SHAMFI

Release 1.19.5

Nov 08, 2022

Contents

SHApelet **M**odelling **F**or **I**nterferometers

SHAMFI is designed to fit images out of radio-frequency interferometers using shapelet basis functions (Refregier 2013). It's currently limited to fitting a FITS file, such as those output by WSClean. It can output models that work with the calibration and imaging software the RTS (Mitchell et al 2008) and the simulator WODEN, so that the image based fits can be used to generate visibilities.

The repo includes:

Script	Overall Function
`fit_shamfi.py`	Takes a FITS file, and fits a shapelet model
`mask_fits_shamfi.py`	Splits a FITS image into multiple images using gaussian masks
`subtract_gauss_from_image_shamfi.py`	Subtracts specified gaussians from an image to make fitting shapelets easier
`combine_srclists_shamfi.py`	Combine multiple RTS/WODEN source catalogues into one
`convert_srclists_shamfi.py`	Convert between RTS and WODEN source catalogue formats

CHAPTER 1

Installation

Grab the source code from this git repo:

```
git clone https://github.com/JLBLine/SHAMFI
```

Then navigate into that directory and run

```
pip install .
```

or alternatively

```
python setup.py install
```

CHAPTER 2

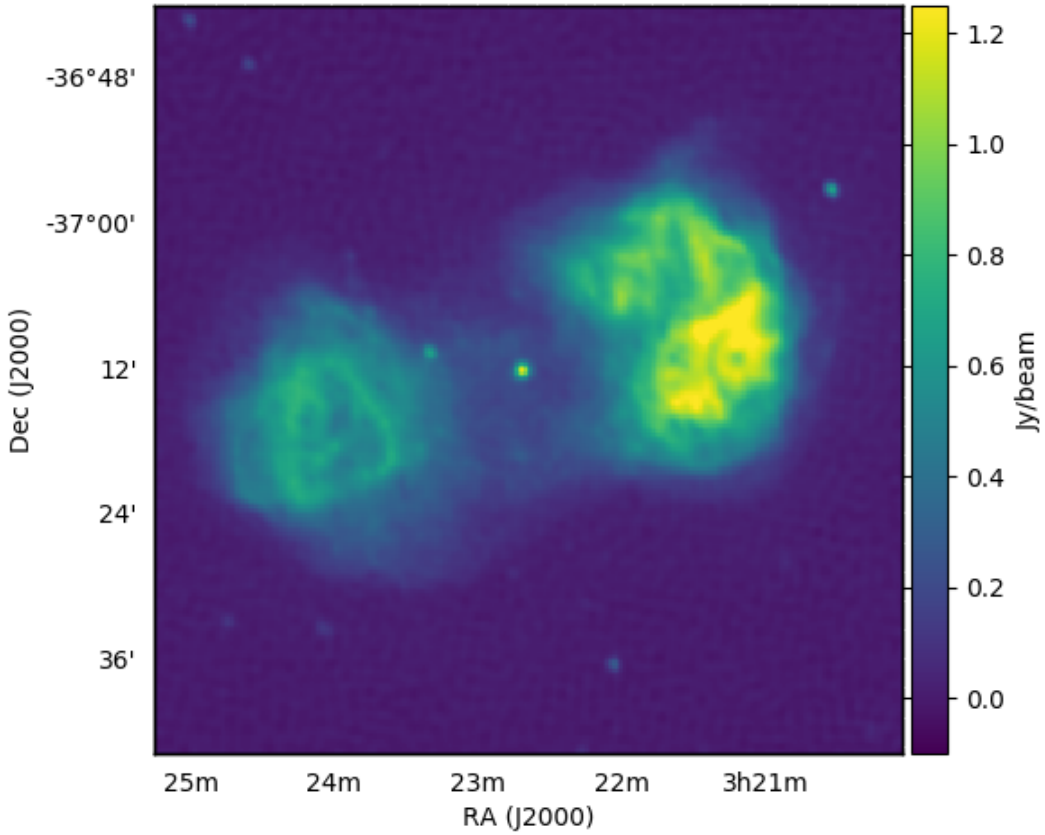
Usage

Check out the tutorial page for an example of how to run the scripts

2.1 Tutorial

We will use fitting Fornax A (as detailed in Line et al. 2020) as an example. First of all, we need a FITS file. Most FITS files should work, but SHAMFI has been tested against FITS files out of WSClean. Fornax A is a complicated source and so requires some manipulation before fitting; if you are fitting something with less structure, you can probably skip the first two steps.

Our FITS file is called `cropped_FornaxA_real_phase1+2.fits` (which you can find in the `tutorial_files` directory of the SHAMFI repo), was made by multi-scale CLEANing MWA data, and looks like this:



2.1.1 Step 1: Remove compact sources (optional but a good idea)

The more compact emission in a source, the more high-order basis functions are required to accurately fit the source. Point-like sources that are towards the edges are particularly problematic. To remove them we run `subtract_gauss_from_image_shamfi.py` with the following arguments:

```
subtract_gauss_from_image_shamfi.py \
   --fits_file=cropped_FornaxA_real_phase1+2.fits \
   --gauss_table=gaussians_to_subtract.txt \
   --outname=gauss-subtracted_phase1+2_real_data.fits
```

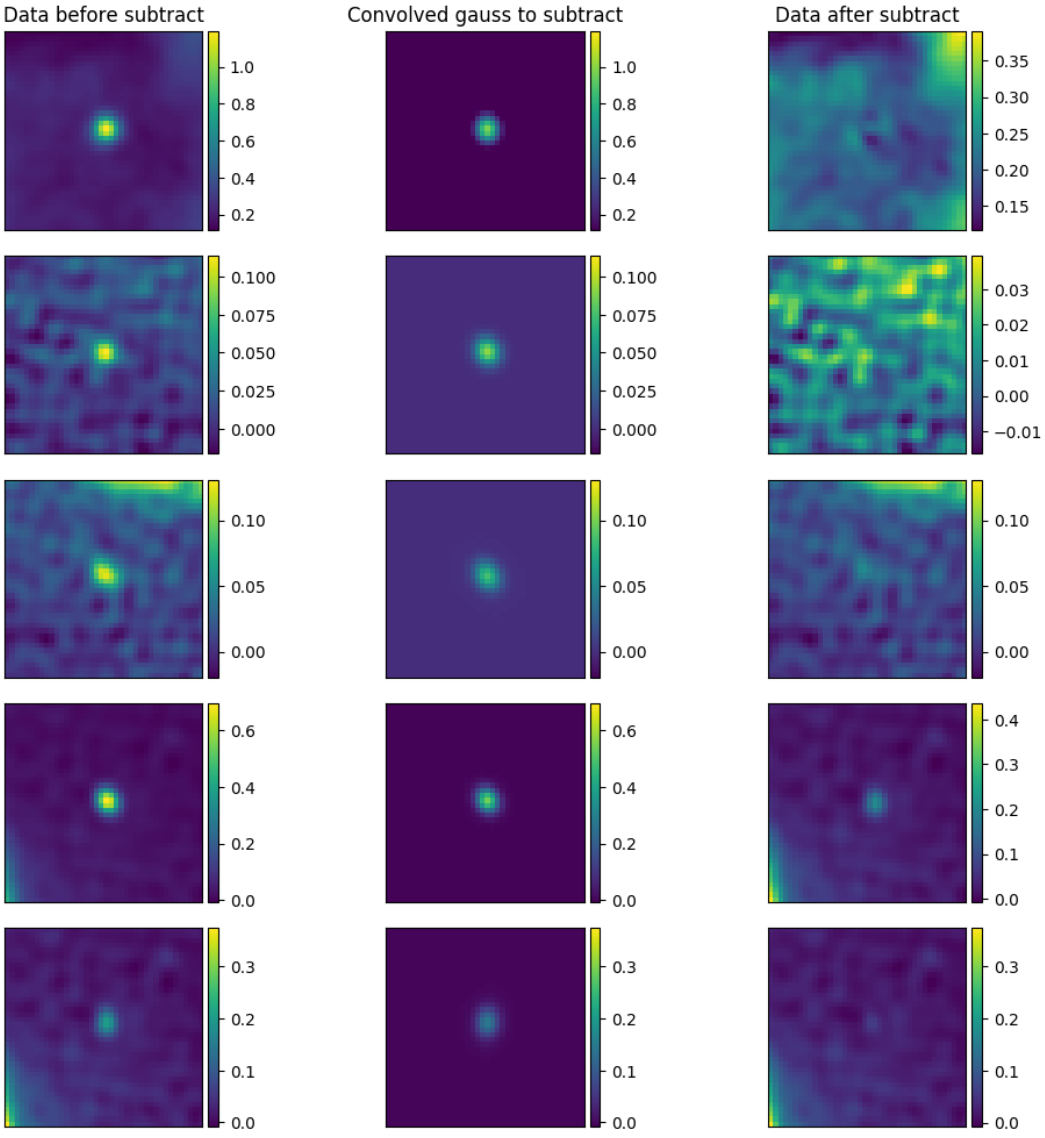
Unfortunately we have to manually create `gaussians_to_subtract.txt`, which requires the following columns: `x_cent(pixels) y_cent(pixels) major(FWHM, arcmins) minor(FWHM, arcmins) pa(deg) int_flux(Jy)`. I have placed a copy in the `tutorial_files` directory, where `gaussians_to_subtract.txt` looks like:

```
## x_cent(pixels) y_cent(pixels) major(FWHM, arcmins) minor(FWHM, arcmins) pa(deg)␣
↪int_flux(Jy)
126 130.8 0.6 0.4 -30.0 1.3
25 45 0.5 0.5 30.0 0.12
58 42 0.7 0.5 40.0 0.123
232 193 0.2 0.2 0.0 0.55
233 192.5 0.8 0.2 -20.0 0.2
```

