# ragavi

*Release 0.2.0*

**Mar 05, 2020**

# Contents:

CHAPTER 1

# 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] https://github.com/ratt-ru/ragavi
[2] https://bokeh.pydata.org/en/latest/index.html

## 1.3 Limitations

`Ragavi` is still under development.

Main dependencies for this project

- Daskms[3]
- Bokeh[4]
- Datashader[5]

[3] https://xarray-ms.readthedocs.io/en/latest/
[4] https://bokeh.pydata.org/en/latest/index.html
[5] http://datashader.org/

CHAPTER 2

# 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

For the gains plotter, the name-space `ragavi-vis` is used. To get help for this

**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-gains -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.

```

```

# CHAPTER 3

## 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 |

## 3.1 Note

- At least a single field, spectral window and correlation **must** be selected in order for plots to show up.

- While antenna data can be made visible through clicking its corresponding legend, this behaviour is not linked to the field, SPW, correlation selection checkboxes. Therefore, clicking the legend for a specific antenna will make data from all fields, SPWs and correlation for that antenna visible. As a workaround, data points can be identifies using tooltip information
- Unless **all** the available fields, SPWs and correlations have not been selected, the antenna legends will appear greyed-out. This is because a single legend is attached to multiple data for each of the available categories. Therefore, clicking on legends without meeting the preceeding condition may lead to some awkward results (a toggling effect).

