suncasa

Release 1.0.6

EOVSA Team

Apr 13, 2024

CONTENTS:

CHAPTER

ONE

INTRODUCTION

Welcome to the documentation for SUNCASA, a Python library dedicated to processing and analyzing solar radio observational data.

1.1 API Reference

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

1.1.1 suncasa

Subpackages

suncasa.dspec

SunCASA Dspec

isort:skip_file

Subpackages

suncasa.dspec.sources

Submodules

suncasa.dspec.sources.eovsa

Module Contents

Functions

get_dspec(filename[, doplot, vmax, vmin, norm, cmap])	Read EOVSA Dynamic Spectrum FITS file <filename> and return a spectrogram dictionary.

[1] Created with sphinx-autoapi

suncasa.dspec.sources.eovsa.**get_dspec**(*filename*, *doplot=False*, *vmax=None*, *vmin=None*, *norm=None*, *cmap=None*)

Read EOVSA Dynamic Spectrum FITS file <filename> and return a spectrogram dictionary. Optionally show an overview plot if doplot switch is set.

Example:

```
>>> from suncasa.eovsa import eovsa_dspec as ds
>>> from astropy.time import Time
>>> from matplotlib.colors import LogNorm
## Read EOVSA Dynamic Spectrum FITS file <filename>
>>> filename = 'EOVSA_TPall_20170713.fts'
>>> s = ds.get_dspec(filename, doplot=True, cmap='gist_heat', norm=LogNorm(vmax=2.
↪1e3, vmin=40))
## To access the data in the spectrogram object, use
>>> spec = s['spectrogram']                    ## (Array of amplitudes in SFU, of␣
↪size nfreq,ntimes)
>>> fghz = s['spectrum_axis']                  ## (Array of frequencies in GHz, of␣
↪size nfreq)
>>> tim = Time(s['time_axis'], format='mjd')   ## (Array of UT times in astropy.
↪time object, of size ntimes)
```

param filename
type filename
filename of the spectrogram fits file

param doplot
type doplot
Boolean, optional

param vmin
type vmin
scalar, optional

param vmax
type vmax
scalar, optional

param When using scalar data and no explicit norm
param vmin and vmax
param define the data range that the colormap covers. By default

:param : :param the colormap covers the complete value range of the supplied data.: :param vmin: :param vmax are ignored if the norm parameter is used.: :param norm: :type norm: *matplotib.colors Normalization object* or str, optional :param The Normalize instance used to scale scalar data to the [0: :param 1]: :param range before mapping to colors using cmap. By default: :param a linear: :param scaling mapping the lowest value to 0 and the highest to 1 is used.: :param This parameter is ignored for RGB(A) data.: :param cmap: :type cmap: *matplotib.colors.Colormap* or str :param A colormap instance or the name of a registered colormap.:

returns
spectrogram

rtype
dictionary

suncasa.dspec.sources.lwa

Module Contents

Functions

rebin1d(arr, new_len)	
rebin2d(arr, new_shape)	
timestamp_to_mjd(times)	
read_data(filename[, stokes, timerange, freqrange, ...])	filename: name of the OVRO-LWA hdf5 beamforming file;

suncasa.dspec.sources.lwa.**rebin1d**(*arr*, *new_len*)

suncasa.dspec.sources.lwa.**rebin2d**(*arr*, *new_shape*)

suncasa.dspec.sources.lwa.**timestamp_to_mjd**(*times*)

suncasa.dspec.sources.lwa.**read_data**(*filename*, *stokes='I'*, *timerange=[]*, *freqrange=[]*, *timebin=1*, *freqbin=1*, *verbose=True*)

filename: name of the OVRO-LWA hdf5 beamforming file;
This can be a string (single file) or a list of strings (multiple files)

stokes: currently supporting 'XX', 'YY', 'I', 'Q', 'U', 'V', 'IV' timerange: list of [start_time, end_time], start_time and end_time should be recognized by astropy.time.Time

e.g., ['2023-09-22T18:00', '2023-09-22T18:10']

freqrange: list of [start_frequency, end_frequency] in MHz. Example: [23, 82] timebin: number to bin in time freqbin: number to bin in frequency verbose: if True, print extra information

Submodules

suncasa.dspec.dspec

Module Contents

Classes

Dspec	A class to handle dynamic spectra from radio observations.

class suncasa.dspec.dspec.**Dspec**(*fname=None*, *specfile=None*, *bl=''*, *uvrange=''*, *field=''*, *scan=''*, *datacolumn='data'*, *domedian=False*, *timeran=None*, *spw=None*, *timebin='0s'*, *regridfreq=False*, *fillnan=None*, *verbose=False*, *usetbtool=True*, *ds_normalised=False*)

A class to handle dynamic spectra from radio observations.

`data`
: The dynamic spectrum data array.

  Type
  numpy.ndarray

`time_axis`
: Time axis of the dynamic spectrum.

  Type
  astropy.time.Time

`freq_axis`
: Frequency axis of the dynamic spectrum in Hz.

  Type
  numpy.ndarray

`telescope`
: Name of the telescope used for the observation.

  Type
  str

`observatory`
: Name of the observatory.

  Type
  str

`t_label`
: Label for the time axis.

  Type
  str

`f_label`
: Label for the frequency axis.

  Type
  str

`bl`
: Baseline information.

  Type
  str

`uvrange`
: UV range information.

  Type
  str

`pol`
: Polarization information.

  Type
  str

spec_unit

Unit of the dynamic spectrum data ('sfu', 'Jy', or 'K').

Type
str, default 'sfu'

__init__(fname=None, specfile=None, **kwargs):

Initializes the Dspec object by reading a FITS file or a saved numpy array.

read(fname, source=None, *args, **kwargs):

Reads dynamic spectrum data from a file.

tofits(fitsfile=None, specdata=None, **kwargs):

Writes the dynamic spectrum data to a FITS file.

get_dspec(fname=None, **kwargs):

Extracts dynamic spectrum data from a measurement set.

wrt_dspec(specfile=None, specdat=None):

Writes the dynamic spectrum data to a binary file.

rd_dspec(specdata, **kwargs):

Reads dynamic spectrum data from a numpy file.

concat_dspec(specfiles, outfile=None, savespec=False):

Concatenates multiple dynamic spectrum files along the time axis.

peek(*args, **kwargs):

Plots the dynamic spectrum on the current axes.

plot(pol='I', **kwargs):

Plots the dynamic spectrum for a given polarization.

data

time_axis

freq_axis

telescope

observatory

t_label

f_label

bl

uvrange

pol

spec_unit = 'sfu'

read(*fname*, *source=None*, **args*, ***kwargs*)

Reads dynamic spectrum data from a file.

Parameters

- **fname** (*str*) – The file name to read the dynamic spectrum data from.

- **source** (*str, optional*) – Specifies the data source ('fits', 'suncasa', or 'lwa') to determine the appropriate reader.
- **source.** (*Additional parameters are passed to the specific reader function based on the*) –

tofits(*fitsfile=None*, *specdata=None*, *spectype='amp'*, *spec_unit='jy'*, *telescope='EOVSA'*, *observatory='Owens Valley Radio Observatory'*, *observer='EOVSA Team'*)

Writes the dynamic spectrum data to a FITS file.

This method exports the stored dynamic spectrum data into a new FITS file, including relevant metadata and optional input spectrum data.

Parameters

- **fitsfile** (*str, optional*) – Path and name of the output FITS file. If not specified, a default name will be generated.
- **specdata** (*dict, optional*) – Input dictionary containing dynamic spectrum data and metadata. If not provided, the method uses the data stored in the Dspec object.
- **spectype** (*str, optional*) – Specifies the type of the spectrum to be saved ('amp' for amplitude, 'pha' for phase). Default is 'amp'.
- **spec_unit** (*str, optional*) – Specifies the unit of the spectrum data ('jy' for Jansky, 'sfu' for Solar Flux Units, 'k' for Kelvin). Default is 'jy'.
- **telescope** (*str, optional*) – Name of the telescope with which the data was obtained. Default is 'EOVSA'.
- **observatory** (*str, optional*) – Name of the observatory. Default is 'Owens Valley Radio Observatory'.
- **observer** (*str, optional*) – Name of the observer or team that made the observation. Default is 'EOVSA Team'.

Return type
None

Raises

- **FileNotFoundError** – If the specified *fitsfile* path does not exist.
- **ValueError** – If *specdata* is provided but does not contain the required keys.

Examples

```
>>> dspec = Dspec()
>>> dspec.tofits(fitsfile='output.fits', specdata=my_specdata, spec_unit='sfu',␣
↪observer='Sun Observer')
```

Note: The method can directly utilize the dynamic spectrum data stored within the *Dspec* object if *specdata* is not provided. Ensure the *Dspec* object has been properly initialized with dynamic spectrum data before calling this method without *specdata*.

get_dspec(*fname=None*, *specfile=None*, *bl=''*, *uvrange=''*, *field=''*, *scan=''*, *datacolumn='data'*, *domedian=False*, *timeran=None*, *spw=None*, *timebin='0s'*, *regridfreq=False*, *hanning=False*, *applyflag=True*, *fillnan=None*, *verbose=False*, *usetbtool=True*, *ds_normalised=False*)

wrt_dspec(*specfile=None*, *specdat=None*)

rd_dspec(*specdata*, *spectype='amp'*, *spec_unit='jy'*)

concat_dspec(*specfiles*, *outfile=None*, *savespec=False*)

concatenate a list of specfiles in time axis :param specfiles: a list of specfile to concatenate :return: concatenated specdata

peek(**args*, ***kwargs*)

Plot dynamaic spectrum onto current axes.

Parameters

- ***args** (*dict*) –
- ****kwargs** (*dict*) – Any additional plot arguments that should be used when plotting.

Returns
fig – A plot figure.

Return type
~matplotlib.Figure

plot(*pol='I'*, *vmin=None*, *vmax=None*, *norm='log'*, *cmap='viridis'*, *cmap2='viridis'*, *vmin2=None*, *vmax2=None*, *timerange=None*, *freqrange=None*, *ignore_gaps=True*, *freq_unit='GHz'*, *spec_unit=None*, *plot_fast=False*, *percentile=[1, 99]*, *minmaxpercentile=False*, ***kwargs*)

Plots the dynamic spectrum for a given polarization.

This method generates a plot of the dynamic spectrum, allowing for customization of the visualization through various parameters such as color maps, normalization, and value ranges.

Parameters

- **pol** (*str, optional*) – Polarization for plotting. Default is 'I'.
- **vmin** (*float, optional*) – Minimum and maximum intensity values for the color scale. Defaults to None, which auto-scales.
- **vmax** (*float, optional*) – Minimum and maximum intensity values for the color scale. Defaults to None, which auto-scales.
- **norm** (str or *matplotlib.colors.Normalize*, optional) – Normalization of the color scale. Can be 'linear', 'log', or a custom normalization. Default is 'log'.
- **cmap** (*str, optional*) – Matplotlib colormap names or instances for the plot and, optionally, a second polarization. Default is 'viridis'.
- **cmap2** (*str, optional*) – Matplotlib colormap names or instances for the plot and, optionally, a second polarization. Default is 'viridis'.
- **vmin2** (*float, optional*) – Minimum and maximum values for the color scale of the second polarization. Only used if a second polarization is plotted.
- **vmax2** (*float, optional*) – Minimum and maximum values for the color scale of the second polarization. Only used if a second polarization is plotted.
- **timerange** (*list of str, optional*) – Time range to plot, formatted as ['start_time', 'end_time']. Times should be in ISO format. Defaults to None, which plots the entire range.
- **freqrange** (*list of float, optional*) – Frequency range to plot, in units specified by *freq_unit*. Defaults to None, which plots the entire range.

- **ignore_gaps** (*bool, optional*) – If True, ignores gaps in the frequency axis. Default is True.
- **freq_unit** (*str, optional*) – Unit for the frequency axis ('kHz', 'MHz', 'GHz'). Default is 'GHz'.
- **spec_unit** (*str, optional*) – Unit for the spectrum data ('sfu', 'Jy', or 'K'). If not specified, uses the unit from the Dspec object.
- **plot_fast** (*bool, optional*) – If True, uses a faster plotting method which may reduce detail. Default is False.
- **percentile** (*list of float, optional*) – Percentile values to use for auto-scaling the color range. Default is [1, 99].
- **minmaxpercentile** (*bool, optional*) – If True, uses percentile for vmin and vmax. Default is False.

Returns

fig – The matplotlib figure object containing the plot.

Return type

matplotlib.figure.Figure

Examples

```
>>> dspec = Dspec()
>>> fig = dspec.plot(pol='I', vmin=0.1, vmax=5, cmap='hot', freqrange=[1, 2],␣
↪timerange=['2021-01-01T00:00:00', '2021-01-01T01:00:00'])
>>> plt.show()
```

Notes

For dual-polarization plots (e.g., 'RRLL'), *cmap2* along with *vmin2* and *vmax2* can be specified to customize the appearance of the second polarization.

Package Contents

Classes

Dspec	A class to handle dynamic spectra from radio observations.

class suncasa.dspec.**Dspec**(*fname=None*, *specfile=None*, *bl=''*, *uvrange=''*, *field=''*, *scan=''*, *datacolumn='data'*, *domedian=False*, *timeran=None*, *spw=None*, *timebin='0s'*, *regridfreq=False*, *fillnan=None*, *verbose=False*, *usetbtool=True*, *ds_normalised=False*)

A class to handle dynamic spectra from radio observations.

data

The dynamic spectrum data array.

Type
numpy.ndarray

`time_axis`

Time axis of the dynamic spectrum.

Type
astropy.time.Time

`freq_axis`

Frequency axis of the dynamic spectrum in Hz.

Type
numpy.ndarray

`telescope`

Name of the telescope used for the observation.

Type
str

`observatory`

Name of the observatory.

Type
str

`t_label`

Label for the time axis.

Type
str

`f_label`

Label for the frequency axis.

Type
str

`bl`

Baseline information.

Type
str

`uvrange`

UV range information.

Type
str

`pol`

Polarization information.

Type
str

`spec_unit`

Unit of the dynamic spectrum data ('sfu', 'Jy', or 'K').

Type
str, default 'sfu'

__init__(fname=None, specfile=None, **kwargs):

Initializes the Dspec object by reading a FITS file or a saved numpy array.

read(fname, source=None, *args, **kwargs):

Reads dynamic spectrum data from a file.

tofits(fitsfile=None, specdata=None, **kwargs):

Writes the dynamic spectrum data to a FITS file.

get_dspec(fname=None, **kwargs):

Extracts dynamic spectrum data from a measurement set.

wrt_dspec(specfile=None, specdat=None):

Writes the dynamic spectrum data to a binary file.

rd_dspec(specdata, **kwargs):

Reads dynamic spectrum data from a numpy file.

concat_dspec(specfiles, outfile=None, savespec=False):

Concatenates multiple dynamic spectrum files along the time axis.

peek(*args, **kwargs):

Plots the dynamic spectrum on the current axes.

plot(pol='I', **kwargs):

Plots the dynamic spectrum for a given polarization.

data

time_axis

freq_axis

telescope

observatory

t_label

f_label

bl

uvrange

pol

spec_unit = 'sfu'

read(*fname*, *source=None*, **args*, ***kwargs*)

Reads dynamic spectrum data from a file.

Parameters

- **fname** (*str*) – The file name to read the dynamic spectrum data from.
- **source** (*str, optional*) – Specifies the data source ('fits', 'suncasa', or 'lwa') to determine the appropriate reader.
- **source.** (*Additional parameters are passed to the specific reader function based on the*) –

tofits(*fitsfile=None*, *specdata=None*, *spectype='amp'*, *spec_unit='jy'*, *telescope='EOVSA'*, *observatory='Owens Valley Radio Observatory'*, *observer='EOVSA Team'*)

Writes the dynamic spectrum data to a FITS file.

This method exports the stored dynamic spectrum data into a new FITS file, including relevant metadata and optional input spectrum data.

Parameters

- **fitsfile** (*str, optional*) – Path and name of the output FITS file. If not specified, a default name will be generated.
- **specdata** (*dict, optional*) – Input dictionary containing dynamic spectrum data and metadata. If not provided, the method uses the data stored in the Dspec object.
- **spectype** (*str, optional*) – Specifies the type of the spectrum to be saved ('amp' for amplitude, 'pha' for phase). Default is 'amp'.
- **spec_unit** (*str, optional*) – Specifies the unit of the spectrum data ('jy' for Jansky, 'sfu' for Solar Flux Units, 'k' for Kelvin). Default is 'jy'.
- **telescope** (*str, optional*) – Name of the telescope with which the data was obtained. Default is 'EOVSA'.
- **observatory** (*str, optional*) – Name of the observatory. Default is 'Owens Valley Radio Observatory'.
- **observer** (*str, optional*) – Name of the observer or team that made the observation. Default is 'EOVSA Team'.

Return type
None

Raises

- **FileNotFoundError** – If the specified *fitsfile* path does not exist.
- **ValueError** – If *specdata* is provided but does not contain the required keys.

Examples

```
>>> dspec = Dspec()
>>> dspec.tofits(fitsfile='output.fits', specdata=my_specdata, spec_unit='sfu',␣
↪observer='Sun Observer')
```

Note: The method can directly utilize the dynamic spectrum data stored within the *Dspec* object if *specdata* is not provided. Ensure the *Dspec* object has been properly initialized with dynamic spectrum data before calling this method without *specdata*.

get_dspec(*fname=None*, *specfile=None*, *bl=''*, *uvrange=''*, *field=''*, *scan=''*, *datacolumn='data'*, *domedian=False*, *timeran=None*, *spw=None*, *timebin='0s'*, *regridfreq=False*, *hanning=False*, *applyflag=True*, *fillnan=None*, *verbose=False*, *usetbtool=True*, *ds_normalised=False*)

wrt_dspec(*specfile=None*, *specdat=None*)

rd_dspec(*specdata*, *spectype='amp'*, *spec_unit='jy'*)

concat_dspec(*specfiles*, *outfile=None*, *savespec=False*)

concatenate a list of specfiles in time axis :param specfiles: a list of specfile to concatenate :return: concatenated specdata

peek(**args*, ***kwargs*)

Plot dynamaic spectrum onto current axes.

Parameters

- ***args** (*dict*) –
- ****kwargs** (*dict*) – Any additional plot arguments that should be used when plotting.

Returns
fig – A plot figure.

Return type
~matplotlib.Figure

plot(*pol='I'*, *vmin=None*, *vmax=None*, *norm='log'*, *cmap='viridis'*, *cmap2='viridis'*, *vmin2=None*, *vmax2=None*, *timerange=None*, *freqrange=None*, *ignore_gaps=True*, *freq_unit='GHz'*, *spec_unit=None*, *plot_fast=False*, *percentile=[1, 99]*, *minmaxpercentile=False*, ***kwargs*)

Plots the dynamic spectrum for a given polarization.

This method generates a plot of the dynamic spectrum, allowing for customization of the visualization through various parameters such as color maps, normalization, and value ranges.

Parameters

- **pol** (*str, optional*) – Polarization for plotting. Default is 'I'.
- **vmin** (*float, optional*) – Minimum and maximum intensity values for the color scale. Defaults to None, which auto-scales.
- **vmax** (*float, optional*) – Minimum and maximum intensity values for the color scale. Defaults to None, which auto-scales.
- **norm** (str or *matplotlib.colors.Normalize*, optional) – Normalization of the color scale. Can be 'linear', 'log', or a custom normalization. Default is 'log'.
- **cmap** (*str, optional*) – Matplotlib colormap names or instances for the plot and, optionally, a second polarization. Default is 'viridis'.
- **cmap2** (*str, optional*) – Matplotlib colormap names or instances for the plot and, optionally, a second polarization. Default is 'viridis'.
- **vmin2** (*float, optional*) – Minimum and maximum values for the color scale of the second polarization. Only used if a second polarization is plotted.
- **vmax2** (*float, optional*) – Minimum and maximum values for the color scale of the second polarization. Only used if a second polarization is plotted.
- **timerange** (*list of str, optional*) – Time range to plot, formatted as ['start_time', 'end_time']. Times should be in ISO format. Defaults to None, which plots the entire range.
- **freqrange** (*list of float, optional*) – Frequency range to plot, in units specified by *freq_unit*. Defaults to None, which plots the entire range.
- **ignore_gaps** (*bool, optional*) – If True, ignores gaps in the frequency axis. Default is True.
- **freq_unit** (*str, optional*) – Unit for the frequency axis ('kHz', 'MHz', 'GHz'). Default is 'GHz'.

- **spec_unit** (*str, optional*) – Unit for the spectrum data ('sfu', 'Jy', or 'K'). If not specified, uses the unit from the Dspec object.
- **plot_fast** (*bool, optional*) – If True, uses a faster plotting method which may reduce detail. Default is False.
- **percentile** (*list of float, optional*) – Percentile values to use for auto-scaling the color range. Default is [1, 99].
- **minmaxpercentile** (*bool, optional*) – If True, uses percentile for vmin and vmax. Default is False.

Returns
fig – The matplotlib figure object containing the plot.

Return type
matplotlib.figure.Figure

Examples

```
>>> dspec = Dspec()
>>> fig = dspec.plot(pol='I', vmin=0.1, vmax=5, cmap='hot', freqrange=[1, 2],␣
↪timerange=['2021-01-01T00:00:00', '2021-01-01T01:00:00'])
>>> plt.show()
```

Notes

For dual-polarization plots (e.g., 'RRLL'), *cmap2* along with *vmin2* and *vmax2* can be specified to customize the appearance of the second polarization.

suncasa.eovsa

Submodules

suncasa.eovsa.eovsa_IDBfiledownloader

Module Contents

Functions

get_times_from_web(link)	
eovsa_filedownloader(trange[, outpath])	

suncasa.eovsa.eovsa_IDBfiledownloader.**get_times_from_web**(*link*)

suncasa.eovsa.eovsa_IDBfiledownloader.**eovsa_filedownloader**(*trange*, *outpath='./'*)

suncasa.eovsa.eovsa_diskmodel

Module Contents

Functions

ant_trange(vis)	Figure out nominal times for tracking of old EOVSA antennas, and return time
gaussian2d(x, y, amplitude, x0, y0, sigma_x, sigma_y, ...)	
writediskxml(dsize, fdens, freq[, xmlfile])	
readdiskxml(xmlfile)	
image_adddisk(eofile, diskinfo[, edgeconvmode, caltbonly])	**param eofile**
read_ms(vis)	Read a CASA ms file and return a dictionary of amplitude, phase, uvdistance,
im2cl(imname, clname[, convol, verbose])	
fit_diskmodel(out, bidx, rstn_flux[, uvfitrange, ...])	Given the result returned by read_ms(), plots the amplitude vs. uvdistance
fit_vs_freq(out)	
calc_diskmodel(slashdate, nbands, freq, defaultfreq)	
mk_diskmodel([outname, direction, reffreq, flux, ...])	Create a blank solar disk model image (or optionally a data cube)
insertdiskmodel(vis[, sizescale, fdens, dsize, ...])	
disk_slfcal(vis[, slfcaltbdir, active, clearcache, pols])	Starting with the name of a calibrated ms (vis, which must have 'UDByyyymmdd' in the name)
fd_images(vis[, cleanup, niter, spws, imgoutdir, ...])	Create standard full-disk images in "images" subdirectory of the current directory.
feature_slfcal(vis[, niter, uvrange, spws, ...])	Uses images from disk-selfcaled data as model for further self-calibration of outer antennas.
plt_eovsa_image(eofiles[, figoutdir])	
pipeline_run(vis[, outputvis, workdir, slfcaltbdir, ...])	

Attributes

tb
spw2band
defaultfreq

suncasa.eovsa.eovsa_diskmodel.**tb**

suncasa.eovsa.eovsa_diskmodel.**spw2band**

suncasa.eovsa.eovsa_diskmodel.**defaultfreq**

suncasa.eovsa.eovsa_diskmodel.**ant_trange**(*vis*)

Figure out nominal times for tracking of old EOVSA antennas, and return time range in CASA format

suncasa.eovsa.eovsa_diskmodel.**gaussian2d**(*x*, *y*, *amplitude*, *x0*, *y0*, *sigma_x*, *sigma_y*, *theta*)

suncasa.eovsa.eovsa_diskmodel.**writediskxml**(*dsize*, *fdens*, *freq*, *xmlfile='SOLDISK.xml'*)

suncasa.eovsa.eovsa_diskmodel.**readdiskxml**(*xmlfile*)

suncasa.eovsa.eovsa_diskmodel.**image_adddisk**(*eofile*, *diskinfo*, *edgeconvmode='frommergeddisk'*, *caltbonly=False*)

Parameters

- **eofile** –
- **diskxmlfile** –
- **edgeconvmode** – available mode: frommergeddisk,frombeam

Returns

suncasa.eovsa.eovsa_diskmodel.**read_ms**(*vis*)

Read a CASA ms file and return a dictionary of amplitude, phase, uvdistance, uvangle, frequency (GHz) and time (MJD). Currently only returns the XX IF channel. vis Name of the visibility (ms) folder

suncasa.eovsa.eovsa_diskmodel.**im2cl**(*imname*, *clname*, *convol=True*, *verbose=False*)

suncasa.eovsa.eovsa_diskmodel.**fit_diskmodel**(*out*, *bidx*, *rstn_flux*, *uvfitrange=[1, 150]*, *angle_tolerance=np.pi / 2*, *doplot=True*)

Given the result returned by read_ms(), plots the amplitude vs. uvdistance separately for polar and equatorial directions rotated for P-angle, then overplots a disk model for a disk enlarged by eqfac in the equatorial direction, and polfac in the polar direction. Also requires the RSTN flux spectrum for the date of the ms, determined from (example for 2019-09-01):

import rstn frq, flux = rstn.rd_rstnflux(t=Time('2019-09-01')) rstn_flux = rstn.rstn2ant(frq, flux, out['fghz']*1000, t=Time('2019-09-01'))

suncasa.eovsa.eovsa_diskmodel.**fit_vs_freq**(*out*)

suncasa.eovsa.eovsa_diskmodel.**calc_diskmodel**(*slashdate*, *nbands*, *freq*, *defaultfreq*)

suncasa.eovsa.eovsa_diskmodel.**mk_diskmodel**(*outname='disk'*, *direction='J2000 10h00m00.0s 20d00m00.0s'*, *reffreq='2.8GHz'*, *flux=660000.0*, *eqradius='16.166arcmin'*, *polradius='16.166arcmin'*, *pangle='21.1deg'*, *overwrite=True*)

Create a blank solar disk model image (or optionally a data cube) outname String to use for part of the image and fits file names (default 'disk') direction String specifying the position of the Sun in RA and Dec. Default

means use the standard string "J2000 10h00m00.0s 20d00m00.0s"

reffreq The reference frequency to use for the disk model (the frequency at which
the flux level applies). Default is '2.8GHz'.

flux The flux density, in Jy, for the entire disk. Default is 66 sfu. eqradius The equatorial radius of the disk. Default is

16 arcmin + 10" (for typical extension of the radio limb)

polradius The polar radius of the disk. Default is
16 arcmin + 10" (for typical extension of the radio limb)

pangle The solar P-angle (geographic position of the N-pole of the Sun) in
degrees E of N. This only matters if eqradius != polradius

index The spectral index to use at other frequencies. Default None means
use a constant flux density for all frequencies.

cell The cell size (assumed square) to use for the image. The image size
is determined from a standard radius of 960" for the Sun, divided by cell size, increased to nearest power of 512 pixels. The default is '2.0arcsec', which results in an image size of 1024 x 1024.

Note that the frequency increment used is '325MHz', which is the width of EOVSA bands
(not the width of individual science channels)

suncasa.eovsa.eovsa_diskmodel.**insertdiskmodel**(*vis*, *sizescale=1.0*, *fdens=None*, *dsize=None*, *xmlfile='SOLDISK.xml'*, *writediskinfoonly=False*, *active=False*, *overwrite=True*)

suncasa.eovsa.eovsa_diskmodel.**disk_slfcal**(*vis*, *slfcaltbdir='./'*, *active=False*, *clearcache=False*, *pols='XX'*)

Starting with the name of a calibrated ms (vis, which must have 'UDByyyymmdd' in the name) add a model disk based on the solar disk size for that date and perform multiple selfcal adjustments (two phase and one amplitude), and write out a final selfcaled database with the disk subtracted. Returns the name of the final database.

suncasa.eovsa.eovsa_diskmodel.**fd_images**(*vis*, *cleanup=False*, *niter=None*, *spws=['0~1', '2~5', '6~10', '11~20', '21~30', '31~43']*, *imgoutdir='./'*, *bright=None*, *stokes='XX'*)

Create standard full-disk images in "images" subdirectory of the current directory. If cleanup is True, delete those images after completion, leaving only the fits images.

suncasa.eovsa.eovsa_diskmodel.**feature_slfcal**(*vis*, *niter=200*, *uvrange='>1.5Klambda'*, *spws=['0~1', '2~5', '6~10', '11~20', '21~30', '31~49']*, *slfcaltbdir='./'*, *bright=None*, *pols='XX'*)

Uses images from disk-selfcaled data as model for further self-calibration of outer antennas. This is only a good idea if there are bright active regions that provide strong signal on the long baselines.

suncasa.eovsa.eovsa_diskmodel.**plt_eovsa_image**(*eofiles*, *figoutdir='./'*)

suncasa.eovsa.eovsa_diskmodel.**pipeline_run**(*vis*, *outputvis=''*, *workdir=None*, *slfcaltbdir=None*, *imgoutdir=None*, *figoutdir=None*, *clearcache=False*, *pols='XX'*)

suncasa.eovsa.eovsa_dspec

Module Contents

Functions

get_dspec(filename[, doplot, vmax, vmin, norm, cmap])	Reads and optionally plots an EOVSA Dynamic Spectrum from a FITS file.

suncasa.eovsa.eovsa_dspec.**get_dspec**(*filename*, *doplot=False*, *vmax=None*, *vmin=None*, *norm=None*, *cmap=None*)

Reads and optionally plots an EOVSA Dynamic Spectrum from a FITS file.

This function reads a FITS file containing EOVSA dynamic spectrum data, returning the data as a dictionary. If requested, it also generates a plot of the dynamic spectrum.

Parameters

- **filename** (*str*) – Path and name of the EOVSA dynamic spectrum FITS file to read.
- **doplot** (*bool, optional*) – If True, generates a plot of the dynamic spectrum. Default is False.
- **vmin** (*float, optional*) – Minimum value for the color scale. Ignored if *norm* is provided. Default is None, which autoscales.
- **vmax** (*float, optional*) – Maximum value for the color scale. Ignored if *norm* is provided. Default is None, which autoscales.
- **norm** (*matplotlib.colors.Normalize* or None, optional) – Normalization for the color scale of the plot. If None, a linear scaling is used. Default is None.
- **cmap** (str or *matplotlib.colors.Colormap*, optional) – Colormap for the dynamic spectrum plot. Can be a colormap name or an instance. Default is None, which uses the default colormap.

Returns

A dictionary containing the dynamic spectrum data (*spectrogram*), frequency axis (*spectrum_axis* in GHz), and time axis (*time_axis* in modified Julian date).

Return type

dict

Example

```
from suncasa.eovsa import eovsa_dspec as ds
from astropy.time import Time
from matplotlib.colors import LogNorm

# Read EOVSA Dynamic Spectrum FITS file <filename>
filename = 'EOVSA_TPall_20170713.fts'
s = ds.get_dspec(filename, doplot=True, cmap='gist_heat', norm=LogNorm(vmax=2.1e3,␣
↪vmin=40))

# To access the data in the spectrogram object, use
spec = s['spectrogram']  # Array of amplitudes in SFU, of size nfreq,ntimes
fghz = s['spectrum_axis']  # Array of frequencies in GHz, of size nfreq
tim = Time(s['time_axis'], format='mjd')   # Array of UT times in astropy.time␣
↪object, of size ntimes
```

suncasa.eovsa.eovsa_fitsutils

Module Contents

Functions

rewriteImageFits(datestr[, verbose, writejp2, ...])	
main([year, month, day, ndays, overwritejp2, ...])	

Attributes

imgfitsdir	
imgfitsbkdir	
year	

suncasa.eovsa.eovsa_fitsutils.**imgfitsdir = '/data1/eovsa/fits/synoptic/'**

suncasa.eovsa.eovsa_fitsutils.**imgfitsbkdir = '/data1/workdir/synoptic_newbk/'**

suncasa.eovsa.eovsa_fitsutils.**rewriteImageFits**(*datestr*, *verbose=False*, *writejp2=False*, *overwritejp2=False*, *overwritefits=False*)

suncasa.eovsa.eovsa_fitsutils.**main**(*year=None*, *month=None*, *day=None*, *ndays=1*, *overwritejp2=False*, *overwritefits=False*)

suncasa.eovsa.eovsa_fitsutils.**year**

suncasa.eovsa.eovsa_flare_calib

Module Contents

Functions

import_calib_idb(trange[, workdir, ncpu, timebin, width])	Script to import and calibrate IDB data based on an input time range

suncasa.eovsa.eovsa_flare_calib.**import_calib_idb**(*trange*, *workdir=None*, *ncpu=1*, *timebin='0s'*, *width=1*)

Script to import and calibrate IDB data based on an input time range :param trange: Example: Time([‘2022-11-12 17:55:00’, ‘2022-11-12 18:10:00’]) :type trange: [begin_time, end_time], in eovsa.util Time format. :param workdir: :type workdir: specify where the working directory is. Default to current path

Returns
vis_out

Return type
concatenated CASA measurement set with initial gain, amplitude, and phase calibrations applied

suncasa.eovsa.eovsa_flare_pipeline

Module Contents

Classes

FlareSelfCalib

Attributes

start
ia
ms
msmd
tb

suncasa.eovsa.eovsa_flare_pipeline.**start**

suncasa.eovsa.eovsa_flare_pipeline.**ia**

suncasa.eovsa.eovsa_flare_pipeline.**ms**

suncasa.eovsa.eovsa_flare_pipeline.**msmd**

suncasa.eovsa.eovsa_flare_pipeline.**tb**

class suncasa.eovsa.eovsa_flare_pipeline.**FlareSelfCalib**(*vis=None*, *workpath='./'*, *logfile=None*)

property vis

Getting the input visibility for self-calibration

property slfcal_spws

Rteurn the spws chosen for selfcal

vis_info()

static get_img_center_heliocoords(*images*)

Provide a set of images in helioprojective coordinates (at different frequencies), find the peak location :param images: :type images: list of fits image files

Returns
xycen

Return type
solar x and y coordinates, in arcsec

static find_sidelobe_level(*image*)

static check_shift(*image*, *shift*, *cell*)

static grow_mask(*image*, *mask*, *thres*)

static get_spw_num(*visibility*)

static get_descids(*visibility*)

Retrieve actual descids from the DATA_DESCRIPTION table. Unusually, the DATA_DESCRIPTION table may contain duplicated rows featuring the same SPECTRAL_WINDOW_ID, yet pointing to null or dummy data. This can cause a KeyError: 'axis_info' when attempting to read the 'data' variable in calc_cellsize, as the loop iterating over 'i' in ms.selectinit(datadescid=i) within calc_cellsize fails to locate the corresponding data due to its non-existence.

Parameters
visibility –

Returns

static calc_cellsize(*visibility*)

static get_ref_freqlist(*visibility*)

static read_bandpass(*bptable*, *nant=16*)

static combine_groups(*group*, *pos*)

static gen_fof_groups(*data3*, *thres*)

gen_mask(*image1*, *image2*, *mask1*, *mask2*, *threshold*, *imsize*, *s*, *make_shifted_mask=False*, *grow_threshold=0.5*)

We will allow for small shifts and small change of size ere

confirm_maximum_pixel(*imagename*, *mask*, *spwran*, *msname*, *uvrange*, *imsize*, *cell*, *s*)

static restore_previous_condition(*imagename*)

static flag_data_gap(*visibility*, *sp*)

get_img_stat(*imagename*)

flare_finder()

Provide input visibility, find out the flare peak time, flare duration, and suitable time ranges for performing self-calibration

produce_required_inputs_from_flare_time(*num_spws*)

find_previous_image(*spw*)

gen_blank_cal(*spw*)

find_phasecenter()

The purpose of this module is to find a new phasecenter at the flare location for imaging :rtype: Updates self.phasecenter (in J2000 RA and DEC) to be the flare location

do_selfcal(*slfcalms*, *sp*, *spwran*, *uvrange=''*, *cell_val='2arcsec'*, *imsize=2048*, *ref_image=''*, *make_shifted_mask=False*, *combine_spws=False*)

calling_do_selfcal(*slfcalms*, *s*, *uvrange=''*, *cell_val='2arcsec'*)

slfcal_init()

flare_ms_calib(*value*)

slfcal_pipeline(*doselfcal=True*, *doimaging=False*)

rename_move_files(*flare_id*, *fitsdir_web_tp*, *movdir_web_tp*, *dorename_fits=False*, *domove_fits=False*, *dorename_mov=False*, *domove_mov=False*, *dormworkdir=False*, *docopy=False*)

Rename EOVSA FITS files and move them to the web folder. 'eovsa.lev1_mbd_12s.2022-11-12T180524Z.image.fits' 'eovsa.lev1_mbd_12s.flare_id_20221112180524.mp4'

suncasa.eovsa.eovsa_pipeline

Module Contents

Classes

Path_config

Functions

getspwfreq(vis)	**param vis**
trange2ms([trange, doimport, verbose, doscaling, ...])	This finds all solar UDBms files within a timerange; If the UDBms file does not exist
calib_pipeline(trange[, workdir, doimport, overwrite, ...])	trange: can be 1) a single Time() object: use the entire day
mk_qlook_image(trange[, doimport, docalib, ncpu, ...])	trange: can be 1) a single Time() object: use the entire day
plt_qlook_image(imres[, figdir, verbose, synoptic])	
qlook_image_pipeline(date[, twidth, ncpu, doimport, ...])	date: date string or Time object. e.g., '2017-07-15' or Time('2017-07-15')
pipeline([year, month, day, ndays, clearcache, ...])	Main pipeline for importing and calibrating EOVSA visibility data.

Attributes

tasks
split
tclean
gencal
clearcal
applycal
gaincal
delmod
tools
qatool
iatool
mstool
tbtool
ms
tb
hostname
pathconfig
udbmsdir
udbmsscldir
udbmsslfcaleddir
udbdir
caltbdir
slfcaltbdir
qlookfitsdir
qlookfigdir
synopticfigdir
parser

suncasa.eovsa.eovsa_pipeline.**tasks**

suncasa.eovsa.eovsa_pipeline.**split**

suncasa.eovsa.eovsa_pipeline.**tclean**

suncasa.eovsa.eovsa_pipeline.**gencal**

suncasa.eovsa.eovsa_pipeline.**clearcal**

suncasa.eovsa.eovsa_pipeline.**applycal**

suncasa.eovsa.eovsa_pipeline.**gaincal**

suncasa.eovsa.eovsa_pipeline.**delmod**

suncasa.eovsa.eovsa_pipeline.**tools**

suncasa.eovsa.eovsa_pipeline.**qatool**

suncasa.eovsa.eovsa_pipeline.**iatool**

suncasa.eovsa.eovsa_pipeline.**mstool**

suncasa.eovsa.eovsa_pipeline.**tbtool**

suncasa.eovsa.eovsa_pipeline.**ms**

suncasa.eovsa.eovsa_pipeline.**tb**

suncasa.eovsa.eovsa_pipeline.**hostname**

class suncasa.eovsa.eovsa_pipeline.**Path_config**

_get_env_var(*env_var*, *default_path*)

suncasa.eovsa.eovsa_pipeline.**pathconfig**

suncasa.eovsa.eovsa_pipeline.**udbmsdir**

suncasa.eovsa.eovsa_pipeline.**udbmsscldir**

suncasa.eovsa.eovsa_pipeline.**udbmsslfcaleddir**

suncasa.eovsa.eovsa_pipeline.**udbdir**

suncasa.eovsa.eovsa_pipeline.**caltbdir**

suncasa.eovsa.eovsa_pipeline.**slfcaltbdir**

suncasa.eovsa.eovsa_pipeline.**qlookfitsdir**

suncasa.eovsa.eovsa_pipeline.**qlookfigdir**

suncasa.eovsa.eovsa_pipeline.**synopticfigdir**

suncasa.eovsa.eovsa_pipeline.**getspwfreq**(*vis*)

Parameters
vis –

Returns
mid frequencies in GHz of each spw in the vis

suncasa.eovsa.eovsa_pipeline.**trange2ms**(*trange=None*, *doimport=False*, *verbose=False*, *doscaling=False*, *overwrite=True*)

This finds all solar UDBms files within a timerange; If the UDBms file does not exist in EOVSAUDBMSSCL, create one by calling importeovsa Required inputs: trange - can be 1) a single string or Time() object in UTC: use the entire day, e.g., '2017-08-01' or Time('2017-08-01')

if just a date, find all scans withing the same date in local time. if a complete time stamp, find the local date first (which may be different from that provided,

and return all scans within that day

2) a range of Time(), e.g., Time(['2017-08-01 00:00','2017-08-01 23:00'])

3) None – use current date Time.now()

doimport - Boolean. If true, call importeovsa to import UDB files that are missing from those found in the directory specified in EOVSAUDBMSSCL. Otherwise, return a list of ms files it has found.

doscaling - Boolean. If true, scale cross-correlation amplitudes by using auto-correlations verbose - Boolean. If true, return more information

suncasa.eovsa.eovsa_pipeline.**calib_pipeline**(*trange*, *workdir=None*, *doimport=False*, *overwrite=False*, *clearcache=False*, *verbose=False*, *pols='XX'*, *version='v1.0'*, *ncpu='auto'*)

trange: can be 1) a single Time() object: use the entire day

2) a range of Time(), e.g., Time(['2017-08-01 00:00','2017-08-01 23:00'])

3) a single or a list of UDBms file(s)

4) None – use current date Time.now()

suncasa.eovsa.eovsa_pipeline.**mk_qlook_image**(*trange*, *doimport=False*, *docalib=False*, *ncpu=10*, *twidth=12*, *stokes=None*, *antenna='0~12'*, *lowcutoff_freq=3.7*, *imagedir=None*, *spws=['1~5', '6~10', '11~15', '16~25']*, *toTb=True*, *overwrite=True*, *doslfcal=False*, *verbose=False*)

trange: can be 1) a single Time() object: use the entire day

2) a range of Time(), e.g., Time(['2017-08-01 00:00','2017-08-01 23:00'])

3) a single or a list of UDBms file(s)

4) None – use current date Time.now()

suncasa.eovsa.eovsa_pipeline.**plt_qlook_image**(*imres*, *figdir=None*, *verbose=True*, *synoptic=False*)

suncasa.eovsa.eovsa_pipeline.**qlook_image_pipeline**(*date*, *twidth=10*, *ncpu=15*, *doimport=False*, *docalib=False*, *synoptic=False*, *overwrite=True*)

date: date string or Time object. e.g., '2017-07-15' or Time('2017-07-15')

suncasa.eovsa.eovsa_pipeline.**pipeline**(*year=None*, *month=None*, *day=None*, *ndays=1*, *clearcache=True*, *overwrite=False*, *doimport=True*, *pols='XX'*, *version='v1.0'*, *ncpu='auto'*, *debugging=False*)

Main pipeline for importing and calibrating EOVSA visibility data.

Name:

eovsa_pipeline — main pipeline for importing and calibrating EOVSA visibility data.

Synopsis:
eovsa_pipeline.py [options]... [DATE_IN_YY_MM_DD]

Description:
Import and calibrate EOVSA visibility data of the date specified by DATE_IN_YY_MM_DD (or from ndays before the DATE_IN_YY_MM_DD if option –ndays/-n is provided). If DATE_IN_YY_MM_DD is omitted, it will be set to 2 days before now by default. There are no mandatory arguments in this command.

Parameters

- **year** (*int, optional*) – The year for which data should be processed, defaults to None.
- **month** (*int, optional*) – The month for which data should be processed, defaults to None.
- **day** (*int, optional*) – The day for which data should be processed, defaults to None.
- **ndays** (*int, optional*) – Number of days before the specified date to include in the processing, defaults to 1.
- **clearcache** (*bool, optional*) – Whether to clear cache after processing, defaults to True.
- **overwrite** (*bool, optional*) – Whether to overwrite existing files, defaults to True.
- **doimport** (*bool, optional*) – Whether to perform the import step, defaults to True.
- **pols** (*str, optional*) – Polarizations to process, can be 'XX' or 'XXYY', defaults to 'XX'.
- **version** (*str, optional*) – Version of the pipeline to use, choices are 'v1.0' or 'v2.0', defaults to 'v1.0'.
- **ncpu** (*str, optional*) – Number of CPUs to use for processing, defaults to 'auto'.
- **debugging** (*bool, optional*) – Whether to run the pipeline in debugging mode, defaults to False.

Raises
ValueError – Raises an exception if the date parameters are out of the valid Gregorian calendar range.

Example:

To process data for November 24th, 2021 using version 2.0 of the pipeline, with all options enabled:

```
>>> python eovsa_pipeline.py --date 2021-11-24T20:00 --clearcache --overwrite --
↪doimport --pols XXYY --version v2.0 --ndays 2
```

If you want to see the help message, you can run:

```
>>> python eovsa_pipeline.py -h
```

suncasa.eovsa.eovsa_pipeline.**parser**

suncasa.eovsa.eovsa_pipelineAlldayFits

Module Contents

suncasa.eovsa.eovsa_pipelineAlldayFits.**argv**

suncasa.eovsa.eovsa_pltQlookImage

Module Contents

Functions

clearImage()	
pltEmptyImage2([dpis_dict])	
pltEmptyImage(datestr, spws, vmaxs, vmins[, dpis_dict])	
pltEovsaQlookImage(datestr, spws, vmaxs, vmins, dpis_dict)	
pltSdoQlookImage(datestr, dpis_dict[, fig, ax, ...])	
pltBbsoQlookImage(datestr, dpis_dict[, fig, ax, ...])	
main([year, month, day, ndays, clearcache, ...])	

Attributes

imgfitsdir	
imgfitstmpdir	
pltfigdir	
year	

suncasa.eovsa.eovsa_pltQlookImage.**imgfitsdir = '/data1/eovsa/fits/synoptic/'**

suncasa.eovsa.eovsa_pltQlookImage.**imgfitstmpdir = '/data1/workdir/fitstmp/'**

suncasa.eovsa.eovsa_pltQlookImage.**pltfigdir = '/common/webplots/SynopticImg/eovsamedia/eovsa-browser/'**

suncasa.eovsa.eovsa_pltQlookImage.**clearImage**()

suncasa.eovsa.eovsa_pltQlookImage.**pltEmptyImage2**(*dpis_dict={'t': 32.0}*)

suncasa.eovsa.eovsa_pltQlookImage.**pltEmptyImage**(*datestr*, *spws*, *vmaxs*, *vmins*, *dpis_dict={'t': 32.0}*)

suncasa.eovsa.eovsa_pltQlookImage.**pltEovsaQlookImage**(*datestr*, *spws*, *vmaxs*, *vmins*, *dpis_dict*, *fig=None*, *ax=None*, *overwrite=False*, *verbose=False*)

suncasa.eovsa.eovsa_pltQlookImage.**pltSdoQlookImage**(*datestr*, *dpis_dict*, *fig=None*, *ax=None*, *overwrite=False*, *verbose=False*, *clearcache=False*)

suncasa.eovsa.eovsa_pltQlookImage.**pltBbsoQlookImage**(*datestr*, *dpis_dict*, *fig=None*, *ax=None*, *overwrite=False*, *verbose=False*, *clearcache=False*)

suncasa.eovsa.eovsa_pltQlookImage.**main**(*year=None*, *month=None*, *day=None*, *ndays=1*, *clearcache=False*, *ovwrite_eovsa=False*, *ovwrite_sdo=False*, *ovwrite_bbso=False*, *show_warning=False*)

suncasa.eovsa.eovsa_pltQlookImage.**year**

suncasa.eovsa.eovsa_pltQlookMovie

Module Contents

Functions

clearImage()	
pltEovsaQlookImageSeries(timobjs, spws, vmaxs, vmins, ...)	
main(year, month[, day, ndays, bd, show_warning])	By default, the subroutine create EOVSA monthly movie

Attributes

imgfitsdir
imgfitstmpdir
pltfigdir
year

suncasa.eovsa.eovsa_pltQlookMovie.**imgfitsdir = '/data1/eovsa/fits/synoptic/'**

suncasa.eovsa.eovsa_pltQlookMovie.**imgfitstmpdir = '/data1/workdir/fitstmp/'**

suncasa.eovsa.eovsa_pltQlookMovie.**pltfigdir** = '/common/webplots/SynopticImg/eovsamedia/eovsa-browser/'

suncasa.eovsa.eovsa_pltQlookMovie.**clearImage**()

suncasa.eovsa.eovsa_pltQlookMovie.**pltEovsaQlookImageSeries**(*timobjs*, *spws*, *vmaxs*, *vmins*, *aiawave*, *bd*, *fig=None*, *axs=None*, *imgoutdir=None*, *overwrite=False*, *verbose=False*)

suncasa.eovsa.eovsa_pltQlookMovie.**main**(*year*, *month*, *day=None*, *ndays=10*, *bd=3*, *show_warning=False*)

By default, the subroutine create EOVSA monthly movie

suncasa.eovsa.eovsa_pltQlookMovie.**year**

suncasa.eovsa.eovsa_readfits

Module Contents

Functions

readfits(eofile)	read eovsa image fits, adjust the date-obs to the mid time.
get_all_coordinate_from_map(sunmap)	

suncasa.eovsa.eovsa_readfits.**readfits**(*eofile*)

read eovsa image fits, adjust the date-obs to the mid time. :param eofile: :return:

suncasa.eovsa.eovsa_readfits.**get_all_coordinate_from_map**(*sunmap*)

suncasa.eovsa.eovsa_scaling

Module Contents

Functions

mk_udbms([trange, outpath, projid, srcid, doscaling])	usage: outfiles = mk_udbms(Time('2017-07-02 15:00'))

suncasa.eovsa.eovsa_scaling.**mk_udbms**(*trange=None*, *outpath=None*, *projid='NormalObserving'*, *srcid='Sun'*, *doscaling=True*)

usage: outfiles = mk_udbms(Time('2017-07-02 15:00'))

Parameters

- **trange** –
- **outpath** –
- **projid** –

- **srcid** –
- **doscaling** –

Returns

suncasa.eovsa.eovsa_synoptic_imaging_pipeline

Module Contents

Classes

FrequencySetup	Manages frequency setup based on observation date for radio astronomy imaging.

Functions

log_print(level, message)	
is_factor_of_60_minutes(tdt)	Check if tdt is a factor of 60 minutes or a harmonic of 60 minutes.
generate_trange_series(tbg, ted, tdt[, snap_to_full_hour])	Generate a list of time ranges using pandas.date_range, with options to snap the ranges to full hours and
trange2timerange(trange)	Convert a time range tuple in datetime format to a string representation.
rotateimage(data, xc_centre, yc_centre, p_angle)	Rotate an image around a specified point (xc_centre, yc_centre) by a given angle.
sunpymap2helioimage(sunmap, out_image)	Rotate a SunPy map from helioprojective to RA-DEC coordinates and write it to a CASA image format.
solar_diff_rot_image(in_map, newtime, out_image[, showplt])	Reproject a SunPy map to account for solar differential rotation to a new observation time.
get_bmsize(cfreq[, refbmsize, reffreq, minbmsize])	Calculate the beam size at given frequencies based on a reference beam size at a reference frequency.
calc_diskmodel(slashdate, nbands, freq, defaultfreq)	
writediskxml(dsize, fdens, freq[, xmlfile])	
readdiskxml(xmlfile)	
gaussian2d(x, y, amplitude, x0, y0, sigma_x, sigma_y, ...)	
image_adddisk(eofile, diskinfo[, edgeconvmode, ...])	**param eofile** input image FITS file
mk_diskmodel([outname, direction, reffreq, flux, ...])	Create a blank solar disk model image (or optionally a data cube)
insertdiskmodel(vis[, sizescale, fdens, dsize, ...])	

continues on next page

Table 1 – continued from previous page

uvrange_uplim_from_freq(x, x0, x1, y0, y1)	Calculate the upper limit of the UV range from frequency using linear interpolation.
disk_slfcal(msfile, tbg, ted, disk_params[, workdir, ...])	Perform disk self-calibration on measurement set data.
shift_corr(mmsfiles, trange_series, spws, image-model, ...)	Corrects smearing effects in solar observation data by aligning them with a model image.
split_mms(msname, timerange_series[, spw, workdir, ...])	Splits a measurement set into multiple subsets based on specified time ranges.
all_paths_exist(paths)	
ant_trange(vis)	Figure out nominal times for tracking of old EOVSA antennas, and return time
format_spw(spw)	Formats the spectral window (spw) string for file naming, ensuring start and end values are separated by a dash and zero-padded to two digits.
rm_imname_extensions(imname[, keep_ext, verbose])	Remove directories and files that match the base image name (*imname*) with specific extensions.
check_image_zeros(imname)	Check if the image contains non-zero data.
format_param(param)	
run_tclean_automasking(vis, sp, trange, uvrange, ...)	Wrapper function for the tclean task in CASA.
fd_images(vis[, cleanup, image_marker, timerange, ...])	Generates full-disk images, optionally cleans up interim images, and performs image registration.
merge_FITSfiles(fitsfilesin, outfits[, ...])	Merges multiple FITS files into a single output file by calculating the mean of stacked data.
process_time_block(tidx_ted_tbg, msfile_in, msname, ...)	
process_imaging_timerange(tbg_ted, msfile_in, spws, ...)	
pipeline_run(vis[, outputvis, workdir, slfcaltbdir, ...])	Executes the EOVSA data processing pipeline for solar observation data.

Attributes

hostname
is_on_server
tasks
gaincal
applycal
clearcal
delmod
ft
uvsub
split
concat
flagmanager
flagdata
tclean
hanningsmooth
imhead
tools
qatool
iatool
cltool
mstool
tbtool
qa
ia
ms
tb
description

suncasa.eovsa.eovsa_synoptic_imaging_pipeline.**hostname**

suncasa.eovsa.eovsa_synoptic_imaging_pipeline.**is_on_server = True**

suncasa.eovsa.eovsa_synoptic_imaging_pipeline.**tasks**

suncasa.eovsa.eovsa_synoptic_imaging_pipeline.**gaincal**

suncasa.eovsa.eovsa_synoptic_imaging_pipeline.**applycal**

suncasa.eovsa.eovsa_synoptic_imaging_pipeline.**clearcal**

suncasa.eovsa.eovsa_synoptic_imaging_pipeline.**delmod**

suncasa.eovsa.eovsa_synoptic_imaging_pipeline.**ft**

suncasa.eovsa.eovsa_synoptic_imaging_pipeline.**uvsub**

suncasa.eovsa.eovsa_synoptic_imaging_pipeline.**split**

suncasa.eovsa.eovsa_synoptic_imaging_pipeline.**concat**

suncasa.eovsa.eovsa_synoptic_imaging_pipeline.**flagmanager**

suncasa.eovsa.eovsa_synoptic_imaging_pipeline.**flagdata**

suncasa.eovsa.eovsa_synoptic_imaging_pipeline.**tclean**

suncasa.eovsa.eovsa_synoptic_imaging_pipeline.**hanningsmooth**

suncasa.eovsa.eovsa_synoptic_imaging_pipeline.**imhead**

suncasa.eovsa.eovsa_synoptic_imaging_pipeline.**tools**

suncasa.eovsa.eovsa_synoptic_imaging_pipeline.**qatool**

suncasa.eovsa.eovsa_synoptic_imaging_pipeline.**iatool**

suncasa.eovsa.eovsa_synoptic_imaging_pipeline.**cltool**

suncasa.eovsa.eovsa_synoptic_imaging_pipeline.**mstool**

suncasa.eovsa.eovsa_synoptic_imaging_pipeline.**tbtool**

suncasa.eovsa.eovsa_synoptic_imaging_pipeline.**qa**

suncasa.eovsa.eovsa_synoptic_imaging_pipeline.**ia**

suncasa.eovsa.eovsa_synoptic_imaging_pipeline.**ms**

suncasa.eovsa.eovsa_synoptic_imaging_pipeline.**tb**

suncasa.eovsa.eovsa_synoptic_imaging_pipeline.**log_print**(*level*, *message*)

suncasa.eovsa.eovsa_synoptic_imaging_pipeline.**is_factor_of_60_minutes**(*tdt*)

Check if tdt is a factor of 60 minutes or a harmonic of 60 minutes.

Parameters

tdt (*timedelta*) – Time duration to check.

Returns

True if tdt is a factor or harmonic of 60 minutes, False otherwise.

Return type
bool

suncasa.eovsa.eovsa_synoptic_imaging_pipeline.**generate_trange_series**(*tbg*, *ted*, *tdt*, *snap_to_full_hour=False*)

Generate a list of time ranges using pandas.date_range, with options to snap the ranges to full hours and adjust the first and last time range based on specific conditions.

Parameters

- **tbg** (*str or datetime-like*) – The start time.
- **ted** (*str or datetime-like*) – The end time.
- **tdt** (*timedelta*) – The duration of each time range.
- **snap_to_full_hour** (*bool*) – Whether to snap the time series to full hours, defaults to False.

Returns
A list of tuples, each representing the start and end time of a range.

Return type
list of tuple

suncasa.eovsa.eovsa_synoptic_imaging_pipeline.**trange2timerange**(*trange*)

Convert a time range tuple in datetime format to a string representation.

Parameters
trange (*tuple*) – A tuple containing start and end times as datetime objects.

Returns
A string representation of the time range.

Return type
str

suncasa.eovsa.eovsa_synoptic_imaging_pipeline.**rotateimage**(*data*, *xc_centre*, *yc_centre*, *p_angle*)

Rotate an image around a specified point (xc_centre, yc_centre) by a given angle.

Parameters

- **data** (*numpy.ndarray*) – The image data.
- **xc_centre** (*int*) – The x-coordinate of the rotation center.
- **yc_centre** (*int*) – The y-coordinate of the rotation center.
- **p_angle** (*float*) – The rotation angle in degrees.

Returns
The rotated image.

Return type
numpy.ndarray

suncasa.eovsa.eovsa_synoptic_imaging_pipeline.**sunpymap2helioimage**(*sunmap*, *out_image*)

Rotate a SunPy map from helioprojective to RA-DEC coordinates and write it to a CASA image format.

Parameters

- **sunmap** (*sunpy.map.Map*) – The input SunPy Map object to be rotated.
- **out_image** (*str*) – The filepath for the output CASA image.

Returns
The filepath to the output CASA image format.

Return type
str

suncasa.eovsa.eovsa_synoptic_imaging_pipeline.**solar_diff_rot_image**(*in_map*, *newtime*, *out_image*, *showplt=False*)

Reproject a SunPy map to account for solar differential rotation to a new observation time.

Parameters

- **in_map** (*sunpy.map.Map*) – The input SunPy Map object to be reprojected.
- **newtime** (*astropy.time.Time*) – The new time to which the map is reprojected.
- **out_image** (*str*) – The path for the output image file in CASA format.
- **showplt** (*bool*) – Boolean flag to show plots of the original and reprojected maps, defaults to False.

Returns
The path to the output CASA image format.

Return type
str

suncasa.eovsa.eovsa_synoptic_imaging_pipeline.**get_bmsize**(*cfreq*, *refbmsize=70.0*, *reffreq=1.0*, *minbmsize=4.0*)

Calculate the beam size at given frequencies based on a reference beam size at a reference frequency. This function supports both single frequency values and lists of frequencies.

Parameters

- **cfreq** (*float or list*) – Input frequencies in GHz, can be a float or a list of floats.
- **refbmsize** (*float, optional*) – Reference beam size in arcsec, defaults to 70.0.
- **reffreq** (*float, optional*) – Reference frequency in GHz, defaults to 1.0.
- **minbmsize** (*float, optional*) – Minimum beam size in arcsec, defaults to 4.0.

Returns
Beam size at the given frequencies, same type as input (float or numpy array).

Return type
float or numpy.ndarray

suncasa.eovsa.eovsa_synoptic_imaging_pipeline.**calc_diskmodel**(*slashdate*, *nbands*, *freq*, *defaultfreq*)

suncasa.eovsa.eovsa_synoptic_imaging_pipeline.**writediskxml**(*dsize*, *fdens*, *freq*, *xmlfile='SOLDISK.xml'*)

suncasa.eovsa.eovsa_synoptic_imaging_pipeline.**readdiskxml**(*xmlfile*)

suncasa.eovsa.eovsa_synoptic_imaging_pipeline.**gaussian2d**(*x*, *y*, *amplitude*, *x0*, *y0*, *sigma_x*, *sigma_y*, *theta*)

suncasa.eovsa.eovsa_synoptic_imaging_pipeline.**image_adddisk**(*eofile*, *diskinfo*, *edgeconvmode='frommergeddisk'*, *caltbonly=False*, *bmfactor=2.0*, *overwrite=True*)

Parameters

- **eofile** – input image FITS file
- **diskinfo** – disk information file
- **edgeconvmode** – edge convolve mode, 'frommergeddisk' or 'frombeam'
- **caltbonly** – calculate the Tb of the disk and return the value
- **bmfactor** – factor to multiply the beam major and minor axes to get the sigma of the Gaussian kernel
- **overwrite** – whether to overwrite the output fits file if it already exists

Returns

suncasa.eovsa.eovsa_synoptic_imaging_pipeline.**mk_diskmodel**(*outname='disk'*, *direction='J2000 10h00m00.0s 20d00m00.0s'*, *reffreq='2.8GHz'*, *flux=660000.0*, *eqradius='16.166arcmin'*, *polradius='16.166arcmin'*, *pangle='21.1deg'*, *overwrite=True*)

Create a blank solar disk model image (or optionally a data cube) outname String to use for part of the image and fits file names (default 'disk') direction String specifying the position of the Sun in RA and Dec. Default

means use the standard string "J2000 10h00m00.0s 20d00m00.0s"

reffreq The reference frequency to use for the disk model (the frequency at which
the flux level applies). Default is '2.8GHz'.

flux The flux density, in Jy, for the entire disk. Default is 66 sfu. eqradius The equatorial radius of the disk. Default is

16 arcmin + 10" (for typical extension of the radio limb)

polradius The polar radius of the disk. Default is
16 arcmin + 10" (for typical extension of the radio limb)

pangle The solar P-angle (geographic position of the N-pole of the Sun) in
degrees E of N. This only matters if eqradius != polradius

index The spectral index to use at other frequencies. Default None means
use a constant flux density for all frequencies.

cell The cell size (assumed square) to use for the image. The image size
is determined from a standard radius of 960" for the Sun, divided by cell size, increased to nearest power of 512 pixels. The default is '2.0arcsec', which results in an image size of 1024 x 1024.

Note that the frequency increment used is '325MHz', which is the width of EOVSA bands
(not the width of individual science channels)

suncasa.eovsa.eovsa_synoptic_imaging_pipeline.**insertdiskmodel**(*vis*, *sizescale=1.0*, *fdens=None*, *dsize=None*, *xmlfile='SOLDISK.xml'*, *writediskinfoonly=False*, *active=False*, *overwrite=False*)

suncasa.eovsa.eovsa_synoptic_imaging_pipeline.**uvrange_uplim_from_freq**(*x*, *x0*, *x1*, *y0*, *y1*)

Calculate the upper limit of the UV range from frequency using linear interpolation.

Parameters

- **x** (*float*) – The frequency for which the UV range upper limit is calculated.
- **x0** (*float*) – The start frequency of the interpolation range.
- **x1** (*float*) – The end frequency of the interpolation range.
- **y0** (*float*) – The UV range upper limit at the start frequency.
- **y1** (*float*) – The UV range upper limit at the end frequency.

Returns
The calculated UV range upper limit for the given frequency.

Return type
float

suncasa.eovsa.eovsa_synoptic_imaging_pipeline.**disk_slfcal**(*msfile*, *tbg*, *ted*, *disk_params*, *workdir='./'*, *overwrite=True*, *iterbands=False*)

Perform disk self-calibration on measurement set data.

Parameters

- **msfile** (*str*) – The measurement set file to be calibrated.
- **tbg** (datetime) – The beginning time of the calibration range.
- **ted** (datetime) – The end time of the calibration range.
- **disk_params** (*dict*) – A dictionary containing disk model parameters.
- **iterbands** (*bool*) – Boolean flag to run gaincal iterating over frequency bands, defaults to False. iterbands = True is only useful when uvrange_uplim is used, which is currently not implemented.

Returns
The filepath of the self-calibrated measurement set.

Return type
str

suncasa.eovsa.eovsa_synoptic_imaging_pipeline.**shift_corr**(*mmsfiles*, *trange_series*, *spws*, *imagemodel*, *imagemodel_fits*, *reftime_master*, *workdir='./'*, *pols='XX'*, *overwrite=False*, *do_featureslfcal=False*, *do_diskslfcal=True*, *disk_params={}*, *do_sbdcal=True*, *verbose=True*)

Corrects smearing effects in solar observation data by aligning them with a model image.

Parameters

- **mmsfiles** (*list of str*) – Paths to the measurement set files to be corrected.
- **trange_series** (*list of tuple*) – Each tuple contains start and end times for the measurement set files.
- **spws** (*list of string*) – Spectral window indices to be considered.
- **imagemodel** (*str*) – Path to the model image in CASA measurement set format.

- **imagemodel_fits** (*str*) – Path to the model image in FITS format.
- **reftime_master** (*astropy.time.Time*) – Reference time for the rotation of the model image.
- **workdir** (*str*) – Working directory for output files, defaults to './'.
- **do_featureslfcal** (*bool*) – Boolean flag to perform feature self-calibration, defaults to False.
- **pols** (*str*) – Polarization types to be considered, defaults to 'XX'.
- **overwrite** (*bool*) – Boolean flag to overwrite existing files, defaults to False.
- **do_diskslfcal** (*bool*) – Boolean flag to perform disk self-calibration, defaults to True.
- **disk_params** (*dict*) – Dictionary containing parameters for disk self-calibration.
- **do_sbdcal** (*bool*) – Boolean flag to perform single-band delay calibration, defaults to True.
- **verbose** (*bool*) – Boolean flag to enable verbose output, defaults to True.

Returns
Paths to the corrected measurement set files.

Return type
list of str

suncasa.eovsa.eovsa_synoptic_imaging_pipeline.**split_mms**(*msname*, *timerange_series*, *spw=''*, *workdir='./'*, *overwrite=False*, *verbose=True*)

Splits a measurement set into multiple subsets based on specified time ranges.

Parameters

- **msname** (*str*) – Path to the original measurement set.
- **timerange_series** (*list of tuple*) – Time ranges for each split, with each tuple containing start and end datetime objects.
- **spw** (*str*) – Spectral window specification for the splitting process, defaults to ''.
- **workdir** (*str*) – Target directory for saving the split measurement sets, defaults to './'.
- **overwrite** (*bool*) – Boolean flag to overwrite existing files, defaults to False.
- **verbose** (*bool*) – Boolean flag to enable verbose output, defaults to True.

Returns
Paths to the created split measurement sets, corresponding to each specified time range.

Return type
list of str

suncasa.eovsa.eovsa_synoptic_imaging_pipeline.**all_paths_exist**(*paths*)

suncasa.eovsa.eovsa_synoptic_imaging_pipeline.**ant_trange**(*vis*)

Figure out nominal times for tracking of old EOVSA antennas, and return time range in CASA format

suncasa.eovsa.eovsa_synoptic_imaging_pipeline.**format_spw**(*spw*)

Formats the spectral window (spw) string for file naming, ensuring start and end values are separated by a dash and zero-padded to two digits.

Parameters
spw (*str*) – The spectral window in the format "start~end", where start and end are integers.

Returns
A formatted string with start and end values zero-padded to two digits, separated by a dash.

Return type
str

suncasa.eovsa.eovsa_synoptic_imaging_pipeline.**rm_imname_extensions**(*imname*, *keep_ext=[]*, *verbose=False*)

Remove directories and files that match the base image name (*imname*) with specific extensions.

Parameters

- **imname** (*str*) – The base name of images/directories to remove specific extensions for.
- **keep_ext** (*list*) – A list of extensions to keep. Default is an empty list.
- **verbose** (*bool*) – If True, print detailed information about the removal process. Default is False.

suncasa.eovsa.eovsa_synoptic_imaging_pipeline.**check_image_zeros**(*imname*)

Check if the image contains non-zero data.

suncasa.eovsa.eovsa_synoptic_imaging_pipeline.**format_param**(*param*)

suncasa.eovsa.eovsa_synoptic_imaging_pipeline.**run_tclean_automasking**(*vis*, *sp*, *trange*, *uvrange*, *datacolumn*, *imname*, *imsize*, *cell*, *stokes*, *scales*, *niter*, *reffreq*, *pbcor*, *savemodel*, *usemask*, *restoringbeam*, *sidelobethreshold*, *noisethreshold*, *lownoisethreshold*, *minbeamfrac*, *negativethreshold*, *growiterations*)

Wrapper function for the tclean task in CASA.

