P-AIRCARS

Release 1.0.0

Devojyoti Kansabanik

Jul 13, 2022

P-AIRCARS

This is the official documentation of the P-AIRCARS. Source code can be found at : https://github.com/devojyoti96/P-AIRCARS.git

CHAPTER

ONE

INTRODUCTION

1.1 P-AIRCARS (Polarimetry using Automated Imaging Routine for Compact Arrays of the Radio Sun)

P-AIRCARS is an automated calibration and imaging routine for polarimetric calibration and imaging of solar observations done with Murchison Widefield Array (MWA) https://www.mwatelescope.org. P-AIRCARS has been developed and maintained by solar physics group at National Centre for Radio Astrophysics, Tata Institute of Fundamental Research (NCRA-TIFR), Pune, India https://www.ncra.tifr.res.in.

P-AIRCARS uses CASA (Common Astronomy Software Application) https://casa.nrao.edu for intensity and bandpass self-calibration and polarisation calibration is performed by our own implementation of the full Jones calibration algorithm described by Mitchell et al. 2008 https://doi.org/10.1109/JSTSP.2008.2005327.

Imaging at P-AIRCARS is performed by WSClean https://wsclean.readthedocs.io. If WSClean is not installed P-AIRCARS performs imaging by CASA. When CASA is used for imaging the computation speed is slow.

Basic philosophy of P-AIRCARS is self-calibration and use the instrumental model to perform the precise solar calibration. Details of the algorithm and implementation can be found in the follwoing papers

1.AIRCARS (Mondal et al. 2020) https://doi.org/10.3847/1538-4357/ab0a01

2.Kansabanik et al. 2021a, in preparation

3.Kansabanik et al. 2021b, in preparation

P-AIRCARS has several modules and scripts. Modules can be useful for any astronomical self-calibration. Users want to use the modules please the the documentation of the Module details. Instructions for local machines, laptops and work stations can be found in the installation section. Implementation for high performance computing (HPC) environment will be available shortly. A basic tutorial is also included.

For any queires and issues reach us at:

dkansabanik@ncra.tifr.res.in, paircarsnotification@gmail.com

1.2 System requirements

1.2.1 Minimum hardware requirements

CPUs: 4 cores

RAM : 4 GB

Installation space : 700 MB

Disk space : 6 times of the data volume

1.2.2 Workstation configuration

P-AIRCARS has been tested on two types of workstations with following configurations.

1. Intel Xeon Gold 6138 2 GHz 40 CPUs - 256 GB RAM

   4 x 10TB (RAID-0) 40TB , 2 sets of 3 x 10 TB (RAID-5) , 2 x 20 TB

2. Boston Intel E5-2699 v3 2.30GHz, 72 CPUs

   256 GB (8 x 32) RAM - 2 x 300 GB SSD System

   Data 41 TB (12 x 4 TB SATA with RAID-5, 256KB stripaize)

   Data1 81 TB (12 x 8 TB SATA with RAID-5, 512KB stripaize)

   960 GFLOPS

1.2.3 Software environment

P-AIRACRS has been tested in the follwoing linux environments

1. CentOS 7
2. Ubuntu 20.0.4

Details of other software requirements is available in installation section.

CHAPTER

TWO

INSTALLATION

P-AIRCARS is a self-contained python package. It is build on python3. Python version greater than 3.6 is required for P-AIRACRS. We recommend to use python virtual environment to install P-AIRCARS, but it is not a necessary requirement. Installation steps are as follows.

2.1 Creating virtual environment

Install python virtual environment:

>> python3 -m pip install –upgrade pip

>> python3 -m pip install virtualenv

Activate virtual environment:

>> python3 -m venv /path/to/virtualenv/paircars

>> source /path/to/virtualenv/paircars

Check the virtual environment path:

>> which python3

It should show "/path/to/virtualenv/paircars/bin/python3"

2.2 Obtaining P-AIRCARS

P-AIRCARS source code can be downloaded from https://github.com/devojyoti96/P-AIRCARS.git.

P-AIRCARS is not public now. It will be public very soon. If you want to use P-AIRCARS before it becomes public, please reach us at dkansabanik@ncra.tifr.res.in, paircarsnotification@gmail.com

2.3 Installing P-AIRCARS

Move to the P-AIRCARS directory.

Run *setup.py*

>> python3 setup.py install

That's all. It will automatically install all required packages and libraries. This installation process is in very early stage. If you find any issues during installation, please reach us at dkansabanik@ncra.tifr.res.in, paircarsnotification@gmail.com.

CHAPTER

THREE

TUTORIAL

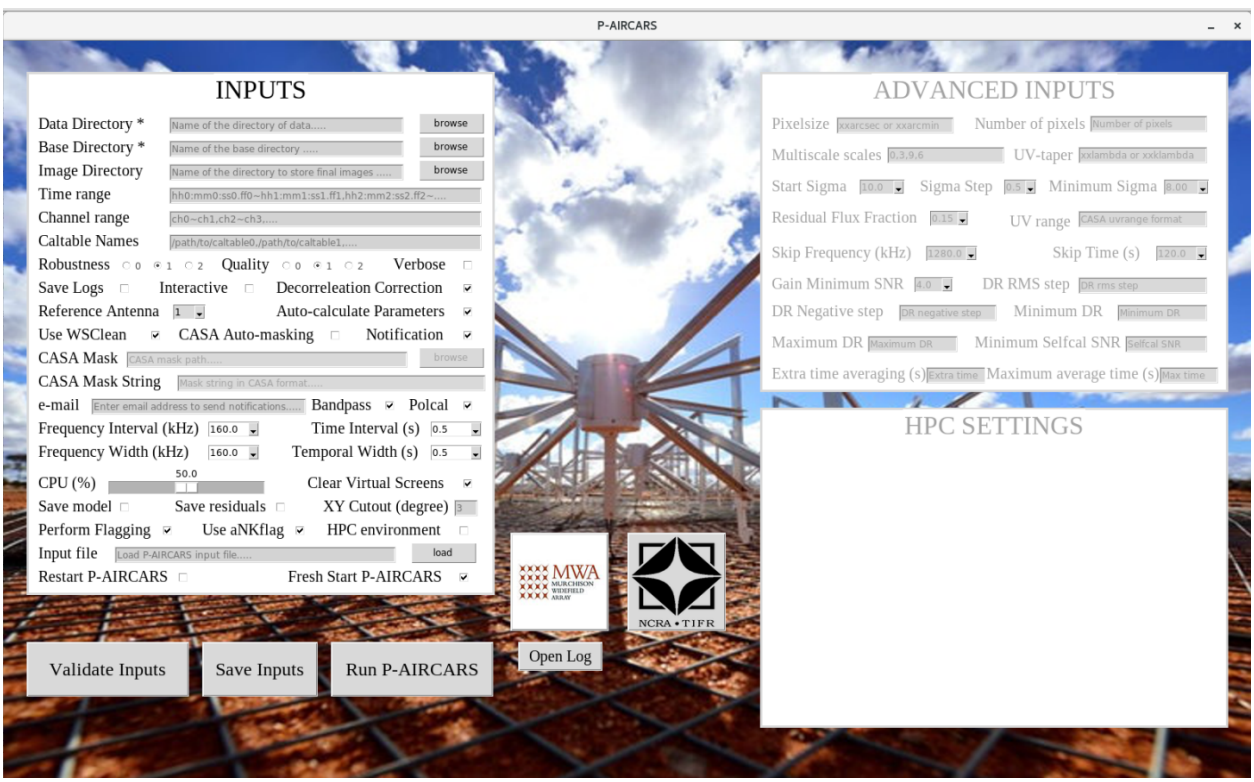
P-AIRCARS is a user-friendly and robust pipeline. In this tutorial we will learn how to run P-AIRCARS on a set of solar observations from MWA in default mode.

3.1 Start the P-AIRCARS

1. Open your terminal
2. Type 'go_paircars' and press enter

   >> `go_paircars`

3. P-AIRCARS input window will pop up.



3.2 Download the data

3.3 Fill up the inputs

3.3.1 Manual input

Two inputs are mandatory

1. *Data directory* : Name of the directory where all measurement setsto be imaged are present. User can type the full directory path in the entry box or use the browse button to select the directory.
2. *Base directory* : Name of the base directory where all the calibration and imaging will be done. User can type the full directory path in the entry box or use the browse button to select the directory.

3.3.2 Load from previous input file

If a previous P-AIRCARS input file with '.paircars' exists, inputs can be filled automatically from that input file. To do that follow the steps

1. Press the *load* button at the right of the **Input file** at the bottom of the INPUTS block and select the input file.
2. Select the file and click *open* and it will automatically load the input file and fill the inputs.

3.4 Validating inputs

After filling the inputs either manually or from a previous input file, user can validate if all inputs are correct or not.

1. Click **Validate Inputs** button at the bottom of the INPUTS.
2. If inputs are correct a pop up message will be shown saying, "Inputs are correct".

3.If mandatory inputs are missing it will show "Mandatory inputs are missing".
Check if all mandatory inputs are given or not.

3.5 Save inputs

If user wants P-AIRCARS inputs can be saved in a '.paircars' file.

1. Click **Save Inputs** button at the bottom.
2. A file dialog box will pop up. Default directory is your present derectory and default file name is 'inputs.paircars'. User may change the directory and file name.
3. Click on *save* to the inputs.
4. If the input file saved successfully a pop up message will show the location where the input file saved. Otherwise it will show error.

3.6 Run the P-AIRCARS

After giving all the mandatory inputs, we are now ready to start the P-AIRCARS.

1. Click on **Run P-AIRCARS** button at the bottom.
2. If P-AIRCARS is already running or interupted at the same base directory, a pop up message will ask 'P-AIRCARS already running in the base directory. Do you really want to over run?'
3. If you want to re-run P-AIRCARS in the same base directory overriding the present running P-AIRCARS, press *yes* , otherwise press *no*.
4. If you press *yes*, P-AIRCARS will start. It may take some time to start. When started a pop up message will show 'P-AIRCARS has started. Job ID : your_job_id'.
5. Note down the Job ID. You can use this Job ID to track the progress of the P-AIRCARS.
6. If you have started P-AIRCARS, it will start after pressing **Run P-AIRCARS** button and pop message will show the Job ID.

3.7 Notification over e-mail

User can get the updates about the progress of p-AIRCARS in their e-mail. To enable this feature follow the steps

1. Select the *Notification* button.
2. Put the e-mail id of the user in the *e-mail* entry box.

3.8 Locate the final images

If you give some custom path to save final images in the *Image Directory*, final images will be saved there. Otherwise, final images will be saved in a sub-directory named 'final_images' inside the *Base Directory*.

CHAPTER

FOUR

DETAILS OF INPUTS

In this section the details of the P-AIRCARS inputs are given.

4.1 Inputs

1. **Data Directory** : Name of the directory where all measurement sets to be imaged are present. If user want to run in *offline* mode, please be sure that all metafits files correcponding to all measurement sets are also present in the *Data Directory*. This is a mandatory input.
2. **Base Directory** : Name of the directory where all calibration and imaging analysis will be done. Either give an existing directory, otherwise P-AIRCARS will create the *Base Directory* in the given path. This is a mandatory input.
3. **Image Directory** : Name of the directory to save final images. If no input is given final images will be saved in a sub-directory named 'final_images' inside the *Base Directory*.
4. **Time range** : Time ranges to be imaged. Time range format is hh0:mm0:ss0.ff0~hh1:mm1:ss1.ff1,hh2:mm2:ss2.ff2~hh3:mm3:ss3.ff3,...
5. **Channel range** : Channel ranges to be imaged. Channel range format is ch0~ch1,ch2~ch3,....
6. **Caltable Names** : Prior caltable names obtained from calibrator observations. Give the full paths separated by commas.
7. **Robustness** : Determine the robustness of the calibration solutions and sef-calibration convergence. It takes three values 0, 1, 2 with increasing robustness. Default value is 1.
8. **Quality** : Determine the image quality. It takes three values 0, 1, 2 with increasing quality of the images. Default value is 1.
9. **Verbose** : If selected, verbose output logs will be saved. Also all the images, measurement sets and calibration tables are also saved. In verbose mode amount of disk space requirements will be more. Turn on only for debuggung purposes. Default it is turned off.
10. **Save Logs** : If selected, logs for self-calibration and imaging rounds will be saved. Default it is turned off.
11. **Interactive** : By default it is turned off. If selected, user can do interactive cleaning and also can change inputs manually during runtime. Only useful for specific debugging.
12. **Decorrelation Correction** : If selected, perform the amplitude decorrelatation correction. By default it is switched on. Details of the correction can be found in Mondal et al. 2020 https://doi.org/10.3847/1538-4357/ab0a01
13. **Reference Antenna** : Select the reference antenna from first 30 core antennas. Default is 1.

