LOFAR-Sun

Release 0.1.0

Peijin Zhang

Dec 04, 2023

QUICK START

LOFAR data processing for Solar and Space Weather.

Project git repository : LOFAR-Sun-tools

Maintainer: Peijin Zhang (pjzhang.cc)

The LOw Frequency ARray (LOFAR) is a large-scale array of radio telescopes, it can perform multiple flexible type of observations, making it a great instrument for solar and space weather studys:

Solar imaging

- **Interferometry imaging**, high spatial resolution (~ arcmin), low time resolution (>0.1s), low frequency resolution (~0.5 MHz).
- **Tied array beam imaging**, high time resolution (1/96s), high frequency resolution (0.02 MHz), low spatial resolution (~ 0.4 Deg).

Solar spectroscopy

- **Beamformed dynamic spectrum**, no spatial resolution, high time resolution (1/96s), high frequency resolution (0.02 MHz).

Heliosphere

- **Beamformed dynamic spectrum** of quasars, for interplanetary scintillation.
- **Pulsar observations** for solar wind plasma and CMEs.

The common data formats for solar and space weather studys:

- (.MS) Interferometry raw data, measurement set. interferometry
- (.h5) Beamformed data, HDF5 format. beamformed
- (xxx-cube.fits) Beamformed data, fits cube.beamformed
- (xxx-image.fits) Interferometry image data.interferometry

CHAPTER

ONE

INSTALL

1.1 LOFAR imaging software

The dependencies of imaging software is very complex, we recommend using docker or singularity to run the software.
LOFAR imaging with docker

1.2 LOFAR Sun

For the dynamic spectrum and imaging postprocess, we use python package `lofarSun` (*lofarSun <https://github.com/peijin94/LOFAR-Sun-tools>* `). We recommend creating a standalone python enviroment for a more isolated and stable runtime.

```
conda create -n lofarsun python=3.9
```

Then activate the enviroment:

```
conda activate lofarsun
```

There are two ways to install lofarSun package:

(1) From pip, the (relatively) stable version:

```
python -m pip install lofarSun
```

(2) From git the (nighty) dev version:

```
git clone https://git.astron.nl/ssw-ksp/lofar-sun-tools.git
cd lofar-sun-tools
python setup.py install
```

Also, we can use docker to run lofarSun: lofarsundocker

CHAPTER

TWO

INTERFEROMETRY

2.1 Download

LOFAR observation data are available at the Long Term Archive (LTA). The LTA is difficult to navigate and thus, we recommend reading the Long Term Archive Howto on the LOFAR wiki.

Note : Not all data on the LTA is public and you may have to request access from the Solar and Space Weather KSP.

Request the data then download the data with **wget**.

2.2 Preprocess

Raw LOFAR data must undergo a series a preprocessing steps before it can be used for imaging. Namely, these are auto-weighting and calibration.

2.2.1 autoweight

Raw LOFAR data will have no 'WEIGHT' column in its measurement set (MS) thus, it is necessary to do the autoweight step:

```
DPPP msin=path/to/rawdata.MS msout=path/to/output.MS steps=[] msin.autoweight=true
```

This calculates the weights for the visibilities from the auto-correlation data of the observation.

The script `autoWeight.sh` can batch this process if there is more than one file.

(This step can be skipped if the measurement set is pre-processed by imaging pipeline.)

2.2.2 Calibration

Calibration of LOFAR data is performed using the Default Preprocessing Pipeline (DPPP).

In order to use DPPP on the LOFAR computing facilities, the following lines must be run:

```
module load lofar
module load dp3
```

The measurement set is calibrated with DPPP in three steps:

- **Predict Calibration** : use the calibrator observation to predict the gain calibration solution.
- **Apply Calibration** : apply the gain calibration solution to the solar observation.

- **Apply Beam** : apply the LOFAR beam to the dataset.

These steps are covered in greater detail in the LOFAR Imaging Cookbook. Below we give a short example of how to calibrate a raw LOFAR observation.

Predict Calibration

1. Create a source.db file with a sky model of the calibrator. Here we use TauA as an example but other skymodels are available from https://github.com/lofar-astron/prefactor/tree/master/skymodels.

```
makesourcedb in=/path/to/TauA.skymodel out=TauA.sourcedb
```

2. Run auto-weight with DPPP (Note: this step should be done for both sun and the calibrator):

```
DPPP msin=/path/to/calibrator.MS
msout=/path/to/calibrator-autow.MS
steps=[]

msin.autoweight=True
```

3. Use the observation of the calibrator to predict the parameters for the calculation applied to the solar observation.

```
DPPP msin=/path2/to/calibrator-autow.ms \
Msout=. \
steps=[gaincal] \
gaincal.usebeammodel=True  \
gaincal.solint=4 \
gaincal.sources=TauA \
gaincal.sourcedb=TauA.sourcedb \
gaincal.onebeamperpatch=True \
gaincal.caltype=diagonal
```

Apply Calibration

4. Apply the parameters predicted by step (3)

```
DPPP msin=/path2/to/sun-autow.ms \
msout=. \
msin.datacolumn=DATA \
msout.datacolumn=CORR_NO_BEAM \
steps=[applycal] \
applycal.parmdb=/path/to/calibrator-autow.MS/instrument \
applycal.updateweights=True
```

Apply Beam

5. Apply the beam model of the calculation for the LOFAR station:

```
DPPP msin=sun-autow.MS \
msout=. \
msin.datacolumn=CORR_NO_BEAM \
msout.datacolumn=CORRECTED_DATA \
steps =[applybeam] \
applybeam.updateweights=True
```

The steps (2)-(5) are integrated in the script `auto_sun_calib.py` to calibrate the MS files in batch.

The script auto_sun_calib.py automizes the calibration of LOFAR observations. It generates the parset file for the calibration and runs the corresponding DPPP commad.

Modify the configuration lines in the code:

```
sources  = 'TauAGG'  # source type
sourcedb = 'taurus_1.sourcedb' # path to the source

sun_MS_dir   = 'MS/' # path to the dir contain sun's MS
calib_MS_dir = 'MS/' # path to the dir contain calibrator's MS

obs_id_sun   = 'L722384' # obsid of the sun
obs_id_calib = 'L701915' # obsid of the calibrator

idx_range_sun  = [32,39] # index range of the subband of the Sun
idx_range_cali = [92,99] # index range of the subband of the Sun

run_step = [0,1,2]; # 0 for predict; 1 for applycal;  2 for applybeam
# [0,1,2] for complete calibration
```

Run the calibration script simply with:

```
python auto_sun_calib.py
```

2.3 Inspecting Calibration Solutions

In many cases, solar radio bursts can contaminate the observation of the calibrator source. It is thus **highly recommended**[1] that the gain calibration solutions obtained with DPPP are visually inspected. The LOFAR Solution Tool (LoSoTo) can be used to plot the calibration solutions for each antenna pair using the `cal_solution_plot.parset` file below

```
[plot_amp]
operation = PLOT
axesInPlot = [time,freq]
axisInTable = ant
axisInCol = pol
soltab = sol000/amplitude000
markerSize = 4
```

(continues on next page)

[1] LoSoTo: LOFAR solutions tool

(continued from previous page)

```
plotFlag = True
makeAntPlot=False
doUnwrap = False
prefix=/path/to/calibration/solution/plots/output_name_amp_
[plot_phase]
operation = PLOT
axesInPlot = [time,freq]
axisInTable = ant
axisInCol = pol
soltab = sol000/phase000
markerSize = 4
plotFlag = True
makeAntPlot=False
doUnwrap = True
prefix=/path/to/calibration/solution/plots/output_name_phase_
```

Use `parmdb2H5parm.py` to convert the calibration solutions stored at `/path/to/calibrator-autow.MS/instrument` to a H5 file compatible with LoSoTo and generate the calibration soultion plots as follows:

```
parmdb2H5parm.py -v cal_solutions.h5  /path/to/calibrator-autow.MS/instrument
losoto -v cal_solutions.h5 cal_solution_plot.parset
```

2.4 Clean

Once the data has been calibrated, it can be imaged. We recommend using WSClean for this. An example of WSClean for the sun:

```
wsclean -j 40 -mem 30 -no-reorder -no-update-model-required \
-mgain 0.3 -weight briggs 0 -size 512 512 \
-scale 10asec -pol I -data-column CORRECTED_DATA \
-niter 1000 -intervals-out 1 -interval 10 11 \
-name /path/to/prefix \
/path/to/sun-autow.MS
```

it is better to keep the parameter **-multiscale** on for the solar image CLEAN, because the solar radio emission is always extended.