Parameters

- **vis** (*str*) – Name of the visibility data set to be imaged.
- **sp** (*str*) – Spectral window selection.
- **trange** (*str*) – Time range selection.
- **uvrange** (*str*) – UV range selection.
- **datacolumn** (*str*) – Specifies which data column to use.
- **imname** (*str*) – Name of the output image.
- **imsize** (*list*) – Image size in pixels.
- **cell** (*list*) – Cell size specification.
- **stokes** (*str*) – Stokes parameters to image.
- **scales** (*list*) – Scales for multi-scale clean.
- **niter** (*int*) – Maximum number of iterations.
- **reffreq** (*float*) – Reference frequency for the image.

- **pbcor** (*bool*) – Specifies whether to perform primary beam correction.
- **savemodel** (*str*) – Specifies whether to save the model.
- **usemask** (*bool*) – Specifies whether to use auto-masking.
- **restoringbeam** (*list*) – Restoring beam parameters.
- **sidelobethreshold** (*float*) – Side lobe threshold for auto-masking.
- **noisethreshold** (*float*) – Noise threshold for auto-masking.
- **lownoisethreshold** (*float*) – Low noise threshold for auto-masking.
- **negativethreshold** (*float*) – Negative threshold for auto-masking.
- **growiterations** (*int*) – Number of grow iterations for auto-masking.

class suncasa.eovsa.eovsa_synoptic_imaging_pipeline.**FrequencySetup**(*tim=None*)

Manages frequency setup based on observation date for radio astronomy imaging.

This class is initialized with an observation time and calculates essential frequency parameters such as effective observing frequencies (eofreq) and spectral windows (spws) based on the observation date. It provides methods to calculate reference frequency and bandwidth for given spectral windows.

Parameters

tim (*astropy.time.Time*) – Observation time used to determine the frequency setup.

Attributes: - tim (astropy.time.Time): The observation time. - spw2band (numpy.ndarray): An array mapping spectral window indices to band numbers. - bandwidth (float): The bandwidth in GHz. - defaultfreq (numpy.ndarray): The default effective observing frequencies in GHz. - nbands (int): Number of bands. - eofreq (numpy.ndarray): Effective observing frequencies based on the observation time. - spws (list of str): Spectral window selections based on the observation time.

Example

```
>>> from astropy.time import Time
>>> tim = Time('2022-01-01T00:00:00', format='isot')
>>> freq_setup = FrequencySetup(tim)
>>> crval, cdelt = freq_setup.get_reffreq_and_cdelt('5~10')
>>> print(crval, cdelt)
```

get_reffreq_and_cdelt(*spw*)

Calculates the reference frequency (CRVAL) and the frequency delta (CDELT) for a given spectral window range.

This method takes a spectral window selection and computes the mean of the effective observing frequencies (eofreq) within that range as the reference frequency. It also calculates the bandwidth covered by the spectral window range as the frequency delta.

Parameters

spw (*str*) – Spectral window selection, specified as a range 'start~end' or a single value.

Returns

A tuple containing the reference frequency and frequency delta, both in GHz.

Return type

(str, str)

Example

```
>>> crval, cdelt = freq_setup.get_reffreq_and_cdelt('5~10')
>>> print(crval, cdelt)
```

suncasa.eovsa.eovsa_synoptic_imaging_pipeline.**fd_images**(*vis*, *cleanup=False*, *image_marker=''*, *timerange=''*, *niter=None*, *cell=['2.5arcsec']*, *imsize=[1024]*, *spws=['0~1', '2~4', '5~10', '11~20', '21~30', '31~43']*, *imgoutdir='./'*, *bright=None*, *stokes='XX'*, *uvrange=''*, *toTb=True*, *pbcor=True*, *datacolumn='data'*, *savemodel='none'*, *usemask='auto-multithresh'*, *overwrite=False*, *compress=False*, *dryrun=False*)

Generates full-disk images, optionally cleans up interim images, and performs image registration.

This function creates full-disk solar images based on visibility data, performs an optional cleanup of interim images, and aligns the resulting images to a standard solar disk model. It supports dynamic image size, cell size, and spectral window (SPW) selection. The function also handles the creation of FITS files from the generated images, ensuring the reference frequency in the FITS header is calculated as the middle of the selected frequency range.

Parameters

- **vis** (*str*) – Path to the visibility data.
- **cleanup** (*bool*) – If True, deletes interim images after processing. Default is False.
- **image_marker** (*str*) – Additional identifier for the image name. Default is ‘’.
- **timerange** (*str*) – Range of time to select from data. Default is ‘’.
- **niter** (*int*) – Number of iterations for the cleaning algorithm. If None, defaults to 5000.
- **cell** (*list*) – Size of the image cell, e.g., [‘2.5arcsec’].
- **imsize** (*list*) – Dimensions of the output image, e.g., [1024].
- **spws** (*list*) – Spectral windows to process, specified as ranges, e.g., [‘0~1’, ‘2~5’].
- **imgoutdir** (*str*) – Output directory for the resulting images and FITS files. Defaults to ‘./’.
- **bright** (*list*) – Flags to indicate brightness processing per SPW. Defaults to all True.
- **stokes** (*str*) – Stokes parameter to use. Default is “XX”.
- **uvrange** (*str*) – UV range to select from data. Default is ‘’.
- **toTb** (*bool*) – If True, converts image to temperature scale. Default is True.
- **pbcor** (*bool*) – If True, applies primary beam correction. Default is True.
- **datacolumn** (*str*) – Data column to use from the visibility data. Default is ‘data’.
- **savemodel** (*str*) – Options for saving the model visibilities. Choices are ‘none’, ‘virtual’, and ‘modelcolumn’. Default is ‘none’.
- **overwrite** (*bool*) – If True, overwrites existing FITS files. Default is False.
- **dryrun** (*bool*) – If True, only returns the paths to the generated images. No actual image processing is performed. Default is False.

Returns

A tuple containing two lists: paths to the generated FITS files and paths to the CASA image files.

Return type

(list of str, list of str)

Note: 1. The reference frequency in the FITS header is calculated as the middle of the selected frequency range. This ensures accurate representation of the frequency information in the generated images. 2. The function internally manages directory creation for images, applies multiscale cleaning based on initial beam size determination, and uses default or specified scales for image cleaning. Errors in beam size determination lead to fallback on default scales. The function finally aligns and converts the images to FITS format, with options for temperature scale conversion and phase center alignment.

suncasa.eovsa.eovsa_synoptic_imaging_pipeline.**merge_FITSfiles**(*fitsfilesin*, *outfits*, *exptime_weight=False*, *suppress_ondiskres=False*, *suppress_thrshd=0.3*, *overwrite=True*)

Merges multiple FITS files into a single output file by calculating the mean of stacked data.

This function stacks the data from a list of input FITS files along a new axis, optionally applying exposure time weighting and suppression of on-disk residuals based on a threshold relative to the disk brightness temperature. The mean of the stacked data is then written to a specified output FITS file.

Parameters

- **fitsfilesin** (*list of str*) – List of file paths to the input FITS files to be merged.
- **outfits** (*str*) – File path for the output FITS file where the merged data will be saved.
- **exptime_weight** (*bool, optional*) – If True, the exposure time is used as a weight for calculating the mean of the stacked data. Defaults to False.
- **suppress_ondiskres** (*bool, optional*) – If True, suppresses on-disk residuals based on *suppress_thrshd*. Defaults to False.
- **suppress_thrshd** (*float, optional*) – Threshold for suppression of on-disk residuals, expressed as a fraction of the disk brightness temperature. May suppress weak but real emission. Defaults to 0.3.
- **overwrite** (*bool, optional*) – If True, the output file is overwritten if it already exists. Defaults to True.

Raises

ValueError – If any of the input FITS files cannot be read.

Return type

None

Notes

The suppression of on-disk residuals is achieved by applying a sigmoid function to pixels within a specified range of the disk brightness temperature, effectively reducing the contribution of residuals while preserving the overall structure.

Examples

Merge three FITS files without exposure time weighting and with on-disk residual suppression:

```
>>> merge_FITSfiles(['sun_01.fits', 'sun_02.fits', 'sun_03.fits'], 'sun_merged.fits
↪',
...                 exptime_weight=False, suppress_ondiskres=True, suppress_
↪thrshd=0.3, overwrite=True)
```

suncasa.eovsa.eovsa_synoptic_imaging_pipeline.**process_time_block**(*tidx_ted_tbg*, *msfile_in*, *msname*, *subdir*, *total_blocks*, *tdt*, *tdtstr*, *spws*, *niter_init*, *reftime_master*, *do_diskslfcal*, *disk_params*, *pols='XX'*, *do_sbdcal=False*, *overwrite=False*)

suncasa.eovsa.eovsa_synoptic_imaging_pipeline.**process_imaging_timerange**(*tbg_ted*, *msfile_in*, *spws*, *subdir*, *overwrite*)

suncasa.eovsa.eovsa_synoptic_imaging_pipeline.**pipeline_run**(*vis*, *outputvis=''*, *workdir=None*, *slfcaltbdir=None*, *imgoutdir=None*, *figoutdir=None*, *clearcache=False*, *pols='XX'*, *mergeFITSonly=False*, *verbose=True*, *do_diskslfcal=True*, *overwrite=False*, *niter_init=200*, *ncpu='auto'*, *tr_series_imaging=None*, *spws_imaging=None*, *hanning=False*, *do_sbdcal=False*)

Executes the EOVSA data processing pipeline for solar observation data.

Parameters

- **vis** (*str*) – Path to the input measurement set (MS) data.
- **outputvis** (*str, optional*) – Output path for the processed MS, defaults to an empty string.
- **workdir** (*str, optional*) – Working directory for intermediate data, defaults to None which sets it to '/data1/workdir'.
- **slfcaltbdir** (*str, optional*) – Directory for storing calibration tables, defaults to None.
- **imgoutdir** (*str, optional*) – Output directory for image files, defaults to None.
- **figoutdir** (*str, optional*) – Output directory for figures, defaults to None.
- **clearcache** (*bool, optional*) – If True, clears the cache after processing, defaults to False.
- **pols** (*str, optional*) – Polarization types to process, defaults to 'XX'.
- **mergeFITSonly** (*bool, optional*) – If True, skips processing and only merges FITS files, defaults to False.
- **verbose** (*bool, optional*) – Enables verbose output during processing, defaults to True.
- **do_diskslfcal** (*bool, optional*) – If True, performs disk self-calibration, defaults to True.

- **overwrite** (*bool, optional*) – If True, overwrites existing files, defaults to False.
- **niter_init** (*int, optional*) – Initial number of iterations for imaging, defaults to 200.
- **ncpu** (*str or int, optional*) – Specifies the number of CPUs for parallel processing, defaults to 'auto'.
- **tr_series_imaging** (*list of tuple, optional*) – Time ranges for imaging, defaults to None.
- **spws_imaging** (*list of str, optional*) – Spectral windows selected for imaging, defaults to None.
- **hanning** (*bool, optional*) – If True, applies Hanning smoothing to the data, defaults to False.
- **do_sbdcal** (*bool*) – Boolean flag to perform single-band delay calibration, defaults to False.

Returns
Path to the processed visibility data.

Return type
str

Example

if you want to specify the spectral windows for imaging >>> spws_imaging = ['5~10', '11~20', '21~30'] ## if you want to specify the time range for imaging >>> from datetime import datetime, timedelta >>> from suncasa.eovsa import eovsa_synoptic_imaging_pipeline as esip >>> tbg_imaging = datetime(2024, 4, 8, 16, 58, 0) >>> ted_imaging = datetime(2024, 4, 8, 19, 00, 0) >>> tdt_imaging = timedelta(minutes=2) >>> tr_series_imaging = esip.generate_trange_series(tbg_imaging, ted_imaging, tdt_imaging)

suncasa.eovsa.eovsa_synoptic_imaging_pipeline.**description = 'this code is trying to address the issue of smearing effect in all-day synthesis images....'**

suncasa.eovsa.impteovsa

Module Contents

Functions

jd2mjds([tjd])	
bl_list2([nant])	Returns a two-dimensional array bl2ord that will translate
get_band([sfreq, sdf, date])	
creatms(idbfile, outpath[, timebin, width])	

Attributes

tools
smtool
metool
me
c_external

suncasa.eovsa.impteovsa.**tools**

suncasa.eovsa.impteovsa.**smtool**

suncasa.eovsa.impteovsa.**metool**

suncasa.eovsa.impteovsa.**me**

suncasa.eovsa.impteovsa.**c_external = False**

suncasa.eovsa.impteovsa.**jd2mjds**(*tjd=None*)

suncasa.eovsa.impteovsa.**bl_list2**(*nant=16*)

Returns a two-dimensional array bl2ord that will translate a pair of antenna indexes (antenna number - 1) to the ordinal number of the baseline in the 'x' key. Note bl2ord(i,j) = bl2ord(j,i), and bl2ord(i,i) = -1.

suncasa.eovsa.impteovsa.**get_band**(*sfreq=None*, *sdf=None*, *date=None*)

suncasa.eovsa.impteovsa.**creatms**(*idbfile*, *outpath*, *timebin=None*, *width=None*)

suncasa.eovsa.msUtils

Module Contents

Functions

getAntennaPosition(vis)	
getObservatoryName(ms)	Returns the observatory name in the specified ms, using the tb tool.
buildConfigurationFile([vis, cfgfile])	

Attributes

tools
tbtool
tb

suncasa.eovsa.msUtils.**tools**

suncasa.eovsa.msUtils.**tbtool**

suncasa.eovsa.msUtils.**tb**

suncasa.eovsa.msUtils.**getAntennaPosition**(*vis*)

suncasa.eovsa.msUtils.**getObservatoryName**(*ms*)

Returns the observatory name in the specified ms, using the tb tool. – Todd Hunter

suncasa.eovsa.msUtils.**buildConfigurationFile**(*vis=''*, *cfgfile=None*)

suncasa.io

Submodules

suncasa.io.ndfits

Module Contents

Functions

is_compressed_fits(fitsfile)	Function to check if the FITS file contains compressed data
headerfix(header[, PC_coor])	this code fixes the header problem of fits out from CASA 5.4+ which leads to a streched solar image.
headerparse(header)	get axis index of polarization
headersqueeze(header, data)	Squeezes single-dimensional entries from an n-dimensional FITS image data array and updates the FITS header accordingly.
get_bdinfo(freq, bw)	get band information from center frequencies and band widths.
read(filepath[, hdus, verbose])	Read a fits file.
write(fname, data, header[, mask, fix_invalid, ...])	Take a data header pair and write a compressed FITS file.
header_to_xml(header)	
write_j2000_image(fname, data, header)	
wrap(fitsfiles[, outfitsfile, docompress, mask, ...])	wrap single frequency fits files into a multiple frequencies fits file
update(fitsfile[, new_data, new_columns, ...])	Updates a FITS file by optionally replacing its primary or compressed image data, adding new columns to the

Attributes

stokesval

suncasa.io.ndfits.**stokesval**

suncasa.io.ndfits.**is_compressed_fits**(*fitsfile*)

Function to check if the FITS file contains compressed data

suncasa.io.ndfits.**headerfix**(*header*, *PC_coor=True*)

this code fixes the header problem of fits out from CASA 5.4+ which leads to a streched solar image.

Setting PC_coor equal to True will reset the rotation matrix.

suncasa.io.ndfits.**headerparse**(*header*)

get axis index of polarization

suncasa.io.ndfits.**headersqueeze**(*header*, *data*)

Squeezes single-dimensional entries from an n-dimensional FITS image data array and updates the FITS header accordingly.

This function is useful for preparing image data for astropy fits compression, which only supports 1D, 2D, or 3D images. It removes any single-dimensional entries from the shape of the data array and updates the corresponding FITS header keys to reflect the new dimensions.

Parameters

- **header** (*astropy.io.fits.Header*) – FITS header object containing the metadata of the image.
- **data** (*numpy.ndarray*) – n-dimensional image data array.

Returns
A tuple of the updated header object and the squeezed data array.

Return type
(astropy.io.fits.Header, numpy.ndarray)

Note: This function only updates the header keys related to dimensions, coordinate types, values, increments, reference pixels, and units. Any specific header keys related to coordinate transformations (e.g., PC matrix) for dimensions higher than the third are also updated if necessary. The function does not handle higher-order WCS transformations beyond simple axis permutations and squeezes.

---

suncasa.io.ndfits.**get_bdinfo**(*freq*, *bw*)

get band information from center frequencies and band widths.

Parameters

- **freq** (*array_like*) – an array of the center frequencies of all frequency bands in Hz
- **bw** (*array_like*) – an array of the band widths of all frequency bands in Hz

Returns
fbounds – A dict of band information

Return type
dict

suncasa.io.ndfits.**read**(*filepath*, *hdus=None*, *verbose=False*, ***kwargs*)

Read a fits file.

Parameters

- **filepath** (*str*) – The fits file to be read.
- **hdus** (*int* or iterable) – The HDU indexes to read from the file.
- **verbose** (*bool*) – if verbose

Returns
pairs – A list of (data, header) tuples

Return type
list

Notes

This routine reads all the HDU's in a fits file and returns a list of the data and a FileHeader instance for each one.

Also all comments in the original file are concatenated into a single "comment" key in the returned FileHeader.

suncasa.io.ndfits.**write**(*fname*, *data*, *header*, *mask=None*, *fix_invalid=True*, *filled_value=0.0*, *overwrite=True*, ***kwargs*)

Take a data header pair and write a compressed FITS file. Caveat: only 1D, 2D, or 3D images are currently supported by Astropy fits compression. To be compressed, the image data array (n-dimensional) must have at least n-3 single-dimensional entries.

Parameters

- **fname** (*str*) – File name, with extension.
- **data** (*numpy.ndarray*) – n-dimensional data array.
- **header** (*dict*) – A header dictionary.
- **compression_type** (*str*, optional) – Compression algorithm: one of 'RICE_1', 'RICE_ONE', 'PLIO_1', 'GZIP_1', 'GZIP_2', 'HCOMPRESS_1'
- **hcomp_scale** (*float*, optional) – HCOMPRESS scale parameter

suncasa.io.ndfits.**header_to_xml**(*header*)

suncasa.io.ndfits.**write_j2000_image**(*fname*, *data*, *header*)

suncasa.io.ndfits.**wrap**(*fitsfiles*, *outfitsfile=None*, *docompress=False*, *mask=None*, *fix_invalid=True*, *filled_value=0.0*, *observatory=None*, *imres=None*, *verbose=False*, ***kwargs*)

wrap single frequency fits files into a multiple frequencies fits file

suncasa.io.ndfits.**update**(*fitsfile*, *new_data=None*, *new_columns=None*, *new_header_entries=None*)

Updates a FITS file by optionally replacing its primary or compressed image data, adding new columns to the first binary table (BinTableHDU), and/or updating header entries in the first image HDU (PrimaryHDU for uncompressed or CompImageHDU for compressed FITS files).

Parameters: - fitsfile (str): Path to the FITS file to be updated. - new_data (np.ndarray, optional): New data array to replace the existing data in the first image HDU.

Defaults to None, which means the data will not be updated.

- new_columns (list of astropy.io.fits.Column, optional): New columns to be added to the first BinTableHDU. Defaults to None, which means no columns will be added.
- new_header_entries (dict, optional): Header entries to update or add in the first image HDU. Each key-value pair represents a header keyword and its new value. Defaults to None, which means no header updates will be made.

Returns: - bool: True if any of the specified updates were successfully applied, False otherwise.

The function determines whether the FITS file is compressed to properly handle the image HDU type. It attempts to update the image HDU's data, the BinTableHDU's columns, and the image HDU's header based on the provided arguments. If all input parameters are None, indicating no updates are specified, the function will print a message and return False.

suncasa.suncasatasks

Subpackages

suncasa.suncasatasks.private

Submodules

suncasa.suncasatasks.private.task_calibeovsa

Module Contents

Functions

calibeovsa([vis, caltype, caltbdir, interp, docalib, ...])	**param vis** EOVSA visibility dataset(s) to be calibrated

Attributes

tasks
split
tclean
gencal
clearcal
applycal
flagdata
casalog
bandpass
tools
tbtool
mstool
qatool
iatool
tb
ms
qa
ia

suncasa.suncasatasks.private.task_calibeovsa.**tasks**

suncasa.suncasatasks.private.task_calibeovsa.**split**

suncasa.suncasatasks.private.task_calibeovsa.**tclean**

suncasa.suncasatasks.private.task_calibeovsa.**gencal**

suncasa.suncasatasks.private.task_calibeovsa.**clearcal**

suncasa.suncasatasks.private.task_calibeovsa.**applycal**

suncasa.suncasatasks.private.task_calibeovsa.**flagdata**

suncasa.suncasatasks.private.task_calibeovsa.**casalog**

suncasa.suncasatasks.private.task_calibeovsa.**bandpass**

suncasa.suncasatasks.private.task_calibeovsa.**tools**

suncasa.suncasatasks.private.task_calibeovsa.**tbtool**

suncasa.suncasatasks.private.task_calibeovsa.**mstool**

suncasa.suncasatasks.private.task_calibeovsa.**qatool**

suncasa.suncasatasks.private.task_calibeovsa.**iatool**

suncasa.suncasatasks.private.task_calibeovsa.**tb**

suncasa.suncasatasks.private.task_calibeovsa.**ms**

suncasa.suncasatasks.private.task_calibeovsa.**qa**

suncasa.suncasatasks.private.task_calibeovsa.**ia**

suncasa.suncasatasks.private.task_calibeovsa.**calibeovsa**(*vis=None*, *caltype=None*, *caltbdir=''*, *interp=None*, *docalib=True*, *doflag=True*, *flagant='13~15'*, *doimage=False*, *imagedir=None*, *antenna=None*, *timerange=None*, *spw=None*, *stokes=None*, *dosplit=False*, *outputvis=None*, *doconcat=False*, *concatvis=None*, *keep_orig_ms=True*)

Parameters

- **vis** – EOVSA visibility dataset(s) to be calibrated
- **caltype** –
- **interp** –
- **docalib** –
- **qlookimage** –
- **flagant** –
- **stokes** –
- **doconcat** –

Returns

suncasa.suncasatasks.private.task_concateovsa

Module Contents

Functions

concateovsa(vis, concatvis[, datacolumn, ...])	

Attributes

tools	
tbtool	
tb	

suncasa.suncasatasks.private.task_concateovsa.**tools**

suncasa.suncasatasks.private.task_concateovsa.**tbtool**

suncasa.suncasatasks.private.task_concateovsa.**tb**

suncasa.suncasatasks.private.task_concateovsa.**concateovsa**(*vis*, *concatvis*, *datacolumn='corrected'*, *keep_orig_ms=True*, *cols2rm='model,corrected'*, *freqtol=''*, *dirtol=''*, *respectname=False*, *timesort=True*, *copypointing=True*, *visweightscale=[]*, *forcesingleephemfield=''*)

suncasa.suncasatasks.private.task_importeovsa

Module Contents

Functions

udb_corr_external(filelist, udbcorr_path[, ...])	
trange2filelist([trange, verbose])	This finds all solar IDB files within a timerange;
importeovsa_iter(filelist, timebin, width, visprefix, ...)	
importeovsa([idbfiles, ncpu, timebin, width, ...])	

Attributes

py3
tasks
split
casalog
tools
tbtool
mstool
qatool
iatool
tb
ms
qa
ia
c_external

suncasa.suncasatasks.private.task_importeovsa.**py3**

suncasa.suncasatasks.private.task_importeovsa.**tasks**

suncasa.suncasatasks.private.task_importeovsa.**split**

suncasa.suncasatasks.private.task_importeovsa.**casalog**

suncasa.suncasatasks.private.task_importeovsa.**tools**

suncasa.suncasatasks.private.task_importeovsa.**tbtool**

suncasa.suncasatasks.private.task_importeovsa.**mstool**

suncasa.suncasatasks.private.task_importeovsa.**qatool**

suncasa.suncasatasks.private.task_importeovsa.**iatool**

suncasa.suncasatasks.private.task_importeovsa.**tb**

suncasa.suncasatasks.private.task_importeovsa.**ms**

suncasa.suncasatasks.private.task_importeovsa.**qa**

suncasa.suncasatasks.private.task_importeovsa.**ia**

suncasa.suncasatasks.private.task_importeovsa.**c_external = False**

suncasa.suncasatasks.private.task_importeovsa.**udb_corr_external**(*filelist*, *udbcorr_path*, *use_exist_udbcorr=False*)

suncasa.suncasatasks.private.task_importeovsa.**trange2filelist**(*trange=[]*, *verbose=False*)

This finds all solar IDB files within a timerange; Required inputs: trange - can be 1) a single string or Time() object in UTC: use the entire day, e.g., '2017-08-01' or Time('2017-08-01')

if just a date, find all scans withing the same date in local time. if a complete time stamp, find the local date first (which may be different from that provided,

and return all scans within that day

2) a range of Time(), e.g., Time(['2017-08-01 00:00','2017-08-01 23:00'])

3) None – use current date Time.now()

suncasa.suncasatasks.private.task_importeovsa.**importeovsa_iter**(*filelist*, *timebin*, *width*, *visprefix*, *nocreatms*, *modelms*, *doscaling*, *keep_nsclms*, *fileidx*)

suncasa.suncasatasks.private.task_importeovsa.**importeovsa**(*idbfiles=None*, *ncpu=None*, *timebin=None*, *width=None*, *visprefix=None*, *udb_corr=True*, *nocreatms=None*, *doconcat=None*, *modelms=None*, *doscaling=False*, *keep_nsclms=False*, *use_exist_udbcorr=False*)

suncasa.suncasatasks.private.task_pimfit

Module Contents

Functions

imfit_iter(imgfiles, doreg, tims, msinfofile, ephem, ...)	
pimfit(imagefiles, ncpu, doreg, timestamps, ...)	

Attributes

tasks
split
tclean
gencal
clearcal
applycal
flagdata
casalog
bandpass
tools
iatool
rgtool
myia
myrg

suncasa.suncasatasks.private.task_pimfit.**tasks**

suncasa.suncasatasks.private.task_pimfit.**split**

suncasa.suncasatasks.private.task_pimfit.**tclean**

suncasa.suncasatasks.private.task_pimfit.**gencal**

suncasa.suncasatasks.private.task_pimfit.**clearcal**

suncasa.suncasatasks.private.task_pimfit.**applycal**

suncasa.suncasatasks.private.task_pimfit.**flagdata**

suncasa.suncasatasks.private.task_pimfit.**casalog**

suncasa.suncasatasks.private.task_pimfit.**bandpass**

suncasa.suncasatasks.private.task_pimfit.**tools**

suncasa.suncasatasks.private.task_pimfit.**iatool**

suncasa.suncasatasks.private.task_pimfit.**rgtool**

suncasa.suncasatasks.private.task_pimfit.**myia**

suncasa.suncasatasks.private.task_pimfit.**myrg**

suncasa.suncasatasks.private.task_pimfit.**imfit_iter**(*imgfiles*, *doreg*, *tims*, *msinfofile*, *ephem*, *box*, *region*, *chans*, *stokes*, *mask*, *includepix*, *excludepix*, *residual*, *model*, *estimates*, *logfile*, *append*, *newestimates*, *complist*, *overwrite*, *dooff* , *offset*, *fixoffset*, *stretch*, *rms*, *noisefwhm*, *summary*, *imidx*)

suncasa.suncasatasks.private.task_pimfit.**pimfit**(*imagefiles*, *ncpu*, *doreg*, *timestamps*, *msinfofile*, *ephemfile*, *box*, *region*, *chans*, *stokes*, *mask*, *includepix*, *excludepix*, *residual*, *model*, *estimates*, *logfile*, *append*, *newestimates*, *complist*, *overwrite*, *dooff* , *offset*, *fixoffset*, *stretch*, *rms*, *noisefwhm*, *summary*)

suncasa.suncasatasks.private.task_pmaxfit

Module Contents

Functions

maxfit_iter(imgfiles, box, width, imidx)	
pmaxfit(imagefiles, ncpu, box, width)	

Attributes

tasks	
casalog	
tools	
iatool	
rgtool	

suncasa.suncasatasks.private.task_pmaxfit.**tasks**

suncasa.suncasatasks.private.task_pmaxfit.**casalog**

suncasa.suncasatasks.private.task_pmaxfit.**tools**

suncasa.suncasatasks.private.task_pmaxfit.**iatool**

suncasa.suncasatasks.private.task_pmaxfit.**rgtool**

suncasa.suncasatasks.private.task_pmaxfit.**maxfit_iter**(*imgfiles*, *box*, *width*, *imidx*)

suncasa.suncasatasks.private.task_pmaxfit.**pmaxfit**(*imagefiles*, *ncpu*, *box*, *width*)

suncasa.suncasatasks.private.task_ptclean

Module Contents

Functions

clean_iter(tim, vis, imageprefix, imagesuffix, twidth, ...)	
ptclean(vis, imageprefix, imagesuffix, ncpu, twidth, ...)	

Attributes

tasks
split
tclean
casalog
tools
tbtool
mstool
qatool
tb
ms
qa
c_external

suncasa.suncasatasks.private.task_ptclean.**tasks**

suncasa.suncasatasks.private.task_ptclean.**split**

suncasa.suncasatasks.private.task_ptclean.**tclean**

suncasa.suncasatasks.private.task_ptclean.**casalog**

suncasa.suncasatasks.private.task_ptclean.**tools**

suncasa.suncasatasks.private.task_ptclean.**tbtool**

suncasa.suncasatasks.private.task_ptclean.**mstool**

suncasa.suncasatasks.private.task_ptclean.**qatool**

suncasa.suncasatasks.private.task_ptclean.**tb**

suncasa.suncasatasks.private.task_ptclean.**ms**

suncasa.suncasatasks.private.task_ptclean.**qa**

suncasa.suncasatasks.private.task_ptclean.**c_external = False**

suncasa.suncasatasks.private.task_ptclean.**clean_iter**(*tim*, *vis*, *imageprefix*, *imagesuffix*, *twidth*, *doreg*, *docompress*, *usephacenter*, *reftime*, *ephem*, *msinfo*, *toTb*, *sclfactor*, *overwrite*, *selectdata*, *field*, *spw*, *uvrange*, *antenna*, *scan*, *observation*, *intent*, *datacolumn*, *imsize*, *cell*, *phasecenter*, *stokes*, *projection*, *startmodel*, *specmode*, *reffreq*, *nchan*, *start*, *width*, *outframe*, *veltype*, *restfreq*, *interpolation*, *gridder*, *facets*, *chanchunks*, *wprojplanes*, *vptable*, *usepointing*, *mosweight*, *aterm*, *psterm*, *wbawp*, *conjbeams*, *cfcache*, *computepastep*, *rotatepastep*, *pblimit*, *normtype*, *deconvolver*, *scales*, *nterms*, *smallscalebias*, *restoration*, *restoringbeam*, *pbcor*, *outlierfile*, *weighting*, *robust*, *npixels*, *uvtaper*, *niter*, *gain*, *threshold*, *nsigma*, *cycleniter*, *cyclefactor*, *minpsffraction*, *maxpsffraction*, *interactive*, *usemask*, *mask*, *pbmask*, *sidelobethreshold*, *noisethreshold*, *lownoisethreshold*, *negativethreshold*, *smoothfactor*, *minbeamfrac*, *cutthreshold*, *growiterations*, *dogrowprune*, *minpercentchange*, *verbose*, *restart*, *savemodel*, *calcres*, *calcpsf*, *parallel*, *subregion*, *tmpdir*, *btidx*)

suncasa.suncasatasks.private.task_ptclean.**ptclean**(*vis*, *imageprefix*, *imagesuffix*, *ncpu*, *twidth*, *doreg*, *usephacenter*, *reftime*, *toTb*, *sclfactor*, *subregion*, *docompress*, *overwrite*, *selectdata*, *field*, *spw*, *timerange*, *uvrange*, *antenna*, *scan*, *observation*, *intent*, *datacolumn*, *imsize*, *cell*, *phasecenter*, *stokes*, *projection*, *startmodel*, *specmode*, *reffreq*, *nchan*, *start*, *width*, *outframe*, *veltype*, *restfreq*, *interpolation*, *gridder*, *facets*, *chanchunks*, *wprojplanes*, *vptable*, *usepointing*, *mosweight*, *aterm*, *psterm*, *wbawp*, *conjbeams*, *cfcache*, *computepastep*, *rotatepastep*, *pblimit*, *normtype*, *deconvolver*, *scales*, *nterms*, *smallscalebias*, *restoration*, *restoringbeam*, *pbcor*, *outlierfile*, *weighting*, *robust*, *npixels*, *uvtaper*, *niter*, *gain*, *threshold*, *nsigma*, *cycleniter*, *cyclefactor*, *minpsffraction*, *maxpsffraction*, *interactive*, *usemask*, *mask*, *pbmask*, *sidelobethreshold*, *noisethreshold*, *lownoisethreshold*, *negativethreshold*, *smoothfactor*, *minbeamfrac*, *cutthreshold*, *growiterations*, *dogrowprune*, *minpercentchange*, *verbose*, *restart*, *savemodel*, *calcres*, *calcpsf*, *parallel*)

suncasa.suncasatasks.private.task_ptclean6

Module Contents

Functions

clean_iter(tim, vis, imageprefix, imagesuffix, twidth, ...)	
ptclean6(vis, imageprefix, imagesuffix, ncpu, twidth, ...)	

Attributes

tasks
split
tclean
casalog
tools
tbtool
mstool
qatool
tb
ms
qa
c_external

suncasa.suncasatasks.private.task_ptclean6.**tasks**

suncasa.suncasatasks.private.task_ptclean6.**split**

suncasa.suncasatasks.private.task_ptclean6.**tclean**

suncasa.suncasatasks.private.task_ptclean6.**casalog**

suncasa.suncasatasks.private.task_ptclean6.**tools**

suncasa.suncasatasks.private.task_ptclean6.**tbtool**

suncasa.suncasatasks.private.task_ptclean6.**mstool**

suncasa.suncasatasks.private.task_ptclean6.**qatool**

suncasa.suncasatasks.private.task_ptclean6.**tb**

suncasa.suncasatasks.private.task_ptclean6.**ms**

suncasa.suncasatasks.private.task_ptclean6.**qa**

suncasa.suncasatasks.private.task_ptclean6.**c_external = False**

suncasa.suncasatasks.private.task_ptclean6.**clean_iter**(*tim*, *vis*, *imageprefix*, *imagesuffix*, *twidth*, *doreg*, *docompress*, *usephacenter*, *reftime*, *ephem*, *msinfo*, *toTb*, *sclfactor*, *subregion*, *overwrite*, *selectdata*, *field*, *spw*, *timerange*, *uvrange*, *antenna*, *scan*, *observation*, *intent*, *datacolumn*, *imagename*, *imsize*, *cell*, *phasecenter*, *stokes*, *projection*, *startmodel*, *specmode*, *reffreq*, *nchan*, *start*, *width*, *outframe*, *veltype*, *restfreq*, *interpolation*, *perchanweightdensity*, *gridder*, *facets*, *psfphasecenter*, *wprojplanes*, *vptable*, *mosweight*, *aterm*, *psterm*, *wbawp*, *conjbeams*, *cfcache*, *usepointing*, *computepastep*, *rotatepastep*, *pointingoffsetsigdev*, *pblimit*, *normtype*, *deconvolver*, *scales*, *nterms*, *smallscalebias*, *restoration*, *restoringbeam*, *pbcor*, *outlierfile*, *weighting*, *robust*, *noise*, *npixels*, *uvtaper*, *niter*, *gain*, *threshold*, *nsigma*, *cycleniter*, *cyclefactor*, *minpsffraction*, *maxpsffraction*, *interactive*, *usemask*, *mask*, *pbmask*, *sidelobethreshold*, *noisethreshold*, *lownoisethreshold*, *negativethreshold*, *smoothfactor*, *minbeamfrac*, *cutthreshold*, *growiterations*, *dogrowprune*, *minpercentchange*, *verbose*, *fastnoise*, *restart*, *savemodel*, *calcres*, *calcpsf*, *psfcutoff*, *parallel*, *btidx*)

suncasa.suncasatasks.private.task_ptclean6.**ptclean6**(*vis*, *imageprefix*, *imagesuffix*, *ncpu*, *twidth*, *doreg*, *usephacenter*, *reftime*, *toTb*, *sclfactor*, *subregion*, *docompress*, *overwrite*, *selectdata*, *field*, *spw*, *timerange*, *uvrange*, *antenna*, *scan*, *observation*, *intent*, *datacolumn*, *imagename*, *imsize*, *cell*, *phasecenter*, *stokes*, *projection*, *startmodel*, *specmode*, *reffreq*, *nchan*, *start*, *width*, *outframe*, *veltype*, *restfreq*, *interpolation*, *perchanweightdensity*, *gridder*, *facets*, *psfphasecenter*, *wprojplanes*, *vptable*, *mosweight*, *aterm*, *psterm*, *wbawp*, *conjbeams*, *cfcache*, *usepointing*, *computepastep*, *rotatepastep*, *pointingoffsetsigdev*, *pblimit*, *normtype*, *deconvolver*, *scales*, *nterms*, *smallscalebias*, *restoration*, *restoringbeam*, *pbcor*, *outlierfile*, *weighting*, *robust*, *noise*, *npixels*, *uvtaper*, *niter*, *gain*, *threshold*, *nsigma*, *cycleniter*, *cyclefactor*, *minpsffraction*, *maxpsffraction*, *interactive*, *usemask*, *mask*, *pbmask*, *sidelobethreshold*, *noisethreshold*, *lownoisethreshold*, *negativethreshold*, *smoothfactor*, *minbeamfrac*, *cutthreshold*, *growiterations*, *dogrowprune*, *minpercentchange*, *verbose*, *fastnoise*, *restart*, *savemodel*, *calcres*, *calcpsf*, *psfcutoff*, *parallel*)

suncasa.suncasatasks.private.task_subvs

Module Contents

Functions

subvs([vis, outputvis, timerange, spw, mode, ...])	Perform vector subtraction for visibilities

Attributes

tasks
casalog
tools
mstool
msmdtool
qatool
datams
ms_in
datamsmd
qa

suncasa.suncasatasks.private.task_subvs.**tasks**

suncasa.suncasatasks.private.task_subvs.**casalog**

suncasa.suncasatasks.private.task_subvs.**tools**

suncasa.suncasatasks.private.task_subvs.**mstool**

suncasa.suncasatasks.private.task_subvs.**msmdtool**

suncasa.suncasatasks.private.task_subvs.**qatool**

suncasa.suncasatasks.private.task_subvs.**datams**

suncasa.suncasatasks.private.task_subvs.**ms_in**

suncasa.suncasatasks.private.task_subvs.**datamsmd**

suncasa.suncasatasks.private.task_subvs.**qa**

suncasa.suncasatasks.private.task_subvs.**subvs**(*vis=None*, *outputvis=None*, *timerange=''*, *spw=''*, *mode='linear'*, *subtime1=''*, *subtime2=''*, *smoothaxis='time'*, *smoothtype='flat'*, *smoothwidth='5'*, *splitsel=True*, *reverse=False*, *overwrite=False*)

Perform vector subtraction for visibilities Keyword arguments: vis – Name of input visibility file (MS)

default: none; example: vis='ngc5921.ms'

outputvis – Name of output uv-subtracted visibility file (MS)
default: none; example: outputvis='ngc5921_src.ms'

timerange – Time range of performing the UV subtraction:
default='' means all times. examples: timerange = 'YYYY/MM/DD/hh:mm:ss~YYYY/MM/DD/hh:mm:ss' timerange = 'hh:mm:ss~hh:mm:ss'

spw – Select spectral window/channel.
default = '' all the spectral channels. Example: spw='0:1~20'

mode – operation mode

default 'linear'
mode = 'linear': use a linear fit for the background to be subtracted mode = 'lowpass': act as a lowpass filter—smooth the data using different

smooth types and window sizes. Can be performed along either time or frequency axis

mode = 'highpass': act as a highpass filter—smooth the data first, and
subtract the smoothed data from the original. Can be performed along either time or frequency axis

mode = 'linear' expandable parameters:

subtime1 – Time range 1 of the background to be subtracted from the data
default='' means all times. format: timerange = 'YYYY/MM/DD/hh:mm:ss~YYYY/MM/DD/hh:mm:ss' timerange = 'hh:mm:ss~hh:mm:ss'

subtime2 – Time range 2 of the backgroud to be subtracted from the data
default='' means all times. examples: timerange = 'YYYY/MM/DD/hh:mm:ss~YYYY/MM/DD/hh:mm:ss' timerange = 'hh:mm:ss~hh:mm:ss'

mode = 'lowpass' or 'highpass' expandable parameters:

smoothaxis – axis of smooth
Default: 'time' smoothaxis = 'time': smooth is along the time axis smoothaxis = 'freq': smooth is along the frequency axis

smoothtype – type of the smooth depending on the convolving kernel
Default: 'flat' smoothtype = 'flat': convolving kernel is a flat rectangle,

equivalent to a boxcar moving smooth

smoothtype = 'hanning': Hanning smooth kernel. See numpy.hanning smoothtype = 'hamming': Hamming smooth kernel. See numpy.hamming smoothtype = 'bartlett': Bartlett smooth kernel. See numpy.bartlett smoothtype = 'blackman': Blackman smooth kernel. See numpy.blackman

smoothwidth – width of the smooth kernel
Default: 5 Examples: smoothwidth=5, meaning the width is 5 pixels

splitsel – True or False. default = False. If splitsel = False, then the entire input
measurement set is copied as the output measurement set (outputvis), with background subtracted at selected timerange and spectral channels. If splitsel = True,then only the selected timerange and spectral channels are copied into the output measurement set (outputvis).

reverse – True or False. default = False. If reverse = False, then the times indicated
by subtime1 and/or subtime2 are treated as background and subtracted; If reverse = True, then reverse the sign of the background-subtracted data. The option can be used for mapping absorptive structure.

overwrite – True or False. default = False. If overwrite = True and
outputvis already exists, the selected subtime and spw in the output measurment set will be replaced with background subtracted visibilities

Submodules

suncasa.suncasatasks.buildsuncasatasks

Module Contents

suncasa.suncasatasks.buildsuncasatasks.**xmlfiles = ['calibeovsa.xml', 'importeovsa.xml']**

suncasa.suncasatasks.calibeovsa

Module Contents

Classes

_calibeovsa	calibeovsa ---- Calibrating EOVSA one or more measurement sets using calibration products in the SQL database.

Attributes

_pc
calibeovsa

suncasa.suncasatasks.calibeovsa.**_pc**

class suncasa.suncasatasks.calibeovsa.**_calibeovsa**

calibeovsa —- Calibrating EOVSA one or more measurement sets using calibration products in the SQL database.

Calibrating EOVSA one or more measurement sets using calibration products in the SQL database. This task currently only works on pipeline.

——— parameter descriptions ———————————————

vis input EOVSA (uncalibrated) measurement set(s). caltype Types of calibrations to perform caltbdir Directory to place calibration tables. interp Temporal interpolation for phacal table(s) (nearest or linear) docalib If False, only create the calibration tables but do not perform applycal. doflag If true then perform flagging. flagant Antennas to be flagged. Follow CASA syntax of "antenna". doimage If True, produce a quicklook image after calibration (sunpy must be installed). imagedir directory to place output images. Default current directory. antenna antenna/baselines to be used for imaging. Follow CASA syntax of "antenna". timerange Timerange to be imaged. Follow CASA syntax of "timerange". Default is the entire duration of the ms. spw spectral windows to be imaged. Follow CASA syntax of "spw". stokes stokes to be imaged. Follow CASA syntax of "stokes". dosplit If True, plit the corrected data column as output visibility file. outputvis Name of output visibility file. Default is the name of the first vis file ended with ".corrected.ms". doconcat If True, and if more than one visibility dataset provided, concatenate all into one visibility. concatvis Name of output visibility file. Default is the name of the first + last vis file ended with ".corrected.ms". keep_orig_ms Keep the original seperated ms datasets after split?

——— examples ———————————————————————

Calibrating EOVSA one or more measurement sets using calibration products in the SQL database.

Detailed Keyword arguments:

vis – Name of input EOVSA measurement set dataset(s) default: none. Must be supplied example: vis = 'IDB20160524000518.ms' example: vis = ['IDB20160524000518.ms','IDB20160524000528.ms']

caltype – list. Type of calibrations to be applied. 'refpha': reference phase calibration 'refamp': reference amplitude calibration (not used anymore) 'phacal': daily phase calibration 'fluxcal': flux calibration based on total-power measurements default value: ['refpha','phacal'] *** note fluxcal is already implemented in udb_corr when doing importeovsa, should not be used anymore **** *** pipeline only uses ['refpha','phacal']

caltbdir – string. Place to hold calibration tables. Default is current directory. Pipeline should use /data1/eovsa/caltable

interp – string. How interpolation is done for phacal? 'nearest' or 'linear'

docalib – boolean. Default True. If False, only create the calibration tables but do not perform applycal

doflag – boolean. Default True. Peforming flags?

flagant – string. Follow CASA antenna selection syntax. Default '13~15'.

doimage – boolean. Default False. If true, make a quicklook image using the specified time range and specified spw range

imagedir – string. Directory to place the output image.

antenna – string. Default '0~12'. Antenna/baselines to be used for imaging. Follow CASA antenna selection syntax.

timerange – string. Default '' (the whole duration of the visibility data). Follow CASA timerange syntax. e.g., '2017/07/11/20:16:00~2017/07/11/20:17:00'

spw – string. Default '1~3'. Follow CASA spw selection syntax.

stokes – string. Which stokes for the quicklook image. CASA syntax. Default 'XX'

dosplit – boolean. Split the corrected data column?

outputvis – string. Output visibility file after split

doconcat – boolean. If more than one visibility dataset provided, concatenate all into one or make separate outputs if True

concatvis – string. Output visibility file after concatenation

keep_orig_ms – boolean. Default True. Inherited from suncasa.eovsa.concateovsa. Keep the original seperated ms datasets after concatenation?

_info_group_ = 'Calibration'

_info_desc_ = 'Calibrating EOVSA one or more measurement sets using calibration products in the SQL database.'

__call__(*vis=''*, *caltype=[]*, *caltbdir=''*, *interp='nearest'*, *docalib=True*, *doflag=True*, *flagant='13~15'*, *doimage=False*, *imagedir='.'*, *antenna='0~12'*, *timerange=''*, *spw='1~3'*, *stokes='XX'*, *dosplit=False*, *outputvis=''*, *doconcat=False*, *concatvis=''*, *keep_orig_ms=True*)

suncasa.suncasatasks.calibeovsa.**calibeovsa**

suncasa.suncasatasks.concateovsa

Module Contents

Classes

_concateovsa	concateovsa ---- Concatenate several EOVSA visibility data sets.

Attributes

_pc
concateovsa

suncasa.suncasatasks.concateovsa.**_pc**

class suncasa.suncasatasks.concateovsa.**_concateovsa**

concateovsa —- Concatenate several EOVSA visibility data sets.

This is a EOVSA version of CASA concat task.

The list of data sets given in the vis argument are chronologically concatenated into an output data set in concatvis, i.e. the data sets in vis are first ordered by the time of their earliest integration and then concatenated.

If there are fields whose direction agrees within the direction tolerance (parameter dirtol), the actual direction in the resulting, merged output field will be the one from the chronologically first input MS.

If concatvis already exists (e.g., it is the same as the first input data set), then the other input data sets will be appended to the concatvis data set. There is no limit to the number of input data sets.

If none of the input data sets have any scratch columns (model and corrected columns), none are created in the concatvis. Otherwise these columns are created on output and initialized to their default value (1 in model column, data in corrected column) for those data with no input columns.

Spectral windows for each data set with the same chanelization, and within a specified frequency tolerance of another data set will be combined into one spectral window.

A field position in one data set that is within a specified direction tolerance of another field position in any other data set will be combined into one field. The field names need not be the same—only their position is used.

Each appended dataset is assigned a new observation id (provided the entries in the observation table are indeed different).

Keyword arguments: vis – Name of input visibility files to be combined default: none; example: vis = ['src2.ms','ngc5921.ms','ngc315.ms'] concatvis – Name of visibility file that will contain the concatenated data note: if this file exits on disk then the input files are added to this file. Otherwise the new file contains the concatenated data. Be careful here when concatenating to an existing file. default: none; example: concatvis='src2.ms' example: concatvis='outvis.ms'

datacolumn – Which data column to use for processing (case-insensitive). default: 'corrected'; example: datacolumn='data' options: 'data', 'corrected'.

freqtol – Frequency shift tolerance for considering data to be in the same spwid. The number of channels must also be the same. default: '' == 1 Hz example: freqtol='10MHz' will not combine spwid unless they are within 10 MHz. Note: This option is useful to combine spectral windows with very slight frequency differences caused by Doppler tracking, for example.

dirtol – Direction shift tolerance for considering data as the same field default: '' == 1 mas (milliarcsec) example: dirtol='1arcsec' will not combine data for a field unless their phase center differ by less than 1 arcsec. If the field names are different in the input data sets, the name in the output data set will be the first relevant data set in the list.

respectname – If true, fields with a different name are not merged even if their direction agrees (within dirtol) default: False

timesort – If true, the output visibility table will be sorted in time. default: false. Data in order as read in. example: timesort=true Note: There is no constraint on data that is simultaneously observed for more than one field; for example multi-source correlation of VLBA data.

copypointing – Make a proper copy of the POINTING subtable (can be time consuming). If False, the result is an empty POINTING table. default: True

visweightscale – The weights of the individual MSs will be scaled in the concatenated output MS by the factors in this list. SIGMA will be scaled by 1/sqrt(factor). Useful for handling heterogeneous arrays. Use plotms to inspect the "Wt" column as a reference for determining the scaling factors. See the cookbook for more details. example: [1.,3.,3.] - scale the weights of the second and third MS by a factor 3 and the SIGMA column of these MS by a factor 1/sqrt(3). default: [] (empty list) - no scaling

forcesingleephemfield – By default, concat will only merge two ephemeris fields if the first ephemeris covers the time range of the second. Otherwise, two separate fields with separate ephemerides are placed in the output MS. In order to override this behaviour and make concat merge the non-overlapping or only partially overlapping input ephemerides, the name or id of the field in question needs to be placed into the list in parameter 'forcesingleephemfield'. example: ['Neptune'] - will make sure that there is only one joint ephemeris for field Neptune in the output MS default: '' - standard treatment of all ephemeris fields

——— parameter descriptions ———————————————

vis Name of input visibility files to be concatenated concatvis Name of output visibility file datacolumn Which data column(s) to concatenate keep_orig_ms If false, input vis files will be removed cols2rm Columns in concatvis to be removed to slim the concatvis freqtol Frequency shift tolerance for considering data as the same spwid dirtol Direction shift tolerance for considering data as the same field respectname If true, fields with a different name are not merged even if their direction agrees timesort If true, sort by TIME in ascending order copypointing Copy all rows of the POINTING table. visweightscale List of the weight scaling factors to be applied to the individual MSs forcesingleephemfield make sure that there is only one joint ephemeris for every field in this list

——— examples ————————————————————

concateovsa(vis=['UDB20180102161402.ms','UDB20180102173518.ms'], concatvis='UDB20180102_allday.ms') will concatenate 'UDB20180102161402.ms' and 'UDB20180102173518.ms' into 'UDB20180102_allday.ms'

_info_group_ = 'utility, manipulation'

_info_desc_ = 'Concatenate several EOVSA visibility data sets.'

__call__(*vis=''*, *concatvis=''*, *datacolumn='corrected'*, *keep_orig_ms=True*, *cols2rm='model,corrected'*, *freqtol=''*, *dirtol=''*, *respectname=False*, *timesort=True*, *copypointing=True*, *visweightscale=[]*, *forcesingleephemfield=''*)

suncasa.suncasatasks.concateovsa.**concateovsa**

suncasa.suncasatasks.importeovsa

Module Contents

Classes

_importeovsa	importeovsa ---- Parallelized import EOVSA idb file(s) to a measurement set or multiple measurement set.

Attributes

_pc
importeovsa

suncasa.suncasatasks.importeovsa.**_pc**

class suncasa.suncasatasks.importeovsa.**_importeovsa**

importeovsa —- Parallelized import EOVSA idb file(s) to a measurement set or multiple measurement set.

Parallelized imports an arbitrary number of EOVSA idb-format data sets into a casa measurement set. If more than one band is present, they will be put in the same measurement set but in a separate spectral window.

——— parameter descriptions ———————————————

idbfiles Name of input EOVSA idb file(s) or observation time range. ncpu Number of cpu cores to use timebin Bin width for time averaging width Width of output channel relative to MS channel (# to average) visprefix Prefix of vis names (may include the path). udb_corr if applying correction to input UDB files before import to MS. nocreatms If setting nocreatms True, will simulate a model measurement set for the first idb file and copy the model for the rest of idl files in list. If False, will simulate a new measurement set for every idbfile in list. doconcat If concatenate multi casa measurement sets to one file. modelms Name of input model measurement set file. If modelms is assigned, no simulation will start. doscaling If creating a new MS file with the amplitude of visibility data rescaled. keep_nsclms Keep the no scaling measurement sets use_exist_udbcorr If use the existed udb_corr results.

——— examples ————————————————————

Parallelized imports an arbitrary number of EOVSA idb-format data sets into a casa measurement set. If more than one band is present, they will be put in the same measurement set but in a separate spectral window.

Detailed Keyword arguments:

idbfiles – Name of input EOVSA idb file(s) default: none. Must be supplied example: idbfiles = 'IDB20160524000518' example: idbfiles=['IDB20160524000518','IDB20160524000528']

ncpu – Number of cpu cores to use default: 8

visprefix – Prefix of vis names (may include the path) default: none; example: visprefix='sun/']

— Data Selection —

nocreatms – If copying a new MS file instead of create one from MS simulator. default: False

modelms – Name of the standard Measurement Set. IF modelms is not provided, use '/home/user/sjyu/20160531/ms/sun/SUN/SUN_20160531T142234-10m.1s.ms' as a standard MS.

doconcat – If outputing one single MS file

— Channel averaging parameter —

width – Number of input channels to average to create an output channel. If a list is given, each bin will apply to one spw in the selection. default: 1 => no channel averaging. options: (int) or [int]

example: chanbin=[2,3] => average 2 channels of 1st selected spectral window and 3 in the second one.

— Time averaging parameters —

timebin – Bin width for time averaging. When timebin is greater than 0s, the task will average data in time. Flagged data will be included in the average calculation, unless the parameter keepflags is set to False. In this case only partially flagged rows will be used in the average. default: '0s'

_info_group_ = 'Import/export'

_info_desc_ = 'Parallelized import EOVSA idb file(s) to a measurement set or multiple measurement set.'

__call__(*idbfiles=''*, *ncpu=int(1)*, *timebin='0s'*, *width=int(1)*, *visprefix=''*, *udb_corr=True*, *nocreatms=False*, *doconcat=False*, *modelms=''*, *doscaling=False*, *keep_nsclms=False*, *use_exist_udbcorr=False*)

suncasa.suncasatasks.importeovsa.**importeovsa**

suncasa.suncasatasks.pimfit

Module Contents

Classes

_pimfit	pimfit ---- Fit one or more elliptical Gaussian components on an image region(s)

Attributes

_pc
pimfit

suncasa.suncasatasks.pimfit.**_pc**

class suncasa.suncasatasks.pimfit.**_pimfit**

pimfit —- Fit one or more elliptical Gaussian components on an image region(s)

——— parameter descriptions ———————————————

imagefiles A list of the input images ncpu Number of cpu cores to use doreg True if use vla_prep to register the image ephemfile emphemeris file generated from vla_prep.read_horizons() timestamps A list of timestamps of the input images msinfofile time-dependent phase center information generated from vla_prep.read_msinfo() box Rectangular region(s) to select in direction plane. See "help par.box" for details. Default is to use the entire direction plane. region Region selection. See "help par.region" for details. Default is to use the full image. chans Channels to use. See "help par.chans" for details. Default is to use all channels. stokes Stokes planes to use. See "help par.stokes" for details. Default is to use first Stokes plane. mask Mask to use. See help par.mask. Default is none. includepix Range of pixel values to include for fitting. excludepix Range of pixel values to exclude for fitting. residual Name of output residual image. model Name of output model image. estimates Name of file containing initial estimates of component parameters. logfile Name of file to write fit results. append If logfile exists, append to it if True or overwrite it if False newestimates File to write fit results which can be used as initial estimates for next run. complist Name of output component list table. overwrite Overwrite component list table if it exists? dooff Also fit a zero level offset? Default is False offset Initial estimate of zero-level offset. Only used if doff is True. Default is 0.0 fixoffset Keep the zero level offset fixed during fit? Default is False stretch Stretch the mask if necessary and possible? See help par.stretch rms RMS to use in calculation of uncertainties. Numeric or valid quantity (record or string). If numeric, it is given units of the input image. If quantity, units must conform to image units. If not positive, the rms of the residual image, in the region of the fit, is used. noisefwhm Noise correlation beam FWHM. If numeric value, interpreted as pixel widths. If quantity (dictionary, string), it must have angular units. summary File name to which to write table of fit parameters. [1;42mRETURNS[1;m void

——— examples ————————————————————

PARAMETER SUMMARY imagename Name of the input image box Rectangular region(s) to select in direction plane. See "help par.box" for details. Default is to use the entire direction plane. eg "100, 120, 200, 220, 300, 300, 400, 400" to use two boxes. region Region selection. See "help par.region" for details. Default is to use the full image. chans Channels to use. See "help par.chans" for details. Default is to use all channels. stokes Stokes planes to use. See "help par.stokes" for details. Default is to use first Stokes plane. mask Mask to use. See help par.mask. Default is none. includepix Range of pixel values to include for fitting. Array of two numeric values assumed to have same units as image pixel values. Only one of includepix or excludepix can be specified. excludepix Range of pixel values to exclude for fitting. Array of two numeric values assumed to have same units as image pixel values. Only one of includepix or excludepix can be specified. residual Name of the residual image to write. model Name of the model image to write. estimates Name of file containing initial estimates of component parameters (see below for formatting details). logfile Name of file to write fit results. append If logfile exists, append to it (True) or overwrite it (False). newestimates File to write fit results which can be used as initial estimates for next run. complist Name of output component list table. overwrite Overwrite component list table if it exists? dooff Simultaneously fit a zero-level offset? offset Initial estimate for the zero-level offset. Only used if dooff is True. fixoffset Hold zero-level offset constant during fit? Only used if dooff is True. stretch Stretch the input mask if necessary and possible. Only used if a mask is specified. See help par.stretch. rms RMS to use in calculation of various uncertainties, assumed to have units of the input image. If not positve, the rms of the residual image is used. noisefwhm Noise correlation beam FWHM. If numeric value, interpreted as

pixel widths. If quantity (dictionary, string), it must have angular units. summary File name to which to write table of fit parameters.

OVERVIEW This application is used to fit one or more two dimensional gaussians to sources in an image as well as an optional zero-level offset. Fitting is limited to a single polarization but can be performed over several contiguous spectral channels. If the image has a clean beam, the report and returned dictionary will contain both the convolved and the deconvolved fit results.

When dooff is False, the method returns a dictionary with three keys, 'converged', 'results', and 'deconvolved'. The value of 'converged' is a boolean array which indicates if the fit converged on a channel by channel basis. The value of 'results' is a dictionary representing a component list reflecting the fit results. In the case of an image containing beam information, the sizes and position angles in the 'results' dictionary are those of the source(s) convolved with the restoring beam, while the same parameters in the 'deconvolved' dictionary represent the source sizes deconvolved from the beam. In the case where the image does not contain a beam, 'deconvolved' will be absent. Both the 'results' and 'deconvolved' dictionaries can be read into a component list tool (default tool is named cl) using the fromrecord() method for easier inspection using tool methods, eg

cl.fromrecord(res['results'])

although this currently only works if the flux density units are conformant with Jy.

There are also values in each component subdictionary not used by cl.fromrecord() but meant to supply additional information. There is a 'peak' subdictionary for each component that provides the peak intensity of the component. It is present for both 'results' and 'deconvolved' components. There is also a 'sum' subdictionary for each component indicated the simple sum of pixel values in the the original image enclosed by the fitted ellipse. There is a 'channel' entry in the 'spectrum' subdictionary which provides the zero-based channel number in the input image for which the solution applies. In addtion, if the image has a beam(s), then there will be a 'beam' subdictionary associated with each component in both the 'results' and 'deconvolved' dictionaries. This subdictionary will have three keys: 'beamarcsec' will be a subdictionary giving the beam dimensions in arcsec, 'beampixels' will have the value of the beam area expressed in pixels, and 'beamster' will have the value of the beam area epressed in steradians. Also, if the image has a beam(s), in the component level dictionaries will be an 'ispoint' entry with an associated boolean value describing if the component is consistent with a point source.

If dooff is True, in addtion to the specified number of gaussians, a zero-level offset will also be fit. The initial estimate for this offset is specified using the offset parameter. Units are assumed to be the same as the image brightness units. The zero level offset can be held constant during the fit by specifying fixoffset=True. In the case of dooff=True, the returned dictionary contains two additional keys, 'zerooff' and 'zerooofferr', which are both dictionaries containing 'unit' and 'value' keys. The values associated with the 'value' keys are arrays containing the the fitted zero level offset value and its error, respectively, for each channel. In cases where the fit did not converge, these values are set to NaN. The value associated with 'unit' is just the i`mage brightness unit.

The region can either be specified by a box(es) or a region. Ranges of pixel values can be included or excluded from the fit. If specified using the box parameter, multiple boxes can be given using the format box="blcx1, blcy1, trcx1, trcy1, blcx2, blcy2, trcx2, trcy2, ... , blcxN, blcyN, trcxN, trcyN" where N is the number of boxes. In this case, the union of the specified boxes will be used.

If specified, the residual and/or model images for successful fits will be written.

If an estimates file is not specified, an attempt is made to estimate initial parameters and fit a single Gaussian. If a multiple Gaussian fit is desired, the user must specify initial estimates via a text file (see below for details).

The user has the option of writing the result of the fit to a log file, and has the option of either appending to or overwriting an existing file.

The user has the option of writing the (convolved) parameters of a successful fit to a file which can be fed back to fitcomponents() as the estimates file for a subsequent run.

The user has the option of writing the fit results in tabular format to a file whose name is specified using the summary parameter.

If specified and positive, the value of rms is used to calculate the parameter uncertainties, otherwise, the rms in the selected region in the relevant channel is used for these calculations.

The noisefwhm parameter represents the noise-correlation beam FWHM. If specified as a quantity, it should have angular units. If specified as a numerical value, it is set equal to that number of pixels. If specified and greater than or equal to the pixel size, it is used to calculate parameter uncertainties using the correlated noise equations (see below). If it is specified but less than a pixel width, the the uncorrelated noise equations (see below) are used to compute the parameter uncertainties. If it is not specified and the image has a restoring beam(s), the the correlated noise equations are used to compute parameter uncertainties using the geometric mean of the relevant beam major and minor axes as the noise-correlation beam FWHM. If noisefwhm is not specified and the image does not have a restoring beam, then the uncorrelated noise equations are used to compute the parameter uncertainties.

SUPPORTED UNITS

Currently only images with brightness units conformant with Jy/beam, Jy.km/s/beam, and K are fully supported for fitting. If your image has some other base brightness unit, that unit will be assumed to be equivalent to Jy/pixel and results will be calculated accordingly. In particular, the flux density (reported as Integrated Flux in the logger and associated with the "flux" key in the returned component subdictionary(ies)) for such a case represents the sum of pixel values.

Note also that converting the returned results subdictionary to a component list via cl.fromrecord() currently only works properly if the flux density units in the results dictionary are conformant with Jy. If you need to be able to run cl.fromrecord() on the resulting dictionary you can first modify the flux density units by hand to be (some prefix)Jy and then run cl.fromrecord() on that dictionary, bearing in mind your unit conversion.

If the input image has units of K, the flux density of components will be reported in units of [prefix]K*rad*rad, where prefix is an SI prefix used so that the numerical value is between 1 and 1000. To convert to units of K*beam, determine the area of the appropriate beam, which is given by pi/(4*ln(2))*bmaj*bmin, where bmaj and bmin are the major and minor axes of the beam, and convert to steradians (=rad*rad). This value is included in the beam portion of the component subdictionary (key 'beamster'). Then divide the numerical value of the logged flux density by the beam area in steradians. So, for example

begin{verbatim} # run on an image with K brightness units res = imfit(…) # get the I flux density in K*beam of component 0 comp = res['results']['component0'] flux_density_kbeam = comp['flux']['value'][0]/comp['beam']['beamster'] end{verbatim}

FITTING OVER MULTIPLE CHANNELS

For fitting over multiple channels, the result of the previous successful fit is used as the estimate for the next channel. The number of gaussians fit cannot be varied on a channel by channel basis. Thus the variation of source structure should be reasonably smooth in frequency to produce reliable fit results.

MASK SPECIFICATION

Mask specification can be done using an LEL expression. For example

mask = '"myimage">5' will use only pixels with values greater than 5.

INCLUDING AND EXCLUDING PIXELS

Pixels can be included or excluded from the fit based on their values using these parameters. Note that specifying both is not permitted and will cause an error. If specified, both take an array of two numeric values.

ESTIMATES

Initial estimates of fit parameters may be specified via an estimates text file. Each line of this file should contain a set of parameters for a single gaussian. Optionally, some of these parameters can be fixed during the fit. The format of each line is

peak intensity, peak x-pixel value, peak y-pixel value, major axis, minor axis, position angle, fixed

The fixed parameter is optional. The peak intensity is assumed to be in the same units as the image pixel values (eg Jy/beam). The peak coordinates are specified in pixel coordinates. The major and minor axes and the position angle are the convolved parameters if the image has been convolved with a clean beam and are specified as quantities. The fixed parameter is optional and is a string. It may contain any combination of the following characters 'f' (peak intensity), 'x' (peak x position), 'y' (peak y position), 'a' (major axis), 'b' (minor axis), 'p' (position angle).

In addition, lines in the file starting with a # are considered comments.

An example of such a file is:

begin{verbatim} # peak intensity must be in map units 120, 150, 110, 23.5arcsec, 18.9arcsec, 120deg 90, 60, 200, 46arcsec, 23arcsec, 140deg, fxp end{verbatim}

This is a file which specifies that two gaussians are to be simultaneously fit, and for the second gaussian the specified peak intensity, x position, and position angle are to be held fixed during the fit.

ERROR ESTIMATES

Error estimates are based on the work of Condon 1997, PASP, 109, 166. Key assumptions made are: * The given model (elliptical Gaussian, or elliptical Gaussian plus constant offset) is an adequate representation of the data * An accurate estimate of the pixel noise is provided or can be derived (see above). For the case of correlated noise (e.g., a CLEAN map), the fit region should contain many "beams" or an independent value of rms should be provided. * The signal-to-noise ratio (SNR) or the Gaussian component is large. This is necessary because a Taylor series is used to linearize the problem. Condon (1997) states that the fractional bias in the fitted amplitude due to this assumption is of order 1/(S*S), where S is the overall SNR of the Gaussian with respect to the given data set (defined more precisely below). For a 5 sigma "detection" of the Gaussian, this is a 4% effect. * All (or practically all) of the flux in the component being fit falls within the selected region. If a constant offset term is simultaneously fit and not fixed, the region of interest should be even larger. The derivations of the expressions summarized in this note assume an effectively infinite region.

Two sets of equations are used to calculate the parameter uncertainties, based on if the noise is correlated or uncorrelated. The rules governing which set of equations are used have been described above in the description of the noisefwhm parameter.

In the case of uncorrelated noise, the equations used are

f(A) = f(I) = f(M) = f(m) = k*s(x)/M = k*s(y)/m = (s(p)/sqrt(2))*((M*M - m*m)/(M*m)) = sqrt(2)/S

where s(z) is the uncertainty associated with parameter z, f(z) = s(z)/abs(z) is the fractional uncertainty associated with parameter z, A is the peak intensity, I is the flux density, M and m are the FWHM major and minor axes, p is the position angle of the component, and k = sqrt(8*ln(2)). s(x) and s(y) are the direction uncertainties of the component measured along the major and minor axes; the resulting uncertainties measured along the principle axes of the image direction coordinate are calculated by propagation of errors using the 2D rotation matrix which enacts the rotation through the position angle plus 90 degrees. S is the overall signal to noise ratio of the component, which, for the uncorrelated noise case is given by

S = (A/(k*h*r))*sqrt(pi*M*m)

where h is the pixel width of the direction coordinate and r is the rms noise (see the discussion above for the rules governing how the value of r is determined).