14. **Auto-calculate Parameters** : If selected, P-AIRCARS will calculate calibration and imaging parameters automatically from the data. By default, it is switched on. When it is turned off, user can give their own calibration and imaging parameters in advanced inputs. Only experienced users should turn this off.
15. **Use WSClean** : Use WSClean for imaging. By default it is switched on. If it is turned off, CASA will be used for imaging. Imaging using CASA is significantly slow compared to WSClean. Thus users are instructed to use WSClean. If WSClean is not installed, P-AIRACARS will automatically use CASA for imaging even if Use WSClean is turned on.
16. **CASA Auto-masking** : When CASA is being used for imaging, CASA auto-masking can be used by enabling this option. By default it is not activated.
17. **Notification** : If selected, P-AIRCARS will automatically send the updates about the progress of calibration and imaging to the user specific e-mail id.
18. **CASA Mask** : When using CASA for imaging, user may supply a CASA mask file here. By default it is not activated. It will be activated when Use WSClean is turned off.
19. **CASA Mask String** : CASA soecific mask string when using CASA for imaging. By default it is not activated. It will be activated when Use WSClean is turned off.
20. **e-mail** : User e-mail address to send notifications. If no e-mail id is given, no notification can be sent.
21. **Bandpass** : Whether perform bandpass self-calibration or not. By default it is turned on.
22. **Polcal** : Whether perform polarisation calibration and imaging. By default it is turned on. When it off, P-AIRCARS will only make Stokes-I images.
23. **Frequency Interval** : Frequency interval to make images. By default it is 160 kHz.
24. **Time Interval** : Time interval to make images. By default it is 0.5 s.
25. **Frequency Width** : Bandwidth to average. By default it is 160 kHz.
26. **Time Width** : Temporal averaging. By default it is 0.5 s.
27. **CPU(%)** : Percentage of total available cpus to be used. By default it is 50%.
28. **Clear Virtual SCreens** : Clear the virtual screens opened during the P-AIRCARS run.
29. **Save models** : If selected, save final model images. By default it is turned off.
30. **Save residuals** : If selected, save final residual images. By default it is turned off.
31. **XY cutout** : Size of the final image cutout. Default size is 3 degree.
32. **Perform Flagging** : Perform flagging during the calibration or not. By default it is on.
33. **Use aNKflag** : If selected, use aNKflag for flagging. If aNKflag is not installed, do not use.
34. **HPC environment** : Switch on if running is high performance computing (HPC) environment. When it is on fill the settings for HPC environment.
35. **Input file** : Load the P-AIRCARS inputs from previous saved P-AIRCARS input file. Extension of P-AIRCARS input file is '.paircars'. After loading inputs will be filled automatically.
36. **Restart P-AIRCARS** : Restart P-AIRCARS from the interupted step. It is off by default.
37. **Fresh start P-AIRCARS** : Start P-AIRCARS in fresh. If selected, all previous analysis in the same *Base Directory* will be deleted. By default it is switched on.

4.2 Advanced Inputs

Advanced input block will be activated when *Auto-calculate Parameters* is switched off. Use *Advanced Inputs* carefully and suggest to use only by experienced users.

1. **Pixelsize** : Pixel size of the image in arcsec or arcmin, e.g. '20.0arcsec' or '1.0arcmin'.
2. **Number of pixels** : Total number of pixels in the image. Product of *Pixelsize* and *Number of pixels* determine the size of the field-of-view.
3. **Multiscale scales** : Multiscale scales in units of number of pixels.
4. **UV-taper** : UV taper string in CASA format
5. **Start Sigma** : Starting value of sigma for rms based thresholding in self-calibration.
6. **Sigma Step** : Stepsize to reduce the sigma value with self-calibration iterations.
7. **Minimum Sigma** : Minimum value of sigma to reduce during self-calibration.
8. **Residual Flux Fraction** : Fraction of flux remained in the residual image to stop the self-calibration.
9. **UV range** : UV range in CASA format for self-calibration.
10. **Skip Frequency** : Frequency step to skip during calibration. Calibration will be performed after *Skip Frequency* steps.
11. **Skip Time** : Time step to skip during calibration. Calibration will be performed after *Skip Time* steps.
12. **Gain Minimum SNR** : Minimum SNR of the gain solutions to be used. Gain solutions less than minimum SNR will be rejected.
13. **DR RMS step** : If rms based dynamic range increases less than this value, self-calibration converged.
14. **DR Negative step** : If negative based dynamic range increases less than this value, self-calibration converged.
15. **Minimum DR** : Minimum dynamic range to save a final image.
16. **Maximum DR** : Maximum dynamic range required. If the imaging dynamic range reached this value, self-calibration stopped.
17. **Minimum Selfcal SNR** : Minimum antenna based SNR required to allow the self-calibration to start.
18. **Extra time averaging** : If large calibration solutions flagged, average over *Extra time* to gain SNR. The value is in second.
19. **Maximum average time** : Maximum allowed value of the *Extra time*.

4.3 High Performance Computing (HPC) settings

CHAPTER

FIVE

ROBUSTNESS AND QUALITY

In this section two major input parameters for P-AIRCARS, *Quality* and *Robustness*, will be discussed.

5.1 Quality (Q)

Quality parameters determines the quality of the final images. With the increase of *Quality* value, antennas will be added in smaller steps, self-calibration will be continued for smaller value of residual fluxes and final CLEANing of the images will be deeper. *Quality* takes three values; 0, 1, 2. Default value is 1.

5.2 Robustness (R)

Robustness parameter determines the robustness of the calibration and imaging. Increasing the robustness parameter will use strong convergence criteria for self-calibration, uses higher value of minimum SNR for gain solutions, performs calibration at smaller time and frequency intervals. *Robustness* takes three values; 0, 1, 2. Default value is 1.

The calibration and imaging parameters are determined based on the combination of *Quality* and *Robustness*. The calibration and imaging parameters for each of the combinations with increasing significance are listed below. With the increasing significance, the calibration and imaging quality will be better, but the computational time will also increase. Default value of 1 for *Quality* and *Robustness* is an optimization between final image quality and computational time.

5.3 Combinations

Q = 0, R = 0

residual_frac = 0.03

start_sigma = 9.0

min_sigma = 6

gain_minsnr = 3.0

DR_delta_rms = 25

DR_delta_neg = 20

min_DR = 20

max_DR = 100

skip_time = 480 s

skip_freq = 2560 kHz

Q = 0, R = 1

residual_frac = 0.03

start_sigma = 9.0

min_sigma = 7

gain_minsnr = 3.0

DR_delta_rms = 22

DR_delta_neg = 18

min_DR = 25

max_DR = 500

skip_time = 240 s

skip_freq = 2560 kHz

Q = 0, R = 2

residual_frac = 0.03

start_sigma = 9.0

min_sigma = 8

gain_minsnr = 3.0

DR_delta_rms = 20

DR_delta_neg = 15

min_DR = 30

max_DR = 1000

skip_time = 120 s

skip_freq = 2560 kHz

Q = 1, R = 0

residual_frac = 0.015

start_sigma = 10.0

min_sigma = 7

gain_minsnr = 4

DR_delta_rms = 20

DR_delta_neg = 15

min_DR = 30

max_DR = 1000

skip_time = 120 s

skip_freq = 1280 kHz

Q = 1, R = 1 (Default)

residual_frac = 0.015

start_sigma = 10.0

min_sigma = 8

gain_minsnr = 4

DR_delta_rms = 18

DR_delta_neg = 12

min_DR = 35

max_DR = 5000

skip_time = 60 s

skip_freq = 1280 kHz

Q = 1, R = 2

residual_frac = 0.015

start_sigma = 10.0

min_sigma = 9

gain_minsnr = 4

DR_delta_rms = 15

DR_delta_neg = 10

min_DR = 40

max_DR = 10000

skip_time = 30 s

skip_freq = 1280 kHz

Q = 2, R = 0

residual_frac = 0.01

start_sigma = 11.0

min_sigma = 8

gain_minsnr = 4.5

DR_delta_rms = 18

DR_delta_neg = 12

min_DR = 40

max_DR = 10000

skip_time = 15 s

skip_freq = 640 kHz

Q = 2, R = 1

residual_frac = 0.01

start_sigma = 11.0

min_sigma = 9

gain_minsnr = 4.5

DR_delta_rms = 15

DR_delta_neg = 10

min_DR = 45

max_DR = 50000

skip_time = 12 s

skip_freq = 640 kHz

Q = 2, R = 2

residual_frac = 0.01

start_sigma = 11.0

min_sigma = 10

gain_minsnr = 4.5

DR_delta_rms = 12

DR_delta_neg = 8

min_DR = 50

max_DR = 100000

skip_time = 10 s

skip_freq = 640 kHz

CHAPTER

SIX

PAIRCARS

6.1 paircars

6.1.1 paircars.access_ms

class paircars.access_ms.**AccessMS**(*msname*)

Bases: object

Generic class to perform measurement set related operation

Parameters
msname (*str*) – Name of the measurement set

calc_bandwidth()

Function to calculate bandwidth of the data

Returns
Bandwidth in Hz

Return type
float

calc_flagfrac()

Function to calculate flag fraction

Returns
Fraction of the data flagged

Return type
float

calc_freqres()

Return frequency resolution of the data

Returns
Frequency resolution in kHz

Return type
float

calc_meanfreq()

Function to return central frequency of the measurement set (Only MS with single SPW)

Returns
Central frequency in Hz

Return type
float

`calc_meanwavelength()`

Function to return central wavelength of the measurement set in meter

Returns
Central wavelength in metre

Return type
float

`calc_ncoarse()`

Calculate the number of MWA coarse channels in the measurement set

Returns
Number of coarse channel in the measurement set

Return type
int

`calc_timeres()`

Function to calculate time resolution of the measurement set

Returns
Time resolution in second

Return type
float

`calc_total_time()`

Function to calculate total time of the measurement set

Returns
Total time in seceond

Return type
float

`convert_mwa_to_iau()`

Convert the data from MWA coordinate (X=E->W and Y=N->S) to IAU coordinate (X=S->N,Y=W->E) system. Thus X=>-Y, and Y=>-X. So, XX=>YY,YY=>XX,XY=>YX,YX=>XY

Returns
Confirmation message of coordinate conversion the measurement set

Return type
str

`get_altaz(`*source_field=0*, *source_scan=1*`)`

Calculate alt az of the phasecenter

Parameters

- **source_field** (*int*) – FIELD ID of the source
- **source_scan** (*int*) – Scan number

Returns

- *float* – Altitude in radian
- *float* – Azimuth in radian

get_antenna_id(*antenna_name=''*)

Function to get antenna id from antenna name

Parameters
antenna_name (*str*) – Name of the antenna

Returns
Antenna id

Return type
int

get_antenna_string()

Function to get total antenna string

Returns
Antenna string

Return type
str

get_flag_times_mjd(*flagfrac=1*)

Function to get the flagged timestamps in MJD seconds if flag fraction is less than certain value

