ragavi

Release 0.2.0

Lexy Andati

Aug 26, 2019

CONTENTS:

CHAPTER

ONE

INTRODUCTION

1.1 Radio Astronomy Gains and Visibility Inspector [ragavi]

Ragavi[1] is a python based software built for visualisation of radio astronomy data reduction by-products such as gains as well as visibility data. As the name implies, it comprises two aspects, a gain plotter aliased by `ragavi-gains` or `ragavi`, which may be used as a command line tool and within a notebook environment (Jupyter), and a visibility plotter aliased by `ragavi-vis`, which is **only available** as a command line tool.

1.2 Motivation

The main motivation for `ragavi` is introduced an in interactivity aspect to radio astronomy oriented plots traditionally be produced as static images in formats such as PNG, JPEG and SVG. This is achieved though a python package known as Bokeh[2] which has the capability of producing stand alone HTML based interactive plots which are JavaScript oriented.

The output interactive plots enable actions such as

- Panning: moving through a plot "frame by frame"
- Zooming: focusing on a smaller or larger area
- Scaling: increasing plot dimension depending on available window size
- Selection:

among others, without requiring any redraw actions or special software to view plots. All that is required to view a plot produced by `ragavi` is a *web-browser*.

1.3 Limitations

`Ragavi` is still under development.

Main dependencies for this project

- Daskms[3]
- Bokeh[4]

[1] https://github.com/ratt-ru/ragavi
[2] https://bokeh.pydata.org/en/latest/index.html
[3] https://xarray-ms.readthedocs.io/en/latest/
[4] https://bokeh.pydata.org/en/latest/index.html

- Datashader[5]

[5] http://datashader.org/

CHAPTER

TWO

INSTALLATION & USAGE

2.1 Installation

`Ragavi` is available on *PYPI* and can be installed via pip or from source.

To install via pip, type on your terminal

```
$ pip install ragavi
```

This is the recommended installation method and will install the latest release.

To install from source:

```
$ git clone https://github.com/ratt-ru/ragavi.git
$ cd ragavi
$ pip install .
```

To check whether installation was successful

```
$ ragavi-gains -h
$ ragavi-vis -h
```

This will bring up a help menu for `ragavi-gains` and `ragavi-vis` respectively.

2.2 Usage

Note: `ragavi` namespace will soon change to `ragavi-vis`

```
$ ragavi-gains -h
```

To use ragavi gain plotter

```
$ ragavi-gains -t /path/to/your/table -g table_type (K / B/ F/ G/ D)
```

Multiple tables can be plotted on the same document simply by adding them in a space separated list to the `-t` / `--table` switch. They must however be accompanied by their respective gain table type in the `-g` switch. e.g

```
$ ragavi -t delay/table/1/ bandpass/table/2 flux/table/3 -g K B F
```

For the visibility plotter, the name-space `ragavi-vis` is used. Help can be obtained by running

```
$ ragavi-vis -h
```

To run ragavi-vis, the arguments `--table`, `--xaxis` and `--yaxis` are basic requirements e.g.

```
$ ragavi-vis --table /my/measurement/set --xaxis time --yaxis amplitude
```

CHAPTER

THREE

RAGAVI-GAINS

To be used for gain table visualisation.

Currently, gain tables supported for visualisation by `ragavi-gains` are :

- Flux calibration (F tables)
- Bandpass calibration (B tables)
- Delay calibration (D tables)
- Gain calibration (G tables)
- D-Jones Leakage tables (D tables)

Mandatory fields are `--table`, `--gain_type`

If a field name is not specified to `ragavi-vis` all the fields will be plotted by default

It is possible to place multiple gain table plots of different [or same] types into a single HTML document using `ragavi` This can be done by specifying the table names and gain types as space separate list as below

```
$ragavi-gains --table table/one/name table/two/name table/three/name table/four/name -
↪-gain_types B G D F --fields 0
```

