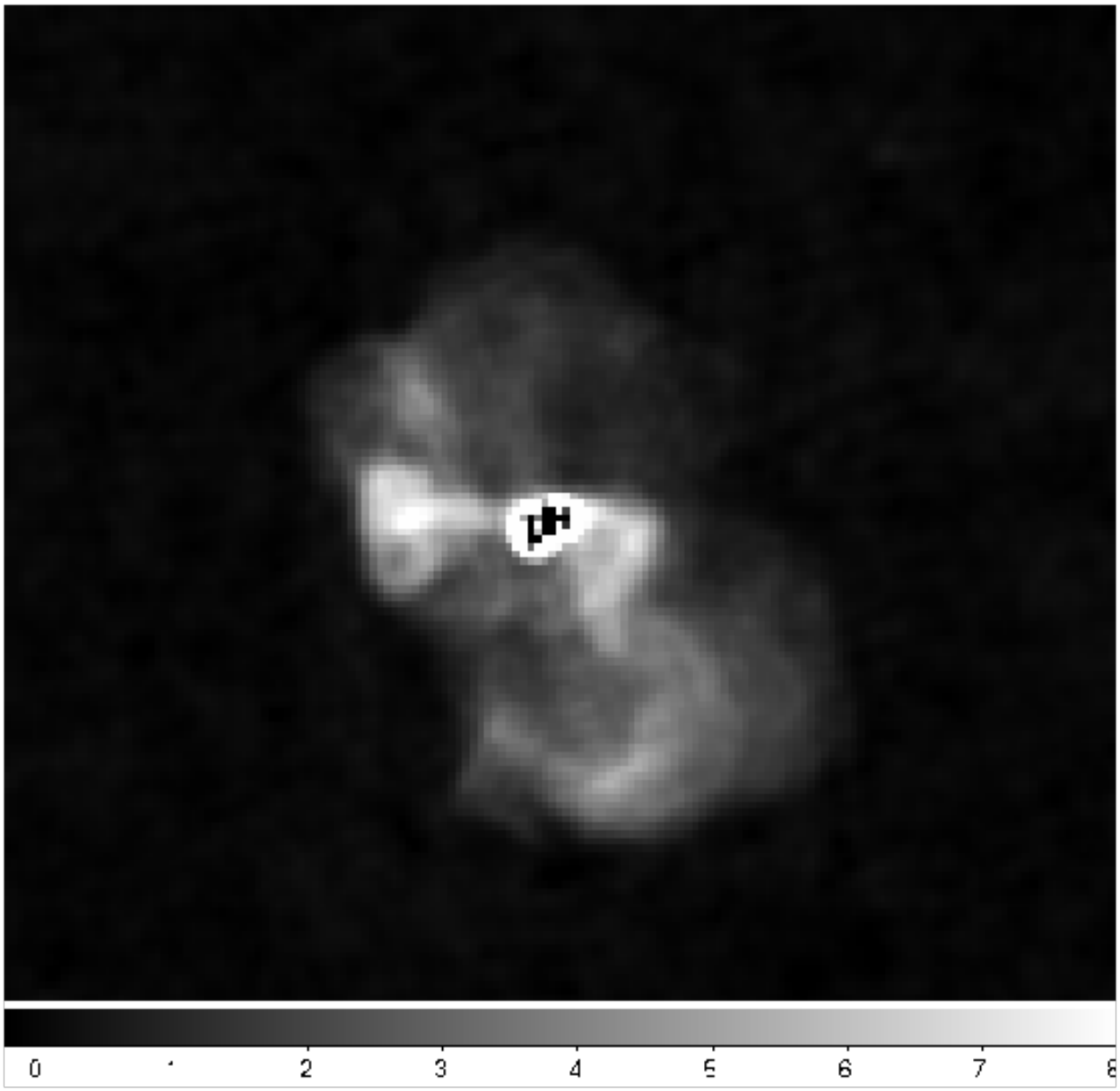


Fig. 12.7: Diffused structure after subtracting the center.

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
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. 12.8.

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. 12.9.

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', SpectralIndex=
↪'[0.0]', Ishapelet) = format
# The above line defines the field order and is required.
, , CENTER, 12:30:45.00, +12.23.48.00
P1C1, POINT, CENTER, 12:30:45.93, +12.23.48.07, 172.155091, 0.0, 0.0, 0.0
P1C2, POINT, CENTER, 12:30:47.39, +12.23.51.92, 141.518663, 0.0, 0.0, 0.0
P1C3, POINT, CENTER, 12:30:47.34, +12.23.31.64, 173.054910, 0.0, 0.0, 0.0
P1C4, POINT, CENTER, 12:30:48.90, +12.23.40.67, 177.304557, 0.0, 0.0, 0.0
P1C5, POINT, CENTER, 12:30:48.75, +12.23.21.23, 155.029319, 0.0, 0.0, 0.0
VirAD, shapelet, CENTER, 12:30:48.317433, 12.23.27.999947, 1.0, , , ,
vira-cen.fits.modes
```

Note that the above model gives **CENTER** as the patch direction.

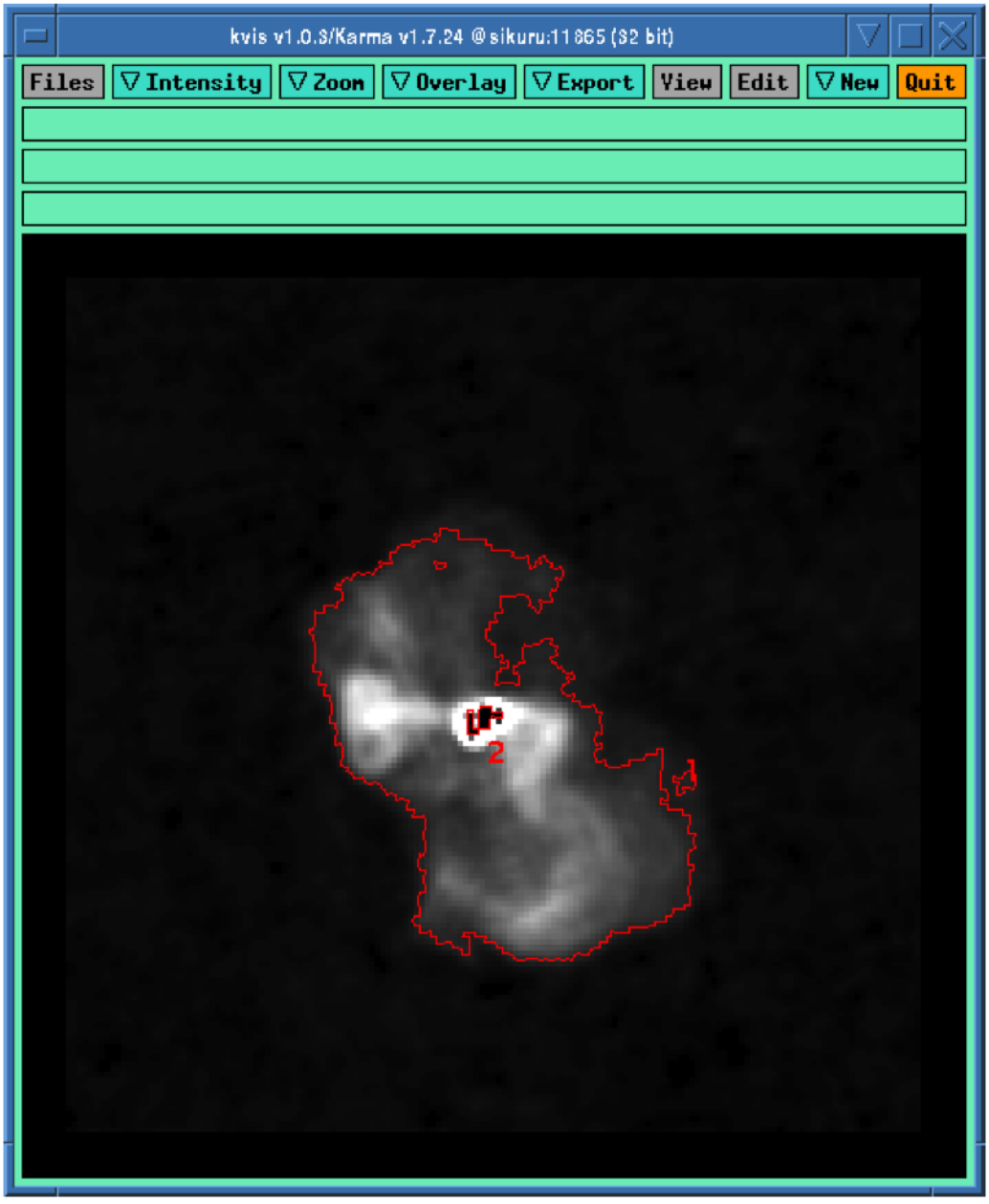


Fig. 12.8: Mask for the diffused structure.

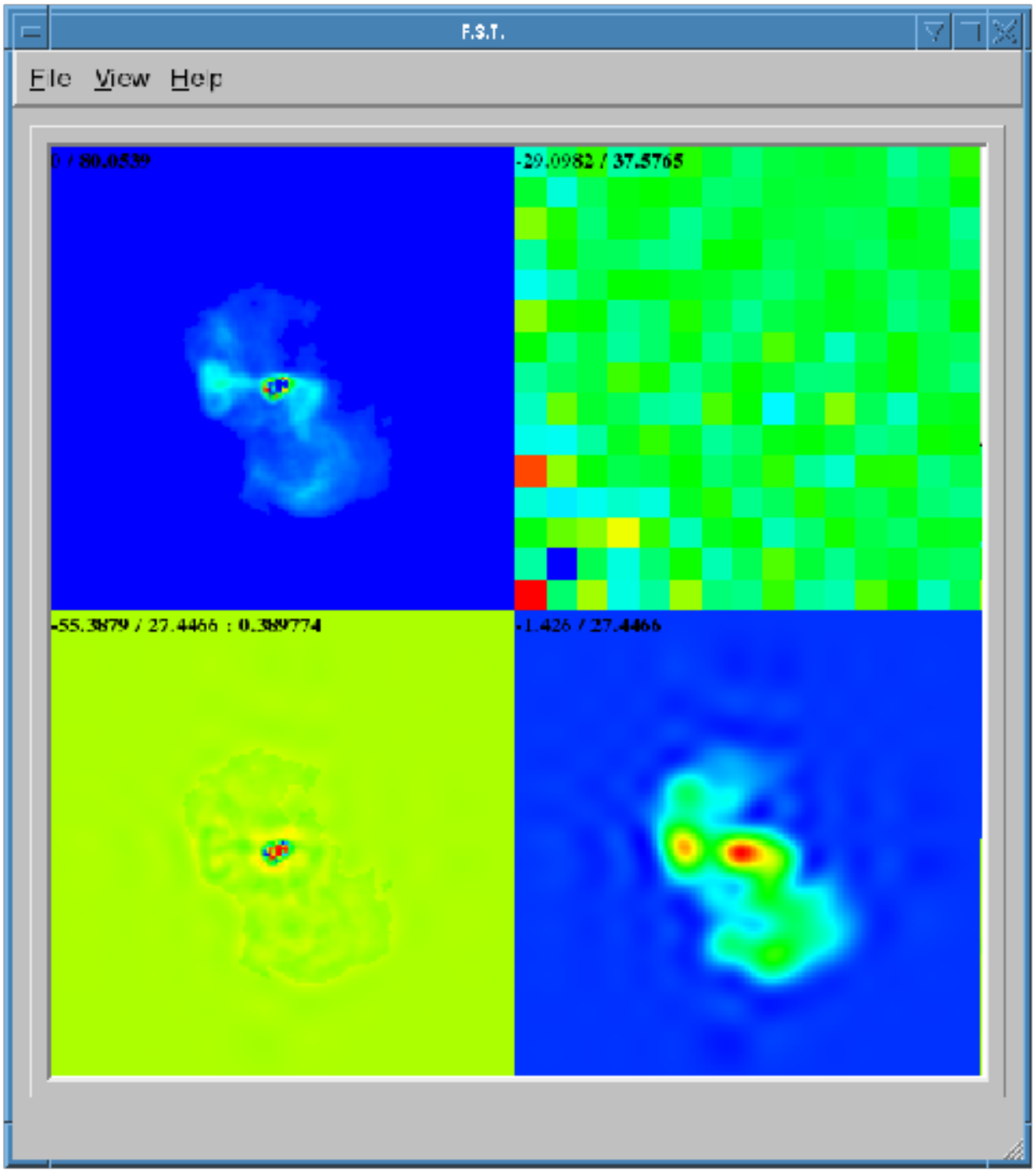


Fig. 12.9: Shapelet model of the diffused structure.

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. 12.10.

By looking at Fig. 12.10, we do not see any major discrepancy in our sky model (although we have lower resolution) so we go ahead with calibration.

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. 12.11.

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

Using Fig. 12.11, 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. 12.12, 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.

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.

12.3.8 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 Sarod Yatawatta.

12.4 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.

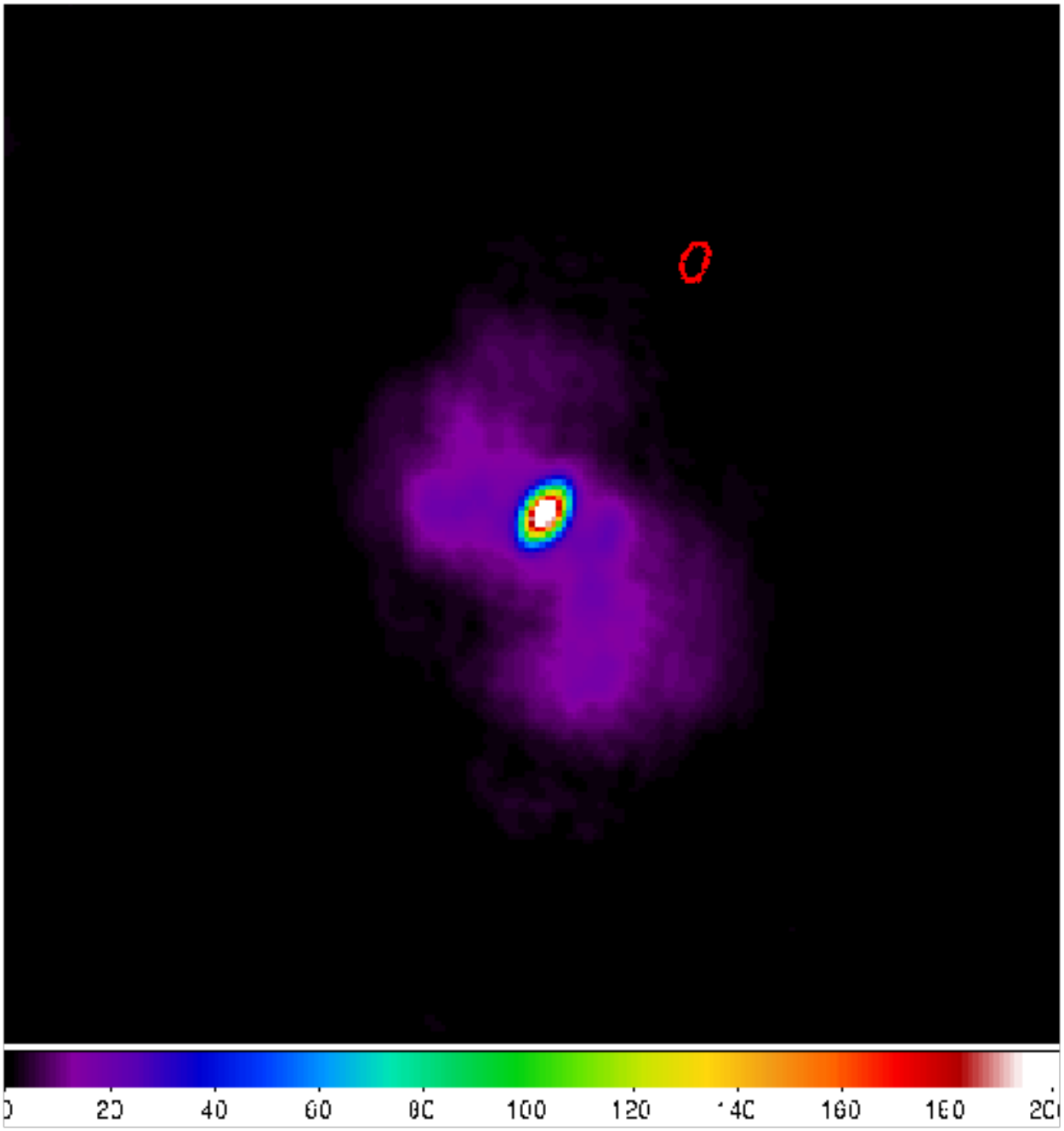


Fig. 12.10: Simulated image of Virgo-A. The red ellipse is the PSF.

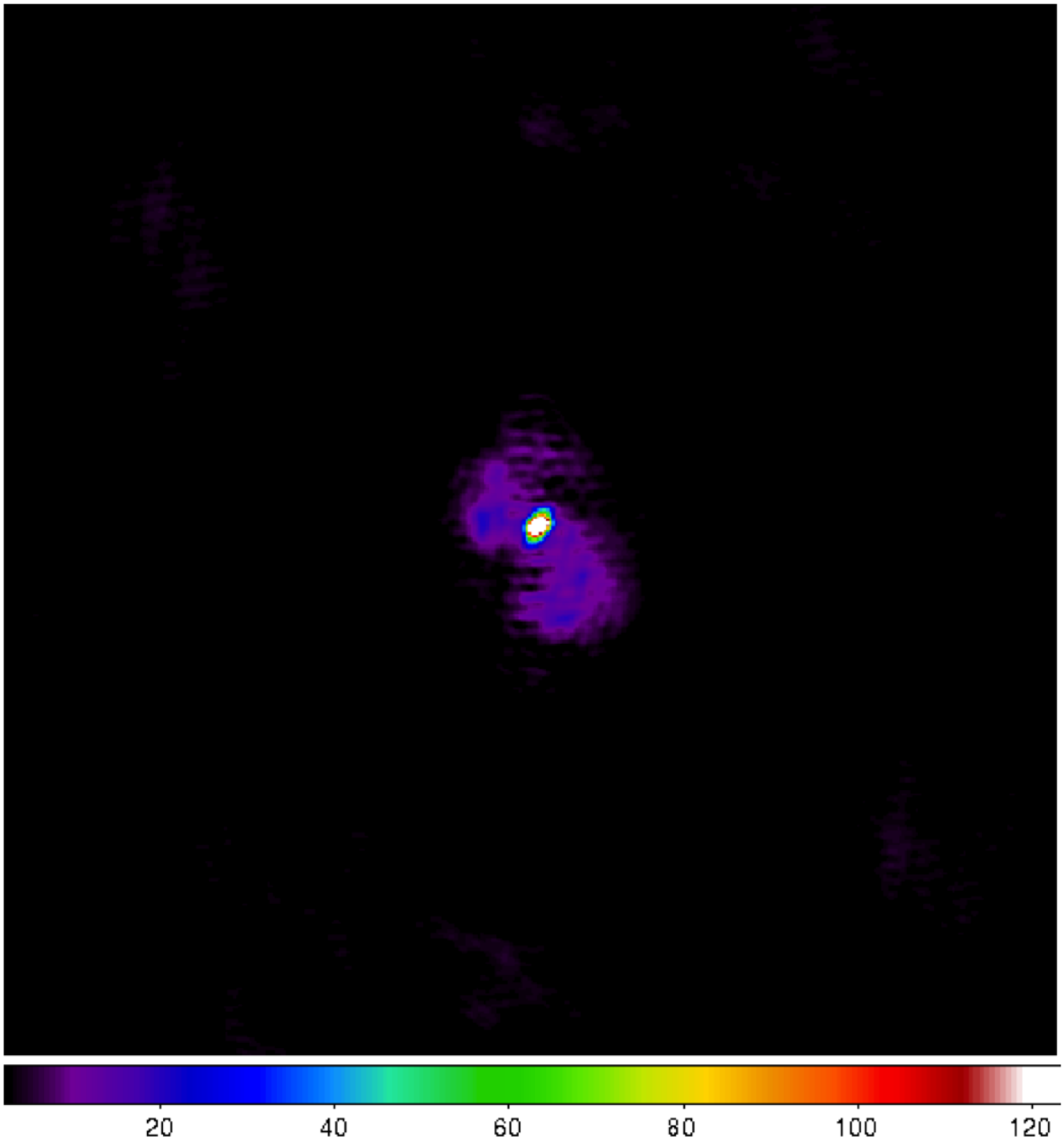


Fig. 12.11: Calibrated image of Virgo-A (uniform weights).

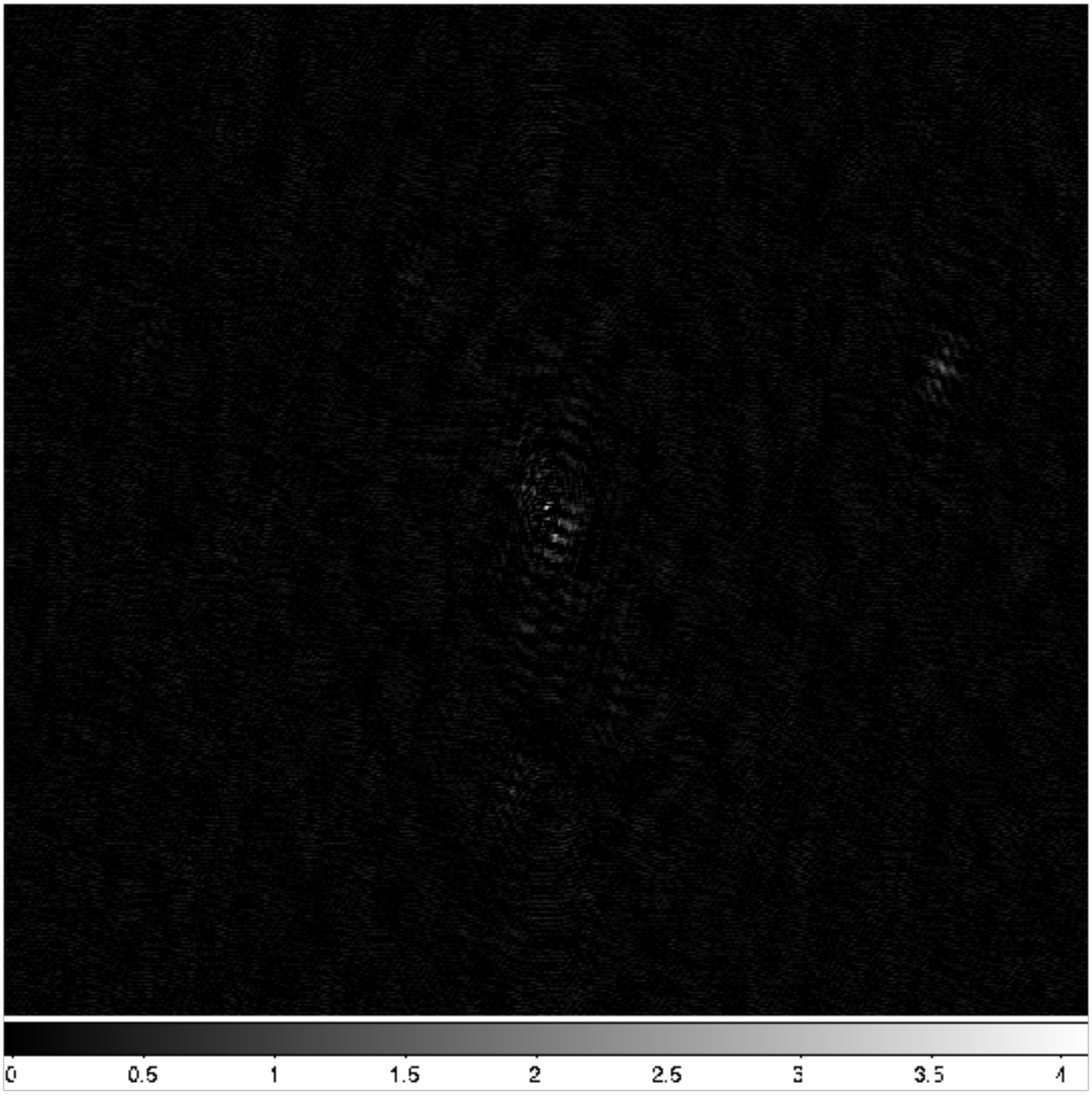


Fig. 12.12: Residual image of Virgo-A. An off center source is present on top right hand corner.

CHAPTER

THIRTEEN

SAGECAL[1]

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

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

13.2 Using SAGECAL

13.2.1 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

[1] The authors of this chapter are Emanuela Orru and Sarod Yatawatta.

on averaged data(!). Another procedure you will need apply to your data before SAGECAL is to calibrate them in the standard way with BBS, solving for the four G Jones elements as well as for the element beam. After BBS, you will need to flag the outliers using DPPP. You are now ready to start the algorithm.

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

13.2.2 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 Sky model construction using Shapelets). Note that you are free to use any other source finder (like PyBDSF) 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. 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.

13.2.3 SAGECAL

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

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

This will read the data from the DATA column of the MS and write the calibrated data to the CORRECTED_DATA column. If these columns are not present, you have to create them first.

If you need to calibrate more than one MS together, first create a text file with all the MS names, line by line. Then run

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

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

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

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

Use a solution interval (e.g. -t 120) that is big enough to get a decent solution and not too big to make the parameters vary too much (about 20 minutes per solution is a reasonable value).

In case of both bright and faint sources in the model, you might need to use different solution intervals for different clusters. In order to do that you need to define the values of the second column of the cluster file in such a way that the cluster with the longest solution interval is 1. While the cluster with shorter solution interval will be equal to n, where n is the number of times the longer solution interval is divided. This means that if -t 120 is used to select 120 timeslots, cluster 0 will find a solution using the full 120 timeslots, while cluster 1 will solve for every 120/3=40 timeslots. The option -k will allow to correct the residuals using the solutions calculated for a specific direction which is defined by the **cluster_id**. The -k option is analogue to the correct step in BBS. See the simulations section later in this chapter for more detailed use of this.

You are now ready to image the residual data. Successively, run "restore" (see Sky model construction using Shapelets) 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.

13.2.4 Robustness

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

13.2.5 Simulations

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

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

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

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

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

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

13.3 Distributed calibration

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

13.3.1 MPI

Using MPI, you can run a program across a network of computers. The basic way to run any program with MPI is

