ragavi

Release 0.5.2

Oct 21, 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
```

Multiple tables can be plotted on the same document simply by adding them in a space separated list to the `-t` / `--table` switch e.g

```
$ ragavi-gains -t delay/table/1/ bandpass/table/2 flux/table/3
```

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 --ms /my/measurement/set --xaxis time --yaxis amplitude
```

For large datasets, it is advisable to supply at least `--ymin` and `--ymax` values to avoid an extra pass over the data.

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 (K tables)
- Gain calibration (G tables)
- D-Jones Leakage tables (D tables)
- Pol-cal tables (Kcross, Xf, Df)

Mandatory argument is `--table`.

If a field name is not specified to `ragavi-vis` all the fields will be plotted by default. This is the same for correlations and spectral windows.

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 as a space separate list as below

```
$ragavi-gains --table table/one/name table/two/name table/three/name table/four/name -
↪-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

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 identified 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).

By default, *ragavi-gains* automatically determines an appropriate x-axis for a given input table. This behaviour can be changed by specifying the *-x/xaxis* argument.

3.1 Use in Jupyter Notebooks

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=[], cmap='viridis', doplot='ri', corr=1, ant='1,2,3,9,10')

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

3.2 Generating Static (Non-Interactive) Images

It is possible to generate png, ps, pdf, svg with `ragavi-gains` via two methods. The first method involves generating the HTML format first and then using the save tool found in the toolbar to download the plots. This method requires minimal effort although it may be a necessary redundancy to achieve the static image goal.

The second method involves supplying the `--plotname` argument, including the desired file extension. For example, `--plotname test.png`. If only this argument is supplied, then only a single static plot is generated. In case of multiple tables, multiple static files will be generated. However, in case one wants both static and interactive plots, both `--htmlname` and

`--plotname` must be supplied.

By default, `ragavi` uses the canvas image backend for interactive plots, due to performance issues associated with SVG image backend as stated in the Bokeh docs[6]. The default plots generated are always in HTML format.

Warning: If the gain tables supplied contain more than 30,000 points, interactive plots become extremely large and barely interactive (and unresponsive).

In an attempt to overcome this, `ragavi-gains` generates static plots, in addition to interactive plots, in this case. It is advisable to **avoid opening the large (upto hundreds of MBs) HTML as they may cause the browser to hang**. Instead, **inspect the static plot first** and **then make an interactive plot containing the antenna / field / corr etc. of your interest** by using the selection arguments provided.

[6] https://docs.bokeh.org/en/latest/docs/user_guide/export.html#svg-generation

3.3 Help

The full help output for `ragavi-gains` is:

```
  usage: ragavi-gains [options] <value>

  A Radio Astronomy Gains and Visibility Inspector

  optional arguments:
    -h, --help            show this help message and exit
    -v, --version         show program's version number and exit

  Required arguments:
    -t   [  ...], --table   [  ...]
                          Table(s) to plot. Multiple tables can be specified as a␣
↪space separated list

Data Selection:
  -a , --ant            Plot only a specific antenna, or comma-separated list
                        of antennas. Defaults to all.
  -c , --corr           Correlation index to plot. Can be a single integer or
                        comma separated integers e.g '0,2'. Defaults to all.
  --ddid                SPECTRAL_WINDOW_ID or ddid number. Defaults to all
  -f  [ ...], --field  [ ...]
                        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
  --t0                  Minimum time to plot [in seconds]. Defaults to full
                        range]
  --t1                  Maximum time to plot [in seconds]. Defaults to full
                        range
  --taql                TAQL where clause