For the correlated noise case, the same equations are used to determine the uncertainties as in the uncorrelated noise case, except for the uncertainty in I (see below). However, S is given by

S = (A/(2*r*N)) * sqrt(M*m) * (1 + ((N*N/(M*M)))**(a/2)) * (1 + ((N*N/(m*m)))**(b/2))

where N is the noise-correlation beam FWHM (see discussion of the noisefwhm parameter for rules governing how this value is determined). "**" indicates exponentiation and a and b depend on which uncertainty is being calculated. For sigma(A), a = b = 3/2. For M and x, a = 5/2 and b = 1/2. For m, y, and p, a = 1/2 and b = 5/2. f(I) is calculated in the correlated noise case according to

f(I) = sqrt(f(A)*f(A) + (N*N/(M*m))*(f(M*f(M) + f(m)*f(m))))

Note well the following caveats: * Fixing Gaussian component parameters will tend to cause the parameter uncertainties reported for free parameters to be overestimated. * Fitting a zero level offset that is not fixed will tend to cause the reported parameter uncertainties to be slightly underestimated. * The parameter uncertainties will be inaccurate at low SNR (a ~10% for SNR = 3). * If the fitted region is not considerably larger than the largest component that is fit, parameter uncertainties may be mis-estimated. * An accurate rms noise measurement, r, for the region in question must be supplied. Alternatively, a sufficiently large signal-free region must be present in the selected region (at least about 25 noise beams in area) to auto-derive such an estimate. * If the image noise is not statistically independent from pixel to pixel, a reasonably accurate noise correlation scale, N, must be provided. If the noise correlation function is not approximately Gaussian, the correlation length can be estimated using

N = sqrt(2*ln(2)/pi)* double-integral(dx dy C(x,y))/sqrt(double-integral(dx dy C(x, y) * C(x,y)))

where C(x,y) is the associated noise-smoothing function * If fitted model components have significan spatial overlap, the parameter uncertainties are likely to be mis-estimated (i.e., correlations between the parameters of separate components are not accounted for). * If the image being analyzed is an interferometric image with poor uv sampling, the parameter uncertainties may be significantly underestimated.

The deconvolved size and position angle errors are computed by taking the maximum of the absolute values of the differences of the best fit deconvolved value of the given parameter and the deconvolved size of the eight possible combinations of (FWHM major axis +/- major axis error), (FWHM minor axis +/- minor axis error), and (position andle +/- position angle error). If the source cannot be deconvolved from the beam (if the best fit convolved source size cannot be deconvolved from the beam), upper limits on the deconvolved source size are sometimes reported. These limits simply come from the maximum major and minor axes of the deconvolved gaussians taken from trying all eight of the aforementioned combinations. In the case none of these combinations produces a deconvolved size, no upper limit is reported.

EXAMPLE:

Here is how one might fit two gaussians to multiple channels of a cube using the fit from the previous channel as the initial estimate for the next. It also illustrates how one can specify a region in the associated continuum image as the region to use as the fit for the channel.

begin{verbatim} default imfit imagename = "co_cube.im" # specify region using region from continuum region = "continuum.im:source.rgn" chans = "2~20" # only use pixels with positive values in the fit excludepix = [-1e10,0] # estimates file contains initial parameters for two Gaussians in channel 2 estimates = "initial_estimates.txt" logfile = "co_fit.log" # append results to the log file for all the channels append = "True" imfit()

_info_group_ = 'analysis'

_info_desc_ = 'Fit one or more elliptical Gaussian components on an image region(s)'

__call__(*imagefiles=[]*, *ncpu=int(8)*, *doreg=False*, *ephemfile=''*, *timestamps=[]*, *msinfofile=''*, *box=''*, *region=''*, *chans=''*, *stokes=''*, *mask=''*, *includepix=[]*, *excludepix=[]*, *residual=''*, *model=''*, *estimates=''*, *logfile=''*, *append=True*, *newestimates=''*, *complist=''*, *overwrite=False*, *dooff=False*, *offset=float(0.0)*, *fixoffset=False*, *stretch=False*, *rms=int(0)*, *noisefwhm=''*, *summary=''*)

suncasa.suncasatasks.pimfit.**pimfit**

suncasa.suncasatasks.pmaxfit

Module Contents

Classes

_pmaxfit	pmaxfit ---- Find maximum and do parabolic fit in the sky

Attributes

_pc
pmaxfit

suncasa.suncasatasks.pmaxfit._pc

class suncasa.suncasatasks.pmaxfit.**_pmaxfit**

pmaxfit —- Find maximum and do parabolic fit in the sky

PARAMETER SUMMARY imagename Name of the input image box Rectangular region(s) to select in direction plane. See "help par.box" for details. Default is to use the entire direction plane. eg "100, 120, 200, 220, 300, 300, 400, 400" to use two boxes.

OVERVIEW This application finds the pixel with the maximum value in the region, and then uses function findsources to generate a Componentlist with one component.

The method returns a dictionary with fours keys, 'succeeded', 'timestamps', 'imagenames' and 'outputs'. The value of 'outputs' is a dictionary representing a component list reflecting the fit results over multiple channels. Both the 'outputs' dictionaries can be read into a component list tool (default tool is named cl) using the fromrecord() method for easier inspection using tool methods, eg

FITTING OVER MULTIPLE CHANNELS

For fitting over multiple channels, the result of the previous successful fit is used as the estimate for the next channel. The number of gaussians fit cannot be varied on a channel by channel basis. Thus the variation of source structure should be reasonably smooth in frequency to produce reliable fit results.

——— parameter descriptions ———————————————

imagefiles A list of the input images ncpu Number of cpu cores to use box Rectangular region(s) to select in direction plane. See "help par.box" for details. Default is to use the entire direction plane. width Half-width of fit grid [1;42mRETURNS[1;m void

——— examples ————————————————————

EXAMPLE:

Here is how one might fit two gaussians to multiple channels of a cube using the fit from the previous channel as the initial estimate for the next. It also illustrates how one can specify a region in the associated continuum image as the region to use as the fit for the channel.

begin{verbatim} default pmaxfit imagename = "co_cube.im" # specify region using region from continuum box = "100,120,200,200" pmaxfit()

_info_group_ = 'analysis'

_info_desc_ = 'Find maximum and do parabolic fit in the sky'

__call__(*imagefiles=[]*, *ncpu=int(8)*, *box=''*, *width=int(5)*)

suncasa.suncasatasks.pmaxfit.**pmaxfit**

suncasa.suncasatasks.ptclean

Module Contents

Classes

_ptclean	ptclean ---- Parallelized tclean in consecutive time steps

Attributes

_pc
ptclean

suncasa.suncasatasks.ptclean.**_pc**

class suncasa.suncasatasks.ptclean.**_ptclean**

ptclean —- Parallelized tclean in consecutive time steps

Parallelized clean in consecutive time steps. Packed over CASA tclean.

——— parameter descriptions ———————————————

vis Name(s) of input visibility file(s)

default: none; example: vis='ngc5921.ms'

vis=['ngc5921a.ms','ngc5921b.ms']; multiple MSes

imageprefix Prefix of output image names (usually useful in defining the output path) imagesuffix Suffix of output image names (usually useful in specifyting the image type, version, etc.) ncpu Number of cpu cores to use twidth Number of time pixels to average doreg True if use vla_prep to register the image usephacenter True if use the phacenter information from the measurement set (e.g., VLA); False to assume the phase center is at the solar disk center (EOVSA) reftime Reference time of the J2000 coordinates associated with the ephemeris target. e.g., "2012/03/03/12:00". This is used for helioimage2fits.py to find the solar x y offset in order to register the image. If not set, use the actual timerange of the image (default) toTb True if convert to brightness temperature sclfactor scale the brightness temperature up by its value subregion The name of a CASA region string

The name of a CASA image or region file or region string. Only locations within the region will output to the fits file. If regions specified fall completely outside of the image, ptclean will throw an error.

Manual mask options/examples :

subregion='box[[224pix,224pix],[288pix,288pix]]' : A CASA region string.

docompress True if compress the output fits files overwrite True if overwrite the image selectdata Enable data selection parameters. field to image or mosaic. Use field id(s) or name(s).

['go listobs' to obtain the list id's or names]

default: ''= all fields
If field string is a non-negative integer, it is assumed to be a field index otherwise, it is assumed to be a

field name

field='0~2'; field ids 0,1,2 field='0,4,5~7'; field ids 0,4,5,6,7 field='3C286,3C295'; field named 3C286 and 3C295 field = '3,4C*'; field id 3, all names starting with 4C For multiple MS input, a list of field strings can be used: field = ['0~2','0~4']; field ids 0-2 for the first MS and 0-4

for the second

field = '0~2'; field ids 0-2 for all input MSes

spw l window/channels

NOTE: channels de-selected here will contain all zeros if
selected by the parameter mode subparameters.

default: ''=all spectral windows and channels
spw='0~2,4'; spectral windows 0,1,2,4 (all channels) spw='0:5~61'; spw 0, channels 5 to 61 spw='<2'; spectral windows less than 2 (i.e. 0,1) spw='0,10,3:3~45'; spw 0,10 all channels, spw 3,

channels 3 to 45.

spw='0~2:2~6'; spw 0,1,2 with channels 2 through 6 in each. For multiple MS input, a list of spw strings can be used: spw=['0','0~3']; spw ids 0 for the first MS and 0-3 for the second spw='0~3' spw ids 0-3 for all input MS spw='3:10~20;50~60' for multiple channel ranges within spw id 3 spw='3:10~20;50~60,4:0~30' for different channel ranges for spw ids 3 and 4 spw='0:0~10,1:20~30,2:1;2;3'; spw 0, channels 0-10,

spw 1, channels 20-30, and spw 2, channels, 1,2 and 3

spw='1~4;6:15~48' for channels 15 through 48 for spw ids 1,2,3,4 and 6

timerange Range of time to select from data

default: '' (all); examples, timerange = 'YYYY/MM/DD/hh:mm:ss~YYYY/MM/DD/hh:mm:ss' Note: if YYYY/MM/DD is missing date defaults to first

day in data set

timerange='09:14:0~09:54:0' picks 40 min on first day timerange='25:00:00~27:30:00' picks 1 hr to 3 hr

30min on NEXT day

timerange='09:44:00' pick data within one integration
of time

timerange='> 10:24:00' data after this time For multiple MS input, a list of timerange strings can be used: timerange=['09:14:0~09:54:0','> 10:24:00'] timerange='09:14:0~09:54:0"; apply the same timerange for

all input MSes

uvrange Select data within uvrange (default unit is meters)
default: '' (all); example: uvrange='0~1000klambda'; uvrange from 0-1000 kilo-lambda uvrange='> 4klambda';uvranges greater than 4 kilo lambda For multiple MS input, a list of uvrange strings can be used: uvrange=['0~1000klambda','100~1000klamda'] uvrange='0~1000klambda'; apply 0-1000 kilo-lambda for all

input MSes

antenna Select data based on antenna/baseline

default: '' (all) If antenna string is a non-negative integer, it is

assumed to be an antenna index, otherwise, it is considered an antenna name.

antenna='5&6'; baseline between antenna index 5 and
index 6.

antenna='VA05&VA06'; baseline between VLA antenna 5
and 6.

antenna='5&6;7&8'; baselines 5-6 and 7-8 antenna='5'; all baselines with antenna index 5 antenna='05'; all baselines with antenna number 05

(VLA old name)

antenna='5,6,9'; all baselines with antennas 5,6,9
index number

For multiple MS input, a list of antenna strings can be used: antenna=['5','5&6']; antenna='5'; antenna index 5 for all input MSes antenna='!DV14'; use all antennas except DV14

scan Scan number range

default: '' (all) example: scan='1~5' For multiple MS input, a list of scan strings can be used: scan=['0~100','10~200'] scan='0~100; scan ids 0-100 for all input MSes

observation Observation ID range
default: '' (all) example: observation='1~5'

intent Scan Intent(s)

default: '' (all) example: intent='TARGET_SOURCE' example: intent='TARGET_SOURCE1,TARGET_SOURCE2' example: intent='TARGET_POINTING*'

datacolumn Data column to image (data or observed, corrected)
default:'data' (If 'corrected' does not exist, it will use 'data' instead)

imsize Number of pixels

example
[imsize = [350,250]] imsize = 500 is equivalent to [500,500]

To take proper advantage of internal optimized FFT routines, the number of pixels must be even and factorizable by 2,3,5,7 only.

cell Cell size
example: cell=['0.5arcsec,'0.5arcsec'] or cell=['1arcmin', '1arcmin'] cell = '1arcsec' is equivalent to ['1arcsec','1arcsec']

phasecenter Phase center of the image (string or field id); if the phasecenter is the name known major solar system object ('MERCURY', 'VENUS', 'MARS', 'JUPITER', 'SATURN', 'URANUS', 'NEPTUNE', 'PLUTO', 'SUN', 'MOON') or is an ephemerides table then that source is tracked and the background sources get smeared. There is a special case, when phasecenter='TRACKFIELD', which will use the ephemerides or polynomial phasecenter in the FIELD table of the MS's as the source center to track.

example: phasecenter=6

phasecenter='J2000 19h30m00 -40d00m00' phasecenter='J2000 292.5deg -40.0deg' phasecenter='J2000 5.105rad -0.698rad' phasecenter='ICRS 13:05:27.2780 -049.28.04.458'

phasecenter='myComet_ephem.tab' phasecenter='MOON' phasecenter='TRACKFIELD'

stokes Stokes Planes to make

default='I'; example: stokes='IQUV';

Options: 'I','Q','U','V','IV','QU','IQ','UV','IQUV','RR','LL','XX','YY','RRLL','XXYY','pseudoI'

Note

[Due to current internal code constraints, if any correlation pair] is flagged, by default, no data for that row in the MS will be used. So, in an MS with XX,YY, if only YY is flagged, neither a Stokes I image nor an XX image can be made from those data points. In such a situation, please split out only the unflagged correlation into a separate MS.

Note

[The 'pseudoI' option is a partial solution, allowing Stokes I imaging] when either of the parallel-hand correlations are unflagged.

The remaining constraints shall be removed (where logical) in a future release.

projection Coordinate projection

Examples : SIN, NCP A list of supported (but untested) projections can be found here : http://casa.nrao.edu/active/docs/doxygen/html/classcasa_1_1Projection.html#a3d5f9ec787e4eabdce57ab5edaf7c0cd

startmodel Name of starting model image

The contents of the supplied starting model image will be copied to the imagename.model before the run begins.

example : startmodel = 'singledish.im'

For deconvolver='mtmfs', one image per Taylor term must be provided. example : startmodel = ['try.model.tt0', 'try.model.tt1']

startmodel = ['try.model.tt0'] will use a starting model only
for the zeroth order term.

startmodel = ['','try.model.tt1'] will use a starting model only
for the first order term.

This starting model can be of a different image shape and size from what is currently being imaged. If so, an image regrid is first triggered to resample the input image onto the target coordinate system.

A common usage is to set this parameter equal to a single dish image

Negative components in the model image will be included as is.

[Note

[If an error occurs during image resampling/regridding,] please try using task imregrid to resample the starting model image onto a CASA image with the target shape and coordinate system before supplying it via startmodel]

specmode Spectral definition mode (mfs,cube,cubedata, cubesource)

mode='mfs'
[Continuum imaging with only one output image channel.] (mode='cont' can also be used here)

mode='cube'
[Spectral line imaging with one or more channels]

Parameters start, width,and nchan define the spectral coordinate system and can be specified either in terms of channel numbers, frequency or velocity in whatever spectral frame is specified in 'outframe'. All internal and output images are made with outframe as the base spectral frame. However imaging code internally uses the fixed spectral frame, LSRK for automatic internal software Doppler tracking so that a spectral line observed over an extended time range will line up appropriately. Therefore the output images have additional spectral frame conversion layer in LSRK on the top the base frame.

(Note
[Even if the input parameters are specified in a frame] other than LSRK, the viewer still displays spectral axis in LSRK by default because of the conversion frame layer mentioned above. The viewer can be used to relabel the spectral axis in any desired frame - via the spectral reference option under axis label properties in the data display options window.)

mode='cubedata'
[Spectral line imaging with one or more channels] There is no internal software Doppler tracking so a spectral line observed over an extended time range may be smeared out in frequency. There is strictly no valid spectral frame with which to label the output

images, but they will list the frame defined in the MS.

mode='cubesource': Spectral line imaging while tracking moving source (near field or solar system objects). The velocity of the source is accounted and the frequency reported is in the source frame. As there is not SOURCE frame defined, the frame reported will be REST (as it may not be in the rest frame emission region may be moving w.r.t the systemic velocity frame)

reffreq Reference frequency of the output image coordinate system

Example : reffreq='1.5GHz' as a string with units.

By default, it is calculated as the middle of the selected frequency range.

For deconvolver='mtmfs' the Taylor expansion is also done about this specified reference frequency.

nchan Number of channels in the output image
For default (=-1), the number of channels will be automatically determined based on data selected by 'spw' with 'start' and 'width'. It is often easiest to leave nchan at the default value. example: nchan=100

start First channel (e.g. start=3,start='1.1GHz',start='15343km/s')
of output cube images specified by data channel number (integer), velocity (string with a unit), or frequency (string with a unit). Default:''; The first channel is automatically determined based on the 'spw' channel selection and 'width'. When the channel number is used along with the channel selection

in 'spw' (e.g. spw='0:6~100'),

'start' channel number is RELATIVE (zero-based) to the selected channels in 'spw'. So for the above example, start=1 means that the first image channel is the second selected data channel, which is channel 7. For specmode='cube', when velocity or frequency is used it is interpreted with the frame defined in outframe. [The parameters of the desired output cube can be estimated by using the 'transform' functionality of 'plotms'] examples: start='5.0km/s'; 1st channel, 5.0km/s in outframe

start='22.3GHz'; 1st channel, 22.3GHz in outframe

width Channel width (e.g. width=2,width='0.1MHz',width='10km/s') of output cube images

specified by data channel number (integer), velocity (string with a unit), or or frequency (string with a unit). Default:''; data channel width The sign of width defines the direction of the channels to be incremented. For width specified in velocity or frequency with '-' in front gives image channels in decreasing velocity or frequency, respectively. For specmode='cube', when velocity or frequency is used it is interpreted with the reference frame defined in outframe. examples: width='2.0km/s'; results in channels with increasing velocity

width='-2.0km/s'; results in channels with decreasing velocity width='40kHz'; results in channels with increasing frequency width=-2; results in channels averaged of 2 data channels incremented from

high to low channel numbers

outframe Spectral reference frame in which to interpret 'start' and 'width'

Options: '','LSRK','LSRD','BARY','GEO','TOPO','GALACTO','LGROUP','CMB' example: outframe='bary' for Barycentric frame

REST – Rest frequency LSRD – Local Standard of Rest (J2000)

– as the dynamical definition (IAU, [9,12,7] km/s in galactic coordinates)

LSRK – LSR as a kinematical (radio) definition

– 20.0 km/s in direction ra,dec = [270,+30] deg (B1900.0)

BARY – Barycentric (J2000) GEO — Geocentric TOPO – Topocentric GALACTO – Galacto centric (with rotation of 220 km/s in direction l,b = [90,0] deg. LGROUP – Local group velocity – 308km/s towards l,b = [105,-7] deg (F. Ghigo)

CMB – CMB velocity – 369.5km/s towards l,b = [264.4, 48.4] deg (F. Ghigo) DEFAULT = LSRK

veltype Velocity type (radio, z, ratio, beta, gamma, optical)

For start and/or width specified in velocity, specifies the velocity definition Options: 'radio','optical','z','beta','gamma','optical' NOTE: the viewer always defaults to displaying the 'radio' frame,

but that can be changed in the position tracking pull down.

The different types (with F = f/f0, the frequency ratio), are:

Z = (-1 + 1/F)

RATIO = (F) * RADIO = (1 - F) OPTICAL == Z BETA = ((1 - F2)/(1 + F2)) GAMMA = ((1 + F2)/2F) * RELATIVISTIC == BETA (== v/c) DEFAULT == RADIO Note that the ones with an '*' have no real interpretation (although the calculation will proceed) if given as a velocity.

restfreq List of rest frequencies or a rest frequency in a string.

Specify rest frequency to use for output image. *Currently it uses the first rest frequency in the list for translation of velocities. The list will be stored in the output images. Default: []; look for the rest frequency stored in the MS, if not available, use center frequency of the selected channels examples: restfreq=['1.42GHz']

restfreq='1.42GHz'

interpolation Spectral interpolation (nearest,linear,cubic)

Interpolation rules to use when binning data channels onto image channels and evaluating visibility values at the centers of image channels.

Note

['linear' and 'cubic' interpolation requires data points on both sides of] each image frequency.

Errors are therefore possible at edge channels, or near flagged data channels. When image channel width is much larger than the data channel width there is nothing much to be gained using linear or cubic thus not worth the extra computation involved.

gridder Gridding options (standard, wproject, widefield, mosaic, awproject)

The following options choose different gridding convolution functions for the process of convolutional resampling of the measured visibilities onto a regular uv-grid prior to an inverse FFT. Model prediction (degridding) also uses these same functions. Several wide-field effects can be accounted for via careful choices of convolution functions. Gridding (degridding) runtime will rise in proportion to the support size of these convolution functions (in uv-pixels).

standard : Prolate Spheroid with 3x3 uv pixel support size

[This mode can also be invoked using 'ft' or 'gridft']

wproject

[W-Projection algorithm to correct for the widefield]

non-coplanar baseline effect. [Cornwell et.al 2008]

wprojplanes is the number of distinct w-values at which to compute and use different gridding convolution functions (see help for wprojplanes).

Convolution function support size can range

from 5x5 to few 100 x few 100.

[This mode can also be invoked using 'wprojectft']

widefield : Facetted imaging with or without W-Projection per facet.

A set of facets x facets subregions of the specified image are gridded separately using their respective phase centers (to minimize max W). Deconvolution is done on the joint full size image, using a PSF from the first subregion.

wprojplanes=1 : standard prolate spheroid gridder per facet. wprojplanes > 1 : W-Projection gridder per facet. nfacets=1, wprojplanes > 1 : Pure W-Projection and no facetting nfacets=1, wprojplanes=1 : Same as standard,ft,gridft

A combination of facetting and W-Projection is relevant only for very large fields of view.

mosaic

[A-Projection with azimuthally symmetric beams without]

sidelobes, beam rotation or squint correction. Gridding convolution functions per visibility are computed from FTs of PB models per antenna. This gridder can be run on single fields as well as mosaics.

VLA : PB polynomial fit model (Napier and Rots, 1982) EVLA : PB polynomial fit model (Perley, 2015) ALMA : Airy disks for a 10.7m dish (for 12m dishes) and

6.25m dish (for 7m dishes) each with 0.75m blockages (Hunter/Brogan 2011). Joint mosaic imaging supports heterogeneous arrays for ALMA.

Typical gridding convolution function support sizes are between 7 and 50 depending on the desired accuracy (given by the uv cell size or image field of view).

[This mode can also be invoked using 'mosaicft' or 'ftmosaic']

awproject
[A-Projection with azimuthally asymmetric beams and]

including beam rotation, squint correction, conjugate frequency beams and W-projection. [Bhatnagar et.al, 2008]

Gridding convolution functions are computed from aperture illumination models per antenna and optionally combined with W-Projection kernels and a prolate spheroid. This gridder can be run on single fields as well as mosaics.

VLA
[Uses ray traced model (VLA and EVLA) including feed] leg and subreflector shadows, off-axis feed location (for beam squint and other polarization effects), and a Gaussian fit for the feed beams (Ref: Brisken 2009)

ALMA
[Similar ray-traced model as above (but the correctness] of its polarization properties remains un-verified).

Typical gridding convolution function support sizes are between 7 and 50 depending on the desired accuracy (given by the uv cell size or image field of view). When combined with W-Projection they can be significantly larger.

[This mode can also be invoked using 'awprojectft']

imagemosaic
[(untested implementation)] Grid and iFT each pointing separately and combine the images as a linear mosaic (weighted by a PB model) in the image domain before a joint minor cycle.

VLA/ALMA PB models are same as for gridder='mosaicft'

—— Notes on PB models :

(1) **Several different sources of PB models are used in the modes**
listed above. This is partly for reasons of algorithmic flexibility and partly due to the current lack of a common beam model repository or consensus on what beam models are most appropriate.

(2) **For ALMA and gridder='mosaic', ray-traced (TICRA) beams**
are also available via the vpmanager tool. For example, call the following before the tclean run.

vp.setpbimage(telescope="ALMA", compleximage='/home/casa/data/trunk/alma/responses/ALMA_0_DV__0_0_3
antnames=['DV'+'%02d'%k for k in range(25)]) vp.saveastable('mypb.tab') Then, supply vptable='mypb.tab' to tclean. (Currently this will work only for non-parallel runs)

—— Note on PB masks :

In tclean, A-Projection gridders (mosaic and awproject) produce a .pb image and use the 'pblimit' subparameter to decide normalization cutoffs and construct an internal T/F mask

in the .pb and .image images. However, this T/F mask cannot directly be used during deconvolution (which needs a 1/0 mask). There are two options for making a pb based deconvolution mask.

– Run tclean with niter=0 to produce the .pb, construct a 1/0 image

with the desired threshold (using ia.open('newmask.im'); ia.calc('iif("xxx.pb">0.3,1.0,0.0)');ia.close() for example), and supply it via the 'mask' parameter in a subsequent run (with calcres=F and calcpsf=F to restart directly from the minor cycle).

– Run tclean with usemask='pb' for it to automatically construct

a 1/0 mask from the internal T/F mask from .pb at a fixed 0.2 threshold.

—– Making PBs for gridders other than mosaic,awproject

After the PSF generation, a PB is constructed using the same models used in gridder='mosaic' but just evaluated in the image domain without consideration to weights.

facets Number of facets on a side

A set of (facets x facets) subregions of the specified image are gridded separately using their respective phase centers (to minimize max W). Deconvolution is done on the joint full size image, using a PSF from the first subregion/facet.

chanchunks Number of channel chunks to grid separately

For large image cubes, the gridders can run into memory limits as they loop over all available image planes for each row of data accessed. To prevent this problem, we can grid subsets of channels in sequence so that at any given time only part of the image cube needs to be loaded into memory. This parameter controls the number of chunks to split the cube into.

Example : chanchunks = 4

[This feature is experimental and may have restrictions on how
chanchunks is to be chosen. For now, please pick chanchunks so that nchan/chanchunks is an integer.]

wprojplanes Number of distinct w-values at which to compute and use different
gridding convolution functions for W-Projection

An appropriate value of wprojplanes depends on the presence/absence of a bright source far from the phase center, the desired dynamic range of an image in the presence of a bright far out source, the maximum w-value in the measurements, and the desired trade off between accuracy and computing cost.

As a (rough) guide, VLA L-Band D-config may require a value of 128 for a source 30arcmin away from the phase center. A-config may require 1024 or more. To converge to an appropriate value, try starting with 128 and then increasing it if artifacts persist. W-term artifacts (for the VLA) typically look like arc-shaped smears in a synthesis image or a shift in source position between images made at different times. These artifacts are more pronounced the further the source is from the phase center.

There is no harm in simply always choosing a large value (say, 1024) but there will be a significant performance cost to doing so, especially for gridder='awproject' where it is combined with A-Projection.

wprojplanes=-1 is an option for gridder='widefield' or 'wproject' in which the number of planes is automatically computed.

vptable vpmanager

vptable=""
[Choose default beams for different telescopes] ALMA : Airy disks EVLA : old VLA models.

Other primary beam models can be chosen via the vpmanager tool.

Step 1 : Set up the vpmanager tool and save its state in a table

vp.setpbpoly(telescope='EVLA', coeff=[1.0, -1.529e-3, 8.69e-7, -1.88e-10])
vp.saveastable('myvp.tab')

Step 2 : Supply the name of that table in tclean.

tclean(...., vptable='myvp.tab',....)

Please see the documentation for the vpmanager for more details on how to choose different beam models. Work is in progress to update the defaults for EVLA and ALMA.

Note

[AWProjection currently does not use this mechanism to choose] beam models. It instead uses ray-traced beams computed from parameterized aperture illumination functions, which are not available via the vpmanager. So, gridder='awproject' does not allow the user to set this parameter.

usepointing Use the pointing table to determine where the beam are for mosaic gridder; if False then phasecenters of the fields selected are used to determine direction of each mosaic pointing. mosweight When doing Brigg's style weighting (including uniform) to perform the weight density calculation for each field indepedently if True. If False the weight density is calculated from the average uv distribution of all the fields. aterm Use aperture illumination functions during gridding

This parameter turns on the A-term of the AW-Projection gridder. Gridding convolution functions are constructed from aperture illumination function models of each antenna.

psterm Use prolate spheroidal during gridding wbawp Use frequency dependent A-terms

Scale aperture illumination functions appropriately with frequency when gridding and combining data from multiple channels.

conjbeams Use conjugate frequency for wideband A-terms

While gridding data from one frequency channel, choose a convolution function from a 'conjugate' frequency such that the resulting baseline primary beam is approximately constant across frequency. For a system in which the primary beam scales with frequency, this step will eliminate instrumental spectral structure from the measured data and leave only the sky spectrum for the minor cycle to model and reconstruct [Bhatnagar et.al,2013].

As a rough guideline for when this is relevant, a source at the half power point of the PB at the center frequency will see an artificial spectral index of -1.4 due to the frequency dependence of the PB [Sault and Wieringa, 1994]. If left uncorrected during gridding, this spectral structure must be modeled in the minor cycle (using the mtmfs algorithm) to avoid dynamic range limits (of a few hundred for a 2:1 bandwidth).

cfcache Convolution function cache directory name

Name of a directory in which to store gridding convolution functions. This cache is filled at the beginning of an imaging run. This step can be time consuming but the cache can be reused across multiple imaging runs that use the same image parameters (cell size, field-of-view, spectral data selections, etc).

By default, cfcache = imagename + '.cf'

computepastep At what parallactic angle interval to recompute aperture
illumination functions (deg)

This parameter controls the accuracy of the aperture illumination function used with AProjection for alt-az mount dishes where the AIF rotates on the sky as the synthesis image is built up.

rotatepastep At what parallactic angle interval to rotate nearest
aperture illumination function (deg)

Instead of recomputing the AIF for every timestep's parallactic angle, the nearest existing AIF is picked and rotated in steps of this amount.

For example, computepastep=360.0 and rotatepastep=5.0 will compute the AIFs at only the starting parallactic angle and all other timesteps will use a rotated version of that AIF at the nearest 5.0 degree point.

pblimit PB gain level at which to cut off normalizations

Divisions by .pb during normalizations have a cut off at a .pb gain level given by pblimit. Outside this limit, image values are set to zero. Additionally, by default, an internal T/F mask is applied to the .pb, .image and .residual images to mask out (T) all invalid pixels outside the pblimit area.

Note
[This internal T/F mask cannot be used as a deconvolution mask.] To do so, please follow the steps listed above in the Notes for the 'gridder' parameter.

Note
[To prevent the internal T/F mask from appearing in anything other] than the .pb and .image.pbcor images, 'pblimit' can be set to a negative number. The absolute value will still be used as a valid 'pblimit'. A tclean restart using existing output images on disk that already have this T/F mask in the .residual and .image but only pblimit set to a negative value, will remove this mask after the next major cycle.

normtype Normalization type (flatnoise, flatsky, pbsquare)

Gridded (and FT'd) images represent the PB-weighted sky image. Qualitatively it can be approximated as two instances of the PB applied to the sky image (one naturally present in the data and one introduced during gridding via the convolution functions).

xxx.weight : Weight image approximately equal to sum (square (pb)) xxx.pb : Primary beam calculated as sqrt (xxx.weight)

normtype='flatnoise'
[Divide the raw image by sqrt(.weight) so that] the input to the minor cycle represents the product of the sky and PB. The noise is 'flat' across the region covered by each PB.

normtype='flatsky'
[Divide the raw image by .weight so that the input] to the minor cycle represents only the sky. The noise is higher in the outer regions of the primary beam where the sensitivity is low.

normtype='pbsquare'
[No normalization after gridding and FFT.] The minor cycle sees the sky times pb square

deconvolver Name of minor cycle algorithm (hogbom,clark,multiscale,mtmfs,mem,clarkstokes)

Each of the following algorithms operate on residual images and psfs from the gridder and produce output model and restored images. Minor cycles stop and a major cycle is triggered when cyclethreshold or cycleniter are reached. For all methods, components are picked from the entire extent of the image or (if specified) within a mask.

hogbom
[An adapted version of Hogbom Clean [Hogbom, 1974]]

- Find the location of the peak residual

- Add this delta function component to the model image
- Subtract a scaled and shifted PSF of the same size as the image from regions of the residual image where the two overlap.
- Repeat

clark
[An adapted version of Clark Clean [Clark, 1980]]

- Find the location of max(I^2+Q^2+U^2+V^2)
- Add delta functions to each stokes plane of the model image
- Subtract a scaled and shifted PSF within a small patch size from regions of the residual image where the two overlap.
- After several iterations trigger a Clark major cycle to subtract components from the visibility domain, but without de-gridding.
- Repeat

(Note
['clark' maps to imagermode='' in the old clean task.]

'clark_exp' is another implementation that maps to
imagermode='mosaic' or 'csclean' in the old clean task but the behavior is not identical. For now, please use deconvolver='hogbom' if you encounter problems.)

clarkstokes : Clark Clean operating separately per Stokes plane

(Note : 'clarkstokes_exp' is an alternate version. See above.)

multiscale
[MultiScale Clean [Cornwell, 2008]]

- Smooth the residual image to multiple scale sizes
- Find the location and scale at which the peak occurs
- Add this multiscale component to the model image
- Subtract a scaled,smoothed,shifted PSF (within a small patch size per scale) from all residual images
- Repeat from step 2

mtmfs
[Multi-term (Multi Scale) Multi-Frequency Synthesis [Rau and Cornwell, 2011]]

- Smooth each Taylor residual image to multiple scale sizes
- Solve a NTxNT system of equations per scale size to compute Taylor coefficients for components at all locations
- **Compute gradient chi-square and pick the Taylor coefficients**
  and scale size at the location with maximum reduction in chi-square
- Add multi-scale components to each Taylor-coefficient model image
- Subtract scaled,smoothed,shifted PSF (within a small patch size per scale) from all smoothed Taylor residual images
- Repeat from step 2

mem

[Maximum Entropy Method [Cornwell and Evans, 1985]]

- Iteratively solve for values at all individual pixels via the MEM method. It minimizes an objective function of

  chi-square plus entropy (here, a measure of difference

  between the current model and a flat prior model).

  (Note

  [This MEM implementation is not very robust.] Improvements will be made in the future.)

scales List of scale sizes (in pixels) for multi-scale and mtmfs algorithms.

–> scales=[0,6,20] This set of scale sizes should represent the sizes (diameters in units of number of pixels) of dominant features in the image being reconstructed.

The smallest scale size is recommended to be 0 (point source), the second the size of the synthesized beam and the third 3-5 times the synthesized beam, etc. For example, if the synthesized beam is 10" FWHM and cell=2",try scales = [0,5,15].

For numerical stability, the largest scale must be smaller than the image (or mask) size and smaller than or comparable to the scale corresponding to the lowest measured spatial frequency (as a scale size much larger than what the instrument is sensitive to is unconstrained by the data making it harder to recovery from errors during the minor cycle).

nterms Number of Taylor coefficients in the spectral model

- nterms=1 : Assume flat spectrum source
- nterms=2 : Spectrum is a straight line with a slope
- nterms=N : A polynomial of order N-1

From a Taylor expansion of the expression of a power law, the spectral index is derived as alpha = taylorcoeff_1 / taylorcoeff_0

Spectral curvature is similarly derived when possible.

The optimal number of Taylor terms depends on the available signal to noise ratio, bandwidth ratio, and spectral shape of the source as seen by the telescope (sky spectrum x PB spectrum).

nterms=2 is a good starting point for wideband EVLA imaging and the lower frequency bands of ALMA (when fractional bandwidth is greater than 10%) and if there is at least one bright source for which a dynamic range of greater than few 100 is desired.

Spectral artifacts for the VLA often look like spokes radiating out from a bright source (i.e. in the image made with standard mfs imaging). If increasing the number of terms does not eliminate these artifacts, check the data for inadequate bandpass calibration. If the source is away from the pointing center, consider including wide-field corrections too.

(Note

[In addition to output Taylor coefficient images .tt0,.tt1,etc] images of spectral index (.alpha), an estimate of error on spectral index (.alpha.error) and spectral curvature (.beta, if nterms is greater than 2) are produced. - These alpha, alpha.error and beta images contain

internal T/F masks based on a threshold computed as peakresidual/10. Additional masking based on

.alpha/.alpha.error may be desirable.

- .alpha.error is a purely empirical estimate derived from the propagation of error during the division of two noisy numbers (alpha = xx.tt1/xx.tt0) where the 'error' on tt1 and tt0 are simply the values picked from the corresponding residual images. The absolute value of the error is not always accurate and it is best to interpret the errors across the image only in a relative sense.)

smallscalebias A numerical control to bias the solution towards smaller scales.

The peak from each scale's smoothed residual is multiplied by (1 - smallscalebias * scale/maxscale) to increase or decrease the amplitude relative to other scales, before the scale with the largest peak is chosen.

smallscalebias=0.6 (default) applies a slight bias towards small
scales, ranging from 1.0 for a point source to 0.4 for the largest scale size

Values larger than 0.6 will bias the solution towards smaller scales. Values smaller than 0.6 will tend towards giving all scales equal weight.

restoration e.

Construct a restored image : imagename.image by convolving the model image with a clean beam and adding the residual image to the result. If a restoringbeam is specified, the residual image is also smoothed to that target resolution before adding it in.

If a .model does not exist, it will make an empty one and create the restored image from the residuals (with additional smoothing if needed). With algorithm='mtmfs', this will construct Taylor coefficient maps from the residuals and compute .alpha and .alpha.error.

restoringbeam ize to use.

- restoringbeam=" or ['] A Gaussian fitted to the PSF main lobe (separately per image plane).
- restoringbeam='10.0arcsec' Use a circular Gaussian of this width for all planes
- restoringbeam=['8.0arcsec','10.0arcsec','45deg'] Use this elliptical Gaussian for all planes
- restoringbeam='common' Automatically estimate a common beam shape/size appropriate for all planes.

Note
[For any restoring beam different from the native resolution] the model image is convolved with the beam and added to residuals that have been convolved to the same target resolution.

pbcor the output restored image

A new image with extension .image.pbcor will be created from the evaluation of .image / .pb for all pixels above the specified pblimit.

Note
[Stand-alone PB-correction can be triggered by re-running] tclean with the appropriate imagename and with niter=0, calcpsf=False, calcres=False, pbcor=True, vptable='vp.tab' (where vp.tab is the name of the vpmanager file.

See the inline help for the 'vptable' parameter)

Note
[Multi-term PB correction that includes a correction for the] spectral index of the PB has not been enabled for the 4.7 release. Please use the widebandpbcor task instead. (Wideband PB corrections are required when the amplitude of the

brightest source is known accurately enough to be sensitive to the difference in the PB gain between the upper and lower end of the band at its location. As a guideline,

the artificial spectral index due to the PB is -1.4 at the 0.5 gain level and less than -0.2 at the 0.9 gain level at the middle frequency)

outlierfile Name of outlier-field image definitions

A text file containing sets of parameter=value pairs, one set per outlier field.

Example : outlierfile='outs.txt'

Contents of outs.txt :

imagename=tst1 nchan=1 imsize=[80,80] cell=[8.0arcsec,8.0arcsec] phasecenter=J2000 19:58:40.895 +40.55.58.543 mask=circle[[40pix,40pix],10pix]

imagename=tst2 nchan=1 imsize=[100,100] cell=[8.0arcsec,8.0arcsec] phasecenter=J2000 19:58:40.895 +40.56.00.000 mask=circle[[60pix,60pix],20pix]

The following parameters are currently allowed to be different between the main field and the outlier fields (i.e. they will be recognized if found in the outlier text file). If a parameter is not listed, the value is picked from what is defined in the main task input.

imagename, imsize, cell, phasecenter, startmodel, mask specmode, nchan, start, width, nterms, reffreq, gridder, deconvolver, wprojplanes

Note

['specmode' is an option, so combinations of mfs and cube]

for different image fields, for example, are supported.

'deconvolver' and 'gridder' are also options that allow different

imaging or deconvolution algorithm per image field.

For example, multiscale with wprojection and 16 w-term planes on the main field and mtmfs with nterms=3 and wprojection with 64 planes on a bright outlier source for which the frequency dependence of the primary beam produces a strong effect that must be modeled. The traditional alternative to this approach is to first image the outlier, subtract it out of the data (uvsub) and then image the main field.

Note

[If you encounter a use-case where some other parameter needs] to be allowed in the outlier file (and it is logical to do so), please send us feedback. The above is an initial list.

weighting Weighting scheme (natural,uniform,briggs,superuniform,radial)

During gridding of the dirty or residual image, each visibility value is multiplied by a weight before it is accumulated on the uv-grid. The PSF's uv-grid is generated by gridding only the weights (weightgrid).

weighting='natural'

[Gridding weights are identical to the data weights] from the MS. For visibilities with similar data weights, the weightgrid will follow the sample density pattern on the uv-plane. This weighting scheme provides the maximum imaging sensitivity at the expense of a possibly fat PSF with high sidelobes. It is most appropriate for detection experiments where sensitivity is most important.

weighting='uniform'
[Gridding weights per visibility data point are the] original data weights divided by the total weight of all data points that map to the same uv grid cell : ' data_weight / total_wt_per_cell '.

The weightgrid is as close to flat as possible resulting in a PSF with a narrow main lobe and suppressed sidelobes. However, since heavily sampled areas of the uv-plane get down-weighted, the imaging sensitivity is not as high as with natural weighting. It is most appropriate for imaging experiments where a well behaved PSF can help the reconstruction.

weighting='briggs'
[Gridding weights per visibility data point are given by] 'data_weight / (A / total_wt_per_cell + B) ' where A and B vary according to the 'robust' parameter.

robust = -2.0 maps to A=1,B=0 or uniform weighting. robust = +2.0 maps to natural weighting. (robust=0.5 is equivalent to robust=0.0 in AIPS IMAGR.)

Robust/Briggs weighting generates a PSF that can vary smoothly between 'natural' and 'uniform' and allow customized trade-offs between PSF shape and imaging sensitivity.

weighting='superuniform'
[This is similar to uniform weighting except that]

the total_wt_per_cell is replaced by the total_wt_within_NxN_cells around the uv cell of interest. (N = subparameter 'npixels')

This method tends to give a PSF with inner sidelobes that are suppressed as in uniform weighting but with far-out sidelobes closer to natural weighting. The peak sensitivity is also closer to natural weighting.

weighting='radial' : Gridding weights are given by ' data_weight * uvdistance '

This method approximately minimizes rms sidelobes for an east-west synthesis array.

For more details on weighting please see Chapter3 of Dan Briggs' thesis (http://www.aoc.nrao.edu/dissertations/dbriggs)

robust Robustness parameter for Briggs weighting.

robust = -2.0 maps to uniform weighting. robust = +2.0 maps to natural weighting. (robust=0.5 is equivalent to robust=0.0 in AIPS IMAGR.)

npixels Number of pixels to determine uv-cell size for super-uniform weighting

(0 defaults to -/+ 3 pixels)

npixels – uv-box used for weight calculation

a box going from -npixel/2 to +npixel/2 on each side

around a point is used to calculate weight density.

npixels=2 goes from -1 to +1 and covers 3 pixels on a side.

npixels=0 implies a single pixel, which does not make sense for
superuniform weighting. Therefore, if npixels=0 it will be forced to 6 (or a box of -3pixels to +3pixels) to cover 7 pixels on a side.

uvtaper uv-taper on outer baselines in uv-plane

Apply a Gaussian taper in addition to the weighting scheme specified via the 'weighting' parameter. Higher spatial frequencies are weighted down relative to lower spatial frequencies to suppress artifacts arising from poorly sampled areas of the uv-plane. It is equivalent to smoothing the PSF obtained by other weighting schemes and can be specified either as a Gaussian in uv-space (eg. units of lambda) or as a Gaussian in the image domain (eg. angular units like arcsec).

uvtaper = [bmaj, bmin, bpa]

NOTE: the on-sky FWHM in arcsec is roughly the uv taper/200 (klambda). default: uvtaper=[]; no Gaussian taper applied

example: uvtaper=['5klambda'] circular taper

FWHM=5 kilo-lambda
uvtaper=['5klambda','3klambda','45.0deg'] uvtaper=['10arcsec'] on-sky FWHM 10 arcseconds uvtaper=['300.0'] default units are lambda

in aperture plane

niter Maximum number of iterations

A stopping criterion based on total iteration count.

Iterations are typically defined as the selecting one flux component and partially subtracting it out from the residual image.

niter=0 : Do only the initial major cycle (make dirty image, psf, pb, etc)

niter larger than zero : Run major and minor cycles.

Note : Global stopping criteria vs major-cycle triggers

In addition to global stopping criteria, the following rules are used to determine when to terminate a set of minor cycle iterations and trigger major cycles [derived from Cotton-Schwab Clean, 1984]

'cycleniter'
[controls the maximum number of iterations per image] plane before triggering a major cycle.

'cyclethreshold'
[Automatically computed threshold related to the]

max sidelobe level of the PSF and peak residual.

Divergence, detected as an increase of 10% in peak residual from the minimum so far (during minor cycle iterations)

The first criterion to be satisfied takes precedence.

Note
[Iteration counts for cubes or multi-field images :]

For images with multiple planes (or image fields) on which the deconvolver operates in sequence, iterations are counted across all planes (or image fields). The iteration count is compared with 'niter' only after all channels/planes/fields have completed their minor cycles and exited either due to 'cycleniter' or 'cyclethreshold'. Therefore, the actual number of iterations reported in the logger can sometimes be larger than the user specified value in 'niter'. For example, with niter=100, cycleniter=20,nchan=10,threshold=0, a total of 200 iterations will be done in the first set of minor cycles before the total is compared with niter=100 and it exits.

Note

[Additional global stopping criteria include]

- no change in peak residual across two major cycles
- a 50% or more increase in peak residual across one major cycle

gain Loop gain

Fraction of the source flux to subtract out of the residual image for the CLEAN algorithm and its variants.

A low value (0.2 or less) is recommended when the sky brightness distribution is not well represented by the basis functions used by the chosen deconvolution algorithm. A higher value can be tried when there is a good match between the true sky brightness structure and the basis function shapes. For example, for extended emission, multiscale clean with an appropriate set of scale sizes will tolerate a higher loop gain than Clark clean (for example).

threshold Stopping threshold (number in units of Jy, or string)

A global stopping threshold that the peak residual (within clean mask) across all image planes is compared to.

threshold = 0.005 : 5mJy threshold = '5.0mJy'

Note

[A 'cyclethreshold' is internally computed and used as a major cycle]

trigger. It is related what fraction of the PSF can be reliably used during minor cycle updates of the residual image. By default the minor cycle iterations terminate once the peak residual reaches the first sidelobe level of the brightest source.

'cyclethreshold' is computed as follows using the settings in
parameters 'cyclefactor','minpsffraction','maxpsffraction','threshold' :

psf_fraction = max_psf_sidelobe_level * 'cyclefactor' psf_fraction = max(psf_fraction, 'minpsffraction'); psf_fraction = min(psf_fraction, 'maxpsffraction'); cyclethreshold = peak_residual * psf_fraction cyclethreshold = max(cyclethreshold, 'threshold')

If nsigma is set (>0.0), the N-sigma threshold is calculated (see the description under nsigma), then cyclethreshold is further modified as,

cyclethreshold = max(cyclethreshold, nsgima_threshold)

'cyclethreshold' is made visible and editable only in the interactive GUI when tclean is run with interactive=True.

nsigma Multiplicative factor for rms-based threshold stopping

N-sigma threshold is calculated as nsigma * rms value per image plane determined from a robust statistics. For nsigma > 0.0, in a minor cycle, a maximum of the two values, the N-sigma threshold and cyclethreshold, is used to trigger a major cycle (see also the descreption under 'threshold'). Set nsigma=0.0 to preserve the previous tclean behavior without this feature.

cycleniter Maximum number of minor-cycle iterations (per plane) before triggering
a major cycle

For example, for a single plane image, if niter=100 and cycleniter=20, there will be 5 major cycles after the initial one (assuming there is no threshold based stopping criterion). At each major cycle boundary, if the number of iterations left over (to reach niter) is less than cycleniter, it is set to the difference.

Note

[cycleniter applies per image plane, even if cycleniter x nplanes] gives a total number of iterations

greater than 'niter'. This is to preserve consistency across image planes within one set of minor cycle iterations.

cyclefactor Scaling on PSF sidelobe level to compute the minor-cycle stopping threshold.

Please refer to the Note under the documentation for 'threshold' that discussed the calculation of 'cyclethreshold'

cyclefactor=1.0 results in a cyclethreshold at the first sidelobe level of the brightest source in the residual image before the minor cycle starts.

cyclefactor=0.5 allows the minor cycle to go deeper. cyclefactor=2.0 triggers a major cycle sooner.

minpsffraction PSF fraction that marks the max depth of cleaning in the minor cycle

Please refer to the Note under the documentation for 'threshold' that discussed the calculation of 'cyclethreshold'

For example, minpsffraction=0.5 will stop cleaning at half the height of the peak residual and trigger a major cycle earlier.

maxpsffraction PSF fraction that marks the minimum depth of cleaning in the minor cycle

Please refer to the Note under the documentation for 'threshold' that discussed the calculation of 'cyclethreshold'

For example, maxpsffraction=0.8 will ensure that at least the top 20 percent of the source will be subtracted out in the minor cycle even if the first PSF sidelobe is at the 0.9 level (an extreme example), or if the cyclefactor is set too high for anything to get cleaned.

interactive Modify masks and parameters at runtime

interactive=True will trigger an interactive GUI at every major cycle boundary (after the major cycle and before the minor cycle).

The interactive mode is currently not available for parallel cube imaging (please also refer to the Note under the documentation for 'parallel' below).

Options for runtime parameter modification are :

Interactive clean mask

[Draw a 1/0 mask (appears as a contour) by hand.] If a mask is supplied at the task interface or if automasking is invoked, the current mask is displayed in the GUI and is available for manual editing.

Note

[If a mask contour is not visible, please] check the cursor display at the bottom of GUI to see which parts of the mask image have ones and zeros. If the entire mask=1 no contours will be visible.

Operation buttons

[– Stop execution now (restore current model and exit)]

– Continue on until global stopping criteria are reached

without stopping for any more interaction

– Continue with minor cycles and return for interaction

after the next major cycle.

Iteration control : – max cycleniter : Trigger for the next major cycle

The display begins with [min(cycleniter, niter - itercount)] and can be edited by hand.

—iterations left : The display begins with [niter-itercount] and can be edited to increase or decrease the total allowed niter.

– threshold : Edit global stopping threshold

—cyclethreshold : The display begins with the automatically computed value (see Note in help for 'threshold'), and can be edited by hand.

All edits will be reflected in the log messages that appear once minor cycles begin.

[For scripting purposes, replacing True/False with 1/0 will get tclean to return an imaging summary dictionary to python]

usemask Type of mask(s) to be used for deconvolution

user: (default) mask image(s) or user specified region file(s) or string CRTF expression(s)
subparameters: mask, pbmask

pb: primary beam mask
subparameter: pbmask

Example: usemask="pb", pbmask=0.2
Construct a mask at the 0.2 pb gain level. (Currently, this option will work only with gridders that produce .pb (i.e. mosaic and awproject) or if an externally produced .pb image exists on disk)

auto-multithresh
[auto-masking by multiple thresholds for deconvolution]

subparameters
[sidelobethreshold, noisethreshold, lownoisethreshold, negativethrehsold, smoothfactor,] minbeamfrac, cutthreshold, pbmask, growiterations, dogrowprune, minpercentchange, verbose

if pbmask is >0.0, the region outside the specified pb gain level is excluded from image statistics in determination of the threshold.

Note: By default the intermediate mask generated by automask at each deconvolution cycle is over-written in the next cycle but one can save them by setting the environment variable, SAVE_ALL_AUTOMASKS="true". (e.g. in the CASA prompt, os.environ['SAVE_ALL_AUTOMASKS']="true") The saved CASA mask image name will be imagename.mask.autothresh#, where # is the iteration cycle number.

mask Mask (a list of image name(s) or region file(s) or region string(s)

The name of a CASA image or region file or region string that specifies a 1/0 mask to be used for deconvolution. Only locations with value 1 will be considered for the centers of flux components in the minor cycle. If regions specified fall completely outside of the image, tclean will throw an error.

Manual mask options/examples :

mask='xxx.mask'
[Use this CASA image named xxx.mask and containing] ones and zeros as the mask. If the mask is only different in spatial coordinates from what is being made it will be resampled to the target coordinate system before being used. The mask has to have the same shape in velocity and Stokes planes as the output image. Exceptions are single velocity and/or single Stokes plane masks. They will be expanded to cover all velocity and/or Stokes planes of the output cube.

[Note

[If an error occurs during image resampling or] if the expected mask does not appear, please try using tasks 'imregrid' or 'makemask' to resample the mask image onto a CASA image with the target shape and coordinates and supply it via the 'mask' parameter.]

mask='xxx.crtf'

[A text file with region strings and the following on the first line] (#CRTFv0 CASA Region Text Format version 0) This is the format of a file created via the viewer's region tool when saved in CASA region file format.

mask='circle[[40pix,40pix],10pix]' : A CASA region string.

mask=['xxx.mask','xxx.crtf', 'circle[[40pix,40pix],10pix]'] : a list of masks

Note

[Mask images for deconvolution must contain 1 or 0 in each pixel.] Such a mask is different from an internal T/F mask that can be held within each CASA image. These two types of masks are not automatically interchangeable, so please use the makemask task to copy between them if you need to construct a 1/0 based mask from a T/F one.

Note

[Work is in progress to generate more flexible masking options and] enable more controls.

pbmask Sub-parameter for usemask='auto-multithresh': primary beam mask

Examples

[pbmask=0.0 (default, no pb mask)] pbmask=0.2 (construct a mask at the 0.2 pb gain level)

sidelobethreshold Sub-parameter for "auto-multithresh": mask threshold based on sidelobe levels: sidelobethreshold * max_sidelobe_level * peak residual noisethreshold Sub-parameter for "auto-multithresh": mask threshold based on the noise level: noisethreshold * rms

The rms is calculated from MAD with rms = 1.4826*MAD.

lownoisethreshold Sub-parameter for "auto-multithresh": mask threshold to grow previously masked regions via binary dilation: lownoisethreshold * rms in residual image

The rms is calculated from MAD with rms = 1.4826*MAD.

negativethreshold Sub-parameter for "auto-multithresh": mask threshold for negative features: -1.0* negativethreshold * rms

The rms is calculated from MAD with rms = 1.4826*MAD.

smoothfactor Sub-parameter for "auto-multithresh": smoothing factor in a unit of the beam minbeamfrac Sub-parameter for "auto-multithresh": minimum beam fraction in size to prune masks smaller than mimbeamfrac * beam

<=0.0 : No pruning

cutthreshold Sub-parameter for "auto-multithresh": threshold to cut the smoothed mask to create a final mask: cutthreshold * peak of the smoothed mask growiterations Sub-parameter for "auto-multithresh": Maximum number of iterations to perform using binary dilation for growing the mask dogrowprune Experimental sub-parameter for "auto-multithresh": Do pruning on the grow mask minpercentchange If the change in the mask size in a particular channel is less than minpercentchange, stop masking that channel in subsequent cycles. This check is only applied when noise based threshold is used and when the previous clean major cycle had a cyclethreshold value equal to the clean threshold. Values equal to -1.0 (or any value less than 0.0) will turn off this check (the default). Automask will still stop masking if the current channel mask is an empty mask and the noise threshold was used to determine the mask. verbose he summary of automasking at the end of each automasking process

is printed in the logger. Following information per channel will be listed in the summary.

chan: channel number masking?: F - stop updating automask for the subsequent iteration cycles RMS: robust rms noise peak: peak in residual image thresh_type: type of threshold used (noise or sidelobe) thresh_value: the value of threshold used N_reg: number of the automask regions N_pruned: number of the automask regions removed by pruning N_grow: number of the grow mask regions N_grow_pruned: number of the grow mask regions removed by pruning N_neg_pix: number of pixels for negative mask regions

Note that for a large cube, extra logging may slow down the process.

restart images (and start from an existing model image)
or automatically increment the image name and make a new image set.

True
[Re-use existing images. If imagename.model exists the subsequent]

run will start from this model (i.e. predicting it using current gridder settings and starting from the residual image). Care must be taken when combining this option with startmodel. Currently, only one or the other can be used.

startmodel='', imagename.model exists :

- Start from imagename.model

startmodel='xxx', imagename.model does not exist :

- Start from startmodel

startmodel='xxx', imagename.model exists :

- Exit with an error message requesting the user to pick only one model. This situation can arise when doing one run with startmodel='xxx' to produce an output imagename.model that includes the content of startmodel, and wanting to restart a second run to continue deconvolution. Startmodel should be set to '' before continuing.

If any change in the shape or coordinate system of the image is desired during the restart, please change the image name and use the startmodel (and mask) parameter(s) so that the old model (and mask) can be regridded to the new coordinate system before starting.

False
[A convenience feature to increment imagename with '_1', '_2',]

etc as suffixes so that all runs of tclean are fresh starts (without having to change the imagename parameter or delete images).

This mode will search the current directory for all existing imagename extensions, pick the maximum, and adds 1. For imagename='try' it will make try.psf, try_2.psf, try_3.psf, etc.

This also works if you specify a directory name in the path : imagename='outdir/try'. If './outdir' does not exist, it will create it. Then it will search for existing filenames inside that directory.

If outlier fields are specified, the incrementing happens for each of them (since each has its own 'imagename'). The counters are synchronized across imagefields, to make it easier to match up sets of output images. It adds 1 to the 'max id' from all outlier names on disk. So, if you do two runs with only the main field

(imagename='try'), and in the third run you add an outlier with imagename='outtry', you will get the following image names for the third run : 'try_3' and 'outtry_3' even though 'outry' and 'outtry_2' have not been used.

savemodel Options to save model visibilities (none, virtual, modelcolumn)

Often, model visibilities must be created and saved in the MS to be later used for self-calibration (or to just plot and view them).

none
[Do not save any model visibilities in the MS. The MS is opened] in read-only mode.

Model visibilities can be predicted in a separate step by restarting tclean with niter=0,savemodel=virtual or modelcolumn and not changing any image names so that it finds the .model on disk (or by changing imagename and setting startmodel to the original imagename).

virtual
[In the last major cycle, save the image model and state of the] gridder used during imaging within the SOURCE subtable of the MS. Images required for de-gridding will also be stored internally. All future references to model visibilities will activate the (de)gridder to compute them on-the-fly. This mode is useful when the dataset is large enough that an additional model data column on disk may be too much extra disk I/O, when the gridder is simple enough that on-the-fly recomputing of the model visibilities is quicker than disk I/O.

modelcolumn
[In the last major cycle, save predicted model visibilities] in the MODEL_DATA column of the MS. This mode is useful when the de-gridding cost to produce the model visibilities is higher than the I/O required to read the model visibilities from disk. This mode is currently required for gridder='awproject'. This mode is also required for the ability to later pull out model visibilities from the MS into a python array for custom processing.

Note 1
[The imagename.model image on disk will always be constructed] if the minor cycle runs. This savemodel parameter applies only to model visibilities created by de-gridding the model image.

Note 2
[It is possible for an MS to have both a virtual model] as well as a model_data column, but under normal operation, the last used mode will get triggered. Use the delmod task to clear out existing models from an MS if confusion arises.

calcres Calculate initial residual image

This parameter controls what the first major cycle does.

calcres=False with niter greater than 0 will assume that a .residual image already exists and that the minor cycle can begin without recomputing it.

calcres=False with niter=0 implies that only the PSF will be made and no data will be gridded.

calcres=True requires that calcpsf=True or that the .psf and .sumwt images already exist on disk (for normalization purposes).

Usage example
[For large runs (or a pipeline scripts) it may be] useful to first run tclean with niter=0 to create an initial .residual to look at and perhaps make a custom mask for. Imaging can be resumed without recomputing it.

calcpsf Calculate PSF

This parameter controls what the first major cycle does.

calcpsf=False will assume that a .psf image already exists and that the minor cycle can begin without recomputing it.

parallel Run major cycles in parallel (this feature is experimental)

Parallel tclean will run only if casa has already been started using mpirun. Please refer to HPC documentation for details on how to start this on your system.

Example : mpirun -n 3 -xterm 0 *which casa*

Continuum Imaging :

- Data are partitioned (in time) into NProc pieces
- Gridding/iFT is done separately per partition
- Images (and weights) are gathered and then normalized
- One non-parallel minor cycle is run
- Model image is scattered to all processes
- Major cycle is done in parallel per partition

Cube Imaging :

- Data and Image coordinates are partitioned (in freq) into NProc pieces
- Each partition is processed independently (major and minor cycles)
- All processes are synchronized at major cycle boundaries for convergence checks
- At the end, cubes from all partitions are concatenated along the spectral axis

Note 1

[Iteration control for cube imaging is independent per partition.]

- **There is currently no communication between them to synchronize**
  information such as peak residual and cyclethreshold. Therefore, different chunks may trigger major cycles at different levels.
- For cube imaging in parallel, there is currently no interactive masking.

(Proper synchronization of iteration control is work in progress.)

[1;42mRETURNS[1;m void

——— examples ————————————————————————

This is the first release of our refactored imager code. Although most features have been used and validated, there are many details that have not been thoroughly tested. Feedback will be much appreciated.

Usage Examples :

(A) A suite of test programs that demo all usable modes of tclean on small test datasets https://svn.cv.nrao.edu/svn/casa/branches/release-4_5/gcwrap/python/scripts/tests/test_refimager.py (B) A set of demo examples for ALMA imaging https://casaguides.nrao.edu/index.php/TCLEAN_and_ALMA

_info_group_ = 'imaging'

_info_desc_ = 'Parallelized tclean in consecutive time steps'

__call__(*vis=''*, *imageprefix=''*, *imagesuffix=''*, *ncpu=int(8)*, *twidth=int(1)*, *doreg=False*, *usephacenter=True*, *reftime=''*, *toTb=False*, *sclfactor=float(1.0)*, *subregion=''*, *docompress=False*, *overwrite=False*, *selectdata=True*, *field=''*, *spw=''*, *timerange=''*, *uvrange=''*, *antenna=''*, *scan=''*, *observation=''*, *intent=''*, *datacolumn='corrected'*, *imsize=[]*, *cell=[]*, *phasecenter=''*, *stokes='I'*, *projection='SIN'*, *startmodel=''*, *specmode='mfs'*, *reffreq=''*, *nchan=int(-1)*, *start=''*, *width=''*, *outframe='LSRK'*, *veltype='radio'*, *restfreq=[]*, *interpolation='linear'*, *gridder='standard'*, *facets=int(1)*, *chanchunks=int(1)*, *wprojplanes=int(1)*, *vptable=''*, *usepointing=False*, *mosweight=True*, *aterm=True*, *psterm=False*, *wbawp=True*, *conjbeams=True*, *cfcache=''*, *computepastep=float(360.0)*, *rotatepastep=float(360.0)*, *pblimit=float(0.2)*, *normtype='flatnoise'*, *deconvolver='hogbom'*, *scales=[]*, *nterms=int(2)*, *smallscalebias=float(0.6)*, *restoration=True*, *restoringbeam=[]*, *pbcor=False*, *outlierfile=''*, *weighting='natural'*, *robust=float(0.5)*, *npixels=int(0)*, *uvtaper=['']*, *niter=int(0)*, *gain=float(0.1)*, *threshold=float(0.0)*, *nsigma=float(0.0)*, *cycleniter=int(-1)*, *cyclefactor=float(1.0)*, *minpsffraction=float(0.05)*, *maxpsffraction=float(0.8)*, *interactive=False*, *usemask='user'*, *mask=''*, *pbmask=float(0.0)*, *sidelobethreshold=float(3.0)*, *noisethreshold=float(5.0)*, *lownoisethreshold=float(1.5)*, *negativethreshold=float(0.0)*, *smoothfactor=float(1.0)*, *minbeamfrac=float(0.3)*, *cutthreshold=float(0.01)*, *growiterations=int(75)*, *dogrowprune=True*, *minpercentchange=float(-1.0)*, *verbose=False*, *restart=True*, *savemodel='none'*, *calcres=True*, *calcpsf=True*, *parallel=False*)

suncasa.suncasatasks.ptclean.**ptclean**

suncasa.suncasatasks.ptclean6

Module Contents

Classes

_ptclean6	ptclean6 ---- Parallelized tclean in consecutive time steps

Attributes

_pc
ptclean6

suncasa.suncasatasks.ptclean6.**_pc**

class suncasa.suncasatasks.ptclean6._ptclean6

ptclean6 —- Parallelized tclean in consecutive time steps

Parallelized clean in consecutive time steps. Packed over CASA 6 tclean.

——— parameter descriptions ———————————————

vis Name(s) of input visibility file(s)

default: none; example: vis='ngc5921.ms'

vis=['ngc5921a.ms','ngc5921b.ms']; multiple MSes

imageprefix Prefix of output image names (usually useful in defining the output path) imagesuffix Suffix of output image names (usually useful in specifyting the image type, version, etc.) ncpu Number of cpu cores to use twidth Number of time pixels to average doreg True if use vla_prep to register the image usephacenter True if use the phacenter information from the measurement set (e.g., VLA); False to assume the phase center is at the solar disk center (EOVSA) reftime Reference time of the J2000 coordinates associated with the ephemeris target. e.g., "2012/03/03/12:00". This is used for helioimage2fits.py to find the solar x y offset in order to register the image. If not set, use the actual timerange of the image (default) toTb True if convert to brightness temperature sclfactor scale the brightness temperature up by its value subregion The name of a CASA region string

The name of a CASA image or region file or region string. Only locations within the region will output to the fits file. If regions specified fall completely outside of the image, ptclean6 will throw an error.

Manual mask options/examples :

subregion='box[[224pix,224pix],[288pix,288pix]]' : A CASA region string.

docompress True if compress the output fits files overwrite True if overwrite the image selectdata Enable data selection parameters. field to image or mosaic. Use field id(s) or name(s).

['go listobs' to obtain the list id's or names]

default: ''= all fields

If field string is a non-negative integer, it is assumed to be a field index otherwise, it is assumed to be a field name field='0~2'; field ids 0,1,2 field='0,4,5~7'; field ids 0,4,5,6,7 field='3C286,3C295'; field named 3C286 and 3C295 field = '3,4C*'; field id 3, all names starting with 4C For multiple MS input, a list of field strings can be used: field = ['0~2','0~4']; field ids 0-2 for the first MS and 0-4

for the second

field = '0~2'; field ids 0-2 for all input MSes

spw l window/channels

NOTE: channels de-selected here will contain all zeros if

selected by the parameter mode subparameters.

default: ''=all spectral windows and channels

spw='0~2,4'; spectral windows 0,1,2,4 (all channels) spw='0:5~61'; spw 0, channels 5 to 61 spw='<2'; spectral windows less than 2 (i.e. 0,1) spw='0,10,3:3~45'; spw 0,10 all channels, spw 3,

channels 3 to 45.

spw='0~2:2~6'; spw 0,1,2 with channels 2 through 6 in each. For multiple MS input, a list of spw strings can be used: spw=['0','0~3']; spw ids 0 for the first MS and 0-3 for the second spw='0~3' spw ids 0-3 for all input MS spw='3:10~20;50~60' for multiple channel ranges within spw id 3 spw='3:10~20;50~60,4:0~30' for different channel ranges for spw ids 3 and 4 spw='0:0~10,1:20~30,2:1;2;3'; spw 0, channels 0-10,

spw 1, channels 20-30, and spw 2, channels, 1,2 and 3

spw='1~4;6:15~48' for channels 15 through 48 for spw ids 1,2,3,4 and 6

timerange Range of time to select from data

default: '' (all); examples, timerange = 'YYYY/MM/DD/hh:mm:ss~YYYY/MM/DD/hh:mm:ss' Note: if YYYY/MM/DD is missing date defaults to first

day in data set

timerange='09:14:0~09:54:0' picks 40 min on first day timerange='25:00:00~27:30:00' picks 1 hr to 3 hr

30min on NEXT day

timerange='09:44:00' pick data within one integration
of time

timerange='> 10:24:00' data after this time For multiple MS input, a list of timerange strings can be used: timerange=['09:14:0~09:54:0','> 10:24:00'] timerange='09:14:0~09:54:0"; apply the same timerange for

all input MSes

uvrange Select data within uvrange (default unit is meters)
default: '' (all); example: uvrange='0~1000klambda'; uvrange from 0-1000 kilo-lambda uvrange='> 4klambda';uvranges greater than 4 kilo lambda For multiple MS input, a list of uvrange strings can be used: uvrange=['0~1000klambda','100~1000klamda'] uvrange='0~1000klambda'; apply 0-1000 kilo-lambda for all

input MSes

antenna Select data based on antenna/baseline

default: '' (all) If antenna string is a non-negative integer, it is assumed to be an antenna index, otherwise, it is considered an antenna name. antenna='5&6'; baseline between antenna index 5 and

index 6.

antenna='VA05&VA06'; baseline between VLA antenna 5
and 6.

antenna='5&6;7&8'; baselines 5-6 and 7-8 antenna='5'; all baselines with antenna index 5 antenna='05'; all baselines with antenna number 05

(VLA old name)

antenna='5,6,9'; all baselines with antennas 5,6,9
index number

For multiple MS input, a list of antenna strings can be used: antenna=['5','5&6']; antenna='5'; antenna index 5 for all input MSes antenna='!DV14'; use all antennas except DV14

scan Scan number range

default: '' (all) example: scan='1~5' For multiple MS input, a list of scan strings can be used: scan=['0~100','10~200'] scan='0~100; scan ids 0-100 for all input MSes

observation Observation ID range
default: '' (all) example: observation='1~5'

intent Scan Intent(s)

default: '' (all) example: intent='TARGET_SOURCE' example: intent='TARGET_SOURCE1,TARGET_SOURCE2' example: intent='TARGET_POINTING*'

datacolumn Data column to image (data or observed, corrected)
default:'corrected' (If 'corrected' does not exist, it will use 'data' instead)

imagename Pre-name of output images

example : imagename='try'

Output images will be (a subset of) :

try.psf - Point spread function try.residual - Residual image try.image - Restored image try.model - Model image (contains only flux components) try.sumwt - Single pixel image containing sum-of-weights.