This will yield an output HTML file, with plots in the order

–table	field	gain
table1	0	B
table2	0	G
table3	0	D
table4	0	F

To use `ragavi-gains` in a notebook environment, run in a notebook cell

```
from ragavi.ragavi import plot_table

#specifying function arguments
args = dict(mytabs=[], gain_types=[], cmap='viridis', doplot='ri', corr=1, ant='1,2,3,
↪9,10')

#inline plotting will be done
plot_table(**args)
```

3.1 API

class ragavi.ragavi.**DataCoreProcessor**(*xds_table_obj*, *ms_name*, *gtype*, *fid=None*, *antenna=None*, *doplot='ap'*, *corr=0*, *flag=True*)

Process gain table data into forms desirable for visualisation.

Parameters

- **antenna** (str) –
- **corr** (int) – Correlation index to plot
- **doplot** (str) – Kind of plot to do. Default is ap
- **flag** (bool) – To flag or not. Default is True
- **fid** (int) – Field id
- **gtype** (str) – Gain table type
- **ms_name** (str) – Name / path to the table
- **xds_table_obj** (xarray.Dataset) – Table object from which data will be extracted

act()

Activate the *ragavi.ragavi.DataCoreProcessor.blackbox()*

blackbox(*xds_table_obj*, *ms_name*, *gtype*, *fid=None*, *antenna=None*, *doplot='ap'*, *corr=0*, *flag=True*)

Get raw input data and churn out processed data. Takes in all inputs from the instance initialising object

This function incorporates all function in the class to get the desired result. It performs:

- xaxis data and error data acquisition
- xaxis data and error preparation and processing
- yaxis data and error data acquisition
- yaxis data and error preparation and processing

Returns **d** (collections.namedtuple) – A named tuple containing all processed x-axis data, errors and label, as well as both pairs of y-axis data, their error margins and labels. Items from this tuple can be gotten by using the dot notation.

ragavi.ragavi.**plot_table**(*mytabs*, *gain_types*, *fields*, ***kwargs*)

Plot gain tables within Jupyter notebooks.

Parameters

- **corr** (int, optional) – Correlation index to plot (usually just 0 or 1, default = 0)',default=0)
- **doplot** (str, optional) – Plot complex values as amp and phase (ap) or real and imag (ri). Default is 'ap'.
- **fields** (str, optional) – Field ID / Name (list of field ids or name) to plot. Default is all.
- **gain_types** (str, list) – Cal-table (list of caltypes) type to be plotted. Can be either 'B'-bandpass, 'D'- D jones leakages, G'-gains, 'K'-delay or 'F'-flux. Default is none
- **image_name** (str, optional) – Output image name. Default is of the form table_corr_doplot_fields

- **mycmap** (*str, optional*) – Matplotlib colour map to use for antennas. Default is coolwarm
- **mytabs** (`str` or `list` required) – The table (list of tables) to be plotted.
- **plotants** (`str, optional`) – Plot only this antenna, or comma-separated string of antennas. Default is all
- **t0** (`int, optional`) – Minimum time [in seconds] to plot. Default is full range
- **t1** (`int, optional`) – Maximum time [in seconds] to plot. Default is full range

CHAPTER

FOUR

RAGAVI-VIS

To be used for visibility plotting. Supported arguments are

For x axis:

- amplitude
- antenna1
- antenna2
- frequency
- phase
- real
- scan
- time
- uvdistance
- uvwave

For y-axis:

- amplitude
- phase
- real
- imaginary

Iterations:

- Correlations (corr)
- Scan (scan)
- Spectral windows (spw)

Mandatory arguments are `--xaxis`, `--yaxis`, `--table`.

4.1 API

class ragavi.visibilities.**DataCoreProcessor**(*xds_table_obj*, *ms_name*, *xaxis*, *yaxis*, *chan=slice(0, None, None)*, *corr=slice(0, None, None)*, *ddid=slice(0, None, None)*, *datacol='DATA'*, *flag=True*)

Process Measurement Set data into forms desirable for visualisation.