```
mpirun -np N -hostfile machines.txt (your program) (program options)
```

where **-np** gives the number of processes (can be different from the number of computers) and **-hostfile** gives the computer names to use (**machines.txt** includes the computer names to use). Always recommended to use the full path for the program to run. If you are only using one computer, you can skip the hostfile option.

13.3.2 Initial setup

Let's say there are 10 subbands, **SB1.MS**, **SB2.MS**, to **SB10.MS** in one computer. First you need to create a text file (say **mslist.txt**)

```
SB1.MS
SB2.MS
...
SM10.MS
```

that includes all the MS names, one per each line. Since we are running on one computer, no need to create a hostfile.

13.3.3 SAGECAL options

Apart from the usual command line options used in running stand alone SAGECal, there are a few special options for running it in a disributed way.

- The number of ADMM iterations is given by **-A** option.
- The type of polynomial in freqyency to use as a regularizer is given by **-Q** option.
- The number of freqyency terms in the polynomial is given by **-P** option.
- The regularization (penalty) factor is given by the **-r** option.

13.3.4 Running

Since there are 10 subbands, we need to invoke 10+1 SAGECal processes, this is done by selecting **-np 11**. The simplest way to run is

```
mpirun -np 11 /home/sarod/bin/sagecal-mpi -A 10 -P 2 -r 10  -s my_skymodel -c my_
↪clustering -t 120 -p my_solutions
```

this will calibrate each subband the usual way, but the solutions will be much better because information across the frequency range covered by the subbands is used. Note that almost all options used for standalone SAGECal can still be used here.

13.4 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.

CHAPTER

FOURTEEN

RUNNING LOFAR IMAGING PIPELINES INSIDE DOCKER[1]

The now standard LOFAR imaging pipelines (like prefactor and factor) depend on a variety of software packages and this can be hard for some users to install all the necessary software packages, and their dependencies on their machines. In this chapter, we present an overview of a docker image that comes pre-packaged with all the tools that a user would need to run the LOFAR pipelines. For other methods to install LOFAR software, see the LOFAR User Software section on the LOFAR wiki.

With this new docker framework[2], initializing your LOFAR processing framework can be as simple as

```
$ docker run --rm -it lofaruser/imaging-pipeline:latest
> NDPPP parsetfile
```

and frees the user (and sysadmins) from building all the required software packages. The following sections provide an overview of how to install and run LOFAR imaging pipelines using the docker.

Any questions concerning the docker image should be addressed to science operations and support. Note however that **this docker framework is experimental and support for it from the SOS group will be provided on a best effort basis**.

14.1 What is in the LOFAR docker image?

The docker image contains all the software packages that a user needs to run the LOFAR imaging pipelines like prefactor and factor. This includes the LOFAR offline software suite (containing tools like NDPPP and genericpipeline) along with other external tools like LoSoTo, AOFlagger, and WSClean. All packages included in this docker image are listed below:

- Casacore (version 2.4.1 including measures data)
- Casarest
- python-casacore (version 2.2.1)
- AOFlagger (version 2.12.1)
- Source finder pyBDSF
- LOFAR software (version 3.2.1)
- WSClean (version 2.6 including support for LOFAR primary beam)
- LoSoTo (release 2.0)
- RMextract

[1] This chapter is maintained by S. Sridhar.

[2] Docker is an opensource platform that makes use of container technology to create, deploy, and run applications easily. Detailed information about docker can be found here and elsewhere on the internet.

- LSMTool
- Dysco
- Prefactor (version 2.0.3)
- Factor (version 1.3)
- Python packages (including numpy, scipy, matplotlib)

14.2 Installing and setting-up docker

Docker is supported on various operating systems. Detailed instructions on how to install docker for different operating systems and cloud computing services can be found online.

In addition to installing the docker client, your system administrator should also add your username to the user group "docker". This can be achieved using the command "**sudo usermod -a -G docker <your username>**". Note that all users who wish to make use of the docker image on your machine must be added to the "docker" user group.

Once your sysadmin has setup the docker client on your machine and has added you to the docker group, you can download the LOFAR docker image using the command

```
docker pull lofaruser/imaging-pipeline:latest
```

14.2.1 Example: how to install and run docker on Fedora 27

On a fresh installation of Fedora, you need the following steps to install and run the docker image. Note that you need to have sudo privilege to carry out the steps listed below:

- Setup the repository using the commands "**sudo dnf install dnf-plugins-core**" and "**sudo dnf config-manager –add-repo https://download.docker.com/linux/fedora/docker-ce.repo**".
- Install the docker client with "**sudo dnf install docker-ce**".
- Start the docker service with "**sudo systemctl start docker**" and "**sudo systemctl enable docker**".
- Add your user to the docker group: "**sudo usermod -a -G docker <your username>**". Logout and login to apply the changes to your user.
- Run the command "**docker run –rm hello-world**" to verify your docker installation. If you see something like the following in your terminal, you have successfully installed docker on your machine.

```
$ docker run --rm hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
ca4f61b1923c: Pull complete
Digest: sha256:97ce6fa4b6cdc0790cda65fe7290b74cfebd9fa0c9b8c38e979330d547d22ce1
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
```

(continues on next page)

(continued from previous page)

```
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://cloud.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/engine/userguide/
```

- Now, say, for example, you (as the sysadmin) want to provide access to a new user (say lof01). You would do "**sudo adduser lof01**" to create the user. Once the user has been created, you need to add lof01 to the usergroup docker using the command "**sudo usermod -a -G docker lof01**".

Once this is done, the user lof01 should be able to run the docker images.

14.3 What happens when I run the docker image?

The LOFAR docker image can be run using the following command

```
docker run --rm -it -v /home/temp/Data:/opt/Data lofaruser/imaging-pipeline:latest
```

As mentioned in the previous section, you can replace **lofar:latest** with **lofar:<version>** to run an older image. When the above command is run in a terminal, docker

- creates an interactive session and attaches it to the terminal (as indicated by the command line flag '-it' in the above command),
- initializes the LOFAR software environment, and
- maps the directory /home/temp/Data on your host machine to the directory /opt/Data/ inside the container.

Now, you (as **root** inside the container) can execute commands like NDPPP or wsclean inside the mapped directory (in this case /opt/Data/). If you do not want to be **root** inside the container, you can map the user on the host to the container as

```
docker run -u `id -u`:`id -g` -v /etc/passwd:/etc/passwd:ro -v /etc/group:/etc/
↪group:ro -e USER -v $HOME:$HOME -e HOME -w $HOME -e DISPLAY --net=host --rm␣
↪lofaruser/imaging-pipeline:latest
```

If you run the command **id**, you should see that the UID and GID of your user inside the container should be the same as that on the host.

14.4 How to run prefactor inside the docker container?

In this section, we will demonstrate how to run the prefactor pipelines inside the docker container. For this example, we will follow the steps needed to run the calibrator part of prefactor, but the steps should be similar for the other pipelines that are part of prefactor. In this case, we will assume that the pipeline will be run from the directory /home/sarrvesh/dockertest/ and that all required measurement sets are available in /home/sarrvesh/dockertest/calibrator/.

Now, start the docker container using the command

```
$ docker run -it -e USER -v $HOME/dockertest:$HOME/dockertest -e HOME -w $HOME --rm␣
↪lofaruser/imaging-pipeline:latest
```

Once inside the container, you need source

```
$ source /lofarsoft/lofarinit.sh
```

Navigate to the working directory /home/sarrvesh/dockertest

```
$ cd /home/sarrvesh/dockertest
$ ls -l calibrator
total 160
drwxr--r-- 17 9322 9000 4096 Apr  9  2018 L640803_SAP000_SB000_uv.MS
drwxr--r-- 17 9322 9000 4096 Apr  9  2018 L640803_SAP000_SB001_uv.MS
drwxr--r-- 17 9322 9000 4096 Apr  9  2018 L640803_SAP000_SB002_uv.MS
drwxr--r-- 17 9322 9000 4096 Apr  9  2018 L640803_SAP000_SB003_uv.MS
drwxr--r-- 17 9322 9000 4096 Apr  9  2018 L640803_SAP000_SB004_uv.MS
drwxr--r-- 17 9322 9000 4096 Apr  9  2018 L640803_SAP000_SB005_uv.MS
drwxr--r-- 17 9322 9000 4096 Apr  9  2018 L640803_SAP000_SB006_uv.MS
drwxr--r-- 17 9322 9000 4096 Apr  9  2018 L640803_SAP000_SB007_uv.MS
drwxr--r-- 17 9322 9000 4096 Apr  9  2018 L640803_SAP000_SB008_uv.MS
drwxr--r-- 17 9322 9000 4096 Apr  9  2018 L640803_SAP000_SB009_uv.MS
drwxr--r-- 17 9322 9000 4096 Apr  9  2018 L640803_SAP000_SB010_uv.MS
drwxr--r-- 17 9322 9000 4096 Apr  9  2018 L640803_SAP000_SB011_uv.MS
drwxr--r-- 17 9322 9000 4096 Apr  9  2018 L640803_SAP000_SB012_uv.MS
drwxr--r-- 17 9322 9000 4096 Apr  9  2018 L640803_SAP000_SB013_uv.MS
drwxr--r-- 17 9322 9000 4096 Apr  9  2018 L640803_SAP000_SB014_uv.MS
drwxr--r-- 17 9322 9000 4096 Apr  9  2018 L640803_SAP000_SB015_uv.MS
drwxr--r-- 17 9322 9000 4096 Apr  9  2018 L640803_SAP000_SB016_uv.MS
drwxr--r-- 17 9322 9000 4096 Apr  9  2018 L640803_SAP000_SB017_uv.MS
drwxr--r-- 17 9322 9000 4096 Apr  9  2018 L640803_SAP000_SB018_uv.MS
drwxr--r-- 17 9322 9000 4096 Apr  9  2018 L640803_SAP000_SB019_uv.MS
drwxr--r-- 17 9322 9000 4096 Apr  9  2018 L640803_SAP000_SB020_uv.MS
drwxr--r-- 17 9322 9000 4096 Apr  9  2018 L640803_SAP000_SB021_uv.MS
drwxr--r-- 17 9322 9000 4096 Apr  9  2018 L640803_SAP000_SB022_uv.MS
drwxr--r-- 17 9322 9000 4096 Apr  9  2018 L640803_SAP000_SB023_uv.MS
drwxr--r-- 17 9322 9000 4096 Apr  9  2018 L640803_SAP000_SB024_uv.MS
drwxr--r-- 17 9322 9000 4096 Apr  9  2018 L640803_SAP000_SB025_uv.MS
drwxr--r-- 17 9322 9000 4096 Apr  9  2018 L640803_SAP000_SB026_uv.MS
drwxr--r-- 17 9322 9000 4096 Apr  9  2018 L640803_SAP000_SB027_uv.MS
drwxr--r-- 17 9322 9000 4096 Apr  9  2018 L640803_SAP000_SB028_uv.MS
drwxr--r-- 17 9322 9000 4096 Apr  9  2018 L640803_SAP000_SB029_uv.MS
drwxr--r-- 17 9322 9000 4096 Apr  9  2018 L640803_SAP000_SB030_uv.MS
drwxr--r-- 17 9322 9000 4096 Apr  9  2018 L640803_SAP000_SB031_uv.MS
drwxr--r-- 17 9322 9000 4096 Apr  9  2018 L640803_SAP000_SB032_uv.MS
drwxr--r-- 17 9322 9000 4096 Apr  9  2018 L640803_SAP000_SB033_uv.MS
drwxr--r-- 17 9322 9000 4096 Apr  9  2018 L640803_SAP000_SB034_uv.MS
drwxr--r-- 17 9322 9000 4096 Apr  9  2018 L640803_SAP000_SB035_uv.MS
drwxr--r-- 17 9322 9000 4096 Apr  9  2018 L640803_SAP000_SB036_uv.MS
drwxr--r-- 17 9322 9000 4096 Apr  9  2018 L640803_SAP000_SB037_uv.MS
drwxr--r-- 17 9322 9000 4096 Apr  9  2018 L640803_SAP000_SB038_uv.MS
drwxr--r-- 17 9322 9000 4096 Apr  9  2018 L640803_SAP000_SB039_uv.MS
```

Next, we will create the working and runtime directories needed to run prefactor. Also, copy the **pipeline.cfg** file to the current directory.

```
$ mkdir working runtime
$ cp /lofarsoft/share/pipeline/pipeline.cfg .
```

Edit the pipeline.cfg file so that the keys runtime_directory, working_directory, and log_file point to the correct locations inside the container.

```
runtime_directory = /home/sarrvesh/dockertest/runtime/
recipe_directories = [%(pythonpath)s/lofarpipe/recipes,/lofarsoft/prefactor-2.0.3/]
working_directory = /home/sarrvesh/dockertest/working/
log_file = /home/sarrvesh/dockertest/log/pipeline-%(job_name)s-%(start_time)s.log
xml_stat_file = /home/sarrvesh/dockertest/log/pipeline-%(job_name)s-%(start_time)s-
↪statistics.xml
```

You should also add the following lines to the end of the pipeline.cfg file

```
[remote]
method = local
```

Next, copy the Pre-Facet-Calibrator.parset

```
$ cp /lofarsoft/prefactor-2.0.3/Pre-Facet-Calibrator.parset .
```

and edit the following keys in the parset

```
! cal_input_path       = /home/sarrvesh/dockertest/calibrator/
! calibrator_path_skymodel = /lofarsoft/prefactor-2.0.3/skymodels/
! inspection_directory     = /home/sarrvesh/dockertest/plots/
! cal_values_directory     = /home/sarrvesh/dockertest/cals/
! calib_cal_parset          = /lofarsoft/prefactor-2.0.3/parsets/calibcal.parset
! find_skymodel_cal_auto    = /lofarsoft/prefactor-2.0.3/scripts/find_skymodel_cal.py
! losoto_importer           = /lofarsoft/prefactor-2.0.3/scripts/losotoImporter.py
! fitclock_script           = /lofarsoft/prefactor-2.0.3/scripts/fit_clocktec_
↪initialguess_losoto.py
! fitamps_script            = /lofarsoft/prefactor-2.0.3/scripts/amplitudes_losoto_3.
↪py
! plotsols_script           = /lofarsoft/prefactor-2.0.3/scripts/examine_npys.py
! fit_XYoffset_script       = /lofarsoft/prefactor-2.0.3/scripts/find_cal_global_
↪phaseoffset_losoto.py
! plotphases_script         = /lofarsoft/prefactor-2.0.3/scripts/plot_solutions_all_
↪stations.py
! losoto_executable         = /lofarsoft/bin/losoto
```

Note that the latest version of the prefactor parset is available inside the container in the /lofarsoft directory.

Finally, run the Pre-Facet-Calibrator.parset using genericpipeline as

```
$ genericpipeline.py -c pipeline.cfg Pre-Facet-Calibrator.parset
```

14.5 How to import the docker image inside singularity?

The docker image discussed in this chapter can be imported and converted into a singularity image using the command

```
singularity build lofar-pipeline.simg docker://lofaruser/imaging-pipeline:latest
```

This will create a new file called lofar-pipeline.simg. You can execute the container with

```
singularity run ./lofar-pipeline.simg
```

Once inside the container, you should source

```
source /lofarsoft/lofarinit.sh
```

14.6 FAQ

Where are the default RFI strategies stored in the docker?

The default RFI strategies (LBAdefault and HBAdefault) are stored in the directory /lofarsoft/share/rfistrategies

Where can I find the pipeline.cfg file inside the docker container?

The pipeline.cfg file that is needed for run prefactor is stored in the directory /lofarsoft/share/pipeline

I get an "Illegal instruction error" when I run the docker image. What does this mean?

This probably means that you are running an older/incompatible CPU. A docker image might have to built on your machine. Please contact Science Operations & Support for further assistance if you encounter this.

CHAPTER

FIFTEEN

THE AW IMAGER[1]

15.1 Introduction

In this section we will describe the necessary steps needed to perform successful imaging of LOFAR data using the AWimager[2].

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

15.2 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.

15.3 Usage

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

```
module load lofar
```

Before running AWimager, it is necessary to calibrate the dataset and correct the visibilities towards the phase center of the observation. This can now be done by not specifying any direction in the **correct** step of BBS.

```
Step.correct.Model.Sources = []
```

AWimager can run in a parallel fashion. The number of processing cores (**n**) to be used during imaging can be specified:

```
export OMP_NUM_THREADS=n
```

If not specified, all cores will be used. AWimager is quite memory hungry, so the number of cores should be limited in case it fails due a **'bad alloc'** error.

[1] This chapter is maintained by Bas van der Tol.

[2] Cyril Tasse, Bas van der Tol, Joris van Zwieten, Ger van Diepen, Sanjay Bhatnagar 2013, **Applying full polarization A-Projection to very wide field of view instruments: An imager for LOFAR}. A&A, 553, A105**.

15.4 Output files

AWimager creates several image output files. Note that in the following list **<image>** is the image name given using the **image** parameter.

- **<image>.model** is the uncorrected dirty image.
- **<image>.model.corr** is the dirty image corrected for the average primary beam.
- **<image>.restored** and **<image>.restored.corr** are the restored images.
- **<image>.residual** and **<image>.residual.corr** are the residual images.
- **<image>.psf** is the point spread function.
- **<image>0.avgpb** is the average primary beam.

Furthermore, a few other files might be created for AWimager's internal use.

15.5 Input 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
```

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.

15.5.1 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.

15.5.2 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.

15.5.3 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.

15.5.4 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.

15.5.5 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

15.5.6 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 commissioning reports (specifically those from September 28 and October 26 2011).

CHAPTER

SIXTEEN

PRACTICAL EXAMPLES[1]

In this Chapter, examples of how to inspect and analyse 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 that the user is working on the CEP3 cluster and is familiar with the working environment.

The tutorial data can be staged and downloaded from the LOFAR Long Term Archive. Using the project link on top of the page, select "other projects" in the list of links which will appear, and click on the "LOFARSCHOOL" link in the list of projects. Then, click on the "Show Latest" link at the top of the page, and on "All observations and pipelines".

Selecting the checkbox in each row and clicking on the "stage selected" link will stage the data and you will get an email with the download instructions. This assumes that you have an user account registered. More info on how to use the LTA interface can be found on the LOFAR wiki.

Sky models and parset files used in the tutorial can be found at the LOFAR GitHub repository.

16.1 Flagging, averaging, and demixing

The naming convention for each measurement set (MS, also referred to as a sub-band) is Laaaaaa_SAPbbb_SBccc_uv.MS, where Laaaaaa is the observation/pipeline ID, SAPbbb is the sub-array pointing (beam), andSBccc is the sub-band number. We can, for example, obtain a summary of a measurement set using msoverview; for more detailed report, we use the "verbose" argument to the command:

```
msoverview in=L456104_SAP000_SB001_uv.MS verbose=T
```

Along with the details of the MS, we get a message: "This is a raw LOFAR MS (stored with LofarStMan)", which means that the data cannot be handled with Casa. To fix this, we can use the following DPPP parset:

```
msin=L456104_SAP000_SB001_uv.MS
msout=L456104_SAP000_SB001_uv_copy.MS
msin.autoweight=True
numthreads=4 # so that the nodes are not overloaded, usually can skip this line
steps = [ ]
```

after execution, the output MS can be opened using the standard CASA tools. Moreover, proper weighting of the data has been implemented.

The data can be inspected using casaplotms[2] as

[1] The author of this Chapter is Wendy Williams. Contribution was also given by many commissioners: Alicia Berciano Alba, Valentina Vacca, Poppy Martin, Maciej Ceglowski, Carmen Toribio, Emanuela Orru, Andre Offringa and Neal Jackson.

[2] Note that you should load the casa module before loading the lofar module on CEP3 for them to function as intended.

```
module load casa
casaplotms &
```

Open the measurement set from the GUI. Plot the XX correlation only by using the 'corr' argument (to speed things up). Select only cross correlations by typing "&" in the 'antenna' field (note: * - any antenna; & - cross correlations). There are many large spikes, likely caused by RFI. Click on the 'Axes' tab and adjust the y-axis range to something more sensible to find the real astronomical signal (Fig. 16.1).

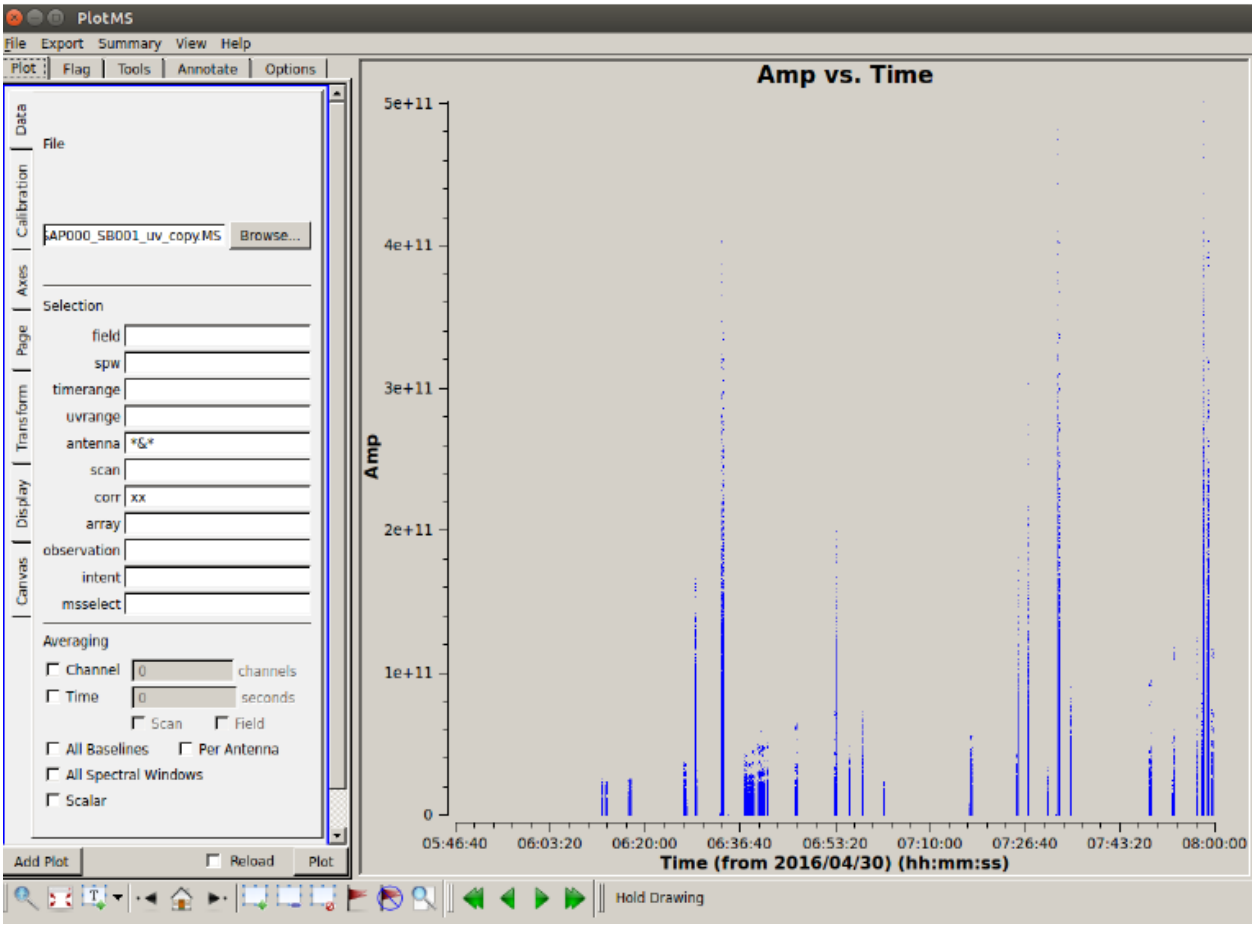


Fig. 16.1: Inspecting the visibilities in casaplotms.

We can also further visualize the impact of RFI in the measurement set using rfigui.

```
rfigui L456104_SAP000_SB001_uv_copy.MS &
```

In the smaller menu that pops up, just choose "Open". Then, in the new window, select index 0 (default) for antenna 1, and index 1 for antenna 2, before clicking on "Load". This will plot the short intra-station baseline CS001HBA0 - CS001HBA1. There is some clear narrow-band RFI. You can create a power spectrum (using "Plot"). rfigui can flag the data using AOFlagger: "Actions", then "Execute Strategy". There are other useful plots in the plot menu.

The AOFlagger can be called with DPPP. Create and run this parset (you can pipe the output to a log file, e.g. **DPPP your.parset | tee DPPP.log**

```
msin=L456104_SAP000_SB001_uv_copy.MS
msout=L456104_SAP000_SB001_uv_copy_flg.MS
```

(continues on next page)

(continued from previous page)

```
numthreads=4
steps=[preflagger,aoflagger]
preflagger.baseline=*&&&;RS208HBA;RS210HBA;RS310HBA;RS406HBA;RS407HBA;RS409HBA;
RS508HBA;RS509HBA;RS205HBA&RS307HBA # all these stations on one line
```

An explanation of the "preflagger" step is as follows. The first argument (arguments separated by semi-colons) removes the autocorrelations (although note that AOFlagger also does this by default). For the next eight arguments, we remove all baselines including the stated remote station as well as one particular baseline between two remote stations (identified as bad data which can affect the calibration). The default AOFlagger strategy is run, but many options can be specified in DPPP (see the LOFAR wiki). We can re-examine the data with casaplotms and see that the RFI has been removed.

The observation we are working on is close to two "A-team" sources: about 16 deg from Cyg A, and 21 deg from Cas A. We need to check if their influence should be "demixed" from the recorded visibilities. Firstly, let's check their elevations during the observation:

```
plot_Ateam_elevation.py L456104_SAP000_SB001_uv_copy_flg.MS
```

We can predict the associated visibilities through simulations. Or, we will run a tool called the Drawer, allowing one to quickly inspect a measurement set and investigate which sources are contributing to the visibilities.

```
/home/tasse/drawMS/drawMS --ms=L456104_SAP000_SB001_uv_copy_demix_demo.MS
```

Note that we are running the Drawer on pre-prepared averaged data (to save time); we will learn how to do averaging shortly. Examine the Drawer output. One can see contributions from Cas A and Cyg A. We need to demix their influence from the data.