(for natural weighting, sensitivity=1/sqrt(sumwt))

try.pb - Primary beam model (values depend on the gridder used)

Widefield projection algorithms (gridder=mosaic,awproject) will compute the following images too. try.weight - FT of gridded weights or the

un-normalized sum of PB-square (for all pointings) Here, PB = sqrt(weight) normalized to a maximum of 1.0

For multi-term wideband imaging, all relevant images above will have additional .tt0,.tt1, etc suffixes to indicate Taylor terms, plus the following extra output images. try.alpha - spectral index try.alpha.error - estimate of error on spectral index try.beta - spectral curvature (if nterms > 2)

Tip
[Include a directory name in 'imagename' for all] output images to be sent there instead of the current working directory : imagename='mydir/try'

Tip
[Restarting an imaging run without changing 'imagename']

implies continuation from the existing model image on disk.

- If 'startmodel' was initially specified it needs to be set to "" for the restart run (or tclean will exit with an error message).
- By default, the residual image and psf will be recomputed but if no changes were made to relevant parameters between the runs, set calcres=False, calcpsf=False to resume directly from the minor cycle without the (unnecessary) first major cycle.

To automatically change 'imagename' with a numerical increment, set restart=False (see tclean docs for 'restart').

Note
[All imaging runs will by default produce restored images.] For a niter=0 run, this will be redundant and can optionally be turned off via the 'restoration=T/F' parameter.

imsize Number of pixels

example
[imsize = [350,250]] imsize = 500 is equivalent to [500,500]

To take proper advantage of internal optimized FFT routines, the number of pixels must be even and factorizable by 2,3,5,7 only.

cell Cell size
example: cell=['0.5arcsec,'0.5arcsec'] or cell=['1arcmin', '1arcmin'] cell = '1arcsec' is equivalent to ['1arcsec','1arcsec']

phasecenter Phase center of the image (string or field id); if the phasecenter is the name known major solar system object ('MERCURY', 'VENUS', 'MARS', 'JUPITER', 'SATURN', 'URANUS', 'NEPTUNE', 'PLUTO', 'SUN', 'MOON') or is an ephemerides table then that source is tracked and the background sources get smeared. There is a special case, when phasecenter='TRACKFIELD', which will use the ephemerides or polynomial phasecenter in the FIELD table of the MS's as the source center to track.

example: phasecenter=6
phasecenter='J2000 19h30m00 -40d00m00' phasecenter='J2000 292.5deg -40.0deg' phasecenter='J2000 5.105rad -0.698rad' phasecenter='ICRS 13:05:27.2780 -049.28.04.458' phasecenter='myComet_ephem.tab' phasecenter='MOON' phasecenter='TRACKFIELD'

stokes Stokes Planes to make

default='I'; example: stokes='IQUV';
Options: 'I','Q','U','V','IV','QU','IQ','UV','IQUV','RR','LL','XX','YY','RRLL','XXYY','pseudoI'

Note
[Due to current internal code constraints, if any correlation pair] is flagged, by default, no data for that row in the MS will be used. So, in an MS with XX,YY, if only YY is flagged, neither a Stokes I image nor an XX image can be made from those data points. In such a situation, please split out only the unflagged correlation into a separate MS.

Note
[The 'pseudoI' option is a partial solution, allowing Stokes I imaging] when either of the parallel-hand correlations are unflagged.

The remaining constraints shall be removed (where logical) in a future release.

projection Coordinate projection
Examples : SIN, NCP A list of supported (but untested) projections can be found here : http://casa.nrao.edu/active/docs/doxygen/html/classcasa_1_1Projection.html#a3d5f9ec787e4eabdce57ab5edaf7c0cd

startmodel Name of starting model image

The contents of the supplied starting model image will be copied to the imagename.model before the run begins.

example : startmodel = 'singledish.im'

For deconvolver='mtmfs', one image per Taylor term must be provided. example : startmodel = ['try.model.tt0', 'try.model.tt1']

startmodel = ['try.model.tt0'] will use a starting model only
for the zeroth order term.

startmodel = ['','try.model.tt1'] will use a starting model only
for the first order term.

This starting model can be of a different image shape and size from what is currently being imaged. If so, an image regrid is first triggered to resample the input image onto the target coordinate system.

A common usage is to set this parameter equal to a single dish image

Negative components in the model image will be included as is.

[Note

[If an error occurs during image resampling/regridding,] please try using task imregrid to resample the starting model image onto a CASA image with the target shape and coordinate system before supplying it via startmodel]

specmode Spectral definition mode (mfs,cube,cubedata, cubesource)

mode='mfs'

[Continuum imaging with only one output image channel.] (mode='cont' can also be used here)

mode='cube'

[Spectral line imaging with one or more channels]

Parameters start, width,and nchan define the spectral coordinate system and can be specified either in terms of channel numbers, frequency or velocity in whatever spectral frame is specified in 'outframe'. All internal and output images are made with outframe as the base spectral frame. However imaging code internally uses the fixed spectral frame, LSRK for automatic internal software Doppler tracking so that a spectral line observed over an extended time range will line up appropriately. Therefore the output images have additional spectral frame conversion layer in LSRK on the top the base frame.

(Note

[Even if the input parameters are specified in a frame] other than LSRK, the viewer still displays spectral axis in LSRK by default because of the conversion frame layer mentioned above. The viewer can be used to relabel the spectral axis in any desired frame - via the spectral reference option under axis label properties in the data display options window.)

mode='cubedata'

[Spectral line imaging with one or more channels] There is no internal software Doppler tracking so a spectral line observed over an extended time range may be smeared out in frequency. There is strictly no valid spectral frame with which to label the output images, but they will list the frame defined in the MS.

mode='cubesource': Spectral line imaging while tracking moving source (near field or solar system objects). The velocity of the source is accounted and the frequency reported is in the source frame. As there is not SOURCE frame defined, the frame reported will be REST (as it may not be in the rest frame emission region may be moving w.r.t the systemic velocity frame)

reffreq Reference frequency of the output image coordinate system

Example : reffreq='1.5GHz' as a string with units.

By default, it is calculated as the middle of the selected frequency range.

For deconvolver='mtmfs' the Taylor expansion is also done about this specified reference frequency.

nchan Number of channels in the output image

For default (=-1), the number of channels will be automatically determined based on data selected by 'spw' with 'start' and 'width'. It is often easiest to leave nchan at the default value. example: nchan=100

start First channel (e.g. start=3,start='1.1GHz',start='15343km/s')
of output cube images specified by data channel number (integer), velocity (string with a unit), or frequency (string with a unit). Default:''; The first channel is automatically determined based on the 'spw' channel selection and 'width'. When the channel number is used along with the channel selection

in 'spw' (e.g. spw='0:6~100'),

'start' channel number is RELATIVE (zero-based) to the selected channels in 'spw'. So for the above example, start=1 means that the first image channel is the second selected data channel, which is channel 7. For specmode='cube', when velocity or frequency is used it is interpreted with the frame defined in outframe. [The parameters of the desired output cube can be estimated by using the 'transform' functionality of 'plotms'] examples: start='5.0km/s'; 1st channel, 5.0km/s in outframe

start='22.3GHz'; 1st channel, 22.3GHz in outframe

width Channel width (e.g. width=2,width='0.1MHz',width='10km/s') of output cube images
specified by data channel number (integer), velocity (string with a unit), or or frequency (string with a unit). Default:''; data channel width The sign of width defines the direction of the channels to be incremented. For width specified in velocity or frequency with '-' in front gives image channels in decreasing velocity or frequency, respectively. For specmode='cube', when velocity or frequency is used it is interpreted with the reference frame defined in outframe. examples: width='2.0km/s'; results in channels with increasing velocity

width='-2.0km/s'; results in channels with decreasing velocity width='40kHz'; results in channels with increasing frequency width=-2; results in channels averaged of 2 data channels incremented from

high to low channel numbers

outframe Spectral reference frame in which to interpret 'start' and 'width'

Options: '','LSRK','LSRD','BARY','GEO','TOPO','GALACTO','LGROUP','CMB' example: outframe='bary' for Barycentric frame

REST – Rest frequency LSRD – Local Standard of Rest (J2000)

– as the dynamical definition (IAU, [9,12,7] km/s in galactic coordinates)

LSRK – LSR as a kinematical (radio) definition
– 20.0 km/s in direction ra,dec = [270,+30] deg (B1900.0)

BARY – Barycentric (J2000) GEO — Geocentric TOPO – Topocentric GALACTO – Galacto centric (with rotation of 220 km/s in direction l,b = [90,0] deg. LGROUP – Local group velocity – 308km/s towards l,b = [105,-7] deg (F. Ghigo)

CMB – CMB velocity – 369.5km/s towards l,b = [264.4, 48.4] deg (F. Ghigo) DEFAULT = LSRK

veltype Velocity type (radio, z, ratio, beta, gamma, optical)
For start and/or width specified in velocity, specifies the velocity definition Options: 'radio','optical','z','beta','gamma','optical' NOTE: the viewer always defaults to displaying the 'radio' frame,

but that can be changed in the position tracking pull down.

The different types (with F = f/f0, the frequency ratio), are:

Z = (-1 + 1/F)

RATIO = (F) * RADIO = (1 - F) OPTICAL == Z BETA = ((1 - F2)/(1 + F2)) GAMMA = ((1 + F2)/2F) * RELATIVISTIC == BETA (== v/c) DEFAULT == RADIO Note that the ones with an '*' have no real interpretation (although the calculation will proceed) if given as a velocity.

restfreq List of rest frequencies or a rest frequency in a string.

Specify rest frequency to use for output image. *Currently it uses the first rest frequency in the list for translation of velocities. The list will be stored in the output images. Default: []; look for the rest frequency stored in the MS, if not available, use center frequency of the selected channels examples: restfreq=['1.42GHz']

restfreq='1.42GHz'

interpolation Spectral interpolation (nearest,linear,cubic)

Interpolation rules to use when binning data channels onto image channels and evaluating visibility values at the centers of image channels.

Note

['linear' and 'cubic' interpolation requires data points on both sides of] each image frequency. Errors are therefore possible at edge channels, or near flagged data channels. When image channel width is much larger than the data channel width there is nothing much to be gained using linear or cubic thus not worth the extra computation involved.

perchanweightdensity When calculating weight density for Briggs

style weighting in a cube, this parameter determines whether to calculate the weight density for each channel independently (the default, True) or a common weight density for all of the selected data. This parameter has no meaning for continuum (specmode='mfs') imaging or for natural and radial weighting schemes. For cube imaging perchanweightdensity=True is a recommended option that provides more uniform sensitivity per channel for cubes, but with generally larger psfs than the perchanweightdensity=False (prior behavior) option. When using Briggs style weight with perchanweightdensity=True, the imaging weight density calculations use only the weights of data that contribute specifically to that channel. On the other hand, when perchanweightdensity=False, the imaging weight density calculations sum all of the weights from all of the data channels selected whose (u,v) falls in a given uv cell on the weight density grid. Since the aggregated weights, in any given uv cell, will change depending on the number of channels included when imaging, the psf calculated for a given frequency channel will also necessarily change, resulting in variability in the psf for a given frequency channel when perchanweightdensity=False. In general, perchanweightdensity=False results in smaller psfs for the same value of robustness compared to perchanweightdensity=True, but the rms noise as a function of channel varies and increases toward the edge channels; perchanweightdensity=True provides more uniform sensitivity per channel for cubes. This may make it harder to find estimates of continuum when perchanweightdensity=False. If you intend to image a large cube in many smaller subcubes and subsequently concatenate, it is advisable to use perchanweightdensity=True to avoid surprisingly varying sensitivity and psfs across the concatenated cube.

gridder Gridding options (standard, wproject, widefield, mosaic, awproject)

The following options choose different gridding convolution functions for the process of convolutional resampling of the measured visibilities onto a regular uv-grid prior to an inverse FFT. Model prediction (degridding) also uses these same functions. Several wide-field effects can be accounted for via careful choices of convolution functions. Gridding (degridding) runtime will rise in proportion to the support size of these convolution functions (in uv-pixels).

standard : Prolate Spheroid with 7x7 uv pixel support size

[This mode can also be invoked using 'ft' or 'gridft']

wproject

[W-Projection algorithm to correct for the widefield]

non-coplanar baseline effect. [Cornwell et.al 2008]

wprojplanes is the number of distinct w-values at which to compute and use different gridding convolution functions (see help for wprojplanes).

Convolution function support size can range
from 5x5 to few 100 x few 100.

[This mode can also be invoked using 'wprojectft']

widefield : Facetted imaging with or without W-Projection per facet.

A set of facets x facets subregions of the specified image are gridded separately using their respective phase centers (to minimize max W). Deconvolution is done on the joint full size image, using a PSF from the first subregion.

wprojplanes=1 : standard prolate spheroid gridder per facet. wprojplanes > 1 : W-Projection gridder per facet. nfacets=1, wprojplanes > 1 : Pure W-Projection and no facetting nfacets=1, wprojplanes=1 : Same as standard,ft,gridft

A combination of facetting and W-Projection is relevant only for very large fields of view. (In our current version of tclean, this

combination runs only with parallel=False.

mosaic
[A-Projection with azimuthally symmetric beams without]

sidelobes, beam rotation or squint correction. Gridding convolution functions per visibility are computed from FTs of PB models per antenna. This gridder can be run on single fields as well as mosaics.

VLA : PB polynomial fit model (Napier and Rots, 1982) EVLA : PB polynomial fit model (Perley, 2015) ALMA : Airy disks for a 10.7m dish (for 12m dishes) and

6.25m dish (for 7m dishes) each with 0.75m blockages (Hunter/Brogan 2011). Joint mosaic imaging supports heterogeneous arrays for ALMA.

Typical gridding convolution function support sizes are between 7 and 50 depending on the desired accuracy (given by the uv cell size or image field of view).

[This mode can also be invoked using 'mosaicft' or 'ftmosaic']

awproject
[A-Projection with azimuthally asymmetric beams and]

including beam rotation, squint correction, conjugate frequency beams and W-projection. [Bhatnagar et.al, 2008]

Gridding convolution functions are computed from aperture illumination models per antenna and optionally combined with W-Projection kernels and a prolate spheroid. This gridder can be run on single fields as well as mosaics.

VLA
[Uses ray traced model (VLA and EVLA) including feed] leg and

subreflector shadows, off-axis feed location (for beam squint and other polarization effects), and a Gaussian fit for the feed beams (Ref: Brisken 2009)

ALMA
[Similar ray-traced model as above (but the correctness] of its polarization properties remains un-verified).

Typical gridding convolution function support sizes are between 7 and 50 depending on the desired accuracy (given by the uv cell size or image field of view). When combined with W-Projection they can be significantly larger.

[This mode can also be invoked using 'awprojectft']

imagemosaic
[(untested implementation)] Grid and iFT each pointing separately and combine the images as a linear mosaic (weighted by a PB model) in the image domain before a joint minor cycle.

VLA/ALMA PB models are same as for gridder='mosaicft'

—— Notes on PB models :

(1) **Several different sources of PB models are used in the modes**
listed above. This is partly for reasons of algorithmic flexibility and partly due to the current lack of a common beam model repository or consensus on what beam models are most appropriate.

(2) **For ALMA and gridder='mosaic', ray-traced (TICRA) beams**
are also available via the vpmanager tool. For example, call the following before the tclean run.

vp.setpbimage(telescope="ALMA", compleximage='/home/casa/data/trunk/alma/responses/ALMA_0_DV__0_0_
antnames=['DV'+'%02d'%k for k in range(25)]) vp.saveastable('mypb.tab') Then, supply vptable='mypb.tab' to tclean. (Currently this will work only for non-parallel runs)

—— Note on PB masks :

In tclean, A-Projection gridders (mosaic and awproject) produce a .pb image and use the 'pblimit' subparameter to decide normalization cutoffs and construct an internal T/F mask in the .pb and .image images. However, this T/F mask cannot directly be used during deconvolution (which needs a 1/0 mask). There are two options for making a pb based deconvolution mask.

– Run tclean with niter=0 to produce the .pb, construct a 1/0 image

with the desired threshold (using ia.open('newmask.im'); ia.calc('iif("xxx.pb">0.3,1.0,0.0)');ia.close() for example), and supply it via the 'mask' parameter in a subsequent run (with calcres=F and calcpsf=F to restart directly from the minor cycle).

– Run tclean with usemask='pb' for it to automatically construct

a 1/0 mask from the internal T/F mask from .pb at a fixed 0.2 threshold.

—— Making PBs for gridders other than mosaic,awproject

After the PSF generation, a PB is constructed using the same models used in gridder='mosaic' but just evaluated in the image domain without consideration to weights.

facets Number of facets on a side

A set of (facets x facets) subregions of the specified image are gridded separately using their respective phase centers (to minimize max W). Deconvolution is done on the joint full size image, using a PSF from the first subregion/facet.

In our current version of tclean, facets>1 may be used only with parallel=False.

psfphasecenter For mosaic use psf centered on this
optional direction. You may need to use this if for example the mosaic does not have any pointing in the center of the image. Another reason; as the psf is approximate for a mosaic, this may help to deconvolve a non central bright source well and quickly.

example:

psfphasecenter=6 #center psf on field 6 psfphasecenter='J2000 19h30m00 -40d00m00' psfphasecenter='J2000 292.5deg -40.0deg' psfphasecenter='J2000 5.105rad -0.698rad' psfphasecenter='ICRS 13:05:27.2780 -049.28.04.458'

wprojplanes Number of distinct w-values at which to compute and use different
gridding convolution functions for W-Projection

An appropriate value of wprojplanes depends on the presence/absence of a bright source far from the phase center, the desired dynamic range of an image in the presence of a bright far out source, the maximum w-value in the measurements, and the desired trade off between accuracy and computing cost.

As a (rough) guide, VLA L-Band D-config may require a value of 128 for a source 30arcmin away from the phase center. A-config may require 1024 or more. To converge to an appropriate value, try starting with 128 and then increasing it if artifacts persist. W-term artifacts (for the VLA) typically look like arc-shaped smears in a synthesis image or a shift in source position between images made at different times. These artifacts are more pronounced the further the source is from the phase center.

There is no harm in simply always choosing a large value (say, 1024) but there will be a significant performance cost to doing so, especially for gridder='awproject' where it is combined with A-Projection.

wprojplanes=-1 is an option for gridder='widefield' or 'wproject' in which the number of planes is automatically computed.

vptable vpmanager

vptable=""
[Choose default beams for different telescopes] ALMA : Airy disks EVLA : old VLA models.

Other primary beam models can be chosen via the vpmanager tool.

Step 1 : Set up the vpmanager tool and save its state in a table

vp.setpbpoly(telescope='EVLA', coeff=[1.0, -1.529e-3, 8.69e-7, -1.88e-10])
vp.saveastable('myvp.tab')

Step 2 : Supply the name of that table in tclean.

tclean(…., vptable='myvp.tab',….)

Please see the documentation for the vpmanager for more details on how to choose different beam models. Work is in progress to update the defaults for EVLA and ALMA.

Note
[AWProjection currently does not use this mechanism to choose] beam models. It instead uses ray-traced beams computed from parameterized aperture illumination functions, which are not available via the vpmanager. So, gridder='awproject' does not allow the user to set this parameter.

mosweight When doing Brigg's style weighting (including uniform) to perform the weight density calculation for each field indepedently if True. If False the weight density is calculated from the average uv distribution of all the fields. aterm Use aperture illumination functions during gridding

This parameter turns on the A-term of the AW-Projection gridder. Gridding convolution functions are constructed from aperture illumination function models of each antenna.

psterm Include the Prolate Spheroidal (PS) funtion as the anti-aliasing
operator in the gridding convolution functions used for gridding.

Setting this parameter to true is necessary when aterm is set to false. It can be set to false when aterm is set to true, though with this setting effects of aliasing may be there in the image, particularly near the edges.

When set to true, the .pb images will contain the fourier transform of the of the PS funtion. The table below enumarates the functional effects of the psterm, aterm and wprojplanes settings. PB referes to the Primary Beam and FT() refers to the Fourier transform operation.

AW-Projection True True >1 FT(PS) x PB
False PB

A-Projection True True 1 FT(PS) x PB
False PB

W-Projection False True >1 FT(PS)

Standard False True 1 FT(PS)

wbawp Use frequency dependent A-terms
Scale aperture illumination functions appropriately with frequency when gridding and combining data from multiple channels.

conjbeams Use conjugate frequency for wideband A-terms

While gridding data from one frequency channel, choose a convolution function from a 'conjugate' frequency such that the resulting baseline primary beam is approximately constant across frequency. For a system in which the primary beam scales with frequency, this step will eliminate instrumental spectral structure from the measured data and leave only the sky spectrum for the minor cycle to model and reconstruct [Bhatnagar et al., ApJ, 2013].

As a rough guideline for when this is relevant, a source at the half power point of the PB at the center frequency will see an artificial spectral index of -1.4 due to the frequency dependence of the PB [Sault and Wieringa, 1994]. If left uncorrected during gridding, this spectral structure must be modeled in the minor cycle (using the mtmfs algorithm) to avoid dynamic range limits (of a few hundred for a 2:1 bandwidth). This works for specmode='mfs' and its value is ignored for cubes

cfcache Convolution function cache directory name

Name of a directory in which to store gridding convolution functions. This cache is filled at the beginning of an imaging run. This step can be time consuming but the cache can be reused across multiple imaging runs that use the same image parameters (cell size, image size , spectral data selections, wprojplanes, wbawp, psterm, aterm). The effect of the wbawp, psterm and aterm settings is frozen-in in the cfcache. Using an existing cfcache made with a different setting of these parameters will not reflect the current settings.

In a parallel execution, the construction of the cfcache is also parallelized and the time to compute scales close to linearly with the number of compute cores used. With the re-computation of Convolution Functions (CF) due to PA rotation turned-off (the computepastep parameter), the total number of in the cfcache can be computed as [No. of wprojplanes x No. of selected spectral windows x 4]

By default, cfcache = imagename + '.cf'

usepointing The usepointing flag informs the gridder that it should utilize the pointing table
to use the correct direction in which the antenna is pointing with respect to the pointing phasecenter.

computepastep Parallactic angle interval after the AIFs are recomputed (deg)

This parameter controls the accuracy of the aperture illumination function used with AProjection for alt-az mount dishes where the AIF rotates on the sky as the synthesis image is built up. Once the PA in the data changes by the given interval, AIFs are re-computed at the new PA.

A value of 360.0 deg (the default) implies no re-computation due to PA rotation. AIFs are computed for the PA value of the first valid data received and used for all of the data.

rotatepastep Parallactic angle interval after which the nearest AIF is rotated (deg)

Instead of recomputing the AIF for every timestep's parallactic angle, the nearest existing AIF is used and rotated after the PA changed by rotatepastep value.

A value of 360.0 deg (the default) disables rotation of the AIF.

For example, computepastep=360.0 and rotatepastep=5.0 will compute the AIFs at only the starting parallactic angle and all other timesteps will use a rotated version of that AIF at the nearest 5.0 degree point.

pointingoffsetsigdev Corrections for heterogenous and time-dependent pointing

offsets via AWProjection are controlled by this parameter. It is a vector of 2 ints or doubles each of which is interpreted in units of arcsec. Based on the first threshold, a clustering algorithm is applied to entries from the POINTING subtable of the MS to determine how distinct antenna groups for which the pointing offset must be computed separately. The second number controls how much a pointing change across time can be ignored and after which an antenna rebinning is required.

Note
[The default value of this parameter is [], due a programmatic constraint.] If run with this value, it will internally pick [600,600] and exercise the option of using large tolerances (10arcmin) on both axes. Please choose a setting explicitly for runs that need to use this parameter.

Note : This option is available only for gridder='awproject' and usepointing=True and

and has been validated primarily with VLASS on-the-fly mosaic data where POINTING subtables have been modified after the data are recorded.

Examples of parameter usage :

[100.0,100.0]
[Pointing offsets of 100 arcsec or less are considered] small enough to be ignored. Using large values for both indicates a homogeneous array.

[10.0, 100.0]
[Based on entries in the POINTING subtable, antennas] are grouped into clusters based on a 10arcsec bin size. All antennas in a bin are given a pointing offset calculated as the average of the offsets of all antennas in the bin. On the time axis, offset changes upto 100 arcsec will be ignored.

[10.0,10.0]
[Calculate separate pointing offsets for each antenna group] (with a 10 arcsec bin size). As a function of time, recalculate the antenna binning if the POINTING table entries change by more than 10 arcsec w.r.to the previously computed binning.

[1.0, 1.0]
[Tight tolerances will imply a fully heterogenous situation where] each antenna gets its own pointing offset. Also, time-dependent offset changes greater than 1 arcsec will trigger recomputes of the phase

gradients. This is the most general situation and is also the most expensive option as it constructs and uses separate phase gradients for all baselines and timesteps.

For VLASS 1.1 data with two kinds of pointing offsets, the recommended setting is [30.0, 30.0].

For VLASS 1.2 data with only the time-dependent pointing offsets, the recommended setting is [300.0, 30.0] to turn off the antenna grouping but to retain the time dependent corrections required from one timestep to the next.

pblimit PB gain level at which to cut off normalizations

Divisions by .pb during normalizations have a cut off at a .pb gain level given by pblimit. Outside this limit, image values are set to zero. Additionally, by default, an internal T/F mask is applied to the .pb, .image and .residual images to mask out (T) all invalid pixels outside the pblimit area.

Note

[This internal T/F mask cannot be used as a deconvolution mask.] To do so, please follow the steps listed above in the Notes for the 'gridder' parameter.

Note

[To prevent the internal T/F mask from appearing in anything other] than the .pb and .image.pbcor images, 'pblimit' can be set to a negative number. The absolute value will still be used as a valid 'pblimit'. A tclean restart using existing output images on disk that already have this T/F mask in the .residual and .image but only pblimit set to a negative value, will remove this mask after the next major cycle.

normtype Normalization type (flatnoise, flatsky, pbsquare)

Gridded (and FT'd) images represent the PB-weighted sky image. Qualitatively it can be approximated as two instances of the PB applied to the sky image (one naturally present in the data and one introduced during gridding via the convolution functions).

xxx.weight : Weight image approximately equal to sum (square (pb)) xxx.pb : Primary beam calculated as sqrt (xxx.weight)

normtype='flatnoise'

[Divide the raw image by sqrt(.weight) so that] the input to the minor cycle represents the product of the sky and PB. The noise is 'flat' across the region covered by each PB.

normtype='flatsky'

[Divide the raw image by .weight so that the input] to the minor cycle represents only the sky. The noise is higher in the outer regions of the primary beam where the sensitivity is low.

normtype='pbsquare'

[No normalization after gridding and FFT.] The minor cycle sees the sky times pb square

deconvolver Name of minor cycle algorithm (hogbom,clark,multiscale,mtmfs,mem,clarkstokes)

Each of the following algorithms operate on residual images and psfs from the gridder and produce output model and restored images. Minor cycles stop and a major cycle is triggered when cyclethreshold or cycleniter are reached. For all methods, components are picked from the entire extent of the image or (if specified) within a mask.

hogbom

[An adapted version of Hogbom Clean [Hogbom, 1974]]

- Find the location of the peak residual

- Add this delta function component to the model image
- Subtract a scaled and shifted PSF of the same size as the image from regions of the residual image where the two overlap.
- Repeat

clark
[An adapted version of Clark Clean [Clark, 1980]]

- Find the location of max(I^2+Q^2+U^2+V^2)
- Add delta functions to each stokes plane of the model image
- Subtract a scaled and shifted PSF within a small patch size from regions of the residual image where the two overlap.
- After several iterations trigger a Clark major cycle to subtract components from the visibility domain, but without de-gridding.
- Repeat

(Note
['clark' maps to imagermode='' in the old clean task.]

'clark_exp' is another implementation that maps to
imagermode='mosaic' or 'csclean' in the old clean task but the behavior is not identical. For now, please use deconvolver='hogbom' if you encounter problems.)

clarkstokes : Clark Clean operating separately per Stokes plane

(Note : 'clarkstokes_exp' is an alternate version. See above.)

multiscale
[MultiScale Clean [Cornwell, 2008]]

- Smooth the residual image to multiple scale sizes
- Find the location and scale at which the peak occurs
- Add this multiscale component to the model image
- Subtract a scaled,smoothed,shifted PSF (within a small patch size per scale) from all residual images
- Repeat from step 2

mtmfs
[Multi-term (Multi Scale) Multi-Frequency Synthesis [Rau and Cornwell, 2011]]

- Smooth each Taylor residual image to multiple scale sizes
- Solve a NTxNT system of equations per scale size to compute Taylor coefficients for components at all locations
- **Compute gradient chi-square and pick the Taylor coefficients**
  and scale size at the location with maximum reduction in chi-square
- Add multi-scale components to each Taylor-coefficient model image
- Subtract scaled,smoothed,shifted PSF (within a small patch size per scale) from all smoothed Taylor residual images
- Repeat from step 2

mem

[Maximum Entropy Method [Cornwell and Evans, 1985]]

- Iteratively solve for values at all individual pixels via the MEM method. It minimizes an objective function of

  chi-square plus entropy (here, a measure of difference

  between the current model and a flat prior model).

  (Note

  [This MEM implementation is not very robust.] Improvements will be made in the future.)

scales List of scale sizes (in pixels) for multi-scale and mtmfs algorithms.

–> scales=[0,6,20] This set of scale sizes should represent the sizes (diameters in units of number of pixels) of dominant features in the image being reconstructed.

The smallest scale size is recommended to be 0 (point source), the second the size of the synthesized beam and the third 3-5 times the synthesized beam, etc. For example, if the synthesized beam is 10" FWHM and cell=2",try scales = [0,5,15].

For numerical stability, the largest scale must be smaller than the image (or mask) size and smaller than or comparable to the scale corresponding to the lowest measured spatial frequency (as a scale size much larger than what the instrument is sensitive to is unconstrained by the data making it harder to recovery from errors during the minor cycle).

nterms Number of Taylor coefficients in the spectral model

- nterms=1 : Assume flat spectrum source
- nterms=2 : Spectrum is a straight line with a slope
- nterms=N : A polynomial of order N-1

From a Taylor expansion of the expression of a power law, the spectral index is derived as alpha = taylorcoeff_1 / taylorcoeff_0

Spectral curvature is similarly derived when possible.

The optimal number of Taylor terms depends on the available signal to noise ratio, bandwidth ratio, and spectral shape of the source as seen by the telescope (sky spectrum x PB spectrum).

nterms=2 is a good starting point for wideband EVLA imaging and the lower frequency bands of ALMA (when fractional bandwidth is greater than 10%) and if there is at least one bright source for which a dynamic range of greater than few 100 is desired.

Spectral artifacts for the VLA often look like spokes radiating out from a bright source (i.e. in the image made with standard mfs imaging). If increasing the number of terms does not eliminate these artifacts, check the data for inadequate bandpass calibration. If the source is away from the pointing center, consider including wide-field corrections too.

(Note

[In addition to output Taylor coefficient images .tt0,.tt1,etc] images of spectral index (.alpha), an estimate of error on spectral index (.alpha.error) and spectral curvature (.beta, if nterms is greater than 2) are produced. - These alpha, alpha.error and beta images contain

internal T/F masks based on a threshold computed as peakresidual/10. Additional masking based on

.alpha/.alpha.error may be desirable.

- .alpha.error is a purely empirical estimate derived from the propagation of error during the division of two noisy numbers (alpha = xx.tt1/xx.tt0) where the 'error' on tt1 and tt0 are simply the values picked from the corresponding residual images. The absolute value of the error is not always accurate and it is best to interpret the errors across the image only in a relative sense.)

smallscalebias A numerical control to bias the scales when using multi-scale or mtmfs algorithms.
The peak from each scale's smoothed residual is multiplied by (1 - smallscalebias * scale/maxscale) to increase or decrease the amplitude relative to other scales, before the scale with the largest peak is chosen. Smallscalebias can be varied between -1.0 and 1.0. A score of 0.0 gives all scales equal weight (default).

A score larger than 0.0 will bias the solution towards smaller scales. A score smaller than 0.0 will bias the solution towards larger scales. The effect of smallscalebias is more pronounced when using multi-scale relative to mtmfs.

restoration e.

Construct a restored image : imagename.image by convolving the model image with a clean beam and adding the residual image to the result. If a restoringbeam is specified, the residual image is also smoothed to that target resolution before adding it in.

If a .model does not exist, it will make an empty one and create the restored image from the residuals (with additional smoothing if needed). With algorithm='mtmfs', this will construct Taylor coefficient maps from the residuals and compute .alpha and .alpha.error.

restoringbeam ize to use.

- restoringbeam=" or ['] A Gaussian fitted to the PSF main lobe (separately per image plane).
- restoringbeam='10.0arcsec' Use a circular Gaussian of this width for all planes
- restoringbeam=['8.0arcsec','10.0arcsec','45deg'] Use this elliptical Gaussian for all planes
- restoringbeam='common' Automatically estimate a common beam shape/size appropriate for all planes.

Note
[For any restoring beam different from the native resolution] the model image is convolved with the beam and added to residuals that have been convolved to the same target resolution.

pbcor the output restored image

A new image with extension .image.pbcor will be created from the evaluation of .image / .pb for all pixels above the specified pblimit.

Note
[Stand-alone PB-correction can be triggered by re-running] tclean with the appropriate imagename and with niter=0, calcpsf=False, calcres=False, pbcor=True, vptable='vp.tab' (where vp.tab is the name of the vpmanager file.

See the inline help for the 'vptable' parameter)

Note
[Multi-term PB correction that includes a correction for the] spectral index of the PB has not been enabled for the 4.7 release. Please use the widebandpbcor task instead. (Wideband PB corrections are required when the amplitude of the

brightest source is known accurately enough to be sensitive to the difference in the PB gain between the upper and lower end of the band at its location. As a guideline,

the artificial spectral index due to the PB is -1.4 at the 0.5 gain level and less than -0.2 at the 0.9 gain level at the middle frequency)

outlierfile Name of outlier-field image definitions

A text file containing sets of parameter=value pairs, one set per outlier field.

Example : outlierfile='outs.txt'

Contents of outs.txt :

imagename=tst1 nchan=1 imsize=[80,80] cell=[8.0arcsec,8.0arcsec] phasecenter=J2000 19:58:40.895 +40.55.58.543 mask=circle[[40pix,40pix],10pix]

imagename=tst2 nchan=1 imsize=[100,100] cell=[8.0arcsec,8.0arcsec] phasecenter=J2000 19:58:40.895 +40.56.00.000 mask=circle[[60pix,60pix],20pix]

The following parameters are currently allowed to be different between the main field and the outlier fields (i.e. they will be recognized if found in the outlier text file). If a parameter is not listed, the value is picked from what is defined in the main task input.

imagename, imsize, cell, phasecenter, startmodel, mask specmode, nchan, start, width, nterms, reffreq, gridder, deconvolver, wprojplanes

Note

['specmode' is an option, so combinations of mfs and cube]

for different image fields, for example, are supported.

'deconvolver' and 'gridder' are also options that allow different

imaging or deconvolution algorithm per image field.

For example, multiscale with wprojection and 16 w-term planes on the main field and mtmfs with nterms=3 and wprojection with 64 planes on a bright outlier source for which the frequency dependence of the primary beam produces a strong effect that must be modeled. The traditional alternative to this approach is to first image the outlier, subtract it out of the data (uvsub) and then image the main field.

Note

[If you encounter a use-case where some other parameter needs] to be allowed in the outlier file (and it is logical to do so), please send us feedback. The above is an initial list.

weighting Weighting scheme (natural,uniform,briggs,superuniform,radial, briggsabs, briggsbwtaper)

During gridding of the dirty or residual image, each visibility value is multiplied by a weight before it is accumulated on the uv-grid. The PSF's uv-grid is generated by gridding only the weights (weightgrid).

weighting='natural'

[Gridding weights are identical to the data weights] from the MS. For visibilities with similar data weights, the weightgrid will follow the sample density pattern on the uv-plane. This weighting scheme provides the maximum imaging sensitivity at the expense of a possibly fat PSF with high sidelobes. It is most appropriate for detection experiments where sensitivity is most important.

weighting='uniform'
[Gridding weights per visibility data point are the] original data weights divided by the total weight of all data points that map to the same uv grid cell : ' data_weight / total_wt_per_cell '.

The weightgrid is as close to flat as possible resulting in a PSF with a narrow main lobe and suppressed sidelobes. However, since heavily sampled areas of the uv-plane get down-weighted, the imaging sensitivity is not as high as with natural weighting. It is most appropriate for imaging experiments where a well behaved PSF can help the reconstruction.

weighting='briggs'
[Gridding weights per visibility data point are given by]

'data_weight / (A *total_wt_per_cell + B) ' where A and B vary according to the 'robust' parameter.

robust = -2.0 maps to A=1,B=0 or uniform weighting. robust = +2.0 maps to natural weighting. (robust=0.5 is equivalent to robust=0.0 in AIPS IMAGR.)

Robust/Briggs weighting generates a PSF that can vary smoothly between 'natural' and 'uniform' and allow customized trade-offs between PSF shape and imaging sensitivity.

weighting='briggsabs'
[Experimental option.] Same as Briggs except the formula is different A= robust*robust and B is dependent on the noise per visibility estimated. Giving noise='0Jy' is a not a reasonable option. In this mode (or formula) robust values from -2.0 to 0.0 only make sense (2.0 and -2.0 will get the same weighting)

weighting='superuniform'
[This is similar to uniform weighting except that]

the total_wt_per_cell is replaced by the total_wt_within_NxN_cells around the uv cell of interest. (N = subparameter 'npixels')

This method tends to give a PSF with inner sidelobes that are suppressed as in uniform weighting but with far-out sidelobes closer to natural weighting. The peak sensitivity is also closer to natural weighting.

weighting='radial'
[Gridding weights are given by ' data_weight * uvdistance '] This method approximately minimizes rms sidelobes for an east-west synthesis array.

weighting='briggsbwtaper'
[A modified version of Briggs weighting for cubes where an inverse uv taper,] which is proportional to the fractional bandwidth of the entire cube, is applied per channel. The objective is to modify cube (perchanweightdensity = True) imaging weights to have a similar density to that of the continuum imaging weights. This is currently an experimental weighting scheme being developed for ALMA.

For more details on weighting please see Chapter3 of Dan Briggs' thesis (http://www.aoc.nrao.edu/dissertations/dbriggs)

robust Robustness parameter for Briggs weighting.

robust = -2.0 maps to uniform weighting. robust = +2.0 maps to natural weighting. (robust=0.5 is equivalent to robust=0.0 in AIPS IMAGR.)

noise noise parameter for briggs abs mode weighting npixels Number of pixels to determine uv-cell size for super-uniform weighting

(0 defaults to -/+ 3 pixels)

npixels – uv-box used for weight calculation

a box going from -npixel/2 to +npixel/2 on each side

around a point is used to calculate weight density.

npixels=2 goes from -1 to +1 and covers 3 pixels on a side.

npixels=0 implies a single pixel, which does not make sense for
superuniform weighting. Therefore, if npixels=0 it will be forced to 6 (or a box of -3pixels to +3pixels) to cover 7 pixels on a side.

uvtaper uv-taper on outer baselines in uv-plane

Apply a Gaussian taper in addition to the weighting scheme specified via the 'weighting' parameter. Higher spatial frequencies are weighted down relative to lower spatial frequencies to suppress artifacts arising from poorly sampled areas of the uv-plane. It is equivalent to smoothing the PSF obtained by other weighting schemes and can be specified either as a Gaussian in uv-space (eg. units of lambda) or as a Gaussian in the image domain (eg. angular units like arcsec).

uvtaper = [bmaj, bmin, bpa]

NOTE: the on-sky FWHM in arcsec is roughly the uv taper/200 (klambda). default: uvtaper=[]; no Gaussian taper applied example: uvtaper=['5klambda'] circular taper

FWHM=5 kilo-lambda

uvtaper=['5klambda','3klambda','45.0deg'] uvtaper=['10arcsec'] on-sky FWHM 10 arcseconds uvtaper=['300.0'] default units are lambda

in aperture plane

niter Maximum number of iterations

A stopping criterion based on total iteration count. Currently the parameter type is defined as an integer therefore the integer value larger than 2147483647 will not be set properly as it causes an overflow.

Iterations are typically defined as the selecting one flux component and partially subtracting it out from the residual image.

niter=0 : Do only the initial major cycle (make dirty image, psf, pb, etc)

niter larger than zero : Run major and minor cycles.

Note : Global stopping criteria vs major-cycle triggers

In addition to global stopping criteria, the following rules are used to determine when to terminate a set of minor cycle iterations and trigger major cycles [derived from Cotton-Schwab Clean, 1984]

'cycleniter'
[controls the maximum number of iterations per image] plane before triggering a major cycle.

'cyclethreshold'
[Automatically computed threshold related to the]

max sidelobe level of the PSF and peak residual.

Divergence, detected as an increase of 10% in peak residual from the minimum so far (during minor cycle iterations)

The first criterion to be satisfied takes precedence.

Note

[Iteration counts for cubes or multi-field images :]

For images with multiple planes (or image fields) on which the deconvolver operates in sequence, iterations are counted across all planes (or image fields). The iteration count is compared with 'niter' only after all channels/planes/fields have completed their minor cycles and exited either due to 'cycleniter' or 'cyclethreshold'. Therefore, the actual number of iterations reported in the logger can sometimes be larger than the user specified value in 'niter'. For example, with niter=100, cycleniter=20,nchan=10,threshold=0, a total of 200 iterations will be done in the first set of minor cycles before the total is compared with niter=100 and it exits.

Note

[Additional global stopping criteria include]

- no change in peak residual across two major cycles
- a 50% or more increase in peak residual across one major cycle

gain Loop gain

Fraction of the source flux to subtract out of the residual image for the CLEAN algorithm and its variants.

A low value (0.2 or less) is recommended when the sky brightness distribution is not well represented by the basis functions used by the chosen deconvolution algorithm. A higher value can be tried when there is a good match between the true sky brightness structure and the basis function shapes. For example, for extended emission, multiscale clean with an appropriate set of scale sizes will tolerate a higher loop gain than Clark clean (for example).

threshold Stopping threshold (number in units of Jy, or string)

A global stopping threshold that the peak residual (within clean mask) across all image planes is compared to.

threshold = 0.005 : 5mJy threshold = '5.0mJy'

Note

[A 'cyclethreshold' is internally computed and used as a major cycle]

trigger. It is related what fraction of the PSF can be reliably used during minor cycle updates of the residual image. By default the minor cycle iterations terminate once the peak residual reaches the first sidelobe level of the brightest source.

'cyclethreshold' is computed as follows using the settings in
parameters 'cyclefactor','minpsffraction','maxpsffraction','threshold' :

psf_fraction = max_psf_sidelobe_level * 'cyclefactor' psf_fraction = max(psf_fraction, 'minpsffraction'); psf_fraction = min(psf_fraction, 'maxpsffraction'); cyclethreshold = peak_residual * psf_fraction cyclethreshold = max(cyclethreshold, 'threshold')

If nsigma is set (>0.0), the N-sigma threshold is calculated (see the description under nsigma), then cyclethreshold is further modified as,

cyclethreshold = max(cyclethreshold, nsgima_threshold)

'cyclethreshold' is made visible and editable only in the interactive GUI when tclean is run with interactive=True.

nsigma Multiplicative factor for rms-based threshold stopping

N-sigma threshold is calculated as nsigma * rms value per image plane determined from a robust statistics. For nsigma > 0.0, in a minor cycle, a maximum of the two values, the N-sigma threshold and cyclethreshold, is used to trigger a major cycle (see also the descreption under 'threshold'). Set nsigma=0.0 to preserve the previous tclean behavior without this feature. The top level parameter, fastnoise is relevant for the rms noise calculation which is used to determine the threshold.

The parameter 'nsigma' may be an int, float, or a double.

cycleniter Maximum number of minor-cycle iterations (per plane) before triggering
a major cycle

For example, for a single plane image, if niter=100 and cycleniter=20, there will be 5 major cycles after the initial one (assuming there is no threshold based stopping criterion). At each major cycle boundary, if the number of iterations left over (to reach niter) is less than cycleniter, it is set to the difference.

Note

[cycleniter applies per image plane, even if cycleniter x nplanes] gives a total number of iterations greater than 'niter'. This is to preserve consistency across image planes within one set of minor cycle iterations.

cyclefactor Scaling on PSF sidelobe level to compute the minor-cycle stopping threshold.

Please refer to the Note under the documentation for 'threshold' that discussed the calculation of 'cyclethreshold'

cyclefactor=1.0 results in a cyclethreshold at the first sidelobe level of the brightest source in the residual image before the minor cycle starts.

cyclefactor=0.5 allows the minor cycle to go deeper. cyclefactor=2.0 triggers a major cycle sooner.

minpsffraction PSF fraction that marks the max depth of cleaning in the minor cycle

Please refer to the Note under the documentation for 'threshold' that discussed the calculation of 'cyclethreshold'

For example, minpsffraction=0.5 will stop cleaning at half the height of the peak residual and trigger a major cycle earlier.

maxpsffraction PSF fraction that marks the minimum depth of cleaning in the minor cycle

Please refer to the Note under the documentation for 'threshold' that discussed the calculation of 'cyclethreshold'

For example, maxpsffraction=0.8 will ensure that at least the top 20 percent of the source will be subtracted out in the minor cycle even if the first PSF sidelobe is at the 0.9 level (an extreme example), or if the cyclefactor is set too high for anything to get cleaned.

interactive Modify masks and parameters at runtime

interactive=True will trigger an interactive GUI at every major cycle boundary (after the major cycle and before the minor cycle).

The interactive mode is currently not available for parallel cube imaging (please also refer to the Note under the documentation for 'parallel' below).

Options for runtime parameter modification are :

Interactive clean mask
[Draw a 1/0 mask (appears as a contour) by hand.] If a mask is supplied at the task interface or if automasking is invoked, the current mask is displayed in the GUI and is available for manual editing.

Note
[If a mask contour is not visible, please] check the cursor display at the bottom of GUI to see which parts of the mask image have ones and zeros. If the entire mask=1 no contours will be visible.

Operation buttons
[– Stop execution now (restore current model and exit)]

– Continue on until global stopping criteria are reached
without stopping for any more interaction

– Continue with minor cycles and return for interaction
after the next major cycle.

Iteration control : – max cycleniter : Trigger for the next major cycle

The display begins with [min(cycleniter, niter - itercount)] and can be edited by hand.

—iterations left : The display begins with [niter-itercount] and can be edited to increase or decrease the total allowed niter.

– threshold : Edit global stopping threshold

—cyclethreshold : The display begins with the automatically computed value (see Note in help for 'threshold'), and can be edited by hand.

All edits will be reflected in the log messages that appear once minor cycles begin.

[For scripting purposes, replacing True/False with 1/0 will get tclean to
return an imaging summary dictionary to python]

usemask Type of mask(s) to be used for deconvolution

user: (default) mask image(s) or user specified region file(s) or string CRTF expression(s)
subparameters: mask, pbmask

pb: primary beam mask
subparameter: pbmask

Example: usemask="pb", pbmask=0.2
Construct a mask at the 0.2 pb gain level. (Currently, this option will work only with gridders that produce .pb (i.e. mosaic and awproject) or if an externally produced .pb image exists on disk)

auto-multithresh
[auto-masking by multiple thresholds for deconvolution]

subparameters
[sidelobethreshold, noisethreshold, lownoisethreshold, negativethrehsold, smoothfactor,] minbeamfrac, cutthreshold, pbmask, growiterations, dogrowprune, minpercentchange, verbose

Additional top level parameter relevant to auto-multithresh: fastnoise

if pbmask is >0.0, the region outside the specified pb gain level is excluded from image statistics in determination of the threshold.

Note: By default the intermediate mask generated by automask at each deconvolution cycle is over-written in the next cycle but one can save them by setting the environment variable, SAVE_ALL_AUTOMASKS="true". (e.g. in the CASA prompt, os.environ['SAVE_ALL_AUTOMASKS']="true") The saved CASA mask image name will be imagename.mask.autothresh#, where # is the iteration cycle number.

mask Mask (a list of image name(s) or region file(s) or region string(s)

The name of a CASA image or region file or region string that specifies a 1/0 mask to be used for deconvolution. Only locations with value 1 will be considered for the centers of flux components in the minor cycle. If regions specified fall completely outside of the image, tclean will throw an error.

Manual mask options/examples :

mask='xxx.mask'

[Use this CASA image named xxx.mask and containing] ones and zeros as the mask. If the mask is only different in spatial coordinates from what is being made it will be resampled to the target coordinate system before being used. The mask has to have the same shape in velocity and Stokes planes as the output image. Exceptions are single velocity and/or single Stokes plane masks. They will be expanded to cover all velocity and/or Stokes planes of the output cube.

[Note

[If an error occurs during image resampling or] if the expected mask does not appear, please try using tasks 'imregrid' or 'makemask' to resample the mask image onto a CASA image with the target shape and coordinates and supply it via the 'mask' parameter.]

mask='xxx.crtf'

[A text file with region strings and the following on the first line] (#CRTFv0 CASA Region Text Format version 0) This is the format of a file created via the viewer's region tool when saved in CASA region file format.

mask='circle[[40pix,40pix],10pix]' : A CASA region string.

mask=['xxx.mask','xxx.crtf', 'circle[[40pix,40pix],10pix]'] : a list of masks

Note

[Mask images for deconvolution must contain 1 or 0 in each pixel.] Such a mask is different from an internal T/F mask that can be held within each CASA image. These two types of masks are not automatically interchangeable, so please use the makemask task to copy between them if you need to construct a 1/0 based mask from a T/F one.

Note

[Work is in progress to generate more flexible masking options and] enable more controls.

pbmask Sub-parameter for usemask='auto-multithresh': primary beam mask

Examples

[pbmask=0.0 (default, no pb mask)] pbmask=0.2 (construct a mask at the 0.2 pb gain level)

sidelobethreshold Sub-parameter for "auto-multithresh": mask threshold based on sidelobe levels: sidelobethreshold * max_sidelobe_level * peak residual noisethreshold Sub-parameter for "auto-multithresh": mask threshold based on the noise level: noisethreshold * rms + location (=median)

The rms is calculated from MAD with rms = 1.4826*MAD.

lownoisethreshold Sub-parameter for "auto-multithresh": mask threshold to grow previously masked regions via binary dilation: lownoisethreshold * rms in residual image + location (=median)

The rms is calculated from MAD with rms = 1.4826*MAD.

negativethreshold Sub-parameter for "auto-multithresh": mask threshold for negative features: -1.0* negativethreshold * rms + location(=median)

The rms is calculated from MAD with rms = 1.4826*MAD.

smoothfactor Sub-parameter for "auto-multithresh": smoothing factor in a unit of the beam minbeamfrac Sub-parameter for "auto-multithresh": minimum beam fraction in size to prune masks smaller than mimbeamfrac * beam

<=0.0 : No pruning

cutthreshold Sub-parameter for "auto-multithresh": threshold to cut the smoothed mask to create a final mask: cutthreshold * peak of the smoothed mask growiterations Sub-parameter for "auto-multithresh": Maximum number of iterations to perform using binary dilation for growing the mask dogrowprune Experimental sub-parameter for "auto-multithresh": Do pruning on the grow mask minpercentchange If the change in the mask size in a particular channel is less than minpercentchange, stop masking that channel in subsequent cycles. This check is only applied when noise based threshold is used and when the previous clean major cycle had a cyclethreshold value equal to the clean threshold. Values equal to -1.0 (or any value less than 0.0) will turn off this check (the default). Automask will still stop masking if the current channel mask is an empty mask and the noise threshold was used to determine the mask. verbose he summary of automasking at the end of each automasking process

is printed in the logger. Following information per channel will be listed in the summary.

chan: channel number masking?: F - stop updating automask for the subsequent iteration cycles RMS: robust rms noise peak: peak in residual image thresh_type: type of threshold used (noise or sidelobe) thresh_value: the value of threshold used N_reg: number of the automask regions N_pruned: number of the automask regions removed by pruning N_grow: number of the grow mask regions N_grow_pruned: number of the grow mask regions removed by pruning N_neg_pix: number of pixels for negative mask regions

Note that for a large cube, extra logging may slow down the process.

fastnoise mask (user='multi-autothresh') and/or n-sigma stopping threshold (nsigma>0.0) are/is used. If it is set to True, a simpler but faster noise calucation is used.

In this case, the threshold values are determined based on classic statistics (using all unmasked pixels for the calculations).

If it is set to False, the new noise calculation method is used based on pre-existing mask.

Case 1: no exiting mask Calculate image statistics using Chauvenet algorithm

Case 2: there is an existing mask Calculate image statistics by classical method on the region outside the mask and inside the primary beam mask.

In all cases above RMS noise is calculated from MAD.

restart images (and start from an existing model image)

or automatically increment the image name and make a new image set.

True

[Re-use existing images. If imagename.model exists the subsequent]

run will start from this model (i.e. predicting it using current gridder settings and starting from the residual image). Care must be taken when combining this option with startmodel. Currently, only one or the other can be used.

startmodel='', imagename.model exists :

- Start from imagename.model

startmodel='xxx', imagename.model does not exist :

- Start from startmodel

startmodel='xxx', imagename.model exists :

- Exit with an error message requesting the user to pick only one model. This situation can arise when doing one run with startmodel='xxx' to produce an output imagename.model that includes the content of startmodel, and wanting to restart a second run to continue deconvolution. Startmodel should be set to '' before continuing.

If any change in the shape or coordinate system of the image is desired during the restart, please change the image name and use the startmodel (and mask) parameter(s) so that the old model (and mask) can be regridded to the new coordinate system before starting.

False

[A convenience feature to increment imagename with '_1', '_2',]

etc as suffixes so that all runs of tclean are fresh starts (without having to change the imagename parameter or delete images).

This mode will search the current directory for all existing imagename extensions, pick the maximum, and adds 1. For imagename='try' it will make try.psf, try_2.psf, try_3.psf, etc.

This also works if you specify a directory name in the path : imagename='outdir/try'. If './outdir' does not exist, it will create it. Then it will search for existing filenames inside that directory.

If outlier fields are specified, the incrementing happens for each of them (since each has its own 'imagename'). The counters are synchronized across imagefields, to make it easier to match up sets of output images. It adds 1 to the 'max id' from all outlier names on disk. So, if you do two runs with only the main field

(imagename='try'), and in the third run you add an outlier with imagename='outtry', you will get the following image names for the third run : 'try_3' and 'outtry_3' even though 'outry' and 'outtry_2' have not been used.

savemodel Options to save model visibilities (none, virtual, modelcolumn)

Often, model visibilities must be created and saved in the MS to be later used for self-calibration (or to just plot and view them).

none

[Do not save any model visibilities in the MS. The MS is opened] in read-only mode.

Model visibilities can be predicted in a separate step by restarting tclean with niter=0,savemodel=virtual or modelcolumn and not changing any image names so that it finds the .model on disk (or by changing imagename and setting startmodel to the original imagename).

virtual

[In the last major cycle, save the image model and state of the] gridder used during imaging within the SOURCE subtable of the MS. Images required for de-gridding will also be stored internally. All future references to model visibilities will activate the (de)gridder to compute them on-the-fly. This mode is useful when the dataset is large enough that an additional model

data column on disk may be too much extra disk I/O, when the gridder is simple enough that on-the-fly recomputing of the model visibilities is quicker than disk I/O. For e.g. that gridder='awproject' does not support virtual model.

modelcolumn
[In the last major cycle, save predicted model visibilities] in the MODEL_DATA column of the MS. This mode is useful when the degridding cost to produce the model visibilities is higher than the I/O required to read the model visibilities from disk. This mode is currently required for gridder='awproject'. This mode is also required for the ability to later pull out model visibilities from the MS into a python array for custom processing.

Note 1
[The imagename.model image on disk will always be constructed] if the minor cycle runs. This savemodel parameter applies only to model visibilities created by degridding the model image.

Note 2
[It is possible for an MS to have both a virtual model] as well as a model_data column, but under normal operation, the last used mode will get triggered. Use the delmod task to clear out existing models from an MS if confusion arises.

Note 3: when parallel=True, use savemodel='none'; Other options are not yet ready
for use in parallel. If model visibilities need to be saved (virtual or modelcolumn): please run tclean in serial mode with niter=0; after the parallel run

calcres Calculate initial residual image

This parameter controls what the first major cycle does.

calcres=False with niter greater than 0 will assume that a .residual image already exists and that the minor cycle can begin without recomputing it.

calcres=False with niter=0 implies that only the PSF will be made and no data will be gridded.

calcres=True requires that calcpsf=True or that the .psf and .sumwt images already exist on disk (for normalization purposes).

Usage example
[For large runs (or a pipeline scripts) it may be] useful to first run tclean with niter=0 to create an initial .residual to look at and perhaps make a custom mask for. Imaging can be resumed without recomputing it.

calcpsf Calculate PSF

This parameter controls what the first major cycle does.

calcpsf=False will assume that a .psf image already exists and that the minor cycle can begin without recomputing it.

psfcutoff When the .psf image is created a 2 dimensional Gaussian is fit to the main lobe of the PSF.
Which pixels in the PSF are fitted is determined by psfcutoff. The default value of psfcutoff is 0.35 and can varied from 0.01 to 0.99. Fitting algorithm:

- **A region of 41 x 41 pixels around the peak of the PSF is compared against the psfcutoff.**
  Sidelobes are ignored by radially searching from the PSF peak.

- Calculate the bottom left corner (blc) and top right corner (trc) from the points. Expand blc and trc with a number of pixels (5).
- Create a new sub-matrix from blc and trc.
- Interpolate matrix to a target number of points (3001) using CUBIC spline.
- **All the non-sidelobe points, in the interpolated matrix, that are above the psfcutoff are used to fit a Gaussian.**
  A Levenberg-Marquardt algorithm is used.
- **If the fitting fails the algorithm is repeated with the psfcutoff decreased (psfcutoff=psfcutoff/1.5).**
  A message in the log will apear if the fitting fails along with the new value of psfcutoff. This will be done up to 50 times if fitting fails.

This Gaussian beam is defined by a major axis, minor axis, and position angle. During the restoration process, this Gaussian beam is used as the Clean beam. Varying psfcutoff might be useful for producing a better fit for highly non-Gaussian PSFs, however, the resulting fits should be carefully checked. This parameter should rarely be changed.

(This is not the support size for clark clean.)

parallel Run major cycles in parallel (this feature is experimental)

Parallel tclean will run only if casa has already been started using mpirun. Please refer to HPC documentation for details on how to start this on your system.

Example : mpirun -n 3 -xterm 0 *which casa*

Continuum Imaging :

- Data are partitioned (in time) into NProc pieces
- Gridding/iFT is done separately per partition
- Images (and weights) are gathered and then normalized
- One non-parallel minor cycle is run
- Model image is scattered to all processes
- Major cycle is done in parallel per partition

Cube Imaging :

- Data and Image coordinates are partitioned (in freq) into NProc pieces
- Each partition is processed independently (major and minor cycles)
- All processes are synchronized at major cycle boundaries for convergence checks
- At the end, cubes from all partitions are concatenated along the spectral axis

Note 1

[Iteration control for cube imaging is independent per partition.]

- **There is currently no communication between them to synchronize**
  information such as peak residual and cyclethreshold. Therefore, different chunks may trigger major cycles at different levels.
- For cube imaging in parallel, there is currently no interactive masking.

(Proper synchronization of iteration control is work in progress.)

[1;42mRETURNS[1;m void

——— examples ————————————————————

This is the first release of our refactored imager code. Although most features have been used and validated, there are many details that have not been thoroughly tested. Feedback will be much appreciated.

Usage Examples :

(A) A suite of test programs that demo all usable modes of tclean on small test datasets https://svn.cv.nrao.edu/svn/casa/branches/release-4_5/gcwrap/python/scripts/tests/test_refimager.py (B) A set of demo examples for ALMA imaging https://casaguides.nrao.edu/index.php/TCLEAN_and_ALMA

_info_group_ = 'imaging'

_info_desc_ = 'Parallelized tclean in consecutive time steps'

__call__(*vis=''*, *imageprefix=''*, *imagesuffix=''*, *ncpu=int(8)*, *twidth=int(1)*, *doreg=False*, *usephacenter=True*, *reftime=''*, *toTb=False*, *sclfactor=float(1.0)*, *subregion=''*, *docompress=False*, *overwrite=False*, *selectdata=True*, *field=''*, *spw=''*, *timerange=''*, *uvrange=''*, *antenna=''*, *scan=''*, *observation=''*, *intent=''*, *datacolumn='corrected'*, *imagename=''*, *imsize=[int(100)]*, *cell=[]*, *phasecenter=''*, *stokes='I'*, *projection='SIN'*, *startmodel=''*, *specmode='mfs'*, *reffreq=''*, *nchan=int(-1)*, *start=''*, *width=''*, *outframe='LSRK'*, *veltype='radio'*, *restfreq=[]*, *interpolation='linear'*, *perchanweightdensity=True*, *gridder='standard'*, *facets=int(1)*, *psfphasecenter=''*, *wprojplanes=int(1)*, *vptable=''*, *mosweight=True*, *aterm=True*, *psterm=False*, *wbawp=True*, *conjbeams=False*, *cfcache=''*, *usepointing=False*, *computepastep=float(360.0)*, *rotatepastep=float(360.0)*, *pointingoffsetsigdev=[]*, *pblimit=float(0.2)*, *normtype='flatnoise'*, *deconvolver='hogbom'*, *scales=[]*, *nterms=int(2)*, *smallscalebias=float(0.0)*, *restoration=True*, *restoringbeam=[]*, *pbcor=False*, *outlierfile=''*, *weighting='natural'*, *robust=float(0.5)*, *noise='1.0Jy'*, *npixels=int(0)*, *uvtaper=['']*, *niter=int(0)*, *gain=float(0.1)*, *threshold=float(0.0)*, *nsigma=float(0.0)*, *cycleniter=int(-1)*, *cyclefactor=float(1.0)*, *minpsffraction=float(0.05)*, *maxpsffraction=float(0.8)*, *interactive=False*, *usemask='user'*, *mask=''*, *pbmask=float(0.0)*, *sidelobethreshold=float(3.0)*, *noisethreshold=float(5.0)*, *lownoisethreshold=float(1.5)*, *negativethreshold=float(0.0)*, *smoothfactor=float(1.0)*, *minbeamfrac=float(0.3)*, *cutthreshold=float(0.01)*, *growiterations=int(75)*, *dogrowprune=True*, *minpercentchange=float(-1.0)*, *verbose=False*, *fastnoise=True*, *restart=True*, *savemodel='none'*, *calcres=True*, *calcpsf=True*, *psfcutoff=float(0.35)*, *parallel=False*)

suncasa.suncasatasks.ptclean6.**ptclean6**

suncasa.suncasatasks.signalsmooth

cookb_signalsmooth.py

from: http://scipy.org/Cookbook/SignalSmooth

Module Contents

Functions

smooth(x[, window_len, window])	smooth the data using a window with requested size.
gauss_kern(size[, sizey])	Returns a normalized 2D gauss kernel array for convolutions
blur_image(im, n[, ny])	blurs the image by convolving with a gaussian kernel of typical
smooth_demo()	

Attributes

Z	

suncasa.suncasatasks.signalsmooth.**smooth**(*x*, *window_len=10*, *window='hanning'*)

smooth the data using a window with requested size.

This method is based on the convolution of a scaled window with the signal. The signal is prepared by introducing reflected copies of the signal (with the window size) in both ends so that transient parts are minimized in the begining and end part of the output signal.

input:

x: the input signal window_len: the dimension of the smoothing window window: the type of window from 'flat', 'hanning', 'hamming', 'bartlett', 'blackman'

flat window will produce a moving average smoothing.

output:

the smoothed signal

example:

import numpy as np t = np.linspace(-2,2,0.1) x = np.sin(t)+np.random.randn(len(t))*0.1 y = smooth(x)

see also:

numpy.hanning, numpy.hamming, numpy.bartlett, numpy.blackman, numpy.convolve scipy.signal.lfilter

NOTE from B. Chen: slightly modified the reflected copies: window_len-1 points are added to both ends. Previous one is not exactly the reflection, the indices are off by one pixel

suncasa.suncasatasks.signalsmooth.**gauss_kern**(*size*, *sizey=None*)

Returns a normalized 2D gauss kernel array for convolutions

suncasa.suncasatasks.signalsmooth.**blur_image**(*im*, *n*, *ny=None*)

blurs the image by convolving with a gaussian kernel of typical size n. The optional keyword argument ny allows for a different size in the y direction.

suncasa.suncasatasks.signalsmooth.**smooth_demo**()

suncasa.suncasatasks.signalsmooth.**Z**

suncasa.suncasatasks.subvs

Module Contents

Classes

_subvs	subvs ---- Vector-subtraction in UV using selected time ranges and spectral channels as background

Attributes

_pc
subvs

suncasa.suncasatasks.subvs.**_pc**

class suncasa.suncasatasks.subvs.**_subvs**

subvs —- Vector-subtraction in UV using selected time ranges and spectral channels as background

Split is the general purpose program to make a new data set that is a subset or averaged form of an existing data set. General selection parameters are included, and one or all of the various data columns (DATA, LAG_DATA and/or FLOAT_DATA, and possibly MODEL_DATA and/or CORRECTED_DATA) can be selected.

Split is often used after the initial calibration of the data to make a smaller measurement set with only the data that will be used in further flagging, imaging and/or self-calibration. split can average over frequency (channels) and time (integrations).

——— parameter descriptions ———————————————

vis Name of input measurement set outputvis Name of output measurement set timerange Select the time range of the input visbility to be subtracted from spw Select the spectral channels of the input visibility to be subtracted from mode Operation: linear, highpass subtime1 Select the first time range as the background for uv subtraction subtime2 Select the second time range as the background for uv subtraction smoothaxis Select the axis along which smooth is performed smoothtype Select the smooth type smoothwidth Select the width of the smoothing window splitsel Split the selected timerange and spectral channels as outputvis reverse Reverse the sign of the background-subtracted data (for absorptive structure) overwrite Overwrite the already existing output measurement set

——— examples ————————————————————

Subvs is a task to do UV vector-subtraction, by selecting time ranges in the data as background. Subvs can be used to subtract the background continuum emission to separate the time-dependent emission, e.g. solar coherent radio bursts.