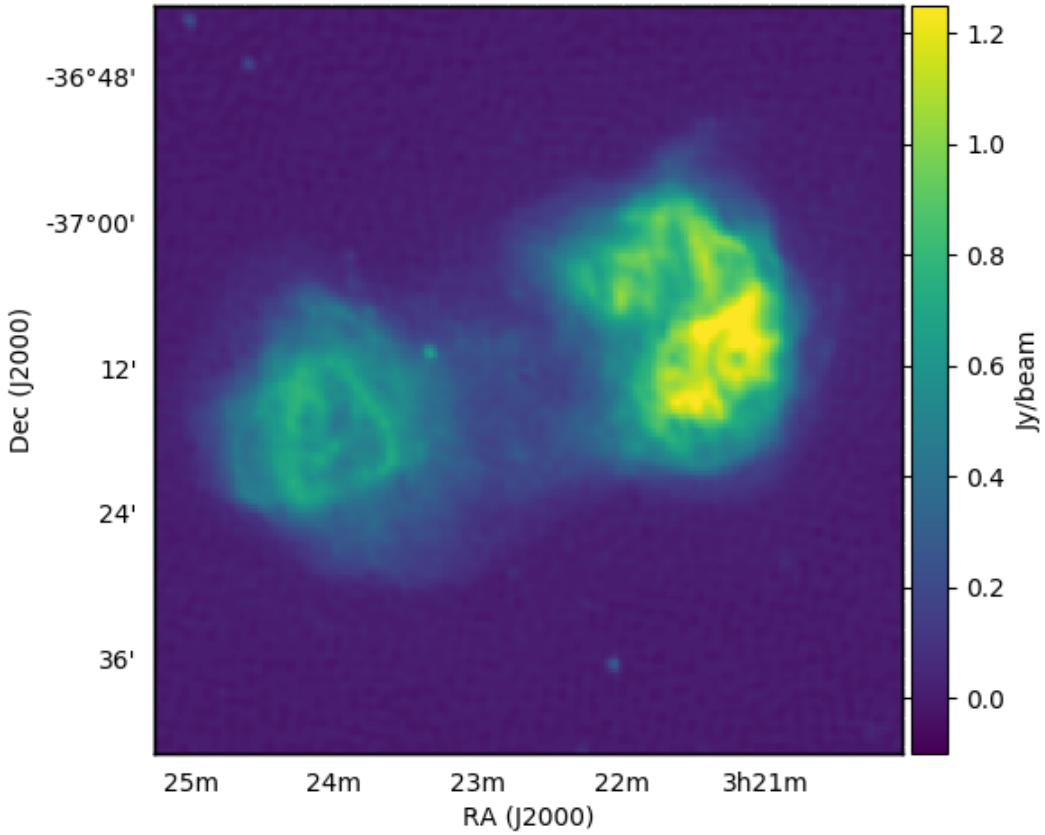
Running the `subtract_gauss_from_image_shamfi.py` command above will produce:

- `gauss-subtracted_phase1+2_real_data.png`
- `gauss-subtracted_phase1+2_real_data.fits`
- `srclist_gaussian-rts.txt`
- `srclist_gaussian-woden.txt`

You can use `gauss-subtracted_phase1+2_real_data.png`, which you can use to tune your parameter estimates for the gaussians. I typically use `kvis` to estimate the x_cent and y_cent pixel locations, and then tune the PA, major, and minor using the output plot, which shows the sources you are trying to subtract before and after subtraction:



It says 'convolved gauss to subtract' as `subtract_gauss_from_image_shamfi.py` convolves the requested gaussian parameters with the restoring beam used to create the CLEANed image. Once you've fiddled the parameters to your liking, you can see what you've done by inspecting `gauss-subtracted_phase1+2_real_data.fits`:



2.1.2 Step 2: Split the galaxy in twain

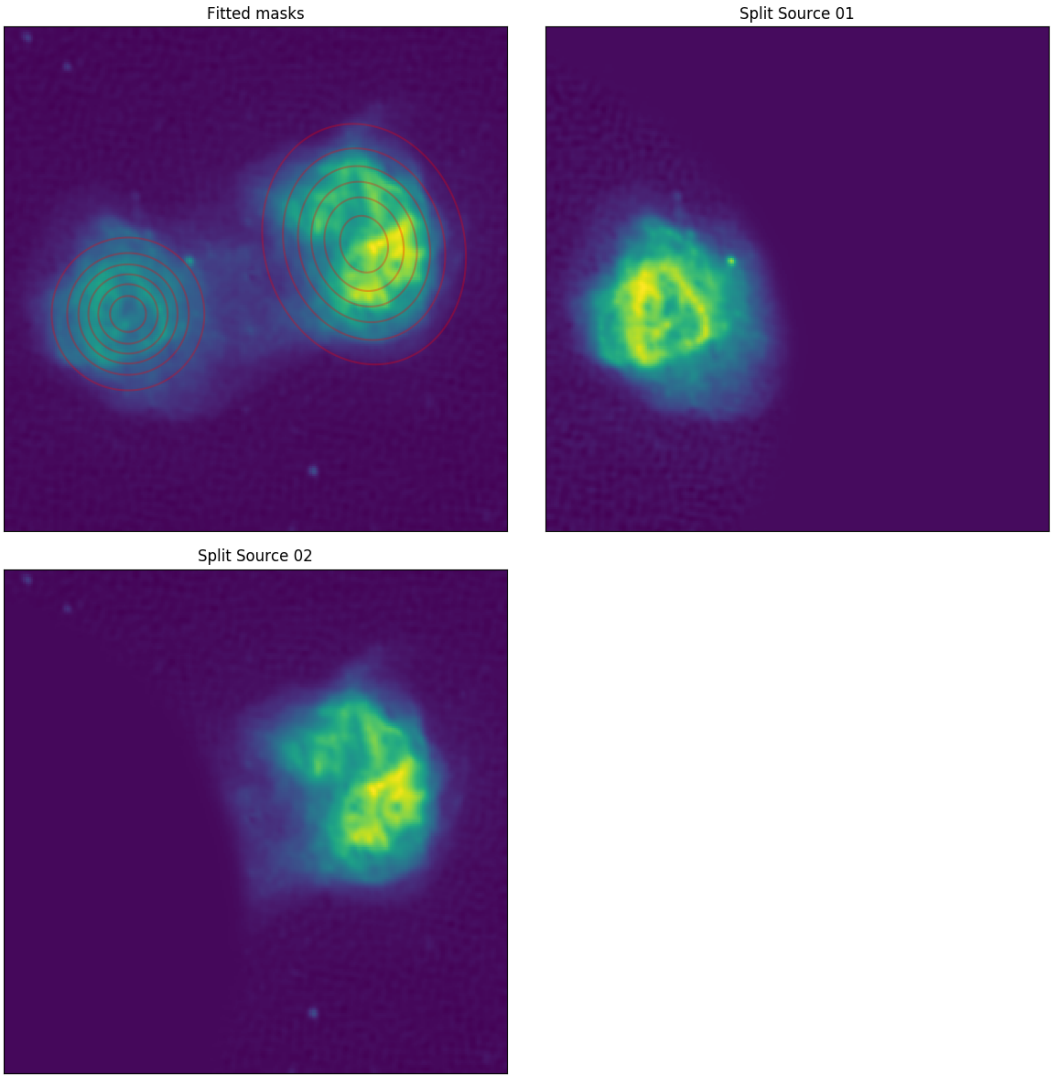
As detailed in Line et al. 2020, the *x,y=0,0* pixel centre of the shapelet basis function greatly effects the quality of the fit. As the lobes of Fornax A are individually complicated, life is easier if we fit each lobe separately. We do that with the following command:

```
mask_fits_shamfi.py \
   --fits_file=gauss-subtracted_phase1+2_real_data.fits \
   --output_tag=real_ForA_phase1+2 \
   --box=6,120,50,170 --box=117,246,75,218
```

The `--box` command outlines two areas in pixel coords (*xmin, xmax, ymin, ymax*) to fit an overall gaussian mask within, to split the image by weighting by the fitted gaussians. Running this command will produce:

- `real_ForA_phase1+2_masked.png`
- `real_ForA_phase1+2_split01.fits`
- `real_ForA_phase1+2_split02.fits`

We can see the result by inspecting `real_ForA_phase1+2_masked.png`:



Ok! Now we've pulled the image to pieces we can finally start modelling it.

2.1.3 Step 3: Fit the lobes

First up, let's look at the commands, and then I'll break them down.

```
fit_shamfi.py \
    --save_tag=real_ForA_phase1+2_lobe1 \
    --fits_file=real_ForA_phase1+2_split01.fits \
    --b1s=3.0,4.0 --b2s=3.0,4.0 --nmax=86 \
    --num_beta_values=5 \
    --edge_pad=25 \
    --fit_box=0,200,50,240 \
    --woden_srclist --plot_resid_grid --plot_edge_pad \
    --compress=90.0,80.0,70.0

fit_shamfi.py \
    --save_tag=real_ForA_phase1+2_lobe2 \
    --fits_file=real_ForA_phase1+2_split02.fits \
```

(continues on next page)

(continued from previous page)

```
    --b1s=3.0,4.0 --b2s=3.0,4.0 --nmax=86 \
    --num_beta_values=5 \
    --fit_box=100,300,80,270 \
    --edge_pad=25 \
    --woden_srclist --plot_resid_grid --plot_edge_pad \
    --compress=90.0,80.0,70.0
```

Running the first command will produce:

- `grid-fit_matrix_real_ForA_phase1+2_lobe1.png`
- `shamfi_real_ForA_phase1+2_lobe1_nmax86_fit.fits`
- `shamfi_real_ForA_phase1+2_lobe1_nmax86_p100_fit.png`
- `srclist-woden_real_ForA_phase1+2_lobe1_nmax086_p100.txt`

Similarly the second command will produce equivalent outputs for 'lobe2'. Here are some arguments and explanations of how I've arrived at these values. First off we need a couple equations to set some arguments:

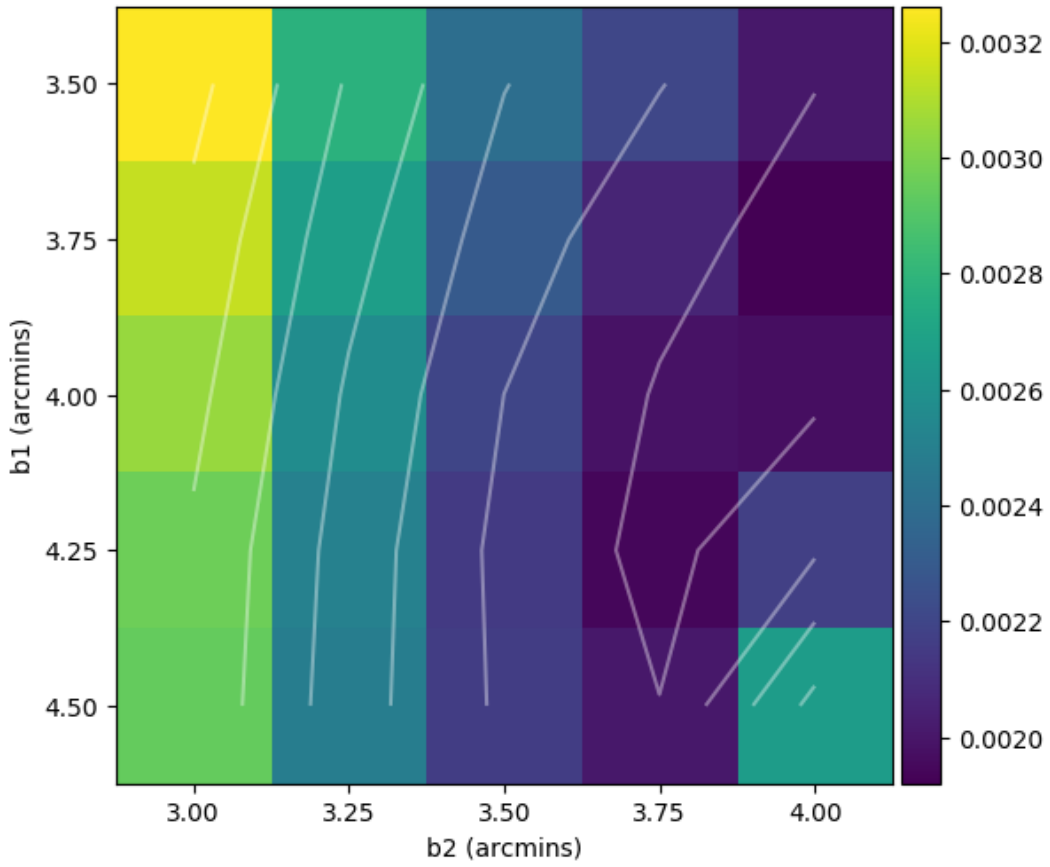
$$n_{\max} \approx \frac{\vartheta_{\max}}{\vartheta_{\min}} - 1$$