We need a model of the A-team sources:

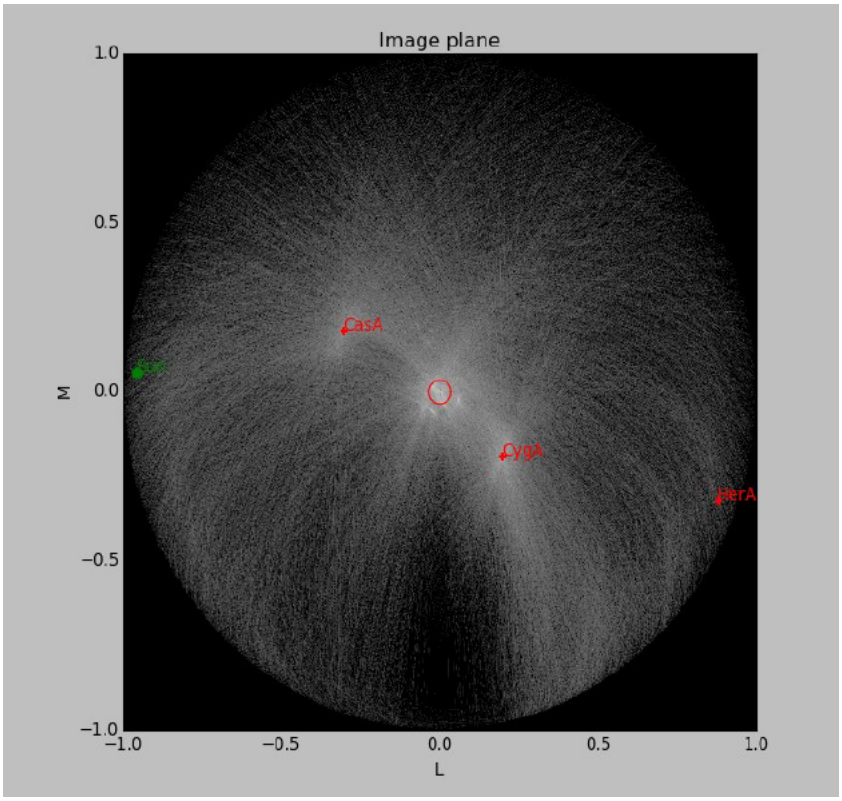
```
showsourcedb in=Ateam.sourcedb mode=patch # verify database contents
```

Use the following parset for DPPP:

```
msin=L456104_SAP000_SB001_uv_copy_flg.MS
msout=L456104_SAP000_SB001_uv_copy_flg_demix_avg.MS
numthreads=4
steps=[demix]
demix.subtractsources=[CasA,CygA]
demix.skymodel=Ateam.sourcedb
demix.timestep=10
demix.freqstep=16
demix.demixtimestep=60
demix.demixfreqstep=64
```

The arguments "timestep" and "freqstep" compress the data in time and frequency, respectively, by the given factors. Demixing two sources at once can be very time consuming. Thus, "demixtimestep" and "demixfreqstep" have been chosen to have rather coarse values (usually default to "timestep" and "freqstep" without needing to be specified).

We can draw the demixed data to see the result:



What if demixing is not needed? We can also just average in time and frequency. Here is an example parset that could be run through DPPP:

```
msin=L456104_SAP000_SB001_uv_copy_flg.MS
msout=L456104_SAP000_SB001_uv_copy_flg_avg.MS
numthreads=4
steps=[averager]
averager.timestep=10
averager.freqstep=16
```

DPPP was designed to run pipelines containing multiple steps. The following parset combines all the steps run so far in this tutorial, except demixing:

```
msin=L456104_SAP000_SB001_uv.MS
msin.autoweight=true
msout=L456104_SAP000_SB001_uv.MS.dppp
numthreads=4
steps=[preflagger,aoflagger,averager]
preflagger.baseline=*&&&;RS208HBA;RS210HBA;RS310HBA;RS406HBA;RS407HBA;RS409HBA;
RS508HBA;RS509HBA;RS205HBA&RS307HBA
averager.timestep=10
averager.freqstep=16
```

16.2 Calibration

You will need to stage and download the calibrator data from the LOFAR LTA with the Observation ID (SAS ID): 456102. Place these in a folder named “calibrator”. The target data can be staged and downloaded in the same manner. Their observation ID is: 456106. Place the target data in a folder named “target”. You can inspect them and if needed,

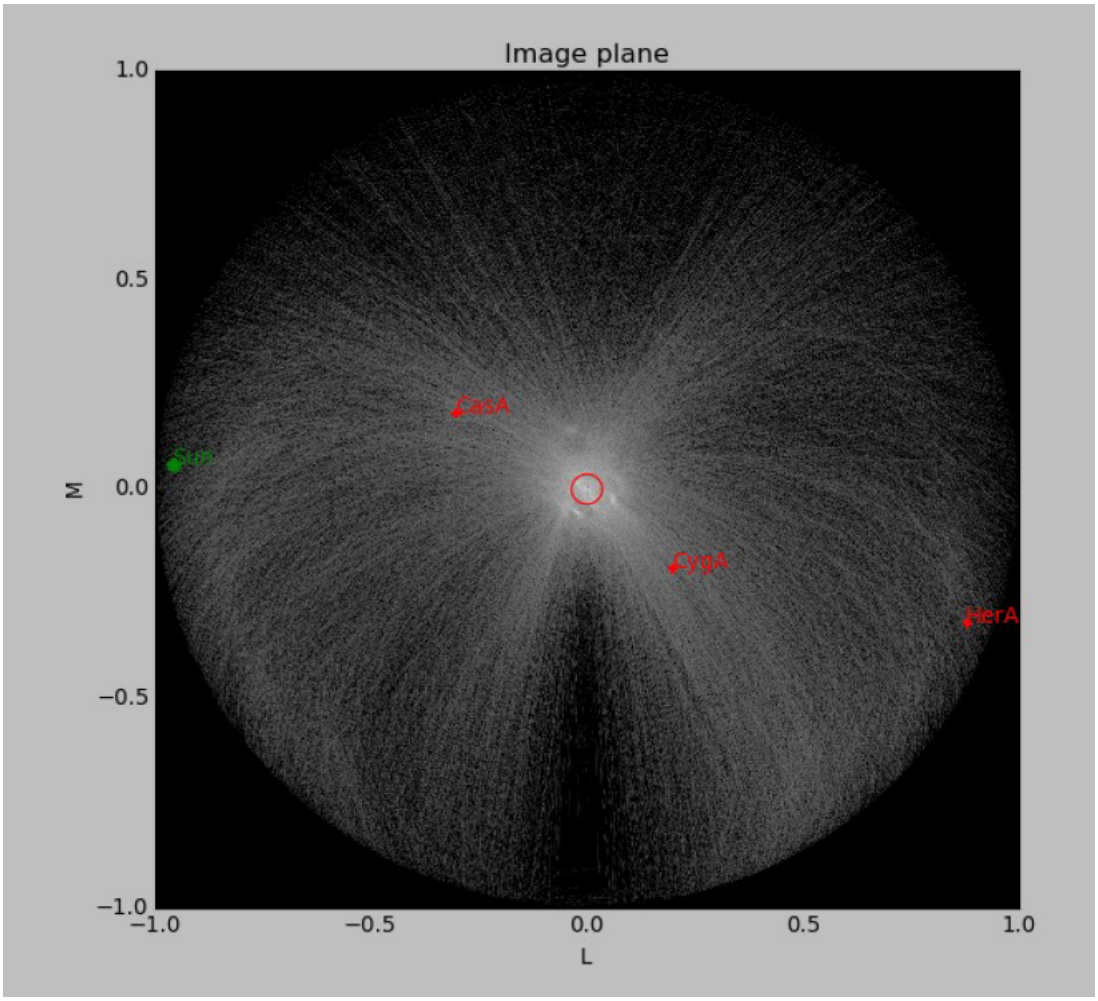


Fig. 16.2: A Drawer plot of the data before and after demixing.

flag outliers in a manner analogous to the procedure given in the previous sub-section. You do not need to process the complete target data set. For example, a subset of 5 SBs can be selected for calibration.

Take the **A-Team_lowres.skymodel** file, place it in the top directory, and run:

```
module load Lofar
makesourcedb in=A-Team_lowres.skymodel out=A-Team_lowres.sourcedb format="<"
```

to create a sky-model in **sourcedb** format.

Move to the calibrator folder, copy the model there and create a **predict_model.parset** file with the contents:

```
msin =
msin.datacolumn = DATA
msin.baseline = [CR]S*&
msout = .
msout.datacolumn = MODEL_DATA
numthreads = 5
steps = [predict]
predict.type=predict
predict.sourcedb=A-Team_lowres.sourcedb
predict.sources=CygA
predict.usebeammodel=True
```

Predict the model data for all of the calibrator sub-bands by executing the following bash script:

```
for i in *MS; do NDPPP predict_model.parset msin=$i msout=$i; done
```

Create the parset we will use for calibration, **gaincal.parset**

```
msin=
msout=.
msin.datacolumn = DATA
msin.baseline = [CR]S*&
msout.datacolumn = CORRECTED_DATA
numthreads = 5
steps=[gaincal]
#gaincal.sourcedb=A-Team_lowres.sourcedb
#gaincal.sources = CygA
#gaincal.usebeammodel=true
gaincal.usemodelcolumn=true
gaincal.parmdb=
gaincal.type=gaincal
gaincal.caltype=diagonal
```

and run DPPP for all the calibrator sub-bands

```
for i in *MS.flg; do NDPPP gaincal.parset msin=$i gaincal.parmdb=$i/instrument msout=
↪$i; done
```

Collect the solutions for all the sub-bands together using:

```
#!/bin/bash
# To copy the instrument tables in a globaldb
mkdir globaldb
i=0
for ms in `ls -d *MS`; do
echo "Copying ${ms}/instrument"
```

(continues on next page)

(continued from previous page)

```
cd $ms
# copy other tables
if [ $i == 0 ]; then
cp -r ANTENNA ../globaldb
cp -r FIELD ../globaldb
cp -r sky ../globaldb
fi
cp -r instrument ../globaldb/instrument-$i
cd ..
i=$((i + 1))
done
```

Then, transform the global solutions into **.h5** format (suitable for LoSoTo):

```
module load losoto
H5parm_importer.py -v cal.h5 globaldb
```

Plot, flag and merge the solutions:

```
losoto -v cal.h5 losoto.parset
```

using the LoSoTo parset:

```
#losoto parset
LoSoTo.Steps = [plotP1, plotP2, plotP3, plotA1, plotA2, plotA3, flag, flagextend,␣
↪merge]
```

Then, export the merged solutions:

```
H5parm_exporter.py -v cal.h5 globaldb
```

and copy back the solutions to the individual calibrator measurement sets:

```
#!/bin/bash
# copy back the instrument tables from a globaldb
# to be run after H5parm_exporter.py
i=0
for ms in `ls -d *MS`; do
echo "Copying back ${ms}/instrument"
rm -r ${ms}/instrument
cp -r globaldb/sol000_instrument-$i ${ms}/instrument
i=$((i + 1))
done
```

Now, we need to transfer the solutions from the calibrator to the target. We have to re-write the solutions we have found so that they are time independent, since the target was observed at a different time:

```
for i in *MS.flg; do parmexportcal in=$i/instrument out=$i/instrument_tind; done
```

then, we can transfer (apply) the solutions to the target using the following parset:

```
msin =
msin.datacolumn = DATA
msin.baseline = [CR]S*&
msout = .
msout.datacolumn = CORRECTED_DATA
```

(continues on next page)

(continued from previous page)

```
numthreads = 5
steps = [applycal, applybeam]
applycal.type = applycal
applycal.correction = gain
applycal.parmdb =
applybeam.type = applybeam
applybeam.invert = True
```

which is needed by DPPP and executed for the complete calibrator and target sub-band list using following Python script (for example):

```
import os
for i in range(0,29,1):
   if i <10:
          calib_instrument = 'calibrator/L456102_SB00'+str(i)+'_uv.dppp.MS/instrument_
↪tind'
      targetMS = 'target/L456106_SB00'+str(i)+'_uv.dppp.MS'
       else:
      calib_instrument= 'calibrator/L456102_SB0'+str(i)+'_uv.dppp.MS/instrument_tind'
      targetMS = 'target/L456106_SB0'+str(i)+'_uv.dppp.MS'
#
       print 'NDPPP appycal.parset msin='+str(targetMS)+' applycal.parmdb='+str(calib_
↪instrument)+''
       os.system('NDPPP applycal.parset msin='+str(targetMS)+' applycal.parmdb=
↪'+str(calib_instrument)+'')
```

Finally, after the flux scale has been set, the target data need to be calibrated (phase only). We need a source model of the target field:

```
cd target/
gsm.py [-p patchname] outfile RA DEC radius [vlssFluxCutoff[assocTheta]]
gsm.py -p P1 P1.sky 315.00000000 52.90000000 5
makesourcedb in=P1.sky out=P1.sourcedb format="<"
```

and, using the following parset:

```
msin=
msout=
msin.datacolumn = CORRECTED_DATA
msin.baseline = [CR]S*&
msout.datacolumn = DATA
numthreads = 5
steps=[gaincal]
gaincal.sourcedb=P1.sourcedb
gaincal.sources =
gaincal.parmdb=
gaincal.type=gaincal
gaincal.caltype=phaseonly
gaincal.usebeammodel=false
gaincal.applysolution=True
```

we calibrate the target data:

```
for i in *flg; do NDPPP ../phaseonly.parset msin=$i msout=$i.ph gaincal.parmdb=$i.ph/
↪instrument; done
```

The last line in the parset applies the solutions to the data. Alternatively, we can omit it, and apply the solutions using

applycal in DPPP with the following parset:

```
msin =
msin.datacolumn = DATA
msin.baseline = [CR]S*&
msout = .
msout.datacolumn = CORRECTED_DATA
steps = [applycal]
applycal.type = applycal
applycal.parmdb =
```

16.3 Imaging

The Wsclean imager will be used for imaging. On CEP3, you can start it as: **module load Wsclean**. You can check the versioning: **wsclean –version**, or get to the command line help: **wsclean**

First, pick a random sb and run wsclean as follows:

```
wsclean -size <width> <height> -scale <val>asec \
-name quick L456106_SB010_uv.dppp.MS.ph
```

Change the sb number to your random sub-band number. Replace **width** and **height** by a number of pixels. **val** is the image resolution, here specified in asec. Determine good values for these for imaging this LOFAR set. You want to go a bit beyond the first beam null. Note that angularwidth $\sim$ width x scale.

Note that **val** and **asec** have no space between them, e.g.: **-scale 2.5asec**. Other units can be specified, e.g.: "6amin", "50masec", "0.1deg" In order to keep processing fast for this tutorial, don't make images $>$ 4k or wider than 20 deg. This quick imaging should not take more than $\sim$ 3 min. WSClean will always automatically perform appropriate w-correction (i.e., corrections necessary for wide-field imaging).

Example command:

```
wsclean -size 1400 1400 -scale 50sec \
-name quick L456106_SB010_uv.dppp.MS.ph
```

This will output "quick-dirty.fits" and "quick-image.fits". Inspect these with your favourite fitsviewer (e.g., kvis, ds9, casaviewer).

The main parameters for cleaning are:

- **-niter <count>** Turns cleaning on and sets max iterations. Normally, cleaning should end at the threshold, not at the max iterations.
- **-mgain <gain>** How much flux of the peak is subtracted before a major iteration is restarted. Depends on how good your beam is. 0.8 is safe, 0.9 almost always works and is faster.
- **-threshold <flux>** Set the apparent flux (in Jy) at which to stop. Should typically be 3 x σ.

Run the following command: (still on a single subband):

```
wsclean -size <width> <height> -scale <val>asec \
-niter <niter> -mgain 0.8 -threshold <flux> \
-name clean L456106_SB010_uv.dppp.MS.ph
```

For example:

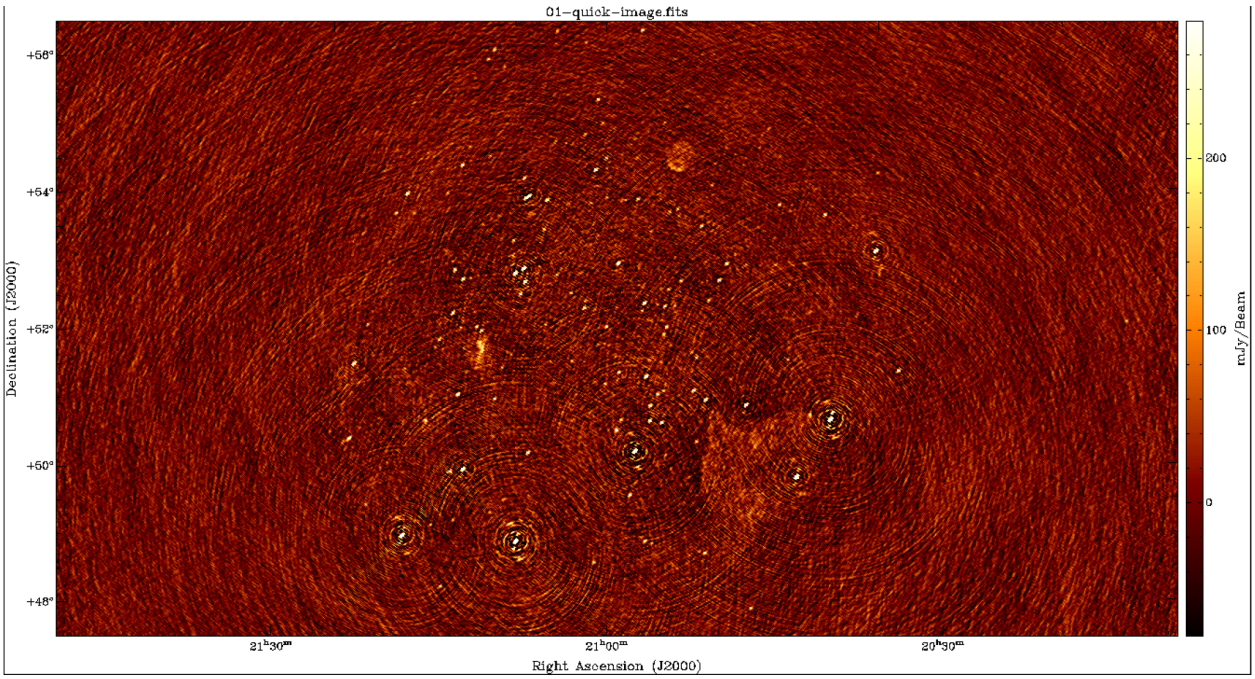


Fig. 16.3: Example image of a random SB.

```
wsclean -size 1400 1400 -scale 50asec \
-niter 50000 -mgain 0.8 -threshold 0.1 \
-name clean L456106_SB010_uv.dppp.MS.ph
```

It is convenient to store the above command in a shell script. Notice in the output the cleaning process:

```
== Cleaning (1) ==
Freed 222 image buffer(s).
Initial peak: 3.2568
Next major iteration at: 0.651359
Iteration 0: (602,465), 3.2568 Jy
[..]
Iteration 100: (731,561), 0.789584 Jy
Stopped on peak 0.646578
[..]
== Cleaning (2) ==
[..]
Stopped on peak 0.130435
[..]
== Cleaning (3) ==
Major iteration threshold reached global threshold of 0.1: final major iteration.
Iteration 2000: (545,542), 0.12621 Jy
Stopped on peak -0.0999906
```

The threshold is reached in 2000 iterations.

Example command:

```
wsclean -size 1400 1400 -scale 50asec \
-niter 50000 -mgain 0.8 -threshold 0.1 \
-name clean L456106_SB010_uv.dppp.MS.ph
```

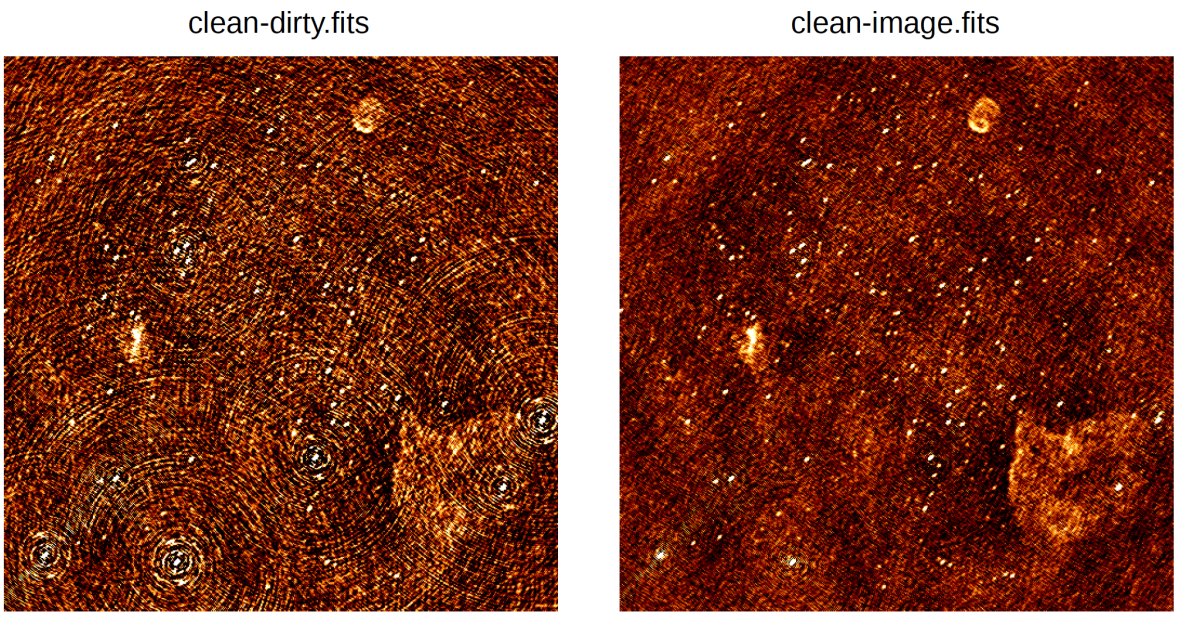


Fig. 16.4: Example of a cleaned vs. dirty image.

The LOFAR beam is applied by adding **-apply-primary-beam**. Note that the beam was already applied on the phase centre during calibration (the "applybeam" step in NDPPP). WSClean needs to know this, otherwise it will use the wrong beam. This is specified by also adding **-use-differential-lofar-beam**. Repeat the previous imaging with the beam, similar to:

```
wsclean -size <width> <height> -scale <val>asec \
-apply-primary-beam -use-differential-lofar-beam \
-niter <niter> -mgain 0.8 -threshold <flux> \
-name lofarbeam L456106_SB010_uv.dppp.MS.ph
```

For example:

```
wsclean -size 1400 1400 -scale 50asec \
-apply-primary-beam -use-differential-lofar-beam \
-niter 50000 -mgain 0.8 -threshold 0.1 \
-name clean L456106_SB010_uv.dppp.MS.ph
```

Read the documentation for **-weight**, **-taper-gaussian** and **-trim**, and optionally other weighting/tapering methods. Repeat the previous imaging, but with settings for these parameters that are useful to:

- accentuate the diffuse emission; and
- to make the beam Gaussian like, to measure the flux of the emission more easily.

Correct for the primary beam as before. Like:

```
wsclean -size <width> <height> -scale <val>asec \
-trim <trimwidth> <trimheight> \
-apply-primary-beam -use-differential-lofar-beam \
-niter <niter> -mgain 0.8 -threshold <flux> \
-weight [briggs <robustness> or natural] \
-taper-gaussian <val>amin \
-name clean L456106_SB010_uv.dppp.MS.ph
```

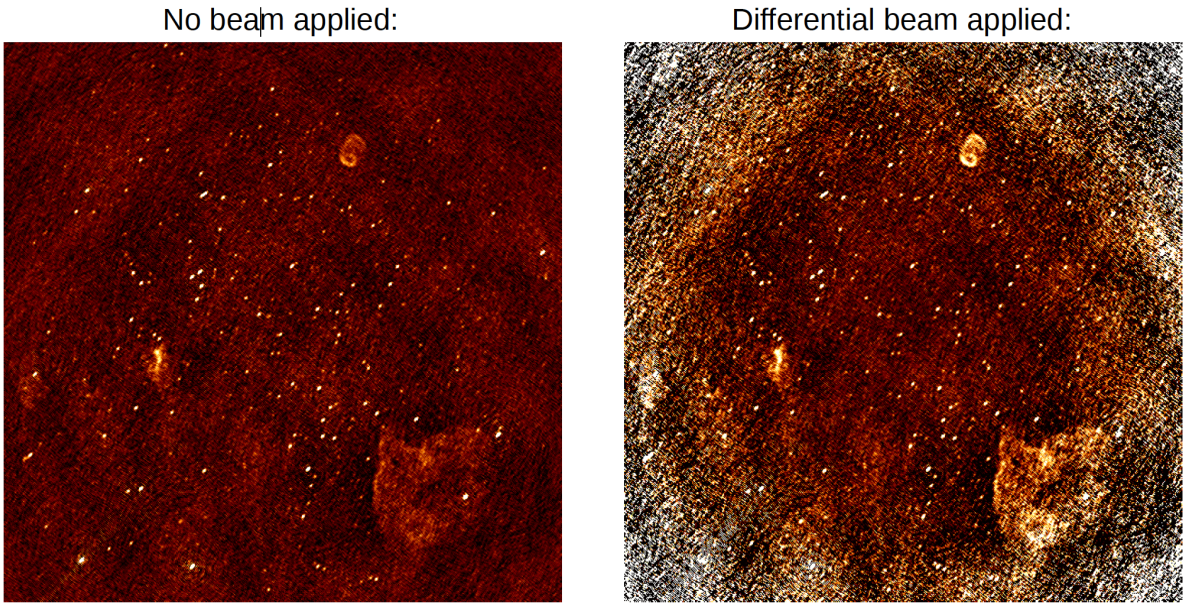


Fig. 16.5: Primary beam correction.