Plot settings:
  --cmap                Bokeh or Colorcet colour map to use for antennas. List
                        of available colour maps can be found at: https://docs
                        .bokeh.org/en/latest/docs/reference/palettes.html or
                        https://colorcet.holoviz.org/user_guide/index.html .
                        Defaults to coolwarm
  -d , --doplot         Plot complex values as amplitude & phase (ap) or real
                        and imaginary (ri) or both (all). Defaults to ap.
  --debug               Enable debug messages
  -g [  [  ...]], --gaintype [  [  ...]]
                        Type of table(s) to be plotted. Can be specified as a
                        single character e.g. "B" if a single table has been
                        provided or space separated list e.g B D G if multiple
                        tables have been specified. Valid choices are B D G K
                        & F
  -kx , --k-xaxis       Choose the x-xaxis for the K table. Valid choices are:
                        time or antenna. Defaults to time.
  -lf , --logfile       The name of resulting log file (with preferred
                        extension) If no file extension is provided, a '.log'
                        extension is appended. The default log file name is
                        ragavi.log
  -o , --htmlname       Name of the resulting HTML file. The '.html' prefix
```

(continues on next page)

(continued from previous page)

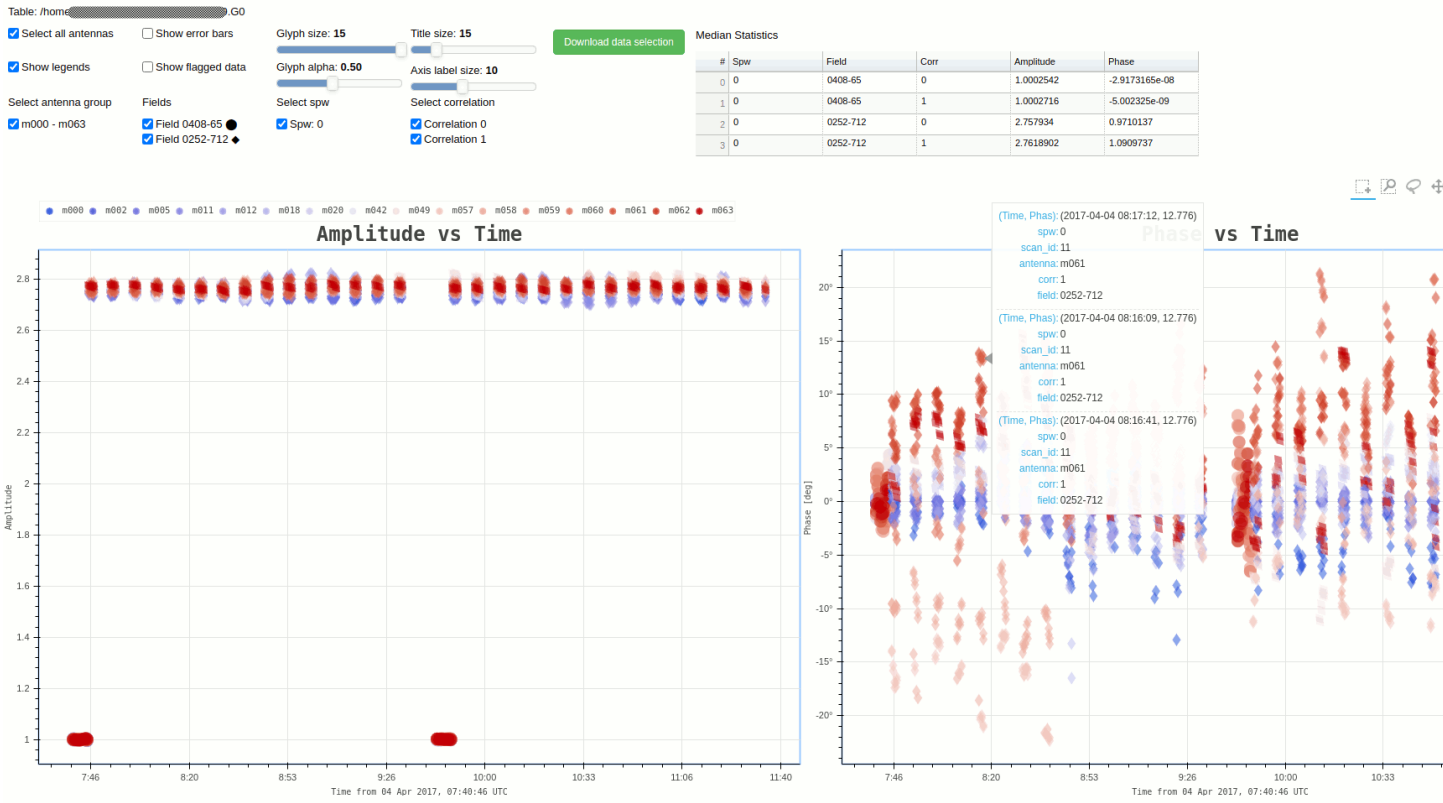