Keyword arguments: vis – Name of input visibility file (MS) default: none; example: vis='ngc5921.ms' outputvis – Name of output uv-subtracted visibility file (MS) default: none; example: outputvis='ngc5921_src.ms' timerange – Time range of performing the UV subtraction: default=" means all times. examples: timerange = 'YYYY/MM/DD/hh:mm:ss~YYYY/MM/DD/hh:mm:ss' timerange = 'hh:mm:ss~hh:mm:ss' spw – Select spectral window/channel. default = '' all the spectral channels. Example: spw='0:1~20' mode – operation mode default 'linear' mode = 'linear': use a linear fit for the background to be subtracted mode = 'lowpass': act as

a lowpass filter—smooth the data using different smooth types and smooth window size. Can be performed along either time or frequency axis mode = 'highpass': act as a highpass filter—smooth the data first, and subtract the smoothed data from the original. Can be performed along either time or frequency axis mode = 'linear' expandable parameters: subtime1 – Time range 1 of the background to be subtracted from the data default='' means all times. format: timerange = 'YYYY/MM/DD/hh:mm:ss~YYYY/MM/DD/hh:mm:ss' timerange = 'hh:mm:ss~hh:mm:ss' subtime2 – Time range 2 of the backgroud to be subtracted from the data default='' means all times. examples: timerange = 'YYYY/MM/DD/hh:mm:ss~YYYY/MM/DD/hh:mm:ss' timerange = 'hh:mm:ss~hh:mm:ss' mode = 'lowpass' or 'highpass' expandable parameters: smoothaxis – axis of smooth Default: 'time' smoothaxis = 'time': smooth is along the time axis smoothaxis = 'freq': smooth is along the frequency axis smoothtype – type of the smooth depending on the convolving kernel Default: 'flat' smoothtype = 'flat': convolving kernel is a flat rectangle, equivalent to a boxcar moving smooth smoothtype = 'hanning': Hanning smooth kernel. See numpy.hanning smoothtype = 'hamming': Hamming smooth kernel. See numpy.hamming smoothtype = 'bartlett': Bartlett smooth kernel. See numpy.bartlett smoothtype = 'blackman': Blackman smooth kernel. See numpy.blackman smoothwidth – width of the smooth kernel Default: 5 Examples: smoothwidth=5, meaning the width is 5 pixels splitsel – True or False. default = False. If splitsel = False, then the entire input measurement set is copied as the output measurement set (outputvis), with background subtracted at selected timerange and spectral channels. If splitsel = True,then only the selected timerange and spectral channels are copied into the output measurement set (outputvis). reverse – True or False. default = False. If reverse = False, then the times indicated by subtime1 and/or subtime2 are treated as background and subtracted; If reverse = True, then reverse the sign of the background-subtracted data. The option can be used for mapping absorptive structure. overwrite – True or False. default = False. If overwrite = True and outputvis already exists, the selected subtime and spw in the output measurment set will be replaced with background subtracted visibilities

_info_group_ = 'misc'

_info_desc_ = 'Vector-subtraction in UV using selected time ranges and spectral channels as background'

__call__(*vis=''*, *outputvis=''*, *timerange=''*, *spw=''*, *mode='linear'*, *subtime1=''*, *subtime2=''*, *smoothaxis='time'*, *smoothtype='flat'*, *smoothwidth=int(5)*, *splitsel=True*, *reverse=False*, *overwrite=False*)

suncasa.suncasatasks.subvs.**subvs**

Package Contents

suncasa.suncasatasks.**ptclean6**

suncasa.suncasatasks.**subvs**

suncasa.suncasatasks.**concateovsa**

suncasa.utils

Submodules

suncasa.utils.DButil

Module Contents

Classes

ButtonsPlayCTRL	Produce A play/stop button widget for bokeh plot

Functions

img2html_movie(imgprefix[, outname, img_fmt])	
my_timer(orig_func)	
spectrogram2wav(spec[, threshld, gama, fs, t_length, ...])	Convert spectrogram to audio in WAV format
smooth(x[, window_len, window])	smooth the data using a window with requested size.
img2movie([imgprefix, img_ext, outname, size, ...])	**param imgprefix**
image_fill_gap(image)	
getspwfromfreq(vis, freqrange)	
initconfig(suncasa_dir)	
ProgressBar(iteration, total[, prefix, suffix, ...])	Call in a loop to create terminal progress bar
getcurtimstr([prefix, suffix])	
getlatestfile([directory, prefix, suffix])	
loadjsonfile(jsonfile[, mustexist])	
updatejsonfile(jsonfile, data)	
getSDOdir(config, database_dir, suncasa_dir)	
getsdodir(filename[, unique])	return a list of the data path relative to the SDOdir
FileNotInList(file2chk, filelist)	return the index of files not in the list
getfreeport()	
normalize_aiamap(aiamap)	do expisure normalization of an aia map
tplt(mapcube)	
sdo_aia_scale_hdr(amap[, sigma])	
sdo_aia_scale_dict([wavelength, imagetype])	rescale the aia image
sdo_aia_scale([image, wavelength])	rescale the aia image
insertchar(source_str, insert_str, pos)	
readsdofile([datadir, wavelength, trange, isexists, ...])	read sdo file from local database
readsdofileX([datadir, filelist, wavelength, trange, ...])	read sdo file from local database

continues on next page

Table 2 – continued from previous page

findDist(x, y)	
paramspline(x, y, length[, s])	
polyfit(x, y, length, deg[, keepxorder])	
htfit_warren2011(x, y, cutlength)	
spline(x, y, length[, s])	
get_curve_grad(x, y)	get the grad of at data point
improfile(z, xi, yi[, interp])	Pixel-value cross-section along line segment in an image
canvaspix_to_data(smap, x, y)	
data_to_mappixel(smap, x, y)	
polsfromfitsheader(header)	get polarisation information from fits header
headerfix(header)	
freqsfromfitsheader(header)	get frequency in GHz from fits header
transfitdict2DF(datain[, gaussfit, getcentroid])	convert the results from pimfit or pmaxfit tasks to pandas DataFrame structure.
getcolctinDF(dspecDF, col)	return the count of how many times of the element starts with col occurs in columns of dspecDF
dspecDFfilter(dspecDF, pol)	filter the unselect polarisation from dspecDF
dspecDF2text(DFfile[, outfile])	
smapmeshgrid2(smap[, angle, rescale, origin])	
map2wcsgrids(snpmap[, cell, antialiased])	**param snpmap**
smapradialfilter(sunpymap[, factor, grid])	
regridimage(values, x, y[, grid, resize])	re-grid the data on a regular grid with uneven grid spacing to an uniform grid
regridspec(spec, x, y[, nxmax, nymax, interp])	**param spec** ndarray of float or complex, shape (npol,nbl,nf,nt) Data values.
get_contour_data(X, Y, Z[, levels])	
c_correlate(a, v[, returnx])	
c_correlateX(a, v[, returnx, returnav, s])	**param a**
XCorrMap(z, x, y[, doxscale])	get the cross correlation map along y axis

Attributes

__author__
__email__

suncasa.utils.DButil.**__author__ = ['Sijie Yu']**

suncasa.utils.DButil.**__email__ = 'sijie.yu@njit.edu'**

suncasa.utils.DButil.**img2html_movie**(*imgprefix*, *outname='movie'*, *img_fmt='png'*)

suncasa.utils.DButil.**my_timer**(*orig_func*)

suncasa.utils.DButil.**spectrogram2wav**(*spec*, *threshld=None*, *gama=1*, *fs=1.0*, *t_length=None*, *w=1*, *wavfile='output.wav'*)

Convert spectrogram to audio in WAV format :param spec: spec.shape (nfreq, ntime) :param threshld: below which is set to be threshold :param gama: :param fs: :param t_length: time duration of output WAV file :param w: width of the smooth window, if apply :param wavfile: :return:

suncasa.utils.DButil.**smooth**(*x*, *window_len=11*, *window='hanning'*)

smooth the data using a window with requested size.

This method is based on the convolution of a scaled window with the signal. The signal is prepared by introducing reflected copies of the signal (with the window size) in both ends so that transient parts are minimized in the begining and end part of the output signal.

input:

x: the input signal window_len: the dimension of the smoothing window; should be an odd integer window: the type of window from 'flat', 'hanning', 'hamming', 'bartlett', 'blackman'

flat window will produce a moving average smoothing.

output:

the smoothed signal

example:

t=linspace(-2,2,0.1) x=sin(t)+randn(len(t))*0.1 y=smooth(x)

see also:

numpy.hanning, numpy.hamming, numpy.bartlett, numpy.blackman, numpy.convolve scipy.signal.lfilter

NOTE: length(output) != length(input), to correct this: return y[(window_len/2-1):-(window_len/2)] instead of just y.

suncasa.utils.DButil.**img2movie**(*imgprefix=''*, *img_ext='png'*, *outname='movie'*, *size=None*, *start_num=0*, *crf=15*, *fps=10*, *overwrite=False*, *crop=[]*, *title=[]*, *dpi=200*, *keeptmp=False*, *usetmp=False*, *autorotate=True*)

Parameters

- **imgprefix** –
- **img_ext** –
- **outname** –

- **size** –
- **start_num** –
- **crf** –
- **fps** –
- **overwrite** –
- **title** – the timestamp on each frame
- **crop** – 4-tuple of integer specifies the cropping pixels [x0, x1, y0, y1]
- **dpi** –
- **keeptmp** –
- **usetmp** – use the image in the default tmp folder

Returns

suncasa.utils.DButil.**image_fill_gap**(*image*)

suncasa.utils.DButil.**getspwfromfreq**(*vis*, *freqrange*)

suncasa.utils.DButil.**initconfig**(*suncasa_dir*)

suncasa.utils.DButil.**ProgressBar**(*iteration*, *total*, *prefix=''*, *suffix=''*, *decimals=1*, *length=100*, *empfill=' '*, *fill='#'*)

Call in a loop to create terminal progress bar @params:

iteration - Required : current iteration (Int) total - Required : total iterations (Int) prefix - Optional : prefix string (Str) suffix - Optional : suffix string (Str) decimals - Optional : positive number of decimals in percent complete (Int) length - Optional : character length of bar (Int) fill - Optional : bar fill character (Str) empfill - Optional : empty bar fill character (Str)

suncasa.utils.DButil.**getcurtimstr**(*prefix='CleanID_'*, *suffix=''*)

suncasa.utils.DButil.**getlatestfile**(*directory='./'*, *prefix='CleanID_'*, *suffix=''*)

suncasa.utils.DButil.**loadjsonfile**(*jsonfile*, *mustexist=True*)

suncasa.utils.DButil.**updatejsonfile**(*jsonfile*, *data*)

suncasa.utils.DButil.**getSDOdir**(*config*, *database_dir*, *suncasa_dir*)

suncasa.utils.DButil.**getsdodir**(*filename*, *unique=True*)

return a list of the data path relative to the SDOdir :param filename: :return:

suncasa.utils.DButil.**FileNotInList**(*file2chk*, *filelist*)

return the index of files not in the list :param file2chk: files to be check :param filelist: the list :return:

suncasa.utils.DButil.**getfreeport**()

suncasa.utils.DButil.**normalize_aiamap**(*aiamap*)

do expisure normalization of an aia map :param aia map made from sunpy.map: :return: normalised aia map

suncasa.utils.DButil.**tplt**(*mapcube*)

suncasa.utils.DButil.**sdo_aia_scale_hdr**(*amap*, *sigma=None*)

suncasa.utils.DButil.**sdo_aia_scale_dict**(*wavelength=None*, *imagetype='image'*)

rescale the aia image :param image: normalised aia image data :param wavelength: :return: byte scaled image data

suncasa.utils.DButil.**sdo_aia_scale**(*image=None*, *wavelength=None*)

rescale the aia image :param image: normalised aia image data :param wavelength: :return: byte scaled image data

suncasa.utils.DButil.**insertchar**(*source_str*, *insert_str*, *pos*)

suncasa.utils.DButil.**readsdofile**(*datadir=None*, *wavelength=None*, *trange=None*, *isexists=False*, *timtol=1*)

read sdo file from local database :param datadir: :param wavelength: :param trange: the timestamp or timerange in Julian days. if is timerange, return a list of files in the timerange :param isexists: check if file exist. if files exist, return file name :param timtol: time difference tolerance in days for considering data as the same timestamp :return:

suncasa.utils.DButil.**readsdofileX**(*datadir=None*, *filelist=None*, *wavelength=None*, *trange=None*, *isexists=False*, *timtol=1*)

read sdo file from local database :param datadir: :param wavelength: :param trange: time object. the timestamp or timerange. if is timerange, return a list of files in the timerange :param isexists: check if file exist. if files exist, return file name :param timtol: time difference tolerance in days for considering data as the same timestamp :return:

suncasa.utils.DButil.**findDist**(*x*, *y*)

suncasa.utils.DButil.**paramspline**(*x*, *y*, *length*, *s=0*)

suncasa.utils.DButil.**polyfit**(*x*, *y*, *length*, *deg*, *keepxorder=False*)

suncasa.utils.DButil.**htfit_warren2011**(*x*, *y*, *cutlength*)

suncasa.utils.DButil.**spline**(*x*, *y*, *length*, *s=0*)

suncasa.utils.DButil.**get_curve_grad**(*x*, *y*)

get the grad of at data point :param x: :param y: :return: grad,posang

suncasa.utils.DButil.**improfile**(*z*, *xi*, *yi*, *interp='cubic'*)

Pixel-value cross-section along line segment in an image :param z: an image array :param xi and yi: equal-length vectors specifying the pixel coordinates of the endpoints of the line segment :param interp: interpolation type to sampling, 'nearest' or 'cubic' :return: the intensity values of pixels along the line

suncasa.utils.DButil.**canvaspix_to_data**(*smap*, *x*, *y*)

suncasa.utils.DButil.**data_to_mappixel**(*smap*, *x*, *y*)

suncasa.utils.DButil.**polsfromfitsheader**(*header*)

get polarisation information from fits header :param header: fits header :return pols: polarisation stokes

suncasa.utils.DButil.**headerfix**(*header*)

suncasa.utils.DButil.**freqsfromfitsheader**(*header*)

get frequency in GHz from fits header :param header: fits header :return pols: polarisation stokes

suncasa.utils.DButil.**transfitdict2DF**(*datain*, *gaussfit=True*, *getcentroid=False*)

convert the results from pimfit or pmaxfit tasks to pandas DataFrame structure. :param datain: The component list from pimfit or pmaxfit tasks :param gaussfit: True if the results is from pimfit, otherwise False. :param getcentroid: If True returns the centroid :return: the pandas DataFrame structure.

suncasa.utils.DButil.**getcolctinDF**(*dspecDF*, *col*)

return the count of how many times of the element starts with col occurs in columns of dspecDF :param dspecDF: :param col: the start string :return: the count and items

suncasa.utils.DButil.**dspecDFfilter**(*dspecDF*, *pol*)

filter the unselect polarisation from dspecDF :param dspecDF: the original dspecDF :param pol: selected polarisation, dtype = string :return: the output dspecDF

suncasa.utils.DButil.**dspecDF2text**(*DFfile*, *outfile=None*)

suncasa.utils.DButil.**smapmeshgrid2**(*smap*, *angle=None*, *rescale=1.0*, *origin=1*)

suncasa.utils.DButil.**map2wcsgrids**(*snpmap*, *cell=True*, *antialiased=True*)

Parameters

- **snpmap** –
- **cell** – if True, return the coordinates of the pixel centers. if False, return the coordinates of the pixel boundaries

Returns

suncasa.utils.DButil.**smapradialfilter**(*sunpymap*, *factor=5*, *grid=None*)

suncasa.utils.DButil.**regridimage**(*values*, *x*, *y*, *grid=None*, *resize=[1.0, 1.0]*)

re-grid the data on a regular grid with uneven grid spacing to an uniform grid :param values: The image data on the regular grid :param x: the points defining the regular grid in x :param y: the points defining the regular grid in y :param grid: new uniform mesh grid [gridx,gridy] :param resize: list of re-size ratio factors of x and y. if resize is not [1.0,1.0], grid is neglected. :return: re-gridded image

suncasa.utils.DButil.**regridspec**(*spec*, *x*, *y*, *nxmax=None*, *nymax=None*, *interp=False*)

Parameters

- **spec** – ndarray of float or complex, shape (npol,nbl,nf,nt) Data values.
- **x** – Data point x coordinates.
- **y** – Data point y coordinates.
- **nxmax** –
- **nymax** –

Returns

suncasa.utils.DButil.**get_contour_data**(*X*, *Y*, *Z*, *levels=[0.5, 0.7, 0.9]*)

suncasa.utils.DButil.**c_correlate**(*a*, *v*, *returnx=False*)

suncasa.utils.DButil.**c_correlateX**(*a*, *v*, *returnx=False*, *returnav=False*, *s=0*)

Parameters

- **a** –
- **v** – a and v can be a dict in following format {'x':[],'y':[]}. The length of a and v can be different.
- **returnx** –

Returns

suncasa.utils.DButil.**XCorrMap**(*z*, *x*, *y*, *doxscale=True*)

get the cross correlation map along y axis :param z: data :param x: x axis :param y: y axis :return:

class suncasa.utils.DButil.**ButtonsPlayCTRL**(*plot_width=None*, **args*, ***kwargs*)

Produce A play/stop button widget for bokeh plot

__slots__ = ['buttons']

suncasa.utils.fit_planet_position

Module Contents

Functions

fpoly(x, np)	
svdfit(xx, yy, sig, ma)	
svd(a)	Compute the singular value decomposition of array.
pythag(a, b)	
svbksb(uu, ww, vv, bb)	
svdvar(vv, ww, ma)	
fit_planet_positions(times, ras, decs[, start_time, ...])	find a fitting polynomial for an ephemeris table.

suncasa.utils.fit_planet_position.**fpoly**(*x*, *np*)

suncasa.utils.fit_planet_position.**svdfit**(*xx*, *yy*, *sig*, *ma*)

suncasa.utils.fit_planet_position.**svd**(*a*)

Compute the singular value decomposition of array.

suncasa.utils.fit_planet_position.**pythag**(*a*, *b*)

suncasa.utils.fit_planet_position.**svbksb**(*uu*, *ww*, *vv*, *bb*)

suncasa.utils.fit_planet_position.**svdvar**(*vv*, *ww*, *ma*)

suncasa.utils.fit_planet_position.**fit_planet_positions**(*times*, *ras*, *decs*, *start_time=None*, *end_time=None*, *distances=None*, *allowed_error=0.01*)

find a fitting polynomial for an ephemeris table.

inputs:

times = list of times at which the ephemeris positions are
tabulated. assumed sorted in ascending order. (MJD).

ras = list of right ascensions at those times (radians). decs = list of declinations at those times (radians).

Optional inputs:
start_time = the start time of the SB (MJD). end_time = the end time of the SB (MJD). distances = list of distances at those times (AU). allowed_error = the allowed error in the fitting polynomials

for ra and dec from the tabulated values (asec).

returned is a list, first element is the return status:
0 -> success 1-7 -> Warning: did not converge to required accuracy.

1 -> right ascension only didn't converge 2 -> declination only didn't converge 3 -> right ascension and declination didn't converge 4 -> distance only didn't converge 5 -> right ascension and distance didn't converge 6 -> declination and distance didn't converge 7 -> none of the three converged (note that in this case the best fitted polynomials are still returned [N.B. i should return the error somewhere…].)

8 -> Error: the time range from start_time to end_time is
not completely contained in the tabulated times.

second element is the time t0 to which the polynomial is
referenced.

third element is the list of right ascension coefficients. fourth element is the list of declination coefficients. fifth element is the list of distance coefficients.

bjb nrao summer 2012

suncasa.utils.helio_coordinates

Module Contents

Functions

hgs2hcc(rsun, lon, lat, B0, L0)	
hcc2hgs(x, y, z, B0, L0)	

Attributes

sin	
cos	

suncasa.utils.helio_coordinates.**sin**

suncasa.utils.helio_coordinates.**cos**

suncasa.utils.helio_coordinates.**hgs2hcc**(*rsun*, *lon*, *lat*, *B0*, *L0*)

suncasa.utils.helio_coordinates.**hcc2hgs**(*x*, *y*, *z*, *B0*, *L0*)

suncasa.utils.helioimage2fits

Module Contents

Functions

ms_clearhistory(msfile)	
normalize(angle[, lower, upper])	Function to normalize any angles to between the lower limit and upper limit
ms_restorehistory(msfile)	
read_horizons([t0, dur, vis, observatory, verbose])	This function visits JPL Horizons to retrieve J2000 topocentric RA and DEC of the solar disk center
read_msinfo([vis, msinfofile, interp_to_scan, verbose])	Module ot read CASA measurement set and return RA, DEC, and time stamps of the phase center.
ephem_to_helio([vis, ephem, msinfo, reftime, ...])	Module for calculating offsets of the phase center to the solar disk center using the following steps
getbeam([imagefile, beamfile])	
imreg([vis, imagefile, timerange, ephem, msinfo, ...])	main routine to register CASA images
calc_phasecenter_from_solxy(vis[, timerange, xycen, ...])	return the phase center in RA and DEC of a given solar coordinates

Attributes

py3
tools
tbtool
mstool
qatool
iatool
tb
ms
qa
ia
sunpy1

suncasa.utils.helioimage2fits.**py3**

suncasa.utils.helioimage2fits.**tools**

suncasa.utils.helioimage2fits.**tbtool**

suncasa.utils.helioimage2fits.**mstool**

suncasa.utils.helioimage2fits.**qatool**

suncasa.utils.helioimage2fits.**iatool**

suncasa.utils.helioimage2fits.**tb**

suncasa.utils.helioimage2fits.**ms**

suncasa.utils.helioimage2fits.**qa**

suncasa.utils.helioimage2fits.**ia**

suncasa.utils.helioimage2fits.**sunpy1**

suncasa.utils.helioimage2fits.**ms_clearhistory**(*msfile*)

suncasa.utils.helioimage2fits.**normalize**(*angle*, *lower=-np.pi*, *upper=np.pi*)

Function to normalize any angles to between the lower limit and upper limit :param angle: input angle in radians :param lower: lower bound of the normalization, default to -pi :param upper: upper bound of the normalization, default to pi :return: normalized angle in radians

suncasa.utils.helioimage2fits.**ms_restorehistory**(*msfile*)

suncasa.utils.helioimage2fits.**read_horizons**(*t0=None*, *dur=None*, *vis=None*, *observatory=None*, *verbose=False*)

This function visits JPL Horizons to retrieve J2000 topocentric RA and DEC of the solar disk center as a function of time.

Keyword arguments: t0: Referece time in astropy.Time format dur: duration of the returned coordinates in days. Default to 1 minute vis: CASA visibility dataset (in measurement set format). If provided, use entire duration from

the visibility data

observatory: observatory code (from JPL Horizons). If not provided, use information from visibility.
if no visibility found, use earth center (code=500)

verbose: True to provide extra information

Usage: >>> from astropy.time import Time >>> out = read_horizons(t0=Time('2017-09-10 16:00:00'), observatory='-81') >>> out = read_horizons(vis = 'mydata.ms')

History: BC (sometime in 2014): function was first wrote, followed by a number of edits by BC and SY BC (2019-07-16): Added docstring documentation

''

suncasa.utils.helioimage2fits.**read_msinfo**(*vis=None*, *msinfofile=None*, *interp_to_scan=False*, *verbose=False*)

Module ot read CASA measurement set and return RA, DEC, and time stamps of the phase center. Options for returning those from the emphemeris table (if available) or use those from the FIELD table. :param vis: :type vis: (required) path to the input CASA measurement set :param msinfofile: :type msinfofile: (optional) path/name of the saved numpy .npz file :param #use_ephem: :type #use_ephem: (optional) if True (default), use the enclosed emphemeris table, otherwise use RA/DEC of the FIELD table :param interp_to_scan: are interpolated to the beginning of each scan. This is mainly for backward compatibility. Default is False. :type interp_to_scan: (optional) if True, the entries of the emphemeris table

Returns

msinfo – vis: CASA measurement set observatory: Name of the observatory. May be used if they default to phase at the solar center. scans: summary info of the scans fieldids: a list of all FIELD ids scan_start_times: list of the start times of all scans, in mjd scan_end_times: list of the end times of all scans, in mjd btimes: time stamps used by ephem_to_helio() to determine the image/phase center shifts, in mjd ras: corresponding RA coordinates used by ephem_to_helio() to determine the image/phase center shifts, in rad decs: corresponding DEC coordinates used by ephem_to_helio() to determine the image/phase center shifts, in rad

Return type

A dictionary contains necessary information for ephem_to_helio()

suncasa.utils.helioimage2fits.**ephem_to_helio**(*vis=None*, *ephem=None*, *msinfo=None*, *reftime=None*, *dopolyfit=True*, *usephacenter=True*, *geocentric=False*, *verbose=False*)

Module for calculating offsets of the phase center to the solar disk center using the following steps 1. Take a solar ms database, read the scan and field information, find out the phase centers (in RA and DEC).

This step is done with read_msinfo()

2. Compare with the ephemeris of the solar disk center (in RA and DEC)
3. Record RA/DEC of the phase center and offsets in RA/DEC and Helioprojective Cartesian coordinates (solar X/Y) inputs:

msinfo: CASA MS information, output from read_msinfo ephem: solar ephem, output from read_horizons reftime: list of reference times (e.g., used for imaging)

CASA standard time format, either a single time (e.g., '2012/03/03/12:00:00' or a time range (e.g., '2012/03/03/12:00:00~2012/03/03/13:00:00'. If the latter, take the midpoint of the timerange for reference. If no date specified, take the date of the first scan

dopolyfit: Bool. Default: True. Works for MS database with only one source with continously tracking.

Disabled if usephacenter=False.

usephacenter: Bool – if True, correct for the RA and DEC in the ms file based on solar empheris.

Otherwise assume the phasecenter is pointed to the solar disk center (EOVSA case)

geocentric: Bool – if True, use geocentric RA & DEC.

If False, use topocentric RA & DEC based on observatory location Default: False

return values:

helio: a list of VLA pointing information

reftimestr: reference time, in FITS format string reftime: reference time, in mjd format ra: actual RA of phasecenter in the ms file at the reference time (interpolated) dec: actual DEC of

phasecenter in the ms file at the reference time (interpolated) # CASA uses only RA and DEC of the closest field (e.g. in clean) # ra_fld: RA of the CASA reference pointing direction, in radian dec_fld: DEC of the CASA reference pointing direction, in radian ra0: RA of the solar disk center, in radian dec0: DEC of the solar disk center, in radian raoff: RA offset of the phasecenter in the ms file to solar disk center, in arcsec decoff: DEC offset of the phasecenter in the ms file to solar disk center, in arcsec refx: heliocentric X offset of the phasecenter in the ms file to solar disk center, in arcsec refy: heliocentric Y offset of the phasecenter in the ms file to solar disk center, in arcsec has_ephem_table: flag to indicate if there is an ephemeris table attached.

######## Example #########

from suncasa.utils import helioimage2fits as hf vis = vis = '22B-174_20221031_sun.1s.cal.ms' ephem = hf.ephem_to_helio(vis=vis, reftime='2022/10/31/20:37:10~2022/10/31/20:37:20', dopolyfit=True,

usephacenter=True, verbose=True)

 # Read out the ms information takes some time. To save time, one can read out the ms information first

and supply the record here for registering multiple images. It will skip the read_msinfo() step.

msinfo = hf.read_msinfo(vis=vis, verbose=True) ephem = hf.ephem_to_helio(vis=vis, msinfo=msinfo, reftime='2022/10/31/20:37:10~2022/10/31/20:37:20',

dopolyfit=True, usephacenter=True, verbose=True)

suncasa.utils.helioimage2fits.**getbeam**(*imagefile=None*, *beamfile=None*)

suncasa.utils.helioimage2fits.**imreg**(*vis=None*, *imagefile=None*, *timerange=None*, *ephem=None*, *msinfo=None*, *fitsfile=None*, *usephacenter=True*, *geocentric=False*, *dopolyfit=True*, *reftime=None*, *offsetfile=None*, *beamfile=None*, *toTb=False*, *sclfactor=1.0*, *p_ang=None*, *overwrite=True*, *deletehistory=False*, *subregion=''*, *docompress=False*, *verbose=False*)

main routine to register CASA images

Required Inputs:

vis: STRING. CASA measurement set from which the image is derived imagefile: STRING or LIST. name of the input CASA image timerange: STRING or LIST. timerange used to generate the CASA image, must have the same length as the input images.

Each element should be in CASA standard time format, e.g., '2012/03/03/12:00:00~2012/03/03/13:00:00'

Optional Inputs:

msinfo: DICTIONARY. CASA MS information, output from read_msinfo. If not provided, generate one from the supplied vis ephem: DICTIONARY. solar ephem, output from read_horizons.

If not provided, query JPL Horizons based on time info of the vis (internet connection required)

fitsfile: STRING or LIST. name of the output registered fits files toTb: Bool. Convert the default Jy/beam to brightness temperature? sclfactor: scale the image values up by its value (e.g., sclfactor = 100 to compensate VLA 20 dB attenuator) p_ang: solar p angle in degrees. If provided, use the supplied value and ignore the empheris verbose: Bool. Show more diagnostic info if True. usephacenter: Bool – if True, correct for the RA and DEC in the ms file based on solar empheris.

Otherwise assume the phasecenter is correctly pointed to the solar disk center (EOVSA case)

geocentric: Bool – if True, use geocentric RA & DEC.
If False, use topocentric RA & DEC based on observatory location Default: False

dopolyfit: Bool – if True, fit the ephemeris from the measurement set using a polynomial fit.
if False, just use linear interpolation

The following two parameters are only meant for temporary fixes ########################## reftime: STRING or LIST. ONLY USED IF ANOTHER TIME (OTHER THAN TIME TO MAKE THE IMAGE)

IS NEEDED TO FIND RA AND DEC. Each element should be in CASA standard time format, e.g., '2012/03/03/12:00:00'

subregion: only write the data within the sub-region selection. See 'help par.region' for details.

Usage: >>> from suncasa.utils import helioimage2fits as hf >>> hf.imreg(vis='mydata.ms', imagefile='myimage.image', fitsfile='myimage.fits',

timerange='2017/08/21/20:21:10~2017/08/21/20:21:18')

The output fits file is 'myimage.fits'

History: BC (sometime in 2014): function was first wrote, followed by a number of edits by BC and SY BC (2019-07-16): Added checks for stokes parameter. Verified that for converting from Jy/beam to brightness temperature,

the convention of 2*k_b*T should always be used. I.e., for unpolarized source, stokes I, RR, LL, XX, YY, etc. in the output CASA images from (t)clean should all have same values of radio intensity (in Jy/beam) and brightness temperature (in K).

suncasa.utils.helioimage2fits.**calc_phasecenter_from_solxy**(*vis*, *timerange=''*, *xycen=None*, *usemsphacenter=True*, *observatory=None*)

return the phase center in RA and DEC of a given solar coordinates

Parameters

- **vis** – input measurement sets file
- **timerange** – can be a string or astropy.time.core.Time object, or a 2-element list of string or Time object
- **xycen** – solar x-pos and y-pos in arcsec
- **usemsphacenter** –

Returns

phasecenter midtim: mid time of the given timerange

suncasa.utils.idlsav2sunmap

Module Contents

Functions

idlsav2sunmap(idlsavfile)

suncasa.utils.idlsav2sunmap.**idlsav2sunmap**(*idlsavfile*)

suncasa.utils.jdutil

Functions for converting dates to/from JD and MJD. Assumes dates are historical dates, including the transition from the Julian calendar to the Gregorian calendar in 1582. No support for proleptic Gregorian/Julian calendars.

Author
Matt Davis

Website
http://github.com/jiffyclub

Module Contents

Classes

datetime	A subclass of *datetime.datetime* that performs math operations by first

Functions

mjd_to_jd(mjd)	Convert Modified Julian Day to Julian Day.
jd_to_mjd(jd)	Convert Julian Day to Modified Julian Day
date_to_jd(year, month, day)	Convert a date to Julian Day.
jd_to_date(jd)	Convert Julian Day to date.
hmsm_to_days([hour, min, sec, micro])	Convert hours, minutes, seconds, and microseconds to fractional days.
days_to_hmsm(days)	Convert fractional days to hours, minutes, seconds, and microseconds.
datetime_to_jd(date)	Convert a *datetime.datetime* object to Julian Day.
jd_to_datetime(jd)	Convert a Julian Day to an *jdutil.datetime* object.
timedelta_to_days(td)	Convert a *datetime.timedelta* object to a total number of days.

suncasa.utils.jdutil.**mjd_to_jd**(*mjd*)

Convert Modified Julian Day to Julian Day.

Parameters
mjd (*float*) – Modified Julian Day

Returns
jd – Julian Day

Return type
float

suncasa.utils.jdutil.**jd_to_mjd**(*jd*)

Convert Julian Day to Modified Julian Day

Parameters
jd (*float*) – Julian Day

Returns
mjd – Modified Julian Day

Return type
float

suncasa.utils.jdutil.**date_to_jd**(*year*, *month*, *day*)

Convert a date to Julian Day.

Algorithm from 'Practical Astronomy with your Calculator or Spreadsheet',
4th ed., Duffet-Smith and Zwart, 2011.

Parameters

- **year** (*int*) – Year as integer. Years preceding 1 A.D. should be 0 or negative. The year before 1 A.D. is 0, 10 B.C. is year -9.
- **month** (*int*) – Month as integer, Jan = 1, Feb. = 2, etc.
- **day** (*float*) – Day, may contain fractional part.

Returns
jd – Julian Day

Return type
float

Examples

Convert 6 a.m., February 17, 1985 to Julian Day

```
>>> date_to_jd(1985,2,17.25)
2446113.75
```

suncasa.utils.jdutil.**jd_to_date**(*jd*)

Convert Julian Day to date.

Algorithm from 'Practical Astronomy with your Calculator or Spreadsheet',
4th ed., Duffet-Smith and Zwart, 2011.

Parameters
jd (*float*) – Julian Day

Returns

- **year** (*int*) – Year as integer. Years preceding 1 A.D. should be 0 or negative. The year before 1 A.D. is 0, 10 B.C. is year -9.
- **month** (*int*) – Month as integer, Jan = 1, Feb. = 2, etc.
- **day** (*float*) – Day, may contain fractional part.

Examples

Convert Julian Day 2446113.75 to year, month, and day.

```
>>> jd_to_date(2446113.75)
(1985, 2, 17.25)
```

suncasa.utils.jdutil.**hmsm_to_days**(*hour=0*, *min=0*, *sec=0*, *micro=0*)

Convert hours, minutes, seconds, and microseconds to fractional days.

Parameters

- **hour** (*int, optional*) – Hour number. Defaults to 0.
- **min** (*int, optional*) – Minute number. Defaults to 0.
- **sec** (*int, optional*) – Second number. Defaults to 0.
- **micro** (*int, optional*) – Microsecond number. Defaults to 0.

Returns

days – Fractional days.

Return type

float

Examples

```
>>> hmsm_to_days(hour=6)
0.25
```

suncasa.utils.jdutil.**days_to_hmsm**(*days*)

Convert fractional days to hours, minutes, seconds, and microseconds. Precision beyond microseconds is rounded to the nearest microsecond.

Parameters

days (*float*) – A fractional number of days. Must be less than 1.

Returns

- **hour** (*int*) – Hour number.
- **min** (*int*) – Minute number.
- **sec** (*int*) – Second number.
- **micro** (*int*) – Microsecond number.

Raises

ValueError – If *days* is >= 1.

Examples

```
>>> days_to_hmsm(0.1)
(2, 24, 0, 0)
```

suncasa.utils.jdutil.**datetime_to_jd**(*date*)

Convert a *datetime.datetime* object to Julian Day.

Parameters
date (*datetime.datetime* instance) –

Returns
jd – Julian day.

Return type
float

Examples

```
>>> d = datetime.datetime(1985,2,17,6)
>>> d
datetime.datetime(1985, 2, 17, 6, 0)
>>> jdutil.datetime_to_jd(d)
2446113.75
```

suncasa.utils.jdutil.**jd_to_datetime**(*jd*)

Convert a Julian Day to an *jdutil.datetime* object.

Parameters
jd (*float*) – Julian day.

Returns
dt – *jdutil.datetime* equivalent of Julian day.

Return type
jdutil.datetime object

Examples

```
>>> jd_to_datetime(2446113.75)
datetime(1985, 2, 17, 6, 0)
```

suncasa.utils.jdutil.**timedelta_to_days**(*td*)

Convert a *datetime.timedelta* object to a total number of days.

Parameters
td (*datetime.timedelta* instance) –

Returns
days – Total number of days in the *datetime.timedelta* object.

Return type
float

Examples

```
>>> td = datetime.timedelta(4.5)
>>> td
datetime.timedelta(4, 43200)
>>> timedelta_to_days(td)
4.5
```

class suncasa.utils.jdutil.**datetime**

Bases: datetime.datetime

A subclass of *datetime.datetime* that performs math operations by first converting to Julian Day, then back to a *jdutil.datetime* object.

Addition works with *datetime.timedelta* objects, subtraction works with *datetime.timedelta*, *datetime.datetime*, and *jdutil.datetime* objects. Not all combinations work in all directions, e.g. *timedelta - datetime* is meaningless.

See also:

datetime.datetime
Parent class.

__add__(*other*)
Add a datetime and a timedelta.

__radd__(*other*)

__sub__(*other*)
Subtract two datetimes, or a datetime and a timedelta.

__rsub__(*other*)

to_jd()
Return the date converted to Julian Day.

to_mjd()
Return the date converted to Modified Julian Day.

suncasa.utils.lightcurves

Module Contents

Functions

lightcurves(timerange[, outdir, specfile, goes, ...])	

suncasa.utils.lightcurves.**lightcurves**(*timerange*, *outdir='./'*, *specfile=None*, *goes=True*, *hessifile=None*, *fermifile=None*, *ylog=False*, *hessi_smoth=0*, *dspec_cmap='cubehelix'*, *vmax=None*, *vmin=None*)

suncasa.utils.lineticks

Module Contents

Classes

LineTicks	

Functions

get_perp_vec(u1, u2[, direction])	Return the unit vector perpendicular to the vector u2-u1.
get_av_vec(u1, u2)	Return the average unit vector between u1 and u2.

suncasa.utils.lineticks.**get_perp_vec**(*u1*, *u2*, *direction=1*)

Return the unit vector perpendicular to the vector u2-u1.

suncasa.utils.lineticks.**get_av_vec**(*u1*, *u2*)

Return the average unit vector between u1 and u2.

class suncasa.utils.lineticks.**LineTicks**(*line*, *idx*, *tick_length*, *tick_shift=3.0*, *direction=1*, *label=None*, *label_color='k'*, *fontsize='xx-small'*, ***kwargs*)

add_ticks(*ax*)

on_change_lims(*ax*)

on_resize(*event*)

suncasa.utils.mod_slftbs

Module Contents

Functions

cpxx2yy([tb_in])	
concat([tb_in, tb_out])	

Attributes

tools	
tbtool	
tb	

suncasa.utils.mod_slftbs.**tools**

suncasa.utils.mod_slftbs.**tbtool**

suncasa.utils.mod_slftbs.**tb**

suncasa.utils.mod_slftbs.**cpxx2yy**(*tb_in=[]*)

suncasa.utils.mod_slftbs.**concat**(*tb_in=[]*, *tb_out=None*)

suncasa.utils.mstools

Module Contents

Functions

get_bandinfo(msfile[, spw, returnbdinfo])	get center frequencies of all spectral windows for msfile
get_bmsize(cfreq[, refbmsize, reffreq, minbmsize])	get beamsize at frequencies definded by cfreq based on refbmsize at reffreq
get_trange(msfile)	
time2filename(msfile[, timerange, spw, desc])	
msclearhistory(msfile)	Clears history in the a measurement sets file
clearflagrow(msfile[, mode])	**param msfile**
splitX(vis[, datacolumn2])	
flagcaltboutliers(caltable[, limit])	
modeltransfer(msfile[, spw, reference, transfer])	
concat_slftb([tb_in, tb_out])	
gaincalXY([vis, caltable, pols, msfileXY, gaintableXY])	
getmodel(vis[, spw])	
putmodel(vis[, spw, model])	

Attributes

tasks
split
tclean
casalog
clearcal
gaincal
tools
tbtool
mstool
qatool
tb
ms
qa

suncasa.utils.mstools.**tasks**

suncasa.utils.mstools.**split**

suncasa.utils.mstools.**tclean**

suncasa.utils.mstools.**casalog**

suncasa.utils.mstools.**clearcal**

suncasa.utils.mstools.**gaincal**

suncasa.utils.mstools.**tools**

suncasa.utils.mstools.**tbtool**

suncasa.utils.mstools.**mstool**

suncasa.utils.mstools.**qatool**

suncasa.utils.mstools.**tb**

suncasa.utils.mstools.**ms**

suncasa.utils.mstools.**qa**

suncasa.utils.mstools.**get_bandinfo**(*msfile*, *spw=None*, *returnbdinfo=False*)

get center frequencies of all spectral windows for msfile spw: [option] return the cfreq of spw. spw can be a a string or a list of string. The syntax of spw follows the standard spw Parameter in CASA if spw is not provided, return the cfreq of all spws in the msfile. return cfreqs is in GHz if returnbounds is True, return a dictionary including comprehensive freq information of the ms.

suncasa.utils.mstools.**get_bmsize**(*cfreq*, *refbmsize=70.0*, *reffreq=1.0*, *minbmsize=4.0*)

get beamsize at frequencies definded by cfreq based on refbmsize at reffreq cfreq: input frequencies at GHz refbmsize: reference beam size in arcsec reffreq: reference frequency in GHz minbmsize: minimum beam size in arcsec

suncasa.utils.mstools.**get_trange**(*msfile*)

suncasa.utils.mstools.**time2filename**(*msfile*, *timerange=''*, *spw=''*, *desc=False*)

suncasa.utils.mstools.**msclearhistory**(*msfile*)

Clears history in the a measurement sets file

Parameters
msfile – string The name of a measurement sets file

Returns

suncasa.utils.mstools.**clearflagrow**(*msfile*, *mode='clear'*)

Parameters

- **msfile** –
- **mode** – FLAG_ROW operation

default: 'clear': (default) clear the FLAG_ROW
'list': to list existing FLAG_ROW

Returns

suncasa.utils.mstools.**splitX**(*vis*, *datacolumn2='MODEL_DATA'*, ***kwargs*)

suncasa.utils.mstools.**flagcaltboutliers**(*caltable*, *limit=[]*)

suncasa.utils.mstools.**modeltransfer**(*msfile*, *spw=''*, *reference='XX'*, *transfer='YY'*)

suncasa.utils.mstools.**concat_slftb**(*tb_in=[]*, *tb_out=None*)

suncasa.utils.mstools.**gaincalXY**(*vis=None*, *caltable=None*, *pols='XXYY'*, *msfileXY=None*, *gaintableXY=None*, ***kwargs*)

suncasa.utils.mstools.**getmodel**(*vis*, *spw=None*)

suncasa.utils.mstools.**putmodel**(*vis*, *spw=None*, *model=None*)

suncasa.utils.plot_map

Module Contents

Functions

map2wcsgrids(sunpymap[, cell, antialiased])	**param sunpymap**
get_map_extent(sunpymap[, rot])	
imshow(sunpymap[, axes, rot])	**param sunpymap**
contour(sunpymap[, axes, rot])	
contourf(sunpymap[, axes, rot, mapx, mapy, rangereverse])	
imshow_RGB(maps[, axes, returndataonly])	

suncasa.utils.plot_map.**map2wcsgrids**(*sunpymap*, *cell=False*, *antialiased=True*)

Parameters

- **sunpymap** –
- **cell** – if True, return the coordinates of the pixel centers. if False, return the coordinates of the pixel boundaries

Returns

suncasa.utils.plot_map.**get_map_extent**(*sunpymap*, *rot=0*)

suncasa.utils.plot_map.**imshow**(*sunpymap*, *axes=None*, *rot=0*, ***kwargs*)

Parameters

- **sunpymap** –
- **axes** –
- **rot** – rotation angle in degrees counter-clockwise. Must be an integer multiple of 90.
- **kwargs** –

Returns

suncasa.utils.plot_map.**contour**(*sunpymap*, *axes=None*, *rot=0*, ***kwargs*)

suncasa.utils.plot_map.**contourf**(*sunpymap*, *axes=None*, *rot=0*, *mapx=None*, *mapy=None*, *rangereverse=False*, ***kwargs*)

suncasa.utils.plot_map.**imshow_RGB**(*maps*, *axes=None*, *returndataonly=False*)

suncasa.utils.plot_mapX

Module Contents

Classes

Sunmap

class suncasa.utils.plot_mapX.**Sunmap**(*sunmap*, *aia=False*)

map2wcsgrids(*sunpymap=None*, *cell=False*)

Parameters

- **sunpymap** –
- **cell** – if True, return the coordinates of the pixel centers. if False, return the coordinates of the pixel boundaries

Returns

get_map_extent(*sunpymap=None*, *rot=0*, *rangereverse=False*)

imshow(*axes=None*, *rot=0*, *rangereverse=False*, *maskon=False*, *image_enhance=False*, ***kwargs*)

Parameters

- **sunpymap** –
- **axes** –
- **rot** – rotation angle in degrees counter-clockwise. Must be an integer multiple of 90.
- **kwargs** –

Returns

contour(*axes=None*, *rot=0*, *mapx=None*, *mapy=None*, *rangereverse=False*, ***kwargs*)

contourf(*axes=None*, *rot=0*, *mapx=None*, *mapy=None*, *rangereverse=False*, ***kwargs*)

draw_limb(*axes=None*, *rangereverse=False*, ***kwargs*)

draw_grid(*axes=None*, *rot=0*, *grid_spacing=None*, ***kwargs*)

draw_rectangle(*bottom_left*, *width*, *height*, *axes=None*, ***kwargs*)

imshow_RGB(*maps*, *axes=None*, *returndataonly=False*, *rangereverse=False*)

suncasa.utils.pltutils

Module Contents

Functions

multicolor_text(axes, x, y, textin[, cmap, wratio, bbox])	
align_marker(marker[, halign, valign])	

suncasa.utils.pltutils.**multicolor_text**(*axes*, *x*, *y*, *textin*, *cmap=None*, *wratio=0.5*, *bbox={}*, ***kw*)

suncasa.utils.pltutils.**align_marker**(*marker*, *halign='center'*, *valign='middle'*)

suncasa.utils.qlookplot

Module Contents

Functions

validate_and_reset_restoringbeam(restoringbm)	Validates the format of the restoringbeam string. If the format is incorrect,
read_imres(imresfile)	
checkspecnan(spec)	
get_goes_data([t, sat_num])	Reads GOES data from https://umbra.nascom.nasa.gov/ repository, for date
ms_clearhistory(msfile)	
ms_restorehistory(msfile)	
get_mapcube_time(mapcube)	
uniq(lst)	
get_colorbar_params(fbounds[, stepfactor])	
download_jp2(tstart, tend, wavelengths, outdir)	
downloadAIAdata(trange[, wavelength, cadence, outdir])	
trange2aiafits(trange, aiawave, aiadir)	
parse_rdata(rdata, meta[, icmap, stokes, sp, ...])	rdata, meta: (required) data and header of a fits file readed using ndfits.read
mk_qlook_image(vis[, ncpu, timerange, twidth, stokes, ...])	
plt_qlook_image(imres[, timerange, spwplt, figdir, ...])	Required inputs:
dspec_external(vis[, workdir, specfile, ds_normalised])	
qlookplot(vis[, timerange, spw, spwplt, workdir, ...])	Generate quick-look plots and dynamic spectra for solar radio observations.

Attributes

systemname
sunpy1
sunpy3
py3
tasks
split
tclean
casalog
tools
tbtool
mstool
qatool
tb
ms
qa
c_external
sunpy1
polmap
aiadir_default

suncasa.utils.qlookplot.**systemname**

suncasa.utils.qlookplot.**sunpy1**

suncasa.utils.qlookplot.**sunpy3**

suncasa.utils.qlookplot.**py3**

suncasa.utils.qlookplot.**tasks**

suncasa.utils.qlookplot.**split**

suncasa.utils.qlookplot.**tclean**

suncasa.utils.qlookplot.**casalog**

suncasa.utils.qlookplot.**tools**

suncasa.utils.qlookplot.**tbtool**

suncasa.utils.qlookplot.**mstool**

suncasa.utils.qlookplot.**qatool**

suncasa.utils.qlookplot.**tb**

suncasa.utils.qlookplot.**ms**

suncasa.utils.qlookplot.**qa**

suncasa.utils.qlookplot.**c_external = False**

suncasa.utils.qlookplot.**sunpy1**

suncasa.utils.qlookplot.**polmap**

suncasa.utils.qlookplot.**validate_and_reset_restoringbeam**(*restoringbm*)

Validates the format of the restoringbeam string. If the format is incorrect, it prints an error message and resets the value to an empty string.

Parameters: - restoringbeam (str): The restoring beam size to validate.

Returns: - str: The original restoringbeam if valid, or an empty string if invalid.

suncasa.utils.qlookplot.**read_imres**(*imresfile*)

suncasa.utils.qlookplot.**checkspecnan**(*spec*)

suncasa.utils.qlookplot.**get_goes_data**(*t=None*, *sat_num=None*)

Reads GOES data from https://umbra.nascom.nasa.gov/ repository, for date and satellite number provided. If sat_num is None, data for all available satellites are downloaded, with some sanity check used to decide the best. If the Time() object t is None, data for the day before the current date are read (since there is a delay of 1 day in availability of the data). :returns: goes_t GOES time array in plot_date format

goes_data GOES 1-8 A lightcurve

suncasa.utils.qlookplot.**ms_clearhistory**(*msfile*)

suncasa.utils.qlookplot.**ms_restorehistory**(*msfile*)

suncasa.utils.qlookplot.**aiadir_default = '/srg/data/sdo/aia/level1/'**

suncasa.utils.qlookplot.**get_mapcube_time**(*mapcube*)

suncasa.utils.qlookplot.**uniq**(*lst*)

suncasa.utils.qlookplot.**get_colorbar_params**(*fbounds*, *stepfactor=1*)

suncasa.utils.qlookplot.**download_jp2**(*tstart*, *tend*, *wavelengths*, *outdir*)

suncasa.utils.qlookplot.**downloadAIAdata**(*trange*, *wavelength=None*, *cadence=None*, *outdir='./'*)

suncasa.utils.qlookplot.**trange2aiafits**(*trange*, *aiawave*, *aiadir*)

suncasa.utils.qlookplot.**parse_rdata**(*rdata*, *meta*, *icmap=None*, *stokes='I,V'*, *sp=None*, *show_warnings=False*)

rdata, meta: (required) data and header of a fits file readed using ndfits.read icmap: (optional) colormap for plotting radio images stokes: (optional) polarizations to visualizing sp: (optional) the spectral window to plot, if there are multiple spectral windows in the fits file :returns: * **cmaps** (*A dictionary contains colormaps for the selected polarizations*)

- **datas** (*A dictionary contains image data for the selected polarizations*)

suncasa.utils.qlookplot.**mk_qlook_image**(*vis*, *ncpu=1*, *timerange=''*, *twidth=12*, *stokes='I,V'*, *antenna=''*, *imagedir=None*, *spws=[]*, *toTb=True*, *sclfactor=1.0*, *overwrite=True*, *doslfcal=False*, *datacolumn='data'*, *phasecenter=''*, *robust=0.0*, *niter=500*, *gain=0.1*, *imsize=[512]*, *cell=['5.0arcsec']*, *pbcor=True*, *reftime=''*, *restoringbeam=['']*, *refbmsize=70.0*, *reffreq=1.0*, *minbmsize=4.0*, *mask=''*, *docompress=False*, *wrapfits=True*, *uvrange=''*, *subregion=''*, *c_external=True*, *show_warnings=False*)

suncasa.utils.qlookplot.**plt_qlook_image**(*imres*, *timerange=''*, *spwplt=None*, *figdir=None*, *specdata=None*, *verbose=True*, *stokes='I,V'*, *fov=None*, *imax=None*, *imin=None*, *icmap=None*, *inorm=None*, *amax=None*, *amin=None*, *acmap=None*, *anorm=None*, *nclevels=None*, *dmax=None*, *dmin=None*, *dcmap=None*, *dnorm=None*, *sclfactor=1.0*, *clevels=None*, *aiafits=''*, *aiadir=None*, *aiawave=171*, *plotaia=True*, *freqbounds=None*, *moviename=''*, *alpha_cont=1.0*, *custom_mapcubes=[]*, *opencontour=False*, *movieformat='html'*, *ds_normalised=False*)

Required inputs: Important optional inputs: Optional inputs:

aiadir: directory to search aia fits files

Example: :param imres: :param timerange: :param figdir: :param specdata: :param verbose: :param stokes: :param fov: :param imax: ## radio image plot setting :param imin: :param icmap: :param inorm: :param amax: ## aia plot setting :param amin: :param acmap: :param anorm: :param nclevels: :param dmax: ## dynamic spectra plot setting :param dmin: :param dcmap: :param dnorm: :param sclfactor: :param clevels: :param aiafits: :param aiadir: :param aiawave: :param plotaia: :param moviename: :param alpha_cont: :param custom_mapcubes: :return:

suncasa.utils.qlookplot.**dspec_external**(*vis*, *workdir='./'*, *specfile=None*, *ds_normalised=False*)

suncasa.utils.qlookplot.**qlookplot**(*vis*, *timerange=None*, *spw=''*, *spwplt=None*, *workdir='./'*, *specfile=None*, *xycen=None*, *fov=[500.0, 500.0]*, *xyrange=None*, *restoringbeam=['']*, *refbmsize=70.0*, *reffreq=1.0*, *minbmsize=4.0*, *antenna=''*, *uvrange=''*, *stokes='RR,LL'*, *robust=0.0*, *weighting='briggs'*, *niter=500*, *imsize=[512]*, *cell=['5.0arcsec']*, *mask=''*, *gain=0.1*, *pbcor=True*, *interactive=False*, *datacolumn='data'*, *reftime=''*, *toTb=True*, *sclfactor=1.0*, *subregion=''*, *usemsphacenter=True*, *imagefile=None*, *outfits=''*, *docompress=True*, *wrapfits=True*, *nclevels=3*, *clevels=None*, *calpha=0.5*, *opencontour=False*, *imax=None*, *imin=None*, *icmap=None*, *inorm=None*, *dmin=None*, *dmax=None*, *dcmap=None*, *dnorm=None*, *plotaia=True*, *aiawave=171*, *aiafits=None*, *aiadir=None*, *amax=None*, *amin=None*, *acmap=None*, *anorm=None*, *goestime=None*, *mkmovie=False*, *ncpu=1*, *twidth=1*, *movieformat='html'*, *cleartmpfits=True*, *overwrite=True*, *clearmshistory=False*, *show_warnings=False*, *verbose=False*, *quiet=False*, *ds_normalised=False*)

Generate quick-look plots and dynamic spectra for solar radio observations. Required inputs:

param vis
Path to the calibrated CASA measurement set.

Important optional inputs:

timerange: Timerange for analysis in standard CASA format. Defaults to entire range, which can be slow. spw: spectral window (SPW) selection following the CASA syntax.

Examples: spw='1:2~60' (spw id 1, channel range 2-60); spw='*:1.2~1.3GHz' (selects all channels within 1.2-1.3 GHz; note the *) spw can be a list of spectral windows, i.e, ['0', '1', '2', '3', '4', '5', '6', '7']

spwplt: Subset of SPW to display, defaults to all specified in *spw*. workdir: Working directory for temporary files, defaults to current directory. specfile: Path to a saved dynamic spectrum file (from suncasa.dspec.dspec.Dspec())

6or generate a median dynamic spectrum on the fly if not provided.

Optional inputs:

goestime: goes plot time, example ['2016/02/18 18:00:00','2016/02/18 23:00:00'] xycen: center of the image in helioprojective coordinates (HPLN/HPLT), in arcseconds. Example: [900, -150.] fov: field of view in arcsecs. Example: [500., 500.] xyrange: field of view in solar XY coordinates. Format: [[x1,x2],[y1,y2]]. Example: [[900., 1200.],[0,300]]

NOTE: THIS PARAMETER OVERWRITES XYCEN AND FOV

Restoring Beam Parameters:

restoringbeam: A list specifying the sizes of the restoring beam explicitly in arc seconds (e.g., ['100arcsec', '80arcsec', '50arcsec', ...]).
Must match the number of SPWs if not ['']. If specified, these values override automatic beam size calculations for each SPW.

refbmsize: The reference beam size in arc seconds. This parameter is used in conjunction with *reffreq* to calculate the beam size for all SPWs,
assuming the beam size is inversely proportional to the frequency. The parameters *refbmsize*,`reffreq` and *minbmsize* are only used if *restoringbeam* is set to [''].

reffreq: The reference frequency in GHz, used together with *refbmsize* to calculate the beam sizes for SPWs.

minbmsize: Minimum beam size in arcseconds, overrides smaller calculated sizes.

CASA tclean parameters: refer to CASA tclean documentation for more details.

antenna: baseline to generate dynamic spectrum uvrange: uvrange to select baselines for generating dynamic spectrum stokes: polarization of the clean image, can be 'RR,LL' or 'I,V' robust: weighting: niter: imsize: cell: mask: only accept CASA region format (https://casaguides.nrao.edu/index.php/CASA_Region_Format) gain: pbcor: interactive: datacolumn:

image registration parameters:

reftime: Reference time for image alignment. toTb: Bool. Convert the default Jy/beam to brightness temperature? sclfactor: scale the image values by its value (e.g., sclfactor = 100 to compensate VLA 20 dB attenuator) subregion: only write the data within the sub-region selection. See 'help par.region' for details. usephacenter: Bool – if True, correct for the RA and DEC in the ms file based on solar empheris.

Otherwise assume the phasecenter is correctly pointed to the solar disk center (EOVSA case)

imagefile: Use specified CASA radio image file for registration; otherwise, generate anew. outfits: Use specified FITS file of a radio image for output; otherwise, generate anew. docompress: if compress the outfits wrapfits: if wrap the fits files of multiple spectral windows at one given time interval into a combined fits file. overwrite: if overwrite the existed outfits file (default: True).

radio image plotting parameters:

nclevels: Number of contour levels for radio image plots. clevels: Specific contour levels for radio image plots. opencontour: Boolean. Plots open contours if True; filled contours otherwise. icmap: Color map (string or Colormap object) for radio images/contours. imax, imin: Color scale range, defining normalization before color mapping. inorm: Normalization method (string or Normalize object), overriding imax and imin.

radio dynamic spectrum plotting parameters:

dcmap: Color map (string or Colormap object) for the dynamic spectrum. dmin, dmax: Color scale range for dynamic spectrum normalization before color mapping. dnorm: Normalization method (string or Normalize object), overriding dmax and dmin.

SDO/AIA image plotting parameters:

plotaia: Boolean. Downloads and plots AIA image at specified aiawave if True. aiawave: AIA image passband to download and display. aiafits: Directly plots AIA image from provided FITS file, skipping download. (note: users can provide any solar image FITS file for plotting). aiadir: Searches this directory for AIA image files to skip download. acmap: Color map (string or Colormap object) for AIA images. amin, amax: Color scale range for AIA image normalization before color mapping. anorm: Normalization method (string or Normalize object), overriding amax and amin.

movie parameters:

mkmovie: Boolean. Generates a movie from radio images over multiple time intervals if True. ncpu: Number of CPUs for parallel clean operations with ptclean. twidth: Time pixel averaging width (default: 1). movieformat: Output movie format, either 'html' or 'mp4'.

suncasa.utils.radio_data_fetch

Module Contents

Functions

get_rstn_data(time[, outdir, ylim])	

suncasa.utils.radio_data_fetch.**get_rstn_data**(*time*, *outdir='./RSTN/'*, *ylim=[0, 800]*)

suncasa.utils.signal_utils

Module Contents

Functions

normalize(y[, ymax, ymin, center, yerr, symgamma])	**param y**
smooth(x[, window_len, window, mode])	smooth the data using a window with requested size.
butter_lowpass(cutoff, fs[, order])	
butter_lowpass_filter(data, cutoff, fs[, order])	
lowps_filter(data, cutoff, fs, ix)	
low_pass_filter(t, data[, fs, cutoff, order, showplot])	
bandpass_filter(t, data[, fs, cutoff, order, showplot])	
c_correlateX(a, v[, returnx, returnav, s, xran, ...])	**param a**
get_xcorr_info(xcorr[, cwidth_guess, showplt, verbose])	calculate the error in time lag using equation (3) in Gaskell & Peterson 1987
plot_wavelet(t, dat, dt, pl, pr[, period_pltlim, ax, ...])	

suncasa.utils.signal_utils.**normalize**(*y*, *ymax=None*, *ymin=None*, *center=None*, *yerr=None*, *symgamma=None*)

Parameters

- **y** –
- **ymax** –
- **ymin** –

- **center** – option —- None, zero, 0, mean
- **symgamma** –

Returns

suncasa.utils.signal_utils.**smooth**(*x*, *window_len=11*, *window='hanning'*, *mode='same'*)

smooth the data using a window with requested size.

This method is based on the convolution of a scaled window with the signal. The signal is prepared by introducing reflected copies of the signal (with the window size) in both ends so that transient parts are minimized in the begining and end part of the output signal.

input:
x: the input signal window_len: the dimension of the smoothing window; should be an odd integer window: the type of window from 'flat', 'hanning', 'hamming', 'bartlett', 'blackman'

flat window will produce a moving average smoothing.

output:
the smoothed signal

example:

t=linspace(-2,2,0.1) x=sin(t)+randn(len(t))*0.1 y=smooth(x)

see also:

numpy.hanning, numpy.hamming, numpy.bartlett, numpy.blackman, numpy.convolve scipy.signal.lfilter

TODO: the window parameter could be the window itself if an array instead of a string NOTE: length(output) != length(input), to correct this: return y[(window_len/2-1):-(window_len/2)] instead of just y.

suncasa.utils.signal_utils.**butter_lowpass**(*cutoff*, *fs*, *order=5*)

suncasa.utils.signal_utils.**butter_lowpass_filter**(*data*, *cutoff*, *fs*, *order=5*)

suncasa.utils.signal_utils.**lowps_filter**(*data*, *cutoff*, *fs*, *ix*)

suncasa.utils.signal_utils.**low_pass_filter**(*t*, *data*, *fs=1.0 / 4*, *cutoff=1.0 / 60*, *order=6*, *showplot=False*)

suncasa.utils.signal_utils.**bandpass_filter**(*t*, *data*, *fs=1.0 / 4*, *cutoff=1.0 / 60*, *order=6*, *showplot=False*)

suncasa.utils.signal_utils.**c_correlateX**(*a*, *v*, *returnx=False*, *returnav=False*, *s=0*, *xran=None*, *coarse=False*, *interp='spl'*)

Parameters

- **a** –
- **v** – a and v can be a dict in following format {'x':[],'y':[]}. The length of a and v can be different.
- **returnx** –

Returns

suncasa.utils.signal_utils.**get_xcorr_info**(*xcorr*, *cwidth_guess=2.5 / 24 / 60*, *showplt=False*, *verbose=False*)

calculate the error in time lag using equation (3) in Gaskell & Peterson 1987 :param xcorr: :param cwidth_guess: :return:

suncasa.utils.signal_utils.**plot_wavelet**(*t*, *dat*, *dt*, *pl*, *pr*, *period_pltlim=None*, *ax=None*, *ax2=None*, *stscale=2*, *siglev=0.95*, *cmap='viridis'*, *title=''*, *levels=None*, *label=''*, *units=''*, *tunits=''*, *sav_img=False*)

suncasa.utils.signalsmooth

cookb_signalsmooth.py

from: http://scipy.org/Cookbook/SignalSmooth

Module Contents

Functions

smooth(x[, window_len, window])	smooth the data using a window with requested size.
gauss_kern(size[, sizey])	Returns a normalized 2D gauss kernel array for convolutions
blur_image(im, n[, ny])	blurs the image by convolving with a gaussian kernel of typical
smooth_demo()	

Attributes

Z

suncasa.utils.signalsmooth.**smooth**(*x*, *window_len=10*, *window='hanning'*)

smooth the data using a window with requested size.

This method is based on the convolution of a scaled window with the signal. The signal is prepared by introducing reflected copies of the signal (with the window size) in both ends so that transient parts are minimized in the begining and end part of the output signal.

input:

x: the input signal window_len: the dimension of the smoothing window window: the type of window from 'flat', 'hanning', 'hamming', 'bartlett', 'blackman'

flat window will produce a moving average smoothing.

output:

the smoothed signal

example:

import numpy as np t = np.linspace(-2,2,0.1) x = np.sin(t)+np.random.randn(len(t))*0.1 y = smooth(x)

see also:

numpy.hanning, numpy.hamming, numpy.bartlett, numpy.blackman, numpy.convolve scipy.signal.lfilter

NOTE from B. Chen: slightly modified the reflected copies: window_len-1 points are added to both ends. Previous one is not exactly the reflection, the indices are off by one pixel

suncasa.utils.signalsmooth.**gauss_kern**(*size*, *sizey=None*)

Returns a normalized 2D gauss kernel array for convolutions

suncasa.utils.signalsmooth.**blur_image**(*im*, *n*, *ny=None*)

blurs the image by convolving with a gaussian kernel of typical size n. The optional keyword argument ny allows for a different size in the y direction.

suncasa.utils.signalsmooth.**smooth_demo**()

suncasa.utils.signalsmooth.**Z**

suncasa.utils.stackplot

Module Contents

Classes

LightCurveBuilder
SpaceTimeSlitBuilder
CutslitBuilder
Stackplot

Functions

resettable(f)	
b_filter(data, lowcut, highcut, fs, ix)	
runningmean(data, window, ix)	
c_correlate(a, v[, returnx])	
XCorrMap(data[, refpix])	
XCorrStackplt(z, x, y[, doxscale])	get the cross correlation map along y axis
FitSlit(xx, yy, cutwidth, cutang, cutlength[, s, ...])	
MakeSlit(pointDF)	
getimprofile(data, cutslit[, xrange, yrange, get_peak])	
smooth(x[, window_len, window])	smooth the data using a window with requested size.
grid(x, y, z[, resX, resY])	Convert 3 column data to matplotlib grid
polyfit(x, y, length, deg)	

suncasa.utils.stackplot.**resettable**(*f*)

suncasa.utils.stackplot.**b_filter**(*data*, *lowcut*, *highcut*, *fs*, *ix*)

suncasa.utils.stackplot.**runningmean**(*data*, *window*, *ix*)

suncasa.utils.stackplot.**c_correlate**(*a*, *v*, *returnx=False*)

suncasa.utils.stackplot.**XCorrMap**(*data*, *refpix=[0, 0]*)

suncasa.utils.stackplot.**XCorrStackplt**(*z*, *x*, *y*, *doxscale=True*)

get the cross correlation map along y axis :param z: data :param x: x axis :param y: y axis :return:

suncasa.utils.stackplot.**FitSlit**(*xx*, *yy*, *cutwidth*, *cutang*, *cutlength*, *s=None*, *method='Polyfit'*, *ascending=True*)

suncasa.utils.stackplot.**MakeSlit**(*pointDF*)

suncasa.utils.stackplot.**getimprofile**(*data*, *cutslit*, *xrange=None*, *yrange=None*, *get_peak=False*)

suncasa.utils.stackplot.**smooth**(*x*, *window_len=11*, *window='hanning'*)

smooth the data using a window with requested size.