Parameters

- **chan** (slice) – Channels that will be selected. Defaults to all
- **corr** (int) – Correlation indices to be selected. Defaults to all
- **datacol** (str) – Data column to be selected. Defaults to 'DATA'
- **ddid** (slice) – Spectral windows to be selected. Defaults to all
- **flag** (bool) – To flag or not. Defaults to True
- **ms_name** (str) – Name / path to the Measurement Set
- **xds_table_obj** (xarray.Dataset) – Table object from which data will be extracted
- **xaxis** (str) – x-axis to be selected for plotting
- **yaxis** (str) – y-axis to be selected for plotting

act()

Activate the *ragavi.ragavi.DataCoreProcessor.blackbox()*

blackbox(*xds_table_obj*, *ms_name*, *xaxis*, *yaxis*, *chan=slice(0, None, None)*, *corr=0*, *datacol='DATA'*, *ddid=None*, *flag=True*)

Get raw input data and churn out processed data.

This function incorporates all function in the class to get the desired result. Takes in all inputs from the instance initialising object. It performs:

- xaxis data and error data acquisition
- xaxis data and error preparation and processing
- yaxis data and error data acquisition
- yaxis data and error preparation and processing

Returns **d** (collections.namedtuple) – A named tuple containing all processed x-axis data, errors and label, as well as both pairs of y-axis data, their error margins and labels. Items from this tuple can be gotten by using the dot notation.

ragavi.visibilities.**hv_plotter**(*x*, *y*, *xaxis*, *xlab=''*, *yaxis='amplitude'*, *ylab=''*, *color='blue'*, *xds_table_obj=None*, *ms_name=None*, *iterate=None*)

Responsible for plotting in this script.

This is responsible for:

- Selection of the iteration column. ie. Setting it to a categorical column
- Creating the image callback to Datashader
- Creating Bokeh canvas onto which image will be placed
- Calculation of maximums and minimums for the plot
- Formatting fonts, axes and titles

Parameters

- **x** (xarray.DataArray) – x data to plot
- **y** (xarray.DataArray) – y data to plot
- **xaxis** (str) – xaxis selected for plotting
- **xlab** (str) – Label to appear on x-axis
- **yaxis** (str) – yaxis selected for plotting
- **ylab** (str) – Label to appear on y-axis
- **iterate** (str) – Column in the dataset over which to iterate. It should be noted that currently iteration is done using colors to denote the different parts of the iteration axis. These colors are explicitly selected in the code and are cycled through. i.e repetitive. This option is akin to the colorise_by function in CASA.
- **ititle** (str) – Title to appear incasea of iteration
- **color** (str, colormap, itertools.cycler) – Color scheme to be used in the plot. It could be a string containing a color, a matplotlib or bokeh or colorcet colormap of a cycler containing specified colors.
- **xds_table_obj** (xarray.Dataset) – Dataset object containing the columns of the MS. This is passed on in case there are items required from the actual dataset.
- **ms_nmae** (str) – Name or [can include path] to Measurement Set

Returns **fig** (bokeh.plotting.figure)

CHAPTER

FIVE

UTILITIES

Some utilities useful internally and externally from `ragavi`

`ragavi.vis_utils.`**`calc_amplitude`**(*ydata*)
Convert complex data to amplitude (absolute value)

Parameters **`ydata`** (`xarray.DataArray`) – y axis data to be processed

Returns **amplitude** (`xarray.DataArray`) – `ydata` converted to an amplitude

`ragavi.vis_utils.`**`calc_imaginary`**(*ydata*)
Extract imaginary part from complex data

Parameters **`ydata`** (`xarray.DataArray`) – y-axis data to be processed

Returns **imag** (`xarray.DataArray`) – Imaginary part of `ydata`

`ragavi.vis_utils.`**`calc_phase`**(*ydata*, *wrap=True*)
Convert complex data to angle in degrees