$$\beta \approx (\vartheta_{\min}\vartheta_{\max})^{\frac{1}{2}}$$

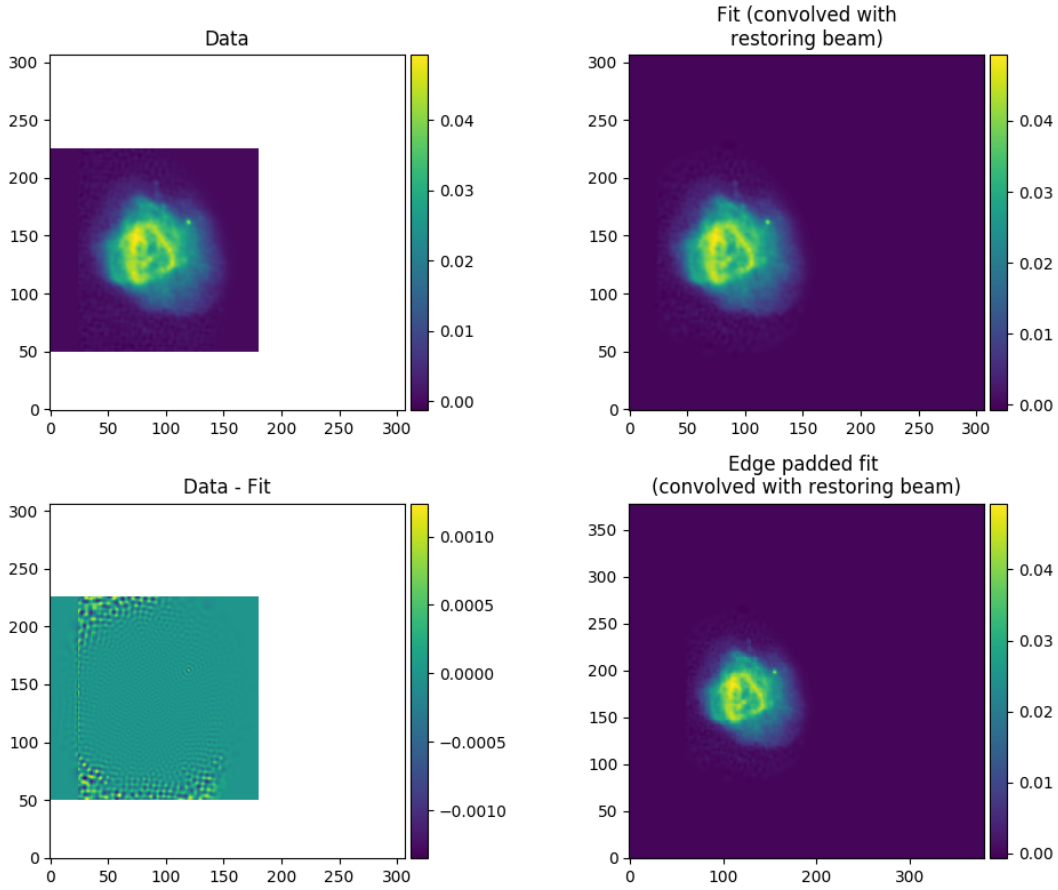
where $n_{\max}$ is the maximum order of the basis functions to fit, $\vartheta_{\max}$ is the maximum scale of the image you are trying to model, and $\vartheta_{\min}$ is the minimum scale, and β is a scaling factor for the basis functions. For this image, $\vartheta_{\max} \sim 0.5°$, and to set $\vartheta_{\min}$ I oversampled the angular resolution of the MWA in this image, by 3. Plugging those values in gives $n_{\max} = 86$ and $\beta \sim 3.2$ arcmins, which give us starting points for the fitting process. Some other arguments and reasoning are below.

Argument	Values and Reasons
`--b1s=3.5,4.5`	The range over which to vary the β scaling parameter for the major axis. Started with values around 3.2 as calculated above and changed the ranges based on fitting outcomes
`--num_beta_values=5`	SHAMFI does a grid search over all β parameters - this means SHAMFI will fit 5 values for both β_1 and β_2, for a total of 25 combinations
`--plot_edge_pad`	If the size of the basis functions exceed the area of the pixels being fitted, the model outside the desired area is unconstrained and you can get nonsense results. This option will plot an edge-padded image of the fitted image so you can check outside the area you fitted
`--edge_pad=25`	If you find you are getting bad results, you can set this to edge pad the image with zero pixels to constrain the model outside the image
`--fit_box=100,290,85,260`	Fitting is expensive when you have a large $n_{\max}$ so you can tell SHAMFI to only fit a certain box of pixels (by specifying a box bounded by pixel number, as *xmin, xmax, ymin, ymax*). Note if you use the `--edge_pad` option here, you'll need to supply bounds with the extra pixels applied.

Once that's finished run, you can inspect the fitting residuals for each combination of β_1 and β_2 by looking at `grid-fit_matrix_real_ForA_phase1+2_lobe1.png`, to see if you need to change your β ranges:



And of course, check out your model fit by looking at `shamfi_real_ForA_phase1+2_lobe1_nmax86_p100_fit.png`. Note that only the box specified by `--fit_box` is plotted for the data so you know what you asked to be fitted.



You now have two separate lobes and a number of gaussian models, so we need to stitch them together into a coherent single model.

2.1.4 Step 4: Combine the models

Simply add as many single source models with the `--srclist` argument, and combine them into a single model:

```
combine_srclists_shamfi.py \
    --srclist=srclist-woden_real_ForA_phase1+2_lobe1_nmax086_p100.txt \
    --srclist=srclist-woden_real_ForA_phase1+2_lobe2_nmax086_p100.txt \
    --srclist=srclist_gaussian-woden.txt \
    --outname=srclist-woden_real_ForA_phase1+2_nmax086_p100.txt
```

That's it! You now have a model that you can plug into WODEN. If you want to plug the model into the RTS as well, you can use `convert_srclists_shamfi.py` to switch between formats (or run SHAMFI with `--rts_srclist` from the start).

```
convert_srclists_shamfi.py \
    --srclist=srclist-woden_real_ForA_phase1+2_nmax086_p100.txt \
    --outname=srclist-rts_real_ForA_phase1+2_nmax086_p100.txt
```

CHAPTER 3

Scripts

3.1 Script Documentation

A low down of all arguments for all scripts included in SHAMFI. These are auto-magically generated from the scripts themselves and can be reproduced on the command line via `script.py --help`.

3.1.1 fit_shamfi.py

A script to fit a shapelet model consistent with the RTS or WODEN

```
usage: fit_shamfi.py [-h] [--rts_srclist] [--woden_srclist] [--no_srclist]
                     [--fits_file FITS_FILE] [--b1s B1S] [--b2s B2S]
                     [--nmax NMAX] [--save_tag SAVE_TAG]
                     [--plot_lims PLOT_LIMS] [--freq FREQ]
                     [--already_jy_per_pixel] [--edge_pad EDGE_PAD]
                     [--num_beta_values NUM_BETA_VALUES]
                     [--exclude_box EXCLUDE_BOX] [--fit_box FIT_BOX]
                     [--plot_resid_grid] [--plot_initial_gaussian_fit]
                     [--plot_edge_pad] [--compress COMPRESS]
                     [--ignore_negative] [--max_baseline MAX_BASELINE]
                     [--version] [--cite]
```

Named Arguments

--rts_srclist Just save an RTS style srclist - default is to saving both RTS and WODEN srclists

Default: False

--woden_srclist Just save a WODEN style srclist - default is to saving both RTS and WODEN srclists

Default: False

--no_srclist Do not save any srclists

Default: False

--fits_file Name of fits file to fit data from

Default: False

--b1s The beta scale range along the major axis (arcmins). Enter as a lower and upper bound separated by a comma, eg 2,10

Default: False

--b2s The beta scale along the minor (arcmins). Enter as a lower and upper bound separated by a comma, eg 2,10

Default: False

--nmax Maximum value of n1 to include in the basis functions - current maximum possible in the RTS is 100 (The bigger the n1, the higher the resolution of the fitted model)

Default: 10

--save_tag A tag to name the outputs with - defaults to "model"

Default: "model"

--plot_lims Flux limits for the plot - enter as vmin,vmax. Default is min(image),max(image)

Default: False

--freq Frequency (Hz) of the image - defaults to looking for keyword FREQ and associated value

Default: "from_FITS"

--already_jy_per_pixel Add to NOT convert pixels from Jy/beam into Jy/pixel

Default: False

--edge_pad Add empty pixels outside image to stop fitting artefacts outside the desired image - defaults to 0 pixels. Set to desired amount using –edge_pad=number.