Example:

```
wsclean -size 1800 1800 -scale 50asec \
-trim 1400 1400 -weight briggs 0 \
-niter 50000 -mgain 0.8 -threshold 0.1 \
-name weighting L456106_SB010_uv.dppp.MS.ph
```

Note the negative areas around the diffuse sources. Inspect the "model" image. How did WSClean model the diffuse emission and SNRs? Repeat the previous imaging, but use multiscale. If you feel adventurous, you can play with **-multiscale-scales** and **-multiscale-scale-bias**. However, for LOFAR this is hardly ever necessary.

```
wsclean -size <width> <height> -scale <val>asec \
-trim <trimwidth> <trimheight> \
-apply-primary-beam -use-differential-lofar-beam \
-niter <niter> -mgain 0.8 -threshold <flux> \
-weight [your weighting choice] \
-taper-gaussian <val>amin \
-multiscale \
-name multiscale L456106_SB010_uv.dppp.MS.ph
```

Baseline-dependent averaging (only available for versions later than v1.12) lowers the number of visibilities that need to be gridded, which therefore speeds up the imaging. To enable it, one adds **-baseline-averaging** to the command line with the number of wavelengths (λ) that can be averaged over. Use this rule: λ = max baseline in $\lambda \times 2 \times \pi \times$ integration time in sec $/(24 * 60 * 60)$. Rerun the previous imaging with b.d. averaging. Turn beam correction off (it does not work together with b.d. averaging yet). Example:

```
wsclean -size 1800 1800 -scale 50asec \
-trim 1400 1400 -weight briggs 0 \
-multiscale \
-niter 100000 -mgain 0.8 -threshold 0.15 \
-baseline-averaging 2.0 -no-update-model-required \
-name bdaveraging L456106_SB010_uv.dppp.MS.ph
```

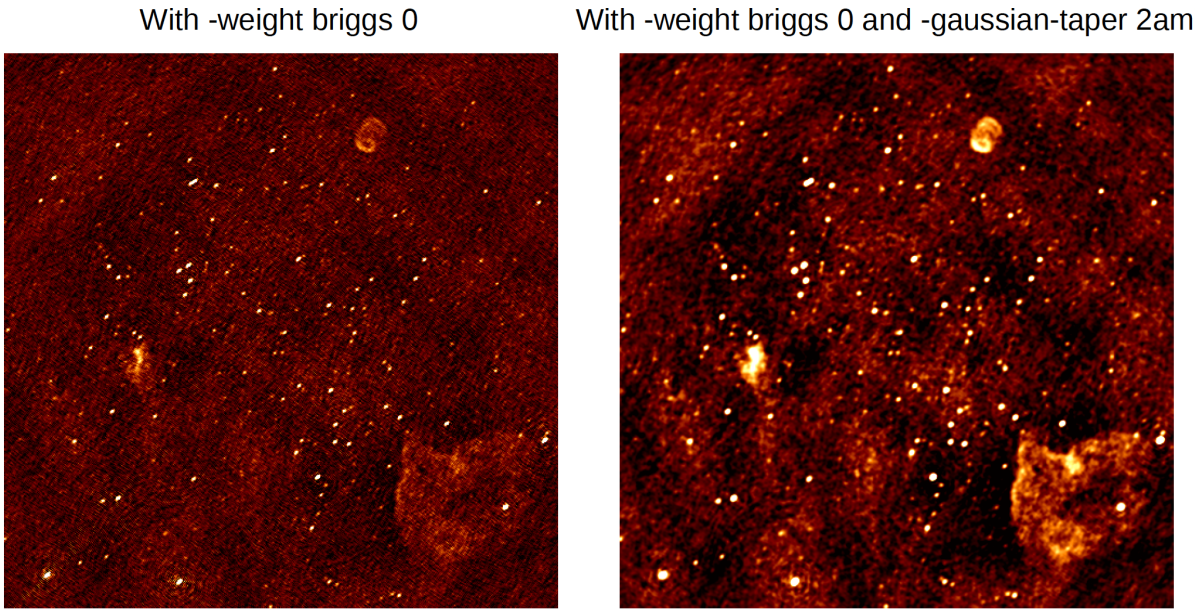


Fig. 16.6: Weighting.

Try a second run with more averaging and inspect the difference between the images. How much averaging is acceptable?

Several approaches are possible for combining all bands (i.e. measurement sets / SBs):

- Run WSClean on each band and combine images afterwards. Only limited cleaning possible.
- Image all MSes in one run with WSClean. Clean deep, but assumes flux is constant over frequency.

```
wsclean -size <width> <height> -scale <val>asec \
[..] \
-name fullbandwidth *.dppp.MS.flg.ph
```

This takes quite a lot of time. If you have time, you can run the command. You can also run it with only a few measurement sets. If you run clean on the full bandwidth, you can decrease the threshold significantly, because the system noise will go down by $\sqrt{N}$.

Image all SBs and use multi-frequency deconvolution. Cleans deep and incorporates frequency dependency. Relevant parameters: **-channelsout <count>**, **-joinchannels**, **-fit-spectral-pol <terms>**, **-deconvolution-channels <count>**. Like:

```
wsclean -size <width> <height> -scale <val>asec \
[..] \
-channelsout <count> -joinchannels \
-fit-spectral-pol <terms> \
-deconvolution-channels <count> \
-name mfclean *.dppp.MS.ph
```

Decrease the threshold to an appropriate level. Example command:

```
wsclean -size 1800 1800 -scale 50asec \
-apply-primary-beam -use-differential-lofar-beam \
-trim 1400 1400 -weight briggs 0 \
-multiscale \
```

(continues on next page)

(continued from previous page)

```
-niter 100000 -mgain 0.8 -threshold 0.15 \
-channelsout 14 -joinchannels -fit-spectral-pol 2 \
-deconvolution-channels 4 \
-name mfclean *.dppp.MS.ph
```

Analyse the individual output images and the MFS images.

16.4 3C295 – A bright source at the centre of the field

In this exercise, the user will calibrate LBA and HBA datasets for 3C295. By the end of the exercise the user should be able to:

- inspect raw LOFAR data,
- automatically and manually flag data with DPPP, including demix the LBA data,
- calibrate the data with DPPP,
- produce maps with WSClean,
- create a sky model from the data, and,
- subtract bright sources using DPPP

The data ara available on the LOFAR LTA, as described at the beginning of this chapter. You will see the HBA data set listed at the top of the page, and if you click on the "Unspecified process" at the bottom, the LBA data set will show up.

Log into one of the compute nodes above, and make a new directory, for example,

```
ssh -Y lof019
cd /data/scratch/<username>/
mkdir tutorial/
mkdir tutorial/3c295/
cd tutorial/3c295
```

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

module load lofar

16.4.1 HBA

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.

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:  /cep3home/williams/tutorial/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 elapsed time = 3599 seconds
   Observed from   12-Nov-2012/12:47:00.0   to   12-Nov-2012/13:46:59.0 (UTC)


Fields: 1
  ID   Code Name                RA               Decl           Epoch        nRows
  0         BEAM_0              14:11:20.500000 +52.12.10.00000 J2000      5337090

Spectral Windows:  (1 unique spectral windows and 1 unique polarization setups)
  SpwID  Name   #Chans   Frame   Ch0(MHz)  ChanWid(kHz)  TotBW(kHz) CtrFreq(MHz)
 Corrs
  0      SB-0      64   TOPO     118.849         3.052       195.3    118.9453   XX
 XY  YX  YY

Antennas: 54:
  ID   Name  Station   Diam.    Long.         Lat.                Offset from array
 center (m)               ITRF Geocentric coordinates (m)
                                                                        East
 North     Elevation               x                 y                 z
  0    CS001HBA0LOFAR     31.3 m   +006.52.07.1  +52.43.34.7         -0.0006       -0.
 1610  6364572.0471  3826896.235000   460979.455000  5064658.203000
  1    CS001HBA1LOFAR     31.3 m   +006.52.02.2  +52.43.31.8         -0.0013       -0.
 1617  6364572.3376  3826979.384000   460897.597000  5064603.189000
  2    CS002HBA0LOFAR     31.3 m   +006.52.07.6  +52.43.46.8         -0.0005       -0.
 1582  6364570.0290  3826600.961000   460953.402000  5064881.136000
  ...
  ...
  43   CS501HBA1LOFAR     31.3 m   +006.51.59.7  +52.44.25.8         -0.0017       -0.
 1489  6364565.9714  3825663.508000   460692.658000  5065607.883000
  44   RS106HBALOFAR      31.3 m   +006.59.05.6  +52.41.21.6          0.0592       -0.
 1926  6364586.7503  3829205.598000   469142.533000  5062181.002000
  ...
  ...
  51   RS503HBALOFAR      31.3 m   +006.51.04.8  +52.45.33.2         -0.0095       -0.
 1330  6364557.2108  3824138.566000   459476.972000  5066858.578000
  52   RS508HBALOFAR      31.3 m   +006.57.13.3  +53.03.21.7          0.0431        0.
 1202  6364441.8110  3797136.484000   463114.447000  5086651.286000
  53   RS509HBALOFAR      31.3 m   +006.47.04.7  +53.13.30.1         -0.0438        0.
 2644  6364384.5199  3783537.525000   450130.064000  5097866.146000

The MS is fully regular, thus suitable for BBS
   nrows=5337090   ntimes=3594   nbands=1   nbaselines=1485 (54 autocorr)
```

From this, you should see that HBA_DUAL_INNER mode was used, i.e. the core stations are split in two HBA subfields, giving a total of 54 stations, and the size of the remote stations is the same as the core stations. The observation was $\sim$ 1 hour (3599 seconds) with 3594 timestamps so the time resolution is $\sim$ 1 s. There are 64 spectral channels

and the frequency is 118.849 MHz for SB000 and 119.044 MHz for SB001, and the total bandwidth for each subband is ~ 0.2 MHz.

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 DPPP. Typically, initial RFI flagging and averaging will be done by the averaging pipeline run by the Radio Observatory. Note that the limitation on the compression in frequency is set by the size of the field you wish to image and the amount of bandwidth smearing at the edges of the field. The time averaging is limited not only by the amount of time smearing you will allow but also by the changes in the ionosphere.

In this example we are flagging the data using the aoflagger algorithm within DPPP. Here we will compress the subband to 4 channels in frequency and 5 s in time. The parset file for the flagging and compression should be copied to your working directory,

```
cat NDPPP_HBA_preprocess.parset

msin = L74759_SAP000_SB000_uv.MS
msin.autoweight=TRUE
msin.datacolumn=DATA

msout = L74759_SAP000_SB000_uv.MS.avg.dppp
msout.datacolumn=DATA

steps=[preflagger0,preflagger1,aoflagger,averager]

preflagger0.chan=[0,1,62,63]
preflagger0.type=preflagger

preflagger1.corrtype=auto
preflagger1.type=preflagger

aoflagger.autocorr=F
aoflagger.timewindow=0
aoflagger.type=aoflagger

averager.freqstep=16     # compresses from 64 to 4 channels
averager.timestep=4     # compresses 4 time-slots into 1, i.e. 4s
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 DPPP use,

```
DPPP NDPPP_HBA_preprocess.parset > log.ndppp.flagavg 2>&1 &
```

In bash, the **2>&1** pipes both the stdout and stderr to the log file and the **&** runs the task in the background so you can continue working on the terminal.

Depending on the use of the cluster, it will take about $sim5$ minutes to flag and average the data. The progress bar reports the stage of the initial DPPP steps, not the entire DPPP run, so it will keep running several minutes after the progress bar reaches 100%.

You can inspect the output log file by using,

```
cat log.ndppp.flagavg
```

The log file lists the input and output parameters, the level of flagging at each step and the total amount of data flagged. You will see that the total data flagged for each of the flagging steps is 4.7%, 3.4% and 2.6% respectively.

Edit the msin and msout fields of the parset to do the same for the second sub-band. Or you can run it again using the same parset and supplying new msin and msout parameters on the command line, i.e.

```
DPPP NDPPP_HBA_preprocess.parset msin=L74759_SAP000_SB001_uv.MS \
msout=L74759_SAP000_SB001_uv.MS.avg.dppp  > log.ndppp.flagavg1 2>&1 &
```

The flagged and compressed data set should now be in your working directory and each MS should have a total size of 333 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
```

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 for full details.

```
module load casa
casaplotms
```

Fig. 16.7 and Fig. 16.8 show the Amp. vs. time and Amp. vs. UV distance (wavelengths) for SB000. Through inspecting the data in casaplotms you should be able to see that CS302HBA0 has consistently low amplitudes and is the only station contributing to the few high amplitudes. If you look back at the logs from the initial NDPPP preprocessing you will notice that about 40-50% of the data for this station was flagged by aoflagger.

We will remove CS302HBA0 now with DPPP. This can be done either by flagging it with a preflagger step or, as we will do here, filtering it out. In this way it is completely removed from the MS so we have to specify a new output MS. The DPPP parsets are:

```
cat NDPPP.split.sb000.parset

msin = L74759_SAP000_SB000_uv.MS.avg.dppp
msin.baseline = !CS302HBA0
msin.datacolumn = DATA
msout = L74759_SAP000_SB000_uv.MS.avg.dppp.flag

steps=[]

DPPP NDPPP.split.sb000.parset > ndppp.flag0.log 2>&1 &
```

and likewise for the other subband:

```
> DPPP NDPPP.split.sb000.parset msin=L74759_SAP000_SB001_uv.MS.avg.dppp \
msout=L74759_SAP000_SB001_uv.MS.avg.dppp.flag > ndppp.flag1.log 2>&1 &
```

The visibilities after removing CS302HBA0 are shown in Fig. 16.9.

Calibration with DPPP

In this example, we will use DPPP to perform the calibration (see also *Calibration* with DPPP) on single sub-bands.

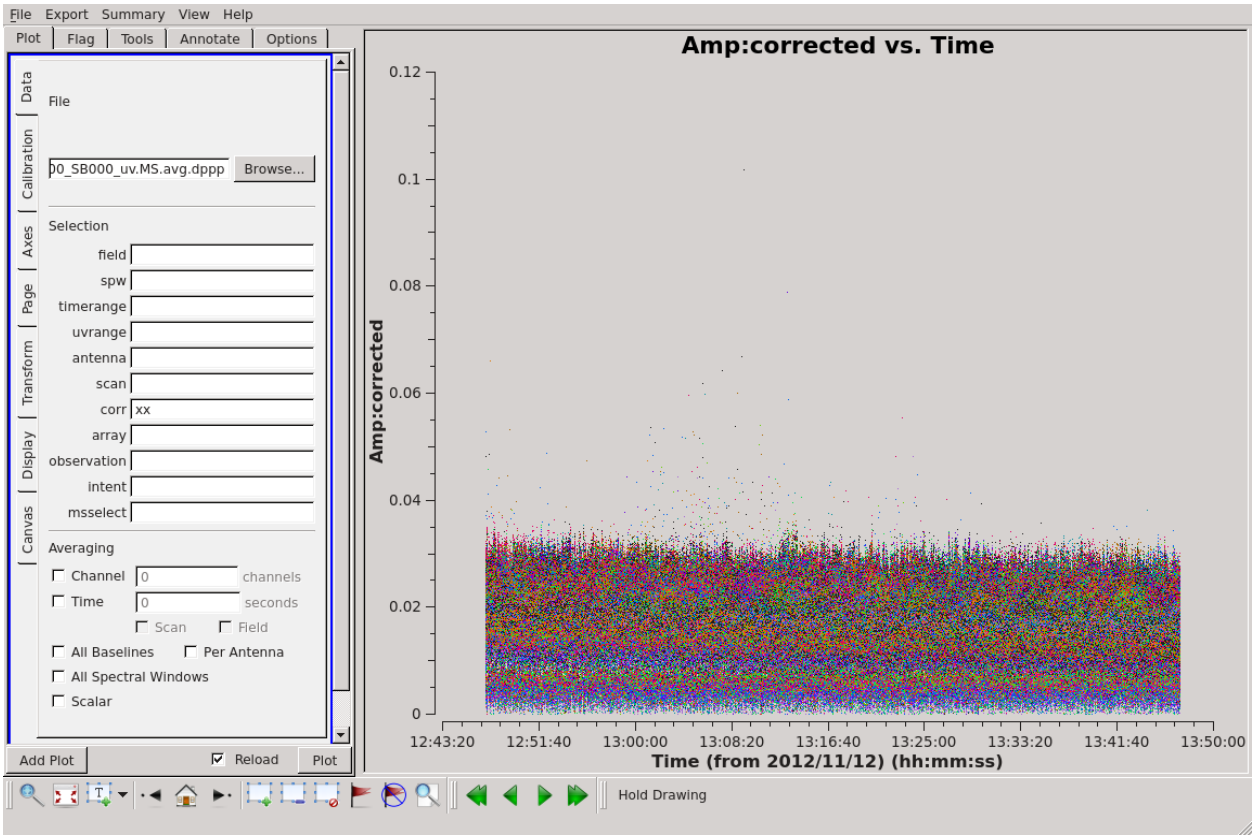


Fig. 16.7: Plotting the XX visibility amplitude against time.

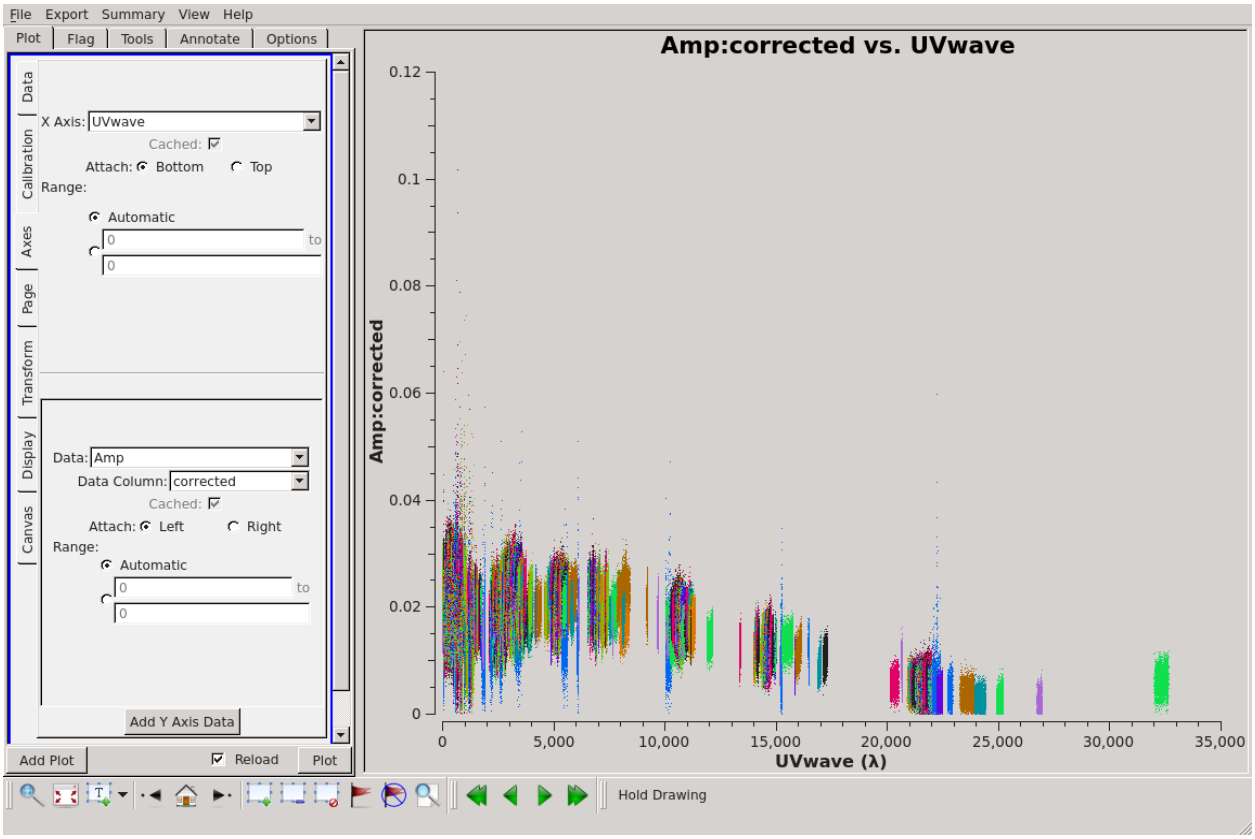


Fig. 16.8: Plotting the visibility amplitude against UV distance in wavelengths.

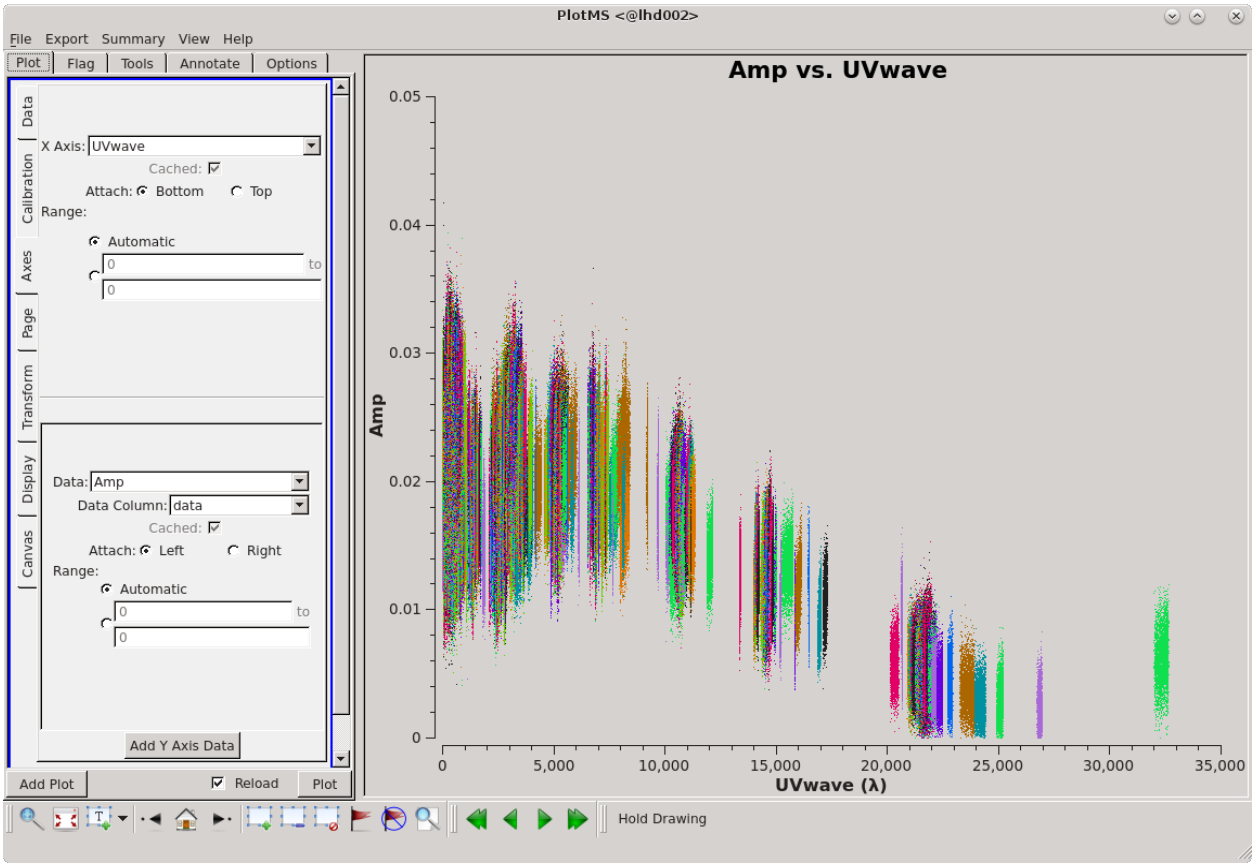


Fig. 16.9: SB000. Plotting the XX visibility amplitude against UV distance in wavelengths after flagging CS302HBA0. (The colour scheme 'Antenna1' is used here.)

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.

```
cat 3C295TWO.skymodel

# (Name, Type, Patch, Ra, Dec, I, ReferenceFrequency='150.e6', SpectralIndex) = format

, , 3c295, 14:11:20.64, +52.12.09.30
3c295A, POINT, 3c295, 14:11:20.49, +52.12.10.70, 48.8815, , [-0.582, -0.298, 0.583, -
↪0.363]
3c295B, POINT, 3c295, 14:11:20.79, +52.12.07.90, 48.8815, , [-0.582, -0.298, 0.583, -
↪0.363]
```

Here you can see that the two point-source components on 3C295 have been grouped in a single 'patch'. Note that there are other sources visible within the field of view, but 3C295 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 *GSM* for more information about the GSM and gsm.py.

Running gsm.py without any arguments will show you the correct usage (help).