A small cheatsheet for solar WSClean:

Command	Parameter	Comment
-j	20	Number of thread used for CLEAN (can be equal to the number of cores).
-mem	80	Maximum memory limit in percent to the system memory. (Don't use 100%)
-weight	briggs 0.2	Weight for the baselines. Briggs 0 works for most of the situations
-size	2048 2048	Size of the image in pixel.
-scale	3asec	The scale of one pixel, can be 0.1asec,3asec, 3min, 3deg. This should be less 1/4 the beam size in order to properly sample the beam.
-pol	I	The polarization for cleaning, can be I,Q,U,V.
-multiscale	\	Whether to use multiscale in the clean. Better to switch on for extended source.
-data-column	CORRECTED_DATA	Be sure to use the calibrated data (CORRECTED_DATA).
-niter	2000	The iteration of clean, for the sun, 400 is necessary, 1000 can be better, 2000 is enough.
-intervals-out	85	How many images you want to produce.
-interval	3000 4000	The index range for the CLEAN.

For the interval index, one can use the `get_datetime_index.py` to find out the starting and ending index.

2.5 Visualization

WSClean produces fits image with astronomy coordinate [RA,DEC] and the unit of Jy/Beam, the module *lofarSun.IM* can transform the coordinate to heliocentric frame and convert the flux to brightness temperature distribution according to the equation given in the Equation given in Flux intensity.

A demo of visualizing lofar interferometry : demo

For the use of jupyterlab, port forwarding can be done with:

```
ssh -L 1234:localhost:1234  username@server_address

source /data/scratch/zhang/conda_start.sh

python -m jupyter notebook --no-browser --port=1234
```

Change username and 1234 accordingly.

2.6 Docker

Above steps requires LOFAR software, which is not easy to install. We can use docker to run steps.

For calibration we use the docker image from here.

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

(in docker) $ source /opt/lofarsoft/lofarinit.sh

(in docker) $ DPPP --version
```

For Visualization we use the docker image "peijin/lofarsun"

```
$ docker run --rm --hostname lofarsoft -p 8899:8899 \
    -v /HDD/path/to/data/:/lofardata peijin/lofarsun \
    /bin/bash -c "jupyter-lab --notebook-dir=/lofardata \
    --ip='*' --port=8899 --no-browser --allow-root"
```

This command will start a jupyter lab server in the docker container, also mount the directory '/HDD/path/to/data/' to '/lofardata'

lofarSun.IM.**get_peak_beam_from_psf**(*fname*, *thresh=0.618*)

Get the peak beam from a fits file

CHAPTER

THREE

LINCSUN

The imaging data processing pipeline based on LINC .

3.1 Requirements

- Singulartiy
- CWL
- LINC

The tool is packaged as a singularity container, no need to install and configure the dependencies and software.

To pull the container:

```
singularity pull docker://peijin94/lincsun:latest
```

To run the container as a shell:

```
singularity -B /my/data:/my/data shell lincsun_latest.sif
```

3.2 Prepare data and src

Data can be downloaded from LTA here.

Assuming the data is stored in the directory '/path/to/data', with all MS files in the subdirectory 'MS'.

The data processing directory is '/path/to/proc'.

Download the source code:

```
cd /path/to/proc
git clone https://github.com/peijin94/LOFAR-Sun-tools.git
cp -r ./LOFAR-Sun-tools/utils/IM/LINC ./
```

3.3 Calibration overview

The workflows and steps of the data procedure is described in CWL files, can be found in the directory 'LINC/lincSun'

We need to prepare a json file to describe the data and the parameters for the data processing.

For example:

```
{
   "msin": [
      {
            "class": "Directory",
            "path": "/path/to/data/MS/Data001.MS"
      },
      {
            "class": "Directory",
            "path": "/path/to/data/MS/Data001.MS"
      }
   ],
   "ATeam_skymodel": {
      "class": "File",
      "path": "/path/to/somewhere/else/A-Team_lowres.skymodel"
   },
   "refant": "CS002LBA"
}
```

This example json file tells the workflow to process Data001.MS and Data002.MS, with the A-Team skymodel and the reference antenna CS002LBA.

Then we can run the workflow with cwltool command inside the container "LINC", for example:

```
singularity exec -B /path/to/data/ /path/to/contianer/linc_latest.sif \
   cwltool --outdir /path/to/proc/save /path/to/proc/LincSun/workflow/calibrator_sun.cwl␣
↪\
   /path/to/proc/calib.json
```

There are two steps in the workflow: **gaincal** and **applycal**, both are done with above way.

The directory or file list of the calibrator or target could be very long when there are many subbands, we can use the script 'create_jsonlist.py' to create the list of files.

3.4 Gaincal

- collect the data for the calibrator files (measurement set files xxx.MS), could be done with the script 'create_jsonlist.py'
- prepare the script to run the cwl workflow, for example:

```
#!/bin/bash
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=20
#SBATCH -J LcalibPJ
#SBATCH --account=xxxxxxxxxx
#SBATCH --mem=128G
```

(continues on next page)

(continued from previous page)

```
#SBATCH --partition=xxxxxxxxxxxxxxxxxx
#SBATCH --time=23:59:00
#SBATCH --mail-user=xxxxxxxxxxxxxxxxxxxxxxx

mytmpdir=/dev/shm/largefastdisk/
export SINGULARITY_TMPDIR=$mytmpdir
export TMPDIR=$mytmpdir
export TEMP=$mytmpdir
export TMP=$mytmpdir
procDir=/path/to/proc/