This method is based on the convolution of a scaled window with the signal. The signal is prepared by introducing reflected copies of the signal (with the window size) in both ends so that transient parts are minimized in the begining and end part of the output signal.

input:

x: the input signal window_len: the dimension of the smoothing window; should be an odd integer window: the type of window from 'flat', 'hanning', 'hamming', 'bartlett', 'blackman'

flat window will produce a moving average smoothing.

output:
the smoothed signal

example:

t=linspace(-2,2,0.1) x=sin(t)+randn(len(t))*0.1 y=smooth(x)

see also:

numpy.hanning, numpy.hamming, numpy.bartlett, numpy.blackman, numpy.convolve scipy.signal.lfilter

NOTE: length(output) != length(input), to correct this: return y[(window_len/2-1):-(window_len/2)] instead of just y.

suncasa.utils.stackplot.**grid**(*x*, *y*, *z*, *resX=20*, *resY=40*)

Convert 3 column data to matplotlib grid

suncasa.utils.stackplot.**polyfit**(*x*, *y*, *length*, *deg*)

class suncasa.utils.stackplot.**LightCurveBuilder**(*stackplt*, *axes*, *scale=1.0*, *color='white'*)

__call__(*event*)

update(*mask=None*)

save(*event*)

delete(*event*)

delete_byindex(*index*)

update_text()

lightcurves_tofile(*outfile=None*, *lightcurves=None*)

lightcurves_fromfile(*infile*, *color=None*)

class suncasa.utils.stackplot.**SpaceTimeSlitBuilder**(*axes*, *cutlength=80*, *cutsmooth=10.0*, *scale=1.0*, *color='white'*)

__call__(*event*)

FitSlit(*xx*, *yy*, *cutlength*, *method='Polyfit'*, *s=0*, *ascending=True*)

polynomial fit

update(*mask=None*)

save(*event*)

delete(*event*)

delete_byindex(*index*)

update_text()

spacetimeslits_tofile(*outfile=None*, *spacetimeslits=None*)

spacetimeslits_fromfile(*infile*, *color=None*)

class suncasa.utils.stackplot.**CutslitBuilder**(*axes*, *cutwidth=5*, *cutang=0*, *cutlength=80*, *cutsmooth=10.0*, *scale=1.0*)

__call__(*event*)

update(*mask=None*)

class suncasa.utils.stackplot.Stackplot(*infile=None*)

property cutslit

property tplt

instrum_meta

aia_lvl1

suncasadb

mapcube

mapcube_diff

mapcube_plot

cutslitbd

stackplt

trange

wavelength

fitsfile

exptime_orig

fov

binpix

dt_data

divider_im

divider_dspec

sCutwdth

sCutang

sCutlngth

fig_mapcube

get_plot_title(*smap*, *title*)

plot_map(*smap*, *dspec=None*, *diff=False*, *norm=None*, *cmap=None*, *SymLogNorm=False*, *linthresh=0.5*, *returnImAx=False*, *layout_vert=False*, *uni_cm=False*, *draw_limb=False*, *draw_grid=False*, *colortitle=None*, *title=['observatory', 'detector', 'wavelength', 'time']*, *fov=fov*, **args*, ***kwargs*)

make_mapcube(*trange*, *outfile=None*, *fov=None*, *wavelength='171'*, *binpix=1*, *dt_data=1*, *derotate=False*, *tosave=True*, *superpixel=False*, *aia_prep=False*, *mapinterp=False*)

mapcube_fromfile(*infile*)

mapcube_tofile(*outfile=None*, *mapcube=None*)

mapcube_drot()

mapcube_resample(*binpix=1*)

mapcube_diff_denoise(*log=False*, *vmax=None*, *vmin=None*)

mapcube_mkdiff(*mode='rdiff'*, *dt=36.0*, *medfilt=None*, *gaussfilt=None*, *bfilter=False*, *lowcut=1 / 10 / 60.0*, *highcut=1 / 1 / 60.0*, *window=[None, None]*, *outfile=None*, *tosave=False*)

Parameters

- **mode** – accept modes: rdiff, rratio, bdiff, bratio, dtrend
- **dt** – time difference in second between frames when [rdiff, rratio, bdiff, bratio] is invoked
- **medfilt** –
- **gaussfilt** –
- **bfilter** – do butter bandpass filter
- **lowcut** – low cutoff frequency in Hz
- **highcut** – high cutoff frequency in Hz
- **outfile** –
- **tosave** –

Returns

plot_mapcube(*mapcube=None*, *hdr=False*, *norm=None*, *vmax=None*, *vmin=None*, *cmap=None*, *diff=False*, *sav_img=False*, *out_dir=None*, *dpi=100*, *anim=False*, *silent=False*, *draw_limb=False*, *draw_grid=False*, *colortitle=None*, *title=['observatory', 'detector', 'wavelength', 'time']*, *fov=[]*, *fps=15*)

Parameters

- **mapcube** –
- **hdr** –
- **vmax** –
- **vmin** –
- **diff** –
- **sav_img** –
- **out_dir** –
- **dpi** –
- **anim** –

Returns

cutslit_fromfile(*infile*, *color=None*, *mask=None*)

cutslit_tofile(*outfile=None*, *cutslit=None*)

make_stackplot(*mapcube*, *frm_range=[]*, *threshold=None*, *gamma=1.0*, *get_peak=False*)

stackplt_wrap()

stackplt_tofile(*outfile=None*, *stackplt=None*)

stackplt_fromfile(*infile*, ***kwargs*)

plot_stackplot(*mapcube=None*, *hdr=False*, *norm=None*, *vmax=None*, *vmin=None*, *cmap=None*, *layout_vert=False*, *diff=False*, *uni_cm=True*, *sav_img=False*, *out_dir=None*, *dpi=100*, *anim=False*, *frm_range=[]*, *cutslitplt=None*, *silent=False*, *refresh=True*, *threshold=None*, *gamma=1.0*, *get_peak=False*)

stackplt_traject_fromfile(*infile*, *frm_range=[]*, *cmap='inferno'*, *vmax=None*, *vmin=None*, *gamma=1.0*)

stackplt_lghtcurv_fromfile(*infile*, *frm_range=[]*, *cmap='inferno'*, *vmax=None*, *vmin=None*, *gamma=1.0*, *log=False*, *axes=None*)

mapcube_info(*mapcube=None*)

classmethod set_fits_dir(*fitsdir*)

suncasa.utils.stackplotX

Module Contents

Classes

LightCurveBuilder
SpaceTimeSlitBuilder
CutslitBuilder
Stackplot

Functions

aiaprep(sunpymap)	
resettable(f)	
b_filter(data, lowcut, highcut, fs, ix)	
runningmean(data, window, mode, ix)	**param data**
c_correlate(a, v[, returnx])	
XCorrMap(data[, refpix])	
XCorrStackplt(z, x, y[, doxscale])	get the cross correlation map along y axis
FitSlit(xx, yy, cutwidth, cutang, cutlength[, s, ...])	
MakeSlit(pointDF)	
getimprofile(data, cutslit[, xrange, yrange, ...])	Get values at a slice
smooth(x[, window_len, window])	smooth the data using a window with requested size.
grid(x, y, z[, resX, resY])	Convert 3 column data to matplotlib grid
polyfit(x, y, length, deg)	

Attributes

sunpy1

suncasa.utils.stackplotX.**sunpy1**

suncasa.utils.stackplotX.**aiaprep**(*sunpymap*)

suncasa.utils.stackplotX.**resettable**(*f*)

suncasa.utils.stackplotX.**b_filter**(*data*, *lowcut*, *highcut*, *fs*, *ix*)

suncasa.utils.stackplotX.**runningmean**(*data*, *window*, *mode*, *ix*)

Parameters

- **data** –
- **window** –
- **ix** –
- **mode** – available options are ratio and diff

Returns

suncasa.utils.stackplotX.**c_correlate**(*a*, *v*, *returnx=False*)

suncasa.utils.stackplotX.**XCorrMap**(*data*, *refpix=[0, 0]*)

suncasa.utils.stackplotX.**XCorrStackplt**(*z*, *x*, *y*, *doxscale=True*)

get the cross correlation map along y axis :param z: data :param x: x axis :param y: y axis :return:

suncasa.utils.stackplotX.**FitSlit**(*xx*, *yy*, *cutwidth*, *cutang*, *cutlength*, *s=None*, *method='Polyfit'*, *ascending=False*)

suncasa.utils.stackplotX.**MakeSlit**(*pointDF*)

suncasa.utils.stackplotX.**getimprofile**(*data*, *cutslit*, *xrange=None*, *yrange=None*, *get_peak=False*, *verbose=False*)

Get values at a slice

Inputs:

data: input image data. Dimension: (ny, nx) or (ny, nx, nwv). nwv is the number of wavelengths/frequencies cutslit: cutslit generated from CutslitBuilder().cutslitplt xrange: [min(xs), max(xs)], where xs is the x coordinate values of the input image data.

If None (default), assume pixel coordinate values in cutslit

yrange: [min(ys), max(ys)], where ys is the y coordinate values of the input image data.
If None (default), assume pixel coordinate values in cutslit

get_peak: If True, return the peak of all pixels across the slit within the slit width.
If False (default), return the average value.

verbose: If True, print out more details in command line. Default is False

Returns

distance from min(cutslit['xcen']), min(cutslit['ycen'])
'y': value on the cut, the shape is (len(cutslit['xcen'], [nwv])}

Return type
A dictionary of {'x'

suncasa.utils.stackplotX.**smooth**(*x*, *window_len=11*, *window='hanning'*)

smooth the data using a window with requested size.

This method is based on the convolution of a scaled window with the signal. The signal is prepared by introducing reflected copies of the signal (with the window size) in both ends so that transient parts are minimized in the begining and end part of the output signal.

input:

x: the input signal window_len: the dimension of the smoothing window; should be an odd integer window: the type of window from 'flat', 'hanning', 'hamming', 'bartlett', 'blackman'

flat window will produce a moving average smoothing.

output:

the smoothed signal

example:

t=linspace(-2,2,0.1) x=sin(t)+randn(len(t))*0.1 y=smooth(x)

see also:

numpy.hanning, numpy.hamming, numpy.bartlett, numpy.blackman, numpy.convolve scipy.signal.lfilter

NOTE: length(output) != length(input), to correct this: return y[(window_len/2-1):-(window_len/2)] instead of just y.

suncasa.utils.stackplotX.**grid**(*x*, *y*, *z*, *resX=20*, *resY=40*)

Convert 3 column data to matplotlib grid

suncasa.utils.stackplotX.**polyfit**(*x*, *y*, *length*, *deg*)

class suncasa.utils.stackplotX.**LightCurveBuilder**(*stackplt*, *axes*, *scale=1.0*, *color='white'*)

__call__(*event*)

update(*mask=None*)

save(*event*)

delete(*event*)

delete_byindex(*index*)

update_text()

lightcurves_tofile(*outfile=None*, *lightcurves=None*)

lightcurves_fromfile(*infile*, *color=None*)

class suncasa.utils.stackplotX.**SpaceTimeSlitBuilder**(*axes*, *dspec*, *cutlength=80*, *cutsmooth=10.0*, *scale=1.0*, *color='white'*)

__call__(*event*)

FitSlit(*xx*, *yy*, *cutlength*, *method='Polyfit'*, *s=0*, *ascending=True*)

polynomial fit

update(*mask=None*)

save(*event*)

delete(*event*)

delete_byindex(*index*)

update_text()

select_distance_along_a_slice(*ixx*)

select points more accurately by doing it on a distance-flux plot

spacetimeslits_tofile(*outfile=None*, *spacetimeslits=None*)

spacetimeslits_fromfile(*infile*, *color=None*)

class suncasa.utils.stackplotX.**CutslitBuilder**(*axes*, *cutwidth=5.0*, *cutlength=150*, *cutang=0.0*, *cutsmooth=10.0*, *scale=1.0*)

__call__(*event*)

update(*mask=None*)

class suncasa.utils.stackplotX.**Stackplot**(*infile=None*)

property cutslit

property tplt

instrum_meta

aia_lvl1

suncasadb

mapseq

mapseq_diff

mapseq_plot

cutslitbd

stackplt

trange

wavelength

fitsfile

exptime_orig

fov

binpix

dt_data

divider_im

divider_dspec

sCutwdth

sCutang

fig_mapseq

pixscale

get_plot_title(*smap*, *title*)

plot_map(*smap*, *dspec=None*, *diff=False*, *norm=None*, *cmap=None*, *SymLogNorm=False*, *linthresh=0.5*, *returnImAx=False*, *layout_vert=False*, *uni_cm=False*, *draw_limb=False*, *draw_grid=False*, *colortitle=None*, *title=['observatory', 'detector', 'wavelength', 'time']*, *fov=fov*, **args*, ***kwargs*)

make_mapseq(*trange*, *outfile=None*, *fov=None*, *wavelength='171'*, *binpix=1*, *dt_data=1*, *derotate=False*, *tosave=True*, *superpixel=False*, *aia_prep=False*, *mapinterp=False*, *overwrite=False*, *dtype=None*, *normalize=True*)

mapseq_fromfile(*infile*)

mapseq_tofile(*outfile=None*, *mapseq=None*)

mapseq_drot()

mapseq_resample(*binpix=1*)

mapseq_diff_denoise(*log=False*, *vmax=None*, *vmin=None*)

mapseq_mkdiff(*mode='rdiff'*, *dt=36.0*, *medfilt=None*, *gaussfilt=None*, *bfilter=False*, *lowcut=1 / 10 / 60.0*, *highcut=1 / 1 / 60.0*, *window=[None, None]*, *outfile=None*, *tosave=False*, *dtype=None*)

Parameters

- **mode** – accept modes: rdiff, rratio, bdiff, bratio, dtrend, dtrend_diff, dtrend_ratio
- **dt** – time difference in second between frames when [rdiff, rratio, bdiff, bratio] is invoked
- **medfilt** –
- **gaussfilt** –
- **bfilter** – do butter bandpass filter
- **lowcut** – low cutoff frequency in Hz
- **highcut** – high cutoff frequency in Hz
- **outfile** –
- **tosave** –

Returns

plot_mapseq(*mapseq=None*, *hdr=False*, *norm=None*, *vmax=None*, *vmin=None*, *cmap=None*, *diff=False*, *sav_img=False*, *out_dir=None*, *dpi=100*, *anim=False*, *silent=False*, *draw_limb=False*, *draw_grid=False*, *colortitle=None*, *title=['observatory', 'detector', 'wavelength', 'time']*, *fov=[]*, *fps=15*)

Parameters

- **mapseq** –
- **hdr** –
- **vmax** –
- **vmin** –
- **diff** –
- **sav_img** –
- **out_dir** –
- **dpi** –
- **anim** –

Returns

cutslit_fromfile(*infile*, *color=None*, *mask=None*)

cutslit_tofile(*outfile=None*, *cutslit=None*)

make_stackplot(*mapseq*, *frm_range=[]*, *threshold=None*, *gamma=1.0*, *get_peak=False*, *trackslit_diffrot=False*, *negval=False*, *movingcut=[]*)

movingcut: [x,y]. x and y are an array of offset in X and Y direction, respectively. the length of x/y is nframes

stackplt_wrap()

stackplt_tofile(*outfile=None*, *stackplt=None*)

stackplt_fromfile(*infile*, *doplot=False*, ***kwargs*)

plot_stackplot(*mapseq=None*, *fov=None*, *hdr=False*, *norm=None*, *vmax=None*, *vmin=None*, *cmap=None*, *layout_vert=False*, *diff=False*, *uni_cm=True*, *sav_img=False*, *out_dir=None*, *dpi=100*, *anim=False*, *frm_range=[]*, *cutslitplt=None*, *silent=False*, *refresh=True*, *threshold=None*, *gamma=1.0*, *get_peak=False*, *trackslit_diffrot=False*, *negval=False*, *movingcut=[]*)

stackplt_traject_fromfile(*infile*, *frm_range=[]*, *cmap='inferno'*, *norm=None*, *gamma=1.0*)

stackplt_lghtcurv_fromfile(*infile*, *frm_range=[]*, *cmap='inferno'*, *norm=None*, *gamma=1.0*, *log=False*, *axes=None*)

mapseq_info(*mapseq=None*)

classmethod set_fits_dir(*fitsdir*)

suncasa.utils.stputils

Module Contents

Functions

insertchar(source_str, insert_str, pos)	
get_curve_grad(x, y)	get the grad of at data point
findDist(x, y)	
paramspline(x, y, length[, s])	
polyfit(x, y, length, deg[, keepxorder])	
improfile(z, xi, yi[, interp])	Pixel-value cross-section along line segment in an image
map2wcsgrids(snpmap[, cell, antialiased])	**param snpmap**
readsdofile([datadir, wavelength, trange, isexists, ...])	read sdo file from local database
get_map_corner_coord(sunpymap)	

Attributes

__author__
__email__

suncasa.utils.stputils.**__author__ = ['Sijie Yu']**

suncasa.utils.stputils.**__email__ = 'sijie.yu@njit.edu'**

suncasa.utils.stputils.**insertchar**(*source_str*, *insert_str*, *pos*)

suncasa.utils.stputils.**get_curve_grad**(*x*, *y*)

get the grad of at data point :param x: :param y: :return: grad,posang

suncasa.utils.stputils.**findDist**(*x*, *y*)

suncasa.utils.stputils.**paramspline**(*x*, *y*, *length*, *s=0*)

suncasa.utils.stputils.**polyfit**(*x*, *y*, *length*, *deg*, *keepxorder=False*)

suncasa.utils.stputils.**improfile**(*z*, *xi*, *yi*, *interp='cubic'*)

Pixel-value cross-section along line segment in an image :param z: an image array :param xi and yi: equal-length vectors specifying the pixel coordinates of the endpoints of the line segment :param interp: interpolation type to sampling, 'nearest' or 'cubic' :return: the intensity values of pixels along the line

suncasa.utils.stputils.**map2wcsgrids**(*snpmap*, *cell=True*, *antialiased=True*)

Parameters

- **snpmap** –
- **cell** – if True, return the coordinates of the pixel centers. if False, return the coordinates of the pixel boundaries

Returns

suncasa.utils.stputils.**readsdofile**(*datadir=None*, *wavelength=None*, *trange=None*, *isexists=False*, *timtol=1*, *ignoreymdpath=False*, *suffix='image_lev1'*)

read sdo file from local database :param datadir: :param wavelength: :param trange: the timestamp or timerange in Julian days. if is timerange, return a list of files in the timerange :param isexists: check if file exist. if files exist, return file name :param timtol: time difference tolerance in days for considering data as the same timestamp :return:

suncasa.utils.stputils.**get_map_corner_coord**(*sunpymap*)

Submodules

suncasa.casa_compat

casa_compat.py

Provides a uniform interface for importing CASA (Common Astronomy Software Applications) tools across different versions and Python environments. It supports both the monolithic (CASA 4/5/6) and modular (CASA 6+) installations.

This script dynamically imports CASA components, addressing the architectural changes between versions. In monolithic installations, components are available as instantiated objects. In modular installations, components are accessed through *casatools* and *casatasks*.

Function: - get_casa_tools(alias_list): Returns a dictionary of requested CASA tool instances. It accepts a list of tool aliases and handles dynamic import or built-in object access, ensuring compatibility across CASA versions.

Parameters: - alias_list (list of str): Aliases for CASA tools to import. Defaults to a common set of tools.

Returns: - dict: Mapping of tool aliases to their instances or objects.

Usage example:

casa_tools = get_casa_tools([‘tbtool’, ‘mstool’, ‘qatool’, ‘iatool’, ‘rgtool’, ‘msmdtool’, ‘smtool’, ‘metool’]) for alias, instance in casa_tools.items():

print(f”{alias}: {instance}”)

The function uses *importlib* for CASA 6+ and falls back to direct access in earlier versions or interactive sessions.

Module Contents

Functions

check_dependencies()	
import_casatools([alias_list])	Dynamically imports and returns CASA tools specified by their aliases.
import_casatasks(*task_names)	Dynamically imports specified CASA tasks from the casatasks module. This is designed to uniformly import CASA tasks

Attributes

tool_mapping

suncasa.casa_compat.**tool_mapping**

suncasa.casa_compat.**check_dependencies**()

suncasa.casa_compat.**import_casatools**(*alias_list=['tbtool', 'mstool', 'qatool', 'iatool', 'rgtool', 'msmdtool', 'smtool', 'metool']*)

Dynamically imports and returns CASA tools specified by their aliases.

Parameters: alias_list (list of str): Aliases of the CASA tools to be imported and returned.

Returns: dict: A dictionary with keys as tool aliases and values as the imported modules or objects.

suncasa.casa_compat.**import_casatasks**(**task_names*)

Dynamically imports specified CASA tasks from the casatasks module. This is designed to uniformly import CASA tasks for both monolithic and modular CASA installations, addressing the issue where direct task import is not supported in monolithic CASA versions.

Intended for modular CASA 6+ installations where tasks are accessed via casatasks. For monolithic CASA (versions 4/5/6), this function provides a unified interface, though direct imports are handled via the global namespace.

Parameters: - task_names (str): Names of CASA tasks to import.

Returns: - dict: Mapping of task names to their corresponding functions.

Raises: - ImportError: If a task is not found in casatasks.

1.1.2 pmaxfit

Module Contents

Classes

_pmaxfit	pmaxfit ---- Find maximum and do parabolic fit in the sky

Functions

static_var(varname, value)

Attributes

pmaxfit

pmaxfit.**static_var**(*varname*, *value*)

class pmaxfit.**_pmaxfit**

pmaxfit —- Find maximum and do parabolic fit in the sky

PARAMETER SUMMARY imagename Name of the input image box Rectangular region(s) to select in direction plane. See “help par.box” for details. Default is to use the entire direction plane. eg “100, 120, 200, 220, 300, 300, 400, 400” to use two boxes.

OVERVIEW This application finds the pixel with the maximum value in the region, and then uses function findsources to generate a Componentlist with one component.

The method returns a dictionary with fours keys, 'succeeded', 'timestamps', 'imagenames' and 'outputs'. The value of 'outputs' is a dictionary representing a component list reflecting the fit results over multiple channels. Both the 'outputs' dictionaries can be read into a component list tool (default tool is named cl) using the fromrecord() method for easier inspection using tool methods, eg

FITTING OVER MULTIPLE CHANNELS

For fitting over multiple channels, the result of the previous successful fit is used as the estimate for the next channel. The number of gaussians fit cannot be varied on a channel by channel basis. Thus the variation of source structure should be reasonably smooth in frequency to produce reliable fit results.

——— parameter descriptions ———————————————

imagefiles A list of the input images ncpu Number of cpu cores to use box Rectangular region(s) to select in direction plane. See "help par.box" for details. Default is to use the entire direction plane. width Half-width of fit grid [1;42mRETURNS[1;m void

——— examples ———————————————————————

EXAMPLE:

Here is how one might fit two gaussians to multiple channels of a cube using the fit from the previous channel as the initial estimate for the next. It also illustrates how one can specify a region in the associated continuum image as the region to use as the fit for the channel.

begin{verbatim} default pmaxfit imagename = "co_cube.im" # specify region using region from continuum box = "100,120,200,200" pmaxfit()

_info_group_ = 'analysis'

_info_desc_ = 'Find maximum and do parabolic fit in the sky'

__schema

__globals_()

__to_string_(*value*)

__validate_(*doc*, *schema*)

__do_inp_output(*param_prefix*, *description_str*, *formatting_chars*)

__imagefiles_dflt(*glb*)

__imagefiles(*glb*)

__ncpu_dflt(*glb*)

__ncpu(*glb*)

__box_dflt(*glb*)

__box(*glb*)

__width_dflt(*glb*)

__width(*glb*)

__imagefiles_inp()

__ncpu_inp()

__box_inp()

__width_inp()

set_global_defaults()

inp()

tget(*file=None*)

__call__(*imagefiles=None*, *ncpu=None*, *box=None*, *width=None*)

pmaxfit.**pmaxfit**

1.1.3 calibeovsa

Module Contents

Classes

_calibeovsa	calibeovsa ---- Calibrating EOVSA one or more measurement sets using calibration products in the SQL database.

Functions

static_var(varname, value)

Attributes

calibeovsa

calibeovsa.**static_var**(*varname*, *value*)

class calibeovsa.**_calibeovsa**

calibeovsa —- Calibrating EOVSA one or more measurement sets using calibration products in the SQL database.

Calibrating EOVSA one or more measurement sets using calibration products in the SQL database. This task currently only works on pipeline.

——— parameter descriptions ———————————————

vis input EOVSA (uncalibrated) measurement set(s). caltype Types of calibrations to perform caltbdir Directory to place calibration tables. interp Temporal interpolation for phacal table(s) (nearest or linear) docalib If False, only create the calibration tables but do not perform applycal. doflag If true then perform flagging. flagant

Antennas to be flagged. Follow CASA syntax of “antenna”. doimage If True, produce a quicklook image after calibration (sunpy must be installed). imagedir directory to place output images. Default current directory. antenna antenna/baselines to be used for imaging. Follow CASA syntax of “antenna”. timerange Timerange to be imaged. Follow CASA syntax of “timerange”. Default is the entire duration of the ms. spw spectral windows to be imaged. Follow CASA syntax of “spw”. stokes stokes to be imaged. Follow CASA syntax of “stokes”. dosplit If True, plit the corrected data column as output visibility file. outputvis Name of output visibility file. Default is the name of the first vis file ended with “.corrected.ms”. doconcat If True, and if more than one visibility dataset provided, concatenate all into one visibility. concatvis Name of output visibility file. Default is the name of the first + last vis file ended with “.corrected.ms”. keep_orig_ms Keep the original seperated ms datasets after split?

——— examples ———————————————————————

Calibrating EOVSA one or more measurement sets using calibration products in the SQL database.

Detailed Keyword arguments:

vis – Name of input EOVSA measurement set dataset(s) default: none. Must be supplied example: vis = ‘IDB20160524000518.ms’ example: vis = [‘IDB20160524000518.ms’,’IDB20160524000528.ms’]

caltype – list. Type of calibrations to be applied. ‘refpha’: reference phase calibration ‘refamp’: reference amplitude calibration (not used anymore) ‘phacal’: daily phase calibration ‘fluxcal’: flux calibration based on total-power measurements default value: [‘refpha’,’phacal’] *** note fluxcal is already implemented in udb_corr when doing importeovsa, should not be used anymore **** *** pipeline only uses [‘refpha’,’phacal’]

caltbdir – string. Place to hold calibration tables. Default is current directory. Pipeline should use /data1/eovsa/caltable

interp – string. How interpolation is done for phacal? ‘nearest’ or ‘linear’

docalib – boolean. Default True. If False, only create the calibration tables but do not perform applycal

doflag – boolean. Default True. Peforming flags?

flagant – string. Follow CASA antenna selection syntax. Default ‘13~15’.

doimage – boolean. Default False. If true, make a quicklook image using the specified time range and specified spw range

imagedir – string. Directory to place the output image.

antenna – string. Default ‘0~12’. Antenna/baselines to be used for imaging. Follow CASA antenna selection syntax.

timerange – string. Default ‘’ (the whole duration of the visibility data). Follow CASA timerange syntax. e.g., ‘2017/07/11/20:16:00~2017/07/11/20:17:00’

spw – string. Default ‘1~3’. Follow CASA spw selection syntax.

stokes – string. Which stokes for the quicklook image. CASA syntax. Default ‘XX’

dosplit – boolean. Split the corrected data column?

outputvis – string. Output visibility file after split

doconcat – boolean. If more than one visibility dataset provided, concatenate all into one or make separate outputs if True

concatvis – string. Output visibility file after concatenation

keep_orig_ms – boolean. Default True. Inherited from suncasa.eovsa.concateovsa. Keep the original seperated ms datasets after concatenation?

`_info_group_ = 'Calibration'`

_info_desc_ = 'Calibrating EOVSA one or more measurement sets using calibration products in the SQL database.'

__schema

__globals_()

__to_string_(*value*)

__validate_(*doc*, *schema*)

__do_inp_output(*param_prefix*, *description_str*, *formatting_chars*)

__doimage_dflt(*glb*)

__doimage(*glb*)

__vis_dflt(*glb*)

__vis(*glb*)

__caltbdir_dflt(*glb*)

__caltbdir(*glb*)

__docalib_dflt(*glb*)

__docalib(*glb*)

__interp_dflt(*glb*)

__interp(*glb*)

__caltype_dflt(*glb*)

__caltype(*glb*)

__doflag_dflt(*glb*)

__doflag(*glb*)

__dosplit_dflt(*glb*)

__dosplit(*glb*)

__doconcat_dflt(*glb*)

__doconcat(*glb*)

__antenna_dflt(*glb*)

__stokes_dflt(*glb*)

__flagant_dflt(*glb*)

__concatvis_dflt(*glb*)

__outputvis_dflt(*glb*)

__keep_orig_ms_dflt(*glb*)

__imagedir_dflt(*glb*)

__spw_dflt(*glb*)

__timerange_dflt(*glb*)

__flagant(*glb*)

__imagedir(*glb*)

__antenna(*glb*)

__timerange(*glb*)

__spw(*glb*)

__stokes(*glb*)

__outputvis(*glb*)

__concatvis(*glb*)

__keep_orig_ms(*glb*)

__vis_inp()

__caltype_inp()

__caltbdir_inp()

__interp_inp()

__docalib_inp()

__doflag_inp()

__flagant_inp()

__doimage_inp()

__imagedir_inp()

__antenna_inp()

__timerange_inp()

__spw_inp()

__stokes_inp()

__dosplit_inp()

__outputvis_inp()

__doconcat_inp()

__concatvis_inp()

__keep_orig_ms_inp()

set_global_defaults()

inp()

tget(*savefile=None*)

tput(*outfile=None*)

__call__(*vis=None*, *caltype=None*, *caltbdir=None*, *interp=None*, *docalib=None*, *doflag=None*, *flagant=None*, *doimage=None*, *imagedir=None*, *antenna=None*, *timerange=None*, *spw=None*, *stokes=None*, *dosplit=None*, *outputvis=None*, *doconcat=None*, *concatvis=None*, *keep_orig_ms=None*)

calibeovsa.**calibeovsa**

1.1.4 concateovsa

Module Contents

Classes

_concateovsa	concateovsa ---- Concatenate several EOVSA visibility data sets.

Functions

static_var(varname, value)

Attributes

concateovsa

concateovsa.**static_var**(*varname*, *value*)

class concateovsa.**_concateovsa**

concateovsa —- Concatenate several EOVSA visibility data sets.

This is a EOVSA version of CASA concat task.

The list of data sets given in the vis argument are chronologically concatenated into an output data set in concatvis, i.e. the data sets in vis are first ordered by the time of their earliest integration and then concatenated.

If there are fields whose direction agrees within the direction tolerance (parameter dirtol), the actual direction in the resulting, merged output field will be the one from the chronologically first input MS.

If concatvis already exists (e.g., it is the same as the first input data set), then the other input data sets will be appended to the concatvis data set. There is no limit to the number of input data sets.

If none of the input data sets have any scratch columns (model and corrected columns), none are created in the concatvis. Otherwise these columns are created on output and initialized to their default value (1 in model column, data in corrected column) for those data with no input columns.

Spectral windows for each data set with the same chanelization, and within a specified frequency tolerance of another data set will be combined into one spectral window.

A field position in one data set that is within a specified direction tolerance of another field position in any other data set will be combined into one field. The field names need not be the same—only their position is used.

Each appended dataset is assigned a new observation id (provided the entries in the observation table are indeed different).

Keyword arguments: vis – Name of input visibility files to be combined default: none; example: vis = ['src2.ms','ngc5921.ms','ngc315.ms'] concatvis – Name of visibility file that will contain the concatenated data note: if this file exits on disk then the input files are added to this file. Otherwise the new file contains the concatenated data. Be careful here when concatenating to an existing file. default: none; example: concatvis='src2.ms' example: concatvis='outvis.ms'

datacolumn – Which data column to use for processing (case-insensitive). default: 'corrected'; example: datacolumn='data' options: 'data', 'corrected'.

freqtol – Frequency shift tolerance for considering data to be in the same spwid. The number of channels must also be the same. default: '' == 1 Hz example: freqtol='10MHz' will not combine spwid unless they are within 10 MHz. Note: This option is useful to combine spectral windows with very slight frequency differences caused by Doppler tracking, for example.

dirtol – Direction shift tolerance for considering data as the same field default: '' == 1 mas (milliarcsec) example: dirtol='1arcsec' will not combine data for a field unless their phase center differ by less than 1 arcsec. If the field names are different in the input data sets, the name in the output data set will be the first relevant data set in the list.

respectname – If true, fields with a different name are not merged even if their direction agrees (within dirtol) default: False

timesort – If true, the output visibility table will be sorted in time. default: false. Data in order as read in. example: timesort=true Note: There is no constraint on data that is simultaneously observed for more than one field; for example multi-source correlation of VLBA data.

copypointing – Make a proper copy of the POINTING subtable (can be time consuming). If False, the result is an empty POINTING table. default: True

visweightscale – The weights of the individual MSs will be scaled in the concatenated output MS by the factors in this list. SIGMA will be scaled by 1/sqrt(factor). Useful for handling heterogeneous arrays. Use plotms to inspect the "Wt" column as a reference for determining the scaling factors. See the cookbook for more details. example: [1.,3.,3.] - scale the weights of the second and third MS by a factor 3 and the SIGMA column of these MS by a factor 1/sqrt(3). default: [] (empty list) - no scaling

forcesingleephemfield – By default, concat will only merge two ephemeris fields if the first ephemeris covers the time range of the second. Otherwise, two separate fields with separate ephemerides are placed in the output MS. In order to override this behaviour and make concat merge the non-overlapping or only partially overlapping input ephemerides, the name or id of the field in question needs to be placed into the list in parameter 'forcesingleephemfield'. example: ['Neptune'] - will make sure that there is only one joint ephemeris for field Neptune in the output MS default: '' - standard treatment of all ephemeris fields

——— parameter descriptions ———————————————

vis Name of input visibility files to be concatenated concatvis Name of output visibility file datacolumn Which data column(s) to concatenate keep_orig_ms If false, input vis files will be removed cols2rm Columns in concatvis to be removed to slim the concatvis freqtol Frequency shift tolerance for considering data as the same spwid dirtol Direction shift tolerance for considering data as the same field respectname If true, fields with a

different name are not merged even if their direction agrees timesort If true, sort by TIME in ascending order copypointing Copy all rows of the POINTING table. visweightscale List of the weight scaling factors to be applied to the individual MSs forcesingleephemfield make sure that there is only one joint ephemeris for every field in this list

——— examples ———————————————————————

concateovsa(vis=['UDB20180102161402.ms','UDB20180102173518.ms'], concatvis='UDB20180102_allday.ms') will concatenate 'UDB20180102161402.ms' and 'UDB20180102173518.ms' into 'UDB20180102_allday.ms'

_info_group_ = 'utility, manipulation'

_info_desc_ = 'Concatenate several EOVSA visibility data sets.'

__schema

__globals_()

__to_string_(*value*)

__validate_(*doc*, *schema*)

__do_inp_output(*param_prefix*, *description_str*, *formatting_chars*)

__forcesingleephemfield_dflt(*glb*)

__forcesingleephemfield(*glb*)

__vis_dflt(*glb*)

__vis(*glb*)

__dirtol_dflt(*glb*)

__dirtol(*glb*)

__timesort_dflt(*glb*)

__timesort(*glb*)

__datacolumn_dflt(*glb*)

__datacolumn(*glb*)

__respectname_dflt(*glb*)

__respectname(*glb*)

__keep_orig_ms_dflt(*glb*)

__keep_orig_ms(*glb*)

__visweightscale_dflt(*glb*)

__visweightscale(*glb*)

__concatvis_dflt(*glb*)

__concatvis(*glb*)

__copypointing_dflt(*glb*)

__copypointing(*glb*)

__cols2rm_dflt(*glb*)

__cols2rm(*glb*)

__freqtol_dflt(*glb*)

__freqtol(*glb*)

__vis_inp()

__concatvis_inp()

__datacolumn_inp()

__keep_orig_ms_inp()

__cols2rm_inp()

__freqtol_inp()

__dirtol_inp()

__respectname_inp()

__timesort_inp()

__copypointing_inp()

__visweightscale_inp()

__forcesingleephemfield_inp()

set_global_defaults()

inp()

tget(*file=None*)

__call__(*vis=None*, *concatvis=None*, *datacolumn=None*, *keep_orig_ms=None*, *cols2rm=None*, *freqtol=None*, *dirtol=None*, *respectname=None*, *timesort=None*, *copypointing=None*, *visweightscale=None*, *forcesingleephemfield=None*)

concateovsa.**concateovsa**

1.1.5 pimfit

Module Contents

Classes

_pimfit	pimfit ---- Fit one or more elliptical Gaussian components on an image region(s)

Functions

static_var(varname, value)

Attributes

pimfit

pimfit.**static_var**(*varname*, *value*)

class pimfit.**_pimfit**

pimfit —- Fit one or more elliptical Gaussian components on an image region(s)

——— parameter descriptions ————————————————

imagefiles A list of the input images ncpu Number of cpu cores to use doreg True if use vla_prep to register the image ephemfile emphemeris file generated from vla_prep.read_horizons() timestamps A list of timestamps of the input images msinfofile time-dependent phase center information generated from vla_prep.read_msinfo() box Rectangular region(s) to select in direction plane. See "help par.box" for details. Default is to use the entire direction plane. region Region selection. See "help par.region" for details. Default is to use the full image. chans Channels to use. See "help par.chans" for details. Default is to use all channels. stokes Stokes planes to use. See "help par.stokes" for details. Default is to use first Stokes plane. mask Mask to use. See help par.mask. Default is none. includepix Range of pixel values to include for fitting. excludepix Range of pixel values to exclude for fitting. residual Name of output residual image. model Name of output model image. estimates Name of file containing initial estimates of component parameters. logfile Name of file to write fit results. append If logfile exists, append to it if True or overwrite it if False newestimates File to write fit results which can be used as initial estimates for next run. complist Name of output component list table. overwrite Overwrite component list table if it exists? dooff Also fit a zero level offset? Default is False offset Initial estimate of zero-level offset. Only used if doff is True. Default is 0.0 fixoffset Keep the zero level offset fixed during fit? Default is False stretch Stretch the mask if necessary and possible? See help par.stretch rms RMS to use in calculation of uncertainties. Numeric or valid quantity (record or string). If numeric, it is given units of the input image. If quantity, units must conform to image units. If not positive, the rms of the residual image, in the region of the fit, is used. noisefwhm Noise correlation beam FWHM. If numeric value, interpreted as pixel widths. If quantity (dictionary, string), it must have angular units. summary File name to which to write table of fit parameters. [1;42mRETURNS[1;m void

——— examples ————————————————————————

PARAMETER SUMMARY imagename Name of the input image box Rectangular region(s) to select in direction plane. See "help par.box" for details. Default is to use the entire direction plane. eg "100, 120, 200, 220, 300, 300, 400, 400" to use two boxes. region Region selection. See "help par.region" for details. Default is to use the full image. chans Channels to use. See "help par.chans" for details. Default is to use all channels. stokes Stokes planes to use. See "help par.stokes" for details. Default is to use first Stokes plane. mask Mask to use. See help par.mask. Default is none. includepix Range of pixel values to include for fitting. Array of two numeric values assumed to have same units as image pixel values. Only one of includepix or excludepix can be specified. excludepix Range of pixel values to exclude for fitting. Array of two numeric values assumed to have same units as image pixel values. Only one of includepix or excludepix can be specified. residual Name of the residual image to write. model Name of the model image to write. estimates Name of file containing initial estimates of component parameters (see below for formatting details). logfile Name of file to write fit results. append If logfile exists, append to it (True) or overwrite it (False). newestimates File to write fit results which can be used as initial estimates for next run. complist Name of output component list table. overwrite Overwrite component

list table if it exists? dooff Simultaneously fit a zero-level offset? offset Initial estimate for the zero-level offset. Only used if dooff is True. fixoffset Hold zero-level offset constant during fit? Only used if dooff is True. stretch Stretch the input mask if necessary and possible. Only used if a mask is specified. See help par.stretch. rms RMS to use in calculation of various uncertainties, assumed to have units of the input image. If not positve, the rms of the residual image is used. noisefwhm Noise correlation beam FWHM. If numeric value, interpreted as pixel widths. If quantity (dictionary, string), it must have angular units. summary File name to which to write table of fit parameters.

OVERVIEW This application is used to fit one or more two dimensional gaussians to sources in an image as well as an optional zero-level offset. Fitting is limited to a single polarization but can be performed over several contiguous spectral channels. If the image has a clean beam, the report and returned dictionary will contain both the convolved and the deconvolved fit results.

When dooff is False, the method returns a dictionary with three keys, 'converged', 'results', and 'deconvolved'. The value of 'converged' is a boolean array which indicates if the fit converged on a channel by channel basis. The value of 'results' is a dictionary representing a component list reflecting the fit results. In the case of an image containing beam information, the sizes and position angles in the 'results' dictionary are those of the source(s) convolved with the restoring beam, while the same parameters in the 'deconvolved' dictionary represent the source sizes deconvolved from the beam. In the case where the image does not contain a beam, 'deconvolved' will be absent. Both the 'results' and 'deconvolved' dictionaries can be read into a component list tool (default tool is named cl) using the fromrecord() method for easier inspection using tool methods, eg

cl.fromrecord(res['results'])

although this currently only works if the flux density units are conformant with Jy.

There are also values in each component subdictionary not used by cl.fromrecord() but meant to supply additional information. There is a 'peak' subdictionary for each component that provides the peak intensity of the component. It is present for both 'results' and 'deconvolved' components. There is also a 'sum' subdictionary for each component indicated the simple sum of pixel values in the the original image enclosed by the fitted ellipse. There is a 'channel' entry in the 'spectrum' subdictionary which provides the zero-based channel number in the input image for which the solution applies. In addtion, if the image has a beam(s), then there will be a 'beam' subdictionary associated with each component in both the 'results' and 'deconvolved' dictionaries. This subdictionary will have three keys: 'beamarcsec' will be a subdictionary giving the beam dimensions in arcsec, 'beampixels' will have the value of the beam area expressed in pixels, and 'beamster' will have the value of the beam area epressed in steradians. Also, if the image has a beam(s), in the component level dictionaries will be an 'ispoint' entry with an associated boolean value describing if the component is consistent with a point source.

If dooff is True, in addtion to the specified number of gaussians, a zero-level offset will also be fit. The initial estimate for this offset is specified using the offset parameter. Units are assumed to be the same as the image brightness units. The zero level offset can be held constant during the fit by specifying fixoffset=True. In the case of dooff=True, the returned dictionary contains two additional keys, 'zerooff' and 'zeroofferr', which are both dictionaries containing 'unit' and 'value' keys. The values associated with the 'value' keys are arrays containing the the fitted zero level offset value and its error, respectively, for each channel. In cases where the fit did not converge, these values are set to NaN. The value associated with 'unit' is just the i`mage brightness unit.

The region can either be specified by a box(es) or a region. Ranges of pixel values can be included or excluded from the fit. If specified using the box parameter, multiple boxes can be given using the format box="blcx1, blcy1, trcx1, trcy1, blcx2, blcy2, trcx2, trcy2, ... , blcxN, blcyN, trcxN, trcyN" where N is the number of boxes. In this case, the union of the specified boxes will be used.

If specified, the residual and/or model images for successful fits will be written.

If an estimates file is not specified, an attempt is made to estimate initial parameters and fit a single Gaussian. If a multiple Gaussian fit is desired, the user must specify initial estimates via a text file (see below for details).

The user has the option of writing the result of the fit to a log file, and has the option of either appending to or overwriting an existing file.

The user has the option of writing the (convolved) parameters of a successful fit to a file which can be fed back to fitcomponents() as the estimates file for a subsequent run.

The user has the option of writing the fit results in tabular format to a file whose name is specified using the summary parameter.

If specified and positive, the value of rms is used to calculate the parameter uncertainties, otherwise, the rms in the selected region in the relevant channel is used for these calculations.

The noisefwhm parameter represents the noise-correlation beam FWHM. If specified as a quantity, it should have angular units. If specified as a numerical value, it is set equal to that number of pixels. If specified and greater than or equal to the pixel size, it is used to calculate parameter uncertainties using the correlated noise equations (see below). If it is specified but less than a pixel width, the the uncorrelated noise equations (see below) are used to compute the parameter uncertainties. If it is not specified and the image has a restoring beam(s), the the correlated noise equations are used to compute parameter uncertainties using the geometric mean of the relevant beam major and minor axes as the noise-correlation beam FWHM. If noisefwhm is not specified and the image does not have a restoring beam, then the uncorrelated noise equations are used to compute the parameter uncertainties.

SUPPORTED UNITS

Currently only images with brightness units conformant with Jy/beam, Jy.km/s/beam, and K are fully supported for fitting. If your image has some other base brightness unit, that unit will be assumed to be equivalent to Jy/pixel and results will be calculated accordingly. In particular, the flux density (reported as Integrated Flux in the logger and associated with the "flux" key in the returned component subdictionary(ies)) for such a case represents the sum of pixel values.

Note also that converting the returned results subdictionary to a component list via cl.fromrecord() currently only works properly if the flux density units in the results dictionary are conformant with Jy. If you need to be able to run cl.fromrecord() on the resulting dictionary you can first modify the flux density units by hand to be (some prefix)Jy and then run cl.fromrecord() on that dictionary, bearing in mind your unit conversion.

If the input image has units of K, the flux density of components will be reported in units of [prefix]K*rad*rad, where prefix is an SI prefix used so that the numerical value is between 1 and 1000. To convert to units of K*beam, determine the area of the appropriate beam, which is given by pi/(4*ln(2))*bmaj*bmin, where bmaj and bmin are the major and minor axes of the beam, and convert to steradians (=rad*rad). This value is included in the beam portion of the component subdictionary (key 'beamster'). Then divide the numerical value of the logged flux density by the beam area in steradians. So, for example

begin{verbatim} # run on an image with K brightness units res = imfit(...) # get the I flux density in K*beam of component 0 comp = res['results']['component0'] flux_density_kbeam = comp['flux']['value'][0]/comp['beam']['beamster'] end{verbatim}

FITTING OVER MULTIPLE CHANNELS

For fitting over multiple channels, the result of the previous successful fit is used as the estimate for the next channel. The number of gaussians fit cannot be varied on a channel by channel basis. Thus the variation of source structure should be reasonably smooth in frequency to produce reliable fit results.

MASK SPECIFICATION

Mask specification can be done using an LEL expression. For example

mask = '"myimage">5' will use only pixels with values greater than 5.

INCLUDING AND EXCLUDING PIXELS

Pixels can be included or excluded from the fit based on their values using these parameters. Note that specifying both is not permitted and will cause an error. If specified, both take an array of two numeric values.

ESTIMATES

Initial estimates of fit parameters may be specified via an estimates text file. Each line of this file should contain a set of parameters for a single gaussian. Optionally, some of these parameters can be fixed during the fit. The format of each line is

peak intensity, peak x-pixel value, peak y-pixel value, major axis, minor axis, position angle, fixed

The fixed parameter is optional. The peak intensity is assumed to be in the same units as the image pixel values (eg Jy/beam). The peak coordinates are specified in pixel coordinates. The major and minor axes and the position angle are the convolved parameters if the image has been convolved with a clean beam and are specified as quantities. The fixed parameter is optional and is a string. It may contain any combination of the following characters 'f' (peak intensity), 'x' (peak x position), 'y' (peak y position), 'a' (major axis), 'b' (minor axis), 'p' (position angle).

In addition, lines in the file starting with a # are considered comments.

An example of such a file is:

begin{verbatim} # peak intensity must be in map units 120, 150, 110, 23.5arcsec, 18.9arcsec, 120deg 90, 60, 200, 46arcsec, 23arcsec, 140deg, fxp end{verbatim}

This is a file which specifies that two gaussians are to be simultaneously fit, and for the second gaussian the specified peak intensity, x position, and position angle are to be held fixed during the fit.

ERROR ESTIMATES

Error estimates are based on the work of Condon 1997, PASP, 109, 166. Key assumptions made are: * The given model (elliptical Gaussian, or elliptical Gaussian plus constant offset) is an adequate representation of the data * An accurate estimate of the pixel noise is provided or can be derived (see above). For the case of correlated noise (e.g., a CLEAN map), the fit region should contain many "beams" or an independent value of rms should be provided. * The signal-to-noise ratio (SNR) or the Gaussian component is large. This is necessary because a Taylor series is used to linearize the problem. Condon (1997) states that the fractional bias in the fitted amplitude due to this assumption is of order 1/(S*S), where S is the overall SNR of the Gaussian with respect to the given data set (defined more precisely below). For a 5 sigma "detection" of the Gaussian, this is a 4% effect. * All (or practically all) of the flux in the component being fit falls within the selected region. If a constant offset term is simultaneously fit and not fixed, the region of interest should be even larger. The derivations of the expressions summarized in this note assume an effectively infinite region.

Two sets of equations are used to calculate the parameter uncertainties, based on if the noise is correlated or uncorrelated. The rules governing which set of equations are used have been described above in the description of the noisefwhm parameter.

In the case of uncorrelated noise, the equations used are

f(A) = f(I) = f(M) = f(m) = k*s(x)/M = k*s(y)/m = (s(p)/sqrt(2))*((M*M - m*m)/(M*m)) = sqrt(2)/S

where s(z) is the uncertainty associated with parameter z, f(z) = s(z)/abs(z) is the fractional uncertainty associated with parameter z, A is the peak intensity, I is the flux density, M and m are the FWHM major and minor axes, p is the position angle of the component, and k = sqrt(8*ln(2)). s(x) and s(y) are the direction uncertainties of the component measured along the major and minor axes; the resulting uncertainties measured along the principle axes of the image direction coordinate are calculated by propagation of errors using the 2D rotation matrix which enacts the rotation through the position angle plus 90 degrees. S is the overall signal to noise ratio of the component, which, for the uncorrelated noise case is given by

S = (A/(k*h*r))*sqrt(pi*M*m)

where h is the pixel width of the direction coordinate and r is the rms noise (see the discussion above for the rules governing how the value of r is determined).

For the correlated noise case, the same equations are used to determine the uncertainties as in the uncorrelated noise case, except for the uncertainty in I (see below). However, S is given by

S = (A/(2*r*N)) * sqrt(M*m) * (1 + ((N*N/(M*M)))**(a/2)) * (1 + ((N*N/(m*m)))**(b/2))

where N is the noise-correlation beam FWHM (see discussion of the noisefwhm parameter for rules governing how this value is determined). "**" indicates exponentiation and a and b depend on which uncertainty is being calculated. For sigma(A), a = b = 3/2. For M and x, a = 5/2 and b = 1/2. For m, y, and p, a = 1/2 and b = 5/2. f(I) is calculated in the correlated noise case according to

f(I) = sqrt(f(A)*f(A) + (N*N/(M*m))*(f(M*f(M) + f(m)*f(m))))

Note well the following caveats: * Fixing Gaussian component parameters will tend to cause the parameter uncertainties reported for free parameters to be overestimated. * Fitting a zero level offset that is not fixed will tend to cause the reported parameter uncertainties to be slightly underestimated. * The parameter uncertainties will be inaccurate at low SNR (a ~10% for SNR = 3). * If the fitted region is not considerably larger than the largest component that is fit, parameter uncertainties may be mis-estimated. * An accurate rms noise measurement, r, for the region in question must be supplied. Alternatively, a sufficiently large signal-free region must be present in the selected region (at least about 25 noise beams in area) to auto-derive such an estimate. * If the image noise is not statistically independent from pixel to pixel, a reasonably accurate noise correlation scale, N, must be provided. If the noise correlation function is not approximately Gaussian, the correlation length can be estimated using

N = sqrt(2*ln(2)/pi)* double-integral(dx dy C(x,y))/sqrt(double-integral(dx dy C(x, y) * C(x,y)))

where C(x,y) is the associated noise-smoothing function * If fitted model components have significan spatial overlap, the parameter uncertainties are likely to be mis-estimated (i.e., correlations between the parameters of separate components are not accounted for). * If the image being analyzed is an interferometric image with poor uv sampling, the parameter uncertainties may be significantly underestimated.

The deconvolved size and position angle errors are computed by taking the maximum of the absolute values of the differences of the best fit deconvolved value of the given parameter and the deconvolved size of the eight possible combinations of (FWHM major axis +/- major axis error), (FWHM minor axis +/- minor axis error), and (position andle +/- position angle error). If the source cannot be deconvolved from the beam (if the best fit convolved source size cannot be deconvolved from the beam), upper limits on the deconvolved source size are sometimes reported. These limits simply come from the maximum major and minor axes of the deconvolved gaussians taken from trying all eight of the aforementioned combinations. In the case none of these combinations produces a deconvolved size, no upper limit is reported.

EXAMPLE:

Here is how one might fit two gaussians to multiple channels of a cube using the fit from the previous channel as the initial estimate for the next. It also illustrates how one can specify a region in the associated continuum image as the region to use as the fit for the channel.

begin{verbatim} default imfit imagename = "co_cube.im" # specify region using region from continuum region = "continuum.im:source.rgn" chans = "2~20" # only use pixels with positive values in the fit excludepix = [-1e10,0] # estimates file contains initial parameters for two Gaussians in channel 2 estimates = "initial_estimates.txt" logfile = "co_fit.log" # append results to the log file for all the channels append = "True" imfit()

_info_group_ = 'analysis'

_info_desc_ = 'Fit one or more elliptical Gaussian components on an image region(s)'

__schema

__globals_()

__to_string_(*value*)

__validate_(*doc*, *schema*)

__do_inp_output(*param_prefix*, *description_str*, *formatting_chars*)

__noisefwhm_dflt(*glb*)

__noisefwhm(*glb*)

__includepix_dflt(*glb*)

__includepix(*glb*)

__rms_dflt(*glb*)

__rms(*glb*)

__estimates_dflt(*glb*)

__estimates(*glb*)

__summary_dflt(*glb*)

__summary(*glb*)

__newestimates_dflt(*glb*)

__newestimates(*glb*)

__model_dflt(*glb*)

__model(*glb*)

__logfile_dflt(*glb*)

__logfile(*glb*)

__dooff_dflt(*glb*)

__dooff(*glb*)

__mask_dflt(*glb*)

__mask(*glb*)

__residual_dflt(*glb*)

__residual(*glb*)

__ncpu_dflt(*glb*)

__ncpu(*glb*)

__stokes_dflt(*glb*)

__stokes(*glb*)

__region_dflt(*glb*)

__region(*glb*)

__excludepix_dflt(*glb*)

__excludepix(*glb*)

__chans_dflt(*glb*)

__chans(*glb*)

__imagefiles_dflt(*glb*)

__imagefiles(*glb*)

__complist_dflt(*glb*)

__complist(*glb*)

__box_dflt(*glb*)

__box(*glb*)

__doreg_dflt(*glb*)

__doreg(*glb*)

__timestamps_dflt(*glb*)

__stretch_dflt(*glb*)

__offset_dflt(*glb*)

__msinfofile_dflt(*glb*)

__ephemfile_dflt(*glb*)

__overwrite_dflt(*glb*)

__fixoffset_dflt(*glb*)

__append_dflt(*glb*)

__ephemfile(*glb*)

__timestamps(*glb*)

__msinfofile(*glb*)

__append(*glb*)

__overwrite(*glb*)

__offset(*glb*)

__fixoffset(*glb*)

__stretch(*glb*)

__imagefiles_inp()

__ncpu_inp()

__doreg_inp()

__ephemfile_inp()

__timestamps_inp()

__msinfofile_inp()

__box_inp()

__region_inp()

__chans_inp()

__stokes_inp()

__mask_inp()

__includepix_inp()

__excludepix_inp()

__residual_inp()

__model_inp()

__estimates_inp()

__logfile_inp()

__append_inp()

__newestimates_inp()

__complist_inp()

__overwrite_inp()

__dooff_inp()

__offset_inp()

__fixoffset_inp()

__stretch_inp()

__rms_inp()

__noisefwhm_inp()

__summary_inp()

set_global_defaults()

inp()

tget(*file=None*)

__call__(*imagefiles=None*, *ncpu=None*, *doreg=None*, *ephemfile=None*, *timestamps=None*, *msinfofile=None*, *box=None*, *region=None*, *chans=None*, *stokes=None*, *mask=None*, *includepix=None*, *excludepix=None*, *residual=None*, *model=None*, *estimates=None*, *logfile=None*, *append=None*, *newestimates=None*, *complist=None*, *overwrite=None*, *dooff=None*, *offset=None*, *fixoffset=None*, *stretch=None*, *rms=None*, *noisefwhm=None*, *summary=None*)

pimfit.**pimfit**

1.1.6 ptclean

Module Contents

Classes

_ptclean	ptclean ---- Parallelized tclean in consecutive time steps

Functions

static_var(varname, value)

Attributes

ptclean

ptclean.**static_var**(*varname*, *value*)

class ptclean.**_ptclean**

ptclean —- Parallelized tclean in consecutive time steps

Parallelized clean in consecutive time steps. Packed over CASA tclean.

——— parameter descriptions ———————————————

vis Name(s) of input visibility file(s)

default: none; example: vis='ngc5921.ms'

vis=['ngc5921a.ms','ngc5921b.ms']; multiple MSes

imageprefix Prefix of output image names (usually useful in defining the output path) imagesuffix Suffix of output image names (usually useful in specifyting the image type, version, etc.) ncpu Number of cpu cores to use twidth Number of time pixels to average doreg True if use vla_prep to register the image usephacenter True if use the phacenter information from the measurement set (e.g., VLA); False to assume the phase center is at the solar disk center (EOVSA) reftime Reference time of the J2000 coordinates associated with the ephemeris target. e.g., "2012/03/03/12:00". This is used for helioimage2fits.py to find the solar x y offset in order to register the image. If not set, use the actual timerange of the image (default) toTb True if convert to brightness temperature sclfactor scale the brightness temperature up by its value subregion The name of a CASA region string

The name of a CASA image or region file or region string. Only locations within the region will output to the fits file. If regions specified fall completely outside of the image, ptclean will throw an error.

Manual mask options/examples :

subregion='box[[224pix,224pix],[288pix,288pix]]' : A CASA region string.

docompress True if compress the output fits files overwrite True if overwrite the image selectdata Enable data selection parameters. field to image or mosaic. Use field id(s) or name(s).

['go listobs' to obtain the list id's or names]

default: ''= all fields
If field string is a non-negative integer, it is assumed to be a field index otherwise, it is assumed to be a

field name

field='0~2'; field ids 0,1,2 field='0,4,5~7'; field ids 0,4,5,6,7 field='3C286,3C295'; field named 3C286 and 3C295 field = '3,4C*'; field id 3, all names starting with 4C For multiple MS input, a list of field strings can be used: field = ['0~2','0~4']; field ids 0-2 for the first MS and 0-4

for the second

field = '0~2'; field ids 0-2 for all input MSes

spw l window/channels

NOTE: channels de-selected here will contain all zeros if
selected by the parameter mode subparameters.

default: ''=all spectral windows and channels
spw='0~2,4'; spectral windows 0,1,2,4 (all channels) spw='0:5~61'; spw 0, channels 5 to 61 spw='<2'; spectral windows less than 2 (i.e. 0,1) spw='0,10,3:3~45'; spw 0,10 all channels, spw 3,

channels 3 to 45.

spw='0~2:2~6'; spw 0,1,2 with channels 2 through 6 in each. For multiple MS input, a list of spw strings can be used: spw=['0','0~3']; spw ids 0 for the first MS and 0-3 for the second spw='0~3' spw ids 0-3 for all input MS spw='3:10~20;50~60' for multiple channel ranges within spw id 3 spw='3:10~20;50~60,4:0~30' for different channel ranges for spw ids 3 and 4 spw='0:0~10,1:20~30,2:1;2;3'; spw 0, channels 0-10,

spw 1, channels 20-30, and spw 2, channels, 1,2 and 3

spw='1~4;6:15~48' for channels 15 through 48 for spw ids 1,2,3,4 and 6

timerange Range of time to select from data

default: '' (all); examples, timerange = 'YYYY/MM/DD/hh:mm:ss~YYYY/MM/DD/hh:mm:ss' Note: if YYYY/MM/DD is missing date defaults to first

day in data set

timerange='09:14:0~09:54:0' picks 40 min on first day timerange='25:00:00~27:30:00' picks 1 hr to 3 hr

30min on NEXT day

timerange='09:44:00' pick data within one integration
of time

timerange='> 10:24:00' data after this time For multiple MS input, a list of timerange strings can be used: timerange=['09:14:0~09:54:0','> 10:24:00'] timerange='09:14:0~09:54:0''; apply the same timerange for

all input MSes

uvrange Select data within uvrange (default unit is meters)
default: '' (all); example: uvrange='0~1000klambda'; uvrange from 0-1000 kilo-lambda uvrange='> 4klambda';uvranges greater than 4 kilo lambda For multiple MS input, a list of uvrange strings can be used: uvrange=['0~1000klambda','100~1000klamda'] uvrange='0~1000klambda'; apply 0-1000 kilo-lambda for all

input MSes

antenna Select data based on antenna/baseline

default: '' (all) If antenna string is a non-negative integer, it is

assumed to be an antenna index, otherwise, it is considered an antenna name.

antenna='5&6'; baseline between antenna index 5 and
index 6.

antenna='VA05&VA06'; baseline between VLA antenna 5
and 6.

antenna='5&6;7&8'; baselines 5-6 and 7-8 antenna='5'; all baselines with antenna index 5 antenna='05'; all baselines with antenna number 05

(VLA old name)

antenna='5,6,9'; all baselines with antennas 5,6,9
index number

For multiple MS input, a list of antenna strings can be used: antenna=['5','5&6']; antenna='5'; antenna index 5 for all input MSes antenna='!DV14'; use all antennas except DV14

scan Scan number range

default: '' (all) example: scan='1~5' For multiple MS input, a list of scan strings can be used: scan=['0~100','10~200'] scan='0~100; scan ids 0-100 for all input MSes

observation Observation ID range
default: '' (all) example: observation='1~5'

intent Scan Intent(s)

default: '' (all) example: intent='TARGET_SOURCE' example: intent='TARGET_SOURCE1,TARGET_SOURCE2' example: intent='TARGET_POINTING*'

datacolumn Data column to image (data or observed, corrected)
default:'data' (If 'corrected' does not exist, it will use 'data' instead)

imsize Number of pixels

example
[imsize = [350,250]] imsize = 500 is equivalent to [500,500]

To take proper advantage of internal optimized FFT routines, the number of pixels must be even and factorizable by 2,3,5,7 only.

cell Cell size
example: cell=['0.5arcsec,'0.5arcsec'] or cell=['1arcmin', '1arcmin'] cell = '1arcsec' is equivalent to ['1arcsec','1arcsec']

phasecenter Phase center of the image (string or field id); if the phasecenter is the name known major solar system object ('MERCURY', 'VENUS', 'MARS', 'JUPITER', 'SATURN', 'URANUS', 'NEPTUNE', 'PLUTO', 'SUN', 'MOON') or is an ephemerides table then that source is tracked and the background sources get smeared. There is a special case, when phasecenter='TRACKFIELD', which will use the ephemerides or polynomial phasecenter in the FIELD table of the MS's as the source center to track.

example: phasecenter=6

phasecenter='J2000 19h30m00 -40d00m00' phasecenter='J2000 292.5deg -40.0deg' phasecenter='J2000 5.105rad -0.698rad' phasecenter='ICRS 13:05:27.2780 -049.28.04.458'

phasecenter='myComet_ephem.tab' phasecenter='MOON' phasecenter='TRACKFIELD'

stokes Stokes Planes to make

default='I'; example: stokes='IQUV';

Options: 'I','Q','U','V','IV','QU','IQ','UV','IQUV','RR','LL','XX','YY','RRLL','XXYY','pseudoI'

Note

[Due to current internal code constraints, if any correlation pair] is flagged, by default, no data for that row in the MS will be used. So, in an MS with XX,YY, if only YY is flagged, neither a Stokes I image nor an XX image can be made from those data points. In such a situation, please split out only the unflagged correlation into a separate MS.

Note

[The 'pseudoI' option is a partial solution, allowing Stokes I imaging] when either of the parallel-hand correlations are unflagged.

The remaining constraints shall be removed (where logical) in a future release.

projection Coordinate projection

Examples : SIN, NCP A list of supported (but untested) projections can be found here : http://casa.nrao.edu/active/docs/doxygen/html/classcasa_1_1Projection.html#a3d5f9ec787e4eabdce57ab5edaf7c0cd

startmodel Name of starting model image

The contents of the supplied starting model image will be copied to the imagename.model before the run begins.

example : startmodel = 'singledish.im'

For deconvolver='mtmfs', one image per Taylor term must be provided. example : startmodel = ['try.model.tt0', 'try.model.tt1']

startmodel = ['try.model.tt0'] will use a starting model only
for the zeroth order term.

startmodel = ['','try.model.tt1'] will use a starting model only
for the first order term.

This starting model can be of a different image shape and size from what is currently being imaged. If so, an image regrid is first triggered to resample the input image onto the target coordinate system.

A common usage is to set this parameter equal to a single dish image

Negative components in the model image will be included as is.

[Note

[If an error occurs during image resampling/regridding,] please try using task imregrid to resample the starting model image onto a CASA image with the target shape and coordinate system before supplying it via startmodel]

specmode Spectral definition mode (mfs,cube,cubedata, cubesource)

mode='mfs'
[Continuum imaging with only one output image channel.] (mode='cont' can also be used here)

mode='cube'
[Spectral line imaging with one or more channels]

Parameters start, width,and nchan define the spectral coordinate system and can be specified either in terms of channel numbers, frequency or velocity in whatever spectral frame is specified in 'outframe'. All internal and output images are made with outframe as the base spectral frame. However imaging code internally uses the fixed spectral frame, LSRK for automatic internal software Doppler tracking so that a spectral line observed over an extended time range will line up appropriately. Therefore the output images have additional spectral frame conversion layer in LSRK on the top the base frame.

(Note
[Even if the input parameters are specified in a frame] other than LSRK, the viewer still displays spectral axis in LSRK by default because of the conversion frame layer mentioned above. The viewer can be used to relabel the spectral axis in any desired frame - via the spectral reference option under axis label properties in the data display options window.)

mode='cubedata'
[Spectral line imaging with one or more channels] There is no internal software Doppler tracking so a spectral line observed over an extended time range may be smeared out in frequency. There is strictly no valid spectral frame with which to label the output

images, but they will list the frame defined in the MS.

mode='cubesource': Spectral line imaging while tracking moving source (near field or solar system objects). The velocity of the source is accounted and the frequency reported is in the source frame. As there is not SOURCE frame defined, the frame reported will be REST (as it may not be in the rest frame emission region may be moving w.r.t the systemic velocity frame)

reffreq Reference frequency of the output image coordinate system

Example : reffreq='1.5GHz' as a string with units.

By default, it is calculated as the middle of the selected frequency range.

For deconvolver='mtmfs' the Taylor expansion is also done about this specified reference frequency.

nchan Number of channels in the output image
For default (=-1), the number of channels will be automatically determined based on data selected by 'spw' with 'start' and 'width'. It is often easiest to leave nchan at the default value. example: nchan=100

start First channel (e.g. start=3,start='1.1GHz',start='15343km/s')
of output cube images specified by data channel number (integer), velocity (string with a unit), or frequency (string with a unit). Default:''; The first channel is automatically determined based on the 'spw' channel selection and 'width'. When the channel number is used along with the channel selection

in 'spw' (e.g. spw='0:6~100'),

'start' channel number is RELATIVE (zero-based) to the selected channels in 'spw'. So for the above example, start=1 means that the first image channel is the second selected data channel, which is channel

7. For specmode='cube', when velocity or frequency is used it is interpreted with the frame defined in outframe. [The parameters of the desired output cube can be estimated by using the 'transform' functionality of 'plotms'] examples: start='5.0km/s'; 1st channel, 5.0km/s in outframe

start='22.3GHz'; 1st channel, 22.3GHz in outframe

width Channel width (e.g. width=2,width='0.1MHz',width='10km/s') of output cube images

specified by data channel number (integer), velocity (string with a unit), or or frequency (string with a unit). Default:''; data channel width The sign of width defines the direction of the channels to be incremented. For width specified in velocity or frequency with '-' in front gives image channels in decreasing velocity or frequency, respectively. For specmode='cube', when velocity or frequency is used it is interpreted with the reference frame defined in outframe. examples: width='2.0km/s'; results in channels with increasing velocity

width='-2.0km/s'; results in channels with decreasing velocity width='40kHz'; results in channels with increasing frequency width=-2; results in channels averaged of 2 data channels incremented from

high to low channel numbers

outframe Spectral reference frame in which to interpret 'start' and 'width'

Options: '','LSRK','LSRD','BARY','GEO','TOPO','GALACTO','LGROUP','CMB' example: outframe='bary' for Barycentric frame

REST – Rest frequency LSRD – Local Standard of Rest (J2000)

– as the dynamical definition (IAU, [9,12,7] km/s in galactic coordinates)

LSRK – LSR as a kinematical (radio) definition
– 20.0 km/s in direction ra,dec = [270,+30] deg (B1900.0)

BARY – Barycentric (J2000) GEO — Geocentric TOPO – Topocentric GALACTO – Galacto centric (with rotation of 220 km/s in direction l,b = [90,0] deg. LGROUP – Local group velocity – 308km/s towards l,b = [105,-7] deg (F. Ghigo)

CMB – CMB velocity – 369.5km/s towards l,b = [264.4, 48.4] deg (F. Ghigo) DEFAULT = LSRK

veltype Velocity type (radio, z, ratio, beta, gamma, optical)

For start and/or width specified in velocity, specifies the velocity definition Options: 'radio','optical','z','beta','gamma','optical' NOTE: the viewer always defaults to displaying the 'radio' frame,

but that can be changed in the position tracking pull down.

The different types (with F = f/f0, the frequency ratio), are:

Z = (-1 + 1/F)

RATIO = (F) * RADIO = (1 - F) OPTICAL == Z BETA = ((1 - F2)/(1 + F2)) GAMMA = ((1 + F2)/2F) * RELATIVISTIC == BETA (== v/c) DEFAULT == RADIO Note that the ones with an '*' have no real interpretation (although the calculation will proceed) if given as a velocity.

restfreq List of rest frequencies or a rest frequency in a string.

Specify rest frequency to use for output image. *Currently it uses the first rest frequency in the list for translation of velocities. The list will be stored in the output images. Default: []; look for the rest frequency stored in the MS, if not available, use center frequency of the selected channels examples: restfreq=['1.42GHz']

restfreq='1.42GHz'

interpolation Spectral interpolation (nearest,linear,cubic)

Interpolation rules to use when binning data channels onto image channels and evaluating visibility values at the centers of image channels.

Note

['linear' and 'cubic' interpolation requires data points on both sides of] each image frequency. Errors are therefore possible at edge channels, or near flagged data channels. When image channel width is much larger than the data channel width there is nothing much to be gained using linear or cubic thus not worth the extra computation involved.

gridder Gridding options (standard, wproject, widefield, mosaic, awproject)

The following options choose different gridding convolution functions for the process of convolutional resampling of the measured visibilities onto a regular uv-grid prior to an inverse FFT. Model prediction (degridding) also uses these same functions. Several wide-field effects can be accounted for via careful choices of convolution functions. Gridding (degridding) runtime will rise in proportion to the support size of these convolution functions (in uv-pixels).

standard : Prolate Spheroid with 3x3 uv pixel support size

[This mode can also be invoked using 'ft' or 'gridft']

wproject

[W-Projection algorithm to correct for the widefield]

non-coplanar baseline effect. [Cornwell et.al 2008]

wprojplanes is the number of distinct w-values at which to compute and use different gridding convolution functions (see help for wprojplanes).

Convolution function support size can range

from 5x5 to few 100 x few 100.

[This mode can also be invoked using 'wprojectft']

widefield : Facetted imaging with or without W-Projection per facet.

A set of facets x facets subregions of the specified image are gridded separately using their respective phase centers (to minimize max W). Deconvolution is done on the joint full size image, using a PSF from the first subregion.

wprojplanes=1 : standard prolate spheroid gridder per facet. wprojplanes > 1 : W-Projection gridder per facet. nfacets=1, wprojplanes > 1 : Pure W-Projection and no facetting nfacets=1, wprojplanes=1 : Same as standard,ft,gridft

A combination of facetting and W-Projection is relevant only for very large fields of view.

mosaic

[A-Projection with azimuthally symmetric beams without]

sidelobes, beam rotation or squint correction. Gridding convolution functions per visibility are computed from FTs of PB models per antenna. This gridder can be run on single fields as well as mosaics.

VLA : PB polynomial fit model (Napier and Rots, 1982) EVLA : PB polynomial fit model (Perley, 2015) ALMA : Airy disks for a 10.7m dish (for 12m dishes) and

6.25m dish (for 7m dishes) each with 0.75m blockages (Hunter/Brogan 2011). Joint mosaic imaging supports heterogeneous arrays for ALMA.

Typical gridding convolution function support sizes are between 7 and 50 depending on the desired accuracy (given by the uv cell size or image field of view).