```
gsm.py

Insufficient arguments given; run as:

  /opt/cep/LofIm/daily/Tue/lofar_build/install/gnu_opt/bin/gsm.py outfile RA DEC␣
↪radius [vlssFluxCutoff [assocTheta]] to select using a cone

  outfile         path-name of the output file
                  It will be overwritten if already existing
  RA              cone center Right Ascension (J2000, degrees)
  DEC             cone center Declination     (J2000, degrees)
  radius          cone radius                (degrees)
  vlssFluxCutoff  minimum flux (Jy) of VLSS sources to use
                  default = 4
  assocTheta      uncertainty in matching     (degrees)
                  default = 0.00278  (10 arcsec)
```

So now we can construct the command to make a model for the 3C295 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 3C295. To be used by DPPP this needs to be converted to sourcedb format:

```
makesourcedb in=3C295TWO.skymodel out=3C295TWO.sourcedb format='<'
```

The parset file for DPPP can be found at,

cat dppp-calibrate.parset msin=L74759_SAP000_SB000_uv.MS.avg.dppp.flag msout=. msout.datacolumn=CORRECTED_DATA steps=[gaincal, applybeam] gaincal.sourcedb=3C295TWO.sourcedb gaincal.caltype=diagonal gaincal.usebeammodel=true gaincal.applysolution=true

This is a very simple parset file that solves and corrects the data. To perform the calibration, use the following command:

```
DPPP dppp-calibrate.parset > log.ndppp.cal 2>&1 &
```

Note that using the redirect '> log.ndppp.cal' command allows you to save and inspect the output of DPPP and using the '&' at the end runs DPPP in the background, allowing you to continue with other tasks, e.g. you can simultaneously run a similar command for the second sub-band, as before updating the msin parameter:

```
DPPP dppp-calibrate.parset msin=L74759_SAP000_SB001_uv.MS.avg.dppp.flag > log.ndppp.
↪cal1 2>&1 &
```

The calibration process should be completed in a few minutes.

Once complete it is useful to look at the calibrated data with parmdbplot.py:

```
parmdbplot.py  L74759_SAP000_SB000_uv.MS.avg.dppp.flag/instrument/
```

It is useful to de-select the 'use resolution' option as this will plot all of the solutions that we solved for. After de-selecting the *use resolution' option select a few stations and look at the solutions. :numref:'tutparmdbplot* to Fig. 16.13 shows some solution plots for SB000.

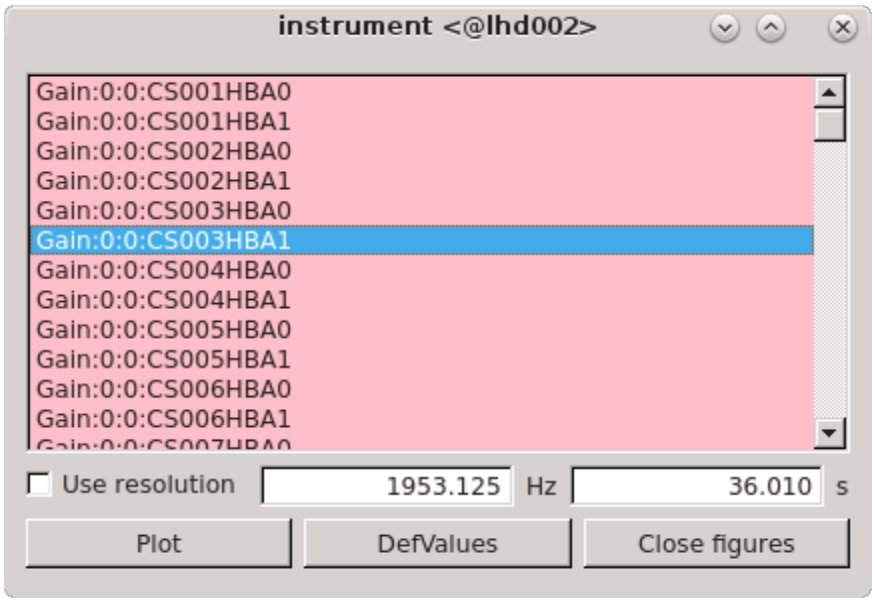


Fig. 16.10: The parmdb window. Use this to select the stations for which you want to inspect the solutions, and to change the resolution that is used to display the solutions.

ASIDE: While parmdbplot.py is very useful and diverse, sometimes you want a quick look at all the solutions. In python you can use **lofar.parmdb** to read and plot the solutions. This example script plots all the phase and amplitude solutions in a single image (see Fig.~ref{fig:solplotall}):

```
python plot_solutions_all_stations_v2.py -p -a L74759_SAP000_SB000_uv.MS.avg.dppp/
↪instrument/ hba_sb000_gains
display hba_sb000_gains_amp.png
display hba_sb000_gains_phase.png
```

Or use George Heald's solplot.py or *LoSoTo*.

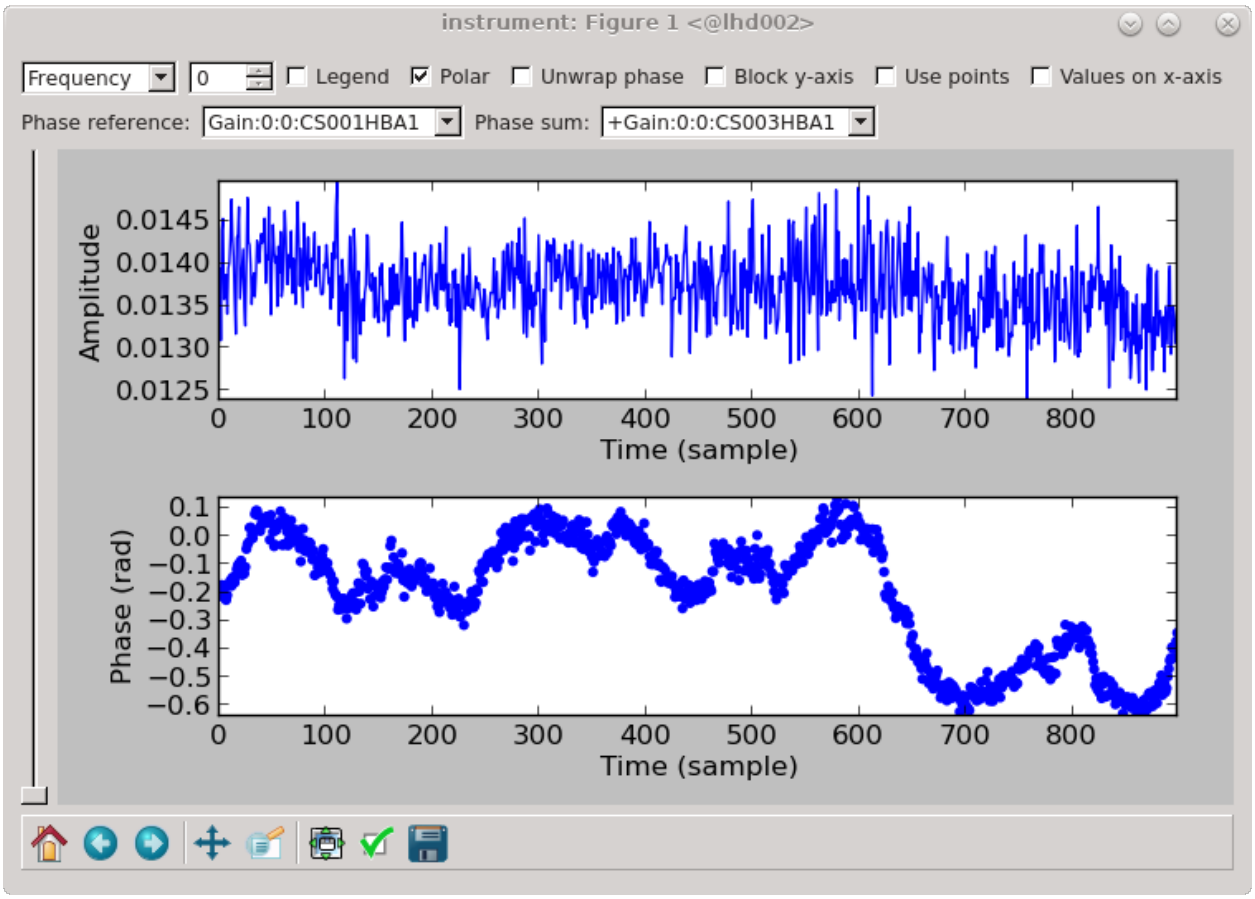


Fig. 16.11: The solutions for CS003HBA1.

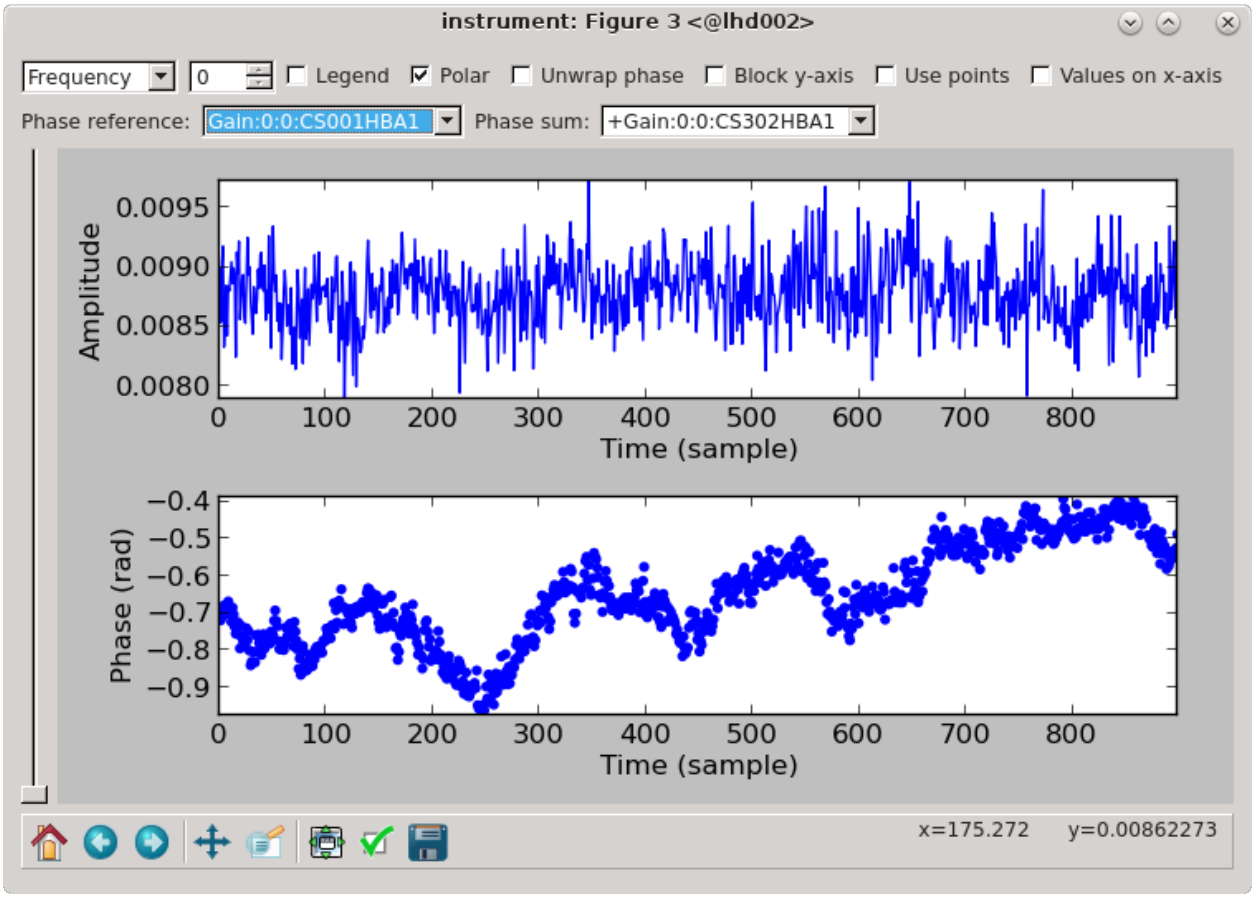


Fig. 16.12: The solutions for CS302HBA1.

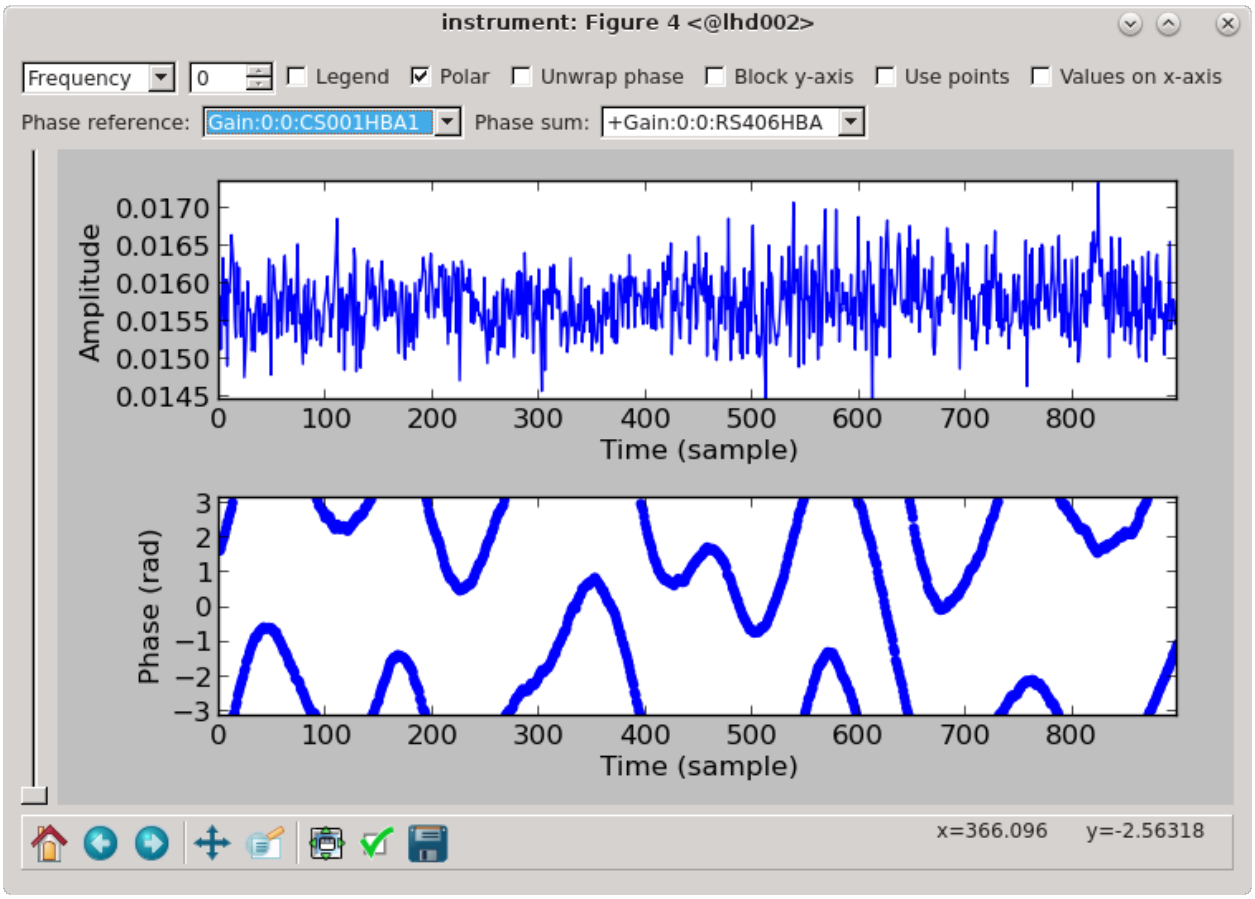


Fig. 16.13: The solutions for RS406HBA.

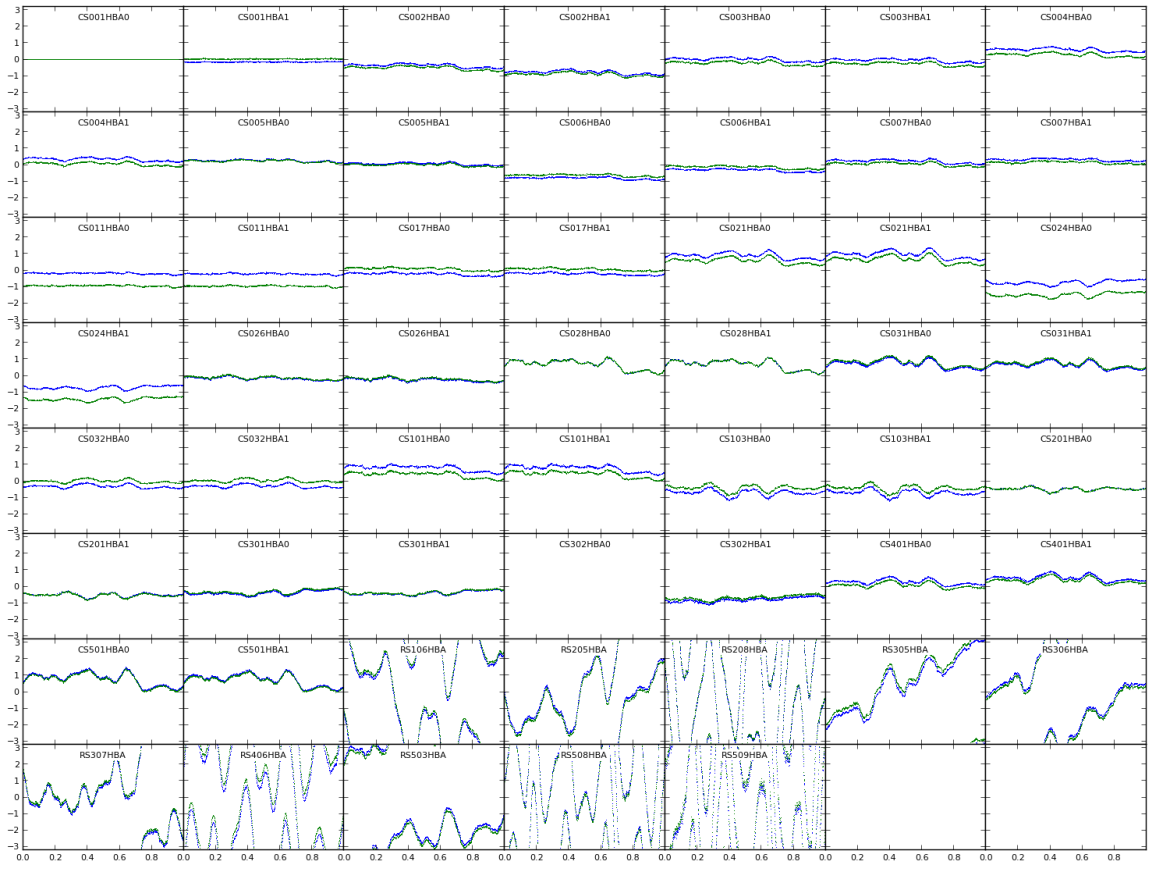


Fig. 16.14: Phase solutions for all stations for SB000 (polarizations in different colours).

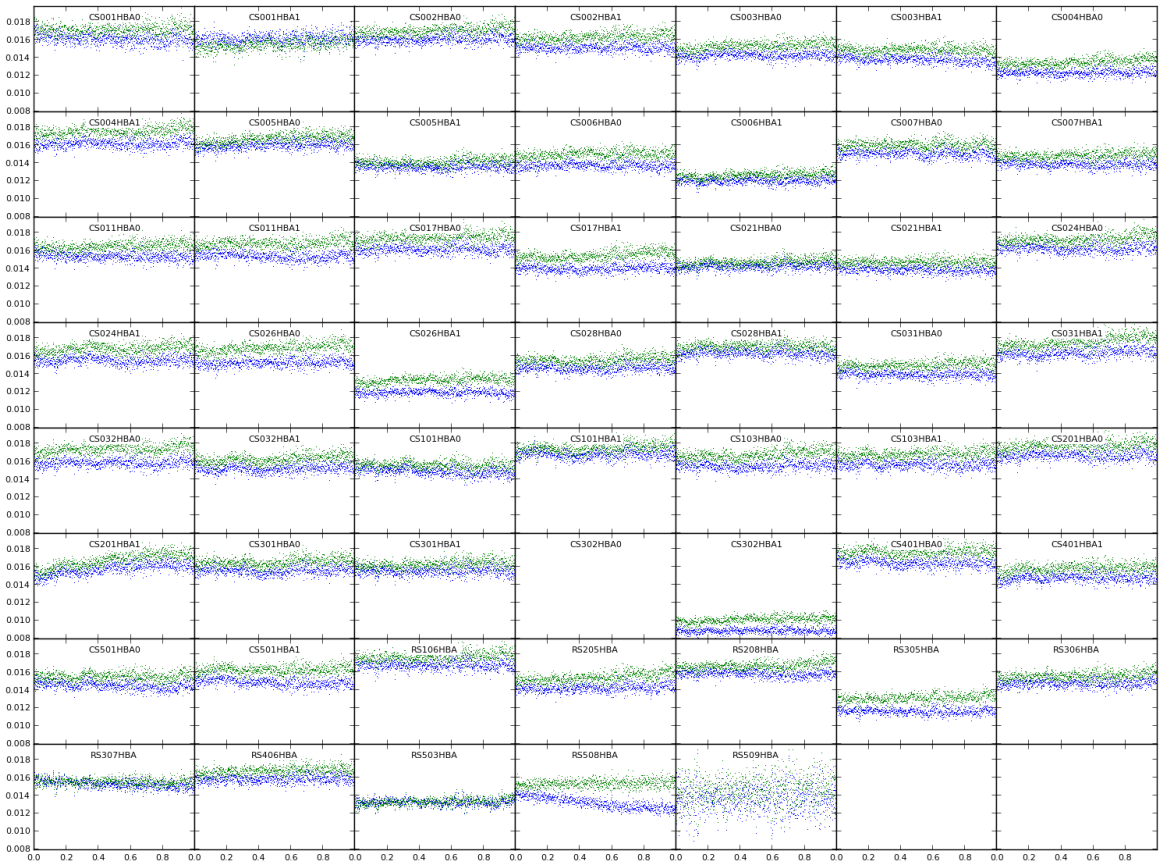


Fig. 16.15: Amplitude solutions for all stations for SB000.

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. Fig. 16.16 to Fig. 16.18 show these plots for SB000 and SB001 respectively. For SB000, it is clear that the solutions for CS302HBA1 are also very noisy. For both sub-bands, baselines RS508HBA&RS509HBA (visible in orange) and RS208HBA&RS509HBA (in green) look bad.

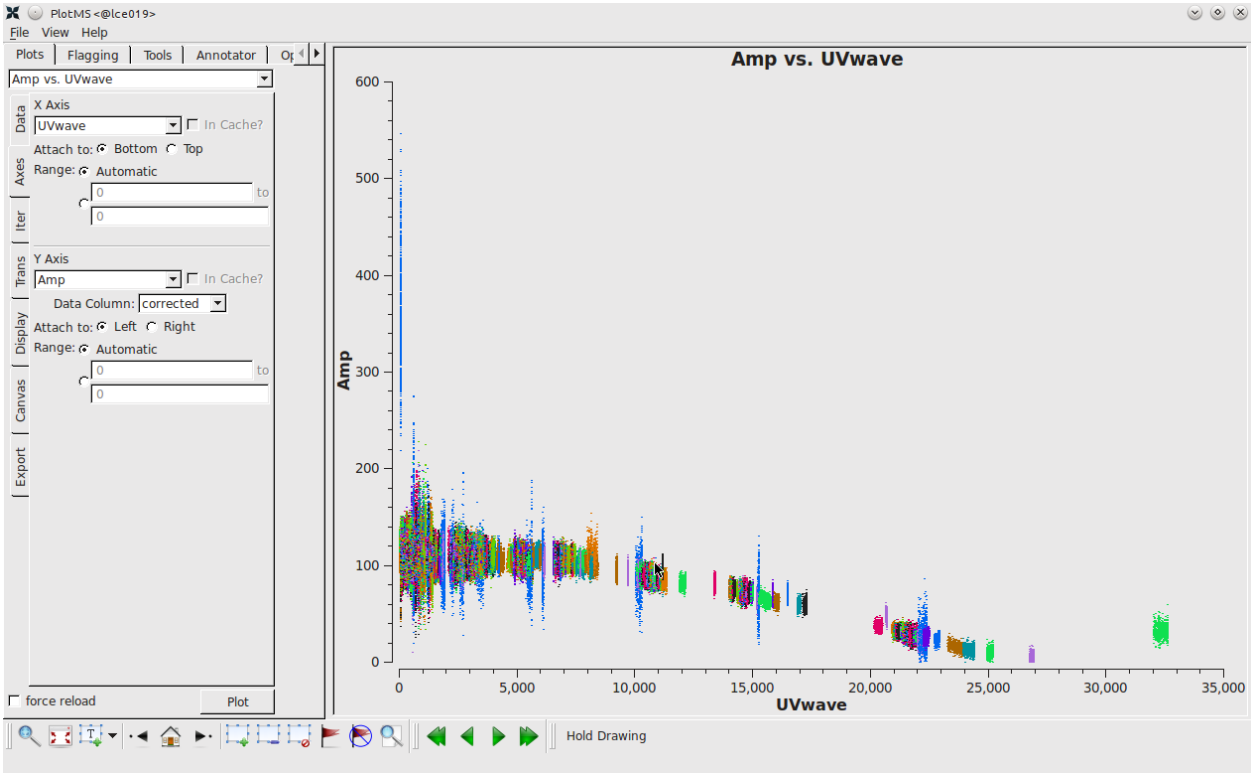


Fig. 16.16: Plotting the XX visibility amplitude against UV distance.

This time we will just flag the station and baselines:

```
> cat NDPPP.flag.sb000.parset

msin = L74759_SAP000_SB001_uv.MS.avg.dppp.flag
msin.datacolumn = DATA
msout =
steps = [flag]
flag.type=preflagger
flag.baseline= CS302HBA1* ; RS508HBA*&RS509HBA* ; RS208HBA*&RS509HBA*

> DPPP NDPPP.flag.sb000.parset
> DPPP NDPPP.flag.sb000.parset msin=L74759_SAP000_SB001_uv.MS.avg.dppp.flag
```

Next we will re-do the calibration (it's probably a good idea after removing some bad data):