singularity exec -B /path/to/data -B $PWD $procDir/linc_latest.sif \
      cwltool  --no-container --basedir $procDir \
      --outdir $procDir/results/ --log-dir $procDir/logs/ \
      --tmpdir-prefix $mytmpdir \
      $procDir/lincSun/workflow/calibrator_sun.cwl $procDir/calibLBA.json
```

The first a few lines of '#SBATCH' is for HPC users, ignore if you are running on local machine or single node.

This will create inspection plots for all antennas.

Please check the phase and amplitude solutions for each antenna.

If there is very bad antenna, we need to do one more extra step to flag out the corrupted antenna.

Successful run will yield a solution.h5 file in the results directory

3.5 Applycal

Apply the solution file to the target MS

CHAPTER

FOUR

BEAMFORMED

4.1 Brief

4.1.1 Format

Dynamic spectrum raw data is stored in the form of HDF5 file, which is a type of self described file. The metadata is in the header of the HDF5 file, and the data is stored in the body of the file. Here is a general exampel of how to retrive the information from header and plot the dynamic spectrum with `h5py`. Tutorial .h5

4.1.2 Size

Raw data is always huge, a typical observation of 2 hours, saved in hdf5 file as `float32`, time resolution 1/96 second, 4096 frequency channels, the data size of one beam single polarization is: $4[\mathrm{Byte}] \cdot 7200[\mathrm{s}] \cdot 96 \cdot 4096[\mathrm{ch}] = 10.54\mathrm{GB}$. To form a tied array beam (TAB) imaging, there will be 127 beams, with 4 polarizations, the total size of 2 hours of TAB observation is: **5.3 TB.**

4.2 Calibration

The dynamic spectrum download from Long-time-archive (LTA) is not calibrated, the observation is usualy performed with a calibrator observation, we can have gaincal calibration for the dynamic spectrum by doing the following steps:

- Download the target and calibrator observation from LTA
- Apply averaging on the calibrator observation to get a instrument reponse of the calibrator ($I_c(\nu)$)
- Calculate the gain with the $\mathrm{gain} = \frac{M_c(\nu)}{I_c(\nu)}$,where M_c is the model of the flux of the target object (e.g. CasA, VirA...).
- Apply the gain to the target (Sun) dynamic spectrum with $I_t = I_{t0} \times \mathrm{gain}$

With this method, we can have a roughly calibrated dynamic spectrum, while the beam pointing is not considered in this procedure, the errorbar could be **very** large.

4.3 Quick View

To cope with the large data size, providing some quick access to the dynamic spectrum, we can downsample the data to a smaller dimension.

4.3.1 FitsCube

This is a demo to show how to use FitsCube to read the data and produce the TAB image. Demo

Quick View of the LOFAR beamform:

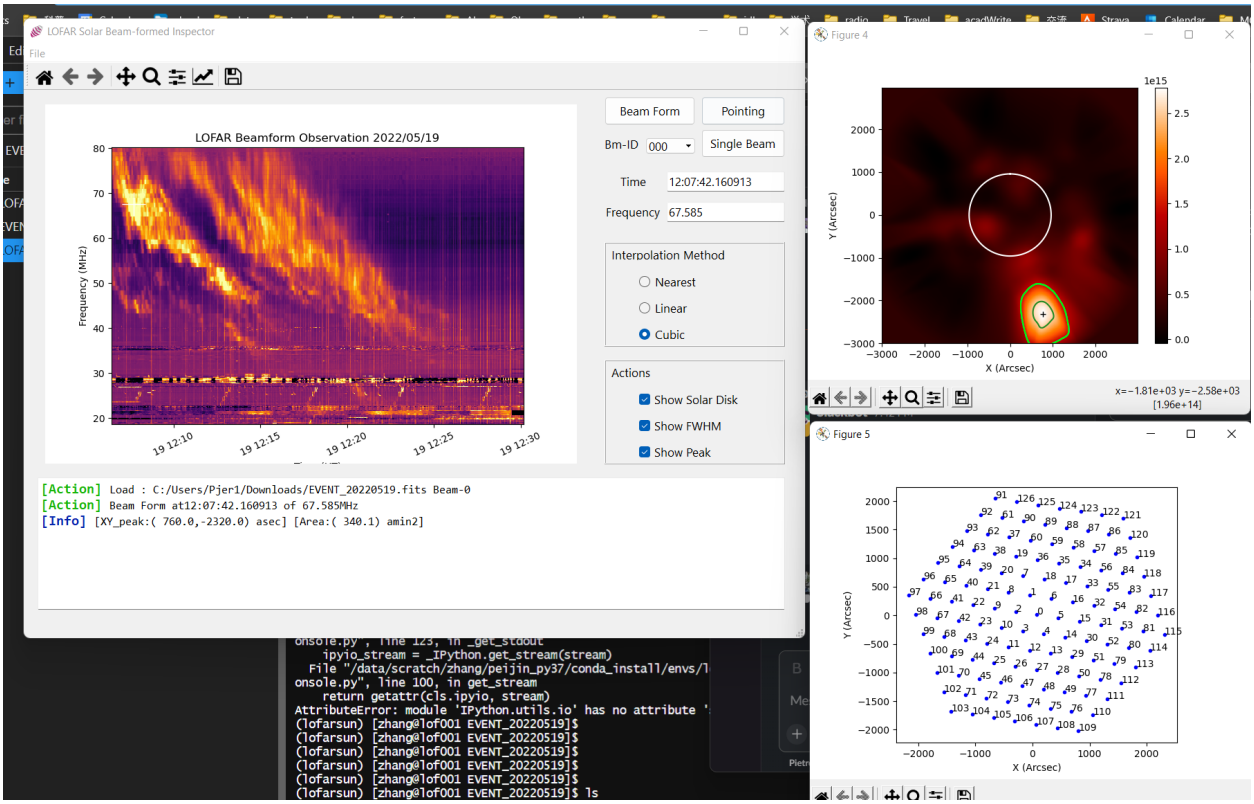
(1) Go to a empty directory, run the following command.

```
git clone https://github.com/peijin94/LOFAR-Sun-tools.git
cd LOFAR-Sun-tools
# (conda activate xxx)
python setup.py install
```

(2) Run quick view for TAB-cube-fits:

```
# (conda activate xxx)
lofarBFcube
```

(3) Then load beamformed imaging fits and preview:



CHAPTER

FIVE

EXECUTABLE CLIENT TOOLS

5.1 GUI

`lofarBFcube` : GUI tool to load fits cube data for LOFAR beamformed observations

5.2 Command line tools

5.2.1 Dynamic Spectrum

- `h5toFitsDS`

```
usage: h5toFitsDS [-h] [--t_c_ratio T_C_RATIO] [--f_c_ratio F_C_RATIO]
                  [--flip_freq] [--minutes_per_chunk MINUTES_PER_CHUNK]
                  [--num_chunks NUM_CHUNKS] [--chop_off] [--averaging]
                  [--flagging] [--t_idx_cut T_IDX_CUT] [--target_directory]
                  [--date_directory] [--simple_fname] [--add_ksp_logo]
                  [--add_idols_logo]
                  beamformed_dataset output_directory

Split and down sample the dynamic spectrum of LOFAR observation Input : Huge
hdf5 file of LOFAR Tied array beam formed observation Output : Small fits file
with json and png quickview (by Peijin.Zhang & Pietro Zucca 2019.08)

positional arguments:
  beamformed_dataset    Input BeamFormed dataproduct
  output_directory      output directory

optional arguments:
  -h, --help            show this help message and exit
  --t_c_ratio T_C_RATIO
                        compression ratio of the time
  --f_c_ratio F_C_RATIO
                        compression ratio of the frequency
  --flip_freq           flip the frequency axis (set to upper start)
  --minutes_per_chunk MINUTES_PER_CHUNK
                        time chunks in minutes
  --num_chunks NUM_CHUNKS
                        number of output chunks, -1 means to the end
  --chop_off            chop every **interger** 15 minutes
```

(continues on next page)

(continued from previous page)

```
                        [00:15,00:30,00:45....], default True
  --averaging           use averaging to downsize, otherwise use sampling
  --flagging            flag data before averaging
  --t_idx_cut T_IDX_CUT
                        length of t-index for convolution, 8192 means about
                        195MB for 6400 channels
  --target_directory    use long directory name, (e.g. out/sun/xxx.fits)
  --date_directory      use long directory name, (e.g.
                        out/2020/12/12/xxx.fits)
  --simple_fname        use simple file name, for daily summary, format
                        YYYYMMDD_hh (e.g. 20201212_02.fits)
  --add_ksp_logo        showing ksp logo
  --add_idols_logo      showing idols logo
```

5.2.2 measurement set

- pymsOverview

```
usage: pymsOverview [-h] [-v] FILE

positional arguments:
  FILE           MS file with full directory

optional arguments:
  -h, --help     show this help message and exit
  -v, --verbose  detailed info
```

- pymsCookWscleanCMD

```
usage: pymsCookWscleanCMD [-h] [--intervals-out INTVO]
                          [--interval INTV1 INTV2] [--elipbeam]
                          [--datacol DATACOL] [--misc [MISC ...]]
                          [--scalefactor SCALEFACTOR] [--multiscale]
                          [--briggs BRIGGS] [--name [NAME ...]] [--fov FOV]
                          FILE

positional arguments:
  FILE                  MS file with full directory

optional arguments:
  -h, --help            show this help message and exit
  --intervals-out INTVO
                        Number of intervals to output, default is -1,
                        representing for snapshot for all
  --interval INTV1 INTV2
                        Index intervals for imaging, default is '-1 -1'
                        representing for all intervals
  --elipbeam            Use eliptical beam for wsclean, default True, set to
                        False to use circular beam
  --datacol DATACOL     Data column to use, default is CORRECTED_DATA
  --misc [MISC ...]     Miscellaneous options for wsclean, default is empty
```

(continues on next page)

(continued from previous page)

```
  --scalefactor SCALEFACTOR
                        Scale factor for wsclean, default is 3.0
  --multiscale          Use multiscale for wsclean, default is False
  --briggs BRIGGS       Briggs robust parameter for wsclean, default is -0.5
  --name [NAME ...]     Name of the output image, default is empty
  --fov FOV             Field of view for wsclean (asec), default is 10000
```

- pymsDatetime2Index

```
usage: pymsDatetime2Index [-h] [-t TIME] [--fmt FORMAT] FILE

positional arguments:
  FILE                  MS file with full directory

optional arguments:
  -h, --help            show this help message and exit
  -t TIME, --time TIME  default time format is %H:%M:%S.%f, can be changed by
                        -fmt
  --fmt FORMAT, --format FORMAT
                        default is %H:%M:%S.%f
```

- pymsIndex2Datetime