Default: 0

--num_beta_values Num of beta params fit over ranges –b1s and –b2s

Default: 5

--exclude_box Any number of areas to exclude from the fit. Specify by user pixel numbers. Add each box as as: x_low,x_high,y_low,y_high. For example, to exclude to areas, one between 0 to 10 in the x range, 10 to 20 in the y range, and the other between 100 to 400 in the x range, 300 to 455 in the y range, enter this on the command line: fit_shapelets.py –exclude_box 0,10,10,20 –exclude_box 100,400,300,455

--fit_box Only fit shapelets to data within the designated box. Add as x_low,x_high,y_low,y_high

Default: False

--plot_resid_grid Add to plot the residuals matrix found for all values of b1 and b2

Default: False

--plot_initial_gaussian_fit Add to plot the intial gaussian fit used to fine b1 and b2

Default: False

--plot_edge_pad When plotting fit results, also plot the fit with an edge pad to check the fit outside the image

Default: False

--compress Add a list of comma separated percentage compression values to apply to the data, e.g. –compress=90,80,70

Default: False

--ignore_negative Add to ignore all negative pixels in image during the fit

Default: False

--max_baseline If no restoring beam information available, use this maximum baseline length in conjunction with the frequency to calculate a resolution to set BMAJ and BMIN

Default: 3000.0

--version Prints the version info and exits

Default: False

--cite Prints a bibtex entry for citing this work (well it will once the paper is published)

Default: False

3.1.2 subtract_gauss_from_image_shamfi.py

Use a table of gaussian properties to subtract gaussian sources from a specfied FITS image. Writes the gaussian subtracted image into a new FITS file. Also writes the subtracted sources into either a WODEN or RTS srclist

```
usage: subtract_gauss_from_image_shamfi.py [-h] [--fits_file FITS_FILE]
                                           [--gauss_table GAUSS_TABLE]
                                           [--output_srclist_name OUTPUT_SRCLIST_NAME]
                                           [--srclist_type SRCLIST_TYPE]
                                           [--outname OUTNAME] [--freq FREQ]
                                           [--no_restore_beam]
```

Named Arguments

--fits_file FITS file to subtract gaussians from

Default: False

--gauss_table Text file containing any number of gaussian source parameters, each line entered as: x_cent(pixels) y_cent(pixels) major(FWHM, arcmins) minor(FWHM, arcmins) pa(deg) int_flux(Jy)

Default: False

--output_srclist_name Name of output srclist containing the subtracted gaussians - will add rts/woden depending on type of srclist to be output

Default: “srclist_gaussian”

--srclist_type Type of srclist to output - options are: woden, rts, both, none. Defaults to both

Default: “both”

--outname Name for gaussian subtracted output FITS file. Defaults to using the input FITS name

Default: "use_fits"

--freq Frequency (Hz) of the image - defaults to looking for keyword FREQ and associated value

Default: "from_FITS"

--no_restore_beam Do not convolve gaussians with restoring beam of CLEANed image

Default: True

3.1.3 mask_fits_shamfi.py

A script to split large radio galaxies into separate portions to be fitted. Outputs as many separated FITS files as directed, as well as plotting them.

```
usage: mask_fits_shamfi.py [-h] [--fits_file FITS_FILE] [--box BOX]
                           [--output_tag OUTPUT_TAG]
```

Named Arguments

--fits_file Name of fits file to split

Default: False

--box Any number of areas to fit a gaussian to - specify by user pixel numbers. Add each box as as: x_low,x_high,y_low,y_high. For example, to fit two gaussians, one between 0 to 10 in the x range, 10 to 20 in the y range, and the other between 100 to 400 in the x range, 300 to 455 in the y range, enter this on the command line: mask_fits.py –box 0,10,10,20 –box 100,400,300,455

--output_tag Tag name to add to outputs

Default: "mask"

3.1.4 combine_srclists_shamfi.py

Combines multiple srclists (RTS or WODEN) into one source. Assumes that each srclist only contains one SOURCE currently

```
usage: combine_srclists_shamfi.py [-h] [--srclist SRCLIST] [--outname OUTNAME]
                                  [--source_name SOURCE_NAME]
```

Named Arguments

--srclist Any number of srclists (of either RTS or WODEN, not a combination) to combine. Repeat the argument as necessary, e.g. –srclist=srclist_one.txt –srclist=srclist_two.txt –srclist=srclist_three.txt

--outname Name for output srclist - defaults to "srclist_combined"+woden or rts

Default: "srclist_combined"

--source_name Name for combined source in srclist - defalts to "combined_source"

Default: "combined_source"

3.1.5 convert_srclists_shamfi.py

Converts a srclist between RTS or WODEN formats Should automatically detect if base catalouge is RTS or WODEN format. Currently assumes only one SOURCE is in the srclist i.e. an output from shamfi.py or combine_srclists_shamfi.py

```
usage: convert_srclists_shamfi.py [-h] [--srclist SRCLIST] [--rts2woden]
                                  [--woden2rts] [--outname OUTNAME]
```

Named Arguments

--srclist srclist to be converted

Default: False

--rts2woden Optional - specify to convert from RTS to WODEN type srclist

Default: False

--woden2rts Optional - specify to convert RTS type srclist

Default: False

--outname Name for output srclist - defaults to "srclist_coverted"+woden or rts

Default: "convert"

CHAPTER 4

Modules

4.1 Module Documentation

4.1.1 read_FITS_image

class read_FITS_image.**FITSInformation**(*fitsfile*)

This class reads in data and metadata from a FITS image file, and stores it for use in generating relevant coord systems and data when fitting shapelets. Expects an CLEANed image, testing against WSClean outputs. See attributes for futher functionality, and variables for what can be accessed.

Parameters **fitsfile** (*str*) – name of a FITS image file to gather relevant data and metadata from

Variables

- **fitsfile** (*str*) – the given fitsfile parameter
- **header** – an astropy header instance of the FITS file
- **data** (*array*) – a 2D numpy array of the image data
- **read_data** (*bool*) – True if successful in reading data
- **ra_reso** (*float*) – RA resolution (deg)
- **dec_reso** (*float*) – Dec resolution (deg)
- **pix_area_rad** (*float*) – pixel area (ster radian)
- **len1** (*int*) – number of pixels in NAXIS1
- **len2** (*int*) – number of pixels in NAXIS2
- **wcs** – an astropy WCS instance based on self.header
- **flat_data** (*array*) – self.data.flatten()
- **bmaj** (*float*) – restoring beam major axis - if keyword BMAJ not in header, set to None

- **bmin** (*float*) – restoring beam minor axis - if keyword BMIN not in header, set to None
- **solid_beam** (*float*) – solid beam angle: (pi*self.bmaj*self.bmin) / (4*log(2))
- **convert2pixel** (*float*) – conversion factor need to convert Jy/beam to Jy/pixel
- **got_convert2pixel** (*bool*) – True if all metadata to create self.convert2pixel was available

covert_to_jansky_per_pix()
Coverts data in Jy/beam to Jy/pixel as stored in self.data and self.flat_data

create_restoring_kernel()
Use the FITS header metadata to create 2D restoring Gaussian beam kernel with the height and width of the kernel set to 8 times larger than BMAJ

Returns 2D array of the restoring beam at the same resolution of the image. Also stored as self.rest_gauss_kern

Return type array

get_radec_edgepad(*edge_pad=False*)
Use FITS information to form values of RA, DEC for all pixels in this image If specified, edge pad the data with zeros, and add extra RA and DEC range accordingly.