[This mode can also be invoked using 'mosaicft' or 'ftmosaic']

awproject
[A-Projection with azimuthally asymmetric beams and]

including beam rotation, squint correction, conjugate frequency beams and W-projection. [Bhatnagar et.al, 2008]

Gridding convolution functions are computed from aperture illumination models per antenna and optionally combined with W-Projection kernels and a prolate spheroid. This gridder can be run on single fields as well as mosaics.

VLA
[Uses ray traced model (VLA and EVLA) including feed] leg and subreflector shadows, off-axis feed location (for beam squint and other polarization effects), and a Gaussian fit for the feed beams (Ref: Brisken 2009)

ALMA
[Similar ray-traced model as above (but the correctness] of its polarization properties remains un-verified).

Typical gridding convolution function support sizes are between 7 and 50 depending on the desired accuracy (given by the uv cell size or image field of view). When combined with W-Projection they can be significantly larger.

[This mode can also be invoked using 'awprojectft']

imagemosaic
[(untested implementation)] Grid and iFT each pointing separately and combine the images as a linear mosaic (weighted by a PB model) in the image domain before a joint minor cycle.

VLA/ALMA PB models are same as for gridder='mosaicft'

—— Notes on PB models :

(1) **Several different sources of PB models are used in the modes** listed above. This is partly for reasons of algorithmic flexibility and partly due to the current lack of a common beam model repository or consensus on what beam models are most appropriate.

(2) **For ALMA and gridder='mosaic', ray-traced (TICRA) beams** are also available via the vpmanager tool. For example, call the following before the tclean run.

vp.setpbimage(telescope="ALMA", compleximage='/home/casa/data/trunk/alma/responses/ALMA_0_DV__0_0_
antnames=['DV'+'%02d'%k for k in range(25)]) vp.saveastable('mypb.tab') Then, supply vptable='mypb.tab' to tclean. (Currently this will work only for non-parallel runs)

—— Note on PB masks :

In tclean, A-Projection gridders (mosaic and awproject) produce a .pb image and use the 'pblimit' subparameter to decide normalization cutoffs and construct an internal T/F mask in the .pb and .image images. However, this T/F mask cannot directly be used during deconvolution (which needs a 1/0 mask). There are two options for making a pb based deconvolution mask.

– Run tclean with niter=0 to produce the .pb, construct a 1/0 image

with the desired threshold (using ia.open('newmask.im'); ia.calc('iif("xxx.pb">0.3,1.0,0.0)');ia.close() for example), and supply it via the 'mask' parameter in a subsequent run (with calcres=F and calcpsf=F to restart directly from the minor cycle).

– Run tclean with usemask='pb' for it to automatically construct

a 1/0 mask from the internal T/F mask from .pb at a fixed 0.2 threshold.

—— Making PBs for gridders other than mosaic,awproject

After the PSF generation, a PB is constructed using the same models used in gridder='mosaic' but just evaluated in the image domain without consideration to weights.

facets Number of facets on a side

A set of (facets x facets) subregions of the specified image are gridded separately using their respective phase centers (to minimize max W). Deconvolution is done on the joint full size image, using a PSF from the first subregion/facet.

chanchunks Number of channel chunks to grid separately

For large image cubes, the gridders can run into memory limits as they loop over all available image planes for each row of data accessed. To prevent this problem, we can grid subsets of channels in sequence so that at any given time only part of the image cube needs to be loaded into memory. This parameter controls the number of chunks to split the cube into.

Example : chanchunks = 4

[This feature is experimental and may have restrictions on how
chanchunks is to be chosen. For now, please pick chanchunks so that nchan/chanchunks is an integer.]

wprojplanes Number of distinct w-values at which to compute and use different
gridding convolution functions for W-Projection

An appropriate value of wprojplanes depends on the presence/absence of a bright source far from the phase center, the desired dynamic range of an image in the presence of a bright far out source, the maximum w-value in the measurements, and the desired trade off between accuracy and computing cost.

As a (rough) guide, VLA L-Band D-config may require a value of 128 for a source 30arcmin away from the phase center. A-config may require 1024 or more. To converge to an appropriate value, try starting with 128 and then increasing it if artifacts persist. W-term artifacts (for the VLA) typically look like arc-shaped smears in a synthesis image or a shift in source position between images made at different times. These artifacts are more pronounced the further the source is from the phase center.

There is no harm in simply always choosing a large value (say, 1024) but there will be a significant performance cost to doing so, especially for gridder='awproject' where it is combined with A-Projection.

wprojplanes=-1 is an option for gridder='widefield' or 'wproject' in which the number of planes is automatically computed.

vptable vpmanager

vptable=""

[Choose default beams for different telescopes] ALMA : Airy disks EVLA : old VLA models.

Other primary beam models can be chosen via the vpmanager tool.

Step 1 : Set up the vpmanager tool and save its state in a table

vp.setpbpoly(telescope='EVLA', coeff=[1.0, -1.529e-3, 8.69e-7, -1.88e-10])
vp.saveastable('myvp.tab')

Step 2 : Supply the name of that table in tclean.

tclean(...., vptable='myvp.tab',....)

Please see the documentation for the vpmanager for more details on how to choose different beam models. Work is in progress to update the defaults for EVLA and ALMA.

Note

[AWProjection currently does not use this mechanism to choose] beam models. It instead uses ray-traced beams computed from parameterized aperture illumination functions, which are not available via the vpmanager. So, gridder='awproject' does not allow the user to set this parameter.

usepointing Use the pointing table to determine where the beam are for mosaic gridder; if False then phasecenters of the fields selected are used to determine direction of each mosaic pointing. mosweight When doing Brigg's style weighting (including uniform) to perform the weight density calculation for each field indepedently if True. If False the weight density is calculated from the average uv distribution of all the fields. aterm Use aperture illumination functions during gridding

This parameter turns on the A-term of the AW-Projection gridder. Gridding convolution functions are constructed from aperture illumination function models of each antenna.

psterm Use prolate spheroidal during gridding wbawp Use frequency dependent A-terms

Scale aperture illumination functions appropriately with frequency when gridding and combining data from multiple channels.

conjbeams Use conjugate frequency for wideband A-terms

While gridding data from one frequency channel, choose a convolution function from a 'conjugate' frequency such that the resulting baseline primary beam is approximately constant across frequency. For a system in which the primary beam scales with frequency, this step will eliminate instrumental spectral structure from the measured data and leave only the sky spectrum for the minor cycle to model and reconstruct [Bhatnagar et.al,2013].

As a rough guideline for when this is relevant, a source at the half power point of the PB at the center frequency will see an artificial spectral index of -1.4 due to the frequency dependence of the PB [Sault and Wieringa, 1994]. If left uncorrected during gridding, this spectral structure must be modeled in the minor cycle (using the mtmfs algorithm) to avoid dynamic range limits (of a few hundred for a 2:1 bandwidth).

cfcache Convolution function cache directory name

Name of a directory in which to store gridding convolution functions. This cache is filled at the beginning of an imaging run. This step can be time consuming but the cache can be reused across

multiple imaging runs that use the same image parameters (cell size, field-of-view, spectral data selections, etc).

By default, cfcache = imagename + '.cf'

computepastep At what parallactic angle interval to recompute aperture
illumination functions (deg)

This parameter controls the accuracy of the aperture illumination function used with AProjection for alt-az mount dishes where the AIF rotates on the sky as the synthesis image is built up.

rotatepastep At what parallactic angle interval to rotate nearest
aperture illumination function (deg)

Instead of recomputing the AIF for every timestep's parallactic angle, the nearest existing AIF is picked and rotated in steps of this amount.

For example, computepastep=360.0 and rotatepastep=5.0 will compute the AIFs at only the starting parallactic angle and all other timesteps will use a rotated version of that AIF at the nearest 5.0 degree point.

pblimit PB gain level at which to cut off normalizations

Divisions by .pb during normalizations have a cut off at a .pb gain level given by pblimit. Outside this limit, image values are set to zero. Additionally, by default, an internal T/F mask is applied to the .pb, .image and .residual images to mask out (T) all invalid pixels outside the pblimit area.

Note
[This internal T/F mask cannot be used as a deconvolution mask.] To do so, please follow the steps listed above in the Notes for the 'gridder' parameter.

Note
[To prevent the internal T/F mask from appearing in anything other] than the .pb and .image.pbcor images, 'pblimit' can be set to a negative number. The absolute value will still be used as a valid 'pblimit'. A tclean restart using existing output images on disk that already have this T/F mask in the .residual and .image but only pblimit set to a negative value, will remove this mask after the next major cycle.

normtype Normalization type (flatnoise, flatsky, pbsquare)

Gridded (and FT'd) images represent the PB-weighted sky image. Qualitatively it can be approximated as two instances of the PB applied to the sky image (one naturally present in the data and one introduced during gridding via the convolution functions).

xxx.weight : Weight image approximately equal to sum (square (pb)) xxx.pb : Primary beam calculated as sqrt (xxx.weight)

normtype='flatnoise'
[Divide the raw image by sqrt(.weight) so that] the input to the minor cycle represents the product of the sky and PB. The noise is 'flat' across the region covered by each PB.

normtype='flatsky'
[Divide the raw image by .weight so that the input] to the minor cycle represents only the sky. The noise is higher in the outer regions of the primary beam where the sensitivity is low.

normtype='pbsquare'
[No normalization after gridding and FFT.] The minor cycle sees the sky times pb square

deconvolver Name of minor cycle algorithm (hogbom,clark,multiscale,mtmfs,mem,clarkstokes)

Each of the following algorithms operate on residual images and psfs from the gridder and produce output model and restored images. Minor cycles stop and a major cycle is triggered when cyclethreshold or cycleniter are reached. For all methods, components are picked from the entire extent of the image or (if specified) within a mask.

hogbom
[An adapted version of Hogbom Clean [Hogbom, 1974]]

- Find the location of the peak residual
- Add this delta function component to the model image
- Subtract a scaled and shifted PSF of the same size as the image from regions of the residual image where the two overlap.
- Repeat

clark
[An adapted version of Clark Clean [Clark, 1980]]

- Find the location of max(I^2+Q^2+U^2+V^2)
- Add delta functions to each stokes plane of the model image
- Subtract a scaled and shifted PSF within a small patch size from regions of the residual image where the two overlap.
- After several iterations trigger a Clark major cycle to subtract components from the visibility domain, but without de-gridding.
- Repeat

(Note
['clark' maps to imagermode='' in the old clean task.]

'clark_exp' is another implementation that maps to
imagermode='mosaic' or 'csclean' in the old clean task but the behavior is not identical. For now, please use deconvolver='hogbom' if you encounter problems.)

clarkstokes : Clark Clean operating separately per Stokes plane

(Note : 'clarkstokes_exp' is an alternate version. See above.)

multiscale
[MultiScale Clean [Cornwell, 2008]]

- Smooth the residual image to multiple scale sizes
- Find the location and scale at which the peak occurs
- Add this multiscale component to the model image
- Subtract a scaled,smoothed,shifted PSF (within a small patch size per scale) from all residual images
- Repeat from step 2

mtmfs
[Multi-term (Multi Scale) Multi-Frequency Synthesis [Rau and Cornwell, 2011]]

- Smooth each Taylor residual image to multiple scale sizes

- Solve a NTxNT system of equations per scale size to compute Taylor coefficients for components at all locations
- **Compute gradient chi-square and pick the Taylor coefficients**
  and scale size at the location with maximum reduction in chi-square
- Add multi-scale components to each Taylor-coefficient model image
- Subtract scaled,smoothed,shifted PSF (within a small patch size per scale) from all smoothed Taylor residual images
- Repeat from step 2

mem
[Maximum Entropy Method [Cornwell and Evans, 1985]]

- Iteratively solve for values at all individual pixels via the MEM method. It minimizes an objective function of

  chi-square plus entropy (here, a measure of difference

  between the current model and a flat prior model).

  (Note
  [This MEM implementation is not very robust.] Improvements will be made in the future.)

scales List of scale sizes (in pixels) for multi-scale and mtmfs algorithms.
–> scales=[0,6,20] This set of scale sizes should represent the sizes (diameters in units of number of pixels) of dominant features in the image being reconstructed.

The smallest scale size is recommended to be 0 (point source), the second the size of the synthesized beam and the third 3-5 times the synthesized beam, etc. For example, if the synthesized beam is 10" FWHM and cell=2",try scales = [0,5,15].

For numerical stability, the largest scale must be smaller than the image (or mask) size and smaller than or comparable to the scale corresponding to the lowest measured spatial frequency (as a scale size much larger than what the instrument is sensitive to is unconstrained by the data making it harder to recovery from errors during the minor cycle).

nterms Number of Taylor coefficients in the spectral model

- nterms=1 : Assume flat spectrum source
- nterms=2 : Spectrum is a straight line with a slope
- nterms=N : A polynomial of order N-1

From a Taylor expansion of the expression of a power law, the spectral index is derived as alpha = taylorcoeff_1 / taylorcoeff_0

Spectral curvature is similarly derived when possible.

The optimal number of Taylor terms depends on the available signal to noise ratio, bandwidth ratio, and spectral shape of the source as seen by the telescope (sky spectrum x PB spectrum).

nterms=2 is a good starting point for wideband EVLA imaging and the lower frequency bands of ALMA (when fractional bandwidth is greater than 10%) and if there is at least one bright source for which a dynamic range of greater than few 100 is desired.

Spectral artifacts for the VLA often look like spokes radiating out from a bright source (i.e. in the image made with standard mfs imaging). If increasing the number of terms does not eliminate these

artifacts, check the data for inadequate bandpass calibration. If the source is away from the pointing center, consider including wide-field corrections too.

(Note

[In addition to output Taylor coefficient images .tt0,.tt1,etc] images of spectral index (.alpha), an estimate of error on spectral index (.alpha.error) and spectral curvature (.beta, if nterms is greater than 2) are produced. - These alpha, alpha.error and beta images contain

internal T/F masks based on a threshold computed as peakresidual/10. Additional masking based on

.alpha/.alpha.error may be desirable.

- .alpha.error is a purely empirical estimate derived from the propagation of error during the division of two noisy numbers (alpha = xx.tt1/xx.tt0) where the 'error' on tt1 and tt0 are simply the values picked from the corresponding residual images. The absolute value of the error is not always accurate and it is best to interpret the errors across the image only in a relative sense.)

smallscalebias A numerical control to bias the solution towards smaller scales.

The peak from each scale's smoothed residual is multiplied by (1 - smallscalebias * scale/maxscale) to increase or decrease the amplitude relative to other scales, before the scale with the largest peak is chosen.

smallscalebias=0.6 (default) applies a slight bias towards small
scales, ranging from 1.0 for a point source to 0.4 for the largest scale size

Values larger than 0.6 will bias the solution towards smaller scales. Values smaller than 0.6 will tend towards giving all scales equal weight.

restoration e.

Construct a restored image : imagename.image by convolving the model image with a clean beam and adding the residual image to the result. If a restoringbeam is specified, the residual image is also smoothed to that target resolution before adding it in.

If a .model does not exist, it will make an empty one and create the restored image from the residuals (with additional smoothing if needed). With algorithm='mtmfs', this will construct Taylor coefficient maps from the residuals and compute .alpha and .alpha.error.

restoringbeam ize to use.

- restoringbeam=" or ['] A Gaussian fitted to the PSF main lobe (separately per image plane).
- restoringbeam='10.0arcsec' Use a circular Gaussian of this width for all planes
- restoringbeam=['8.0arcsec','10.0arcsec','45deg'] Use this elliptical Gaussian for all planes
- restoringbeam='common' Automatically estimate a common beam shape/size appropriate for all planes.

Note

[For any restoring beam different from the native resolution] the model image is convolved with the beam and added to residuals that have been convolved to the same target resolution.

pbcor the output restored image

A new image with extension .image.pbcor will be created from the evaluation of .image / .pb for all pixels above the specified pblimit.

Note

[Stand-alone PB-correction can be triggered by re-running] tclean with the appropriate imagename and with niter=0, calcpsf=False, calcres=False, pbcor=True, vptable='vp.tab' (where vp.tab is the name of the vpmanager file.

See the inline help for the 'vptable' parameter)

Note

[Multi-term PB correction that includes a correction for the] spectral index of the PB has not been enabled for the 4.7 release. Please use the widebandpbcor task instead. (Wideband PB corrections are required when the amplitude of the

brightest source is known accurately enough to be sensitive to the difference in the PB gain between the upper and lower end of the band at its location. As a guideline, the artificial spectral index due to the PB is -1.4 at the 0.5 gain level and less than -0.2 at the 0.9 gain level at the middle frequency)

outlierfile Name of outlier-field image definitions

A text file containing sets of parameter=value pairs, one set per outlier field.

Example : outlierfile='outs.txt'

Contents of outs.txt :

imagename=tst1 nchan=1 imsize=[80,80] cell=[8.0arcsec,8.0arcsec] phasecenter=J2000 19:58:40.895 +40.55.58.543 mask=circle[[40pix,40pix],10pix]

imagename=tst2 nchan=1 imsize=[100,100] cell=[8.0arcsec,8.0arcsec] phasecenter=J2000 19:58:40.895 +40.56.00.000 mask=circle[[60pix,60pix],20pix]

The following parameters are currently allowed to be different between the main field and the outlier fields (i.e. they will be recognized if found in the outlier text file). If a parameter is not listed, the value is picked from what is defined in the main task input.

imagename, imsize, cell, phasecenter, startmodel, mask specmode, nchan, start, width, nterms, reffreq, gridder, deconvolver, wprojplanes

Note

['specmode' is an option, so combinations of mfs and cube]

for different image fields, for example, are supported.

'deconvolver' and 'gridder' are also options that allow different

imaging or deconvolution algorithm per image field.

For example, multiscale with wprojection and 16 w-term planes on the main field and mtmfs with nterms=3 and wprojection with 64 planes on a bright outlier source for which the frequency dependence of the primary beam produces a strong effect that must be modeled. The traditional alternative to this approach is to first image the outlier, subtract it out of the data (uvsub) and then image the main field.

Note

[If you encounter a use-case where some other parameter needs] to be allowed in the outlier file (and it is logical to do so), please send us feedback. The above is an initial list.

weighting Weighting scheme (natural,uniform,briggs,superuniform,radial)

During gridding of the dirty or residual image, each visibility value is multiplied by a weight before it is accumulated on the uv-grid. The PSF's uv-grid is generated by gridding only the weights (weightgrid).

weighting='natural'
[Gridding weights are identical to the data weights] from the MS. For visibilities with similar data weights, the weightgrid will follow the sample density pattern on the uv-plane. This weighting scheme provides the maximum imaging sensitivity at the expense of a possibly fat PSF with high sidelobes. It is most appropriate for detection experiments where sensitivity is most important.

weighting='uniform'
[Gridding weights per visibility data point are the] original data weights divided by the total weight of all data points that map to the same uv grid cell : ' data_weight / total_wt_per_cell '.

The weightgrid is as close to flat as possible resulting in a PSF with a narrow main lobe and suppressed sidelobes. However, since heavily sampled areas of the uv-plane get down-weighted, the imaging sensitivity is not as high as with natural weighting. It is most appropriate for imaging experiments where a well behaved PSF can help the reconstruction.

weighting='briggs'
[Gridding weights per visibility data point are given by] 'data_weight / (A / total_wt_per_cell + B) ' where A and B vary according to the 'robust' parameter.

robust = -2.0 maps to A=1,B=0 or uniform weighting. robust = +2.0 maps to natural weighting. (robust=0.5 is equivalent to robust=0.0 in AIPS IMAGR.)

Robust/Briggs weighting generates a PSF that can vary smoothly between 'natural' and 'uniform' and allow customized trade-offs between PSF shape and imaging sensitivity.

weighting='superuniform'
[This is similar to uniform weighting except that]

the total_wt_per_cell is replaced by the total_wt_within_NxN_cells around the uv cell of interest. (N = subparameter 'npixels')

This method tends to give a PSF with inner sidelobes that are suppressed as in uniform weighting but with far-out sidelobes closer to natural weighting. The peak sensitivity is also closer to natural weighting.

weighting='radial' : Gridding weights are given by ' data_weight * uvdistance '

This method approximately minimizes rms sidelobes for an east-west synthesis array.

For more details on weighting please see Chapter3 of Dan Briggs' thesis (http://www.aoc.nrao.edu/dissertations/dbriggs)

robust Robustness parameter for Briggs weighting.

robust = -2.0 maps to uniform weighting. robust = +2.0 maps to natural weighting. (robust=0.5 is equivalent to robust=0.0 in AIPS IMAGR.)

npixels Number of pixels to determine uv-cell size for super-uniform weighting

(0 defaults to -/+ 3 pixels)

npixels – uv-box used for weight calculation

a box going from -npixel/2 to +npixel/2 on each side

around a point is used to calculate weight density.

npixels=2 goes from -1 to +1 and covers 3 pixels on a side.

npixels=0 implies a single pixel, which does not make sense for
superuniform weighting. Therefore, if npixels=0 it will be forced to 6 (or a box of -3pixels to +3pixels) to cover 7 pixels on a side.

uvtaper uv-taper on outer baselines in uv-plane

Apply a Gaussian taper in addition to the weighting scheme specified via the 'weighting' parameter. Higher spatial frequencies are weighted down relative to lower spatial frequencies to suppress artifacts arising from poorly sampled areas of the uv-plane. It is equivalent to smoothing the PSF obtained by other weighting schemes and can be specified either as a Gaussian in uv-space (eg. units of lambda) or as a Gaussian in the image domain (eg. angular units like arcsec).

uvtaper = [bmaj, bmin, bpa]

NOTE: the on-sky FWHM in arcsec is roughly the uv taper/200 (klambda). default: uvtaper=[]; no Gaussian taper applied

example: uvtaper=['5klambda'] circular taper

FWHM=5 kilo-lambda
uvtaper=['5klambda','3klambda','45.0deg'] uvtaper=['10arcsec'] on-sky FWHM 10 arcseconds uvtaper=['300.0'] default units are lambda

in aperture plane

niter Maximum number of iterations

A stopping criterion based on total iteration count.

Iterations are typically defined as the selecting one flux component and partially subtracting it out from the residual image.

niter=0 : Do only the initial major cycle (make dirty image, psf, pb, etc)

niter larger than zero : Run major and minor cycles.

Note : Global stopping criteria vs major-cycle triggers

In addition to global stopping criteria, the following rules are used to determine when to terminate a set of minor cycle iterations and trigger major cycles [derived from Cotton-Schwab Clean, 1984]

'cycleniter'
[controls the maximum number of iterations per image] plane before triggering a major cycle.

'cyclethreshold'
[Automatically computed threshold related to the]

max sidelobe level of the PSF and peak residual.

Divergence, detected as an increase of 10% in peak residual from the minimum so far (during minor cycle iterations)

The first criterion to be satisfied takes precedence.

Note

[Iteration counts for cubes or multi-field images :]

For images with multiple planes (or image fields) on which the deconvolver operates in sequence, iterations are counted across all planes (or image fields). The iteration count is compared with 'niter' only after all channels/planes/fields have completed their minor cycles and exited either due to 'cycleniter' or 'cyclethreshold'. Therefore, the actual number of iterations reported in the logger can sometimes be larger than the user specified value in 'niter'. For example, with niter=100, cycleniter=20,nchan=10,threshold=0, a total of 200 iterations will be done in the first set of minor cycles before the total is compared with niter=100 and it exits.

Note

[Additional global stopping criteria include]

- no change in peak residual across two major cycles
- a 50% or more increase in peak residual across one major cycle

gain Loop gain

Fraction of the source flux to subtract out of the residual image for the CLEAN algorithm and its variants.

A low value (0.2 or less) is recommended when the sky brightness distribution is not well represented by the basis functions used by the chosen deconvolution algorithm. A higher value can be tried when there is a good match between the true sky brightness structure and the basis function shapes. For example, for extended emission, multiscale clean with an appropriate set of scale sizes will tolerate a higher loop gain than Clark clean (for example).

threshold Stopping threshold (number in units of Jy, or string)

A global stopping threshold that the peak residual (within clean mask) across all image planes is compared to.

threshold = 0.005 : 5mJy threshold = '5.0mJy'

Note

[A 'cyclethreshold' is internally computed and used as a major cycle]

trigger. It is related what fraction of the PSF can be reliably used during minor cycle updates of the residual image. By default the minor cycle iterations terminate once the peak residual reaches the first sidelobe level of the brightest source.

'cyclethreshold' is computed as follows using the settings in
parameters 'cyclefactor','minpsffraction','maxpsffraction','threshold' :

psf_fraction = max_psf_sidelobe_level * 'cyclefactor' psf_fraction = max(psf_fraction, 'minpsffraction'); psf_fraction = min(psf_fraction, 'maxpsffraction'); cyclethreshold = peak_residual * psf_fraction cyclethreshold = max(cyclethreshold, 'threshold')

If nsigma is set (>0.0), the N-sigma threshold is calculated (see the description under nsigma), then cyclethreshold is further modified as,

cyclethreshold = max(cyclethreshold, nsgima_threshold)

'cyclethreshold' is made visible and editable only in the interactive GUI when tclean is run with interactive=True.

nsigma Multiplicative factor for rms-based threshold stopping

N-sigma threshold is calculated as nsigma * rms value per image plane determined from a robust statistics. For nsigma > 0.0, in a minor cycle, a maximum of the two values, the N-sigma threshold and cyclethreshold, is used to trigger a major cycle (see also the descreption under 'threshold'). Set nsigma=0.0 to preserve the previous tclean behavior without this feature.

cycleniter Maximum number of minor-cycle iterations (per plane) before triggering
a major cycle

For example, for a single plane image, if niter=100 and cycleniter=20, there will be 5 major cycles after the initial one (assuming there is no threshold based stopping criterion). At each major cycle boundary, if the number of iterations left over (to reach niter) is less than cycleniter, it is set to the difference.

Note

[cycleniter applies per image plane, even if cycleniter x nplanes] gives a total number of iterations greater than 'niter'. This is to preserve consistency across image planes within one set of minor cycle iterations.

cyclefactor Scaling on PSF sidelobe level to compute the minor-cycle stopping threshold.

Please refer to the Note under the documentation for 'threshold' that discussed the calculation of 'cyclethreshold'

cyclefactor=1.0 results in a cyclethreshold at the first sidelobe level of the brightest source in the residual image before the minor cycle starts.

cyclefactor=0.5 allows the minor cycle to go deeper. cyclefactor=2.0 triggers a major cycle sooner.

minpsffraction PSF fraction that marks the max depth of cleaning in the minor cycle

Please refer to the Note under the documentation for 'threshold' that discussed the calculation of 'cyclethreshold'

For example, minpsffraction=0.5 will stop cleaning at half the height of the peak residual and trigger a major cycle earlier.

maxpsffraction PSF fraction that marks the minimum depth of cleaning in the minor cycle

Please refer to the Note under the documentation for 'threshold' that discussed the calculation of 'cyclethreshold'

For example, maxpsffraction=0.8 will ensure that at least the top 20 percent of the source will be subtracted out in the minor cycle even if the first PSF sidelobe is at the 0.9 level (an extreme example), or if the cyclefactor is set too high for anything to get cleaned.

interactive Modify masks and parameters at runtime

interactive=True will trigger an interactive GUI at every major cycle boundary (after the major cycle and before the minor cycle).

The interactive mode is currently not available for parallel cube imaging (please also refer to the Note under the documentation for 'parallel' below).

Options for runtime parameter modification are :

Interactive clean mask

[Draw a 1/0 mask (appears as a contour) by hand.] If a mask is supplied at the task interface or if automasking is invoked, the current mask is displayed in the GUI and is available for manual editing.

Note

[If a mask contour is not visible, please] check the cursor display at the bottom of GUI to

see which parts of the mask image have ones and zeros. If the entire mask=1 no contours will be visible.

Operation buttons
[– Stop execution now (restore current model and exit)]

– Continue on until global stopping criteria are reached
without stopping for any more interaction

– Continue with minor cycles and return for interaction
after the next major cycle.

Iteration control : – max cycleniter : Trigger for the next major cycle

The display begins with [min(cycleniter, niter - itercount)] and can be edited by hand.

—iterations left : The display begins with [niter-itercount] and can be edited to increase or decrease the total allowed niter.

– threshold : Edit global stopping threshold

—cyclethreshold : The display begins with the automatically computed value (see Note in help for 'threshold'), and can be edited by hand.

All edits will be reflected in the log messages that appear once minor cycles begin.

[For scripting purposes, replacing True/False with 1/0 will get tclean to
return an imaging summary dictionary to python]

usemask Type of mask(s) to be used for deconvolution

user: (default) mask image(s) or user specified region file(s) or string CRTF expression(s)
subparameters: mask, pbmask

pb: primary beam mask
subparameter: pbmask

Example: usemask="pb", pbmask=0.2
Construct a mask at the 0.2 pb gain level. (Currently, this option will work only with gridders that produce .pb (i.e. mosaic and awproject) or if an externally produced .pb image exists on disk)

auto-multithresh
[auto-masking by multiple thresholds for deconvolution]

subparameters
[sidelobethreshold, noisethreshold, lownoisethreshold, negativethrehsold, smoothfactor,] minbeamfrac, cutthreshold, pbmask, growiterations, dogrowprune, minpercentchange, verbose

if pbmask is >0.0, the region outside the specified pb gain level is excluded from image statistics in determination of the threshold.

Note: By default the intermediate mask generated by automask at each deconvolution cycle
is over-written in the next cycle but one can save them by setting the environment variable, SAVE_ALL_AUTOMASKS="true". (e.g. in the CASA prompt, os.environ['SAVE_ALL_AUTOMASKS']="true") The saved CASA mask image name will be imagename.mask.autothresh#, where # is the iteration cycle number.

mask Mask (a list of image name(s) or region file(s) or region string(s)

The name of a CASA image or region file or region string that specifies a 1/0 mask to be used for deconvolution. Only locations with value 1 will be considered for the centers of flux components in the minor cycle. If regions specified fall completely outside of the image, tclean will throw an error.

Manual mask options/examples :

mask='xxx.mask'
[Use this CASA image named xxx.mask and containing] ones and zeros as the mask. If the mask is only different in spatial coordinates from what is being made it will be resampled to the target coordinate system before being used. The mask has to have the same shape in velocity and Stokes planes as the output image. Exceptions are single velocity and/or single Stokes plane masks. They will be expanded to cover all velocity and/or Stokes planes of the output cube.

[Note
[If an error occurs during image resampling or] if the expected mask does not appear, please try using tasks 'imregrid' or 'makemask' to resample the mask image onto a CASA image with the target shape and coordinates and supply it via the 'mask' parameter.]

mask='xxx.crtf'
[A text file with region strings and the following on the first line] (#CRTFv0 CASA Region Text Format version 0) This is the format of a file created via the viewer's region tool when saved in CASA region file format.

mask='circle[[40pix,40pix],10pix]' : A CASA region string.

mask=['xxx.mask','xxx.crtf', 'circle[[40pix,40pix],10pix]'] : a list of masks

Note
[Mask images for deconvolution must contain 1 or 0 in each pixel.] Such a mask is different from an internal T/F mask that can be held within each CASA image. These two types of masks are not automatically interchangeable, so please use the makemask task to copy between them if you need to construct a 1/0 based mask from a T/F one.

Note
[Work is in progress to generate more flexible masking options and] enable more controls.

pbmask Sub-parameter for usemask='auto-multithresh': primary beam mask

Examples
[pbmask=0.0 (default, no pb mask)] pbmask=0.2 (construct a mask at the 0.2 pb gain level)

sidelobethreshold Sub-parameter for "auto-multithresh": mask threshold based on sidelobe levels: sidelobethreshold * max_sidelobe_level * peak residual noisethreshold Sub-parameter for "auto-multithresh": mask threshold based on the noise level: noisethreshold * rms

The rms is calculated from MAD with rms = 1.4826*MAD.

lownoisethreshold Sub-parameter for "auto-multithresh": mask threshold to grow previously masked regions via binary dilation: lownoisethreshold * rms in residual image

The rms is calculated from MAD with rms = 1.4826*MAD.

negativethreshold Sub-parameter for "auto-multithresh": mask threshold for negative features: -1.0* negativethreshold * rms

The rms is calculated from MAD with rms = 1.4826*MAD.

smoothfactor Sub-parameter for "auto-multithresh": smoothing factor in a unit of the beam minbeamfrac Sub-parameter for "auto-multithresh": minimum beam fraction in size to prune masks smaller than mimbeamfrac * beam

<=0.0 : No pruning

cutthreshold Sub-parameter for "auto-multithresh": threshold to cut the smoothed mask to create a final mask: cutthreshold * peak of the smoothed mask growiterations Sub-parameter for "auto-multithresh": Maximum number of iterations to perform using binary dilation for growing the mask dogrowprune Experimental sub-parameter for "auto-multithresh": Do pruning on the grow mask minpercentchange If the change in the mask size in a particular channel is less than minpercentchange, stop masking that channel in subsequent cycles. This check is only applied when noise based threshold is used and when the previous clean major cycle had a cyclethreshold value equal to the clean threshold. Values equal to -1.0 (or any value less than 0.0) will turn off this check (the default). Automask will still stop masking if the current channel mask is an empty mask and the noise threshold was used to determine the mask. verbose he summary of automasking at the end of each automasking process

is printed in the logger. Following information per channel will be listed in the summary.

chan: channel number masking?: F - stop updating automask for the subsequent iteration cycles RMS: robust rms noise peak: peak in residual image thresh_type: type of threshold used (noise or sidelobe) thresh_value: the value of threshold used N_reg: number of the automask regions N_pruned: number of the automask regions removed by pruning N_grow: number of the grow mask regions N_grow_pruned: number of the grow mask regions removed by pruning N_neg_pix: number of pixels for negative mask regions

Note that for a large cube, extra logging may slow down the process.

restart images (and start from an existing model image)
or automatically increment the image name and make a new image set.

True
[Re-use existing images. If imagename.model exists the subsequent]

run will start from this model (i.e. predicting it using current gridder settings and starting from the residual image). Care must be taken when combining this option with startmodel. Currently, only one or the other can be used.

startmodel='', imagename.model exists :

- Start from imagename.model

startmodel='xxx', imagename.model does not exist :

- Start from startmodel

startmodel='xxx', imagename.model exists :

- Exit with an error message requesting the user to pick only one model. This situation can arise when doing one run with startmodel='xxx' to produce an output imagename.model that includes the content of startmodel, and wanting to restart a second run to continue deconvolution. Startmodel should be set to '' before continuing.

If any change in the shape or coordinate system of the image is desired during the restart, please change the image name and use the startmodel (and mask) parameter(s) so that the old model (and mask) can be regridded to the new coordinate system before starting.

False
[A convenience feature to increment imagename with '_1', '_2',]

etc as suffixes so that all runs of tclean are fresh starts (without having to change the imagename parameter or delete images).

This mode will search the current directory for all existing imagename extensions, pick the maximum, and adds 1. For imagename='try' it will make try.psf, try_2.psf, try_3.psf, etc.

This also works if you specify a directory name in the path : imagename='outdir/try'. If './outdir' does not exist, it will create it. Then it will search for existing filenames inside that directory.

If outlier fields are specified, the incrementing happens for each of them (since each has its own 'imagename'). The counters are synchronized across imagefields, to make it easier to match up sets of output images. It adds 1 to the 'max id' from all outlier names on disk. So, if you do two runs with only the main field

(imagename='try'), and in the third run you add an outlier with imagename='outtry', you will get the following image names for the third run : 'try_3' and 'outtry_3' even though 'outry' and 'outtry_2' have not been used.

savemodel Options to save model visibilities (none, virtual, modelcolumn)

Often, model visibilities must be created and saved in the MS to be later used for self-calibration (or to just plot and view them).

none
[Do not save any model visibilities in the MS. The MS is opened] in read-only mode.

Model visibilities can be predicted in a separate step by restarting tclean with niter=0,savemodel=virtual or modelcolumn and not changing any image names so that it finds the .model on disk (or by changing imagename and setting startmodel to the original imagename).

virtual
[In the last major cycle, save the image model and state of the] gridder used during imaging within the SOURCE subtable of the MS. Images required for de-gridding will also be stored internally. All future references to model visibilities will activate the (de)gridder to compute them on-the-fly. This mode is useful when the dataset is large enough that an additional model data column on disk may be too much extra disk I/O, when the gridder is simple enough that on-the-fly recomputing of the model visibilities is quicker than disk I/O.

modelcolumn
[In the last major cycle, save predicted model visibilities] in the MODEL_DATA column of the MS. This mode is useful when the de-gridding cost to produce the model visibilities is higher than the I/O required to read the model visibilities from disk. This mode is currently required for gridder='awproject'. This mode is also required for the ability to later pull out model visibilities from the MS into a python array for custom processing.

Note 1
[The imagename.model image on disk will always be constructed] if the minor cycle runs. This savemodel parameter applies only to model visibilities created by de-gridding the model image.

Note 2
[It is possible for an MS to have both a virtual model] as well as a model_data column, but under normal operation, the last used mode will get triggered. Use the

delmod task to clear out existing models from an MS if confusion arises.

calcres Calculate initial residual image

This parameter controls what the first major cycle does.

calcres=False with niter greater than 0 will assume that a .residual image already exists and that the minor cycle can begin without recomputing it.

calcres=False with niter=0 implies that only the PSF will be made and no data will be gridded.

calcres=True requires that calcpsf=True or that the .psf and .sumwt images already exist on disk (for normalization purposes).

Usage example

[For large runs (or a pipeline scripts) it may be] useful to first run tclean with niter=0 to create an initial .residual to look at and perhaps make a custom mask for. Imaging can be resumed without recomputing it.

calcpsf Calculate PSF

This parameter controls what the first major cycle does.

calcpsf=False will assume that a .psf image already exists and that the minor cycle can begin without recomputing it.

parallel Run major cycles in parallel (this feature is experimental)

Parallel tclean will run only if casa has already been started using mpirun. Please refer to HPC documentation for details on how to start this on your system.

Example : mpirun -n 3 -xterm 0 *which casa*

Continuum Imaging :

- Data are partitioned (in time) into NProc pieces
- Gridding/iFT is done separately per partition
- Images (and weights) are gathered and then normalized
- One non-parallel minor cycle is run
- Model image is scattered to all processes
- Major cycle is done in parallel per partition

Cube Imaging :

- Data and Image coordinates are partitioned (in freq) into NProc pieces
- Each partition is processed independently (major and minor cycles)
- All processes are synchronized at major cycle boundaries for convergence checks
- At the end, cubes from all partitions are concatenated along the spectral axis

Note 1

[Iteration control for cube imaging is independent per partition.]

- **There is currently no communication between them to synchronize**
  information such as peak residual and cyclethreshold. Therefore, different chunks may trigger major cycles at different levels.
- For cube imaging in parallel, there is currently no interactive masking.

(Proper synchronization of iteration control is work in progress.)

[1;42mRETURNS[1;m void

——— examples ————————————————————————

This is the first release of our refactored imager code. Although most features have been used and validated, there are many details that have not been thoroughly tested. Feedback will be much appreciated.

Usage Examples :

(A) A suite of test programs that demo all usable modes of tclean on small test datasets https://svn.cv.nrao.edu/svn/casa/branches/release-4_5/gcwrap/python/scripts/tests/test_refimager.py (B) A set of demo examples for ALMA imaging https://casaguides.nrao.edu/index.php/TCLEAN_and_ALMA

_info_group_ = 'imaging'

_info_desc_ = 'Parallelized tclean in consecutive time steps'

__schema

__globals_()

__to_string_(*value*)

__validate_(*doc*, *schema*)

__do_inp_output(*param_prefix*, *description_str*, *formatting_chars*)

__phasecenter_dflt(*glb*)

__phasecenter(*glb*)

__projection_dflt(*glb*)

__projection(*glb*)

__vis_dflt(*glb*)

__vis(*glb*)

__imagesuffix_dflt(*glb*)

__imagesuffix(*glb*)

__parallel_dflt(*glb*)

__parallel(*glb*)

__twidth_dflt(*glb*)

__twidth(*glb*)

__datacolumn_dflt(*glb*)

__datacolumn(*glb*)

__restart_dflt(*glb*)

__restart(*glb*)

__cell_dflt(*glb*)

__cell(*glb*)

__startmodel_dflt(*glb*)

__startmodel(*glb*)

__deconvolver_dflt(*glb*)

__deconvolver(*glb*)

__calcpsf_dflt(*glb*)

__calcpsf(*glb*)

__niter_dflt(*glb*)

__niter(*glb*)

__selectdata_dflt(*glb*)

__selectdata(*glb*)

__imageprefix_dflt(*glb*)

__imageprefix(*glb*)

__imsize_dflt(*glb*)

__imsize(*glb*)

__outlierfile_dflt(*glb*)

__outlierfile(*glb*)

__calcres_dflt(*glb*)

__calcres(*glb*)

__ncpu_dflt(*glb*)

__ncpu(*glb*)

__savemodel_dflt(*glb*)

__savemodel(*glb*)

__usemask_dflt(*glb*)

__usemask(*glb*)

__specmode_dflt(*glb*)

__specmode(*glb*)

__restoration_dflt(*glb*)

__restoration(*glb*)

__stokes_dflt(*glb*)

__stokes(*glb*)

__weighting_dflt(*glb*)

__weighting(*glb*)

__gridder_dflt(*glb*)

__gridder(*glb*)

__overwrite_dflt(*glb*)

__overwrite(*glb*)

__doreg_dflt(*glb*)

__doreg(*glb*)

__chanchunks_dflt(*glb*)

__antenna_dflt(*glb*)

__smoothfactor_dflt(*glb*)

__negativethreshold_dflt(*glb*)

__minbeamfrac_dflt(*glb*)

__mask_dflt(*glb*)

__sclfactor_dflt(*glb*)

__field_dflt(*glb*)

__cutthreshold_dflt(*glb*)

__pblimit_dflt(*glb*)

__smallscalebias_dflt(*glb*)

__maxpsffraction_dflt(*glb*)

__verbose_dflt(*glb*)

__intent_dflt(*glb*)

__interpolation_dflt(*glb*)

__subregion_dflt(*glb*)

__nterms_dflt(*glb*)

__nchan_dflt(*glb*)

__reffreq_dflt(*glb*)

__conjbeams_dflt(*glb*)

__restoringbeam_dflt(*glb*)

__sidelobethreshold_dflt(*glb*)

__reftime_dflt(*glb*)

__cycleniter_dflt(*glb*)

__minpsffraction_dflt(*glb*)

__scan_dflt(*glb*)

__computepastep_dflt(*glb*)

__minpercentchange_dflt(*glb*)

__wbawp_dflt(*glb*)

__docompress_dflt(*glb*)

__interactive_dflt(*glb*)

__npixels_dflt(*glb*)

__mosweight_dflt(*glb*)

__pbcor_dflt(*glb*)

__normtype_dflt(*glb*)

__uvtaper_dflt(*glb*)

__cyclefactor_dflt(*glb*)

__toTb_dflt(*glb*)

__restfreq_dflt(*glb*)

__pbmask_dflt(*glb*)

__growiterations_dflt(*glb*)

__gain_dflt(*glb*)

__scales_dflt(*glb*)

__robust_dflt(*glb*)

__vptable_dflt(*glb*)

__aterm_dflt(*glb*)

__usephacenter_dflt(*glb*)

__usepointing_dflt(*glb*)

__rotatepastep_dflt(*glb*)

__threshold_dflt(*glb*)

__veltype_dflt(*glb*)

__outframe_dflt(*glb*)

__dogrowprune_dflt(*glb*)

__uvrange_dflt(*glb*)

__psterm_dflt(*glb*)

__start_dflt(*glb*)

__observation_dflt(*glb*)

__lownoisethreshold_dflt(*glb*)

__facets_dflt(*glb*)

__noisethreshold_dflt(*glb*)

__width_dflt(*glb*)

__spw_dflt(*glb*)

__timerange_dflt(*glb*)

__nsigma_dflt(*glb*)

__cfcache_dflt(*glb*)

__wprojplanes_dflt(*glb*)

__usephacenter(*glb*)

__reftime(*glb*)

__toTb(*glb*)

__sclfactor(*glb*)

__subregion(*glb*)

__docompress(*glb*)

__field(*glb*)

__spw(*glb*)

__timerange(*glb*)

__uvrange(*glb*)

__antenna(*glb*)

__scan(*glb*)

__observation(*glb*)

__intent(*glb*)

__reffreq(*glb*)

__nchan(*glb*)

__start(*glb*)

__width(*glb*)

__outframe(*glb*)

__veltype(*glb*)

__restfreq(*glb*)

__interpolation(*glb*)

__facets(*glb*)

__chanchunks(*glb*)

__wprojplanes(*glb*)

__vptable(*glb*)

__usepointing(*glb*)

__mosweight(*glb*)

__aterm(*glb*)

__psterm(*glb*)

__wbawp(*glb*)

__conjbeams(*glb*)

__cfcache(*glb*)

__computepastep(*glb*)

__rotatepastep(*glb*)

__pblimit(*glb*)

__normtype(*glb*)

__scales(*glb*)

__nterms(*glb*)

__smallscalebias(*glb*)

__restoringbeam(*glb*)

__pbcor(*glb*)

__robust(*glb*)

__npixels(*glb*)

__uvtaper(*glb*)

__gain(*glb*)

__threshold(*glb*)

__nsigma(*glb*)

__cycleniter(*glb*)

__cyclefactor(*glb*)

__minpsffraction(*glb*)

__maxpsffraction(*glb*)

__interactive(*glb*)

__mask(*glb*)

__pbmask(*glb*)

__sidelobethreshold(*glb*)

__noisethreshold(*glb*)

__lownoisethreshold(*glb*)

__negativethreshold(*glb*)

__smoothfactor(*glb*)

__minbeamfrac(*glb*)

__cutthreshold(*glb*)

__growiterations(*glb*)

__dogrowprune(*glb*)

__minpercentchange(*glb*)

__verbose(*glb*)

__vis_inp()

__imageprefix_inp()

__imagesuffix_inp()

__ncpu_inp()

__twidth_inp()

__doreg_inp()

__usephacenter_inp()

__reftime_inp()

__toTb_inp()

__sclfactor_inp()

__subregion_inp()

__docompress_inp()

__overwrite_inp()

__selectdata_inp()

__field_inp()

__spw_inp()

__timerange_inp()

__uvrange_inp()

__antenna_inp()

__scan_inp()

__observation_inp()

__intent_inp()

__datacolumn_inp()

__imsize_inp()

__cell_inp()

__phasecenter_inp()

__stokes_inp()

__projection_inp()

__startmodel_inp()

__specmode_inp()

__reffreq_inp()

__nchan_inp()

__start_inp()

__width_inp()

__outframe_inp()

__veltype_inp()

__restfreq_inp()

__interpolation_inp()

__gridder_inp()

__facets_inp()

__chanchunks_inp()

__wprojplanes_inp()

__vptable_inp()

__usepointing_inp()

__mosweight_inp()

__aterm_inp()

__psterm_inp()

__wbawp_inp()

__conjbeams_inp()

__cfcache_inp()

__computepastep_inp()

__rotatepastep_inp()

__pblimit_inp()

__normtype_inp()

__deconvolver_inp()

__scales_inp()

__nterms_inp()

__smallscalebias_inp()

__restoration_inp()

__restoringbeam_inp()

__pbcor_inp()

__outlierfile_inp()

__weighting_inp()

__robust_inp()

__npixels_inp()

__uvtaper_inp()

__niter_inp()

__gain_inp()

__threshold_inp()

__nsigma_inp()

__cycleniter_inp()

__cyclefactor_inp()

__minpsffraction_inp()

__maxpsffraction_inp()

__interactive_inp()

__usemask_inp()

__mask_inp()

__pbmask_inp()

__sidelobethreshold_inp()

__noisethreshold_inp()

__lownoisethreshold_inp()

__negativethreshold_inp()

__smoothfactor_inp()

__minbeamfrac_inp()

__cutthreshold_inp()

__growiterations_inp()

__dogrowprune_inp()

__minpercentchange_inp()

__verbose_inp()

__restart_inp()

__savemodel_inp()

__calcres_inp()

__calcpsf_inp()

__parallel_inp()

set_global_defaults()

inp()

tget(*file=None*)

__call__(*vis=None*, *imageprefix=None*, *imagesuffix=None*, *ncpu=None*, *twidth=None*, *doreg=None*, *usephacenter=None*, *reftime=None*, *toTb=None*, *sclfactor=None*, *subregion=None*, *docompress=None*, *overwrite=None*, *selectdata=None*, *field=None*, *spw=None*, *timerange=None*, *uvrange=None*, *antenna=None*, *scan=None*, *observation=None*, *intent=None*, *datacolumn=None*, *imsize=None*, *cell=None*, *phasecenter=None*, *stokes=None*, *projection=None*, *startmodel=None*, *specmode=None*, *reffreq=None*, *nchan=None*, *start=None*, *width=None*, *outframe=None*, *veltype=None*, *restfreq=None*, *interpolation=None*, *gridder=None*, *facets=None*, *chanchunks=None*, *wprojplanes=None*, *vptable=None*, *usepointing=None*, *mosweight=None*, *aterm=None*, *psterm=None*, *wbawp=None*, *conjbeams=None*, *cfcache=None*, *computepastep=None*, *rotatepastep=None*, *pblimit=None*, *normtype=None*, *deconvolver=None*, *scales=None*, *nterms=None*, *smallscalebias=None*, *restoration=None*, *restoringbeam=None*, *pbcor=None*, *outlierfile=None*, *weighting=None*, *robust=None*, *npixels=None*, *uvtaper=None*, *niter=None*, *gain=None*, *threshold=None*, *nsigma=None*, *cycleniter=None*, *cyclefactor=None*, *minpsffraction=None*, *maxpsffraction=None*, *interactive=None*, *usemask=None*, *mask=None*, *pbmask=None*, *sidelobethreshold=None*, *noisethreshold=None*, *lownoisethreshold=None*, *negativethreshold=None*, *smoothfactor=None*, *minbeamfrac=None*, *cutthreshold=None*, *growiterations=None*, *dogrowprune=None*, *minpercentchange=None*, *verbose=None*, *restart=None*, *savemodel=None*, *calcres=None*, *calcpsf=None*, *parallel=None*)

ptclean.**ptclean**

1.1.7 importeovsa

Module Contents

Classes

_importeovsa	importeovsa ---- Parallelized import EOVSA idb file(s) to a measurement set or multiple measurement set.

Functions

static_var(varname, value)

Attributes

importeovsa

importeovsa.**static_var**(*varname*, *value*)

class importeovsa.**_importeovsa**

importeovsa —- Parallelized import EOVSA idb file(s) to a measurement set or multiple measurement set.

Parallelized imports an arbitrary number of EOVSA idb-format data sets into a casa measurement set. If more than one band is present, they will be put in the same measurement set but in a separate spectral window.

——— parameter descriptions ————————————————

idbfiles Name of input EOVSA idb file(s) or observation time range. ncpu Number of cpu cores to use timebin Bin width for time averaging width Width of output channel relative to MS channel (# to average) visprefix Prefix of vis names (may include the path). udb_corr if applying correction to input UDB files before import to MS. nocreatms If setting nocreatms True, will simulate a model measurement set for the first idb file and copy the model for the rest of idl files in list. If False, will simulate a new measurement set for every idbfile in list. doconcat If concatenate multi casa measurement sets to one file. modelms Name of input model measurement set file. If modelms is assigned, no simulation will start. doscaling If creating a new MS file with the amplitude of visibility data rescaled. keep_nsclms Keep the no scaling measurement sets use_exist_udbcorr If use the existed udb_corr results.

——— examples ————————————————————————

Parallelized imports an arbitrary number of EOVSA idb-format data sets into a casa measurement set. If more than one band is present, they will be put in the same measurement set but in a separate spectral window.

Detailed Keyword arguments:

idbfiles – Name of input EOVSA idb file(s) default: none. Must be supplied example: idbfiles = ‘IDB20160524000518’ example: idbfiles=[‘IDB20160524000518’,’IDB20160524000528’]

ncpu – Number of cpu cores to use default: 8

visprefix – Prefix of vis names (may include the path) default: none; example: visprefix='sun/']

— Data Selection —

nocreatms – If copying a new MS file instead of create one from MS simulator. default: False

modelms – Name of the standard Measurement Set. IF modelms is not provided, use '/home/user/sjyu/20160531/ms/sun/SUN/SUN_20160531T142234-10m.1s.ms' as a standard MS.

doconcat – If outputing one single MS file

— Channel averaging parameter —

width – Number of input channels to average to create an output channel. If a list is given, each bin will apply to one spw in the selection. default: 1 => no channel averaging. options: (int) or [int]

example: chanbin=[2,3] => average 2 channels of 1st selected spectral window and 3 in the second one.

— Time averaging parameters —

timebin – Bin width for time averaging. When timebin is greater than 0s, the task will average data in time. Flagged data will be included in the average calculation, unless the parameter keepflags is set to False. In this case only partially flagged rows will be used in the average. default: '0s'

_info_group_ = 'Import/export'

_info_desc_ = 'Parallelized import EOVSA idb file(s) to a measurement set or multiple measurement set.'

__schema

__globals_()

__to_string_(*value*)

__validate_(*doc*, *schema*)

__do_inp_output(*param_prefix*, *description_str*, *formatting_chars*)

__width_dflt(*glb*)

__width(*glb*)

__idbfiles_dflt(*glb*)

__idbfiles(*glb*)

__doscaling_dflt(*glb*)

__doscaling(*glb*)

__ncpu_dflt(*glb*)

__ncpu(*glb*)

__modelms_dflt(*glb*)

__modelms(*glb*)

__visprefix_dflt(*glb*)

__visprefix(*glb*)

__timebin_dflt(*glb*)

__timebin(*glb*)

__udb_corr_dflt(*glb*)

__udb_corr(*glb*)

__nocreatms_dflt(*glb*)

__nocreatms(*glb*)

__doconcat_dflt(*glb*)

__doconcat(*glb*)

__use_exist_udbcorr(*glb*)

__use_exist_udbcorr_dflt(*glb*)

__keep_nsclms_dflt(*glb*)

__keep_nsclms(*glb*)

__idbfiles_inp()

__ncpu_inp()

__timebin_inp()

__width_inp()

__visprefix_inp()

__udb_corr_inp()

__nocreatms_inp()

__doconcat_inp()

__modelms_inp()

__doscaling_inp()

__keep_nsclms_inp()

__use_exist_udbcorr_inp()

set_global_defaults()

inp()

tget(*savefile=None*)

tput(*outfile=None*)

__call__(*idbfiles=None*, *ncpu=None*, *timebin=None*, *width=None*, *visprefix=None*, *udb_corr=None*, *nocreatms=None*, *doconcat=None*, *modelms=None*, *doscaling=None*, *keep_nsclms=None*, *use_exist_udbcorr=None*)

importeovsa.**importeovsa**

1.1.8 ptclean6

Module Contents

Classes

_ptclean6	ptclean6 ---- Parallelized tclean in consecutive time steps

Functions

static_var(varname, value)

Attributes

ptclean6

ptclean6.**static_var**(*varname*, *value*)

class ptclean6.**_ptclean6**

ptclean6 —- Parallelized tclean in consecutive time steps

Parallelized clean in consecutive time steps. Packed over CASA 6 tclean.

——— parameter descriptions ———————————————

vis Name(s) of input visibility file(s)

default: none; example: vis='ngc5921.ms'

vis=['ngc5921a.ms','ngc5921b.ms']; multiple MSes

imageprefix Prefix of output image names (usually useful in defining the output path) imagesuffix Suffix of output image names (usually useful in specifyting the image type, version, etc.) ncpu Number of cpu cores to use twidth Number of time pixels to average doreg True if use vla_prep to register the image usephacenter True if use the phacenter information from the measurement set (e.g., VLA); False to assume the phase center is at the solar disk center (EOVSA) reftime Reference time of the J2000 coordinates associated with the ephemeris target. e.g., "2012/03/03/12:00". This is used for helioimage2fits.py to find the solar x y offset in order to register the image. If not set, use the actual timerange of the image (default) toTb True if convert to brightness temperature sclfactor scale the brightness temperature up by its value subregion The name of a CASA region string

The name of a CASA image or region file or region string. Only locations within the region will output to the fits file. If regions specified fall completely outside of the image, ptclean6 will throw an error.

Manual mask options/examples :

subregion='box[[224pix,224pix],[288pix,288pix]]' : A CASA region string.

docompress True if compress the output fits files overwrite True if overwrite the image selectdata Enable data selection parameters. field to image or mosaic. Use field id(s) or name(s).

['go listobs' to obtain the list id's or names]

default: ''= all fields
If field string is a non-negative integer, it is assumed to be a field index otherwise, it is assumed to be a field name field='0~2'; field ids 0,1,2 field='0,4,5~7'; field ids 0,4,5,6,7 field='3C286,3C295'; field named 3C286 and 3C295 field = '3,4C*'; field id 3, all names starting with 4C For multiple MS input, a list of field strings can be used: field = ['0~2','0~4']; field ids 0-2 for the first MS and 0-4

for the second

field = '0~2'; field ids 0-2 for all input MSes

spw l window/channels

NOTE: channels de-selected here will contain all zeros if
selected by the parameter mode subparameters.

default: ''=all spectral windows and channels
spw='0~2,4'; spectral windows 0,1,2,4 (all channels) spw='0:5~61'; spw 0, channels 5 to 61 spw='<2'; spectral windows less than 2 (i.e. 0,1) spw='0,10,3:3~45'; spw 0,10 all channels, spw 3,

channels 3 to 45.

spw='0~2:2~6'; spw 0,1,2 with channels 2 through 6 in each. For multiple MS input, a list of spw strings can be used: spw=['0','0~3']; spw ids 0 for the first MS and 0-3 for the second spw='0~3' spw ids 0-3 for all input MS spw='3:10~20;50~60' for multiple channel ranges within spw id 3 spw='3:10~20;50~60,4:0~30' for different channel ranges for spw ids 3 and 4 spw='0:0~10,1:20~30,2:1;2;3'; spw 0, channels 0-10,

spw 1, channels 20-30, and spw 2, channels, 1,2 and 3

spw='1~4;6:15~48' for channels 15 through 48 for spw ids 1,2,3,4 and 6

timerange Range of time to select from data

default: '' (all); examples, timerange = 'YYYY/MM/DD/hh:mm:ss~YYYY/MM/DD/hh:mm:ss' Note: if YYYY/MM/DD is missing date defaults to first

day in data set

timerange='09:14:0~09:54:0' picks 40 min on first day timerange='25:00:00~27:30:00' picks 1 hr to 3 hr

30min on NEXT day

timerange='09:44:00' pick data within one integration
of time

timerange='> 10:24:00' data after this time For multiple MS input, a list of timerange strings can be used: timerange=['09:14:0~09:54:0','> 10:24:00'] timerange='09:14:0~09:54:0"; apply the same timerange for

all input MSes

uvrange Select data within uvrange (default unit is meters)
default: '' (all); example: uvrange='0~1000klambda'; uvrange from 0-1000 kilo-lambda uvrange='> 4klambda';uvranges greater than 4 kilo lambda For multiple MS input, a list of uvrange strings can

be used: uvrange=['0~1000klambda','100~1000klamda'] uvrange='0~1000klambda'; apply 0-1000 kilolambda for all

input MSes

antenna Select data based on antenna/baseline

default: '' (all) If antenna string is a non-negative integer, it is assumed to be an antenna index, otherwise, it is considered an antenna name. antenna='5&6'; baseline between antenna index 5 and

index 6.

antenna='VA05&VA06'; baseline between VLA antenna 5
and 6.

antenna='5&6;7&8'; baselines 5-6 and 7-8 antenna='5'; all baselines with antenna index 5 antenna='05'; all baselines with antenna number 05

(VLA old name)

antenna='5,6,9'; all baselines with antennas 5,6,9
index number

For multiple MS input, a list of antenna strings can be used: antenna=['5','5&6']; antenna='5'; antenna index 5 for all input MSes antenna='!DV14'; use all antennas except DV14

scan Scan number range

default: '' (all) example: scan='1~5' For multiple MS input, a list of scan strings can be used: scan=['0~100','10~200'] scan='0~100; scan ids 0-100 for all input MSes

observation Observation ID range
default: '' (all) example: observation='1~5'

intent Scan Intent(s)

default: '' (all) example: intent='TARGET_SOURCE' example: intent='TARGET_SOURCE1,TARGET_SOURCE2' example: intent='TARGET_POINTING*'

datacolumn Data column to image (data or observed, corrected)
default:'corrected' (If 'corrected' does not exist, it will use 'data' instead)

imagename Pre-name of output images

example : imagename='try'

Output images will be (a subset of) :

try.psf - Point spread function try.residual - Residual image try.image - Restored image try.model - Model image (contains only flux components) try.sumwt - Single pixel image containing sum-of-weights.

(for natural weighting, sensitivity=1/sqrt(sumwt))

try.pb - Primary beam model (values depend on the gridder used)

Widefield projection algorithms (gridder=mosaic,awproject) will compute the following images too. try.weight - FT of gridded weights or the

un-normalized sum of PB-square (for all pointings) Here, PB = sqrt(weight) normalized to a maximum of 1.0

For multi-term wideband imaging, all relevant images above will have additional .tt0,.tt1, etc suffixes to indicate Taylor terms, plus the following extra output images. try.alpha - spectral index try.alpha.error - estimate of error on spectral index try.beta - spectral curvature (if nterms > 2)

Tip

[Include a directory name in 'imagename' for all] output images to be sent there instead of the current working directory : imagename='mydir/try'

Tip

[Restarting an imaging run without changing 'imagename']

implies continuation from the existing model image on disk.

- If 'startmodel' was initially specified it needs to be set to "" for the restart run (or tclean will exit with an error message).
- By default, the residual image and psf will be recomputed but if no changes were made to relevant parameters between the runs, set calcres=False, calcpsf=False to resume directly from the minor cycle without the (unnecessary) first major cycle.

To automatically change 'imagename' with a numerical increment, set restart=False (see tclean docs for 'restart').

Note

[All imaging runs will by default produce restored images.] For a niter=0 run, this will be redundant and can optionally be turned off via the 'restoration=T/F' parameter.

imsize Number of pixels

example

[imsize = [350,250]] imsize = 500 is equivalent to [500,500]

To take proper advantage of internal optimized FFT routines, the number of pixels must be even and factorizable by 2,3,5,7 only.

cell Cell size

example: cell=['0.5arcsec,'0.5arcsec'] or cell=['1arcmin', '1arcmin'] cell = '1arcsec' is equivalent to ['1arcsec','1arcsec']

phasecenter Phase center of the image (string or field id); if the phasecenter is the name known major solar system object ('MERCURY', 'VENUS', 'MARS', 'JUPITER', 'SATURN', 'URANUS', 'NEPTUNE', 'PLUTO', 'SUN', 'MOON') or is an ephemerides table then that source is tracked and the background sources get smeared. There is a special case, when phasecenter='TRACKFIELD', which will use the ephemerides or polynomial phasecenter in the FIELD table of the MS's as the source center to track.

example: phasecenter=6

phasecenter='J2000 19h30m00 -40d00m00' phasecenter='J2000 292.5deg -40.0deg' phasecenter='J2000 5.105rad -0.698rad' phasecenter='ICRS 13:05:27.2780 -049.28.04.458' phasecenter='myComet_ephem.tab' phasecenter='MOON' phasecenter='TRACKFIELD'

stokes Stokes Planes to make

default='I'; example: stokes='IQUV';

Options: 'I','Q','U','V','IV','QU','IQ','UV','IQUV','RR','LL','XX','YY','RRLL','XXYY','pseudoI'

Note

[Due to current internal code constraints, if any correlation pair] is flagged, by default, no data for that row in the MS will be used. So, in an MS with XX,YY, if only YY is

flagged, neither a Stokes I image nor an XX image can be made from those data points. In such a situation, please split out only the unflagged correlation into a separate MS.

Note
[The 'pseudoI' option is a partial solution, allowing Stokes I imaging] when either of the parallel-hand correlations are unflagged.

The remaining constraints shall be removed (where logical) in a future release.

projection Coordinate projection
Examples : SIN, NCP A list of supported (but untested) projections can be found here : http://casa.nrao.edu/active/docs/doxygen/html/classcasa_1_1Projection.html#a3d5f9ec787e4eabdce57ab5edaf7c0cd

startmodel Name of starting model image

The contents of the supplied starting model image will be copied to the imagename.model before the run begins.

example : startmodel = 'singledish.im'

For deconvolver='mtmfs', one image per Taylor term must be provided. example : startmodel = ['try.model.tt0', 'try.model.tt1']

startmodel = ['try.model.tt0'] will use a starting model only
for the zeroth order term.

startmodel = ['','try.model.tt1'] will use a starting model only
for the first order term.

This starting model can be of a different image shape and size from what is currently being imaged. If so, an image regrid is first triggered to resample the input image onto the target coordinate system.

A common usage is to set this parameter equal to a single dish image

Negative components in the model image will be included as is.

[Note
[If an error occurs during image resampling/regridding,] please try using task imregrid to resample the starting model image onto a CASA image with the target shape and coordinate system before supplying it via startmodel]

specmode Spectral definition mode (mfs,cube,cubedata, cubesource)

mode='mfs'
[Continuum imaging with only one output image channel.] (mode='cont' can also be used here)

mode='cube'
[Spectral line imaging with one or more channels]

Parameters start, width,and nchan define the spectral coordinate system and can be specified either in terms of channel numbers, frequency or velocity in whatever spectral frame is specified in 'outframe'. All internal and output images are made with outframe as the base spectral frame. However imaging code internally uses the fixed spectral frame, LSRK for automatic internal software Doppler tracking so that a spectral line observed over an extended time range will line up appropriately. Therefore the output images have additional spectral frame conversion layer in LSRK on the top the base frame.

(Note

[Even if the input parameters are specified in a frame] other than LSRK, the viewer still displays spectral axis in LSRK by default because of the conversion frame layer mentioned above. The viewer can be used to relabel the spectral axis in any desired frame - via the spectral reference option under axis label properties in the data display options window.)

mode='cubedata'

[Spectral line imaging with one or more channels] There is no internal software Doppler tracking so a spectral line observed over an extended time range may be smeared out in frequency. There is strictly no valid spectral frame with which to label the output images, but they will list the frame defined in the MS.

mode='cubesource': Spectral line imaging while tracking moving source (near field or solar system objects). The velocity of the source is accounted and the frequency reported is in the source frame. As there is not SOURCE frame defined, the frame reported will be REST (as it may not be in the rest frame emission region may be moving w.r.t the systemic velocity frame)

reffreq Reference frequency of the output image coordinate system

Example : reffreq='1.5GHz' as a string with units.

By default, it is calculated as the middle of the selected frequency range.

For deconvolver='mtmfs' the Taylor expansion is also done about this specified reference frequency.

nchan Number of channels in the output image

For default (=-1), the number of channels will be automatically determined based on data selected by 'spw' with 'start' and 'width'. It is often easiest to leave nchan at the default value. example: nchan=100

start First channel (e.g. start=3,start='1.1GHz',start='15343km/s')

of output cube images specified by data channel number (integer), velocity (string with a unit), or frequency (string with a unit). Default:''; The first channel is automatically determined based on the 'spw' channel selection and 'width'. When the channel number is used along with the channel selection

in 'spw' (e.g. spw='0:6~100'),

'start' channel number is RELATIVE (zero-based) to the selected channels in 'spw'. So for the above example, start=1 means that the first image channel is the second selected data channel, which is channel 7. For specmode='cube', when velocity or frequency is used it is interpreted with the frame defined in outframe. [The parameters of the desired output cube can be estimated by using the 'transform' functionality of 'plotms'] examples: start='5.0km/s'; 1st channel, 5.0km/s in outframe

start='22.3GHz'; 1st channel, 22.3GHz in outframe

width Channel width (e.g. width=2,width='0.1MHz',width='10km/s') of output cube images

specified by data channel number (integer), velocity (string with a unit), or or frequency (string with a unit). Default:''; data channel width The sign of width defines the direction of the channels to be incremented. For width specified in velocity or frequency with '-' in front gives image channels in decreasing velocity or frequency, respectively. For specmode='cube', when velocity or frequency is used it is interpreted with the reference frame defined in outframe. examples: width='2.0km/s'; results in channels with increasing velocity

width='-2.0km/s'; results in channels with decreasing velocity width='40kHz'; results in channels with increasing frequency width=-2; results in channels averaged of 2 data channels incremented from

high to low channel numbers

outframe Spectral reference frame in which to interpret 'start' and 'width'

Options: '','LSRK','LSRD','BARY','GEO','TOPO','GALACTO','LGROUP','CMB' example: outframe='bary' for Barycentric frame

REST – Rest frequency LSRD – Local Standard of Rest (J2000)

– as the dynamical definition (IAU, [9,12,7] km/s in galactic coordinates)

LSRK – LSR as a kinematical (radio) definition
– 20.0 km/s in direction ra,dec = [270,+30] deg (B1900.0)

BARY – Barycentric (J2000) GEO — Geocentric TOPO – Topocentric GALACTO – Galacto centric (with rotation of 220 km/s in direction l,b = [90,0] deg. LGROUP – Local group velocity – 308km/s towards l,b = [105,-7] deg (F. Ghigo)

CMB – CMB velocity – 369.5km/s towards l,b = [264.4, 48.4] deg (F. Ghigo) DEFAULT = LSRK

veltype Velocity type (radio, z, ratio, beta, gamma, optical)
For start and/or width specified in velocity, specifies the velocity definition Options: 'radio','optical','z','beta','gamma','optical' NOTE: the viewer always defaults to displaying the 'radio' frame,

but that can be changed in the position tracking pull down.