Parameters
flagfrac (*float*) – Flag fraction per channel (default : 1.0)

Returns
List of flagged times in MJD seconds

Return type
list

get_flag_timestamps(*flagfrac=1*)

Function to get the flagged timestamps in MJD seconds if flag fraction is less than certain value

Parameters
flagfrac (*float*) – Flag fraction per channel (default : 1.0)

Returns
List of flagged timestamps

Return type
list

get_freqs()

Function to return list of channel frequencies

Returns
List of frequencies in Hz

Return type
list

get_listobs(*listobsfile=''*)

Function to save listobs

Parameters
listobsfile (*str*) – File name to save listobs output

Returns
listobs file name

Return type
str

`get_max_baseline`(*includeflag=False*)

Get the maximum baseline in meter :param includeflag: Include flag data or not whilw calculating maximum baseline :type includeflag: bool

Returns
Maximum baseline length in meter

Return type
float

`get_model_nomodel_chan`()

Function to get the channels wioth no model data

Returns
List of model and nomodel channels

Return type
list

`get_nbaseline`(*autocor=True*)

Function to get number of baselines

Parameters
autocor (*bool*) – Include auto-correlations into account or not

Returns
Number of baselines

Return type
int

`get_num_antenna`()

Function to get total number of antennas in the measurement set

Returns
Number of antennas

Return type
int

`get_num_channels`()

Function to calculate number of channels

Returns
Number of channels

Return type
int

`get_num_timestamps`()

Function to calculate number of timestamps

Returns
Number of timestamps

Return type
int

get_obs_date(*format=0*)

Function to get observation date

Parameters

format (*str*) –

Date string format

0: 'YYYY/MM/DD'

1: 'YYYY-MM-DD'

2: 'YYYY_MM_DD'

Returns

- *str* – Observation start date
- *str* – Observation end date

get_obs_startend_time()

Function to get observation start time

Returns

- *str* – Observation start time in 'YYYY/MM/DD/hh:mm:ss' format
- *str* – Observation end time in 'YYYY/MM/DD/hh:mm:ss' format
- *float* – Observation start time in MJD second
- *float* – Observation end time in MJD second

get_observatory_loc()

Give the observatory geodetic location

Returns

- *float* – Latitude in degree
- *float* – Longitude in degree
- *float* – Altitude in meter

get_parang(*ra*, *dec*, *source_field=0*, *combine='field'*)

Calculate parallactic for phasecenter at a given Earth location for a given RA DEC Note = Parallactic angle is defined as the orientation of the sky in telescope coordinate. All angles are defined positive in IAU defined sky coordiunate (North to East). So, the rotation of the sky with respect to telescope is negative in the sky coordinate. To account this effect parallactic angle is given in 360-parang form.

Parameters

- **ra** (*float*) – RA in degree
- **dec** (*float*) – DEC in degree
- **source_field** (*int*) – FIELD id of the source
- **combine** (*str*) – Combine 'field' or 'scan' for calculating parallactic angle or '' for no combine

Returns

- *float* –

  1. combine = 'field', A single parallactic angle for the entire field in degree

- *dict* –

2. combine = 'scan', A python dictionary {'scan':parang} format

- *list* –

3. combine = '', A list of parang for all timestamps

get_phasecenter()

Get phasecenter of the measurement set

Returns

- *str* – Phasecenter of the measurement set in ['RA','DEC'] in hh mm ss dd mm ss format
- *float* – RA in degree
- *float* – DEC in degree

get_phasecenter_parang(*source_field=0*, *combine='field'*)

Calculate parallactic for phasecenter at a given Earth location. Note = Parallactic angle is defined as the orientation of the sky in telescope coordinate. All angles are defined positive in IAU defined sky coordiunate (North to East). So, the rotation of the sky with respect to telescope is negative in the sky coordinate. To account this effect parallactic angle is given in 360-parang form.

Parameters

- **source_field** (*int*) – FIELD id of the source
- **combine** (*str*) – Combine 'field' or 'scan' for calculating parallactic angle or '' for no combine

Returns

- *float* –

1. combine = 'field', A single parallactic angle for the entire field in degree

- *dict* –

2. combine = 'scan', A python dictionary {'scan':parang} format

- *list* –

3. combine = '', A list of parang for all timestamps

get_pol()

Function to get number of polarisations :returns: Number of correlation products :rtype: int

get_scan_startend_time()

Function to get scan start time

Returns

- *str* – Scan start time in 'YYYY/MM/DD/hh:mm:ss' format
- *str* – Scan end time in 'YYYY/MM/DD/hh:mm:ss' format
- *float* – Scan start time in MJD second
- *float* – Scan end time in MJD second

get_timestamps(*format=0*)

Function to return list of timestamps

Parameters

format (*int*) –

Datetime string format
(0 : 'YYYY/MM/DD/hh:mm:ss', 1: 'YYYY-MM-DDThh:mm:ss', 2: 'YYYY-MM-DD hh:mm:ss', 3: 'YYYY_MM_DD_hh_mm_ss')

Returns
List of timestamps in UTC at the format 'YYYY/MM/DD/hh:mm:ss'

Return type
list

`get_timestamps_in_mjdsecs()`

Function to return list of timestamps

Returns

- *list* – List of timestamps in MJD seconds
- *list* – List of timestamps with errors

`get_unflag_chan(flagfrac=1)`

Function to get the unflagged channels if flag fraction is less than certain value

Parameters
flagfrac (*float*) – Flag fraction per channel (default : 1.0)

Returns
List of unflagged channels

Return type
list

`get_unflag_times_mjd(flagfrac=1)`

Function to get the unflagged timestamps in MJD seconds if flag fraction is less than certain value

Parameters
flagfrac (*float*) – Flag fraction per channel (default : 1.0)

Returns
List of unflagged times in MJD second

Return type
list

`get_unflag_timestamps(flagfrac=1)`

Function to get the unflagged timestamps in MJD seconds if flag fraction is less than certain value

Parameters
flagfrac (*float*) – Flag fraction per channel (default : 1.0)

Returns
List of unflagged channels

Return type
list

`make_antenna_list(num_bins=5, antenna_list_file='')`

Make the antenna addition list splited in number of bins

Parameters

- **num_bin** (*int*) – Number of antenna bins
- **antenna_list_file** (*str*) – Name of the file to save antenna list. If an antenna list file given, antenna list will be loaded from that file

Returns

- *list* – Lists of antennas to be added in steps
- *int* – Number of antenna

model_imported()

Fuction to check whether model imported or model column exists or not :returns: True if model exists and False if does not :rtype: bool

move_phasecenter_to_source(*radec=''*)

Function to move the phasecenter of measurement set at particular RA-DEC

Parameters
radec (*str*) – RA DEC string, format 'J2000 00h00m00s +00d00m00s'

Returns
Message of the phasecenter of the measurement set is moved to the new phasecenter

Return type
str

move_phasecenter_to_sun()

Move the phasecenter of the measurement set at the center of the Sun

Returns
Message of the measurement set phase center chaged at the center of the Sun

Return type
str

radec_sun()

RA DEC of the Sun at the middle of the measurement set

Returns

- *str* – RA DEC of the Sun in J2000
- *float* – RA in degree
- *float* – DEC in degree

paircars.access_ms.**splited_ms_rename**(*msname*, *ref_time_chan=True*, *change_msname=True*)

Convert the name of a splited measurement set according to its frequency and timestamp

Parameters

- **msname** (*str*) – Name of the measurement set
- **ref_time_chan** (*bool*) – Whether the time and frequency slice is refernce
- **change_msname** (*bool*) – Change the msname or just return the name

Returns
Modified name of the measurement set

Return type
str

6.1.2 paircars.basic_func

class paircars.basic_func.**CalcParams**(*msname*, *quality_factor*, *safety_factor*)

Bases: `object`

Calculate calibration parameters

Parameters

- **msname** (*str*) – Name of the measurement set
- **quality_factor** (*int*) – Imaging quality factor (0,1,2)
- **safety_factor** (*int*) – Safety factor for calibration (0,1,2)

calc_calibration_params()

Return calibration parameters automatically based on given quality factor and safety standard For details read the documentation of quality_factor and safety_factor

Returns

- *float* – Starting sigma value for calibration
- *float* – Step to reduce sigma value with self-calibration
- *float* – Residual flux fraction to stop the calibration
- *float* – Minimum allowed sigma for threshold
- *float* – Minimum gain SNR
- *float* – Increment of DR_rms
- *float* – Increment of DR_neg
- *float* – Minimum allowed dynamic range
- *float* – Maximum dynamic range
- *float* – Minimum antenna based SNR for self-calibration
- *float* – Time interval for calibration
- *float* – Frequency interval of calibration
- *float* – uvrange for calibration

class paircars.basic_func.**ImageBasic**(*msname*, *includeflag=False*)

Bases: `object`

Generic class to calculate different imaging related parameters

Parameters

- **msname** (*str*) – Name of the measurement set
- **includeflag** (*bool*) – Include flag data or not whilw calculating maximum baseline

calc_calib_uvrange(*max_angular_scale*, *uvbin=10*, *includeflag=False*)

This function calculate the uvrange to be used for calibration.

Parameters

- **max_angular_scale** (*float*) – Maximum angular scale to exclude short baselines from calibration
- **uvbin** (*float*) – Binning in uv-lambda

- **includeflag** (*bool*) – Include flag data or not whilw calculating maximum baseline

Returns

- *str* – uv-range string (uvmin~uvmax lambda)
- *float* – uvmin in meter
- *float* – uvmax in meter
- *float* – uvmin in wavelength unit
- *float* – uvmax in wavelength unit

calc_cellsize(*num_pixel_in_psf*, *freq=0*)

Calculate pixel size in arcsec

Parameters

- **num_pixel_in_psf** (*int*) – Number of pixels in one PSF
- **freq** (*float*) – Frequency in MHz (default : 0, using central frequency of the ms)

Returns
Pixel size in arcsec

Return type
float

calc_max_baseline(*includeflag=False*)

Get the maximum baseline in meter :param includeflag: Include flag data or not whilw calculating maximum baseline :type includeflag: bool

Returns
Maximum baseline length in meter

Return type
float

calc_max_uvw()

Get the maximum u,v,w

Returns

- *float* – Maximum U
- *float* – Maximum V
- *float* – Maximum W

calc_psf(*freq=0*)

Function to calculate PSF size in arcsec

Parameters
freq (*float*) – Frequency in MHz (default : 0, using central frequency of the ms)

Returns
PSF size in arcsec

Return type
float

calc_suntaper(*uvbin=10*, *includeflag=False*)

Function return uv-taper value to treat Sun as a point source of size 16 arcmin :param uvbin: Binning in uv-lambda :type uvbin: float :param includeflag: Include flag data or not whilw calculating maximum baseline :type includeflag: bool

Returns
UV taper (uvtaper lambda)

Return type
str

calc_uvtaper(*uvbin=10*, *includeflag=False*)

Function return uv-taper value :param uvbin: Binning in uv-lambda :type uvbin: float :param includeflag: Include flag data or not whilw calculating maximum baseline :type includeflag: bool

Returns
UV taper (uvtaper lambda)

Return type
str

choose_scales(*num_pixel_in_psf*, *max_size*, *freq=0*)

Function to calculate multiscale scales

Parameters

- **num_pixel_in_psf** (*int*) – Number of pixels in one PSF
- **max_size** (*float*) – Maximum source size in arcsec
- **freq** (*float*) – Frequency in MHz (default : 0, using central frequency of the ms)

Returns
List of multiscale lists in number of pixels

Return type
list

field_of_view(*freq=0*)

Calculate optimum field of view in arcsec

Parameters
freq (*float*) – Frequency in MHz (default : 0, using central frequency of the ms)

Returns
Field of view in arcsec

Return type
float

num_pixels(*num_pixel_in_psf*, *freq=0*)