```
usage: pymsIndex2Datetime [-h] [-i IDX] [--fmt FORMAT] FILE

positional arguments:
  FILE                  MS file with full directory

optional arguments:
  -h, --help            show this help message and exit
  -i IDX, --idx IDX     index of the slot
  --fmt FORMAT, --format FORMAT
                        default is %H:%M:%S.%f
```

CHAPTER

SIX

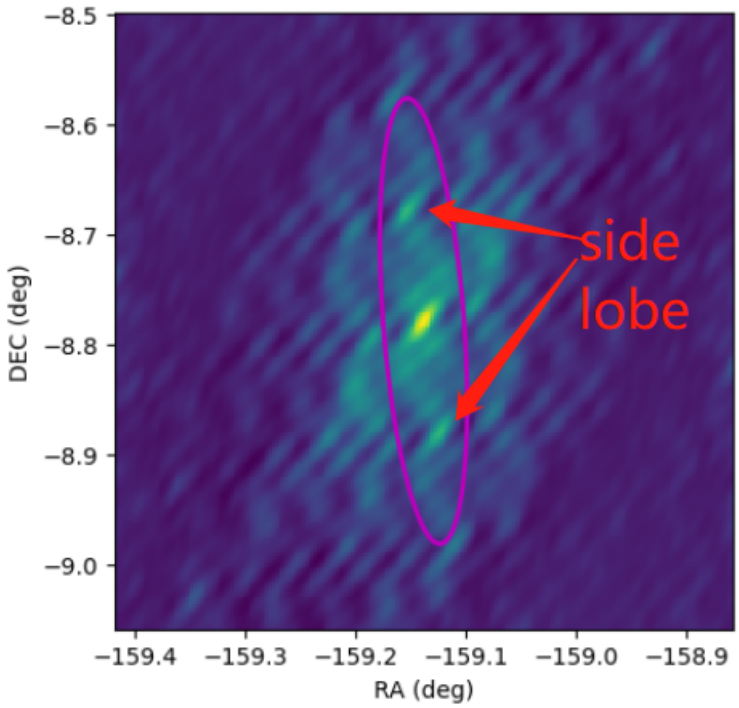
INTERFEROMETRY BEAM

For snapshot imaging with remote stations, (which is the most common imaging way for solar radio observations to resolve time domain variation), the uv coverage is usually sparse. As a result, the synthesized beam is usually irregular with significant side-lobes. And the beam is varying with time due to the earth rotation:

6.1 Restoring beam shape

Common procedure is to create PSF from UV configuration and restore beam from PSF, then use the beam to deconvolve the image.

By default, wsclean fit the PSF to a 3-parameter elliptic Gaussian function to get the restored-beam, but some times the fitting take the side-lobe into account, especially for snapshots with long baselines (sparse UV distribution creates irregular PSF).



there are several parameters to tweak when this happens:

1. `-circular-beam` : force the beam to be circular then the beam is highly elliptical (https://wsclean.readthedocs.io/en/latest/restoring_beam_size.html)
2. `-theoretical-beam` : use the theoretical beam instead of the fitted beam, usually used in inspections or quick imaging, not recommended for final image.
3. `-beam-shape` : input a beam shape as the restored-beam to do the convolution. (`useMyBeam` in next section)

6.2 useMyBeam

To have more flexibility, we can generate beam shape from the PSF with our own algorithm, and use the beam shape to do the convolution.

Steps:

1. Create PSF with the same input parameter as the planed final image, but with the option `-make-psf-only` and `-no-fit-beam`

```
wsclean -mem 90 -no-reorder -no-update-model-required  -mgain 0.8 \
   -weight  briggs 0.5  -auto-mask 3 -auto-threshold 0.3 -size 934 934 \
   -scale 10.701100379915806asec -pol I -data-column CORRECTED_DATA \
   -interval 100 101 -intervals-out 1 -make-psf-only -no-fit-beam \
   -niter 1200 -name ./tmp/dummy path/to/MS
```

this command will create a PSF fits file at `./tmp/dummp-psf.fits`

2. use customized method to obtain a beam shape:

```
pymsPSFfitPeakGauss ./tmp/dummy-psf.fits
```

this command returns something like:

```
59.572246348asec 24.846714688asec -148.7979535deg
```

3. use the beam shape to do the convolution:

```
wsclean -mem 90 -no-reorder -no-update-model-required  -mgain 0.8 \
   -weight  briggs 0.5  -auto-mask 3 -auto-threshold 0.3 -size 934 934 \
   -scale 10.701100379915806asec -pol I -data-column CORRECTED_DATA \
   -interval 100 101 -intervals-out 1 -no-fit-beam \
   -beam-shape 59.572246348asec 24.846714688asec -148.7979535deg \
   -niter 1200 -name ./imgfits/IM path/to/MS
```

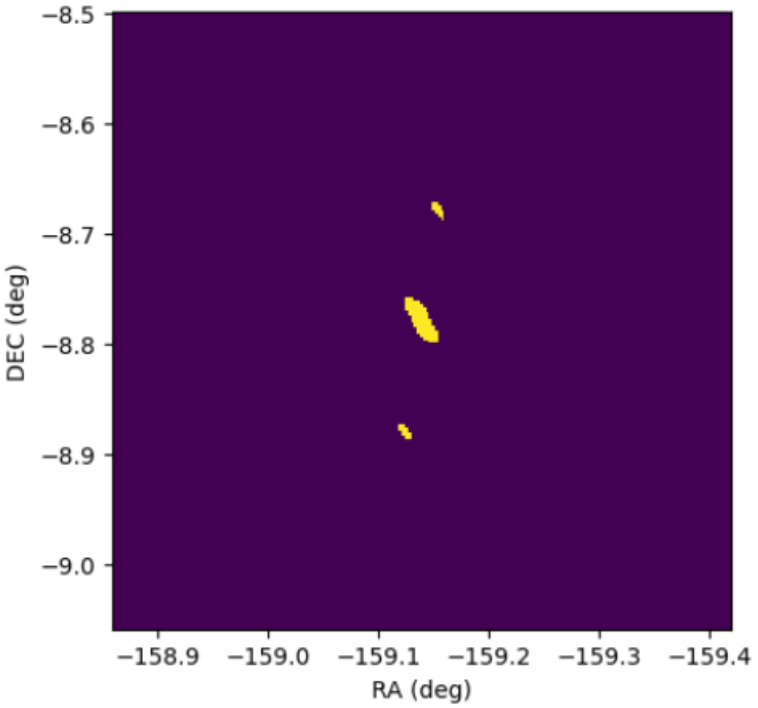
6.2.1 Algorithm used for beam shape determination

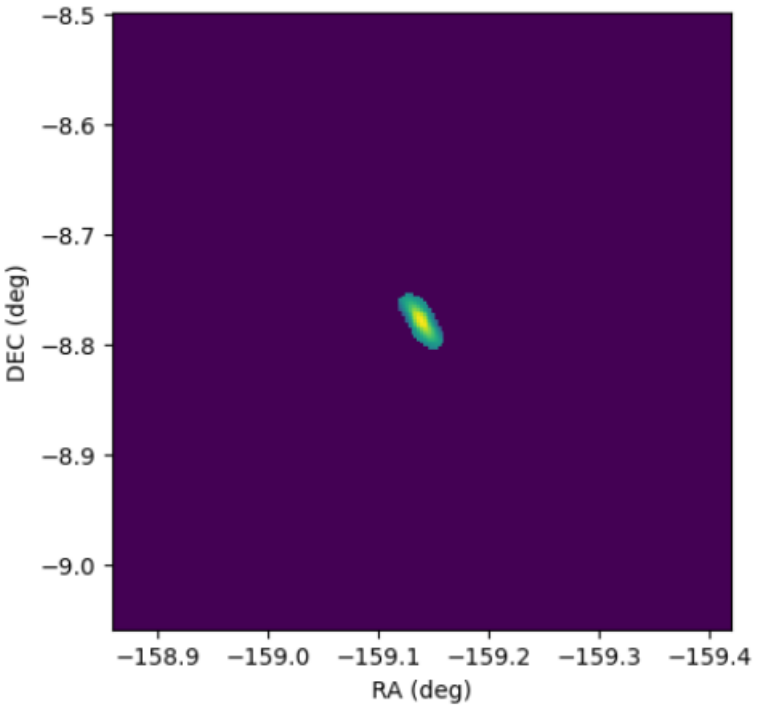
Threshold the PSF image the a ratio (e.g. 0.5) of the maximum value,

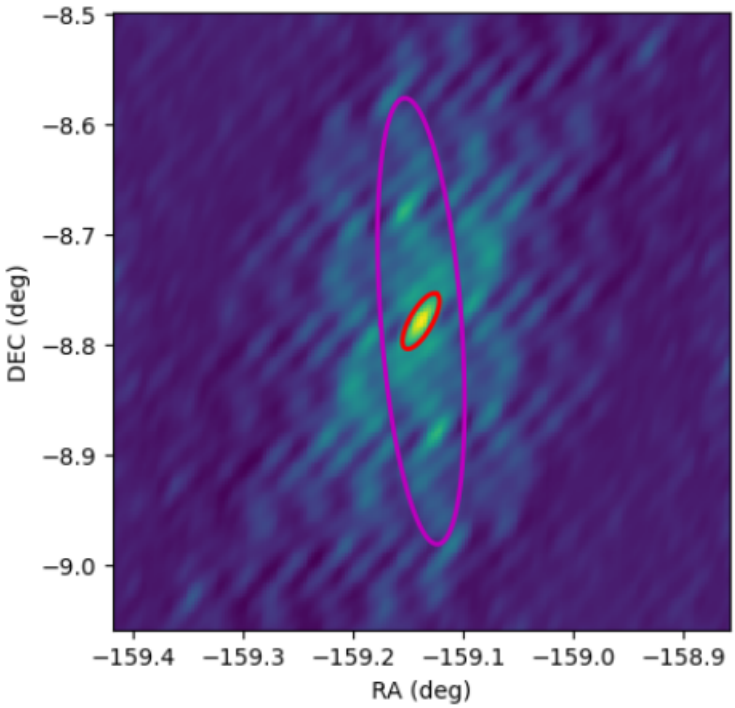
Select only the region covers the pixel of maximum. Fit Gaussian functions with the pixels inside.

Convert the fitted Gaussian parameters to the beam shape parameters. result:

and return to stdout a formated string as `[bmaj]asec [bmin]asec [bpa]deg` for `-beam-shape` option.


−8.5
−8.6
−8.7
−8.8
−8.9
−9.0
DEC (deg)
−158.9
−159.0
−159.1
−159.2
−159.3
−159.4
RA (deg)


−8.5
−8.6
−8.7
−8.8
−8.9
−9.0
DEC (deg)
−158.9
−159.0
−159.1
−159.2
−159.3
−159.4
RA (deg)


−8.5
−8.6
−8.7
−8.8
−8.9
−9.0
DEC (deg)
−159.4
−159.3
−159.2
−159.1
−159.0
−158.9
RA (deg)

CHAPTER

SEVEN

API REFERENCE

This page contains auto-generated API reference documentation[1].

7.1 lofarSun

7.1.1 Subpackages

lofarSun.BF

Subpackages

lofarSun.BF.GUI

Submodules

lofarSun.BF.GUI.layout_ui

Module Contents

Classes

Ui_MainWindow

class lofarSun.BF.GUI.layout_ui.**Ui_MainWindow**

Bases: `object`

setupUi(*MainWindow*)

retranslateUi(*MainWindow*)

[1] Created with sphinx-autoapi

lofarSun.BF.GUI.lofarBFgui

Module Contents

Classes

MatplotlibWidget

Functions

main()

Attributes

here

lofarSun.BF.GUI.lofarBFgui.**here**

class lofarSun.BF.GUI.lofarBFgui.**MatplotlibWidget**

Bases: QMainWindow

btnstate(*b*)

static move_window(*window*, *dx*, *dy*)

Move a matplotlib window to a given x and y offset, independent of backend

init_graph()

update_lofarBF()

update_lofar_cube()

update_lofar_fits()

draw_ds_after_load(*idx_cur=0*, *conn_click=True*)

spectroPlot_func()

onclick(*event*)

showPointing()

keyUp()

keyDown()

keyLeft()

keyRight()

stepNear(*f_move*, *t_move*)

showBeamForm()

lofarSun.BF.GUI.lofarBFgui.**main**()

lofarSun.BF.GUI.mplw

Module Contents

Classes

mplw

class lofarSun.BF.GUI.mplw.**mplw**(*parent=None*)

Bases: QWidget

lofarSun.BF.GUI.resource_rc

Module Contents

Functions

qInitResources()
qCleanupResources()

Attributes

qt_resource_data
qt_resource_name
qt_resource_struct_v1
qt_resource_struct_v2
qt_version
rcc_version

lofarSun.BF.GUI.resource_rc.**qt_resource_data = b'\x00\x00NC\x89PNG\r\n\x1a\n\x00\x00\x00\rIHDR\x00\x00\x00\xc8\x00\x00\x00\xc8\x08\x02\x00\x00\x...**

lofarSun.BF.GUI.resource_rc.**qt_resource_name = b'\x00\x03\x00\x00L\x99\x00G\x00U\x00I\x00\x08\x0c\xa6\xc7\x95\x00r\x00e\x00s\x00o\x00u\x00r\x00c...**

lofarSun.BF.GUI.resource_rc.**qt_resource_struct_v1 = b'\x00\x00\x00\x00\x00\x02\x00\x00\x00\x01\x00\x00\x00\x01\x00\x00\x00\x00\x00\x02\x00\x00\x00\x0...**

lofarSun.BF.GUI.resource_rc.**qt_resource_struct_v2 = b'\x00\x00\x00\x00\x00\x02\x00\x00\x00\x01\x00\x00\x00\x01\x00\x00\x00\x00\x00\x00\x00\x00\x0...**

lofarSun.BF.GUI.resource_rc.**qt_version**

lofarSun.BF.GUI.resource_rc.**rcc_version = 1**

lofarSun.BF.GUI.resource_rc.**qInitResources**()

lofarSun.BF.GUI.resource_rc.**qCleanupResources**()

Submodules

lofarSun.BF.BFdata

Module Contents

Classes

BFcube

class lofarSun.BF.BFdata.**BFcube**

load_sav_xy(*fname*, *header_name='ds'*)

load_sav_radec(*fname*, *header_name='cube_ds'*)

load_fits(*fname*)

bf_image_by_idx(*f_idx*, *t_idx*, *fov=3000*, *asecpix=20*, *extrap=True*, *interpm='cubic'*)

bf_image_by_freq_time(*freq*, *time*, *fov=3000*, *asecpix=20*, *extrap=True*, *interpm='cubic'*, *verbout=True*)

bf_time_to_idx(*time*)

bf_freq_to_idx(*freq*)

bf_peak_size(*X*, *Y*, *data_bf*, *asecpix*)

bf_fit_gauss_source_by_idx(*f_idx*, *t_idx*, *drawfig=True*, *verb=True*)

plot_bf_image_by_idx(*f_idx*, *t_idx*)

plot_bf_dyspec(*beam_idx=0*, *ax_cur=None*)

write_fits(*fdir*, *fprefix*, *f_idx*, *t_idx*)
write the data into fits files

write_fits_full(*fdir*, *fprefix*)

lofarSun.BF.RFIconvFlag

Module Contents

Classes

RFIconv	Base class for all neural network modules.

Functions

init_RFIconv(net[, aggressive_factor, device])

Attributes

aggressive_factor

class lofarSun.BF.RFIconvFlag.**RFIconv**(*device='cpu'*)

Bases: `torch.nn.Module`

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. You can assign the submodules as regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

Submodules assigned in this way will be registered, and will have their parameters converted too when you call `to()`, etc.

Note: As per the example above, an `__init__()` call to the parent class must be made before assignment on the child.

Variables

training (*bool*) – Boolean represents whether this module is in training or evaluation mode.

forward(*x*)

lofarSun.BF.RFIconvFlag.**aggressive_factor = [1.6, 1.65, 0.5, 0.5]**

lofarSun.BF.RFIconvFlag.**init_RFIconv**(*net*, *aggressive_factor=[1.6, 1.65, 0.5, 0.5]*, *device='cpu'*)

lofarSun.BF.bftools

Module Contents

Functions

h5_fetch_meta(f[, SAP])	get info from the h5 file
downsample_h5_seg_by_time_ratio(data_array_uri, t_all, ...)	Downsample the h5 file by time ratio
cook_fits_spectr_hdu(data_fits, t_fits, f_fits, ...)	
avg_1d(x, N)	very simple averaging for 1D array
averaging_stride(arr_query, n_point[, axis, ...])	Perform downsampling of a 2D array by averaging over strided subarrays.
averaging_walk(arr_query, n_point[, axis, start_idx, ...])	Perform downsampling of a 2D array by averaging over contiguous subarrays.
model_flux(calibrator, frequency)	Calculates the model flux for calibration using a known set of calibrators.
partition_avg(arr, ratio_range)	
get_cal_bandpass(freq_idx, h5dir, h5name[, ratio_range])	
avg_with_lightening_flag(array_dirty, idx_start, idx_end)	It's an averaging process but it can flag-out the time points with local discharges,
lin_interp(x, y, i, half)	
FWHM(x, y)	Determine the FWHM position [x] of a distribution [y]
DecayExpTime(x, y)	
fit_biGaussian(x, y)	Derive the best fit curve for the flux-time distribution
biGaussian(x, x0, sig1, sig2, A)	
mask_extend_xy_npix(mask, n_pix_x, n_pix_y)	Extend a 2D boolean mask along both x and y axes by a specified number of pixels.
flag_frequency_slices(dynspec_cal, mask_cal[, ...])	Flag individual pixels in each frequency slice of the calibration dynamic spectrum.
perform_linear_interpolation(dynspec_cal_copy, ...)	Perform linear interpolation for flagged pixels in each frequency slice of the dynamic spectrum.
proc_calib_dynspec(dynspec_sun, dynspec_cal, time_sun, ...)	Calibrate a dynamic spectrum of the Sun using a given calibration dynamic spectrum.
proc_selfcalib_dynspec(dynspec_sun, time_sun, freq_sun)	Perform self-calibration on a dynamic spectrum of the Sun.

Attributes

device

lofarSun.BF.bftools.**device**

lofarSun.BF.bftools.**h5_fetch_meta**(*f*, *SAP='000'*)

get info from the h5 file

Parameters

- **f** (*h5 file*) – target h5 file
- **SAP** (*str, optional*) – Sub-Array-Pointing. Defaults to "000".

Returns
metadata of the h5 file

Return type
list

lofarSun.BF.bftools.**downsample_h5_seg_by_time_ratio**(*data_array_uri*, *t_all*, *t_ratio_start*, *t_ratio_end*, *t_idx_count*, *t_c_ratio*, *f_c_ratio*, *averaging=True*, *flagging=False*, *t_idx_cut=256*, *agg_factor=[1.66, 1.66, 0.45, 0.45]*, *subband_edge=False*, *subband_ch=16*, *device=device*)

Downsample the h5 file by time ratio

Parameters

- **data_array_uri** (*array*) – dynamic spectrum
- **t_all** (*1d array*) – all time stamps
- **t_ratio_start** (*float*) – start time ratio (0-1)
- **t_ratio_end** (*float*) – end time ratio (0-1)
- **t_idx_count** (*int*) – number of total time stamps
- **t_c_ratio** (*int*) – time compression ratio
- **f_c_ratio** (*int*) – frequency compression ratio
- **averaging** (*bool, optional*) – averaging or direct sample. Defaults to True.
- **flagging** (*bool, optional*) – flagging or not. Defaults to False.
- **t_idx_cut** (*int, optional*) – factor to contour segment length for processing, to make use of the memory. Defaults to 256.
- **agg_factor** (*list, optional*) – factor for flagging. Defaults to [1.66, 1.66, 0.45, 0.45].
- **subband_edge** (*bool, optional*) – remove one channel every [subband num] channels. Defaults to False.
- **subband_ch** (*int, optional*) – number of subbands per channel. Defaults to 16.
- **device** (*device, optional*) – GPU or CPU, should be something like torch.device("cuda:0") . Defaults to device.

Returns
[averaged dynamic spectrum, time stamps]

Return type
list

lofarSun.BF.bftools.**cook_fits_spectr_hdu**(*data_fits*, *t_fits*, *f_fits*, *t_start_fits*, *t_end_fits*, *stokes_key*, *antenna_set_name*, *telescop_name*, *target_name*, *pointing_ra*, *pointing_dec*, *pointing_x*, *pointing_y*)

lofarSun.BF.bftools.**avg_1d**(*x*, *N*)

very simple averaging for 1D array

Parameters

- **x** (*array*) – input 1D array
- **N** (*int*) – down-sample ratio

Returns
averaged array

Return type
array

lofarSun.BF.bftools.**averaging_stride**(*arr_query*, *n_point*, *axis=0*, *start_idx=-1*, *end_idx=-1*)

Perform downsampling of a 2D array by averaging over strided subarrays.

This function is designed for scenarios where the 2D array is large but the number of points to average (n_point) is small. The resulting array is not significantly smaller than the original.

Parameters

- **arr_query** (*numpy.ndarray*) – The 2D array to be downsampled.
- **n_point** (*int*) – Number of points in each strided subarray over which the averaging is performed.
- **axis** (*int, optional*) – The axis along which to average, either 0 or 1. Default is 0.
- **start_idx** (*int, optional*) – The starting index for averaging. Default is the first index of the array.
- **end_idx** (*int, optional*) – The ending index for averaging. Default is the last index of the array.

Returns
The resulting downsampled array.

Return type
numpy.ndarray

Examples