The different types (with F = f/f0, the frequency ratio), are:

Z = (-1 + 1/F)

RATIO = (F) * RADIO = (1 - F) OPTICAL == Z BETA = ((1 - F2)/(1 + F2)) GAMMA = ((1 + F2)/2F) * RELATIVISTIC == BETA (== v/c) DEFAULT == RADIO Note that the ones with an '*' have no real interpretation (although the calculation will proceed) if given as a velocity.

restfreq List of rest frequencies or a rest frequency in a string.
Specify rest frequency to use for output image. *Currently it uses the first rest frequency in the list for translation of velocities. The list will be stored in the output images. Default: []; look for the rest frequency stored in the MS, if not available, use center frequency of the selected channels examples: restfreq=['1.42GHz']

restfreq='1.42GHz'

interpolation Spectral interpolation (nearest,linear,cubic)

Interpolation rules to use when binning data channels onto image channels and evaluating visibility values at the centers of image channels.

Note
['linear' and 'cubic' interpolation requires data points on both sides of] each image frequency. Errors are therefore possible at edge channels, or near flagged data channels. When image channel width is much larger than the data channel width there is nothing much to be gained using linear or cubic thus not worth the extra computation involved.

perchanweightdensity When calculating weight density for Briggs
style weighting in a cube, this parameter determines whether to calculate the weight density for each channel independently (the default, True) or a common weight density for all of the selected data. This parameter has no meaning for continuum (specmode='mfs') imaging or for natural and radial weighting schemes. For cube imaging perchanweightdensity=True is a recommended option that provides more uniform sensitivity per channel for cubes, but with generally larger psfs than the perchanweightdensity=False (prior behavior) option. When using Briggs style weight with perchanweightdensity=True, the imaging weight density calculations use only the weights of data that contribute specifically to that channel. On the other hand, when perchanweightdensity=False, the imaging weight density calculations sum all of the

weights from all of the data channels selected whose (u,v) falls in a given uv cell on the weight density grid. Since the aggregated weights, in any given uv cell, will change depending on the number of channels included when imaging, the psf calculated for a given frequency channel will also necessarily change, resulting in variability in the psf for a given frequency channel when perchanweightdensity=False. In general, perchanweightdensity=False results in smaller psfs for the same value of robustness compared to perchanweightdensity=True, but the rms noise as a function of channel varies and increases toward the edge channels; perchanweightdensity=True provides more uniform sensitivity per channel for cubes. This may make it harder to find estimates of continuum when perchanweightdensity=False. If you intend to image a large cube in many smaller subcubes and subsequently concatenate, it is advisable to use perchanweightdensity=True to avoid surprisingly varying sensitivity and psfs across the concatenated cube.

gridder Gridding options (standard, wproject, widefield, mosaic, awproject)

The following options choose different gridding convolution functions for the process of convolutional resampling of the measured visibilities onto a regular uv-grid prior to an inverse FFT. Model prediction (degridding) also uses these same functions. Several wide-field effects can be accounted for via careful choices of convolution functions. Gridding (degridding) runtime will rise in proportion to the support size of these convolution functions (in uv-pixels).

standard : Prolate Spheroid with 7x7 uv pixel support size

[This mode can also be invoked using 'ft' or 'gridft']

wproject

[W-Projection algorithm to correct for the widefield]

non-coplanar baseline effect. [Cornwell et.al 2008]

wprojplanes is the number of distinct w-values at which to compute and use different gridding convolution functions (see help for wprojplanes).

Convolution function support size can range

from 5x5 to few 100 x few 100.

[This mode can also be invoked using 'wprojectft']

widefield : Facetted imaging with or without W-Projection per facet.

A set of facets x facets subregions of the specified image are gridded separately using their respective phase centers (to minimize max W). Deconvolution is done on the joint full size image, using a PSF from the first subregion.

wprojplanes=1 : standard prolate spheroid gridder per facet. wprojplanes > 1 : W-Projection gridder per facet. nfacets=1, wprojplanes > 1 : Pure W-Projection and no facetting nfacets=1, wprojplanes=1 : Same as standard,ft,gridft

A combination of facetting and W-Projection is relevant only for very large fields of view. (In our current version of tclean, this

combination runs only with parallel=False.

mosaic

[A-Projection with azimuthally symmetric beams without]

sidelobes, beam rotation or squint correction. Gridding convolution functions per visibility are computed from FTs of PB models per antenna. This gridder can be run on single fields as well as mosaics.

VLA : PB polynomial fit model (Napier and Rots, 1982) EVLA : PB polynomial fit model (Perley, 2015) ALMA : Airy disks for a 10.7m dish (for 12m dishes) and

6.25m dish (for 7m dishes) each with 0.75m blockages (Hunter/Brogan 2011). Joint mosaic imaging supports heterogeneous arrays for ALMA.

Typical gridding convolution function support sizes are between 7 and 50 depending on the desired accuracy (given by the uv cell size or image field of view).

[This mode can also be invoked using 'mosaicft' or 'ftmosaic']

awproject
[A-Projection with azimuthally asymmetric beams and]

including beam rotation, squint correction, conjugate frequency beams and W-projection. [Bhatnagar et.al, 2008]

Gridding convolution functions are computed from aperture illumination models per antenna and optionally combined with W-Projection kernels and a prolate spheroid. This gridder can be run on single fields as well as mosaics.

VLA
[Uses ray traced model (VLA and EVLA) including feed] leg and subreflector shadows, off-axis feed location (for beam squint and other polarization effects), and a Gaussian fit for the feed beams (Ref: Brisken 2009)

ALMA
[Similar ray-traced model as above (but the correctness] of its polarization properties remains un-verified).

Typical gridding convolution function support sizes are between 7 and 50 depending on the desired accuracy (given by the uv cell size or image field of view). When combined with W-Projection they can be significantly larger.

[This mode can also be invoked using 'awprojectft']

imagemosaic
[(untested implementation)] Grid and iFT each pointing separately and combine the images as a linear mosaic (weighted by a PB model) in the image domain before a joint minor cycle.

VLA/ALMA PB models are same as for gridder='mosaicft'

—— Notes on PB models :

(1) **Several different sources of PB models are used in the modes**
listed above. This is partly for reasons of algorithmic flexibility and partly due to the current lack of a common beam model repository or consensus on what beam models are most appropriate.

(2) **For ALMA and gridder='mosaic', ray-traced (TICRA) beams** are also available via the vpmanager tool. For example, call the following before the tclean run.

vp.setpbimage(telescope="ALMA", compleximage='/home/casa/data/trunk/alma/responses/ALMA_0_DV__0_0_ antnames=['DV'+'%02d'%k for k in range(25)]) vp.saveastable('mypb.tab') Then, supply vptable='mypb.tab' to tclean. (Currently this will work only for non-parallel runs)

—— Note on PB masks :

In tclean, A-Projection gridders (mosaic and awproject) produce a .pb image and use the 'pblimit' subparameter to decide normalization cutoffs and construct an internal T/F mask in the .pb and .image images. However, this T/F mask cannot directly be used during deconvolution (which needs a 1/0 mask). There are two options for making a pb based deconvolution mask.

– Run tclean with niter=0 to produce the .pb, construct a 1/0 image

with the desired threshold (using ia.open('newmask.im'); ia.calc('iif("xxx.pb">0.3,1.0,0.0)');ia.close() for example), and supply it via the 'mask' parameter in a subsequent run (with calcres=F and calcpsf=F to restart directly from the minor cycle).

– Run tclean with usemask='pb' for it to automatically construct

a 1/0 mask from the internal T/F mask from .pb at a fixed 0.2 threshold.

—— Making PBs for gridders other than mosaic,awproject

After the PSF generation, a PB is constructed using the same models used in gridder='mosaic' but just evaluated in the image domain without consideration to weights.

facets Number of facets on a side

A set of (facets x facets) subregions of the specified image are gridded separately using their respective phase centers (to minimize max W). Deconvolution is done on the joint full size image, using a PSF from the first subregion/facet.

In our current version of tclean, facets>1 may be used only with parallel=False.

psfphasecenter For mosaic use psf centered on this
optional direction. You may need to use this if for example the mosaic does not have any pointing in the center of the image. Another reason; as the psf is approximate for a mosaic, this may help to deconvolve a non central bright source well and quickly.

example:

psfphasecenter=6 #center psf on field 6 psfphasecenter='J2000 19h30m00 -40d00m00' psfphasecenter='J2000 292.5deg -40.0deg' psfphasecenter='J2000 5.105rad -0.698rad' psfphasecenter='ICRS 13:05:27.2780 -049.28.04.458'

wprojplanes Number of distinct w-values at which to compute and use different
gridding convolution functions for W-Projection

An appropriate value of wprojplanes depends on the presence/absence of a bright source far from the phase center, the desired dynamic range of an image in the presence of a bright far out source, the maximum w-value in the measurements, and the desired trade off between accuracy and computing cost.

As a (rough) guide, VLA L-Band D-config may require a value of 128 for a source 30arcmin away from the phase center. A-config may require 1024 or more. To converge to an appropriate value, try starting with 128 and then increasing it if artifacts persist. W-term artifacts (for the VLA) typically look like arc-shaped

smears in a synthesis image or a shift in source position between images made at different times. These artifacts are more pronounced the further the source is from the phase center.

There is no harm in simply always choosing a large value (say, 1024) but there will be a significant performance cost to doing so, especially for gridder='awproject' where it is combined with A-Projection.

wprojplanes=-1 is an option for gridder='widefield' or 'wproject' in which the number of planes is automatically computed.

vptable vpmanager

vptable=""
[Choose default beams for different telescopes] ALMA : Airy disks EVLA : old VLA models.

Other primary beam models can be chosen via the vpmanager tool.

Step 1 : Set up the vpmanager tool and save its state in a table

vp.setpbpoly(telescope='EVLA', coeff=[1.0, -1.529e-3, 8.69e-7, -1.88e-10])
vp.saveastable('myvp.tab')

Step 2 : Supply the name of that table in tclean.

tclean(....., vptable='myvp.tab',....)

Please see the documentation for the vpmanager for more details on how to choose different beam models. Work is in progress to update the defaults for EVLA and ALMA.

Note
[AWProjection currently does not use this mechanism to choose] beam models. It instead uses ray-traced beams computed from parameterized aperture illumination functions, which are not available via the vpmanager. So, gridder='awproject' does not allow the user to set this parameter.

mosweight When doing Brigg's style weighting (including uniform) to perform the weight density calculation for each field indepedently if True. If False the weight density is calculated from the average uv distribution of all the fields. aterm Use aperture illumination functions during gridding

This parameter turns on the A-term of the AW-Projection gridder. Gridding convolution functions are constructed from aperture illumination function models of each antenna.

psterm Include the Prolate Spheroidal (PS) funtion as the anti-aliasing
operator in the gridding convolution functions used for gridding.

Setting this parameter to true is necessary when aterm is set to false. It can be set to false when aterm is set to true, though with this setting effects of aliasing may be there in the image, particularly near the edges.

When set to true, the .pb images will contain the fourier transform of the of the PS funtion. The table below enumarates the functional effects of the psterm, aterm and wprojplanes settings. PB referes to the Primary Beam and FT() refers to the Fourier transform operation.

AW-Projection True True >1 FT(PS) x PB
False PB

A-Projection True True 1 FT(PS) x PB
False PB

W-Projection False True >1 FT(PS)

Standard False True 1 FT(PS)

wbawp Use frequency dependent A-terms
Scale aperture illumination functions appropriately with frequency when gridding and combining data from multiple channels.

conjbeams Use conjugate frequency for wideband A-terms

While gridding data from one frequency channel, choose a convolution function from a 'conjugate' frequency such that the resulting baseline primary beam is approximately constant across frequency. For a system in which the primary beam scales with frequency, this step will eliminate instrumental spectral structure from the measured data and leave only the sky spectrum for the minor cycle to model and reconstruct [Bhatnagar et al., ApJ, 2013].

As a rough guideline for when this is relevant, a source at the half power point of the PB at the center frequency will see an artificial spectral index of -1.4 due to the frequency dependence of the PB [Sault and Wieringa, 1994]. If left uncorrected during gridding, this spectral structure must be modeled in the minor cycle (using the mtmfs algorithm) to avoid dynamic range limits (of a few hundred for a 2:1 bandwidth). This works for specmode='mfs' and its value is ignored for cubes

cfcache Convolution function cache directory name

Name of a directory in which to store gridding convolution functions. This cache is filled at the beginning of an imaging run. This step can be time consuming but the cache can be reused across multiple imaging runs that use the same image parameters (cell size, image size , spectral data selections, wprojplanes, wbawp, psterm, aterm). The effect of the wbawp, psterm and aterm settings is frozen-in in the cfcache. Using an existing cfcache made with a different setting of these parameters will not reflect the current settings.

In a parallel execution, the construction of the cfcache is also parallelized and the time to compute scales close to linearly with the number of compute cores used. With the re-computation of Convolution Functions (CF) due to PA rotation turned-off (the computepastep parameter), the total number of in the cfcache can be computed as [No. of wprojplanes x No. of selected spectral windows x 4]

By default, cfcache = imagename + '.cf'

usepointing The usepointing flag informs the gridder that it should utilize the pointing table
to use the correct direction in which the antenna is pointing with respect to the pointing phasecenter.

computepastep Parallactic angle interval after the AIFs are recomputed (deg)

This parameter controls the accuracy of the aperture illumination function used with AProjection for alt-az mount dishes where the AIF rotates on the sky as the synthesis image is built up. Once the PA in the data changes by the given interval, AIFs are re-computed at the new PA.

A value of 360.0 deg (the default) implies no re-computation due to PA rotation. AIFs are computed for the PA value of the first valid data received and used for all of the data.

rotatepastep Parallactic angle interval after which the nearest AIF is rotated (deg)

Instead of recomputing the AIF for every timestep's parallactic angle, the nearest existing AIF is used and rotated after the PA changed by rotatepastep value.

A value of 360.0 deg (the default) disables rotation of the AIF.

For example, computepastep=360.0 and rotatepastep=5.0 will compute the AIFs at only the starting parallactic angle and all other timesteps will use a rotated version of that AIF at the nearest 5.0 degree point.

pointingoffsetsigdev Corrections for heterogenous and time-dependent pointing

offsets via AWProjection are controlled by this parameter. It is a vector of 2 ints or doubles each of which is interpreted in units of arcsec. Based on the first threshold, a clustering algorithm is applied to entries from the POINTING subtable of the MS to determine how distinct antenna groups for which the pointing offset must be computed separately. The second number controls how much a pointing change across time can be ignored and after which an antenna rebinning is required.

Note

[The default value of this parameter is [], due a programmatic constraint.] If run with this value, it will internally pick [600,600] and exercise the option of using large tolerances (10arcmin) on both axes. Please choose a setting explicitly for runs that need to use this parameter.

Note : This option is available only for gridder='awproject' and usepointing=True and

and has been validated primarily with VLASS on-the-fly mosaic data where POINTING subtables have been modified after the data are recorded.

Examples of parameter usage :

[100.0,100.0]

[Pointing offsets of 100 arcsec or less are considered] small enough to be ignored. Using large values for both indicates a homogeneous array.

[10.0, 100.0]

[Based on entries in the POINTING subtable, antennas] are grouped into clusters based on a 10arcsec bin size. All antennas in a bin are given a pointing offset calculated as the average of the offsets of all antennas in the bin. On the time axis, offset changes upto 100 arcsec will be ignored.

[10.0,10.0]

[Calculate separate pointing offsets for each antenna group] (with a 10 arcsec bin size). As a function of time, recalculate the antenna binning if the POINTING table entries change by more than 10 arcsec w.r.to the previously computed binning.

[1.0, 1.0]

[Tight tolerances will imply a fully heterogenous situation where] each antenna gets its own pointing offset. Also, time-dependent offset changes greater than 1 arcsec will trigger recomputes of the phase gradients. This is the most general situation and is also the most expensive option as it constructs and uses separate phase gradients for all baselines and timesteps.

For VLASS 1.1 data with two kinds of pointing offsets, the recommended setting is [30.0, 30.0].

For VLASS 1.2 data with only the time-dependent pointing offsets, the recommended setting is [300.0, 30.0] to turn off the antenna grouping but to retain the time dependent corrections required from one timestep to the next.

pblimit PB gain level at which to cut off normalizations

Divisions by .pb during normalizations have a cut off at a .pb gain level given by pblimit. Outside this limit, image values are set to zero. Additionally, by default, an internal T/F mask is applied to the .pb, .image and .residual images to mask out (T) all invalid pixels outside the pblimit area.

Note

[This internal T/F mask cannot be used as a deconvolution mask.] To do so, please follow the steps listed above in the Notes for the 'gridder' parameter.

Note

[To prevent the internal T/F mask from appearing in anything other] than the .pb and .image.pbcor images, 'pblimit' can be set to a negative number. The absolute value will still be

used as a valid 'pblimit'. A tclean restart using existing output images on disk that already have this T/F mask in the .residual and .image but only pblimit set to a negative value, will remove this mask after the next major cycle.

normtype Normalization type (flatnoise, flatsky, pbsquare)

Gridded (and FT'd) images represent the PB-weighted sky image. Qualitatively it can be approximated as two instances of the PB applied to the sky image (one naturally present in the data and one introduced during gridding via the convolution functions).

xxx.weight : Weight image approximately equal to sum (square (pb)) xxx.pb : Primary beam calculated as sqrt (xxx.weight)

normtype='flatnoise'
[Divide the raw image by sqrt(.weight) so that] the input to the minor cycle represents the product of the sky and PB. The noise is 'flat' across the region covered by each PB.

normtype='flatsky'
[Divide the raw image by .weight so that the input] to the minor cycle represents only the sky. The noise is higher in the outer regions of the primary beam where the sensitivity is low.

normtype='pbsquare'
[No normalization after gridding and FFT.] The minor cycle sees the sky times pb square

deconvolver Name of minor cycle algorithm (hogbom,clark,multiscale,mtmfs,mem,clarkstokes)

Each of the following algorithms operate on residual images and psfs from the gridder and produce output model and restored images. Minor cycles stop and a major cycle is triggered when cyclethreshold or cycleniter are reached. For all methods, components are picked from the entire extent of the image or (if specified) within a mask.

hogbom
[An adapted version of Hogbom Clean [Hogbom, 1974]]

- Find the location of the peak residual
- Add this delta function component to the model image
- Subtract a scaled and shifted PSF of the same size as the image from regions of the residual image where the two overlap.
- Repeat

clark
[An adapted version of Clark Clean [Clark, 1980]]

- Find the location of max(I^2+Q^2+U^2+V^2)
- Add delta functions to each stokes plane of the model image
- Subtract a scaled and shifted PSF within a small patch size from regions of the residual image where the two overlap.
- After several iterations trigger a Clark major cycle to subtract components from the visibility domain, but without de-gridding.
- Repeat

(Note
['clark' maps to imagermode='' in the old clean task.]

'clark_exp' is another implementation that maps to
imagermode='mosaic' or 'csclean' in the old clean task but the behavior is not identical. For now, please use deconvolver='hogbom' if you encounter problems.)

clarkstokes : Clark Clean operating separately per Stokes plane

(Note : 'clarkstokes_exp' is an alternate version. See above.)

multiscale
[MultiScale Clean [Cornwell, 2008]]

- Smooth the residual image to multiple scale sizes
- Find the location and scale at which the peak occurs
- Add this multiscale component to the model image
- Subtract a scaled,smoothed,shifted PSF (within a small patch size per scale) from all residual images
- Repeat from step 2

mtmfs
[Multi-term (Multi Scale) Multi-Frequency Synthesis [Rau and Cornwell, 2011]]

- Smooth each Taylor residual image to multiple scale sizes
- Solve a NTxNT system of equations per scale size to compute Taylor coefficients for components at all locations
- **Compute gradient chi-square and pick the Taylor coefficients**
and scale size at the location with maximum reduction in chi-square
- Add multi-scale components to each Taylor-coefficient model image
- Subtract scaled,smoothed,shifted PSF (within a small patch size per scale) from all smoothed Taylor residual images
- Repeat from step 2

mem
[Maximum Entropy Method [Cornwell and Evans, 1985]]

- Iteratively solve for values at all individual pixels via the MEM method. It minimizes an objective function of

chi-square plus entropy (here, a measure of difference

between the current model and a flat prior model).

(Note
[This MEM implementation is not very robust.] Improvements will be made in the future.)

scales List of scale sizes (in pixels) for multi-scale and mtmfs algorithms.
–> scales=[0,6,20] This set of scale sizes should represent the sizes (diameters in units of number of pixels) of dominant features in the image being reconstructed.

The smallest scale size is recommended to be 0 (point source), the second the size of the synthesized beam and the third 3-5 times the synthesized beam, etc. For example, if the synthesized beam is 10" FWHM and cell=2",try scales = [0,5,15].

For numerical stability, the largest scale must be smaller than the image (or mask) size and smaller than or comparable to the scale corresponding to the lowest measured spatial frequency (as a scale size much

larger than what the instrument is sensitive to is unconstrained by the data making it harder to recovery from errors during the minor cycle).

nterms Number of Taylor coefficients in the spectral model

- nterms=1 : Assume flat spectrum source
- nterms=2 : Spectrum is a straight line with a slope
- nterms=N : A polynomial of order N-1

From a Taylor expansion of the expression of a power law, the spectral index is derived as alpha = taylorcoeff_1 / taylorcoeff_0

Spectral curvature is similarly derived when possible.

The optimal number of Taylor terms depends on the available signal to noise ratio, bandwidth ratio, and spectral shape of the source as seen by the telescope (sky spectrum x PB spectrum).

nterms=2 is a good starting point for wideband EVLA imaging and the lower frequency bands of ALMA (when fractional bandwidth is greater than 10%) and if there is at least one bright source for which a dynamic range of greater than few 100 is desired.

Spectral artifacts for the VLA often look like spokes radiating out from a bright source (i.e. in the image made with standard mfs imaging). If increasing the number of terms does not eliminate these artifacts, check the data for inadequate bandpass calibration. If the source is away from the pointing center, consider including wide-field corrections too.

(Note

[In addition to output Taylor coefficient images .tt0,.tt1,etc] images of spectral index (.alpha), an estimate of error on spectral index (.alpha.error) and spectral curvature (.beta, if nterms is greater than 2) are produced. - These alpha, alpha.error and beta images contain

internal T/F masks based on a threshold computed as peakresidual/10. Additional masking based on

.alpha/.alpha.error may be desirable.

- .alpha.error is a purely empirical estimate derived from the propagation of error during the division of two noisy numbers (alpha = xx.tt1/xx.tt0) where the 'error' on tt1 and tt0 are simply the values picked from the corresponding residual images. The absolute value of the error is not always accurate and it is best to interpret the errors across the image only in a relative sense.)

smallscalebias A numerical control to bias the scales when using multi-scale or mtmfs algorithms.

The peak from each scale's smoothed residual is multiplied by (1 - smallscalebias * scale/maxscale) to increase or decrease the amplitude relative to other scales, before the scale with the largest peak is chosen. Smallscalebias can be varied between -1.0 and 1.0. A score of 0.0 gives all scales equal weight (default).

A score larger than 0.0 will bias the solution towards smaller scales. A score smaller than 0.0 will bias the solution towards larger scales. The effect of smallscalebias is more pronounced when using multi-scale relative to mtmfs.

restoration e.

Construct a restored image : imagename.image by convolving the model image with a clean beam and adding the residual image to the result. If a restoringbeam is specified, the residual image is also smoothed to that target resolution before adding it in.

If a .model does not exist, it will make an empty one and create the restored image from the residuals (with additional smoothing if needed). With algorithm='mtmfs', this will construct Taylor coefficient maps from the residuals and compute .alpha and .alpha.error.

restoringbeam ize to use.

- restoringbeam=" or ['] A Gaussian fitted to the PSF main lobe (separately per image plane).
- restoringbeam='10.0arcsec' Use a circular Gaussian of this width for all planes
- restoringbeam=['8.0arcsec','10.0arcsec','45deg'] Use this elliptical Gaussian for all planes
- restoringbeam='common' Automatically estimate a common beam shape/size appropriate for all planes.

Note

[For any restoring beam different from the native resolution] the model image is convolved with the beam and added to residuals that have been convolved to the same target resolution.

pbcor the output restored image

A new image with extension .image.pbcor will be created from the evaluation of .image / .pb for all pixels above the specified pblimit.

Note

[Stand-alone PB-correction can be triggered by re-running] tclean with the appropriate imagename and with niter=0, calcpsf=False, calcres=False, pbcor=True, vptable='vp.tab' (where vp.tab is the name of the vpmanager file.

See the inline help for the 'vptable' parameter)

Note

[Multi-term PB correction that includes a correction for the] spectral index of the PB has not been enabled for the 4.7 release. Please use the widebandpbcor task instead. (Wideband PB corrections are required when the amplitude of the

brightest source is known accurately enough to be sensitive to the difference in the PB gain between the upper and lower end of the band at its location. As a guideline, the artificial spectral index due to the PB is -1.4 at the 0.5 gain level and less than -0.2 at the 0.9 gain level at the middle frequency)

outlierfile Name of outlier-field image definitions

A text file containing sets of parameter=value pairs, one set per outlier field.

Example : outlierfile='outs.txt'

Contents of outs.txt :

imagename=tst1 nchan=1 imsize=[80,80] cell=[8.0arcsec,8.0arcsec] phasecenter=J2000 19:58:40.895 +40.55.58.543 mask=circle[[40pix,40pix],10pix]

imagename=tst2 nchan=1 imsize=[100,100] cell=[8.0arcsec,8.0arcsec] phasecenter=J2000 19:58:40.895 +40.56.00.000 mask=circle[[60pix,60pix],20pix]

The following parameters are currently allowed to be different between the main field and the outlier fields (i.e. they will be recognized if found in the outlier text file). If a parameter is not listed, the value is picked from what is defined in the main task input.

imagename, imsize, cell, phasecenter, startmodel, mask specmode, nchan, start, width, nterms, reffreq, gridder, deconvolver, wprojplanes

Note

['specmode' is an option, so combinations of mfs and cube]

for different image fields, for example, are supported.

'deconvolver' and 'gridder' are also options that allow different

imaging or deconvolution algorithm per image field.

For example, multiscale with wprojection and 16 w-term planes on the main field and mtmfs with nterms=3 and wprojection with 64 planes on a bright outlier source for which the frequency dependence of the primary beam produces a strong effect that must be modeled. The traditional alternative to this approach is to first image the outlier, subtract it out of the data (uvsub) and then image the main field.

Note

[If you encounter a use-case where some other parameter needs] to be allowed in the outlier file (and it is logical to do so), please send us feedback. The above is an initial list.

weighting Weighting scheme (natural,uniform,briggs,superuniform,radial, briggsabs, briggsbwtaper)

During gridding of the dirty or residual image, each visibility value is multiplied by a weight before it is accumulated on the uv-grid. The PSF's uv-grid is generated by gridding only the weights (weightgrid).

weighting='natural'

[Gridding weights are identical to the data weights] from the MS. For visibilities with similar data weights, the weightgrid will follow the sample density pattern on the uv-plane. This weighting scheme provides the maximum imaging sensitivity at the expense of a possibly fat PSF with high sidelobes. It is most appropriate for detection experiments where sensitivity is most important.

weighting='uniform'

[Gridding weights per visibility data point are the] original data weights divided by the total weight of all data points that map to the same uv grid cell : ' data_weight / total_wt_per_cell '.

The weightgrid is as close to flat as possible resulting in a PSF with a narrow main lobe and suppressed sidelobes. However, since heavily sampled areas of the uv-plane get down-weighted, the imaging sensitivity is not as high as with natural weighting. It is most appropriate for imaging experiments where a well behaved PSF can help the reconstruction.

weighting='briggs'

[Gridding weights per visibility data point are given by]

'data_weight / (A *total_wt_per_cell + B) ' where A and B vary according to the 'robust' parameter.

robust = -2.0 maps to A=1,B=0 or uniform weighting. robust = +2.0 maps to natural weighting. (robust=0.5 is equivalent to robust=0.0 in AIPS IMAGR.)

Robust/Briggs weighting generates a PSF that can vary smoothly between 'natural' and 'uniform' and allow customized trade-offs between PSF shape and imaging sensitivity.

weighting='briggsabs'
[Experimental option.] Same as Briggs except the formula is different A= robust*robust and B is dependent on the noise per visibility estimated. Giving noise='0Jy' is a not a reasonable option. In this mode (or formula) robust values from -2.0 to 0.0 only make sense (2.0 and -2.0 will get the same weighting)

weighting='superuniform'
[This is similar to uniform weighting except that]

the total_wt_per_cell is replaced by the total_wt_within_NxN_cells around the uv cell of interest. (N = subparameter 'npixels')

This method tends to give a PSF with inner sidelobes that are suppressed as in uniform weighting but with far-out sidelobes closer to natural weighting. The peak sensitivity is also closer to natural weighting.

weighting='radial'
[Gridding weights are given by ' data_weight * uvdistance '] This method approximately minimizes rms sidelobes for an east-west synthesis array.

weighting='briggsbwtaper'
[A modified version of Briggs weighting for cubes where an inverse uv taper,] which is proportional to the fractional bandwidth of the entire cube, is applied per channel. The objective is to modify cube (perchanweightdensity = True) imaging weights to have a similar density to that of the continuum imaging weights. This is currently an experimental weighting scheme being developed for ALMA.

For more details on weighting please see Chapter3 of Dan Briggs' thesis (http://www.aoc.nrao.edu/dissertations/dbriggs)

robust Robustness parameter for Briggs weighting.

robust = -2.0 maps to uniform weighting. robust = +2.0 maps to natural weighting. (robust=0.5 is equivalent to robust=0.0 in AIPS IMAGR.)

noise noise parameter for briggs abs mode weighting npixels Number of pixels to determine uv-cell size for super-uniform weighting

(0 defaults to -/+ 3 pixels)

npixels – uv-box used for weight calculation

a box going from -npixel/2 to +npixel/2 on each side

around a point is used to calculate weight density.

npixels=2 goes from -1 to +1 and covers 3 pixels on a side.

npixels=0 implies a single pixel, which does not make sense for
superuniform weighting. Therefore, if npixels=0 it will be forced to 6 (or a box of -3pixels to +3pixels) to cover 7 pixels on a side.

uvtaper uv-taper on outer baselines in uv-plane

Apply a Gaussian taper in addition to the weighting scheme specified via the 'weighting' parameter. Higher spatial frequencies are weighted down relative to lower spatial frequencies to suppress artifacts arising from poorly sampled areas of the uv-plane. It is equivalent to smoothing the PSF obtained by other weighting schemes and can be specified either as a Gaussian in uv-space (eg. units of lambda) or as a Gaussian in the image domain (eg. angular units like arcsec).

uvtaper = [bmaj, bmin, bpa]

NOTE: the on-sky FWHM in arcsec is roughly the uv taper/200 (klambda). default: uvtaper=[]; no Gaussian taper applied example: uvtaper=['5klambda'] circular taper

FWHM=5 kilo-lambda

uvtaper=['5klambda','3klambda','45.0deg'] uvtaper=['10arcsec'] on-sky FWHM 10 arcseconds uvtaper=['300.0'] default units are lambda

in aperture plane

niter Maximum number of iterations

A stopping criterion based on total iteration count. Currently the parameter type is defined as an integer therefore the integer value larger than 2147483647 will not be set properly as it causes an overflow.

Iterations are typically defined as the selecting one flux component and partially subtracting it out from the residual image.

niter=0 : Do only the initial major cycle (make dirty image, psf, pb, etc)

niter larger than zero : Run major and minor cycles.

Note : Global stopping criteria vs major-cycle triggers

In addition to global stopping criteria, the following rules are used to determine when to terminate a set of minor cycle iterations and trigger major cycles [derived from Cotton-Schwab Clean, 1984]

'cycleniter'
[controls the maximum number of iterations per image] plane before triggering a major cycle.

'cyclethreshold'
[Automatically computed threshold related to the]

max sidelobe level of the PSF and peak residual.

Divergence, detected as an increase of 10% in peak residual from the minimum so far (during minor cycle iterations)

The first criterion to be satisfied takes precedence.

Note
[Iteration counts for cubes or multi-field images :]

For images with multiple planes (or image fields) on which the deconvolver operates in sequence, iterations are counted across all planes (or image fields). The iteration count is compared with 'niter' only after all channels/planes/fields have completed their minor cycles and exited either due to 'cycleniter' or 'cyclethreshold'. Therefore, the actual number of iterations reported in the logger can sometimes be larger than the user specified value in 'niter'. For example, with niter=100, cycleniter=20,nchan=10,threshold=0, a total of 200 iterations will be done in the first set of minor cycles before the total is compared with niter=100 and it exits.

Note
[Additional global stopping criteria include]

- no change in peak residual across two major cycles
- a 50% or more increase in peak residual across one major cycle

gain Loop gain

Fraction of the source flux to subtract out of the residual image for the CLEAN algorithm and its variants.

A low value (0.2 or less) is recommended when the sky brightness distribution is not well represented by the basis functions used by the chosen deconvolution algorithm. A higher value can be tried when there is a good match between the true sky brightness structure and the basis function shapes. For example, for extended emission, multiscale clean with an appropriate set of scale sizes will tolerate a higher loop gain than Clark clean (for example).

threshold Stopping threshold (number in units of Jy, or string)

A global stopping threshold that the peak residual (within clean mask) across all image planes is compared to.

threshold = 0.005 : 5mJy threshold = '5.0mJy'

Note

[A 'cyclethreshold' is internally computed and used as a major cycle]

trigger. It is related what fraction of the PSF can be reliably used during minor cycle updates of the residual image. By default the minor cycle iterations terminate once the peak residual reaches the first sidelobe level of the brightest source.

'cyclethreshold' is computed as follows using the settings in
parameters 'cyclefactor','minpsffraction','maxpsffraction','threshold' :

psf_fraction = max_psf_sidelobe_level * 'cyclefactor' psf_fraction = max(psf_fraction, 'minpsffraction'); psf_fraction = min(psf_fraction, 'maxpsffraction'); cyclethreshold = peak_residual * psf_fraction cyclethreshold = max(cyclethreshold, 'threshold')

If nsigma is set (>0.0), the N-sigma threshold is calculated (see the description under nsigma), then cyclethreshold is further modified as,

cyclethreshold = max(cyclethreshold, nsgima_threshold)

'cyclethreshold' is made visible and editable only in the interactive GUI when tclean is run with interactive=True.

nsigma Multiplicative factor for rms-based threshold stopping

N-sigma threshold is calculated as nsigma * rms value per image plane determined from a robust statistics. For nsigma > 0.0, in a minor cycle, a maximum of the two values, the N-sigma threshold and cyclethreshold, is used to trigger a major cycle (see also the descreption under 'threshold'). Set nsigma=0.0 to preserve the previous tclean behavior without this feature. The top level parameter, fastnoise is relevant for the rms noise calculation which is used to determine the threshold.

The parameter 'nsigma' may be an int, float, or a double.

cycleniter Maximum number of minor-cycle iterations (per plane) before triggering
a major cycle

For example, for a single plane image, if niter=100 and cycleniter=20, there will be 5 major cycles after the initial one (assuming there is no threshold based stopping criterion). At each major cycle boundary, if the number of iterations left over (to reach niter) is less than cycleniter, it is set to the difference.

Note

[cycleniter applies per image plane, even if cycleniter x nplanes] gives a total number of iterations greater than 'niter'. This is to preserve consistency across image planes within one set of minor cycle iterations.

cyclefactor Scaling on PSF sidelobe level to compute the minor-cycle stopping threshold.

Please refer to the Note under the documentation for 'threshold' that discussed the calculation of 'cyclethreshold'

cyclefactor=1.0 results in a cyclethreshold at the first sidelobe level of the brightest source in the residual image before the minor cycle starts.

cyclefactor=0.5 allows the minor cycle to go deeper. cyclefactor=2.0 triggers a major cycle sooner.

minpsffraction PSF fraction that marks the max depth of cleaning in the minor cycle

Please refer to the Note under the documentation for 'threshold' that discussed the calculation of 'cyclethreshold'

For example, minpsffraction=0.5 will stop cleaning at half the height of the peak residual and trigger a major cycle earlier.

maxpsffraction PSF fraction that marks the minimum depth of cleaning in the minor cycle

Please refer to the Note under the documentation for 'threshold' that discussed the calculation of 'cyclethreshold'

For example, maxpsffraction=0.8 will ensure that at least the top 20 percent of the source will be subtracted out in the minor cycle even if the first PSF sidelobe is at the 0.9 level (an extreme example), or if the cyclefactor is set too high for anything to get cleaned.

interactive Modify masks and parameters at runtime

interactive=True will trigger an interactive GUI at every major cycle boundary (after the major cycle and before the minor cycle).

The interactive mode is currently not available for parallel cube imaging (please also refer to the Note under the documentation for 'parallel' below).

Options for runtime parameter modification are :

Interactive clean mask
[Draw a 1/0 mask (appears as a contour) by hand.] If a mask is supplied at the task interface or if automasking is invoked, the current mask is displayed in the GUI and is available for manual editing.

Note
[If a mask contour is not visible, please] check the cursor display at the bottom of GUI to see which parts of the mask image have ones and zeros. If the entire mask=1 no contours will be visible.

Operation buttons
[– Stop execution now (restore current model and exit)]

– Continue on until global stopping criteria are reached
without stopping for any more interaction

– Continue with minor cycles and return for interaction
after the next major cycle.

Iteration control : – max cycleniter : Trigger for the next major cycle

The display begins with [min(cycleniter, niter - itercount)] and can be edited by hand.

—iterations left : The display begins with [niter-itercount] and can be edited to increase or decrease the total allowed niter.

– threshold : Edit global stopping threshold

—cyclethreshold : The display begins with the automatically computed value (see Note in help for 'threshold'), and can be edited by hand.

All edits will be reflected in the log messages that appear once minor cycles begin.

[For scripting purposes, replacing True/False with 1/0 will get tclean to return an imaging summary dictionary to python]

usemask Type of mask(s) to be used for deconvolution

user: (default) mask image(s) or user specified region file(s) or string CRTF expression(s)
subparameters: mask, pbmask

pb: primary beam mask
subparameter: pbmask

Example: usemask="pb", pbmask=0.2
Construct a mask at the 0.2 pb gain level. (Currently, this option will work only with gridders that produce .pb (i.e. mosaic and awproject) or if an externally produced .pb image exists on disk)

auto-multithresh
[auto-masking by multiple thresholds for deconvolution]

subparameters
[sidelobethreshold, noisethreshold, lownoisethreshold, negativethrehsold, smoothfactor,] minbeamfrac, cutthreshold, pbmask, growiterations, dogrowprune, minpercentchange, verbose

Additional top level parameter relevant to auto-multithresh: fastnoise

if pbmask is >0.0, the region outside the specified pb gain level is excluded from image statistics in determination of the threshold.

Note: By default the intermediate mask generated by automask at each deconvolution cycle is over-written in the next cycle but one can save them by setting the environment variable, SAVE_ALL_AUTOMASKS="true". (e.g. in the CASA prompt, os.environ['SAVE_ALL_AUTOMASKS']="true") The saved CASA mask image name will be imagename.mask.autothresh#, where # is the iteration cycle number.

mask Mask (a list of image name(s) or region file(s) or region string(s)

The name of a CASA image or region file or region string that specifies a 1/0 mask to be used for deconvolution. Only locations with value 1 will be considered for the centers of flux components in the minor cycle. If regions specified fall completely outside of the image, tclean will throw an error.

Manual mask options/examples :

mask='xxx.mask'
[Use this CASA image named xxx.mask and containing] ones and zeros as the mask. If the mask is only different in spatial coordinates from what is being made it will be resampled to the target coordinate system before being used. The mask has to have the same shape in velocity and Stokes planes as the output image. Exceptions are single velocity and/or single Stokes plane masks. They will be expanded to cover all velocity and/or Stokes planes of the output cube.

[Note
[If an error occurs during image resampling or] if the expected mask does not appear, please try using tasks 'imregrid' or 'makemask' to resample the mask image onto a CASA image with the target shape and coordinates and supply it via the 'mask' parameter.]

mask='xxx.crtf'

[A text file with region strings and the following on the first line] (#CRTFv0 CASA Region Text Format version 0) This is the format of a file created via the viewer's region tool when saved in CASA region file format.

mask='circle[[40pix,40pix],10pix]' : A CASA region string.

mask=['xxx.mask','xxx.crtf', 'circle[[40pix,40pix],10pix]'] : a list of masks

Note

[Mask images for deconvolution must contain 1 or 0 in each pixel.] Such a mask is different from an internal T/F mask that can be held within each CASA image. These two types of masks are not automatically interchangeable, so please use the makemask task to copy between them if you need to construct a 1/0 based mask from a T/F one.

Note

[Work is in progress to generate more flexible masking options and] enable more controls.

pbmask Sub-parameter for usemask='auto-multithresh': primary beam mask

Examples

[pbmask=0.0 (default, no pb mask)] pbmask=0.2 (construct a mask at the 0.2 pb gain level)

sidelobethreshold Sub-parameter for "auto-multithresh": mask threshold based on sidelobe levels: sidelobethreshold * max_sidelobe_level * peak residual noisethreshold Sub-parameter for "auto-multithresh": mask threshold based on the noise level: noisethreshold * rms + location (=median)

The rms is calculated from MAD with rms = 1.4826*MAD.

lownoisethreshold Sub-parameter for "auto-multithresh": mask threshold to grow previously masked regions via binary dilation: lownoisethreshold * rms in residual image + location (=median)

The rms is calculated from MAD with rms = 1.4826*MAD.

negativethreshold Sub-parameter for "auto-multithresh": mask threshold for negative features: -1.0* negativethreshold * rms + location(=median)

The rms is calculated from MAD with rms = 1.4826*MAD.

smoothfactor Sub-parameter for "auto-multithresh": smoothing factor in a unit of the beam minbeamfrac Sub-parameter for "auto-multithresh": minimum beam fraction in size to prune masks smaller than mimbeamfrac * beam

<=0.0 : No pruning

cutthreshold Sub-parameter for "auto-multithresh": threshold to cut the smoothed mask to create a final mask: cutthreshold * peak of the smoothed mask growiterations Sub-parameter for "auto-multithresh": Maximum number of iterations to perform using binary dilation for growing the mask dogrowprune Experimental sub-parameter for "auto-multithresh": Do pruning on the grow mask minpercentchange If the change in the mask size in a particular channel is less than minpercentchange, stop masking that channel in subsequent cycles. This check is only applied when noise based threshold is used and when the previous clean major cycle had a cyclethreshold value equal to the clean threshold. Values equal to -1.0 (or any value less than 0.0) will turn off this check (the default). Automask will still stop masking if the current channel mask is an empty mask and the noise threshold was used to determine the mask. verbose he summary of automasking at the end of each automasking process

is printed in the logger. Following information per channel will be listed in the summary.

chan: channel number masking?: F - stop updating automask for the subsequent iteration cycles RMS: robust rms noise peak: peak in residual image thresh_type: type of threshold used (noise or sidelobe) thresh_value: the value of threshold used N_reg: number of the automask regions N_pruned: number of the automask regions removed by pruning N_grow: number of the grow

mask regions N_grow_pruned: number of the grow mask regions removed by pruning N_neg_pix: number of pixels for negative mask regions

Note that for a large cube, extra logging may slow down the process.

fastnoise mask (user='multi-autothresh') and/or n-sigma stopping threshold (nsigma>0.0) are/is used. If it is set to True, a simpler but faster noise calucation is used.

In this case, the threshold values are determined based on classic statistics (using all unmasked pixels for the calculations).

If it is set to False, the new noise calculation method is used based on pre-existing mask.

Case 1: no exiting mask Calculate image statistics using Chauvenet algorithm

Case 2: there is an existing mask Calculate image statistics by classical method on the region outside the mask and inside the primary beam mask.

In all cases above RMS noise is calculated from MAD.

restart images (and start from an existing model image)

or automatically increment the image name and make a new image set.

True

[Re-use existing images. If imagename.model exists the subsequent]

run will start from this model (i.e. predicting it using current gridder settings and starting from the residual image). Care must be taken when combining this option with startmodel. Currently, only one or the other can be used.

startmodel='', imagename.model exists :

- Start from imagename.model

startmodel='xxx', imagename.model does not exist :

- Start from startmodel

startmodel='xxx', imagename.model exists :

- Exit with an error message requesting the user to pick only one model. This situation can arise when doing one run with startmodel='xxx' to produce an output imagename.model that includes the content of startmodel, and wanting to restart a second run to continue deconvolution. Startmodel should be set to '' before continuing.

If any change in the shape or coordinate system of the image is desired during the restart, please change the image name and use the startmodel (and mask) parameter(s) so that the old model (and mask) can be regridded to the new coordinate system before starting.

False

[A convenience feature to increment imagename with '_1', '_2',]

etc as suffixes so that all runs of tclean are fresh starts (without having to change the imagename parameter or delete images).

This mode will search the current directory for all existing imagename extensions, pick the maximum, and adds 1. For imagename='try' it will make try.psf, try_2.psf, try_3.psf, etc.

This also works if you specify a directory name in the path : imagename='outdir/try'. If './outdir' does not exist, it will create it. Then it will search for existing filenames inside that directory.

If outlier fields are specified, the incrementing happens for each of them (since each has its own 'imagename'). The counters are synchronized across imagefields, to make it easier to match up sets of output images. It adds 1 to the 'max id' from all outlier names on disk. So, if you do two runs with only the main field

(imagename='try'), and in the third run you add an outlier with imagename='outtry', you will get the following image names for the third run : 'try_3' and 'outtry_3' even though 'outry' and 'outtry_2' have not been used.

savemodel Options to save model visibilities (none, virtual, modelcolumn)

Often, model visibilities must be created and saved in the MS to be later used for self-calibration (or to just plot and view them).

none
[Do not save any model visibilities in the MS. The MS is opened] in read-only mode.

Model visibilities can be predicted in a separate step by restarting tclean with niter=0,savemodel=virtual or modelcolumn and not changing any image names so that it finds the .model on disk (or by changing imagename and setting startmodel to the original imagename).

virtual
[In the last major cycle, save the image model and state of the] gridder used during imaging within the SOURCE subtable of the MS. Images required for de-gridding will also be stored internally. All future references to model visibilities will activate the (de)gridder to compute them on-the-fly. This mode is useful when the dataset is large enough that an additional model data column on disk may be too much extra disk I/O, when the gridder is simple enough that on-the-fly recomputing of the model visibilities is quicker than disk I/O. For e.g. that gridder='awproject' does not support virtual model.

modelcolumn
[In the last major cycle, save predicted model visibilities] in the MODEL_DATA column of the MS. This mode is useful when the de-gridding cost to produce the model visibilities is higher than the I/O required to read the model visibilities from disk. This mode is currently required for gridder='awproject'. This mode is also required for the ability to later pull out model visibilities from the MS into a python array for custom processing.

Note 1
[The imagename.model image on disk will always be constructed] if the minor cycle runs. This savemodel parameter applies only to model visibilities created by de-gridding the model image.

Note 2
[It is possible for an MS to have both a virtual model] as well as a model_data column, but under normal operation, the last used mode will get triggered. Use the delmod task to clear out existing models from an MS if confusion arises.

Note 3: when parallel=True, use savemodel='none'; Other options are not yet ready
for use in parallel. If model visibilities need to be saved (virtual or modelcolumn): please run tclean in serial mode with niter=0; after the parallel run

calcres Calculate initial residual image

This parameter controls what the first major cycle does.

calcres=False with niter greater than 0 will assume that a .residual image already exists and that the minor cycle can begin without recomputing it.

calcres=False with niter=0 implies that only the PSF will be made and no data will be gridded.

calcres=True requires that calcpsf=True or that the .psf and .sumwt images already exist on disk (for normalization purposes).

Usage example

[For large runs (or a pipeline scripts) it may be] useful to first run tclean with niter=0 to create an initial .residual to look at and perhaps make a custom mask for. Imaging can be resumed without recomputing it.

calcpsf Calculate PSF

This parameter controls what the first major cycle does.

calcpsf=False will assume that a .psf image already exists and that the minor cycle can begin without recomputing it.

psfcutoff When the .psf image is created a 2 dimensional Gaussian is fit to the main lobe of the PSF.

Which pixels in the PSF are fitted is determined by psfcutoff. The default value of psfcutoff is 0.35 and can varied from 0.01 to 0.99. Fitting algorithm:

- **A region of 41 x 41 pixels around the peak of the PSF is compared against the psfcutoff.**
  Sidelobes are ignored by radially searching from the PSF peak.
- Calculate the bottom left corner (blc) and top right corner (trc) from the points. Expand blc and trc with a number of pixels (5).
- Create a new sub-matrix from blc and trc.
- Interpolate matrix to a target number of points (3001) using CUBIC spline.
- **All the non-sidelobe points, in the interpolated matrix, that are above the psfcutoff are used to fit a Gaussian.**
  A Levenberg-Marquardt algorithm is used.
- **If the fitting fails the algorithm is repeated with the psfcutoff decreased (psfcutoff=psfcutoff/1.5).**
  A message in the log will apear if the fitting fails along with the new value of psfcutoff. This will be done up to 50 times if fitting fails.

This Gaussian beam is defined by a major axis, minor axis, and position angle. During the restoration process, this Gaussian beam is used as the Clean beam. Varying psfcutoff might be useful for producing a better fit for highly non-Gaussian PSFs, however, the resulting fits should be carefully checked. This parameter should rarely be changed.

(This is not the support size for clark clean.)

parallel Run major cycles in parallel (this feature is experimental)

Parallel tclean will run only if casa has already been started using mpirun. Please refer to HPC documentation for details on how to start this on your system.

Example : mpirun -n 3 -xterm 0 *which casa*

Continuum Imaging :

- Data are partitioned (in time) into NProc pieces

- Gridding/iFT is done separately per partition
- Images (and weights) are gathered and then normalized
- One non-parallel minor cycle is run
- Model image is scattered to all processes
- Major cycle is done in parallel per partition

Cube Imaging :

- Data and Image coordinates are partitioned (in freq) into NProc pieces
- Each partition is processed independently (major and minor cycles)
- All processes are synchronized at major cycle boundaries for convergence checks
- At the end, cubes from all partitions are concatenated along the spectral axis

Note 1

[Iteration control for cube imaging is independent per partition.]

- **There is currently no communication between them to synchronize** information such as peak residual and cyclethreshold. Therefore, different chunks may trigger major cycles at different levels.
- For cube imaging in parallel, there is currently no interactive masking.

(Proper synchronization of iteration control is work in progress.)

[1;42mRETURNS[1;m void

——— examples ————————————————————————

This is the first release of our refactored imager code. Although most features have been used and validated, there are many details that have not been thoroughly tested. Feedback will be much appreciated.

Usage Examples :

(A) A suite of test programs that demo all usable modes of tclean on small test datasets https://svn.cv.nrao.edu/svn/casa/branches/release-4_5/gcwrap/python/scripts/tests/test_refimager.py (B) A set of demo examples for ALMA imaging https://casaguides.nrao.edu/index.php/TCLEAN_and_ALMA

_info_group_ = 'imaging'

_info_desc_ = 'Parallelized tclean in consecutive time steps'

__schema

__globals_()

__to_string_(*value*)

__validate_(*doc*, *schema*)

__do_inp_output(*param_prefix*, *description_str*, *formatting_chars*)

__phasecenter_dflt(*glb*)

__phasecenter(*glb*)

__projection_dflt(*glb*)

__projection(*glb*)

__vis_dflt(*glb*)

__vis(*glb*)

__imagesuffix_dflt(*glb*)

__imagesuffix(*glb*)

__parallel_dflt(*glb*)

__parallel(*glb*)

__twidth_dflt(*glb*)

__twidth(*glb*)

__datacolumn_dflt(*glb*)

__datacolumn(*glb*)

__restart_dflt(*glb*)

__restart(*glb*)

__cell_dflt(*glb*)

__cell(*glb*)

__startmodel_dflt(*glb*)

__startmodel(*glb*)

__deconvolver_dflt(*glb*)

__deconvolver(*glb*)

__imsize_dflt(*glb*)

__imsize(*glb*)

__calcpsf_dflt(*glb*)

__calcpsf(*glb*)

__niter_dflt(*glb*)

__niter(*glb*)

__selectdata_dflt(*glb*)

__selectdata(*glb*)

__imageprefix_dflt(*glb*)

__imageprefix(*glb*)

__outlierfile_dflt(*glb*)

__outlierfile(*glb*)

__calcres_dflt(*glb*)

__calcres(*glb*)

__ncpu_dflt(*glb*)

__ncpu(*glb*)

__savemodel_dflt(*glb*)

__savemodel(*glb*)

__usemask_dflt(*glb*)

__usemask(*glb*)

__specmode_dflt(*glb*)

__specmode(*glb*)

__restoration_dflt(*glb*)

__restoration(*glb*)

__stokes_dflt(*glb*)

__stokes(*glb*)

__fastnoise_dflt(*glb*)

__fastnoise(*glb*)

__imagename_dflt(*glb*)

__imagename(*glb*)

__weighting_dflt(*glb*)

__weighting(*glb*)

__gridder_dflt(*glb*)

__gridder(*glb*)

__overwrite_dflt(*glb*)

__overwrite(*glb*)

__doreg_dflt(*glb*)

__doreg(*glb*)

__antenna_dflt(*glb*)

__smoothfactor_dflt(*glb*)

__negativethreshold_dflt(*glb*)

__minbeamfrac_dflt(*glb*)

__psfphasecenter_dflt(*glb*)

__mask_dflt(*glb*)

__sclfactor_dflt(*glb*)

__field_dflt(*glb*)

__cutthreshold_dflt(*glb*)

__pblimit_dflt(*glb*)

__smallscalebias_dflt(*glb*)

__maxpsffraction_dflt(*glb*)

__verbose_dflt(*glb*)

__intent_dflt(*glb*)

__noise_dflt(*glb*)

__interpolation_dflt(*glb*)

__subregion_dflt(*glb*)

__nterms_dflt(*glb*)

__pointingoffsetsigdev_dflt(*glb*)

__nchan_dflt(*glb*)

__reffreq_dflt(*glb*)

__conjbeams_dflt(*glb*)

__restoringbeam_dflt(*glb*)

__sidelobethreshold_dflt(*glb*)

__reftime_dflt(*glb*)

__cycleniter_dflt(*glb*)

__minpsffraction_dflt(*glb*)

__scan_dflt(*glb*)

__computepastep_dflt(*glb*)

__minpercentchange_dflt(*glb*)

__wbawp_dflt(*glb*)

__docompress_dflt(*glb*)

__interactive_dflt(*glb*)

__npixels_dflt(*glb*)

__mosweight_dflt(*glb*)

__pbcor_dflt(*glb*)

__normtype_dflt(*glb*)

__uvtaper_dflt(*glb*)

__cyclefactor_dflt(*glb*)

__toTb_dflt(*glb*)

__restfreq_dflt(*glb*)

__pbmask_dflt(*glb*)

__growiterations_dflt(*glb*)

__gain_dflt(*glb*)

__scales_dflt(*glb*)

__psfcutoff_dflt(*glb*)

__robust_dflt(*glb*)

__vptable_dflt(*glb*)

__perchanweightdensity_dflt(*glb*)

__aterm_dflt(*glb*)

__usephacenter_dflt(*glb*)

__usepointing_dflt(*glb*)

__rotatepastep_dflt(*glb*)

__threshold_dflt(*glb*)

__veltype_dflt(*glb*)

__outframe_dflt(*glb*)

__dogrowprune_dflt(*glb*)

__uvrange_dflt(*glb*)

__psterm_dflt(*glb*)

__start_dflt(*glb*)

__observation_dflt(*glb*)

__lownoisethreshold_dflt(*glb*)

__facets_dflt(*glb*)

__noisethreshold_dflt(*glb*)

__width_dflt(*glb*)

__spw_dflt(*glb*)

__timerange_dflt(*glb*)

__nsigma_dflt(*glb*)

__cfcache_dflt(*glb*)

__wprojplanes_dflt(*glb*)

__usephacenter(*glb*)

__reftime(*glb*)

__toTb(*glb*)

__sclfactor(*glb*)

__subregion(*glb*)

__docompress(*glb*)

__field(*glb*)

__spw(*glb*)

__timerange(*glb*)

__uvrange(*glb*)

__antenna(*glb*)

__scan(*glb*)

__observation(*glb*)

__intent(*glb*)

__reffreq(*glb*)

__nchan(*glb*)

__start(*glb*)

__width(*glb*)

__outframe(*glb*)

__veltype(*glb*)

__restfreq(*glb*)

__interpolation(*glb*)

__perchanweightdensity(*glb*)

__facets(*glb*)

__psfphasecenter(*glb*)

__wprojplanes(*glb*)

__vptable(*glb*)

__mosweight(*glb*)

__aterm(*glb*)

__psterm(*glb*)

__wbawp(*glb*)

__conjbeams(*glb*)

__cfcache(*glb*)

__usepointing(*glb*)

__computepastep(*glb*)

__rotatepastep(*glb*)

__pointingoffsetsigdev(*glb*)

__pblimit(*glb*)

__normtype(*glb*)

__scales(*glb*)

__nterms(*glb*)

__smallscalebias(*glb*)

__restoringbeam(*glb*)

__pbcor(*glb*)

__robust(*glb*)

__noise(*glb*)

__npixels(*glb*)

__uvtaper(*glb*)

__gain(*glb*)

__threshold(*glb*)

__nsigma(*glb*)

__cycleniter(*glb*)

__cyclefactor(*glb*)

__minpsffraction(*glb*)

__maxpsffraction(*glb*)

__interactive(*glb*)

__mask(*glb*)

__pbmask(*glb*)

__sidelobethreshold(*glb*)

__noisethreshold(*glb*)

__lownoisethreshold(*glb*)

__negativethreshold(*glb*)

__smoothfactor(*glb*)

__minbeamfrac(*glb*)

__cutthreshold(*glb*)

__growiterations(*glb*)

__dogrowprune(*glb*)

__minpercentchange(*glb*)

__verbose(*glb*)

__psfcutoff(*glb*)

__vis_inp()

__imageprefix_inp()

__imagesuffix_inp()

__ncpu_inp()

__twidth_inp()

__doreg_inp()

__usephacenter_inp()

__reftime_inp()

__toTb_inp()

__sclfactor_inp()

__subregion_inp()

__docompress_inp()

__overwrite_inp()

__selectdata_inp()

__field_inp()

__spw_inp()

__timerange_inp()

__uvrange_inp()

__antenna_inp()

__scan_inp()

__observation_inp()

__intent_inp()

__datacolumn_inp()

__imagename_inp()

__imsize_inp()

__cell_inp()

__phasecenter_inp()

__stokes_inp()

__projection_inp()

__startmodel_inp()

__specmode_inp()

__reffreq_inp()

__nchan_inp()

__start_inp()

__width_inp()

__outframe_inp()

__veltype_inp()

__restfreq_inp()

__interpolation_inp()

__perchanweightdensity_inp()

__gridder_inp()

__facets_inp()

__psfphasecenter_inp()

__wprojplanes_inp()

__vptable_inp()

__mosweight_inp()

__aterm_inp()

__psterm_inp()

__wbawp_inp()

__conjbeams_inp()

__cfcache_inp()

__usepointing_inp()

__computepastep_inp()

__rotatepastep_inp()

__pointingoffsetsigdev_inp()

__pblimit_inp()

__normtype_inp()

__deconvolver_inp()

__scales_inp()

__nterms_inp()

__smallscalebias_inp()

__restoration_inp()

__restoringbeam_inp()

__pbcor_inp()

__outlierfile_inp()

__weighting_inp()

__robust_inp()

__noise_inp()

__npixels_inp()

__uvtaper_inp()

__niter_inp()

__gain_inp()

__threshold_inp()

__nsigma_inp()

__cycleniter_inp()

__cyclefactor_inp()

__minpsffraction_inp()

__maxpsffraction_inp()

__interactive_inp()

__usemask_inp()

__mask_inp()

__pbmask_inp()

__sidelobethreshold_inp()

__noisethreshold_inp()

__lownoisethreshold_inp()

__negativethreshold_inp()

__smoothfactor_inp()

__minbeamfrac_inp()

__cutthreshold_inp()

__growiterations_inp()

__dogrowprune_inp()

__minpercentchange_inp()

__verbose_inp()

__fastnoise_inp()

__restart_inp()

__savemodel_inp()

__calcres_inp()

__calcpsf_inp()

__psfcutoff_inp()

__parallel_inp()

set_global_defaults()

inp()

tget(*file=None*)

__call__(*vis=None*, *imageprefix=None*, *imagesuffix=None*, *ncpu=None*, *twidth=None*, *doreg=None*, *usephacenter=None*, *reftime=None*, *toTb=None*, *sclfactor=None*, *subregion=None*, *docompress=None*, *overwrite=None*, *selectdata=None*, *field=None*, *spw=None*, *timerange=None*, *uvrange=None*, *antenna=None*, *scan=None*, *observation=None*, *intent=None*, *datacolumn=None*, *imagename=None*, *imsize=None*, *cell=None*, *phasecenter=None*, *stokes=None*, *projection=None*, *startmodel=None*, *specmode=None*, *reffreq=None*, *nchan=None*, *start=None*, *width=None*, *outframe=None*, *veltype=None*, *restfreq=None*, *interpolation=None*, *perchanweightdensity=None*, *gridder=None*, *facets=None*, *psfphasecenter=None*, *wprojplanes=None*, *vptable=None*, *mosweight=None*, *aterm=None*, *psterm=None*, *wbawp=None*, *conjbeams=None*, *cfcache=None*, *usepointing=None*, *computepastep=None*, *rotatepastep=None*, *pointingoffsetsigdev=None*, *pblimit=None*, *normtype=None*, *deconvolver=None*, *scales=None*, *nterms=None*, *smallscalebias=None*, *restoration=None*, *restoringbeam=None*, *pbcor=None*, *outlierfile=None*, *weighting=None*, *robust=None*, *noise=None*, *npixels=None*, *uvtaper=None*, *niter=None*, *gain=None*, *threshold=None*, *nsigma=None*, *cycleniter=None*, *cyclefactor=None*, *minpsffraction=None*, *maxpsffraction=None*, *interactive=None*, *usemask=None*, *mask=None*, *pbmask=None*, *sidelobethreshold=None*, *noisethreshold=None*, *lownoisethreshold=None*, *negativethreshold=None*, *smoothfactor=None*, *minbeamfrac=None*, *cutthreshold=None*, *growiterations=None*, *dogrowprune=None*, *minpercentchange=None*, *verbose=None*, *fastnoise=None*, *restart=None*, *savemodel=None*, *calcres=None*, *calcpsf=None*, *psfcutoff=None*, *parallel=None*)

ptclean6.**ptclean6**

1.1.9 subvs

Module Contents

Classes

_subvs	subvs ---- Vector-subtraction in UV using selected time ranges and spectral channels as background

Functions

static_var(varname, value)

Attributes

subvs

subvs.**static_var**(*varname*, *value*)

class subvs._subvs

subvs —- Vector-subtraction in UV using selected time ranges and spectral channels as background

Split is the general purpose program to make a new data set that is a subset or averaged form of an existing data set. General selection parameters are included, and one or all of the various data columns (DATA, LAG_DATA and/or FLOAT_DATA, and possibly MODEL_DATA and/or CORRECTED_DATA) can be selected.

Split is often used after the initial calibration of the data to make a smaller measurement set with only the data that will be used in further flagging, imaging and/or self-calibration. split can average over frequency (channels) and time (integrations).

——— parameter descriptions ———————————————

vis Name of input measurement set outputvis Name of output measurement set timerange Select the time range of the input visbility to be subtracted from spw Select the spectral channels of the input visibility to be subtracted from mode Operation: linear, highpass subtime1 Select the first time range as the background for uv subtraction subtime2 Select the second time range as the background for uv subtraction smoothaxis Select the axis along which smooth is performed smoothtype Select the smooth type smoothwidth Select the width of the smoothing window splitsel Split the selected timerange and spectral channels as outputvis reverse Reverse the sign of the background-subtracted data (for absorptive structure) overwrite Overwrite the already existing output measurement set

——— examples ————————————————————

Subvs is a task to do UV vector-subtraction, by selecting time ranges in the data as background. Subvs can be used to subtract the background continuum emission to separate the time-dependent emission, e.g. solar coherent radio bursts.

Keyword arguments: vis – Name of input visibility file (MS) default: none; example: vis='ngc5921.ms' outputvis – Name of output uv-subtracted visibility file (MS) default: none; example: outputvis='ngc5921_src.ms' timerange – Time range of performing the UV subtraction: default='' means all times. examples: timerange = 'YYYY/MM/DD/hh:mm:ss~YYYY/MM/DD/hh:mm:ss' timerange = 'hh:mm:ss~hh:mm:ss' spw – Select spectral window/channel. default = '' all the spectral channels. Example: spw='0:1~20' mode – operation mode default 'linear' mode = 'linear': use a linear fit for the background to be subtracted mode = 'lowpass': act as a lowpass filter—smooth the data using different smooth types and smooth window size. Can be performed along either time or frequency axis mode = 'highpass': act as a highpass filter—smooth the data first, and subtract the smoothed data from the original. Can be performed along either time or frequency axis mode = 'linear' expandable parameters: subtime1 – Time range 1 of the background to be subtracted from the data default='' means all times. format: timerange = 'YYYY/MM/DD/hh:mm:ss~YYYY/MM/DD/hh:mm:ss' timerange = 'hh:mm:ss~hh:mm:ss' subtime2 – Time range 2 of the backgroud to be subtracted from the data default='' means all times. examples: timerange = 'YYYY/MM/DD/hh:mm:ss~YYYY/MM/DD/hh:mm:ss' timerange = 'hh:mm:ss~hh:mm:ss' mode = 'lowpass' or 'highpass' expandable parameters: smoothaxis – axis of smooth Default: 'time' smoothaxis = 'time': smooth is along the time axis smoothaxis = 'freq': smooth is along the frequency axis smoothtype – type of the smooth depending on the convolving kernel Default: 'flat' smoothtype = 'flat': convolving kernel is a flat rectangle, equivalent to a boxcar moving smooth smoothtype = 'hanning': Hanning smooth kernel. See numpy.hanning smoothtype = 'hamming': Hamming smooth kernel. See numpy.hamming smoothtype = 'bartlett': Bartlett smooth kernel. See numpy.bartlett smoothtype = 'blackman': Blackman smooth kernel. See numpy.blackman smoothwidth – width of the smooth kernel Default: 5 Examples: smoothwidth=5, meaning the width is 5 pixels splitsel – True or False. default = False. If splitsel = False, then the entire input measurement set is copied as the output measurement set (outputvis), with background subtracted at selected timerange and spectral channels. If splitsel = True,then only the selected timerange and spectral channels are copied into the output measurement set (outputvis). reverse – True or False. default = False. If reverse = False, then the times indicated by subtime1 and/or subtime2 are treated as background and subtracted; If reverse = True, then reverse the sign of the background-subtracted data. The option can be used for mapping absorptive structure. overwrite – True or False. default = False. If overwrite = True and outputvis already exists, the selected subtime and spw in the output measurment set will be replaced with background subtracted visibilities

_info_group_ = 'misc'

_info_desc_ = 'Vector-subtraction in UV using selected time ranges and spectral channels as background'

__schema

__globals_()

__to_string_(*value*)

__validate_(*doc*, *schema*)

__do_inp_output(*param_prefix*, *description_str*, *formatting_chars*)

__vis_dflt(*glb*)

__vis(*glb*)

__spw_dflt(*glb*)

__spw(*glb*)

__mode_dflt(*glb*)

__mode(*glb*)

__outputvis_dflt(*glb*)

__outputvis(*glb*)

__timerange_dflt(*glb*)

__timerange(*glb*)

__splitsel_dflt(*glb*)

__splitsel(*glb*)

__reverse_dflt(*glb*)

__reverse(*glb*)

__overwrite_dflt(*glb*)

__overwrite(*glb*)

__smoothaxis_dflt(*glb*)

__smoothwidth_dflt(*glb*)

__subtime2_dflt(*glb*)

__subtime1_dflt(*glb*)

__smoothtype_dflt(*glb*)

__subtime1(*glb*)

__subtime2(*glb*)

__smoothaxis(*glb*)

__smoothtype(*glb*)

__smoothwidth(*glb*)

__vis_inp()

__outputvis_inp()

__timerange_inp()

__spw_inp()

__mode_inp()

__subtime1_inp()

__subtime2_inp()

__smoothaxis_inp()

__smoothtype_inp()

__smoothwidth_inp()

__splitsel_inp()

__reverse_inp()

__overwrite_inp()

set_global_defaults()

inp()

tget(*file=None*)

__call__(*vis=None*, *outputvis=None*, *timerange=None*, *spw=None*, *mode=None*, *subtime1=None*, *subtime2=None*, *smoothaxis=None*, *smoothtype=None*, *smoothwidth=None*, *splitsel=None*, *reverse=None*, *overwrite=None*)

subvs.**subvs**

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX

Symbols

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z