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)
```

### 3.1.1 API

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

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*, *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*, ***kwargs*)

Plot gain tables within Jupyter notebooks.

**Parameters**

- **`_corr`** (`int, optional`) – Correlation index to plot. Can be a single integer or comma separated integers e.g '0,2'. Defaults to all.
- **`doplot`** (`str, optional`) – Plot complex values as amp and phase (ap) or real and imag (ri). Default is 'ap'.
- **`ddid`** (`int`) – SPECTRAL_WINDOW_ID or ddid number. Defaults to all
- **`fields`** (`str, optional`) – Field ID(s) / NAME(s) to plot. Can be specified as "0", "0,2,4", "0~3" (inclusive range), "0:3" (exclusive range), "3:" (from 3 to last) or using a field name or comma separated field names. Defaults to 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
- **`taql`** (`str, optional`) – TAQL where clause
- **`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 4

## 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=None*, *corr=None*, *cbin=None*, *ddid=<class 'int'>*, *datacol='DATA'*, *flag=True*)

Process Measurement Set data into forms desirable for visualisation.

**Parameters**

- **chan** (slice or int) – Channels that will be selected. Defaults to all
- **cbin** (int) – Channel averaging bin size
- **corr** (int or slice) – 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*, *cbin=None*, *chan=None*, *corr=None*, *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*, *x_min=None*, *x_max=None*, *y_min=None*, *y_max=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
- **xmin** (float) – Minimum x value to be plotted

  ---

  **Note:** This may be difficult to achieve in the case where xaxis is time because time in ragavi-vis is converted into milliseconds from epoch for ease of plotting by bokeh.

  ---

- **xmax** (float) – Maximum x value to be plotted
- **ymin** (float) – Minimum y value to be plotted
- **ymax** (float) – Maximum y value to be plotted

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

CHAPTER 5

# Utilities

Some utilities useful internally and externally from `ragavi`

# CHAPTER 6

# Appendix

## 6.1 Gains

ragavi.ragavi.**add_axis**(*fig*, *axis_range*, *ax_label*, *ax_name*)
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
- **ax_name** (str) – Name of the new model for the extra axis incase of multiple spectral windows.

**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) – # bokeh.models.annotations.LegendItem>‘_ List containing legend items of the form (label, renders) as described in: ‘Bokeh legend items <https://bokeh.pydata.org/en/latest/docs/reference/models/annotations.html

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

ragavi.ragavi.**create_legend_batches**(*num_leg_objs*, *li_ax1*, *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) – # bokeh.models.annotations.LegendItem>‘_ for antennas for 1st figure # items are in the form (antenna_legend, [renderer]) List containing all ‘legend items <https://bokeh.pydata.org/en/latest/docs/reference/models/annotations.html
- **num_leg_objs** (int) – Number of legend objects to be created

**Returns**

**bax1** (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*)
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

**Returns** **lo_ax1** (dict) – Dictionaries with legend objects for figure1 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*

**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_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_time_range**(*tab_name*, *unix_time=True*)
Get the first TIME column before selections

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 gains 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*, ***kwargs*)
Plot gain tables within Jupyter notebooks.

**Parameters**

- **_corr** (int, optional) – Correlation index to plot. Can be a single integer or comma separated integers e.g '0,2'. Defaults to all.
- **doplot** (str, optional) – Plot complex values as amp and phase (ap) or real and imag (ri). Default is 'ap'.
- **ddid** (int) – SPECTRAL_WINDOW_ID or ddid number. Defaults to all
- **fields** (str, optional) – Field ID(s) / NAME(s) to plot. Can be specified as "0", "0,2,4", "0~3" (inclusive range), "0:3" (exclusive range), "3:" (from 3 to last) or using a field name or comma separated field names. Defaults to 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
- **taql** (str, optional) – TAQL where clause
- **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[6]

---

**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

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

**Note:** Two images will emerge. the python package selenium, node package phantomjs are required. More information Exporting bokeh plots[7]

**Parameters**

- **img_name** (str) – Desired image name
- **figa** (bokeh.plotting.figure) – First figure
- **figb** (bokeh.plotting.figure) – Second figure
- **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*, *f_names=None*, *flag=True*, *spwid=None*)
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

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

**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.**average_ms**(*ms_name*, *tbin=None*, *cbin=None*, *chunk_size=None*, *taql=''*, *columns=None*, *chan=None*)

Perform MS averaging :param ms_name: Name of the input MS :type ms_name: str :param tbin: Time bin in seconds :type tbin: float :param cbin: Number of channels to bin together :type cbin: int :param chunk_size: Size of resulting MS chunks. :type chunk_size: dict :param taql: TAQL clause to pass to xarrayms :type taql: str

**Returns** **x_dataset** (list) – List of xarray.Dataset containing averaged MS. The MSs are split by Spectral windows

ragavi.visibilities.**create_bl_data_array**(*ant1*, *ant2*)

Make a dataArray containing baseline numbers

**Parameters**

- **ant1** (xarray.DataArray) – ANTENNA1 dataArray
- **ant2** (xarray.DataArray) – ANTENNA2 dataArray

**Returns** **baseline** (xarray.DataArray) – DataArray containing baseline numbers

ragavi.visibilities.**get_ms**(*ms_name*, *chunks=None*, *data_col='DATA'*, *ddid=None*, *fid=None*, *scan=None*, *where=None*, *cbin=None*, *tbin=None*, *chan_select=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
- **chunks** (str) – Chunk sizes for the resulting dataset.
- **cbin** (int) – Number of channels binned together for channel averaging
- **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
- **tbin** (float) – Time in seconds to bin for time averaging
- **where** (str) – TAQL where clause to be used with the MS.
- **chan_select** (int or slice) – Channels to be selected

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

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

Main function that launches the visibilities plotter

ragavi.visibilities.**make_cbar**(*cats*, *category*, *cmap=None*)

Initiate a colorbar for categorical data :param cats: Available category ids e.g scan_number, field id etc :type cats: np.ndarray :param category: Name of the categorizing axis :type category: str :param cmap: Matplotlib colormap :type cmap: matplotlib.cmap

**Returns** **cbar** (`bokeh.models`) – Colorbar instance

CHAPTER 7

# Changelog

## 7.1 0.3.0

- All argument parsers moved to `arguments.py`
- **ragavi-vis**
  - Introduced MS averaging in `ragavi-vis`
  - `--cbin` and `--tbin` added for channel and time averaging
  - `--mem-limit` and `--num-cores` for specifying memory limit per core and number of cores dask should use
  - Remove `--image-name` argument from `ragavi-vis`
- **ragavi-gains** - Fixed field, correlation selection bugs #50 - Fixed spectral window selection bug - Added spectral window selection widgets - Moved stats from plot titles to table below the plots - Changed time xaxis to UTC time - Added new download selected data button - All availabe times displayed for bandpass plots

## 7.2 0.2.3

- Add option *-kx* , *–k-xaxis* to allow selection of K table's x-axis (`ragavi-gains`)
- Values in *–field* can now be either comma or space separated

## 7.3 0.2.2

- Add name of gain table plotted to the plot
- Delay (K) now plotted over time (Fixing #45)
- Fix bug with relative times (Fixing $46)

## 7.4 0.2.1

- Fix some bugs with missing fields and correlations
- Only supporting python3 now

## 7.5 0.2.0

- Introduced ragavi visibility plotter accessible by ragavi-vis
- Improved documentation
- Added progress bar for `ragavi-vis`
- Changed gain plotter name to `ragavi-gains`. Deprecating `ragavi`
- Added `--xmin`, `--xmax`, `--ymin`, `--ymax` options in *ragavi-vis* for selection of x and y data ranges
- Added `--chunks` command line option for user specified chunking strategies in `ragavi-vis`
- Migrate from `xarray-ms` to `dask-ms` for table functions
- Added correlation selector on gain plots. All correlations plotted by default
- Removed `--yu0, --yu1, --yl0, --yl1` from *ragavi-gains*
- Fixed field selection and errorbar size bugs
- `--field` arguments in `ragavi-gains` **MUST** now be comma separated rather than space separated.

## 7.6 0.1.0

- Error bars now have caps
- Introduced linked legends
- Default displayed data is now flagged
- Flagged data shown using inverted-triangle

## 7.7 0.0.9

- Added flag button on plot
- Plotting D-Jones tables now supported
- Fixed bug in field_name to field_id converter

## 7.8 0.0.8

- Fixed bug due to string encoding for python2.7

## 7.9 0.0.7

- Updated version number

## 7.10 0.0.6

- Now supporting python3
- All fields plotted by default on the same plot
- `--field` command line switch is now optional
- Different fields now plotted with different markers
- Migrated to `xarray-ms` from `python-casacore`
- Added glyph alpha selector, glyph size selector, and field selector
- Reorganise selector panel
- Added title and axis label size selectors
- Add field symbols alongside field names on checkboxes
- Allow automatic plot scaling
- Medians now shown in plot titles

## 7.11 0.0.5

- Added support for multiple table, fields and gaintype inputs
- Multiple table single field single gaintype input also allowed
- Plots from multiple tables plotted on single html file
- Added slider to change plot sizes
- All notifications and errors now logged to ragavi.log

## 7.12 0.0.4

- Removed msname flag, Antenna names now show up in legends by default
- Support for string field names in addition to field indices
- Spectral window id, antenna name and scan id displayed on tooltip
- Remove second plot (for correlation 2) from delay table

## 7.13 0.0.3

- Travis realease on tag
- Now plotting Flux callibration tables

- Extra frequency axis for bandpass plot

## 7.14 0.0.2

- Module importable
- Table parameter option

CHAPTER 8

# Indices and tables

- genindex
- modindex
- search

# Python Module Index

## r

# Index

## T