```
                        will be appended automatically.
  -p , --plotname       Static image name. The suffix of this name determines
                        the type of plot. If foo.png, the output will be PNG,
                        else if foo.svg, the output will be of the SVG format.
```

3.4 Examples

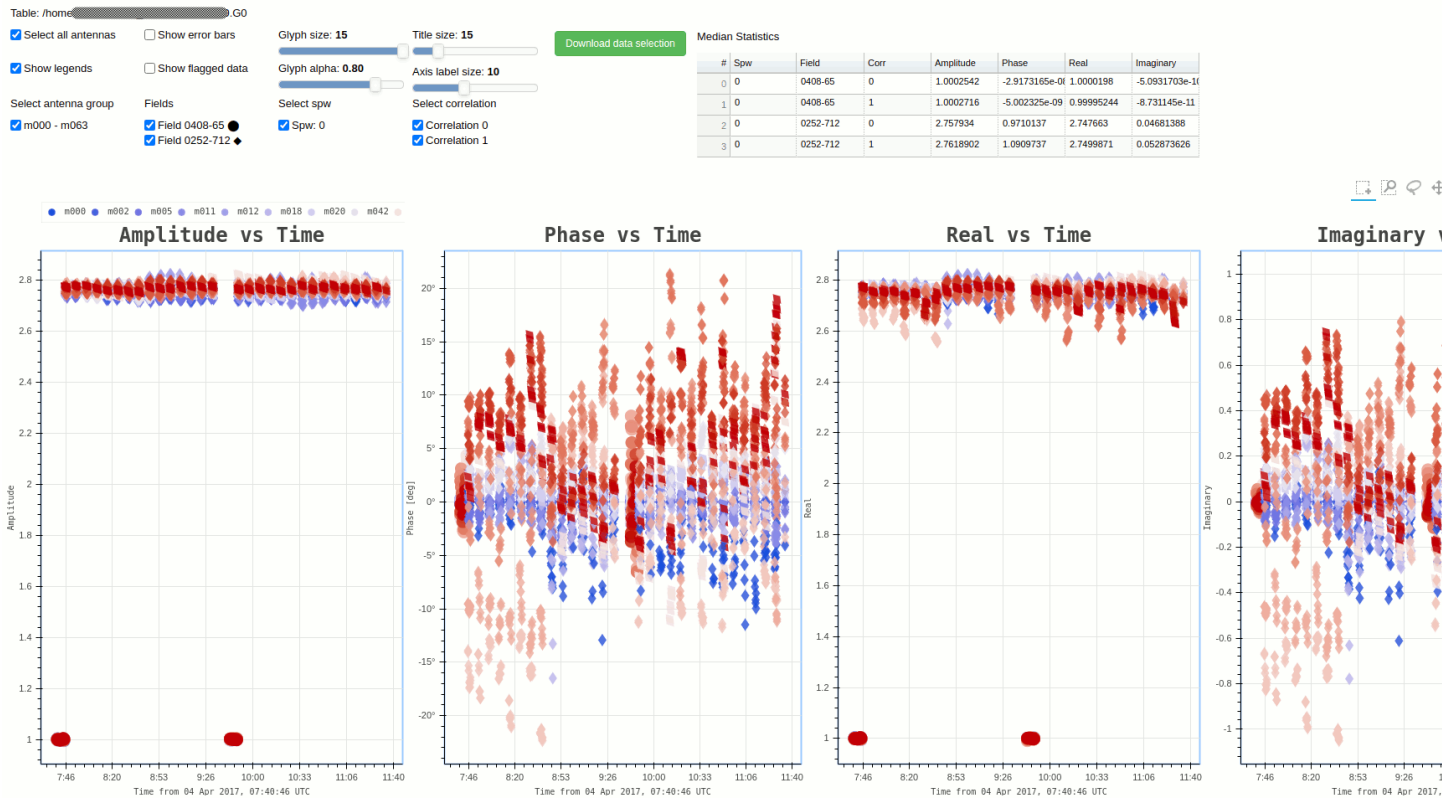
This subsection demonstrates the kind of plots that are generated

3.4.1 Interactive plots

Command: `ragavi-gains --table test.B0 --doplot ap --htmlname test`

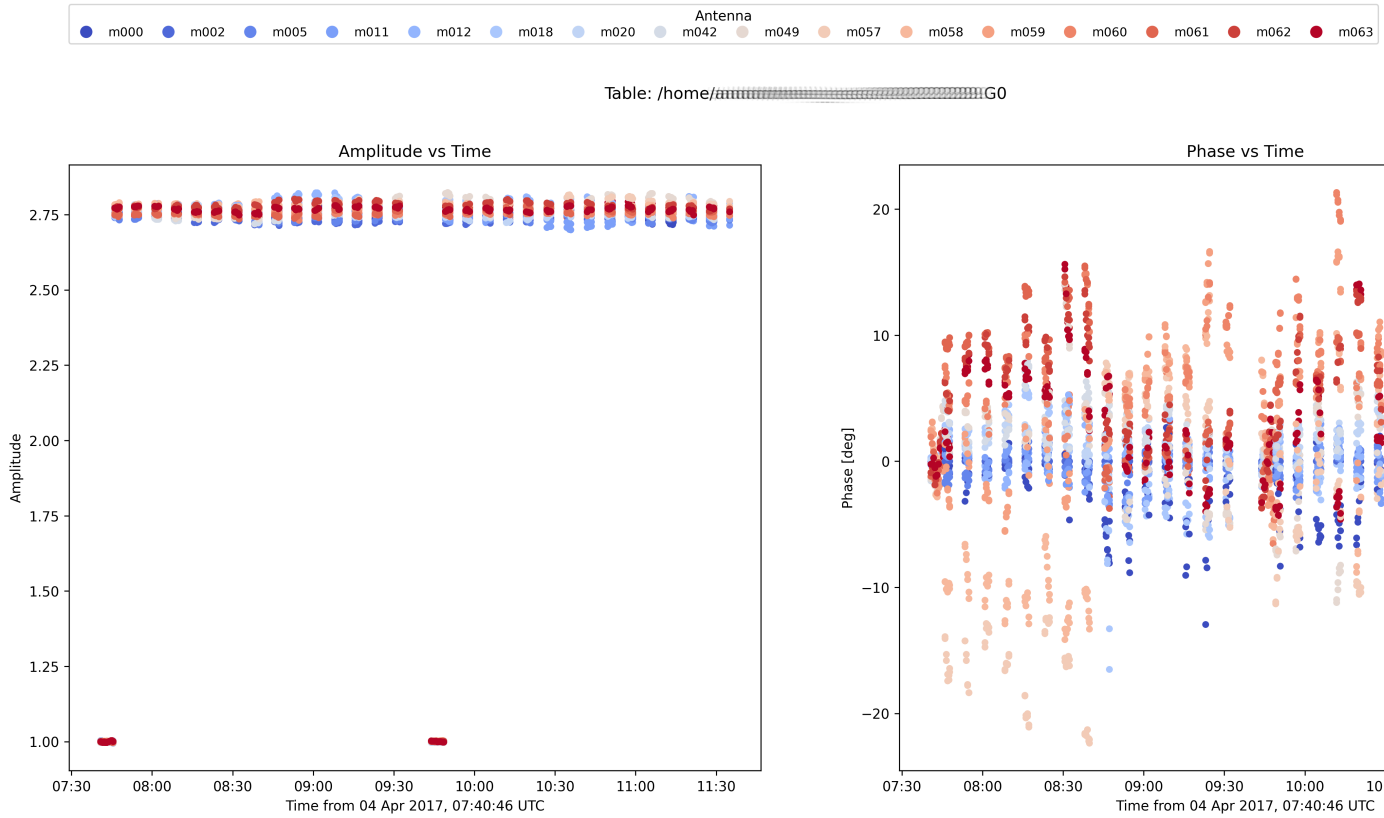


Command: `ragavi-gains --table test.B0 --doplot all --htmlname test`

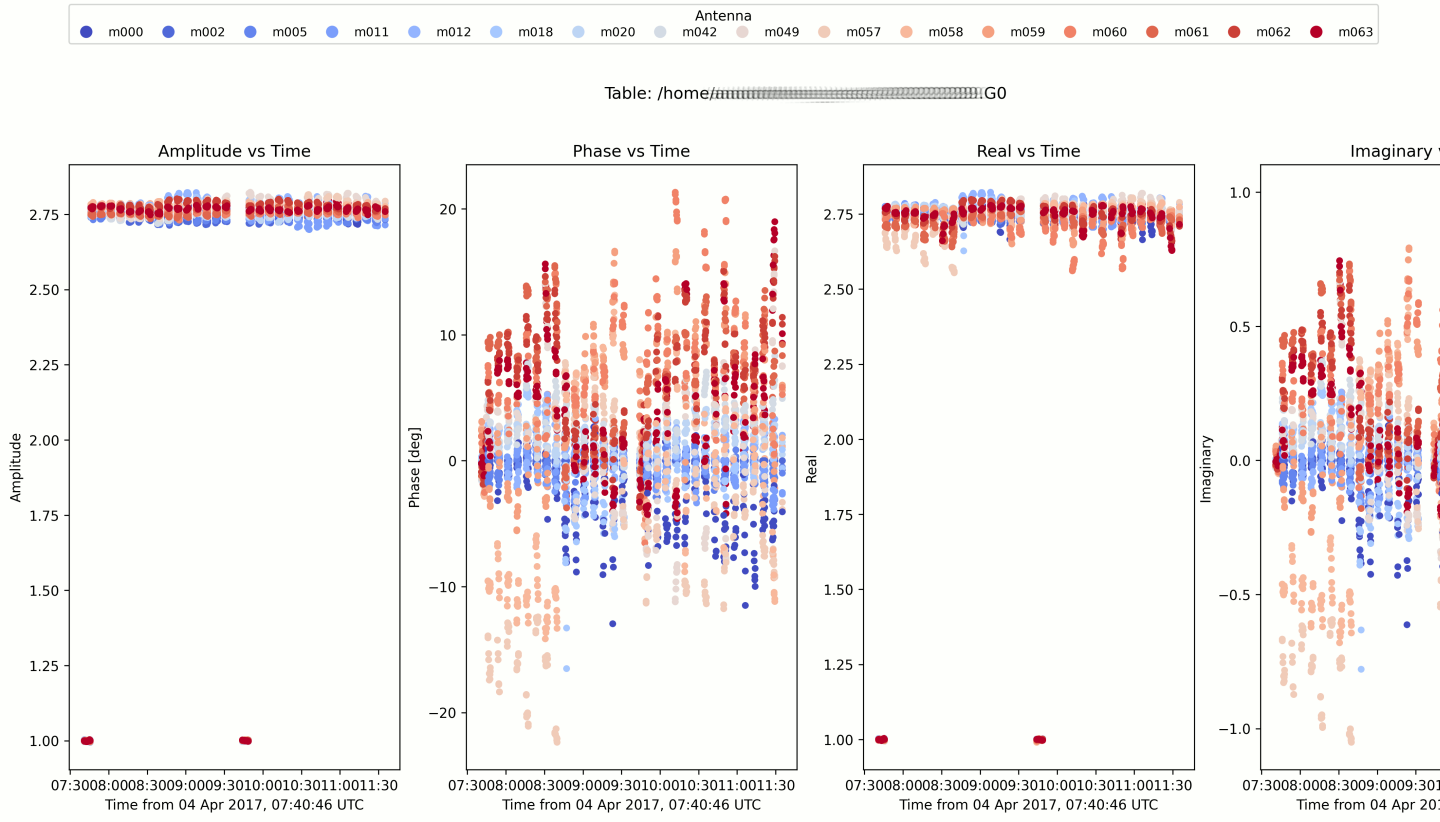


3.4.2 Static plots

Command: `ragavi-gains --table test.B0 --doplot ap --plotname test.png`



Command: `ragavi-gains --table test.B0 --doplot all --plotname test.png`



3.5 Jupyter Notebook function

ragavi.ragavi.**plot_table**(***kwargs*)

Plot gain tables within Jupyter notebooks. Parameter names correspond to the long names of each argument (i.e those with –) from the *ragavi-vis* command line help

Parameters

- **table** (str or list) – The table (list of tables) to be plotted.
- **ant** (str, optional) – Plot only specific antennas, or comma-separated list of antennas.
- **corr** (int, optional) – Correlation index to plot. Can be a single integer or comma separated integers e.g "0,2". Defaults to all.
- **cmap** (*str, optional*) – Matplotlib colour map to use for antennas. Default is coolwarm