Parameters

- **`wrap`** (`bool`) – whether to wrap angles between 0 and 2pi
- **`ydata`** (`xarray.DataArray`) – y-axis data to be processed

Returns **phase** (*xarray.DataArray*) – `ydata` data converted to degrees

`ragavi.vis_utils.`**`calc_real`**(*ydata*)
Extract real part from complex data

Parameters **`ydata`** (`xarray.DataArray`) – y-axis data to be processed

Returns **real** (`xarray.DataArray`) – Real part of `ydata`

`ragavi.vis_utils.`**`calc_uvdist`**(*uvw*)
Calculate uv distance in metres

Parameters **`uvw`** (`xarray.DataArray`) – UVW column from measurement set

Returns **uvdist** (`xarray.DataArray`) – uv distance in meters

`ragavi.vis_utils.`**`calc_uvwave`**(*uvw*, *freq*)
Calculate uv distance in wavelength for availed frequency. This function also calculates the corresponding wavelength. Uses output from *ragavi.vis_utils.calc_uvdist()*

Parameters

- **`freq`** (`xarray.DataArray or :obj:`float`) – Frequency(ies) from which corresponding wavelength will be obtained.
- **`uvw`** (`xarray.DataArray`) – UVW column from the MS dataset

Returns **uvwave** (`xarray.DataArray`) – uv distance in wavelength for specific frequency

`ragavi.vis_utils.`**`get_antennas`**(*ms_name*)
Function to get antennae names from the ANTENNA subtable.

Parameters **ms_name** (`str`) – Name of MS or table

Returns **ant_names** (`xarray.DataArray`) – A `xarray.DataArray` containing names for all the antennas available.

`ragavi.vis_utils.`**`get_fields`**(*ms_name*)
Get field names from the FIELD subtable.

Parameters **ms_name** (`str`) – Name of MS or table

Returns **field_names** (`xarray.DataArray`) – String names for the available fields

`ragavi.vis_utils.`**`get_flags`**(*xds_table_obj*, *corr=None*, *chan=slice(0*, *None*, *None)*)
Get Flag values from the FLAG column. Allow for selections in the channel dimension or the correlation dimension

Parameters

- **corr** (`int`) – Correlation number to select.
- **xds_table_obj** (`xarray.Dataset`) – MS as xarray dataset from xarrayms

Returns **flags** (`xarray.DataArray`) – Data array containing values from FLAG column selected by correlation if index is available.

`ragavi.vis_utils.`**`get_frequencies`**(*ms_name*, *spwid=slice(0*, *None*, *None)*, *chan=slice(0*, *None*, *None)*)
Function to get channel frequencies from the SPECTRAL_WINDOW subtable

Parameters

- **chan** (`slice` or `numpy.ndarray`) – A slice object or numpy array to select some or all of the channels. Default is all the channels
- **ms_name** (`str`) – Name of MS or table
- **spwid** (`int`) – Spectral window id number. Defaults to 0

Returns **freqs** (`xarray.DataArray`) – Channel centre frequencies for specified spectral window.

`ragavi.vis_utils.`**`get_polarizations`**(*ms_name*)
Get the type of polarizations available in the measurement set

Parameters **ms_name** (`str`) – Name of MS / table

Returns **cor2stokes** (`list`) – Returns a list containing the types of correlation

`ragavi.vis_utils.`**`name_2id`**(*tab_name*, *field_name*)
Translate field name to field id

Parameters

- **tab_name** (*:obj:str`*) – MS or Table name
- **field_name** (`str`) – Field name to convert to field ID

Returns **field_id** (`int`) – Integer field id

`ragavi.vis_utils.`**`resolve_ranges`**(*inp*)
Create a TAQL string that can be parsed given a range of values

Parameters **inp** (str) – A range of values to be constructed. Can be in the form of: "5", "5,6,7", "5~7" (inclusive range), "5:8" (exclusive range), "5:" (from 5 to last)

Returns **res** (str) – Interval string conforming to TAQL sets and intervals as shown in Casa TAQL Notes[6]

ragavi.vis_utils.**slice_data**(*inp*)

Creates a slicer for an array. To be used to get a data subset such as correlation or channel subsets.