```
> DPPP dppp-calibrate.parset > log.ndppp.cal 2>&1 &
> DPPP dppp-calibrate.parset msin=L74759_SAP000_SB001_uv.MS.avg.dppp.flag > log.ndppp.
↪cal1 2>&1 &
```

The amplitude against time for the flagged corrected data is plotted in Fig. 16.19.

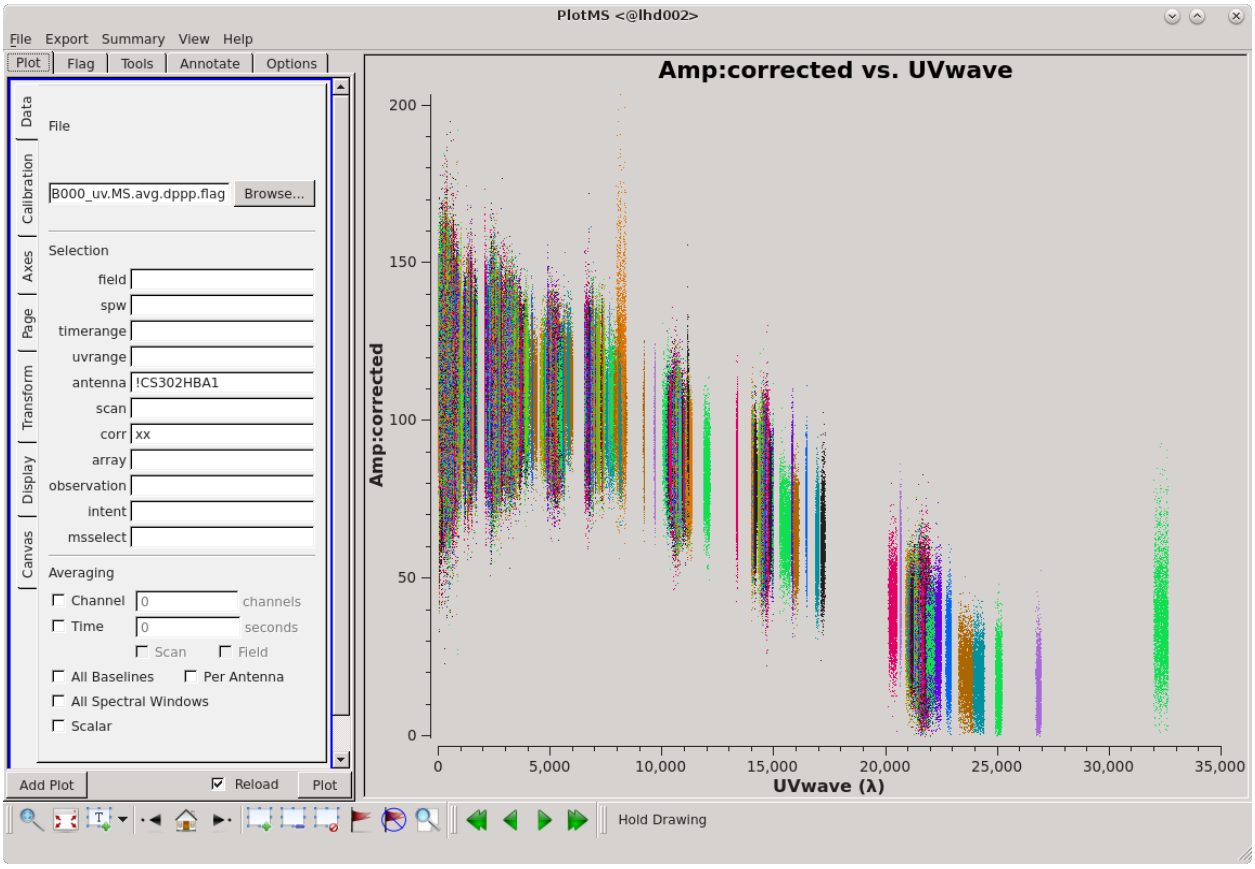


Fig. 16.17: Same as Fig. 16.16 but excluding antenna CS302HBA1.

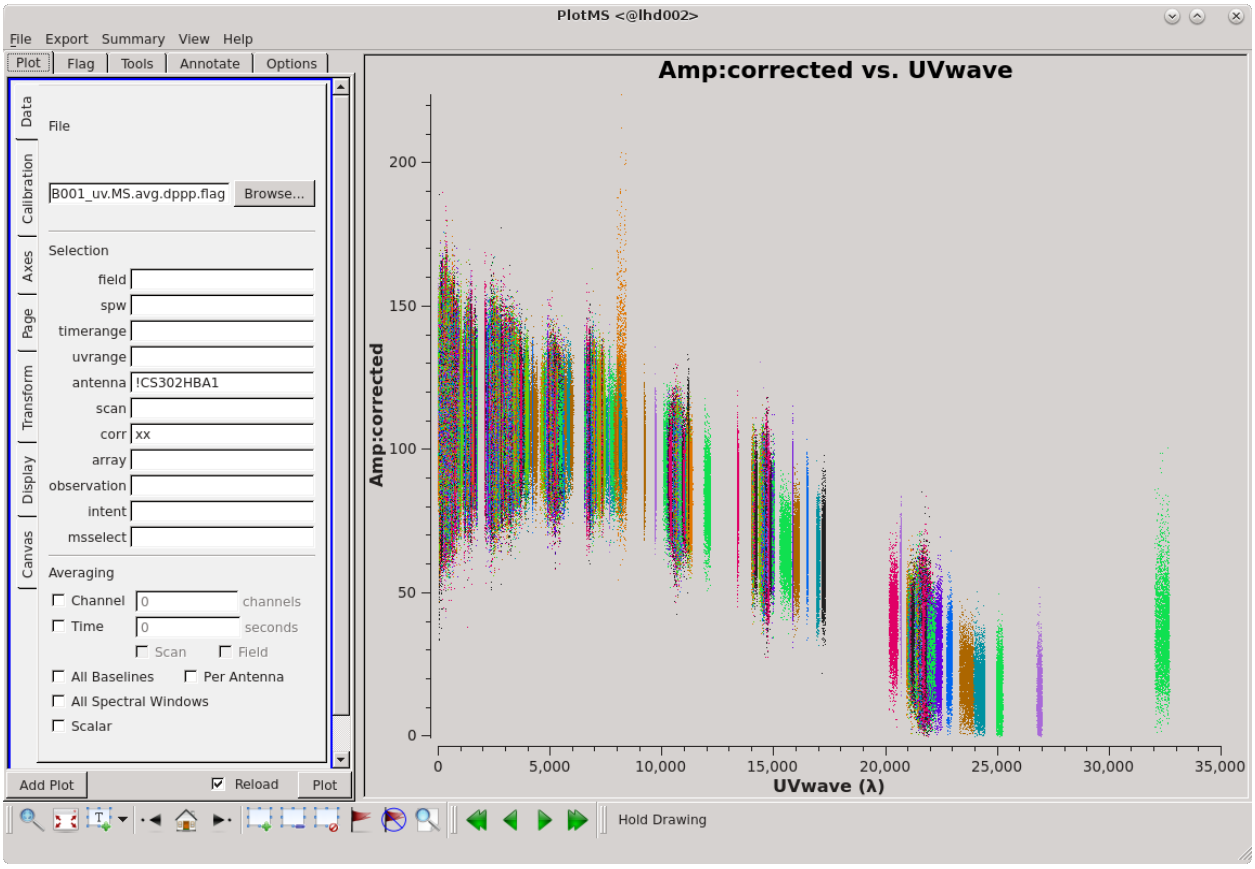


Fig. 16.18: Plotting the XX visibility amplitude against UV distance.

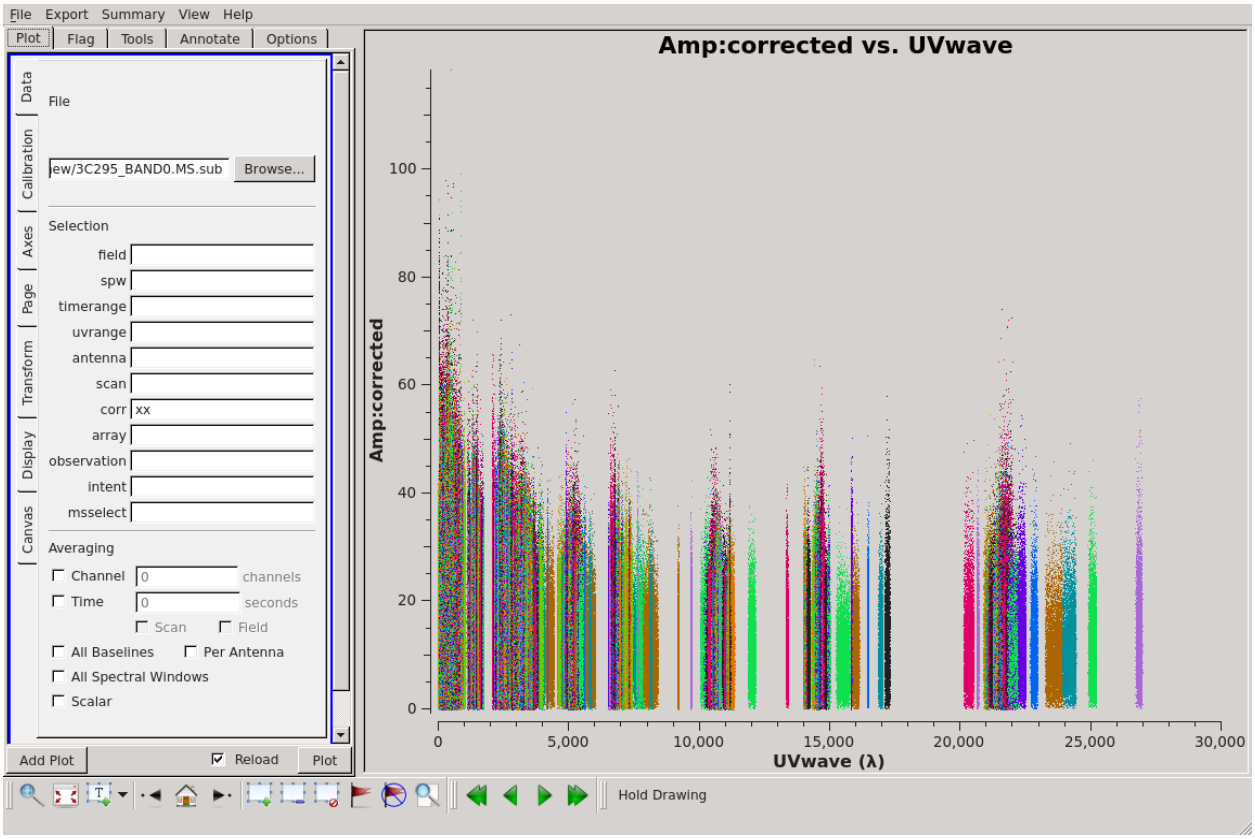


Fig. 16.19: Plotting the XX visibility amplitude against UV distance for SB000 after flagging.

Imaging

Here we will use WSClean to do the deconvolution combining both subbands. While 3C295 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 WSClean. The list of parameters along with a brief description of each can be shown with:

```
wsclean
```

At 120 MHz the LOFAR (NL Remote) field of view is 4.5 deg (FWHM) and the theoretical resolution should be around $8''$. However, the maximum resolution can only be achieved with a longer observation providing better uv coverage combined with direction-dependent calibration so we will limit ourselves here to a resolution of $25 - -30''$. This is realised with a maximum uv cut of 7000λ and a slightly uniform weighting (robust -0.5). $5''$ pixels will sample the beam well. WSClean will take a list of MS's as input, so it is not necessary to concatenate them before imaging. The full wsclean command is:

```
wsclean -name 3C295 -niter 40000 -threshold 0.01 -mgain 0.85 -pol I \
-weighting-rank-filter 3  -minuv-l 80 -maxuv-l 7000  -cleanborder 0  \
-scale 5asec -weight briggs -0.5 -size 9600 9600 -trim 8000 8000  \
L74759_SAP000_SB000_uv.MS.avg.dppp.flag L74759_SAP000_SB001_uv.MS.avg.dppp.flag
```

The default major cycle clean gain (**mgain**) is 1, so it is best to change this to a lower value. Also, wsclean does no internal padding like casa so to pad by a factor of 1.2 we set the size and trim parameters appropriately to avoid aliasing effects. Both subbands are given. This will take $10 - 15$ minutes. This is a very large image but you will see that it includes sources all over.

Cleaning should be done within masks, so we will use PyBDSF (see *Source detection: PyBDSF*) with some default settings to create an island mask.

```
> pybdsm
BDSM [1]: process_image(filename='3C295-image.fits', rms_box=(60,20))
BDSM [2]: export_image(img_type='island_mask', mask_dilation=0)
```

The resulting image **3C295-image.pybdsm_island_mask.fits** can now be given to wsclean as a fitsmask. We will also turn on the differential beam calculation (the data have already been corrected for the beam at the phase centre by the DPPP applybeam step).

```
> wsclean -name 3C295_mask -niter 40000 -threshold 0.01 -mgain 0.85 \
-pol I -weighting-rank-filter 3  -minuv-l 80 -maxuv-l 7000  \
-scale 5asec -weight briggs -0.5 -size 9600 9600 -trim 8000 8000  -cleanborder 0  \
-apply-primary-beam -use-differential-lofar-beam \
-fitsmask 3C295-image.pybdsm_island_mask.fits \
L74759_SAP000_SB000_uv.MS.avg.dppp.flag L74759_SAP000_SB001_uv.MS.avg.dppp.flag
```

WSClean produces a lot of images as output, including

```
root-image.fits        # uncorrected model image
root-image-pb.fits     # corrected model image
root-model.fits        # clean component image
root-psf.fits          # point spread function
```

The output image can be inspected with ds9 (or casaviewer if you prefer)

```
> ds9 -scale limits -0.1 1 root-image.fits
```

Fig. 16.20 and Fig. 16.21 show the cleaned corrected image (**-image-pb.fits**). One can see 3C295 at the centre of the field and even though only 3C295 was in our calibration model there are clearly many other sources visible in the field. The noise in the image is about 20 mJy/beam and 3C295 has a flux density of about 110 Jy. This noise is about what

is expected from the LOFAR noise calculator, 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. The uv cut we are using would also result in a higher noise as not all the data is used in the imaging.

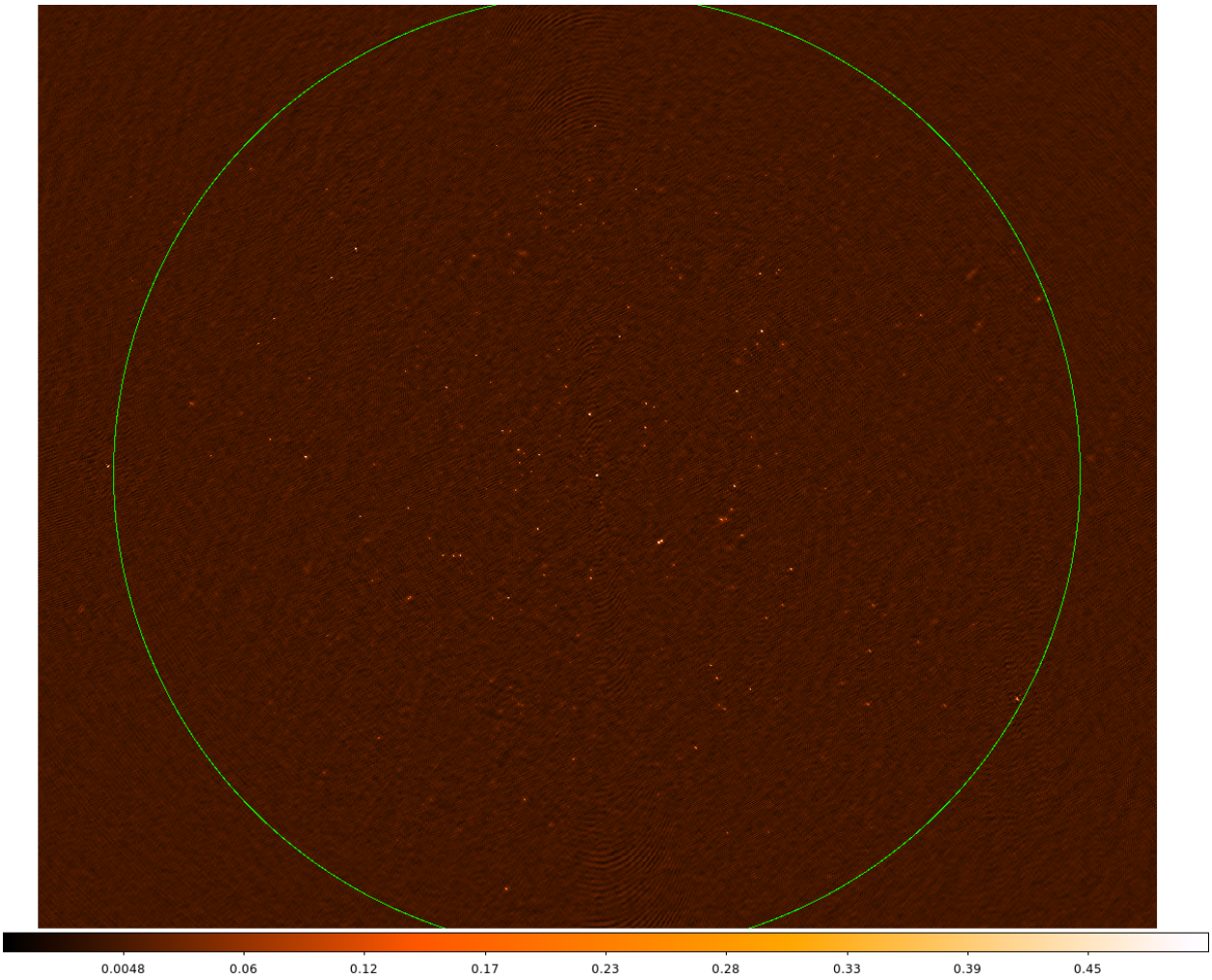


Fig. 16.20: The cleaned image for 3C295. The radius of the circle is 4.8 deg.

Combining Measurement Sets

While we did not do this above, it can often be useful to combine calibrated MS's for separate sub-bands into a single MS, to allow allow for combined processing in subsequent DPPP runs.

```
> cat NDPPP.combineMS.parset
msin = L74759_SAP000_SB*_uv.MS.avg.dppp
msin.datacolumn = DATA
msout = 3C295_BAND0.MS

steps = []

> DPPP dppp-calibrate.parset msin=3C295_BAND0.MS gaincal.nchan=2 > log.ndppp.cal.B0 2>
↪&1 &
```

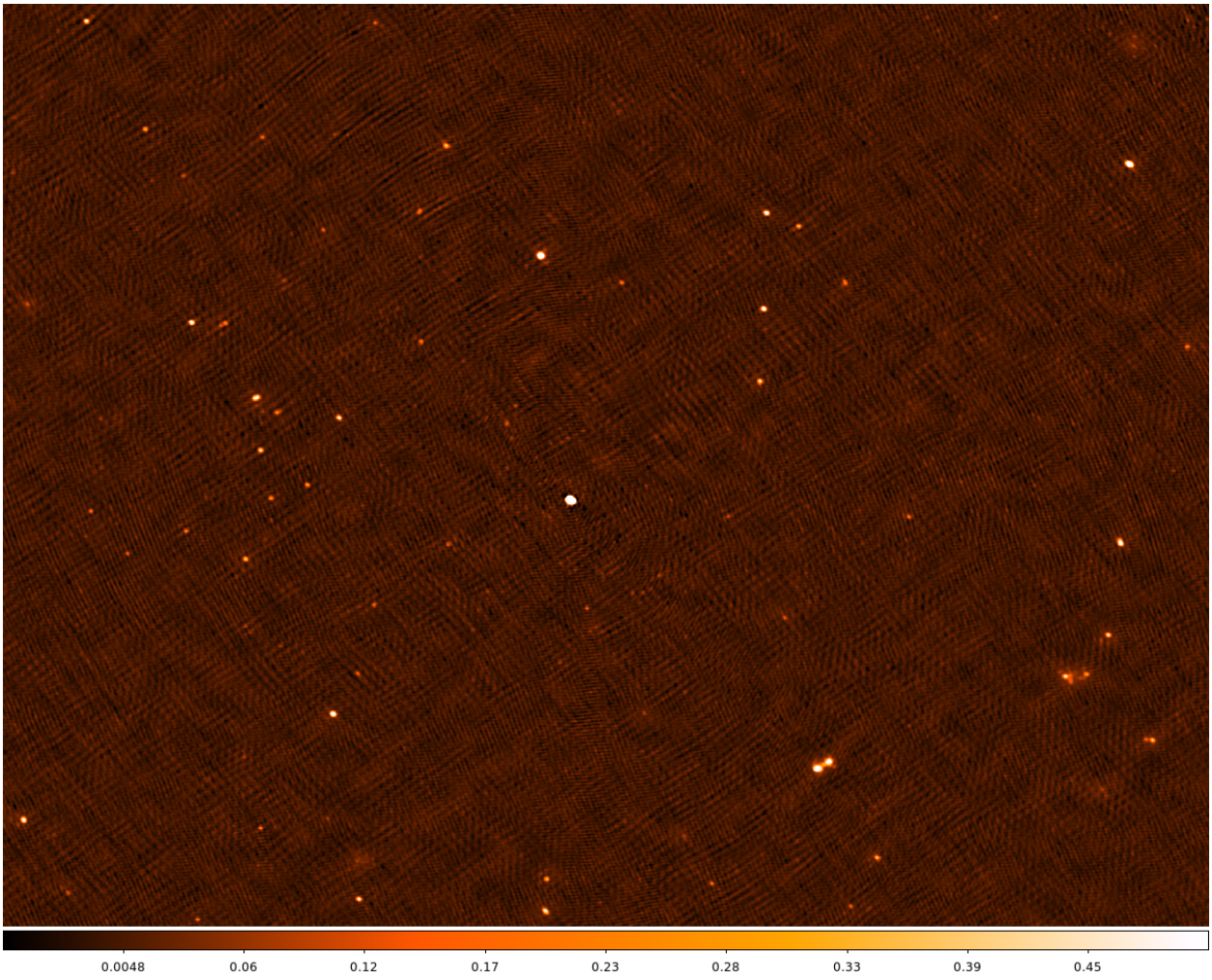


Fig. 16.21: A zoom-in of the central part of the field. The data range is set to [-0.05, 0.5].

The wild card in line one of this very simple parset means that all sub-bands of the observation will be combined. Line two means that the DATA column from the input MSs will be written to the DATA column in the output. Here we only have two subbands to combine but the method works for many. If you do msoverview now you will see that there are 8 spectral channels. At this point you need to redo the calibration because we have copied the DATA column.

Here we set **gaincal.nchan=2** to compute a solution every 2 channels. At around 100 Jy at 150 MHz, 3C295 is bright enough that there is enough signal to do this.

Subtraction of 3C295

3C295 is the dominant source at the center of the field. Sometimes it is necessary to subtract sources and here we will subtract 3C295. In this specific case this is easy because we already have a very good model for this calibrator source so we will use that instead of making a model from the image.

We require a parset that includes a subtract step for the source 3C295. We have already done a solve step to obtain our gain solutions so we will simply subtract the source using those solutions

```
> cat NDPPP_subtract3c295.parset
msin=3C295_BAND0.MS
msout=3C295_BAND0.MS.sub
steps=[subtract3c295, applycal, applybeam]
subtract3c295.type=predict
subtract3c295.operation=subtract
subtract3c295.sourcedb=3C295TWO.sourcedb
subtract3c295.sources=[3c295]
subtract3c295.usebeammodel=true
subtract3c295.applycal.parmdb=3C295_BAND0.MS/instrument
applycal.parmdb=3C295_BAND0.MS/instrument

> DPPP NDPPP_subtract3c295.parset > log.dppp.gaincalsubtract 2>&1 &
```

The **subtract3c295.applycal.parmdb** specifies the solutions to use for the subtract – in this case the model visibilities are corrupted by the gain solutions before being subtracted. The applycal step is because we still need to apply the gain solutions to the rest of the field and the applybeam step so that the beam is taken out at the phase centre before imaging. The output is to the DATA column of the new MS.

The 3C295-subtracted visibility amplitudes are plotted against time in Fig. 16.22.

We can make an image of the subtracted data (there is no corrected data column so wsclean will use the data column)

```
> wsclean -name 3C295_sub -niter 40000 -threshold 0.01 -mgain 0.85 -pol I \
-weighting-rank-filter 3  -minuv-l 80 -maxuv-l 7000   -cleanborder 0  \
-scale 5asec -weight briggs -0.5 \
-size 4800 4800 -trim 4000 4000  3C295_BAND0.MS.sub
```

The resulting image is shown in Fig. 16.23. There is a small residual (0.5 Jy) where 3C295 was.

16.4.2 LBA

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
```

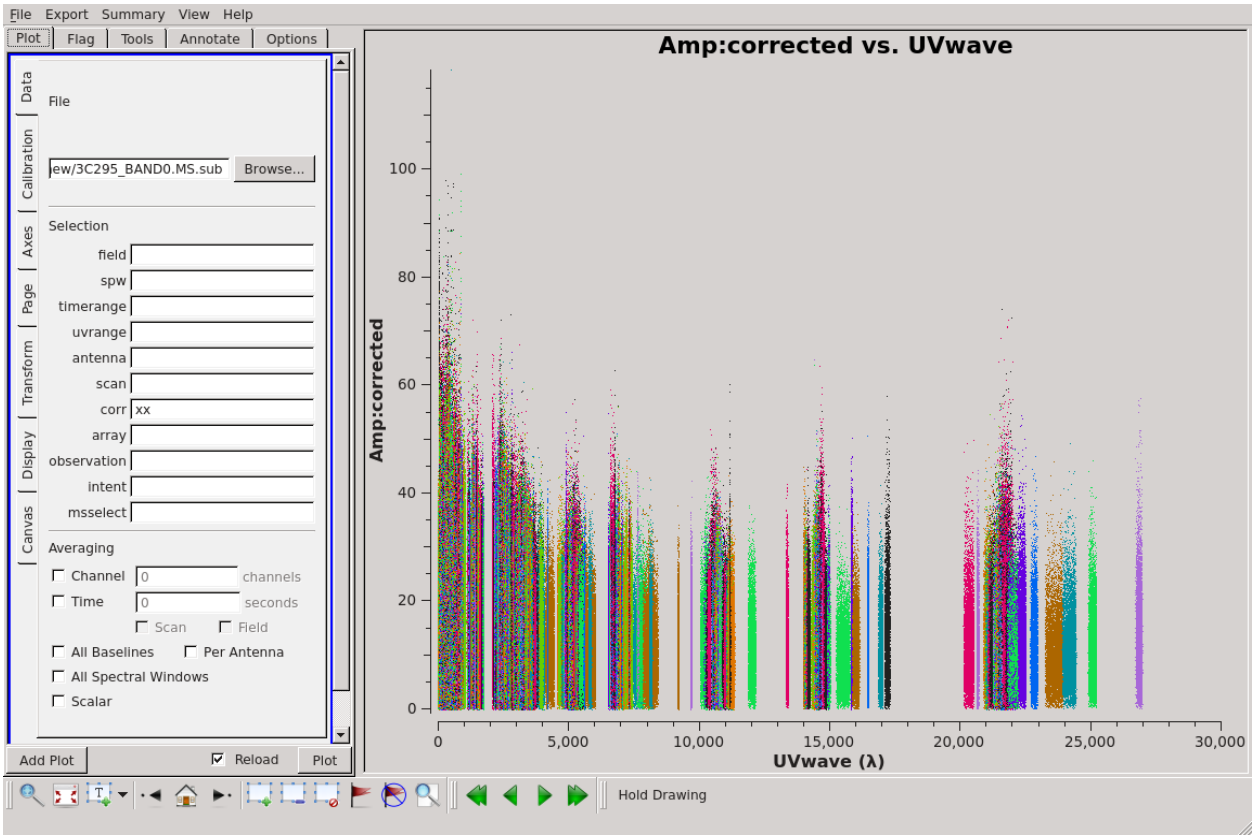


Fig. 16.22: Plotting the XX visibility amplitude against UV distance after flagging.