- **ddid** (int) – SPECTRAL_WINDOW_ID or ddid number. Defaults to all
- **doplot** (str, optional) – Plot complex values as amp and phase (ap) or real and imag (ri). Default is "ap".
- **field** (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.
- **k-xaxis** (str) – Choose the x-xaxis for the K table. Valid choices are: time or antenna. Defaults to time.
- **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

This is the visibility plotter. Supported arguments are as follows

xaxis	yaxis	iter-axis	colour-axis
amplitude	amplitude	antenna	antenna1
antenna1	phase	antenna1	antenna2
antenna2	real	antenna2	spw
channel	imaginary	baseline	baseline
frequency		corr	corr
imaginary		field	field
phase		scan	scan
real		spw	
scan			
time			
uvdistance (uv distamce m)			
uvwave (uv distance lambda)			

Some of the arguments can be shortened using the following aliases

axis	alias
amplitude	amp
antenna	ant
antenna1	ant1
antenna2	ant2
baseline	bl
channel	chan
frequency	freq
imaginary	imag
uvdistance	uvdist
uvwave	uvidistl / uvdist_l

Iteration can also be activated through the `-ia / --iter-axis` option. It is also possible to colour over some

axis (most of the iteration axes are supported as shown in the table above) and this can be activated through the `-ca / --colour-axis` argument. Please note that the mandatory arguments are `--xaxis`, `--yaxis`, `--ms`.