Parameters **edge_pad** (*int*) – If True, edge pad the data with the specified number of pixels

Variables

- **ras** (*array*) – RA values of all pixels in image, flattened into a 1D array (radians)
- **decs** (*array*) – DEC values of all pixels in image, flattened into a 1D array (radians)

4.1.2 shapelet_coords

class shapelet_coords.**ShapeletCoords**(*fits_data=False*)
Takes a shamfi.read_FITS_image class, and uses it to generate a cartesian coord system to fit shapelets in. Include attributes to flux centre the coord system, and to mask the data as required. See the attribute docs below for details.

Parameters **fits_data** – a shamfi.read_FITS_image.FITSInformation class that has already been initialised

Variables **fits_data** – The supplied shamfi.read_FITS_image.FITSInformation class parameter

find_good_pixels(*fit_box=False*, *exclude_box=False*, *ignore_negative=False*)
Generates a mask to ignore pixels when fitting the shapelets based on the given arguments

Parameters

- **fit_box** (*str*) – only use the pixels as bounded by fit_box = "low_x,high_x,low_y,high_y" (pixel coords)
- **exclude_box** (*list*) – exclude pixels bounded by the boxes defined in exclude_box, e.g. exclude_box=["low_x,high_x,low_y,high_y", "low_x,high_x,low_y,high_y"]
- **ignore_negative** (*bool*) – mask all pixels with negative values in self.fits_data.flat_data

Variables

- **pixel_inds_to_use** (*array*) – array of pixel indexes to use
- **negative_pix_mask** (*array*) – index of positive data pixels

fit_gauss_and_centre_coords(*b1_max=False*, *b2_max=False*)
Try and fit a Gaussian to the image, using the flux weighted central pixel location, and maximum b1 and b2 values, as an initial parameter estimate

Parameters

- **b1_max** (*float*) – maximum beta scaling for major axis direction
- **b2_max** (*float*) – maximum beta scaling for minor axis direction

Variables

- **ra_cent** (*float*) – RA value of the flux weighted central pixel
- **dec_cent** (*float*) – Dec value of the flux weighted central pixel
- **ras** (*array*) – all RA values, zeroed on the fluxed weighted pixel (deg)
- **decs** (*array*) – all Dec values, zeroed on the fluxed weighted pixel (deg)

radec2xy(*b1*, *b2*, *crop=False*)
Transforms the RA/DEC coords system into the shapelet x/y system for given b1,b2 parameters, and the self.pa rotation angle. Also optinally applies the self.pixel_inds_to_use cut to only return the pixels to be used in the shapelet fit

Parameters

- **b1** (*float*) – beta parameter to scale the x-axis (radians)
- **b2** (*float*) – beta parameter to scale the y-axis (radians)
- **crop** (*bool*) – If True, apply self.pixel_inds_to_use to the transformation

Return xrot,yrot The x,y coords in 1D numpy arrays

Return type arrays

shapelet_coords.**twoD_Gaussian**(*xy*, *amplitude*, *xo*, *yo*, *sigma_x*, *sigma_y*, *theta*)
Creates a model for a 2D Gaussian, by taking two 2D coordinate arrays in x,y. Returns a flattened array of the model to make fitting more straight forward

Parameters

- **xy** (*list*) – list containing two 2D numpy arrays A list containing the x and y coordinates to calculate the gaussian at. x and y are separated into individual 2D arrays. xy = [x(2D), y(2D)]
- **amplitude** (*float*) – float Amplitude to scale the Gaussian by
- **xo** (*float*) – float Value of the central x pixel
- **yo** (*float*) – float Value of the central y pixel
- **sigma_x** (*float*) – float Sigma value for the x dimension
- **sigma_y** (*float*) – float Sigma value for the y dimension
- **theta** (*float*) – float Rotation angle (radians)

Return gaussian.flatten() A 2D gaussian model, flattened into a 1D numpy array

Return type array

4.1.3 shapelets

class shapelets.**FitShapelets**(*fits_data=False*, *shpcoord=False*)

This class takes in FITS data and fitting parameters, and sets up and performs shapelet model fitting and generation. The main work-horse of the SHAMFI package

Parameters

- **fits_data** (*shamfi.read_FITS_image.FITSInformation instance*) – A FITSInformation class containing the data to be fit
- **shpcoord** (*shamfi.shapelet_coords.ShapeletCoords instance*) – A ShapeletCoords class containing the shapelet coordinate system

do_grid_search_fit(*b1_grid*, *b2_grid*, *nmax*, *pa=False*, *convolve_kern=False*, *save_FITS=True*, *save_tag='shapelet'*)

Do a grid search over all b1, b2 values specified in b1_grid, b2_grid, fitting all basis functions up to nmax, donig a least squares minimisation for each combination of b1, b2 to generate models

Parameters

- **b1_grid** (*array*) – A range of major axis beta scaling parameters to fit over
- **b2_grid** – A range of minor axis beta scaling parameters to fit over
- **nmax** (*int*) – The maximum order of basis function to generate up to
- **pa** (*float*) – If provided, use this postion angle to rotate the basis functions, instead of that found when fitting a Gaussian using shamfi.shapelet_coords.ShapeletCoords.fit_gauss_and_centre_coords
- **convolve_kern** (*2D numpy array*) – If provided, use this convolution kernel instead of the restoring beam of the CLEANed image
- **save_FITS** (*bool*) – Save the fitted shapelet model image to a FITS file
- **save_tag** (*string*) – A tag to add into the file name to save the plot to

do_grid_search_fit_compressed(*compress_value*, *save_FITS=True*, *save_tag='shapelet'*)

Do a grid search over all b1, b2 values for a given compression value. Can only be run after fitting the full model, via *self.do_grid_search_fit* so all b1,b2,nmax options have already been set and stored internally to the class.

Parameters

- **compress_value** (*float*) – A value to compress (truncate) the fit results to (percentage, e.g. 80 for 80%)
- **save_FITS** (*bool*) – Save the fitted shapelet model image to a FITS file
- **save_tag** (*string*) – A tag to add into the file name to save the plot to

find_flux_order_of_basis_functions()

After fitting models, this function orders the fitted basis functions by absolution value in image space, to find those that contribute the most flux. To do this an 'A' matrix has to be generated to create an image

Variables

- **basis_sums** (*array*) – an array containing the sum of the flux of each basis function
- **sums_order** (*array*) – an array of the argsorted index of the sums of the flux of each basis function

save_srclist(*save_tag='shapelet'*, *rts_srclist=True*, *woden_srclist=True*)

Uses the best fitted parameters and creates an RTS/WODEN style srclist with them, saved as text files

Parameters

- **save_tag** (*string*) – A tag to add into the file name to save the plot to
- **rts_srclist** (*bool*) – If True, save a sky model compatible with the RTS (Mitchell et al, 2008)
- **woden_srclist** (*bool*) – If True, save a sky model compatible with WODEN (Line et al, 2020)

shapelets.**gen_A_shape_matrix**(*n1s=None*, *n2s=None*, *xrot=None*, *yrot=None*, *nmax=None*, *b1=None*, *b2=None*, *convolve_kern=False*, *shape=False*)

Generates the 'A' matrix in Ax = b when fitting shapelets, where:

- b = 1D matrix of data points to mode
- x = 1D matrix containing coefficients for basis functions
- A = 2D matrix containg shapelet basis function values