Parameters **inp** (str) – This can be of the form "5", "10~20" (10 to 20 inclusive), "10:21" (same), "10:" (from 10 to end), ":10:2" (0 to 9 inclusive, stepped by 2), "~9:2" (same)

Returns **sl** (slice) – slicer for an iterable object

ragavi.vis_utils.**time_convert**(*xdata*)

Convert time from MJD to UTC time

Parameters **xdata** (xarray.DataArray) – TIME column from the MS or table xarray dataset in MJD format.

Returns **newtime** (xarray.DataArray) – TIME column in a more human readable UTC format. Stored as numpy.datetime type.

ragavi.vis_utils.**time_wrapper**(*func*)

A decorator function to compute the execution time of a function

[6] https://casa.nrao.edu/aips2_docs/notes/199/node5.html#TAQL:EXPRESSIONS

CHAPTER

SIX

APPENDIX

6.1 Gains

ragavi.ragavi.**add_axis**(*fig*, *axis_range*, *ax_label*)
Add an extra axis to the current figure

Parameters

- **fig** (bokeh.plotting.figure) – The figure onto which to add extra axis
- **axis_range** (float, float) – Starting and ending point for the range
- **ax_label** (str) – Label of the new axis

Returns **fig** (bokeh.plotting.figure) – Figure containing the extra axis

ragavi.ragavi.**alpha_slider_callback**()
JS callback to alter alpha of glyphs

Returns **code** (str)

ragavi.ragavi.**ant_select_callback**()
JS callback for the selection and de-selection of antennas

Returns **code** (str)

ragavi.ragavi.**autofill_gains**(*t*, *g*)
Normalise length of f and g lists to the length of t. This function is meant to support the ability to specify multiple gain tables while only specifying single values for gain table types.

Note: An assumption will be made that for all the specified tables, the same gain table type will be used.

Parameters

- **g** (list) – type of gain table [B,G,K,F].
- **t** (list) – list of the gain tables.

Returns **f** (list) – lists of length ..code::*len(t)* containing gain types.

ragavi.ragavi.**axis_fs_callback**()
JS callback to alter axis label font sizes

Returns **code** (str)

ragavi.ragavi.**batch_select_callback**()
JS callback for batch selection Checkboxes

Returns **code** (str)

ragavi.ragavi.**condense_legend_items**(*inlist*)

Combine renderers of legend items with the same legend labels. Must be done in case there are groups of renderers which have the same label due to iterations, to avoid a case where there are two or more groups of renderers containing the same label name.

Parameters **inlist** (list) – List containing legend items of the form (label, renders) as described in: Bokeh legend items[7]

Returns **outlist** (list) – A reduction of inlist

ragavi.ragavi.**create_legend_batches**(*num_leg_objs*, *li_ax1*, *li_ax2*, *batch_size=16*)

Automates creation of antenna **batches of 16** each unless otherwise.

This function takes in a long list containing all the generated legend items from the main function's iteration and divides this list into batches, each of size batch_size. The outputs provides the inputs to *ragavi.ragavi.create_legend_objs()*.