Number of image pixels

Parameters

- **num_pixel_in_psf** (*int*) – Number of pixels in one PSF
- **freq** (*float*) – Frequency in MHz (default : 0, using central frequency of the ms)

Returns
Number of pixels in the image

Return type
int

`paircars.basic_func.altaz_to_radec(alt, az, obstime, LAT, LON, ALT)`

Function to convert altaz to radec for a given Earth location

Parameters

- **alt** (*float*) – Elevation in degree
- **az** (*float*) – Azimuth in degree
- **obstime** (*str*) – Time of the observation in 'yyyy-mm-dd hh:mm:ss' format
- **LAT** (*float*) – Latitude of the Earth location in degree
- **LON** (*float*) – Longitude of Earth location in degree
- **ALT** (*float*) – Altitude of the Earth location in meter

Returns
['RA','DEC'] in hh mm ss dd mm ss format

Return type
numpy.array

`paircars.basic_func.calc_flag_chans_caltable(caltable, flag_frac=1.0)`

Calculate flaged channels from caltable

Parameters

- **caltable** (*str*) – Name of the CASA caltable
- **flag_frac** (*float*) – Minimum fraction of data flagged for a single channel

Returns

- *list* – Flagged channels list
- *float* – flag fraction

`paircars.basic_func.calc_flag_fraction(msname)`

Function to calculate the fraction of total data flagged

Parameters
msname (*str*) – Name of the measurement set

Returns
Fraction of the total data flagged

Return type
float

`paircars.basic_func.calc_flag_fraction_caltable(caltable)`

Calculate flaged fraction from caltable

Parameters
caltable (*str*) – Name of the CASA caltable

Returns
Fraction of the total solutions are flagged

Return type
float

`paircars.basic_func.check_internet(url='')`

Function to check internet connection :param url: An URL to check internet connectivity. (default : https://www.google.com) :type url: str

Returns
Whether internet connection is available or not

Return type
bool

`paircars.basic_func.compress_files(filelist, outputfile)`

Compress a list of numpy files

Parameters

- **filelist** (*list*) – List of numpy table files
- **outputfile** (*str*) – Output compressed file name

Returns
Compressed file name (Compressed file have arrays in format [original_filename,data_array])

Return type
str

`paircars.basic_func.download_metafits(msname, outdir)`

Function to download MWA metafits of a given measurement set (Require internet connection)

Parameters

- **msname** (*str*) – Name of the measurement set
- **outdir** (*str*) – Name of the outputdir

Returns
Metafits file name

Return type
str

`paircars.basic_func.error_msgs(err_code)`

Function to transform error code to error message

Parameters
err_code (*int*) – Error code

Returns
Error message

Return type
str

`paircars.basic_func.freq_to_MWA_coarse(freq)`

Frequency to MWA coarse channel conversion

Parameters
freq (*float*) – Frequency in MHz

Returns
MWA coarse channel number

Return type
int

`paircars.basic_func.get_MWA_phase(metafits)`

Function to get MWA phase

Parameters
metafits (*str*) – Name of the metafits file

Returns
MWA phase

Return type
str

paircars.basic_func.**get_OBSID_for_timerange**(*timerange=''*, *caldata=True*, *project_ID='G0002'*)

Function to get MWA observation IDs for a given time range :param timerange: Time range to search, format 'yy0/mm0/dd0/hh0:mm0:ss0.ff0~yy1/mm1/dd1/hh1:mm1:ss1.ff1,yy2/mm2/dd2/hh2:mm2:ss2.ff2~....' :type timerange: str :param caldata: Whether search for calibrator observations also or not :type caldata: bool :param project_ID: Project ID to search data :type project_ID: str

Returns

- *list* – OBSID list
- *list* – Calibration OBSID list

paircars.basic_func.**get_OBSID_from_metafits**(*metafits*)

Function to return OBSID of an MWA observation

Parameters
metafits (*str*) – Name of the metafits file

Returns
MWA Observation ID

Return type
int

paircars.basic_func.**get_OBSID_from_ms**(*msname*)

Function to return OBSID of an MWA observation

Parameters
msname (*str*) – Name of the measurement set

Returns
MWA Observation ID

Return type
int

paircars.basic_func.**get_available_paircars_instance**(*basedir*, *jobid*, *total_instances*)

Function to get available P-AIRCARS instances

Parameters

- **basedir** (*str*) – Name of the P-AIRCARS base directory
- **job_id** (*int*) – P-AIRCARS job ID
- **total_instances** (*int*) – Total P-AIRCARS instances

Returns
Available P-AIRCARS instance

Return type
int

`paircars.basic_func.get_caltable_metadata(caltable)`

Function to get caltable metadata

Parameters
caltable (*str*) – Name of the caltable

Returns
A python dictionary with keywords MSNAME, JonesType, Channel 0 frequency (MHz), Central channel frequency (MHz), Channel width (kHz), Bandwidth (MHz), Start time, End time

Return type
dict

`paircars.basic_func.get_final_psf(msname, imager='wsclean', weight='briggs', robust=0.5)`

Function to get a common final psf parameters

Parameters

- **msname** (*str*) – Name of the measurement set
- **imager** (*str*) – Imager, 'wsclean' or 'CASA'
- **weight** (*str*) – Visibility weighting for imaging
- **robust** (*float*) – Robust parameter for 'briggs' weighting

Returns

- *float* – Major axis (FWHM) in arcsec
- *float* – Minor axis (FWHM) in arcsec
- *float* – Position angle in degree

`paircars.basic_func.getnearpos(array, value)`

Function to return index of two elements nearest to a given number

Parameters

- **array** (*numpy.array*) – Input numpy array
- **value** (*float*) – Value to which nearest element search

Returns
Index of two elements nearest to the value

Return type
float

`paircars.basic_func.mjdsec_to_timestamp(mjdsec, includedate=True, format=0)`

Convert CASA MJD seceonds to CASA timestamp

Parameters

- **mjdsec** (*float*) – CASA MJD seconds
- **includedate** (*bool*) – Include date in timestamp
- **format** (*int*) –

  Datetime string format
  0: 'YYYY/MM/DD/hh:mm:ss'

  1: 'YYYY-MM-DDThh:mm:ss'

  2: 'YYYY-MM-DD hh:mm:ss'

3: 'YYYY_MM_DD_hh_mm_ss'

Returns
CASA time stamp in UTC at the format

Return type
str

paircars.basic_func.**multifreq_gaincal_interpolate**(*gaintables=[]*, *output_gaintable=''*, *outputfreq=0*)

Function to calculate linearly interpolated gain phase from a set of gaintables (at-least two) at multiple frequencies

Parameters

- **gaintables** (*list*) – List of gaintables at multiple frequencies
- **output_gaintable** (*str*) – Name of the output gaintable
- **outputfreq** (*float*) – Output frequency in MHz

Returns
Output gaintable name

Return type
str

paircars.basic_func.**paircars_instance_runner**(*cmd*, *basedir*, *paircars_basedir*, *screen_name*, *finished_touch_file*, *jobid*, *prefix_cmds=[]*)

Function to run a P-AIRCARS instance

Parameters

- **cmd** (*str*) – Command to run
- **basedir** (*str*) – Base directory of the measurement set
- **paircars_basedir** (*str*) – Base directory for a particular P-AIRCARS job
- **screen_name** (*str*) – Name of the screen
- **finished_touch_file** (*str*) – Hidden file to generate at the base directory after finishing the instance

paircars.basic_func.**radec_con_deg_to_hhmmss**(*radeg*, *decdeg*)

Convert RA-DEC from degree to hh mm ss dd mm ss format

Parameters

- **radeg** (*float*) – Value of the RA in degree
- **decdeg** (*float*) – Value of the DEC in degree

Returns
['RA','DEC'] in hh mm ss dd mm ss format

Return type
numpy.array

paircars.basic_func.**radec_to_altaz**(*ra*, *dec*, *obstime*, *LAT*, *LON*, *ALT*)

Function to convert radec to altaz for a given Earth location

Parameters

- **ra** (*str*) – RA either in degree or 'hh:mm:ss' or '%fh%fm%fs' format
- **dec** (*str*) – DEC either in degree or 'dd:mm:ss' or '%fd%fm%fs'format

- **obstime** (*str*) – Time of the observation in 'yyyy-mm-dd hh:mm:ss' format
- **LAT** (*float*) – Latitude of the Earth location in degree
- **LON** (*float*) – Longitude of Earth location in degree
- **ALT** (*float*) – Altitude of the Earth location in meter

Returns

- *float* – Elevation
- *float* – Azimuth in degree

`paircars.basic_func.timestamp_to_mjdsec(timestamp, format=0)`

Convert timestamp to mjd second

Parameters

- **timestamp** (*str*) – Time stamp to convert
- **format** (*int*) –

  Datetime string format

  0: 'YYYY/MM/DD/hh:mm:ss'

  1: 'YYYY-MM-DDThh:mm:ss'

  2: 'YYYY-MM-DD hh:mm:ss'

  3: 'YYYY_MM_DD_hh_mm_ss'

Returns

Return correspondong MJD second of the day

Return type

float

`paircars.basic_func.update_mwa_obsids(obsid_file='', verbose=False)`

Function to update MWA OBSIDs

Parameters

- **obsid_file** (*str*) – Name of the file to save MWA OBSIDs
- **verbose** (*bool*) – Verbose output

Returns

- *str* – OBSID file name
- *int* – Update success or failure code (0 or 1)

6.1.3 paircars.decor

`paircars.decor.apply_acorr(dat, ants1, ants2, idx_ant, elen, tile, uvw, BW, fout)`

Amplitude correction for the beamformer-to-receiver cable length differences ('decorrelation')

Parameters

- **dat** (*numpy.array*) – Data array
- **ants1** (*numpy.array*) – Antenna 1 array
- **ants2** (*numpy.array*) – Antenna 2 array

- **idx_ant** (*numpy.array*) – Antenna indices
- **elen** (*numpy.array*) – Electronic length of the tiles
- **tile** (*numpy.array*) – Tile names
- **uvw** (*numpy.array*) – UVW array
- **BW** (*float*) – Channel width
- **fout** (*str*) – File to wrire correction factors

Returns
Amplitude corrected data array

Return type
numpy.array

`paircars.decor.decor(msname, metafits, n_tblk, single_time)`

Function to correct the MS for amplitude decorrelation. MS convention is also converted from MWA coordinate (X=E->W and Y=N->S) to IAU coordinate (X=S->N, Y=W->E).