Note: Output files genrated by `ragavi-vis` can be very large for iterated plots because of the resolution of each generated plot. For this reason, unless `--canvas-width` and `--canvas-height` are explicitly provided, `ragavi-gains` will automatically shrink the canvas sizes to 200 by 200 in order to minimise the resulting output.

4.1 Averaging

`ragavi-vis` has the ability to perform averaging before a plot is generated, by specifying the `--cbin` or `--tbin` arguments. These enable channel averaging and time averaging respectively. It is made possible through the use of the Codex[7] africanus averaging API.

Averaging is performed over per SPW, FIELD and Scan. This means that data is grouped by DATA_DESC_ID, FIELD_ID and SCAN_NUMBER beforehand. However, data selection (such as selection of spws, fields, scans and baselines to be present) is done before the averaging to avoid processing data that is unnecessary for the plot.

4.2 Computing Resources

Number of computer cores to be used, memory per core and the size of chunks to be loaded to each core may also be specified using `-nc / --num-cores`, `-ml / --memory-limit` and `-cs / --chunk-size` respectively. These may play an active role in improving the performance and memory management as `ragavi-vis` runs. However, finding an optimal combination may be a tricky task but is well worth while.

By default, `ragavi-vis` will use a maximum of 10 cores, with the maximum memory associated to each core being 1GB, and a chunk size in the row axis as 5000. The number of cores to be used, however, is dependent on the amount of RAM that is available on the host machine, in order to try and ensure that:

```
Number of cores x memory limit per core < total amount of available RAM
```

This means that, if the number of cores is less than 10, then by default, `ragavi-vis` will attempt to match the number of cores to those available. Given that visibility data has the shape (rows x channels x correlations), chunk sizes may also be chosen per each dimension using comma separated values (see the help[8] section on this page). As mentioned, the default chunk size is 5000 in the row axis, while the chunks sizes in the rest of the dimension are determined by the sizes of those dimensions (hence remaining as they are). Therefore, the true size of the chunks during processing will be translated to (nrows x nchannels x ncorrelations).

It is worth noting that supplying the x-axis and y-axis minimums and maximums may also significantly cut down the plotting time. This is because for minimum and maximum values to be calculated, `ragavi-vis`' backends must pass through the entire dataset at least once before plotting begins and again as plotting continues, therefore, taking a longer time. While the effect of this may be minimal in small datasets, it is certainly amplified in large datasets.

4.3 Help

The output of the help is as follows:

[7] https://codex-africanus.readthedocs.io/en/latest/
[8] https://ragavi.readthedocs.io/en/dev/vis.html#help

```
  usage: ragavi-vis [options] <value>

  optional arguments:
    -h, --help            show this help message and exit
    -v, --version         show program's version number and exit

  Required arguments:
    --ms  [ ...]          MS to plot. Default is None
    -x , --xaxis          X-axis to plot
    -y , --yaxis          Y-axis to plot