Parameters

- **batch_size** (int, optional) – Number of antennas in a legend object. Default is 16
- **li_ax1** (list) – List containing all legend items[8] for antennas for 1st figure items are in the form (antenna_legend, [renderer])
- **li_ax2** (list) – List containing all legend items for antennas for 2nd figure items are in the form (antenna_legend, [renderer])
- **num_leg_objs** (int) – Number of legend objects to be created

Returns

(bax1, bax2) ((list, list)) – Tuple containing List of lists which each have batch_size number of legend items for each batch. bax1 are batches for figure1 antenna legends, and ax2 batches for figure2 antenna legends

e.g bax1 = [[batch0], [batch1], . . . , [batch_numOfBatches]]

ragavi.ragavi.**create_legend_objs**(*num_leg_objs*, *bax1*, *bax2*)

Creates legend objects using items from batches list Legend objects allow legends be positioning outside the main plot

Parameters

- **num_leg_objs** (int) – Number of legend objects to be created
- **bax1** (list) – Batches for antenna legends of 1st figure
- **bax2** (list) – Batches for antenna legends of 2nd figure

Returns **lo_ax1, lo_ax2** (dict, dict) – Tuple containing dictionaries with legend objects for figure1 antenna legend objects, figure 2 antenna legend objects.

ragavi.ragavi.**determine_table**(*table_name*)

Find pattern at end of string to determine table to be plotted. The search is not case sensitive

Parameters **table_name** (str) – Name of table / gain type to be plotted

Returns **result** (str) – Table type of (if valid) of the input table_name

ragavi.ragavi.**errorbar**(*fig*, *x*, *y*, *yerr=None*, *color='red'*)

Add errorbars to Figure object based on x, y and attr:*yerr*

[7] https://bokeh.pydata.org/en/latest/docs/reference/models/annotations.html#bokeh.models.annotations.LegendItem
[8] https://bokeh.pydata.org/en/latest/docs/reference/models/annotations.html#bokeh.models.annotations.LegendItem

Parameters

- **color** (str) – Color for the error bars
- **fig** (bokeh.plotting.figure) – Figure onto which the error-bars will be added
- **x** (numpy.ndarray) – x_axis data
- **y** (numpy.ndarray) – y_axis data
- **yerr** (numpy.ndarray, numpy.ndarray) – Tuple with high and low limits for y

Returns **ebars** (bokeh.models.Whisker) – Return the object containing error bars

ragavi.ragavi.**field_selector_callback**()
Return JS callback for field selection checkboxes

Returns **code** (str)

ragavi.ragavi.**flag_callback**()
JS callback for the flagging button

Returns **code** (str)

ragavi.ragavi.**gen_checkbox_labels**(*batch_size*, *num_leg_objs*, *antnames*)
Auto-generating Check box labels

Parameters

- **batch_size** (int) – Number of items in a single batch
- **num_leg_objs** (int) – Number of legend objects / Number of batches

Returns **labels** (list) – Batch labels for the batch selection check box group

ragavi.ragavi.**gen_flag_data_markers**(*y*, *fid=None*, *markers=None*, *fmarker='circle_x'*)
Generate different markers for where data has been flagged.

Parameters

- **fid** (int) – field id number to identify the marker to be used
- **fmarker** (str) – Marker to be used for flagged data
- **markers** (list) – A list of all available bokeh markers
- **y** (numpy.ndarray) – The flagged data containing NaNs

Returns **masked_markers_arr** (numpy.ndarray) – Returns an n-d array of shape y.shape containing markers for valid data and fmarker where the data was NaN.

ragavi.ragavi.**get_argparser**()
Create command line arguments for ragavi-gains

Returns **parser** (ArgumentParser()) – Argument parser object that contains command line argument's values

ragavi.ragavi.**get_table**(*tab_name*, *antenna=None*, *fid=None*, *spwid=None*, *where=None*)
Get xarray Dataset objects containing gain table columns of the selected data

Parameters

- **antenna** (str, optional) – A string containing antennas whose data will be selected
- **fid** (int, optional) – FIELD_ID whose data will be selected
- **spwid** (int, optional) – DATA_DESC_ID or spectral window whose data will be selected
- **tab_name** (str) – name of your table or path including its name

- **where** (str, optional) – TAQL where clause to be used with the MS.