Parameters

- **msname** (*str*) – Name of the measurement set
- **metafits** (*str*) – Name of the metafits file of the observation
- **n_tblk** (*int*) – Number of time block for one round to avoid memory load in case of large dataset
- **single_time** (*bool*) – Single time block or not

Return type
Decorrelation corrected measurement set in IAU convention

6.1.4 paircars.dynamic_spectrum

`class paircars.dynamic_spectrum.DynamicSpectrum(msname)`

Bases: `object`

`cal_norm_crosscorr()`

Function to obtain normalised cross correlation amplitude

6.1.5 paircars.flagger

`paircars.flagger.casa_autoflag(msname, mode='rflag', datacolumn='corrected', sigma_thresh=5.0, flagbackup=False, verbose=False)`

Function to perform CASA flag :param msname: Name of the measurement set :type msname: str :param mode: CASA flag mode, 'rflag' or 'tfcrop' :type mode: str :param datacolumn: Datacolumn to flag, 'data', 'corrected' or 'residual' :type datacolumn: str :param sigma_thresh: Flagging threshold :type sigma_thresh: float :param flagbackup: Backup flags or not :type flagbackup: bool :param verbose: Verbose output :type verbose: bool

Returns
Success code, 0 or 1

Return type
int

`paircars.flagger.`**`do_uvsub_ankflag`**(*msname*, *model=''*, *nthread=0*, *verbose=False*, *flagbackup=True*, *extendpols=False*, *chantime_minfrac=0.5*, *casaflag=''*)

Perform flagging on uv sub data using aNKflagger

Parameters

- **msname** (*str*) – Name of the measurement set
- **model** (*str*) – Model image name, Keep blank if model is already in modelcolumn
- **nthread** (*int*) – Number of CPU threads to be used by aNKflag. If 0, it will use 25% of the total available CPU threads.
- **verbose** (*bool*) – If True keep all records
- **flagbackup** (*bool*) – Keep flagbackup
- **extendpols** (*bool*) – Extend flag if one polarisation is flagged
- **chantime_minfrac** (*float*) – Minimum fraction of data flagged for a single channel and time to extend the flag
- **casaflag** (*str*) – Perform CASA flags ('tfcrop' or 'rflag')

Returns
Flagged measurement set name

Return type
str

`paircars.flagger.`**`do_uvsub_flagger`**(*msname*, *model=''*, *mode=''*, *rmsthresh=[]*, *flagbackup=True*)

Flagger on residual data

Parameters

- **msname** (*str*) – Name of the measurement set
- **model** (*str*) – Name of the model image, keep blank if model is alrealy in modelcolumn
- **rmsthresh** (*list*) – List of rms threshold
- **flagbackup** (*bool*) – Keep flagbackup

Returns
New number of flaggs

Return type
int

`paircars.flagger.`**`flag_MWA_coarse`**(*msname*, *edgewidth=80*, *do_flag=True*, *force=False*, *flagbackup=True*)

A function to generate the list of coarse-channels edges and the central channel in each coarse channel to be flagged.

Parameters

- **msname** (*str*) – Name of the masurement set
- **edgewidth** (*float*) – Flag edge channels width in kHz
- **do_flag** (*bool*) – If true flag edge channels, otherwise return only the good channels list
- **force** (*bool*) – If True flag again if even if the flag keyword is in header
- **flagbackup** (*bool*) – Keep flag backup or not

Returns

- *str* – Unflagged channels
- *dict* – One central channel per coarse channel

`paircars.flagger.flag_MWA_quack(msname, metafits, quacktime=0.0, flagbackup=False, force=False)`

Function to flag MWA Quack times

Parameters

- **msname** (*str*) – Name of the measurment set
- **metafits** (*str*) – Name of the metafits file
- **quacktime** (*float*) – Quack time
- **flagbackup** (*bool*) – Backup flags or not
- **force** (*bool*) – If True flag again if even if the flag keyword is in header

Returns
Quack time

Return type
float

`paircars.flagger.flag_zeros(msname, flagbackup=False, force=False)`

Function to flag zero data

Parameters

- **msname** (*str*) – Name of the measurement set
- **flagbackup** (*bool*) – Backup flags or not
- **force** (*bool*) – Force to flag even if the header saying flagging is already done

`paircars.flagger.get_bad_ants(msname, flagfrac=1.0)`

Function to get the antennas for which a certain amount of data are flagged

Parameters

- **msname** (*str*) – Name of the measurement set
- **flagfrac** (*float*) – Fraction of data flagged to consider the antennas as bad (default : 1.0)

Returns
Bad antenna strings

Return type
str

6.1.6 paircars.fullpol_selfcal_LTS

`class paircars.fullpol_selfcal_LTS.PolSelfcal(msname, metafits, maximum_emission_scale, num_pixel_in_psf=5, largest_scale=12, verbose=False, interactive=False, savelog=True, use_wsclean=True)`

Bases: `object`

Generic class to perform polarisation self-calibration (Using Andre Offringa's CALIBRATE code on based Mitchcal algorithm)

Parameters

- **msname** (*str*) – Name of the measurement set

- **metafits** (*str*) – Name of the MWA metafits file
- **num_pixel_in_psf** (*int*) – Number of pixels side one point-spread-funtion
- **maximum_emission_scale** (*float*) – Maximum scale of the emission present in the image
- **largest_scale** (*float*) – Largest spatial scale in degree used for self calibration
- **verbose** (*bool*) – If True keep all the intermediate images, model, residuals, caltables and details of the log to detailed analysis
- **interactive** (*bool*) – If True user have interactive control on self-calibration
- **savelog** (*bool*) – Save log
- **use_wsclean** (*bool*) – Use WSClean or not

B_to_IQUV(*B*, *pol_basis='Linear'*)

Convert brightness matrix in instrumental basis to I, Q, U, V

Parameters

- **B** (*numpy.array*) – Brightness matrix in instrumental basis
- **pol_basis** (*str*) – Polarisation basis of the instrument, Linear or Circular (Circular basis not implemented)

Returns
Stokes I, Q, U,V

Return type
dict

IMSTAT_record(*DRI*, *DR_neg*, *FXI*, *FXQ*, *FXU*, *FXV*, *FXT*, *FXP*, *record_filename*, *init=True*)

Function to keep the record of image statistics at different self calibration steps

Parameters

- **DRI** (*float*) – RMS based dynamic range of the Stokes I
- **DR_neg** (*float*) – Negative based dynamic range of the Stokes I
- **FXI** (*float*) – Total Stokes I flux
- **FXQ** (*float*) – Total Stokes Q flux
- **FXU** (*float*) – Total Stokes U flux
- **FXV** (*float*) – Total Stokes V flux
- **FXT** (*float*) – Total Stokes T flux
- **FXP** (*float*) – Total Stokes P flux
- **record_filename** (*str*) – Name of the file to stro dynamic ranges
- **init** (*bool*) – Initiating a new record from the current selfcal iteration

Returns
Image statistic record array; shape [7,num_of_record]

Return type
numpy.array

antenna_string(*antenna_list*, *antenna_list_index*)

Function to return antenna string from antenna list

Parameters

- **antenna_list** (*list*) – Antenna list or array
- **antenna_list_index** (*int*) – Bin number of antenna list

Returns
Antenna string

Return type
str

apply_cross_hand_phase(*cross_phase=15*, *caltable=''*, *polbasis='Linear'*, *datacolumn='DATA'*, *modify_datacolumn=False*)

Function to applycal cross hand phase solution

Parameters

- **cross_phase** (*float*) – Cross hand phase different in degree
- **caltable** (*str*) – Name of the caltable to store cross hand Jones matrices
- **polbasis** (*str*) – 'Linear' or 'Circular' (Circular basis not implemented)
- **datacolumn** (*str*) – Datacolumn to apply the solution ('DATA', or 'CORRECTED_DATA')
- **modify_datacolumn** (*bool*) – Modify the DATA column or not

Returns
Caltable name

Return type
str

cal_poldistortion(*gaintable*, *poldistortion_matrix='UH'*)

Function to calculate the estmated Jones matrices for poldistortion after correcting for instrumental Jones matrix Note : Saved X matrix is for B'=XBX^dagger, which is inverse of poldistortion_matrix of correct_poldistortion function

Parameters

- **gaintable** (*str*) – Name of the gaintable (Assuming CALIBRATE gaintable format only right now)
- **_matrix** (*poldistortion*) – 'UH' or 'HU', where H is polconversion matrix and U is the polrotation matrix

Returns

- *numpy.matrix* – Poldistortion matrix
- *numpy.matrix* – Inverse of poldistortion matrix
- *numpy.matrix* – Polconversion
- *numpy.matrix* – Inversion of polconversion
- *numpy.matrix* – Polrotation
- *numpy.matrix* – Inverse of polrotation
- *str* – Filename to save poldistortion matrix

cal_solar_qu_leakage(*imagename*, *sigma=10*, *do_fluxcal=False*)

Function to calculate Stokes QU leakage for solar observation (Not vaild for any other astrophysical observation)

Parameters

- **imagename** (*str*) – Name of the image
- **do_fluxcal** (*bool*) – Perform flux calibration or not
- **outfile_path** (*str*) – Name of the directory to save leakage data numpy table (default : image directory)

Returns

- *float* – Stokes Q leakage
- *float* – Stokes U leakage
- *(Two numpy table will also be saved)*

cal_solar_v_leakage(*imagename*, *sigma=10*, *do_fluxcal=False*)

Function to calculate Stokes V leakage for solar observation (Not vaild for any other astrophysical observation)

Parameters

- **imagename** (*str*) – Name of the image
- **sigma** (*float*) – Sigma value for thresholding
- **do_fluxcal** (*bool*) – Perform flux calibration or not

Returns
Stokes V leakage (A numpy table will also be saved)

Return type
float

calc_dyn_range(*imagename*, *sigma*, *box_width=3*, *stokes_list=['I']*)

Calculate the dynamic range of the full Stokes image cube

Parameters

- **imagename** (*str*) – Name of the CASA image
- **sigma** (*float*) – nsigma value to put a mask for calculating total flux
- **box_width** (*float*) – Negative box width around the maximum pixel in degree (default : 3 degree)
- **stokes_list** (*list*) – List of stokes planes in the image

Returns

- *dict* – {'STOKES':[rms dynamic range,rms,total_flux(non-negative)]}
- *float* – negative dynamic range for Stokes I

calc_iter_num(*safety_factor*, *quality_factor*, *scratch=True*)

Function to calculate minimum number of selfcal iteration based on safety standard and quality factor

Parameters

- **safety_factor** (*int*) – Factor to determine the robustness of the selfcal
- **quality_factor** (*int*) – Factor to determine the quality of the images

- **scratch** (*bool*) – Whether start the selfcal from scratch or not

Returns

- *int* – Minimum iteration at fixed sigma
- *int* – Minimum iteration
- *int* – Maximum iteration
- *int* – Number of antenna bins
- *float* – Fraction change in flux for convergence
- *float* – Minimum value of allowed sigma

compare_leakage_for_sun(*present_image=''*, *previous_image=''*, *present_model=''*, *previous_model=''*, *outputfile_prefix=''*, *sigma=10*, *overwrite=False*, *qucor_step=False*)

Function to compare Stokes I to Stokes Q,U,V leakage from quiet Sun part with the previous image

Parameters

- **present_image** (*str*) – Name of the present CASA image
- **previous_image** (*str*) – Name of the previous CASA image
- **present_model** (*str*) – Name of the present CASA model
- **previous_model** (*str*) – Name of the previous CASA model
- **outputfile_prefix** (*str*) – Final output file prefix name
- **sigma** (*float*) – Sigma value for rms based thresholding
- **overwrite** (*bool*) – Overwrite the present image or not
- **qucor_step** (*bool*) – Performed images based Stokes Q, U correction at last selfcal iteration

Returns
Output file name

Return type
str

compare_leakages(*present_image=''*, *previous_image=''*, *present_model=''*, *previous_model=''*, *outputfile_prefix=''*, *overwrite=False*, *TB_limit=- 1*)

Function to compare Stokes I to Stokes Q,U,V leakages with the previous image

Parameters

- **present_image** (*str*) – Name of the present CASA image
- **previous_image** (*str*) – Name of the previous CASA image
- **present_model** (*str*) – Name of the present CASA model
- **previous_model** (*str*) – Name of the previous CASA model
- **outputfile_prefix** (*str*) – Final output file prefix name
- **overwrite** (*bool*) – Overwrite the present image or not
- **TB_limit** (*float*) – Brightness temperature limit to calculate polarised flux density

Returns
Output file name

Return type
str

`correct_for_single_beam_jones`(*imagename*, *outfile*, *beam_jones*, *imagetype='FITS'*, *outtype='FITS'*, *pol_basis='Linear'*)

Correct Stokes IQUV image cube for full Stokes Beam Jones at a single pointing

Parameters

- **imagename** (*str*) – Name of the image of model
- **outfile** (*str*) – Name of the beam corrected image or model
- **beam_jones** (*numpy.array*) – Beam jones matrix
- **imagetype** (*str*) – Type of the image, CASA or FITS
- **outtype** (*str*) – Output image type, CASA or FITS
- **pol_basis** (*str*) – Polarisation basis of the instrument, Linear or Circular (Circular basis not implemented)