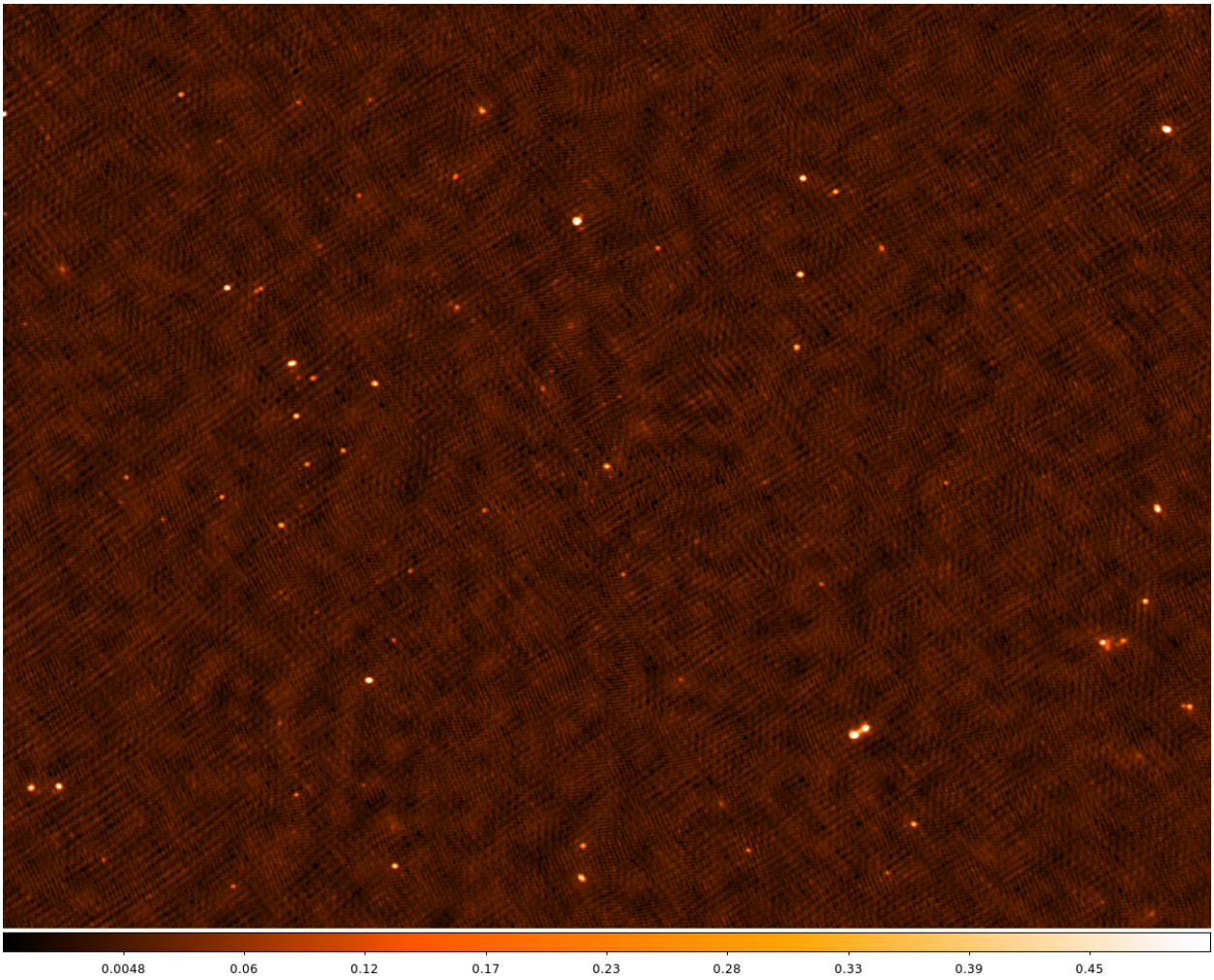


Fig. 16.23: The cleaned image for 3C295 after 3C295 has been subtracted. The data range is set to [-0.05, 0.5].

for the two sub-bands.

for the two sub-bands.

16.5 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:  /cep3home/williams/lba_tutorial/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 elapsed time = 3599 seconds
   Observed from   12-Nov-2012/14:06:00.0   to   12-Nov-2012/15:05:59.0 (UTC)


Fields: 1
  ID   Code Name                RA               Decl           Epoch        nRows
  0          BEAM_0              14:11:20.516698 +52.12.09.92762 J2000      1897632

Spectral Windows:  (1 unique spectral windows and 1 unique polarization setups)
  SpwID  Name   #Chans   Frame   Ch0(MHz)  ChanWid(kHz)  TotBW(kHz) CtrFreq(MHz) ␣
↪Corrs
  0      SB-0      64   TOPO      59.474         3.052       195.3     59.5703   XX ␣
↪XY  YX  YY

Antennas: 32:
  ID   Name  Station   Diam.    Long.         Lat.                Offset from array␣
↪center (m)               ITRF Geocentric coordinates (m)
                                                                        East         ␣
↪North     Elevation               x                y               z
  0    CS001LBALOFAR     86.0 m   +006.52.03.5  +52.43.34.0         -0.0011       -0.
↪1612  6364572.1238  3826923.546000   460915.441000  5064643.489000
  1    CS002LBALOFAR     86.0 m   +006.52.11.4  +52.43.47.4          0.0000       -0.
↪1581  6364569.9708  3826577.066000   461022.948000  5064892.786000
  2    CS003LBALOFAR     86.0 m   +006.52.06.9  +52.43.50.3         -0.0006       -0.
↪1574  6364569.6900  3826516.748000   460930.066000  5064946.457000
  3    CS004LBALOFAR     86.0 m   +006.52.06.5  +52.43.44.7         -0.0007       -0.
↪1587  6364570.2302  3826654.197000   460939.576000  5064842.426000
  ...
  30   RS508LBALOFAR     86.0 m   +006.57.11.4  +53.03.19.2          0.0429        0.
↪1196  6364442.0450  3797202.116000   463087.509000  5086605.037000
  31   RS509LBALOFAR     86.0 m   +006.47.07.0  +53.13.28.2         -0.0435        0.
↪2640  6364384.4231  3783579.131000   450178.882000  5097830.835000

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 $\sim$ 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.

16.6 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 DPPP. We will compress the sub-band to 4 channel in frequency and 4s 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 has been implemented in DPPP. 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.py** script

```
> plot_Ateam_elevation.py L74762_SAP000_SB000_uv.MS
```

the output of which is shown in Fig. 16.24. 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 using the aoflagger algorithm 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.

```
> 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

steps=[preflagger0,preflagger1,aoflagger,demixer]

preflagger0.chan=[0,1,62,63]
preflagger0.type=preflagger

preflagger1.corrtype=auto
preflagger1.type=preflagger

aoflagger.strategy=LBAdefault
aoflagger.autocorr=F
aoflagger.count.save=F
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=16     # compresses to 4 channels
demixer.timestep=4     # compresses 4 time-slots, i.e. 4s
```

(continues on next page)

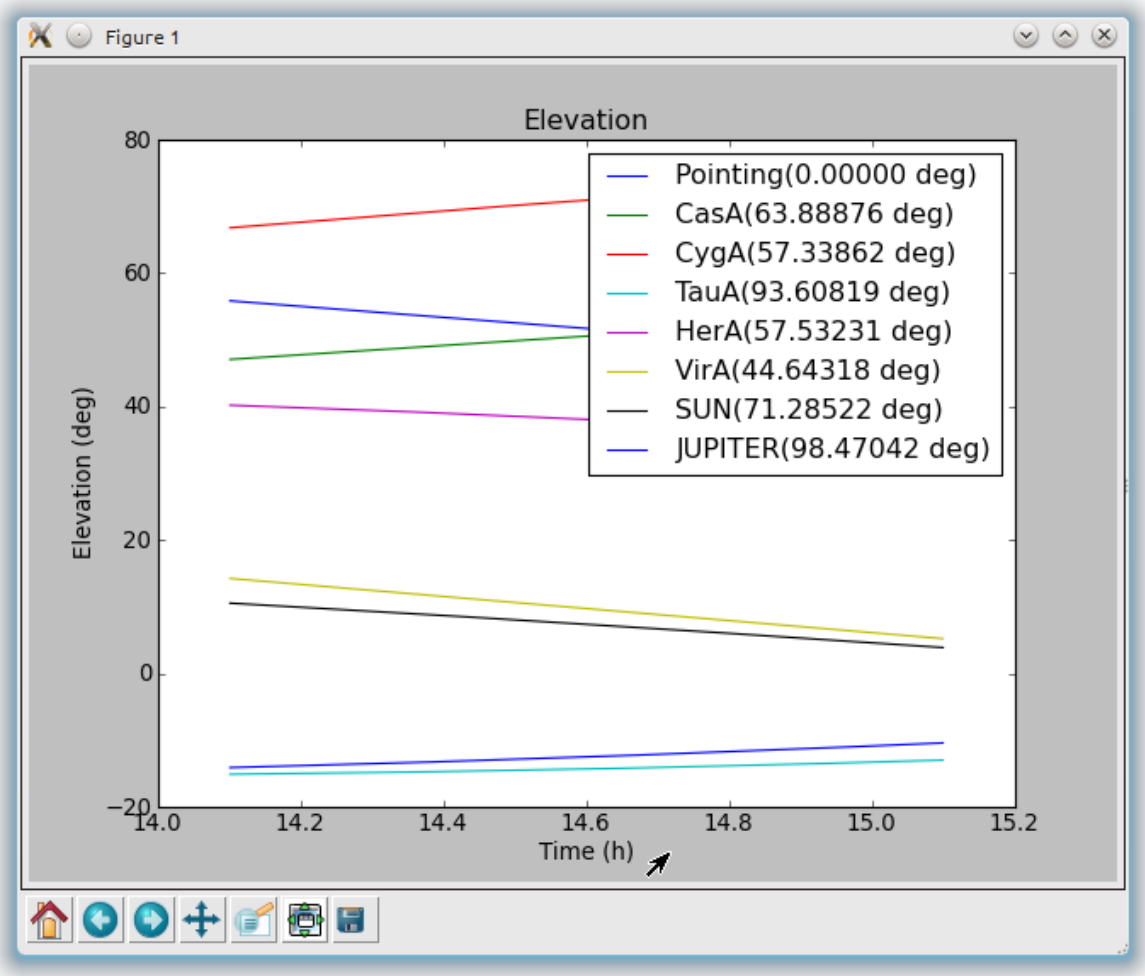


Fig. 16.24: A-team elevation for the LBA observation.

(continued from previous page)

```
demixer.demixfreqstep=64      # compresses to 1 channel
demixer.demixtimestep=12      # compresses 12 time-slots, i.e. 12s
demixer.skymodel=Ateam_LBA_CC.sky
demixer.subtractsources=[CasA, CygA]  # which sources to demix
demixer.type=demixer

> NDPPP NDPPP_LBA_preprocess.parset > log.ndppp.demix 2>&1 &
```

You can run a similar parset without demixing and with averaging only to compare the output.

```
> cat NDPPP_LBA_preprocess_avgonly.parset

msin = L74762_SAP000_SB001_uv.MS
msin.autoweight=TRUE
msin.datacolumn=DATA

msout = L74762_SAP000_SB001_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.strategy=LBAdefault
aoflagger.autocorr=F
aoflagger.count.save=F
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=16     # compresses from 64 to 4 channels
averager.timestep=4     # compresses 5 time-slots, i.e. 5s
averager.type=averager
```

The visibilities after averaging only and after demixing are compared in Fig. 16.25 and in Fig. 16.26. You can clearly see how the demixing has removed the strong A-team signal.

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 120 MB, which is much more manageable than before. You can use msoverview to look at a summary of this data set using.

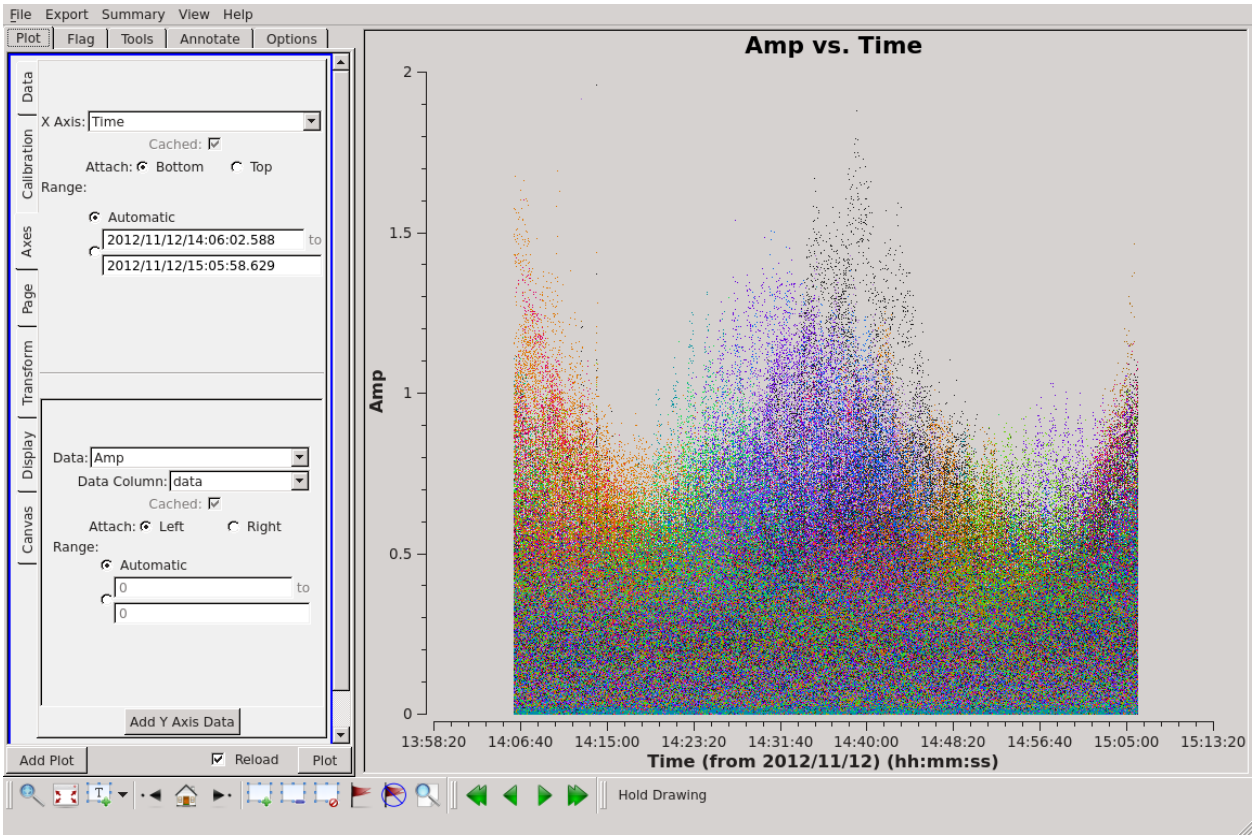


Fig. 16.25: Plotting the XX visibility amplitude against time – averaging only.

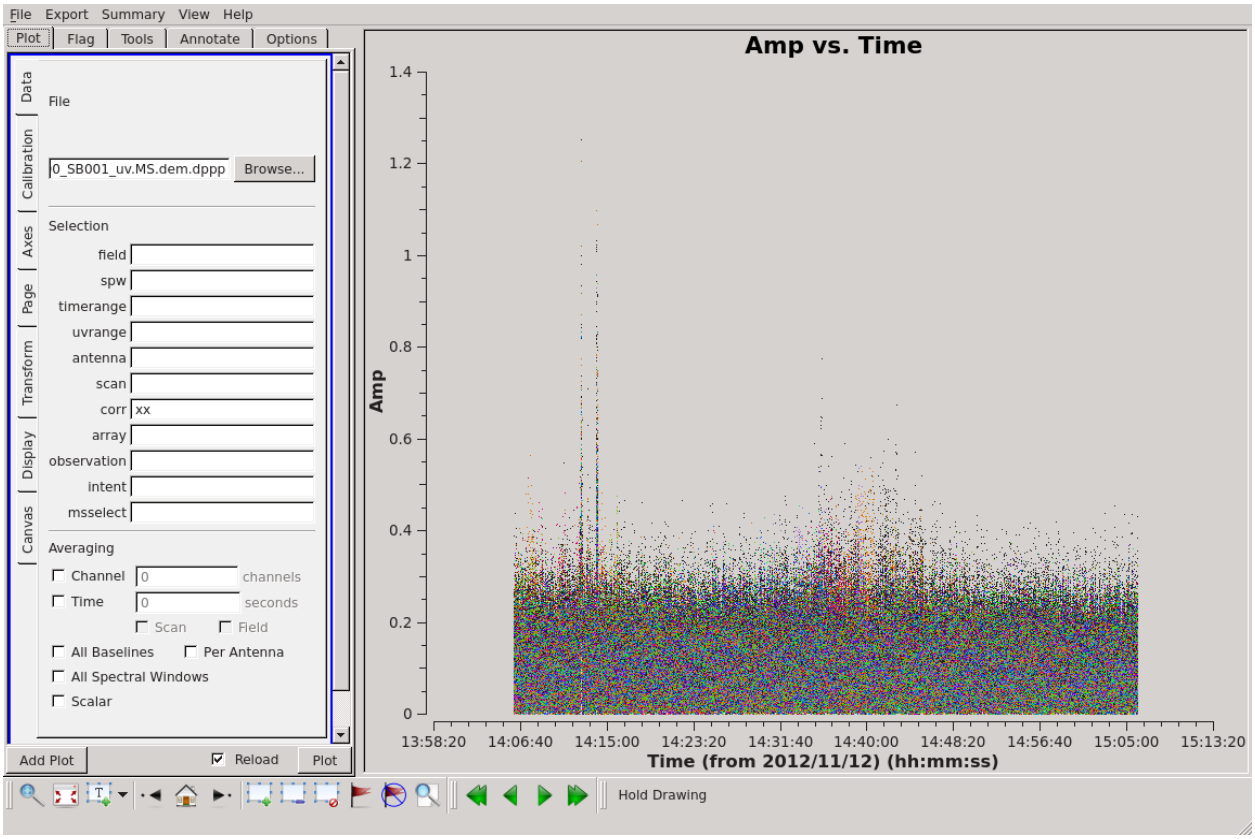


Fig. 16.26: Same as Fig. 16.25 but after demixing. Colourise by 'Antenna1' to obtain these colours.

```
> 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
```

16.7 Post-compression data inspection and flagging

We will use the CASA task plotms to inspect the data. Figure~ref{fig:plotms1_lba} shows the Amp. vs Time and Amp. vs UV distance (wavelengths) for SB000. We can see that there are a few short baselines with large fluctuating amplitudes. We will do some limited flagging for now.

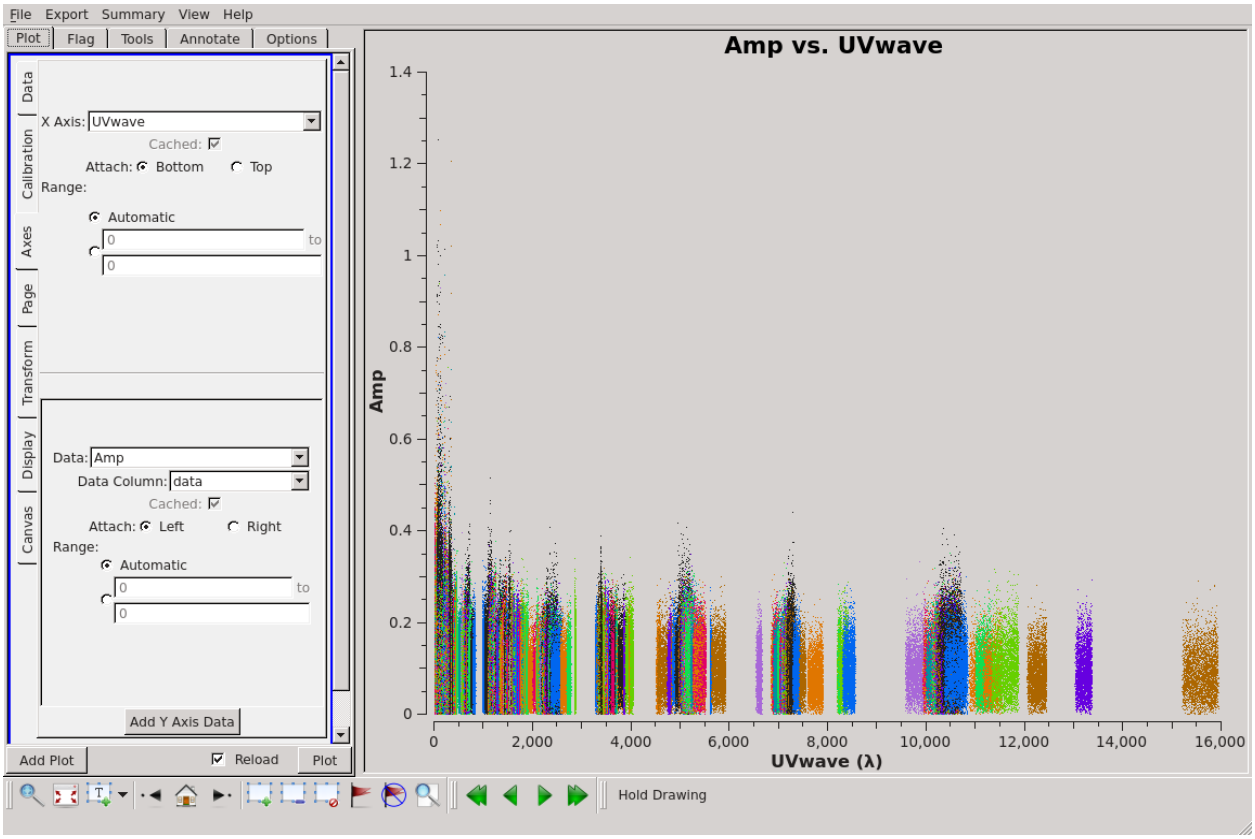


Fig. 16.27: Plotting the XX visibility amplitude against UV distance in wavelengths. Colourise by 'Antenna2' to obtain these colours.

Plotting amplitude against baseline we see that the amplitudes for all baselines to RS305LBA are too low. Also CS302LBA seems to have unusually large amplitudes on most of its baselines.

```
> cat NDPPP_HBA_preprocess.parset

msin = L74762_SAP000_SB000_uv.MS.dem.dppp
msin.datacolumn=DATA
msout =
```

(continues on next page)

(continued from previous page)

```
steps=[flag1, clip]

flag1.type=preflagger
flag1.baseline = [ [CS302LBA], [RS305LBA] ]
```

16.8 Combining Measurement Sets

With the HBA we were able to obtain good solutions for 3C295 within a single subband. At LBA, this is not the case, due to the lower sensitivity of the LBA antennae. For this reason we need to first combine the subbands so we can obtain a single solution in frequency per solution time interval. This is done by

```
> cat NDPPP.combineMS.parset
msin = L74762_SAP000_SB00[01]_uv.MS.dem.dppp
msin.datacolumn = DATA
msout = 3C295_LBA_BAND0.MS

steps = []
```

16.9 Calibration with DPPP

Here we will use DPPP to calibrate the combined MS. We use the same sky model as before

```
> cat 3C295TWO.skymodel
# (Name, Type, Patch, Ra, Dec, I, ReferenceFrequency='150.e6', SpectralIndex) = format

, , 3c295, 14:11:20.64, +52.12.09.30
3c295A, POINT, 3c295, 14:11:20.49, +52.12.10.70, 48.8815, , [-0.582, -0.298, 0.583, -
↪0.363]
3c295B, POINT, 3c295, 14:11:20.79, +52.12.07.90, 48.8815, , [-0.582, -0.298, 0.583, -
↪0.363]
```

and parset file

```
> cat NDPPP.calibrate.parset

msin=3C295_LBA_BAND0.MS
msout=.
msout.datacolumn=CORRECTED_DATA
steps=[gaincal, applybeam]
gaincal.sourcedb=3C295TWO.sourcedb
gaincal.caltype=diagonal
gaincal.nchan=0
gaincal.solint=4
gaincal.usebeammodel=true
gaincal.applysolution=true
```

This is a very simple parset file that solves and corrects the data. Here we are combining all the channels (**nchan=0**) and using a solution interval of 16 sec (**solint=4**, the time resolution of the data is 4 sec). The longer time interval compared to the HBA is again to improve the signal-to-noise. To run DPPP, use the following command

```
> DPPP NDPPP.calibrate.parset  > log.dppp.solve 2>&1  &
```

The calibration process should be completed in a few minutes. You can simultaneously run a similar command for the second sub-band. When DPPP is complete we can look at the calibrated data with parmdbplot.py. Select a few stations and look at the solutions. Fig. 16.28 and Fig. 16.31 shows some solution plots. It is clear that there are a few spikes in the solutions.

```
> python plot_solutions_all_stations_v2.py -p -a 3C295_LBA_BAND0.MS/instrument/ lba_
↪sb000_gains
> display lba_gains_amp.png
> display lba_gains_phase.png
```

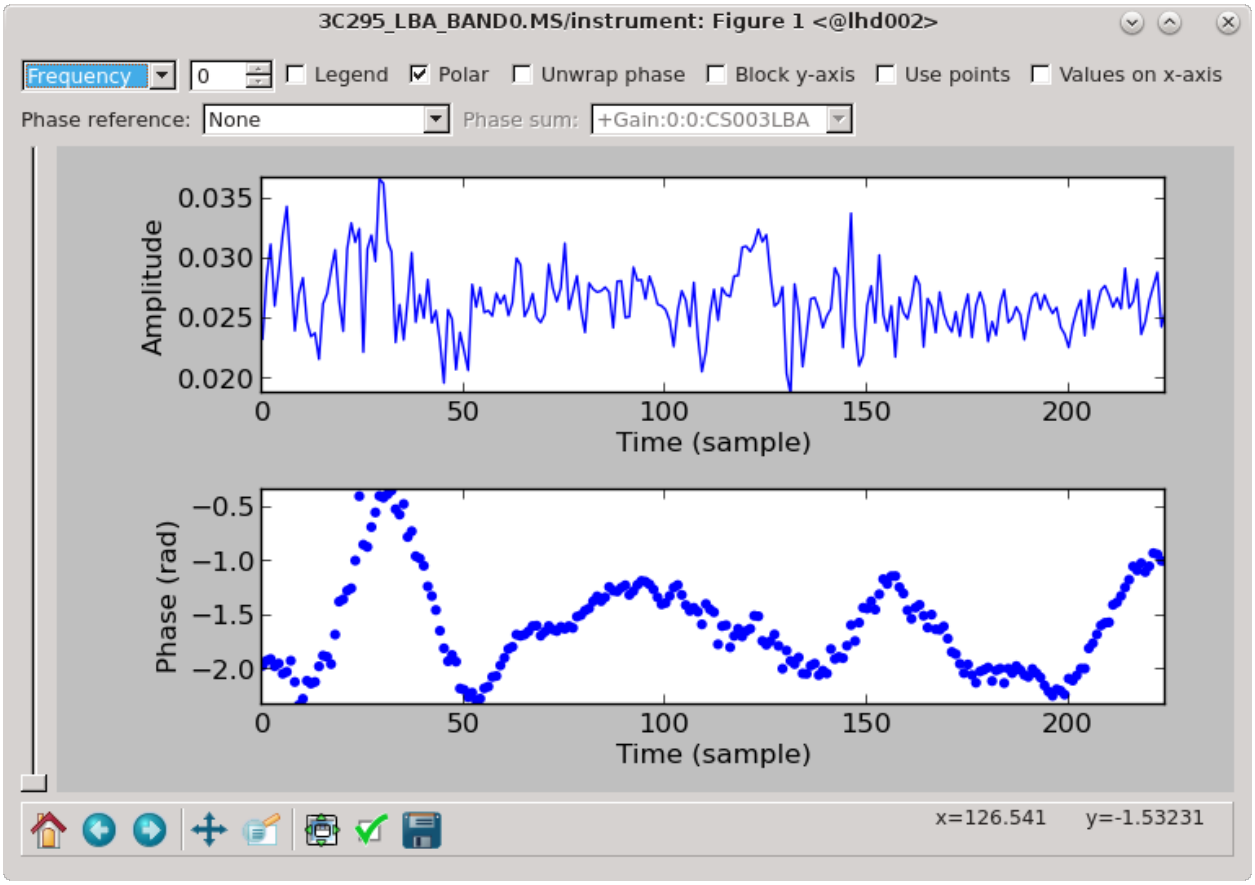


Fig. 16.28: The solutions for CS003LBA.