Plot settings:
  -ch , --canvas-height
                        Set height resulting image. Note: This is not the plot
                        height. Default is 720
  -cw , --canvas-width
                        Set width of the resulting image. Note: This is not
                        the plot width. Default is 1080.
  --cmap                Colour or colour map to use.A list of valid cmap
                        arguments can be found at:
                        https://colorcet.pyviz.org/user_guide/index.html Note
                        that if the argument "colour-axis" is supplied, a
                        categorical colour scheme will be adopted. Default is
                        blue.
  --cols                Number of columns in grid if iteration is active.
                        Default is 9.
  -ca , --colour-axis   Select column to colourise by. This will result in a
                        single image. Default is None.
  --debug               Enable debug messages
  -ia , --iter-axis     Select column to iterate over. This will result in a
                        grid. Default is None.
  -lf , --logfile       The name of resulting log file (with preferred
                        extension) If no file extension is provided, a '.log'
                        extension is appended. The default log file name is
                        ragavi.log
  -o , --htmlname       Output HTML file name (without '.html')

Data Selection:
  -a , --ant            Select baselines where ANTENNA1 corresponds to the
                        supplied antenna(s). "Can be specified as e.g. "4",
                        "5,6,7", "5~7" (inclusive range), "5:8" (exclusive
                        range), 5:(from 5 to last). Default is all.
  --chan                Channels to select. Can be specified using syntax i.e
                        "0:5" (exclusive range) or "20" for channel 20 or
                        "10~20" (inclusive range) (same as 10:21) "::10" for
                        every 10th channel or "0,1,3" etc. Default is all.
  -c , --corr           Correlation index or subset to plot. Can be specified
                        using normal python slicing syntax i.e "0:5" for
                        0<=corr<5 or "::2" for every 2nd corr or "0" for corr
                        0 or "0,1,3". Can also be specified using comma
                        separated corr labels e.g 'xx,yy' or specifying 'diag'
                        / 'diagonal' for diagonal correlations and 'off-diag'
                        / 'off-diagonal' for of diagonal correlations. Default
                        is all.
  -dc , --data-column   MS column to use for data. Default is DATA.
  --ddid                DATA_DESC_ID(s) /spw to select. Can be specified as
                        e.g. "5", "5,6,7", "5~7" (inclusive range), "5:8"
```

(continues on next page)

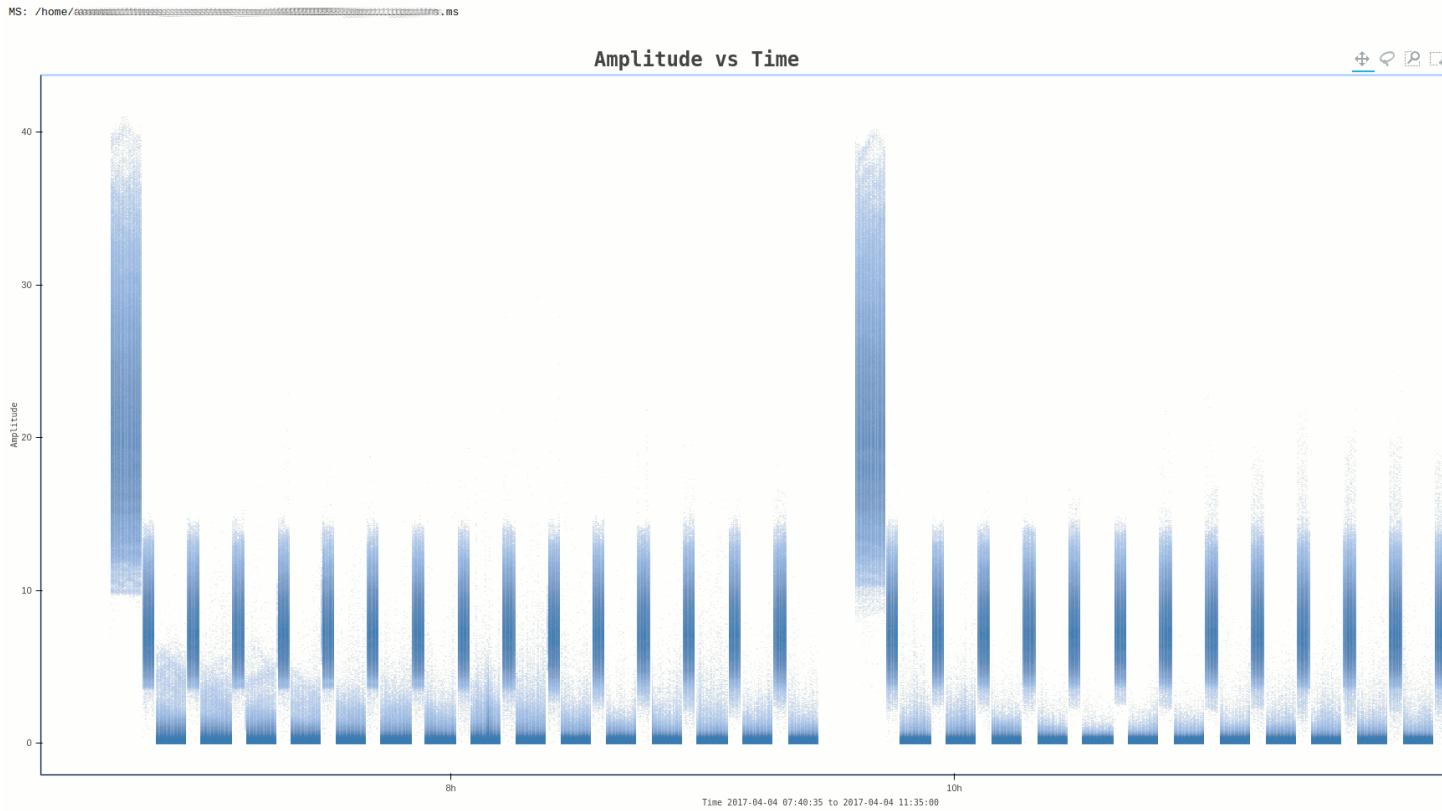
(continued from previous page)