```
>>> arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]])
>>> averaging_stride(arr, 2, axis=0)
Output will be the downsampled array along axis 0.
```

Notes

This function is optimized for small downsampling ratios where n_point is small.

lofarSun.BF.bftools.**averaging_walk**(*arr_query*, *n_point*, *axis=0*, *start_idx=-1*, *end_idx=-1*)

Perform downsampling of a 2D array by averaging over contiguous subarrays.

This function is designed for scenarios where the 2D array is large and the number of points to average (n_point) is also large. The result is a much smaller array.

Parameters

- **arr_query** (*numpy.ndarray*) – The 2D array to be downsampled.
- **n_point** (*int*) – Number of points in each subarray over which the averaging is performed.
- **axis** (*int, optional*) – The axis along which to average, either 0 or 1. Default is 0.

- **start_idx** (*int, optional*) – The starting index for averaging. Default is the first index of the array.
- **end_idx** (*int, optional*) – The ending index for averaging. Default is the last index of the array.

Returns
The resulting downsampled array.

Return type
numpy.ndarray

Examples

```
>>> arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]])
>>> averaging_walk(arr, 2, axis=0)
Output will be the downsampled array along axis 0.
```

Notes

This function is optimized for large downsampling ratios where n_point is large.

lofarSun.BF.bftools.**model_flux**(*calibrator*, *frequency*)

Calculates the model flux for calibration using a known set of calibrators.

Parameters

- **calibrator** (*str*) – Name of the calibrator source.
- **frequency** (*float*) – Frequency in MHz for which to calculate the flux.

Returns
Model flux in sfu (solar flux units).

Return type
float

Notes

The parameters for each calibrator source are sourced from https://arxiv.org/pdf/1609.05940.pdf.

lofarSun.BF.bftools.**partition_avg**(*arr*, *ratio_range*)

lofarSun.BF.bftools.**get_cal_bandpass**(*freq_idx*, *h5dir*, *h5name*, *ratio_range=[0.2, 0.8]*)

lofarSun.BF.bftools.**avg_with_lightening_flag**(*array_dirty*, *idx_start*, *idx_end*, *f_avg_range=[1600, 3500]*, *peak_ratio=1.08*, *stride=96*, *rm_bandpass=True*)

It's an averaging process but it can flag-out the time points with local discharges, (the very bright and vertical lines)

lofarSun.BF.bftools.**lin_interp**(*x*, *y*, *i*, *half*)

lofarSun.BF.bftools.**FWHM**(*x*, *y*)

Determine the FWHM position [x] of a distribution [y]

lofarSun.BF.bftools.**DecayExpTime**(*x*, *y*)

lofarSun.BF.bftools.**fit_biGaussian**(*x*, *y*)

Derive the best fit curve for the flux-time distribution

lofarSun.BF.bftools.**biGaussian**(*x*, *x0*, *sig1*, *sig2*, *A*)

lofarSun.BF.bftools.**mask_extend_xy_npix**(*mask*, *n_pix_x*, *n_pix_y*)

Extend a 2D boolean mask along both x and y axes by a specified number of pixels.

This function takes a 2D mask and extends the 'False' values in both x and y directions based on the number of pixels specified. The purpose is to enlarge masked areas in a 2D array by including adjacent pixels.

Parameters

- **mask** (*ndarray (2D)*) – A 2D boolean mask where 'True' represents areas to keep and 'False' represents areas to mask.
- **n_pix_x** (*int*) – Number of pixels to extend the mask in the x-direction.
- **n_pix_y** (*int*) – Number of pixels to extend the mask in the y-direction.

Returns
mask_extend – The extended 2D mask.

Return type
ndarray (2D)

lofarSun.BF.bftools.**flag_frequency_slices**(*dynspec_cal*, *mask_cal*, *ratio_flag=1.5*, *lower_perc=15*, *upper_perc=40*)

Flag individual pixels in each frequency slice of the calibration dynamic spectrum.

This function flags pixels based on a condition that compares the pixel value to the mean of a specific range of values in the dynamic spectrum.

Parameters

- **dynspec_cal** (*ndarray*) – The calibration dynamic spectrum.
- **mask_cal** (*ndarray*) – The existing mask for the calibration dynamic spectrum.

Returns
mask_cal – Updated mask for the calibration dynamic spectrum.

Return type
ndarray

lofarSun.BF.bftools.**perform_linear_interpolation**(*dynspec_cal_copy*, *mask_cal*, *t_cal*)

Perform linear interpolation for flagged pixels in each frequency slice of the dynamic spectrum.

Parameters

- **dynspec_cal_copy** (*ndarray*) – Copy of the calibration dynamic spectrum that will be modified.
- **mask_cal** (*ndarray*) – The existing mask for the calibration dynamic spectrum.
- **t_cal** (*ndarray*) – Time array corresponding to the calibration dynamic spectrum.

Returns
dynspec_cal_copy – The modified calibration dynamic spectrum after performing interpolation.

Return type
ndarray

Examples