Returns **tab_objs** (list) – A list containing xarray.Dataset objects where each item on the list is determined by how the data is grouped

ragavi.ragavi.**get_tooltip_data**(*xds_table_obj*, *gtype*, *antnames*, *freqs*)
Get the data to be displayed on the tool-tip of the plots

Parameters

- **antnames** (list) – List containing antenna names
- **gtype** (str) – Type of gain table being plotted
- **xds_table_obj** (*xarray.Dataset*) – xarray-ms table object

Returns

- **spw_id** (numpy.ndarray) – Spectral window ids
- **scan_no** (numpy.ndarray) – scan ids
- **ttip_antnames** (numpy.ndarray) – Antenna names to show up on the tool-tips

ragavi.ragavi.**legend_toggle_callback**()
JS callback for legend toggle Dropdown menu

Returns **code** (str)

ragavi.ragavi.**main**(***kwargs*)
Main function that launches the visibility plotter

ragavi.ragavi.**make_plots**(*source*, *ax1*, *ax2*, *fid=0*, *color='red'*, *y1err=None*, *y2err=None*)
Generate a pair of plots

Parameters

- **ax1** (bokeh.plotting.figure) – First figure
- **ax2** (bokeh.plotting.figure) – Second Figure
- **color** (str) – Glyph color[s]
- **fid** (int) – field id number to set the line width
- **source** (bokeh.models.ColumnDataSource) – Data source for the plot
- **y1err** (numpy.ndarray, optional) – y1 Error margins for figure *ax1* data
- **y2err** (numpy.ndarray, optional) – y2 error margins for figure *ax2* data

Returns **(p1, p1_err, p2, p2_err)** ((bokeh.models.renderers.GlyphRenderer, bokeh.models.Whisker, bokeh.models.renderers.GlyphRenderer, bokeh.models.Whisker)) – Tuple of containing bokeh.models.renderers.GlyphRenderer with the data glyphs as well as errorbars. *p1* and *p2*: renderers containing data for ax1 and ax2 respectively. p1_err, p2_err outputs from *ragavi.ragavi.errorbar()* for ax1 and ax2 respectively.

ragavi.ragavi.**plot_table**(*mytabs*, *gain_types*, *fields*, ***kwargs*)
Plot gain tables within Jupyter notebooks.

Parameters

- **corr** (int, optional) – Correlation index to plot (usually just 0 or 1, default = 0)',default=0)

- **doplot** (str, optional) – Plot complex values as amp and phase (ap) or real and imag (ri). Default is 'ap'.
- **fields** (str, optional) – Field ID / Name (list of field ids or name) to plot. Default is all.
- **gain_types** (str, list) – Cal-table (list of caltypes) type to be plotted. Can be either 'B'-bandpass, 'D'- D jones leakages, G'-gains, 'K'-delay or 'F'-flux. Default is none
- **image_name** (str, optional) – Output image name. Default is of the form table_corr_doplot_fields
- **mycmap** (*str, optional*) – Matplotlib colour map to use for antennas. Default is coolwarm
- **mytabs** (str or list required) – The table (list of tables) to be plotted.
- **plotants** (str, optional) – Plot only this antenna, or comma-separated string of antennas. Default is all
- **t0** (int, optional) – Minimum time [in seconds] to plot. Default is full range
- **t1** (int, optional) – Maximum time [in seconds] to plot. Default is full range

ragavi.ragavi.**save_html**(*hname*, *plot_layout*)
Save plots in HTML format

Parameters

- **hname** (str) – HTML Output file name
- **plot_layout** (bokeh.layouts) – Layout of the Bokeh plot, could be row, column, gridplot.

ragavi.ragavi.**save_png_image**(*img_name*, *disp_layout*)
Save plots in PNG format

Note: One image will emerge for each figure in *disp_layout*. To save png images, the python package selenium, node package phantomjs are required. More information Exporting bokeh plots[9]