Once again, we can inspect the corrected data with casaplotms. Figure Fig. 16.32 shows the corrected Amp. vs. Time plots. From this we see again that there are some scans with bad solutions and there is a lot of scatter overall to high amplitudes.

Now we will do some basic flagging with NDPPP using the aoflagger on the CORRECTED_DATA. Fig. 16.33 shows the Amp. vs UV distance plots after flagging.

```
> cat NDPPP_LBA_flag.parset

msin = 3C295_LBA_BAND0.MS
msin.datacolumn=CORRECTED_DATA

msout =
```

(continues on next page)

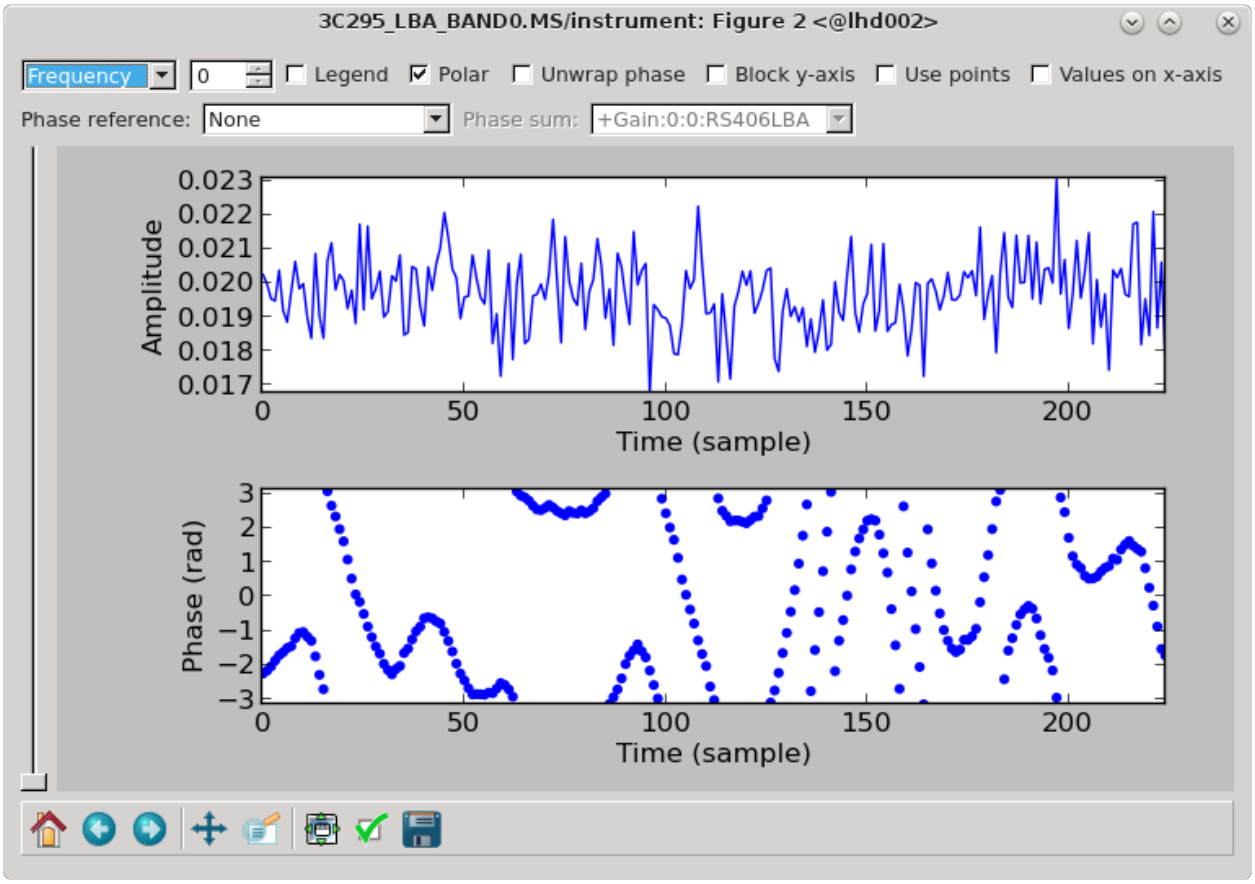


Fig. 16.29: The solutions for RS406LBA.

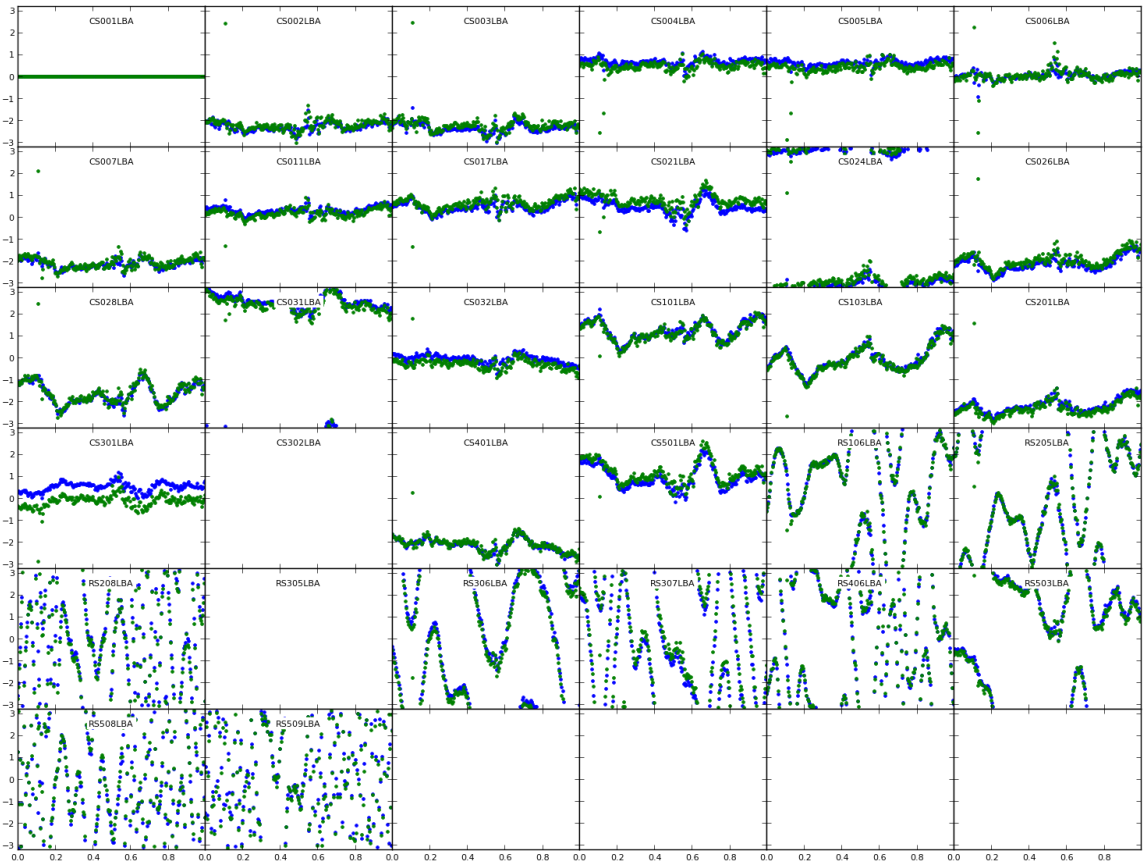


Fig. 16.30: Phase solutions for all stations.

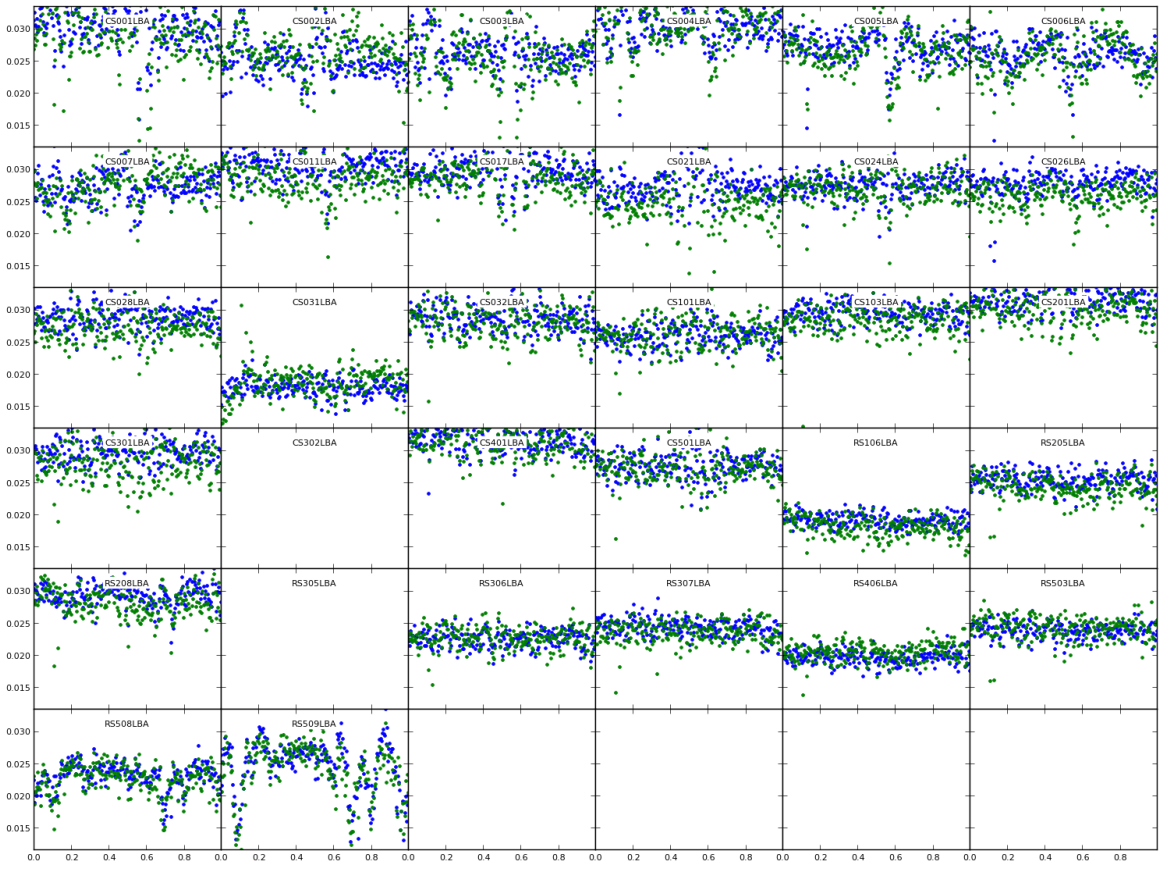


Fig. 16.31: Amplitude solutions for all stations.

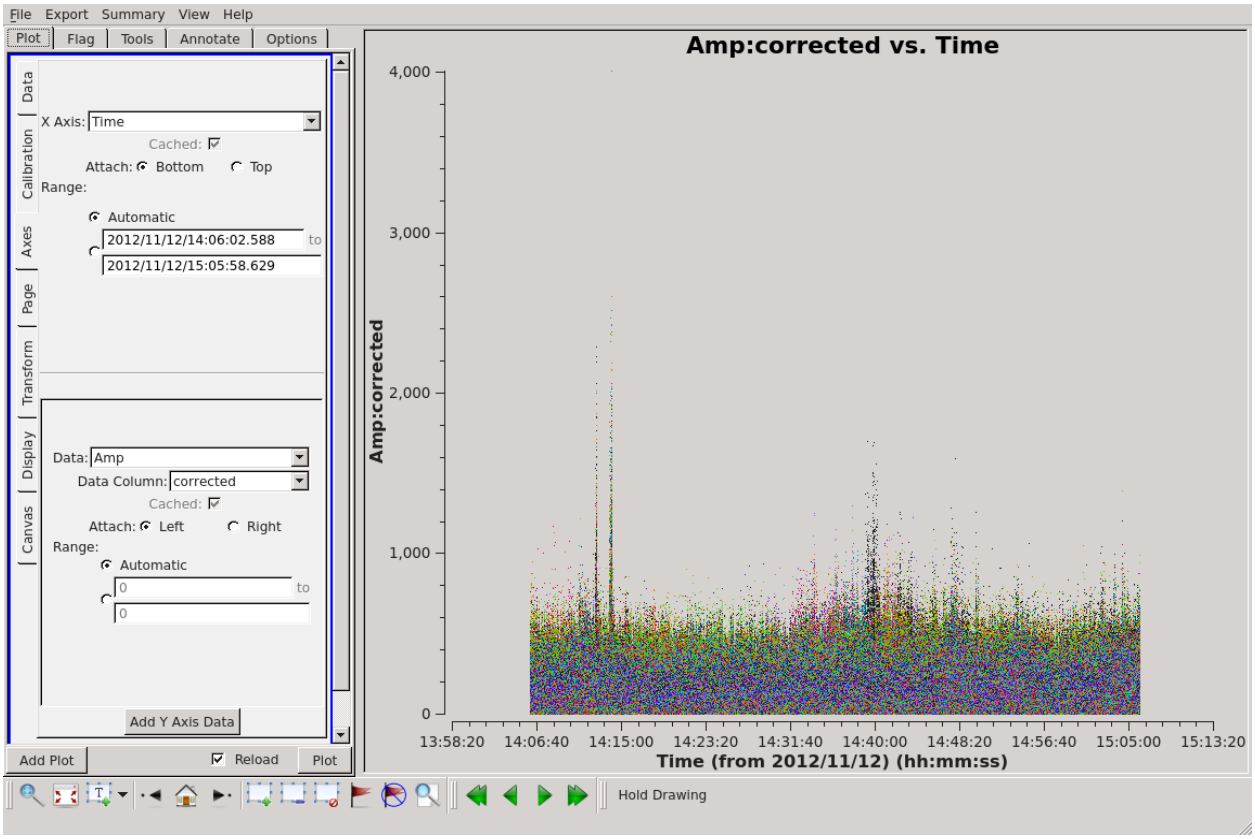


Fig. 16.32: Plotting the corrected XX visibility amplitude against time.

(continued from previous page)

```
steps=[aoflagger]

aoflagger.autocorr=F
aoflagger.timewindow=0
aoflagger.type=aoflagger

> NDPPP NDPPP_LBA_flag.parset > log.ndppp.flag 2>&1 &
```

This should flag about 3-4% of the data.

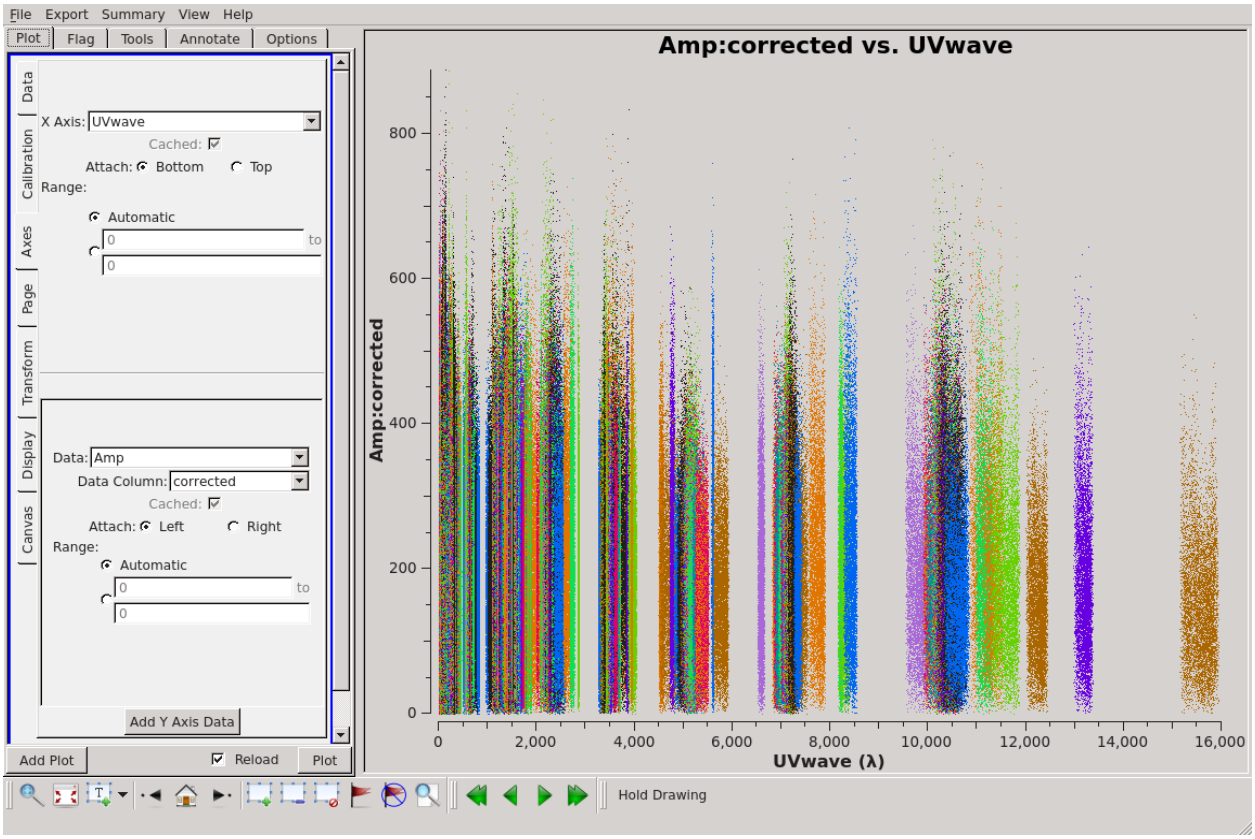


Fig. 16.33: Plotting the XX visibility amplitude against UV distance after flagging.

16.10 Imaging

Here we will again use WSClean to do the deconvolution. 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:

```
> wsclean -name 3C295_LBA -mgain 0.85 -niter 5000 -threshold 0.01  \
-scale 15asec -minuv-l 80 -maxuv-l 2000 -weighting-rank-filter 3 \
-size 2500 2500 -trim 2048 2048  -cleanborder 0   3C295_LBA_BAND0.MS
```

Fig. 16.34 shows the cleaned corrected image. One can see 3C295 at the centre of the field as well as a few other sources. The noise is around 0.15 Jy/beam and 3C295 has a flux density of 136 Jy.

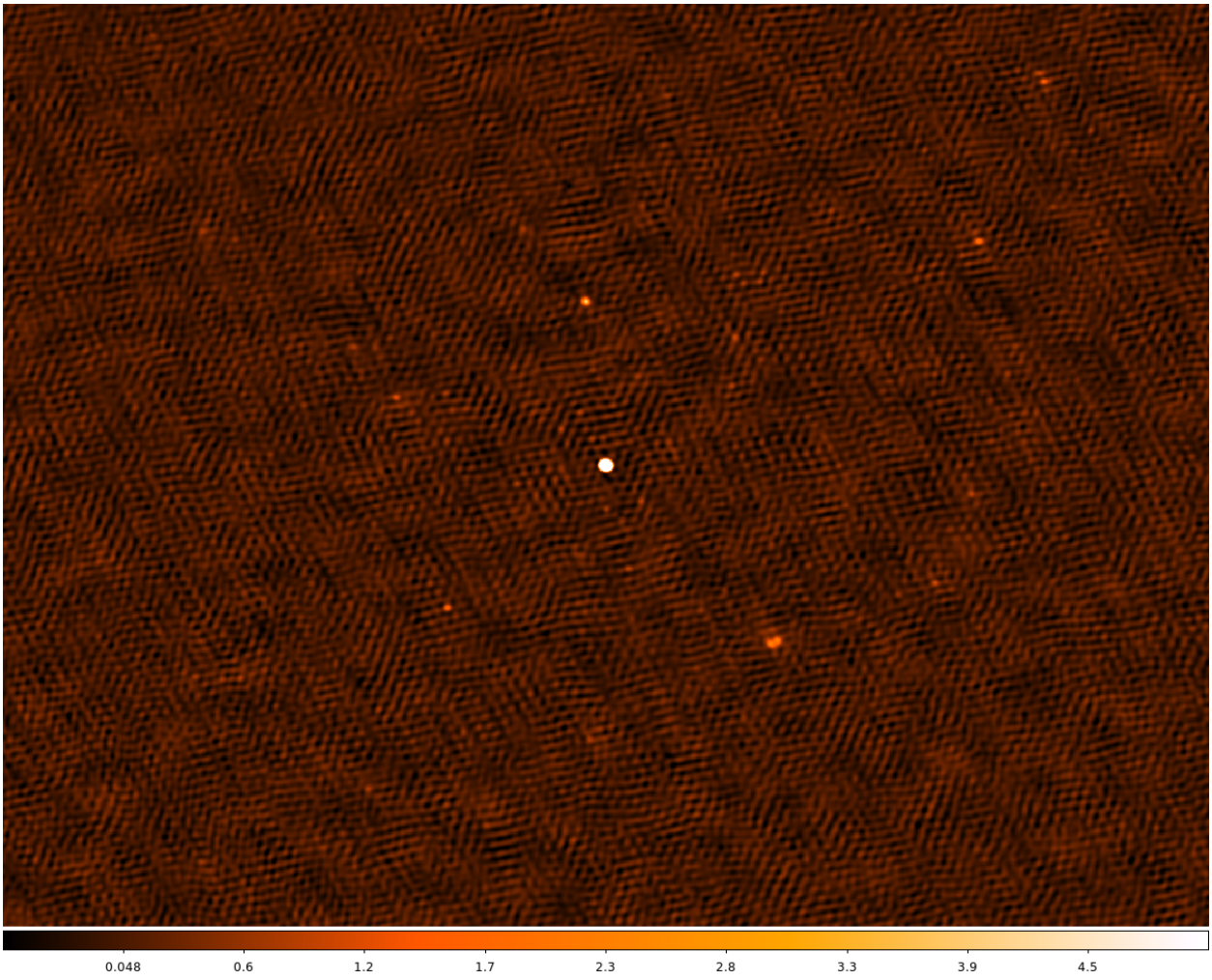


Fig. 16.34: The cleaned LBA image for 3C295. The scale is set to [-0.5, 5].

CHAPTER

SEVENTEEN

ACKNOWLEDGEMENTS

Many commissioners and software developers have contributed to this cookbook. 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 Sarrvesh Sridhar.

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

- **Version 1.0**: Louise Ker
- **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, Andr'e 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, Andr'e 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, Andr'e 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, Andr'e 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, Andr'e Offringa, Emanuela Orr'u, David Rafferty, Cyril Tasse, Bas van der Tol, Reinout van Weeren, Sarod Yatawatta , and Joris van Zwieten

- **Version 13.0**: Roberto Francesco Pizzo, Ger van Diepen, Tammo Jan Dijkema, John McKean, Andr'e Offringa, Emanuela Orr'u, 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, Andr'e Offringa, Emanuela Orr'u, David Rafferty, Cyril Tasse, Bas van der Tol, Valentina Vacca, Nicolas Vilchez, Reinout van Weeren, Wendy Williams and Sarod Yatawatta,
- **Version 15.0**: Roberto~F.~Pizzo, Ger van Diepen, Tammo Jan Dijkema, G. Heald, Francesco de Gasperin, M. Iacobelli, John McKean, Maaijke Mevius, Andr'e Offringa, Emanuela Orr'u, David Rafferty, Cyril Tasse, Bas van der Tol, Valentina Vacca, Nicolas Vilchez, Reinout van Weeren, Wendy Williams and Sarod Yatawatta
- **Version 16.0**: Roberto~F.~Pizzo, Ger van Diepen, Tammo Jan Dijkema, G. Heald, Francesco de Gasperin, M. Iacobelli, John McKean, Maaijke Mevius, Andr'e Offringa, Emanuela Orr'u, David Rafferty, Cyril Tasse, Bas van der Tol, Valentina Vacca, Nicolas Vilchez, Reinout van Weeren, Wendy Williams and Sarod Yatawatta
- **Version 17.0**: Roberto~F.~Pizzo, Ger van Diepen, Tammo Jan Dijkema, G. Heald, Francesco de Gasperin, M. Iacobelli, John McKean, Maaijke Mevius, Andr'e Offringa, Emanuela Orr'u, David Rafferty, Cyril Tasse, Bas van der Tol, Valentina Vacca, Nicolas Vilchez, Reinout van Weeren, Wendy Williams and Sarod Yatawatta
- **Version 18.0**: Aleksandar~Shulevski, Roberto~F.~Pizzo, Ger van Diepen, Tammo Jan Dijkema, G. Heald, Francesco de Gasperin, M. Iacobelli, John McKean, Maaijke Mevius, Andr'e Offringa, Emanuela Orr'u, David Rafferty, Cyril Tasse, Bas van der Tol, T. J. Dijkema, Valentina Vacca, Nicolas Vilchez, Reinout van Weeren, Wendy Williams and Sarod Yatawatta
- **Version 19.0 - 22.0**: Aleksandar~Shulevski, Roberto~F.~Pizzo, Ger van Diepen, Tammo Jan Dijkema, G. Heald, Francesco de Gasperin, M. Iacobelli, John McKean, Maaijke Mevius, Andr'e Offringa, Emanuela Orr'u, David Rafferty, Cyril Tasse, Bas van der Tol, T. J. Dijkema, Valentina Vacca, Nicolas Vilchez, Reinout van Weeren, Wendy Williams and Sarod Yatawatta

CHAPTER

EIGHTEEN

CHANGELOG

The latest released version of the cookbook is available online.

This link is advertised on the LOFAR wiki. The very latest (development) version of the cookbook can also be found on the USG repository.

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.

18.1 Overview of recent changes

18.1.1 Version 23.2 (2019/06/18)

- Chapter "Running LOFAR pipelines inside Docker" updated for singularity

18.1.2 Version 23.0 (2018/04/11)

- Chapter "Running LOFAR pipelines inside Docker" added

18.1.3 Version 22 (2018/01/10)

- Converted source from LATEX to reStructuredText.
- Removed now-obsolete appendix on "Automated self-calibration" and "GNU Screen"
- Replaced all "use" to "module load".
- Replaced DSA with RSA in generating SSH keys.
- Added a new section on generating skymodels from TGSS-ADR.
- Added a section on drawer to Data Inspection.