Returns
Beam corrected image or model

Return type
str

`correct_image_for_cross_phase`(*imagename*, *modelname*, *outfile*, *cross_phase=15*, *imagetype='FITS'*, *outtype='FITS'*, *pol_basis='Linear'*, *do_fluxcal=False*)

Correct Stokes IQUV image cube for full Stokes Beam Jones at a single pointing

Parameters

- **imagename** (*str*) – Name of the image
- **modelname** (*str*) – Name of the model
- **outfile** (*str*) – Prefix name of the beam corrected image and model
- **cross_phase** (*str*) – Cross hand phase in degree
- **imagetype** (*str*) – Type of the image, CASA or FITS
- **outtype** (*str*) – Output image type, CASA or FITS
- **pol_basis** (*str*) – Polarisation basis of the instrument, Linear or Circular
- **do_fluxcal** (*str*) – Perform flux scaling or not

Returns
Cross hand phase corrected image and model

Return type
str

`correct_poldistortion`(*gaintable*, *outfile*, *poldistortion_matrix*)

Function to applycal poldistortion correction (either polconversion or polrotation or both) to the gaintable

Parameters

- **gaintable** (*str*) – Name of the gaintable
- **outfile** (*str*) – Name of the output poldistortion corrected gaintable
- **poldistortion_matrix** (*numpy.array*) – Poldistortion matrix

Returns
Poldistortion corrected gaintable name

Return type
str

`correct_solar_quv_leakage`(*imagename*, *modelname*, *sigma*, *overwrite=False*, *stokes='QUV'*)

Function to correction for solar Stokes Q, U and V leakage (It is based on the fact that we do not expect any linear polarisation from the Quiet Sun emission)

Parameters

- **imagename** (*str*) – Name of image
- **modelname** (*str*) – Name of the model
- **sigma** (*float*) – N-sigma threshold to choose Stokes I emission region
- **overwrite** (*bool*) – Overwrite the image and model or not
- **stokes** (*str*) – Stokes plane to correct

Returns

- *str* – Stokes Q ,U, V leakage corrected image name
- *str* – Stokes Q ,U, V leakage corrected model name
- *float* – Stokes Q fractional change
- *float* – Stokes U fractional change
- *float* – Stokes V fractional change

`correct_visibility_single_beam_jones`(*datacolumn='DATA'*, *modify_datacolumn=True*, *force=False*, *skip_freq=1.28*, *save_beamfile=''*)

Correct visibility data for a single pointing beam jones

Parameters

- **datacolumn** (*str*) – 'DATA', datacolumn to apply beam correction
- **modify_datacolumn** (*bool*) – Modify the DATA column, otherwise beam corrected visibilities will be saved on CORRECTED_DATA
- **force** (*bool*) – Beam correct forcefully avoiding ms header info
- **skip_freq** (*float*) – Frequency interval in MHz to make independent beams (default : 1.28 MHz). If anything greater than 1.28 MHz is given it will be overwritten to 1.28 MHz
- **save_beamfile** (*str*) – Save beam file in this given name

Returns
Name of the beam jones file, Beam Jones matrix.

Return type
str

`create_circular_mask`(*h*, *w*, *center=None*, *radius=None*)

Function to create a circular mask

Parameters

- **h** (*int*) – Number of pixels along Y axis
- **w** (*int*) – Number of pixels along X axis

- **center** (*tuple*) – (x_cen,y_cen), center of the mask circle
- **radius** (*float*) – Radius in number of pixels

Returns
Cirular mask

Return type
numpy.array

estimateSkyBrightnessMatrix(*beam_jones*, *Vij*)

Return beam corrected brightness matrix

Parameters

- **beam_jones** (*numpy.array*) – Beam Jones matrix
- **Vij** (*numpy.array*) – Instrumental brightness matrix

Returns
Beam corrected brightness matrix in instrumental basis

Return type
numpy.array

file_remover_and_keeper(*num_iter*, *msg_code*, *ref_time_chan=True*)

This function keep and remove caltables, ms, imaging related files based on the need

Parameters

- **num_iter** (*int*) – Number of self-calibration iteration
- **msg_code** (*str*) – Selfcal message code
- **ref_timechan** (*bool*) – Reference time channel or not

get_IQUV(*imagename*, *imagetype='FITS'*)

Stokes I,Q,U,V from a Stokes IQUV image cube

Parameters

- **imagename** (*str*) – Name of the image
- **imagetype** (*str*) – Type of the image, CASA or FITS

Returns
{'STOKES':imagedata}

Return type
dict

get_inst_pols(*stokes_image*, *imagetype='FITS'*, *pol_basis='Linear'*)

Return instrumental polarisation matrix (Vij)

Parameters

- **stokes_image** (*str*) – Name of the Stokes IQUV image cube
- **imagetype** (*str*) – Type of the image, CASA or FITS
- **pol_basis** (*str*) – Polarisation basis of the instrument, Linear or Circular (Circular basis not implemented)

Returns
Instrumental polarisation matrix

Return type
numpy.array

`make_crosshand_phase_caltable`(*cross_phase=15*, *caltable=''*, *polbasis='Linear'*)

Function to make cross hand phase Jones matrices

Parameters

- **cross_phase** (*float*) – Cross hand phase different in degree
- **caltable** (*str*) – Name of the caltable to store cross hand Jones matrices
- **polbasis** (*str*) – 'Linear' or 'Circular' (Circular basis not implemented)

Returns
Caltable name

Return type
str

`mwa_solar_fluxcal`(*imagename*, *outfile*, *atten=10*)

Function to flux calibrate MWA solar observations using method described in Kansabanik et al. 2021

Parameters

- **imagename** (*str*) – Name of the image
- **outfile** (*str*) – Output file name
- **atten** (*float*) – Attenuator gain value in dB

Returns
Flux calibrated image

Return type
str

`negative_box`(*max_pix*, *box_width=3*)

Create a 3 degree box about the maximum pixel of image to search negative

Parameters

- **max_pix** (*list*) – Maximum pixel [xxmax,yymax]
- **box_width** (*int*) – Box width in degree (default : 3 degree)

Returns
CASA box 'xblc,yblc,xrtc,yrtc'

Return type
str

`pol_model_threshold`(*imagename*, *modelname*, *sigma*, *polmodel_thresh*)

Function to put rms based threshold on polarisation model

Parameters

- **imagename** (*str*) – Name of the image
- **modelname** (*str*) – Name of the model
- **sigma** (*float*) – Sigma value for threshold
- **polmodel_thresh** (*float*) – Multiplying factor to sigma for polarisation model threshold

Returns

- *str* – Imagename
- *str* – Modelname

polselfcal_iteration(*num_iter*, *rms_thresh*, *mask_str*, *sigma*, *maskfile*, *antenna_to_use*, *startmodel*, *startmask*, *want_auto_masking=False*, *TB_limit=- 1*, *solar_imaging=True*, *stokes=''*, *interactive=False*, *use_ankflagger=False*, *do_flag=False*, *poldistortion_correction=True*, *poldistortion_type='poldistortion'*, *crossphase=- 1*, *poldistortion_matrix='UH'*, *do_solarqu_cor=False*, *box_width=3*, *previous_image=''*, *previous_model=''*, *calibrator_caltable=[]*, *maskregion=''*, *quvcor_stokes='QUV'*, *polmodel_threshold=- 1*, *cpus=5*, *absmem=2*, *wlayers=1*, *weight='briggs'*, *robust=0.5*)

Function to perform a polarisation self-calibration loop, make an image, put the model in the measurement set, and perform the calibration

Parameters

- **num_iter** (*int*) – Number of self-calibration iteration
- **rms_thresh** (*float*) – RMS for threshold
- **maskstr** (*str*) – Mask string for CLEANing (Only for CASA)
- **sigma** (*float*) – Threshold sigma
- **maskfile** (*str*) – Maskfile for CLEANing (Only for CASA)
- **antenna_to_use** (*list*) – List of antennas for CLEANing
- **startmodel** (*str*) – Model to start the CLEANing (Only for CASA)
- **startmask** (*str*) – Mask to start (Only for CASA)
- **want_auto_masking** (*bool*) – If True use CASA auto-multithresh for auto masking
- **TB_limit** (*float*) – Brightness temperature limit to calculate polarised flux to compare between two polarisation self-calibration rounds (Only for non MWA solar observations)
- **solar_imaging** (*bool*) – Performing Solar image calibration
- **stokes** (*str*) – Stokes plane to image
- **interactive** (*bool*) – Perform interactive CLEAN (Only for CASA)
- **use_ankflagger** (*bool*) – Use aNKflagger for flagging after each selfcal round
- **do_flag** (*bool*) – Flag after selfcal round
- **poldistortion_correction** (*bool*) – Correct poldistortion using the known ideal Jones matrix of the instrument
- **poldistortion_type** (*str*) – 'polconversion ; Stokes I to STOKES Q,U,V leakages' or 'polrotation; changes between Stokes Q,U,V' or 'poldistortion' (default : poldistortion)
- **crossphase** (*float*) – Cross hand phase in degree for image based correction
- **poldistortion_matrix** (*str*) – 'UH or HU ' , where H is polconversion and U is polrotation
- **do_solarqu_cor** (*bool*) – Correct solar Stokes I to Q,U imaged based leakage correction
- **box_width** (*float*) – Length of negative box width in degree (default : 3 degree)
- **previous_image** (*str*) – Name of the previous round image to compare leakage
- **previous_model** (*str*) – Name of the previouos round model

- **calibrator_caltable** (*list*) – List of calilbrator caltables
- **maskregion** (*str*) – Mask region in case of auto-masking (Only for CASA)
- **quvcor_stokes** (*str*) – Stokes plane for images based leakage correction
- **polmodel_threshold** (*float*) – Sigma value for thresholding on the polarisation model
- **cpus** (*int*) – Number of cpu threads to use
- **absmem** (*float*) – Memory in GB to use
- **wlayers** (*int*) – Number of w-stacking layers (For wsclean only)
- **weight** (*str*) – Visibility weighting during imaging , 'uniform', 'natural' or 'briggs'. Default is 'briggs'
- **robust** (*float*) – Robust parameter for briggs weighting, default : 0.5

Returns

- *int* – Message code
- *dict* – Dynamic range information [DR dictionary : {'STOKES':[rms dynamic range,rms,total_flux]}]
- *float* – Negative based dynamic range

reduce_sigma(*imagename*, *nsigma*, *sigma_step*, *minsigma*, *pre_residual=0.0*, *residual_frac=0.01*, *stokes_list=['I']*)

Function to determine whether reduce the CLEAN sigma or not

Parameters

- **imagename** (*str*) – Name of the image
- **nsigma** (*float*) – Value of the present n-sigma
- **sigma_step** (*float*) – Step to reduce sigma value
- **minsigma** (*float*) – Minimum allowed sigma
- **pre_residual** (*float*) – Previous residual fraction to compare (default : 0.0)
- **residual_frac** (*float*) – Residual flux fraction to reduce sigma (default : 0.01)
- **stokes_list** (*list*) – Stokes plane list

Returns

- *float* – Reduced value of n-sigma and median residual fraction if residual flux is more than given percentage (default : 1%) of the total flux in Stokes I or in all Stokes Q,U,V.
- *float* – Median residual flux fraction over all stokes planes

remove_model_negative(*imagename*, *modelname*, *sigma=10*, *overwrite=False*)

Function to remove negatives from model image

Parameters

- **imagename** (*str*) – Name of the image
- **modelname** (*str*) – Name of the model
- **sigma** (*float*) – Sigma value for thresholding
- **overwrite** (*bool*) – Overwrite the model image or not

Returns
Model imagename without negatives

Return type
str

solarcir_pol_minimise(*datai*, *datav*, *l*, *rmsv*, *mean_i_flux*, *sigma*)

Polarisation minimisation function for Sun

Parameters

- **datai** (*numpy.array*) – Stokes I image data
- **datav** (*numpy.array*) – Stokes V image data
- **l** (*float*) – Trial leakage (-1 to 1)
- **rmsv** (*float*) – RMS of the Stokes V map
- **mean_i_flux** (*float*) – Mean brightness in Jy/beam
- **sigma** (*float*) – Sigma value for thresholding
- **Return** –
- **int** – The number of pixels having polarisation fraction greater than rmsl/i_flux

solarlin_pol_minimise(*datai*, *datal*, *l*, *rmsl*, *i_flux*)

Polarisation minimisation function for Sun

Parameters

- **datai** (*numpy.array*) – Stokes I image data
- **datal** (*numpy.array*) – Stokes Q or U image data
- **l** (*float*) – Trial leakage (-1 to 1)
- **rmsl** (*float*) – RMS of the Stokes map
- **i_flux** (*float*) – Mean brightness in Jy/beam

Returns
The number of pixels having polarisation fraction greater than rmsl/i_flux

Return type
int

subtract_background_sources(*stokes*, *rms_thresh*, *sigma*, *modelthres*, *maskregion=''*, *includeregion=False*, *overwrite=False*, *modify_datacolumn=False*, *cpus=5*, *absmem=2*, *weight='briggs'*, *robust=0.5*)

Function to subtract background sources outside the mask region

Parameters

- **stokes** (*str*) – Stokes parameters to image
- **rms_thresh** (*list*) – RMS list for each Stokes plane
- **sigma** (*float*) – Sigma value for rms based thresholding
- **modelthresh** (*float*) – Sigma value to put a threshold on model
- **maskregion** (*str*) – Region outside which the sources are subtracted.
- **includeregion** (*bool*) – Include the maskregion for subtraction (True) or exclude the mask region for subtraction (False).