Parameters

- **disp_layout** (bokeh.layouts) – Layout object containing the renderers
- **img_name** (str) – Name of output image

ragavi.ragavi.**save_svg_image**(*img_name*, *figa*, *figb*, *glax1*, *glax2*)
Save plots in SVG format

Note: Two images will emerge. the python package selenium, node package phantomjs are required. More information Exporting bokeh plots[10]

Parameters

- **img_name** (str) – Desired image name
- **figa** (bokeh.plotting.figure) – First figure
- **figb** (bokeh.plotting.figure) – Second figure

[9] https://bokeh.pydata.org/en/latest/docs/user_guide/export.html
[10] https://bokeh.pydata.org/en/latest/docs/user_guide/export.html

- **glax1** (list) – Contains glyph metadata saved during execution
- **glax2** (list) – Contains glyph metadata saved during execution

ragavi.ragavi.**size_slider_callback**()
JS callback to select size of glyphs

Returns **code** (str)

ragavi.ragavi.**stats_display**(*tab_name*, *gtype*, *ptype*, *corr*, *field*, *flag=True*)
Display some statistics on the plots. These statistics are derived from a specific correlation and a specified field of the data.

Note: Currently, only the medians of these plots are displayed.

Parameters

- **corr** (int) – Correlation number of the data to be displayed
- **field** (int) – Integer field id of the field being plotted. If a string name was provided, it will be converted within the main function by *ragavi.vis_utils.name_2id()*.
- **gtype** (str) – Type of gain table to be plotted.
- **ptype** (str) – Type of plot ap / ri

Returns **pre** (bokeh.models.widgets) – Pre-formatted text containing the medians for both model. The object returned must then be placed within the widget box for display.

ragavi.ragavi.**title_fs_callback**()
JS callback for title font size slider

Returns **code** (str)

ragavi.ragavi.**toggle_err_callback**()
JS callback for Error toggle Toggle button

Returns **code** (str)

6.2 Visibilites

ragavi.visibilities.**add_axis**(*fig*, *axis_range*, *ax_label*)
Add an extra axis to the current figure

Parameters

- **fig** (bokeh.plotting.figure) – The figure onto which to add extra axis
- **axis_range** (list, tuple) – A range of sorted values or a tuple with 2 values containing or the order (min, max). This can be any ordered iteraable that can be indexed.

Returns **fig** (bokeh.plotting.figure) – Bokeh gigure with an extra axis added

ragavi.visibilities.**get_argparser**()
Create command line arguments for ragavi-gains

Returns **parser** (ArgumentParser()) – Argument parser object that contains command line argument's values

ragavi.visibilities.**get_ms**(*ms_name*, *data_col='DATA'*, *ddid=None*, *fid=None*, *scan=None*, *where=None*)

Get xarray Dataset objects containing Measurement Set columns of the selected data

Parameters

- **ms_name** (str) – Name of your MS or path including its name
- **data_col** (str) – Data column to be used. Defaults to 'DATA'
- **ddid** (int) – DATA_DESC_ID or spectral window to choose. Defaults to all
- **fid** (int) – Field id to select. Defaults to all
- **scan** (int) – SCAN_NUMBER to select. Defaults to all
- **where** (str) – TAQL where clause to be used with the MS.

Returns **tab_objs** (list) – A list containing the specified table objects as xarray.Dataset

ragavi.visibilities.**main**(***kwargs*)

Main function

ragavi.visibilities.**save_png_image**(*name*, *disp_layout*)

Save plots in PNG format .. note:: One image will emerge for each figure in *disp_layout*. To save png images, the python package selenium, node package phantomjs are required. More information Exporting bokeh plots[11]

Parameters

- **disp_layout** (bokeh.layouts) – Layout object containing the renderers
- **name** (str) – Name of output image

[11] https://bokeh.pydata.org/en/latest/docs/user_guide/export.html

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

r

INDEX

S

T