```
>>> dynspec_cal_copy = perform_linear_interpolation(dynspec_cal_copy, mask_cal, t_
↪cal)
```

lofarSun.BF.bftools.**proc_calib_dynspec**(*dynspec_sun*, *dynspec_cal*, *time_sun*, *freq_sun*, *t_cal*, *f_cal*, *abs_thresh=100000000000000.0*)

Calibrate a dynamic spectrum of the Sun using a given calibration dynamic spectrum.

Parameters

- **dynspec_sun** (*ndarray*) – 2D array representing the dynamic spectrum of the Sun.
- **dynspec_cal** (*ndarray*) – 2D array representing the calibration dynamic spectrum.
- **time_sun** (*ndarray*) – 1D array of time values corresponding to dynspec_sun.
- **freq_sun** (*ndarray*) – 1D array of frequency values corresponding to dynspec_sun.
- **t_cal** (*ndarray*) – 1D array of time values corresponding to dynspec_cal.
- **f_cal** (*ndarray*) – 1D array of frequency values corresponding to dynspec_cal.
- **abs_thresh** (*float, optional*) – Absolute threshold for masking, default is 1e14.

Returns

- **calibrated_dynspec** (*ndarray*) – Calibrated dynamic spectrum.
- **dynspec_cal** (*ndarray*) – Input calibration dynamic spectrum.
- **dynspec_cal_copy** (*ndarray*) – Dynamic spectrum after interpolation.
- **mask_cal** (*ndarray*) – Mask after initial flagging.
- **mask_cal_2nd** (*ndarray*) – Mask after second-round flagging.

Examples

```
>>> (calibrated_dynspec, dynspec_cal, dynspec_cal_copy, mask_cal, mask_cal_2nd) =␣
↪proc_calib_dynspec(dynspec_sun, dynspec_cal, time_sun, freq_sun, t_cal, f_cal)
```

Notes

1. Initial masking based on absolute threshold.
2. Flagging pixels in each frequency slice.
3. Extending the mask spatially.
4. Linear interpolation for flagged pixels.
5. Second round of flagging based on snapshot median.
6. Interpolation after second-round flagging.
7. Averaging the dynamic spectrum.
8. Final calibration using the average dynamic spectrum.

lofarSun.BF.bftools.**proc_selfcalib_dynspec**(*dynspec_sun*, *time_sun*, *freq_sun*, *abs_thresh=200000000000000.0*)

Perform self-calibration on a dynamic spectrum of the Sun.

This function takes a dynamic spectrum of the Sun and performs various steps to calibrate it, including masking based on thresholds, flagging, interpolation, and averaging. The calibration is done in a 'self-calibration' mode, meaning it uses the Sun's own dynamic spectrum to create a bandpass for calibration.

Parameters

- **dynspec_sun** (*ndarray*) – Dynamic spectrum of the Sun.
- **time_sun** (*ndarray*) – Time array corresponding to dynspec_sun.
- **freq_sun** (*ndarray*) – Frequency array corresponding to dynspec_sun.

Returns

- **calibrated_dynspec** (*ndarray*) – Calibrated dynamic spectrum.
- **mask_cal** (*ndarray*) – Mask after initial flagging and extending.
- **dynspec_cal_bp** (*ndarray*) – Averaged dynamic spectrum used for bandpass calculation.

Examples

```
>>> calibrated_dynspec, mask_cal, dynspec_cal_bp = proc_selfcalib_dynspec(dynspec_
↪sun, time_sun, freq_sun)
# size of dynspec_sun(M*N), time_sun(M), freq_sun(N) should match
```

Notes

1. Handle NaN values in the Sun's dynamic spectrum.
2. Create a copy of the Sun's dynamic spectrum for bandpass calculation.
3. Average the copied dynamic spectrum.
4. Mask the averaged spectrum based on an absolute threshold.
5. Perform further flagging of frequency slices.
6. Extend the mask in time and frequency.
7. Interpolate the masked values.
8. Perform a second round of flagging based on snapshot median.
9. Interpolate again after the second round of flagging.
10. Average the dynamic spectrum for bandpass calculation.
11. Calibrate each frequency slice.

lofarSun.BF.lofarJ2000xySun

A script to convert between the J2000 and the sun.

Module Contents

Functions

j2000xy(RA, DEC, t_sun)	

lofarSun.BF.lofarJ2000xySun.**j2000xy**(*RA*, *DEC*, *t_sun*)

Package Contents

Classes

BFcube	

class lofarSun.BF.**BFcube**

load_sav_xy(*fname*, *header_name='ds'*)

load_sav_radec(*fname*, *header_name='cube_ds'*)

load_fits(*fname*)

bf_image_by_idx(*f_idx*, *t_idx*, *fov=3000*, *asecpix=20*, *extrap=True*, *interpm='cubic'*)

bf_image_by_freq_time(*freq*, *time*, *fov=3000*, *asecpix=20*, *extrap=True*, *interpm='cubic'*, *verbout=True*)

bf_time_to_idx(*time*)

bf_freq_to_idx(*freq*)

bf_peak_size(*X*, *Y*, *data_bf*, *asecpix*)

bf_fit_gauss_source_by_idx(*f_idx*, *t_idx*, *drawfig=True*, *verb=True*)

plot_bf_image_by_idx(*f_idx*, *t_idx*)

plot_bf_dyspec(*beam_idx=0*, *ax_cur=None*)

write_fits(*fdir*, *fprefix*, *f_idx*, *t_idx*)

write the data into fits files

write_fits_full(*fdir*, *fprefix*)

lofarSun.IM

Submodules

lofarSun.IM.IMdata

Module Contents

Classes

IMdata	

Functions

func_gaussian(xdata, s0, x_cent, y_cent, tile, x_sig, ...)	
get_tile_ellipse_from_fit(popt)	
get_peak_beam_from_psf(fname[, thresh, cbox])	Get the peak beam from a fits file

class lofarSun.IM.IMdata.**IMdata**

load_fits(*fname*)

get_cur_solar_centroid(*t_obs*)

get_obs_image_centroid(*header*)

get_axis_obs(*header*)

RA_DEC_shift_xy0(*RA*, *DEC*, *RA_cent*, *DEC_cent*)

sun_coord_trasform(*data*, *header*, *act_r=True*, *act_s=True*)

get_beam()

make_map(*fov=2500*)

plot_image(*log_scale=False*, *fov=2500*, *FWHM=False*, *gaussian_sigma=0*, *ax_plt=None*, ***kwargs*)

peak_xy_coord_pix()

peak_xy_coord_arcsec()

fit_gaussian2d(*thresh=0.5*, *cbox=[[0, 0], [0, 0]]*, ***kwargs*)

Fit a 2D Gaussian function to the image data, feature update: can specify a box to fit the gaussian function

Parameters

- **thresh** (*float, optional*) – threshold to limit the fit region. Defaults to 0.5.

- **cbox** (*list, optional*) – the range to specify the bounding box, [[x0,x1],[y0,y1]], default is no bounding. Defaults to [[0,0],[0,0]].

Returns
fitting results

Return type
list

lofarSun.IM.IMdata.**func_gaussian**(*xdata*, *s0*, *x_cent*, *y_cent*, *tile*, *x_sig*, *y_sig*)

lofarSun.IM.IMdata.**get_tile_ellipse_from_fit**(*popt*)

lofarSun.IM.IMdata.**get_peak_beam_from_psf**(*fname*, *thresh=0.618*, *cbox=[[0, 0], [0, 0]]*)
Get the peak beam from a fits file

lofarSun.IM.MSinspect

Package Contents

Classes

IMdata	

Functions

func_gaussian(xdata, s0, x_cent, y_cent, tile, x_sig, ...)	
get_tile_ellipse_from_fit(popt)	
get_peak_beam_from_psf(fname[, thresh, cbox])	Get the peak beam from a fits file

class lofarSun.IM.**IMdata**

load_fits(*fname*)

get_cur_solar_centroid(*t_obs*)

get_obs_image_centroid(*header*)

get_axis_obs(*header*)

RA_DEC_shift_xy0(*RA*, *DEC*, *RA_cent*, *DEC_cent*)

sun_coord_trasform(*data*, *header*, *act_r=True*, *act_s=True*)

get_beam()

make_map(*fov=2500*)

plot_image(*log_scale=False*, *fov=2500*, *FWHM=False*, *gaussian_sigma=0*, *ax_plt=None*, ***kwargs*)

peak_xy_coord_pix()

peak_xy_coord_arcsec()

fit_gaussian2d(*thresh=0.5*, *cbox=[[0, 0], [0, 0]]*, ***kwargs*)

Fit a 2D Gaussian function to the image data, feature update: can specify a box to fit the gaussian function

Parameters

- **thresh** (*float, optional*) – threshold to limit the fit region. Defaults to 0.5.
- **cbox** (*list, optional*) – the range to specify the bounding box, [[x0,x1],[y0,y1]], default is no bounding. Defaults to [[0,0],[0,0]].

Returns
fitting results

Return type
list

lofarSun.IM.**func_gaussian**(*xdata*, *s0*, *x_cent*, *y_cent*, *tile*, *x_sig*, *y_sig*)

lofarSun.IM.**get_tile_ellipse_from_fit**(*popt*)

lofarSun.IM.**get_peak_beam_from_psf**(*fname*, *thresh=0.618*, *cbox=[[0, 0], [0, 0]]*)

Get the peak beam from a fits file

lofarSun.cli

Submodules

lofarSun.cli.h5_to_fits_spec

Module Contents

Functions

parse_args()	
compress_h5(fname_DS, out_dir, t_c_ratio, t_idx_cut, ...)	fname_DS: relative (or absolute) directory+fname to the .h5
main()	

Attributes

DESCRIPTION
device
lofar_logo_base64
idols_logo_base64

lofarSun.cli.h5_to_fits_spec.**DESCRIPTION = Multiline-String**