- **overwrite** (*bool*) – Overwrite the corrected datacolumn with the subtracted visibilities.
- **modelify_datacolumn** (*bool*) – Modify the datacolumn
- **weight** (*str*) – Visibility weighting during imaging , 'uniform', 'natural' or 'briggs'. Default is 'briggs'
- **robust** (*float*) – Robust parameter for briggs weighting, default : 0.5

Returns

- *str* – Backgroud source subtracted ms
- *bool* – Background source subtracted or not

subtract_leakage_surface(*imagename*, *modelname*, *sigma=10*, *do_fluxcal=False*, *overwrite=False*)

Function to subtract quadratic leakage surface

Parameters

- **imagename** (*str*) – Name of the image
- **modelname** (*str*) – Name of the model
- **sigma** (*float*) – N-sigma value above which any emission is considered to be real
- **do_fluxcal** (*bool*) – Perform polynomial based flux calibration (See details Kansabanik et al. 2021, submitted to ApJ)
- **overwrite** (*bool*) – Overwrite the input image and model

Returns

- *str* – Leakage surface subtracted image
- *str* – Leakage surface subtracted model
- *float* – Stokes Q fractional change
- *float* – Stokes U fractional change

subtract_stokesV_solar_leakage_surface(*imagename*, *modelname*, *sigma=10*, *do_fluxcal=False*, *overwrite=False*)

Function to subtract quadratic leakage surface

Parameters

- **imagename** (*str*) – Name of the image
- **modelname** (*str*) – Name of the model
- **sigma** (*float*) – N-sigma value above which any emission is considered to be real
- **do_fluxcal** (*bool*) – Perform polynomial based flux calibration (See details Kansabanik et al. 2021, submitted to ApJ)
- **overwrite** (*bool*) – Overwrite the input image and model

Returns

- *str* – Leakage surface subtracted image
- *str* – Leakage surface subtracted model
- *float* – Stokes V change fraction

uncorrect_for_single_beam_jones(*imagename*, *outfile*, *inv_beam_jones*, *imagetype='FITS'*, *outtype='FITS'*, *pol_basis='Linear'*)

Undo the beam correction for Stokes IQUV image cube for full Stokes Beam Jones at a single pointing

Parameters

- **imagename** (*str*) – Name of the image of model
- **outfile** (*str*) – Name of the beam corrected image or model
- **inv_beam_jones** (*numpy.array*) – Inverse of Beam jones matrix
- **imagetype** (*str*) – Type of the image, CASA or FITS
- **outtype** (*str*) – Output image type, CASA or FITS
- **pol_basis** (*str*) – Polarisation basis of the instrument, Linear or Circular

Returns
Beam un-corrected image or model

Return type
str

uncorrect_visibility_model_single_beam_jones(*force=False*, *skip_freq=1.28*)

Undo Correct visibility data for a single pointing beam jones

Parameters

- **force** (*bool*) – Undo beam correct forcefully avoiding ms header info
- **skip_freq** (*float*) – Frequency interval in MHz to make independent beams (default : 1.28 MHz). If anything greater than 1.28 MHz is given it will be overwritten to 1.28 MHz

Returns
Name of the beam jones file

Return type
str

uncorrect_visibility_single_beam_jones(*modify_datacolumn=True*, *force=False*, *skip_freq=1.28*)

Undo Correct visibility data for a single pointing beam jones

Parameters

- **modify** (*bool*) – Modify the DATA column, otherwise beam corrected visibilities will be saved on CORRECTED_DATA
- **force** (*bool*) – Undo beam correct forcefully avoiding ms header info
- **skip_freq** (*float*) – Frequency interval in MHz to make independent beams (default : 1.28 MHz). If anything greater than 1.28 MHz is given it will be overwritten to 1.28 MHz

Returns
Name of the beam jones file

Return type
str

wsclean_to_casaimage(*wsclean_images=[]*, *casaimage_prefix='CASA'*, *imagetype='image'*, *keep_wsclean_images=True*)

Function to convert WSClean image in CASA image (Stokes modes : 'IQUV', 'XXYY', 'I', 'QU', 'IV')

Parameters

- **wsclean_images** (*list*) – List of WSClean images
- **casaimage_prefix** (*str*) – Output CASA image name prefix (default : 'CASA_')
- **imagetype** (*str*) – 'image', 'model', 'residual' or 'dirty' (default : 'image')
- **keep_wsclean_images** (*bool*) – Keep the WSClean images or not

Returns
Output CASA imagename

Return type
str

6.1.7 paircars.intensity_selfcal_LTS

class paircars.intensity_selfcal_LTS.**IntensitySelfcal**(*msname*, *metafits*, *maximum_emission_scale*, *num_pixel_in_psf=5*, *largest_scale=20*, *verbose=False*, *interactive=False*, *use_wsclean=True*, *savelog=True*)

Bases: `object`

Generic class to perform intensity self-calibration

Parameters

- **msname** (*str*) – Name of the measurement set
- **metafits** (*str*) – Name of the metafits file
- **num_pixel_in_psf** (*int*) – Number of pixels side one point-spread-funtion
- **maximum_emission_scale** (*float*) – Maximum scale of the emission present in the image in arcsec
- **largest_scale** (*float*) – Largest spatial scale in degree used for self calibration
- **verbose** (*bool*) – If True keep all the intermediate images, model, residuals, caltables and details of the log to detailed analysis
- **interactive** (*bool*) – If True user have interactive control on self-calibration
- **use_wsclean** (*bool*) – Use WSClean or not

DR_delta(*quality_factor*, *DR_array*, *frac_increase*, *safety_factor*)

Function to calculate the dynamic range increse step (This function is not tested in the pipeline and not used. Use it at your own risk)

Parameters

- **quality_factor** (*int*) – Factor to determine the image quality
- **DR_array** (*numpy.array*) – Dynamic range records array
- **frac_increase** (*float*) – Fractional increase of the DR step from previous DR steps
- **safety_factor** (*int*) – Factor to determine the robustness of the selfcal

Returns
Dynamic range increase step

Return type
float

DR_record(*DR*, *record_filename*, *init=True*)

Function to keep the record of dynamic ranges at different self calibration steps

Parameters

- **DR** (*float*) – Dynamic range of the current selfcal iteration to record
- **record_filename** (*str*) – Name of the file to store dynamic ranges
- **init** (*bool*) – Initiating a new record from the current selfcal iteration

Returns
Dynamic range record array

Return type
numpy.array

antenna_string(*antenna_list*, *antenna_list_index*)

Function to return antenna string from antenna list

Parameters

- **antenna_list** (*list*) – Antenna list
- **antenna_list_index** (*int*) – Bin number of antenna list

Returns
Antenna string

Return type
str

bpass_solver(*caltable*, *spw=''*, *timerange=''*, *calmode='ap'*, *uvrange=''*, *solnorm=True*, *refant='1'*, *minsnr=3*, *solmode=''*, *rmsthresh=[]*, *gaintable=[]*)

Function to solve bandpass phase-only or amplitude-phase

Parameters

- **caltable** (*str*) – Name of the caltable
- **spw** (*str*) – Spectral window range
- **timerange** (*str*) – CASA timerange
- **calmode** (*str*) – 'p' for phase-only or 'ap' for amplitude-phase calibration
- **uvrange** (*str*) – UV-range for calibration
- **solnorm** (*bool*) – Normalise solution or not
- **refant** (*str*) – Reference antenna
- **minsnr** (*float*) – Minimum gain SNR
- **solmode** (*str*) – 'R' for robust calibration
- **rmsthresh** (*list*) – List of n-sigma values for rms based flagging (Do not put below 6)
- **gaintable** (*list*) – List of any previous caltables

Returns
Name of the caltable

Return type
str

cal_solar_phaseshift(*imagename*, *sigma=10*)

Calculate the different between solar center and phase center of the image

Parameters

- **imagename** (*str*) – Name of the image
- **sigma** (*float*) – Threshold for model fitting (default =10)

Returns

- *float* – New RA of the phasecenter in degree
- *float* – New DEC of the phasecenter in degree
- *bool* – Whther phase shift required or not. NOt required if less than image pixel size

calc_dyn_range(*num_iter*, *sigma*, *box_width=3*, *stokes_list=['I']*)

Calculate the dynamic range of the full Stokes image cube

Parameters

- **num_iter** (*int*) – Number of selfcal iteration
- **sigma** (*float*) – nsigma value to put a mask for calculating total flux
- **box_width** (*float*) – Negative box width around the maximum pixel in degree (default : 3 degree)
- **stokes_list** (*list*) – List of stokes planes in the image

Returns

- *dict* – Dynamic range information {'STOKES':[rms dynamic range,rms,total_flux(non-negative)]}
- *float* – Negative dynamic range for Stokes I

calc_iter_num(*safety_factor*, *quality_factor*, *scratch=True*, *bandpass_selfcal=False*)

Function to calculate minimum number of selfcal iteration based on safety standard and quality factor

Parameters

- **safety_factor** (*int*) – Factor to determine the robustness of the selfcal
- **quality_factor** (*int*) – Factor to determine the quality of the images
- **scratch** (*bool*) – Whether the selfcal started from scratch or not
- **bandpass_selfcal** (*bool*) – Performing bandpass selfcal or not

Returns

- *int* – Minimum iteration at fixed sigma
- *int* – Minimum iteration
- *int* – Maximum iteration
- *int* – Number of antenna bins
- *float* – Fractional flux change
- *float* – Minimum allowed sigma

calc_selfcal_snr(*imagename*, *sigma*, *num_antenna_to_use*)

Function to calculate the SNR of the present iteration for selfcal

Parameters

- **imagename** (*str*) – Name of the image
- **sigma** (*float*) – Present sigma value used for making the image
- **num_antenna_to_use** (*int*) – Number of antenna used for imaging

Returns
SNR per antenna for self-calibration

Return type
float

casaimage_to_wsclean(*casaimage=''*, *wsclean_imagename=''*, *imagetype='image'*, *wsclean_imsize=1024*, *keep_casaimage=True*)

Function to convert CASA image to WSClean image (Stokes modes : 'I' and 'XX,YY')