Generates basis function values using a lookup table method

Parameters

- **n1s** (*array*) – The first orders of the basis functions to generate
- **n2s** (*array*) – The second orders of the basis function to generate
- **xrot** (*numpy array*) – 1D array of the x-coords to generate the basis functions at
- **yrot** (*numpy array*) – 1D array of the y-coords to generate the basis functions at
- **b1** (*float*) – The major axis beta scaling parameter (radians)
- **b2** (*float*) – The minor axis beta scaling parameter (radians)
- **convolve_kern** (*2D numpy array*) – A kernel to convolve the basis functions with - if modelling a CLEANed image this should be the restoring beam
- **shape** (*tuple*) – The 2D shape of the image being modelled, needed if convolving with a kernel

Returns

- **n1s** (*array*) – The first orders of the basis functions to generate
- **n2s** (*array*) – The second orders of the basis function to generate
- **A_shape_basis** (*2D array*) – The generated 2D 'A' matrix

shapelets.**gen_A_shape_matrix_direct**(*n1s=None*, *n2s=None*, *xrot=None*, *yrot=None*, *b1=None*, *b2=None*, *convolve_kern=False*, *shape=False*)

Generates the 'A' matrix in Ax = b when fitting shapelets, where:

- b = 1D matrix of data points to mode
- x = 1D matrix containing coefficients for basis functions
- A = 2D matrix containg shapelet basis function values

Generates the basis functions directly using scipy

Parameters

- **n1s** (*array*) – The first orders of the basis functions to generate
- **n2s** (*array*) – The second orders of the basis function to generate

- **xrot** (*numpy array*) – 1D array of the x-coords to generate the basis functions at
- **yrot** (*numpy array*) – 1D array of the y-coords to generate the basis functions at
- **b1** (*float*) – The major axis beta scaling parameter (radians)
- **b2** (*float*) – The minor axis beta scaling parameter (radians)
- **convolve_kern** (*2D numpy array*) – A kernel to convolve the basis functions with - if modelling a CLEANed image this should be the restoring beam
- **shape** (*tuple*) – The 2D shape of the image being modelled, needed if convolving with a kernel

Returns

- **checked_n1s** (*array*) – The first orders of the basis functions to generate - checks for any errors when generating the basis functions and omits those n1,n2 values
- **checked_n2s** (*array*) – The second orders of the basis function to generate - checks for any errors when generating the basis functions and omits those n1,n2 values
- **A_shape_basis** (*2D array*) – The generated 2D 'A' matrix

shapelets.**gen_shape_basis**(*n1=None*, *n2=None*, *xrot=None*, *yrot=None*, *b1=None*, *b2=None*, *convolve_kern=False*, *shape=False*)

Generates the shapelet basis function for given n1,n2,b1,b2 parameters, using lookup tables and interpolation, at the given coords xrot,yrot. b1,b2 should be in radians.

Parameters

- **n1** (*int*) – The first order of the basis function to generate
- **n2** (*int*) – The second order of the basis function to generate
- **xrot** (*numpy array*) – 1D array of the x-coords to generate the basis functions at
- **yrot** (*numpy array*) – 1D array of the y-coords to generate the basis functions at
- **b1** (*float*) – The major axis beta scaling parameter (radians)
- **b2** (*float*) – The minor axis beta scaling parameter (radians)
- **convolve_kern** (*2D numpy array*) – A kernel to convolve the basis functions with - if modelling a CLEANed image this should be the restoring beam
- **shape** (*tuple*) – The 2D shape of the image being modelled, needed if convolving with a kernel

Returns **basis** – A 1D array of the values of the basis function

Return type numpy array

shapelets.**gen_shape_basis_direct**(*n1=None*, *n2=None*, *xrot=None*, *yrot=None*, *b1=None*, *b2=None*, *convolve_kern=False*, *shape=False*)

Directly generates the shapelet basis function for given n1,n2,b1,b2 parameters, at the given coords xrot,yrot. b1,b2 should be in radians. Uses scipy to generate factorials and hermite polynomials

Parameters

- **n1** (*int*) – The first order of the basis function to generate
- **n2** (*int*) – The second order of the basis function to generate
- **xrot** (*numpy array*) – 1D array of the x-coords to generate the basis functions at
- **yrot** (*numpy array*) – 1D array of the y-coords to generate the basis functions at

- **b1** (*float*) – The major axis beta scaling parameter (radians)
- **b2** (*float*) – The minor axis beta scaling parameter (radians)
- **convolve_kern** (*2D numpy array*) – A kernel to convolve the basis functions with - if modelling a CLEANed image this should be the restoring beam
- **shape** (*tuple*) – The 2D shape of the image being modelled, needed if convolving with a kernel

Returns **basis** – A 1D array of the values of the basis function

Return type numpy array

shapelets.**interp_basis**(*xrot=None*, *yrot=None*, *n1=None*, *n2=None*)

Uses basis lookup tables to generate 2D shapelet basis function for given xrot, yrot coords and n1,n2 orders. Does NOT include the b1,b2 normalisation, this is applied by the function gen_shape_basis

Parameters

- **n1** (*int*) – The first order of the basis function to generate
- **n2** (*int*) – The second order of the basis function to generate
- **xrot** (*numpy array*) – 1D array of the x-coords to generate the basis functions at
- **yrot** (*numpy array*) – 1D array of the y-coords to generate the basis functions at

Returns **basis** – A 1D array of the values of the basis function

Return type numpy array

4.1.4 image_manipulation

image_manipulation.**subtract_gauss**(*ind*, *x*, *y*, *major*, *minor*, *pa*, *flux*, *ax1*, *ax2*, *ax3*, *fig*, *fits_data*, *convolve=True*)

Takes a 2D CLEAN restored image array (data) and subtracts a Gaussian using the specified parameters. The subtracted Gaussian is convolved with the restoring beam kernel to ensure the correct Gaussian properties. Plots the subtracted gaussians with postage stamps before and after subtraction

Parameters

- **ind** (*int*) – index of the Gaussian being fit, used to correctly title the outputs when plotting
- **x** (*int*) – The central x coord of the Gaussian to subtract
- **y** (*int*) – The central y coord of the Gaussian to subtract
- **major** (*float*) – The major axis of the Gaussian to subtract (arcmin)
- **minor** (*float*) – The minor axis of the Gaussian to subtract (arcmin)
- **pa** (*float*) – The pa of the Gaussian to subtract (deg)
- **flux** (*float*) – The integrated flux density of the Gaussian to subract (Jy)
- **ax1** (*matplotlib.pyplot.figure.add_subplot instance*) – The axis to plot the data before subtraction on
- **ax2** (*matplotlib.pyplot.figure.add_subplot instance*) – The axis to plot the Gaussian to be subtracted on
- **ax3** (*matplotlib.pyplot.figure.add_subplot instance*) – The axis to plot the data after subtraction on
- **fig** (*matplotlib.pyplot.figure*) – The figure to plot on

- **fits_data** (*shamfi.read_FITS_image.FITSInformation instance*) – A `FITSInformation` class containing the image data
- **convolve** (*boolean*) – By default, gaussians to subtract are convolved by the restoring beam to account correctly for resolution effects. Set this to False if you know you gaussian parameters work with the current image. Default convolve=True.