```
"""
Split and down sample the dynamic spectrum of LOFAR observation

Input  :  Huge hdf5 file of LOFAR Tied array beam formed observation
Output :  Small fits file with json and png quickview

(by Peijin.Zhang & Pietro Zucca 2019.08)
"""
```

lofarSun.cli.h5_to_fits_spec.**device**

lofarSun.cli.h5_to_fits_spec.**lofar_logo_base64 =**
'iVBORw0KGgoAAAANSUhEUgAAAEAAAABACAIAAAAlC+aJAAAURklEQVR4nLx6CXRcZfn3733vMlsmy2RfmrXpkhRo0i0tXUEE2...'

lofarSun.cli.h5_to_fits_spec.**idols_logo_base64 =**
'iVBORw0KGgoAAAANSUhEUgAAAIAAAAA1CAYAAACJOeMNAAAWsHpUWHRSYXcgcHJvZmlsZSB0eXBlIGV4aWYAAHjarZppkiO3t...'

lofarSun.cli.h5_to_fits_spec.**parse_args**()

lofarSun.cli.h5_to_fits_spec.**compress_h5**(*fname_DS*, *out_dir*, *t_c_ratio*, *t_idx_cut*, *f_c_ratio*, *minutes_per_chunk*, *num_chunks*, *chop_off*, *averaging*, *flagging*, *target_directory*, *date_directory*, *simple_fname*, *add_ksp_logo*, *add_idols_logo*, *flip_freq*)

fname_DS: relative (or absolute) directory+fname to the .h5 out_dir: relative (or absolute) directory+fname to the .h5

lofarSun.cli.h5_to_fits_spec.**main**()

lofarSun.cli.pyms_datetime_to_index

by Peijin Zhang version 0.1 2023-1-8 00:24:52: Initial version

lofarSun.cli.pyms_overview

lofarSun.cli.pyms_utils

command line tools to process the lofar solar data

by Peijin Zhang version 0.1 2022-5-23 00:24:52: Initial version

Module Contents

Classes

bcolors	

Functions

info_print(header, content)	
human_format(number)	
get_obs_info_from_ms(fname)	get observation information from ms
get_t_from_ms(fname)	get time information from ms
get_freq_from_ms(fname)	
ms_datetime_to_index(fname, t[, t_format])	convert datetime to index
ms_index_to_datetime(fname, idx)	
cook_wsclean_cmd(fname[, mode, multiscale, weight, ...])	
pyms_overview_main()	
pyms_datetime_to_index_main()	
pyms_cook_wsclean_cmd_main()	
pyms_index_to_datetime_main()	
pyms_psf_fit_peak_gauss_main()	

class lofarSun.cli.pyms_utils.**bcolors**

HEADER = '\x1b[95m'

OKBLUE = '\x1b[94m'

OKCYAN = '\x1b[96m'

OKGREEN = '\x1b[92m'

WARNING = '\x1b[93m'

FAIL = '\x1b[91m'

ENDC = '\x1b[0m'

BOLD = '\x1b[1m'

UNDERLINE = '\x1b[4m'

lofarSun.cli.pyms_utils.**info_print**(*header*, *content*)

lofarSun.cli.pyms_utils.**human_format**(*number*)

lofarSun.cli.pyms_utils.**get_obs_info_from_ms**(*fname*)

get observation information from ms

Parameters
fname (*string*) – measurement set name

Returns
list of antenna name int : number of baselines string : telescope name

Return type
list

lofarSun.cli.pyms_utils.**get_t_from_ms**(*fname*)

get time information from ms

Parameters
fname (*string*) – measurement set name

Returns
total number of time index list : time range

Return type
int

lofarSun.cli.pyms_utils.**get_freq_from_ms**(*fname*)

lofarSun.cli.pyms_utils.**ms_datetime_to_index**(*fname*, *t*, *t_format='%H:%M:%S.%f'*)

convert datetime to index

Parameters

- **fname** (*string*) – measurement set name
- **t** (*datetime*) – datetime

Returns
index

Return type
int

lofarSun.cli.pyms_utils.**ms_index_to_datetime**(*fname*, *idx*)

lofarSun.cli.pyms_utils.**cook_wsclean_cmd**(*fname*, *mode='default'*, *multiscale=True*, *weight='briggs 0'*, *mgain=0.8*, *thresholding='-auto-mask 3 -auto-threshold 0.3'*, *len_baseline_eff=35000*, *FOV=10000*, *scale_factor=4.3*, *circbeam=True*, *niter=1200*, *pol='I'*, *data_col='CORRECTED_DATA'*, *misc=''*, *name=''*, *interval=[-1, -1]*, *intervals_out=-1*)

lofarSun.cli.pyms_utils.**pyms_overview_main**()

lofarSun.cli.pyms_utils.**pyms_datetime_to_index_main**()

lofarSun.cli.pyms_utils.**pyms_cook_wsclean_cmd_main**()

lofarSun.cli.pyms_utils.**pyms_index_to_datetime_main**()

lofarSun.cli.pyms_utils.**pyms_psf_fit_peak_gauss_main**()

lofarSun.cli.spec_utils

CHAPTER

EIGHT

CITE AS

https://arxiv.org/abs/2205.00065

PYTHON MODULE INDEX

l

INDEX

H

I

J

K

L

M

O

P

Q

R

S

U

W