Parameters

- **casaimage** (*str*) – Name of the CASA image or model
- **wsclean_imagename** (*str*) – WSClean imagename prefix
- **imagetype** (*str*) – Image type, 'image', 'model' or 'residual'
- **wsclean_imsize** (*int*) – WSClean image size, CASA image size should be larger than that
- **keep_casaimage** (*bool*) – Keep CASA image or not

Returns

- *list* – Output WSClean images
- *int* – WSClean image size
- *list* – WSClean image stokes axes

change_start_sigma(*nsigma*, *sigma_step*, *min_sigma*)

Function to calculate start sigma

Parameters

- **nsigma** (*float*) – Present sigma value
- **sigma_step** (*float*) – Sigma reduction step
- **min_sigma** (*float*) – Minimum value of sigma to reduce

Returns
Value of the start sigma

Return type
float

dirty_image(*start_sigma*, *box_width=3*, *antenna_to_use=''*, *weight='briggs'*, *robust=0.5*)

Function make dirty image

Parameters

- **start_sigma** (*float*) – Start sigma

- **box_width** (*float*) – Negative box width around the maximum pixel in degree (default : 3 degree)
- **antenna_to_use** (*list*) – List of antennas
- **weight** (*str*) – Visibility weighting during imaging , 'uniform', 'natural' or 'briggs'. Default is 'briggs'
- **robust** (*float*) – Robust parameter for briggs weighting, default : 0.5

Returns

- *float* – rms based dynamic range
- *float* – negative based dynamic range
- *float* – Per antenna SNR for self-calibration
- *int* – Error message code

file_remover_and_keeper(*num_iter*, *msg_code*, *do_bandpass=False*, *ref_time_chan=True*)

This function keep and remove caltables, ms, imaging related files based on the need

Parameters

- **num_iter** (*int*) – Number of self-calibration iteration
- **msg_code** (*int*) – Selfcal message code
- **do_bandpass** (*bool*) – Performing bandpass or not
- **ref_timechan** (*bool*) – Reference time channel or not

image_source_true_loc(*outdir*, *do_bandpass=False*, *weight='briggs'*, *robust=0.5*)

Function to make quick image for finding source true location with respect to reference time and channel

Parameters

- **Outdir** (*str*) – Output directory
- **do_bandpass** (*bool*) – Source true location for bandpass self-calibration or not
- **weight** (*str*) – Visibility weighting during imaging , 'uniform', 'natural' or 'briggs'. Default is 'briggs'
- **robust** (*float*) – Robust parameter for briggs weighting, default : 0.5

Returns

Output imagename

Return type

str

initial_bpass(*modelname*, *num_iter*, *rms_thresh*, *ref_ant*, *minsnr*, *calmode*, *calibrator_caltable=[]*)

Function to perform and apply bandpass calibration

Parameters

- **modelname** (*str*) – Name of the initial model
- **num_iter** (*int*) – Selfcal iteration number
- **rms_thresh** (*list*) – List of n-sigma value for rms based flagging (Do not go below 6)
- **ref_ant** (*int*) – Reference antenna number
- **calmode** (*str*) – 'p' for phase-only or 'ap' for amplitude-phase calibration

- **calibrator_caltable** (*list*) – List of any previous caltables

Returns
Name of the initial bandpass table

Return type
str

negative_box(*max_pix*, *box_width=3*)

Create a box about the maximum pixel of image to search negative

Parameters

- **max_pix** (*list*) – Maximum pixel [xxmax,yymax]
- **box_width** (*float*) – Box width in degree (default : 3 degree)

Returns
CASA box 'xblc,yblc,xrtc,yrtc'

Return type
str

reduce_sigma(*imagename*, *nsigma*, *sigma_step*, *minsigma*, *pre_residual=0.0*, *residual_frac=0.01*, *stokes_list=['I']*)

Function to determine whether reduce the CLEAN sigma or not

Parameters

- **imagename** (*str*) – Prefix of the image name
- **nsigma** (*float*) – Value of the present n-sigma
- **sigma_step** (*float*) – Step to reduce sigma value
- **minsigma** (*float*) – Minimum allowed sigma
- **pre_residual** (*float*) – Previous residual fraction to compare (default : 0.0)
- **residual_frac** (*float*) – Residual flux fraction to reduce sigma (default : 0.01)
- **stokes_list** (*list*) – Stokes plane list

Returns
Reduced value of n-sigma if residual flux is more than given percentage (default : 1%) of the total flux in Stokes I or in all Stokes Q,U,V.

Return type
float

remove_model_negative(*imagename*, *modelname*, *sigma=10*, *overwrite=False*)

Function to remove negatives from model image

Parameters

- **imagename** (*str*) – Name of the image
- **modelname** (*str*) – Name of the model
- **sigma** (*float*) – Sigma value for thresholding
- **overwrite** (*bool*) – Overwrite the model image or not

Returns
Model image without negatives

Return type
str

selfcal_iteration(*num_iter*, *rms_thresh*, *sigma*, *maskstr*, *antenna_to_use*, *startmodel*, *startmask*, *ref_ant*, *minsnr*, *calmode*, *maskfile=''*, *want_auto_masking=False*, *cpus=5*, *absmem=10*, *wlayers=1*, *stokes='I'*, *interactive=False*, *do_bandpass=False*, *solmode='R'*, *correct_phasecenter=False*, *ra=0*, *dec=0*, *box_width=3*, *calibrator_caltable=[]*, *maskregion=''*, *weight='briggs'*, *robust=0.5*)

Function to perform a self-calibration loop, make an image, put the model in the measurement set, and perform the calibration

Parameters

- **num_iter** (*int*) – Number of self-calibration iteration
- **rms_thresh** (*list*) – List of rms for threshold
- **sigma** (*float*) – Threshold sigma
- **maskstr** (*str*) – CASA mask string for CLEANing (For CASA only)
- **antenna_to_use** (*list*) – List of antennas for CLEANing
- **startmodel** (*str*) – CASA model to start the CLEANing (For both CASA imager and WSClean, for WSClean imager, CASA models will be converted to WSClean models)
- **startmask** (*str*) – CASA mask to start (For CASA only)
- **ref_ant** (*int*) – Reference antenna
- **minsnr** (*float*) – Minimum SNR of gain solutions
- **calmode** (*str*) – 'p' for phase-only and 'ap' for amplitude-phase calibration
- **maskfile** (*str*) – Name of the maskfile
- **want_auto_masking** (*bool*) – If True use CASA auto-multithresh for auto masking
- **cpus** (*int*) – Number of cpu threads to use
- **absmem** (*float*) – Memory in GB to use
- **wlayers** (*int*) – Number of w-stacking layers (For wsclean only)
- **stokes** (*str*) – Stokes plane to image
- **interactive** (*bool*) – Perform interactive selfcal, change options on the fly
- **do_bandpass** (*str*) – Perform bandpass calibration
- **solmode** (*str*) – 'R', 'L1R', 'L1' for gaincal and only 'R' for bandpass
- **correct_phasecenter** (*bool*) – Correct the phase center
- **ra** (*float*) – New RA in degree to change phasecenter
- **dec** (*float*) – New DEC in degree to change phasecenter
- **box_width** (*float*) – Negative box width around the maximum pixel in degree (default : 3 degree)
- **calibrator_caltable** (*list*) – List of calilbrator caltables
- **maskregion** (*str*) – Mask region in case of auto-masking (Only for CASA)
- **weight** (*str*) – Visibility weighting during imaging , 'uniform', 'natural' or 'briggs'. Default is 'briggs'

- **robust** (*float*) – Robust parameter for briggs weighting, default : 0.5

Returns

- *float* – rms based dynamic range
- *float* – Negative based dynamic range
- *int* – Message code

shift_phasecenter(*imagename*, *ra*, *dec*)

Function to shift image reference RA DEC to a certain value

Parameters

- **imagename** (*str*) – Name of the image
- **ra** (*float*) – RA in degree
- **dec** (*float*) – DEC in degree

Returns
Success code (0 or 1)

Return type
int

wsclean_to_casaimage(*wsclean_images=[]*, *casaimage_prefix='CASA_'*, *imagetype='image'*, *keep_wsclean_images=True*)

Function to convert WSClean image to CASA image (Stokes modes : 'I' and 'XX,YY')

Parameters

- **wsclean_images** (*list*) – List of WSClean images
- **casaimage_prefix** (*str*) – Output CASA image name prefix (default = 'CASA_')
- **imagetype** (*str*) – 'image', 'model', 'residual' or 'dirty' (default = 'image')
- **keep_wsclean_images** (*bool*) – Keep the WSClean images or not

Returns
Output CASA imagename

Return type
str

CHAPTER

SEVEN

PAIRCARS SCRIPTS

7.1 paircars scripts

7.1.1 control_paircars

Usage: PAIRCARS master controller for each day calibration

Options: -h, --help show this help message and exit

--basedir=Directory path, Name of base directory for a given day

7.1.2 run_paircars

Usage: Perform self calibration of a single time and frequency slice

Options: -h, --help show this help message and exit

--msname=Measurement Set, Name of measurement set of a single time and frequency slice

--metafits=Metafits file, Name of metafits file of the observation

--basedir=Directory path, Name of the base directory

--workdir=Directory path, Name of the working directory

--ref_freq_avg=Float, Frequency averaging for reference ms

--ref_time_avg=Float, Time averaging for reference ms

--ref_time_freq=Boolean, Reference measurement set or not

--do_bandpass=Boolean, Perform bandpass calibration or not

--do_polcal=Boolean, Perform polarisation calibration or not

--cal_attenuation=Float, Attenuation in dB for calibrator observation

--caltables=String, comma separated, Previous calibration tables

--scratch=Boolean, Start from scratch or not for reference time frequency slice

--wsclean=Boolean, Use WSClean for imaging or not

--cpu_frac=Float, Fraction of cpu to use

7.1.3 run_intensity_selfcal

Usage: Perform intensity self calibration of a single time and frequency slice

Options: -h, --help show this help message and exit

--msname=Measurement Set, Name of measurement set of a single time anf frequen slice

--metafits=Metafits file, Name of metafits file of the observation

--workdir=Directory path, Name of the working directory

--dopoint=Boolean, Want to try with point source model

--verbose=Boolean, Verbose mode

--interactive=Boolean, Interactive mode

--fresh=Boolean, Start fresh self calibration loop

--reduce_flags=Boolean, Try to reduce flag solutions if it is more than 5%

--scratch=Boolean, Start from scratch or not for reference time frequency slice

--caltables=String, comma separated, Previous caltables

--wsclean=Boolean, Use WSClean for imaging or not

7.1.4 run_bandpass_selfcal

Usage: Perform bandpass self calibration

Options: -h, --help show this help message and exit

--msname=Measurement Set, Name of measurement set of a single time and frequency slice

--metafits=Metafits file, Name of metafits file of the observation

--workdir=Directory path, Name of the working directory

--verbose=Boolean, Verbose mode

--interactive=Boolean, Interactive mode

--fresh=Boolean, Start fresh self calibration loop

--caltables=String, comma separated, Previous caltables

--wsclean=Boolean, Use WSClean for imaging or not

7.1.5 run_pol_selfcal

Usage: Perform polarisation self calibration of a single time and frequency slice

Options: -h, --help show this help message and exit

--msname=Measurement Set, Name of measurement set of a single time anf frequency slice

--metafits=Metafits file, Name of metafits file

--workdir=Directory path, Name of the working directory

--verbose=Boolean, Verbose mode

--interactive=Boolean, Interactive mode

--fresh=Boolean, Start fresh self calibration loop

--gaincal=Boolean, Perform gaincal using leakage corrected model (Only do when no calibrator observation is present)

--caltables=String, comma separated, Previous caltables

--wsclean=Boolean, Use WSClean for imaging or not

PYTHON MODULE INDEX

p

INDEX

A

B

C

D

E

F

G

I

M

N

P

R

S

T

U

W