Returns

- **data** (*2D numpy array*) – 2D numpy image array with the Gaussian subtracted
- **ra** (*float*) – The RA of the the subtracted Gaussian (deg)
- **dec** (*float*) – The Dec of the the subtracted Gaussian (deg)

4.1.5 shamfi_plotting

`shamfi_plotting.`**`add_colourbar`**(*fig=None*, *ax=None*, *im=None*, *label=False*, *top=False*)

Adds a colourbar (colorbar, fine) in a nice way to a subplot

Parameters

- **fig** (*matplotlib.pyplot.figure instance*) – The figure that the plot lives on
- **ax** (*matplotlib.pyplot.figure.add_subplot instance*) – The axis to append a colorbar to
- **im** (*ax.imshow output*) – The output of imshow to base the colourbar on
- **label** (*string*) – Optional - add a label to the colorbar
- **top** (*Bool*) – Optional - put the colorbar above the axis instead of to the right

`shamfi_plotting.`**`do_subplot`**(*fig*, *ax*, *data*, *label*, *vmin*, *vmax*)

Plots a 2D numpy array (data) with a colorbar. Optionally can set vlim and vmin to control the colour scale

Parameters

- **fig** (*matplotlib.pyplot.figure instance*) – The figure that the plot lives on
- **ax** (*matplotlib.pyplot.figure.add_subplot instance*) – The axis to plot on
- **data** (*2D numpy array*) – The data to plot
- **label** (*string*) – The title for the subplot
- **vmin** (*float*) – Optional - lower value for the colour scale, passed to imshow
- **vmax** (*float*) – Optional - upper value for the colour scale, passed to imshow

`shamfi_plotting.`**`make_masked_image`**(*flat_data*, *shapelet_fitter*)

Create a 2D array for plotting purposes, where all the pixels that were originally masked in the fit are set to NaN so they don't show during imshow

Parameters

- **flat_data** (*array*) – numpy array of data to mask
- **shapelet_fitter** (*shamfi.shapelets.FitShapelets instance*) – The `FitShapelets` used to run the shapelet fitting

Returns **masked_data** – A 2D array in the original shape of the image to be fitted, with the cuts that were applied during fitted set to NaNs for plotting

Return type 2D array

shamfi_plotting.**plot_full_shamfi_fit**(*shapelet_fitter*, *save_tag*, *plot_edge_pad=False*)
Take a FitShapelets class that has been run, and plot the results. Plots the data with top left, fit top right, and residuals bottom left. Optionally, plot an edge padded version of the final fit bottom right - useful to check that unconstrained areas outside of the fitting region haven't ended up with crazy modelled flux.

Parameters

- **shapelet_fitter** (*shamfi.shapelets.FitShapelets instance*) – The FitShapelets used to run the shapelet fitting
- **save_tag** (*string*) – A tag to add into the file name to save the plot to
- **plot_edge_pad** (*bool*) – If True, plot a version of the model using edge padded coords, to check for run away modelling outside the fitting area

shamfi_plotting.**plot_gaussian_fit**(*shpcoord*, *save_tag*)
Plot a contour of an intial gaussian fit over an image, using the results of a ShapeletCoords

Parameters

- **shpcoord** (*shamfi.shapelet_coords.ShapeletCoords instance*) – The FitShapelets used to run the shapelet fitting
- **save_tag** (*string*) – A tag to add into the file name to save the plot to

shamfi_plotting.**plot_grid_search**(*shapelet_fitter*, *save_tag*)
Plot a matrix of the image based residuals as a function of b1, b2

Parameters

- **shapelet_fitter** (*shamfi.shapelets.FitShapelets instance*) – The FitShapelets used to run the shapelet fitting
- **save_tag** (*string*) – A tag to add into the file name to save the plot to

4.1.6 srclists

class srclists.**Component_Info**
Class to contain an RTS source information

Variables

- **comp_type** (*str*) – The component type: either POINT, GAUSSIAN, or SHAPELET
- **pa** (*float*) – Position Angle of the component
- **major** (*float*) – Major angle of the component
- **minor** (*float*) – Minor angle of the component
- **shapelet_coeffs** (*list*) – List to contain lists of shapelet coeffs if the source is a SHAPELET

class srclists.**RTS_source**
Class to contain an RTS source information

Variables

- **name** (*str*) – Source name
- **ras** (*list*) – List of all RA for all components in this source
- **decs** (*list*) – List of all Dec for all components in this source
- **flux_lines** (*list*) – List to contain list of flux lines for all components in this source

- **component_infos** (*list*) – List to contain *Component_Info* classes for all components in this source

srclists.**check_rts_or_woden_get_lines**(*filename*)
Opens the file at path *filename*, and splits into lines by return carriage. Ignores all lines commented with #. Uses the first line of the file to determine if this is an RTS or WODEN style srclist

Parameters **filename** (*string*) – Path to a text file

Returns

- **type** (*string*) – The type of srclist, either "woden" or "rts"
- **lines** (*list*) – The lines of the file as string, split by return carriage, in a list

srclists.**get_RTS_sources**(*srclist*, *all_RTS_sources*)
Takes a path to an RTS srclist, breaks it up into SOURCES, populates this information into RTS_source classes and appends them to all_RTS_sources list

Parameters

- **srclist** (*string*) – Path to a texfile of an RTS srclist
- **all_RTS_sources** (*list*) – All sources found in *srclist* are used to populate an *RTS_source* class, and appended to *all_RTS_sources*

Returns **all_RTS_sources** – The original *all_RTS_sources* list, with any new *RTS_source* s appended

Return type list of *RTS_source*

srclists.**write_singleRTS_from_RTS_sources**(*RTS_sources*, *outname*, *name='combined_name'*)
Takes a list RTS_sources containg RTS_source classes, and writes them out into a single SOURCE RTS srclist of name outname

srclists.**write_woden_component_as_RTS**(*lines*, *outfile*, *name=False*)
Take in a number of text lines in the WODEN format and writes them out to *outfile* in the RTS format

Parameters

- **lines** (*list*) – A list of strings, each a line from a WODEN style srclist
- **outfile** (*open(filename) instance*) – An opened textfile to write outputs to
- **name** (*string*) – If a name is supplied, this is the first component of a SOURCE, which requires extra formatting in an RTS srclist

srclists.**write_woden_from_RTS_sources**(*RTS_sources*, *outname*)
Takes a list of *RTS_source* classes and uses the to write a WODEN style srclist called *outname*

Parameters

- **RTS_sources** (*list*) – A list of *RTS_source* s to write out to a WODEN style srclist
- **outname** (*string*) – Path to save the output WODEN srclist text file to

4.1.7 check_files

check_files.**check_file_exists**(*filename*, *argname*)
Checks if a file exists, and throws an error and exits if that file does not exist

Parameters

- **filename** (*string*) – Path to a file

- **argname** (*string*) – The argument that was passed to a script, for error checking purposes

Returns **filename** – Path to a file

Return type string

check_files.**check_if_fits_extension**(*name*)
Checks if *name* ends in ".fits", appends if not

Parameters **name** (*string*) – String to check

Returns **outname** – A string that definitely ends in ".fits"

Return type string

check_files.**check_if_txt_extension**(*name*)
Checks if string *name* ends in ".txt", appends if not

Parameters **name** (*string*) – String to check

Returns **outname** – A string that definitely ends in ".txt"

Return type string

4.1.8 git_helper

git_helper.**get_commandline_output**(*command_list*)
Takes a command line entry separated into list entries, and returns the output from the command line as a string

Parameters **command_list** (*list of strings*) – list of strings that when combined form a coherent command to input into the command line

Returns **output** – the output result of running the command

Return type string

git_helper.**get_gitdict**()
Get the git dictionary that was created by setup.py and/or during pip install and was stored in a json file

Returns **git_dict** – A dictionary containing git information with keywords: describe, date, branch

Return type dictionary

git_helper.**make_gitdict**()
Makes a dictionary containing key git information about the repo by running specific commands on the command line

Returns **git_dict** – A dictionary containing git information with keywords: describe, date, branch

Return type dictionary

git_helper.**print_version_info**(*script_loc*)
Takes the location of the script calling this function, and prints out useful git information

Parameters **script_loc** (*string*) – Absolute path of the file calling this function

git_helper.**write_git_header**(*outfile*)
Takes a textfile and writes a git summary at the top, commented with #

Parameters **outfile** (*text file instance*) – Text file that gets useful version information appended to it

Python Module Index

Index

S

T

W