```
                        (exclusive range), 5:(from 5 to last). Default is all.
  -f , --field          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. Default is
                        all
  -if, --include-flagged   Include flagged data in the plot. (Plots both flagged
                        and unflagged data.)
  -s , --scan           Scan Number to select. Default is all.
  --taql                TAQL where
  --xmin                Minimum x value to plot
  --xmax                Maximum x value to plot
  --ymin                Minimum y value to plot
  --ymax                Maximum y value to plot

Averaging settings:
  --cbin                Size of channel bins over which to average .e.g
                        setting this to 50 will average over every 5 channels
  --tbin                Time in seconds over which to average .e.g setting
                        this to 120.0 will average over every 120.0 seconds

Resource configurations:
  -cs , --chunks        Chunk sizes to be applied to the dataset. Can be an
                        integer e.g "1000", or a comma separated string e.g
                        "1000,100,2" for multiple dimensions. The available
                        dimensions are (row, chan, corr) respectively. If an
                        integer, the specified chunk size will be applied to
                        all dimensions. If comma separated string, these chunk
                        sizes will be applied to each dimension respectively.
                        Default is 5,000 in the row axis.
  -ml , --mem-limit     Memory limit per core e.g '1GB' or '128MB'. Default is
                        1GB
  -nc , --num-cores     Number of CPU cores to be used by Dask. Default is 10
                        cores. Unless specified, however, this value may
                        change depending on the amount of RAM on this machine
                        to ensure that: num-cores * mem-limit < total RAM
                        available
```

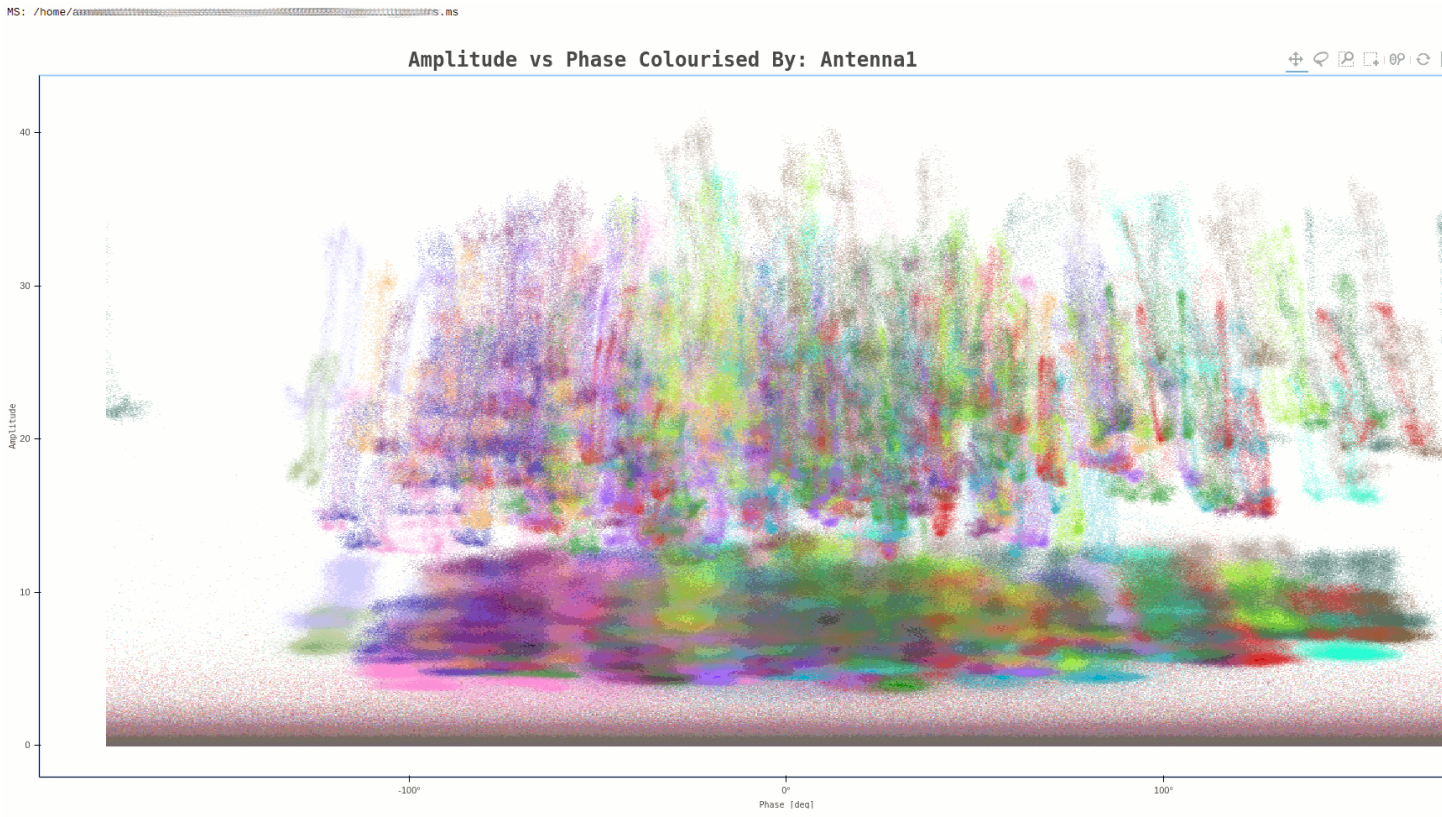
4.4 Examples

Some example plots generated by `ragavi-vis`

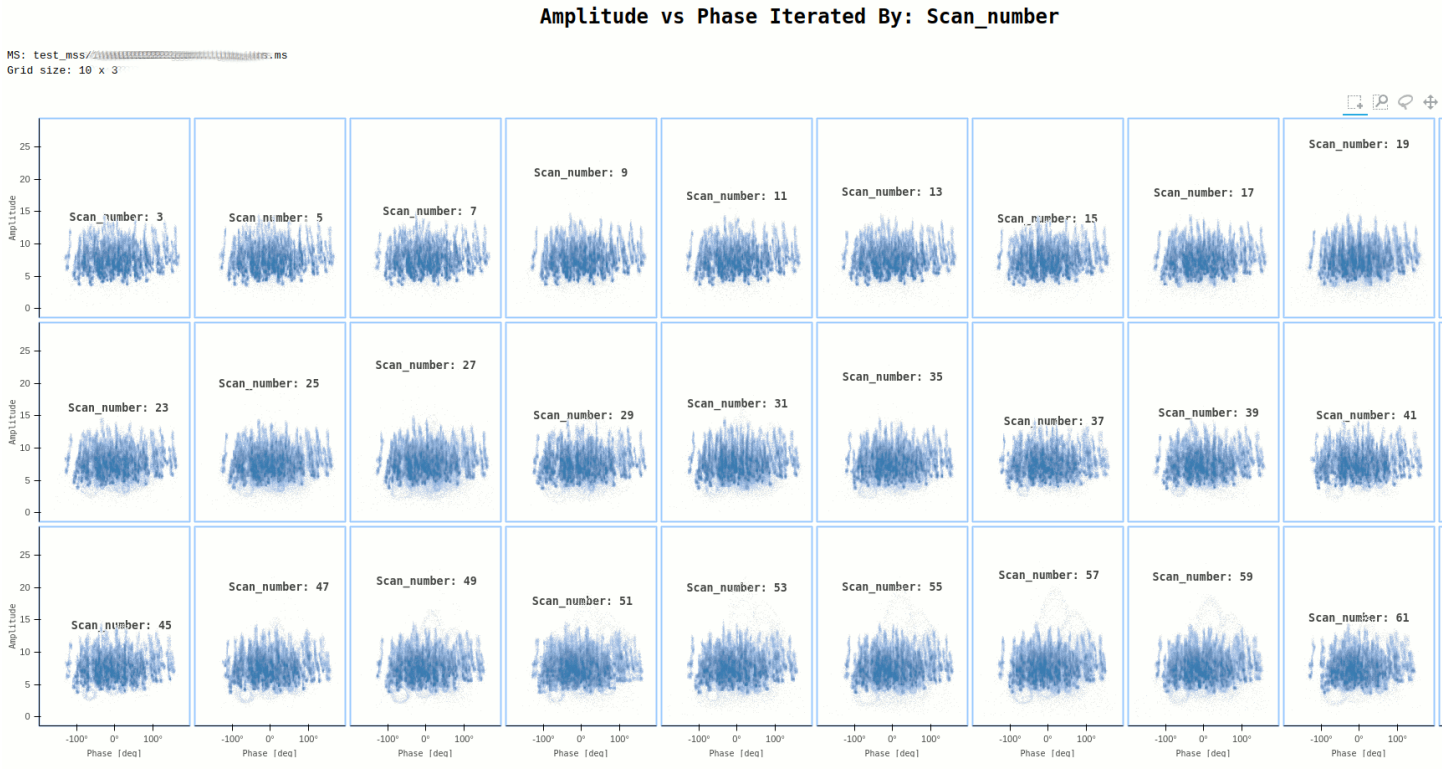
Command: `ragavi-vis --ms dir/test.ms --xaxis time --yaxis amp`



Command: `ragavi-vis --ms dir/test.ms --xaxis phase --yaxis amp --colour-axis antenna1`

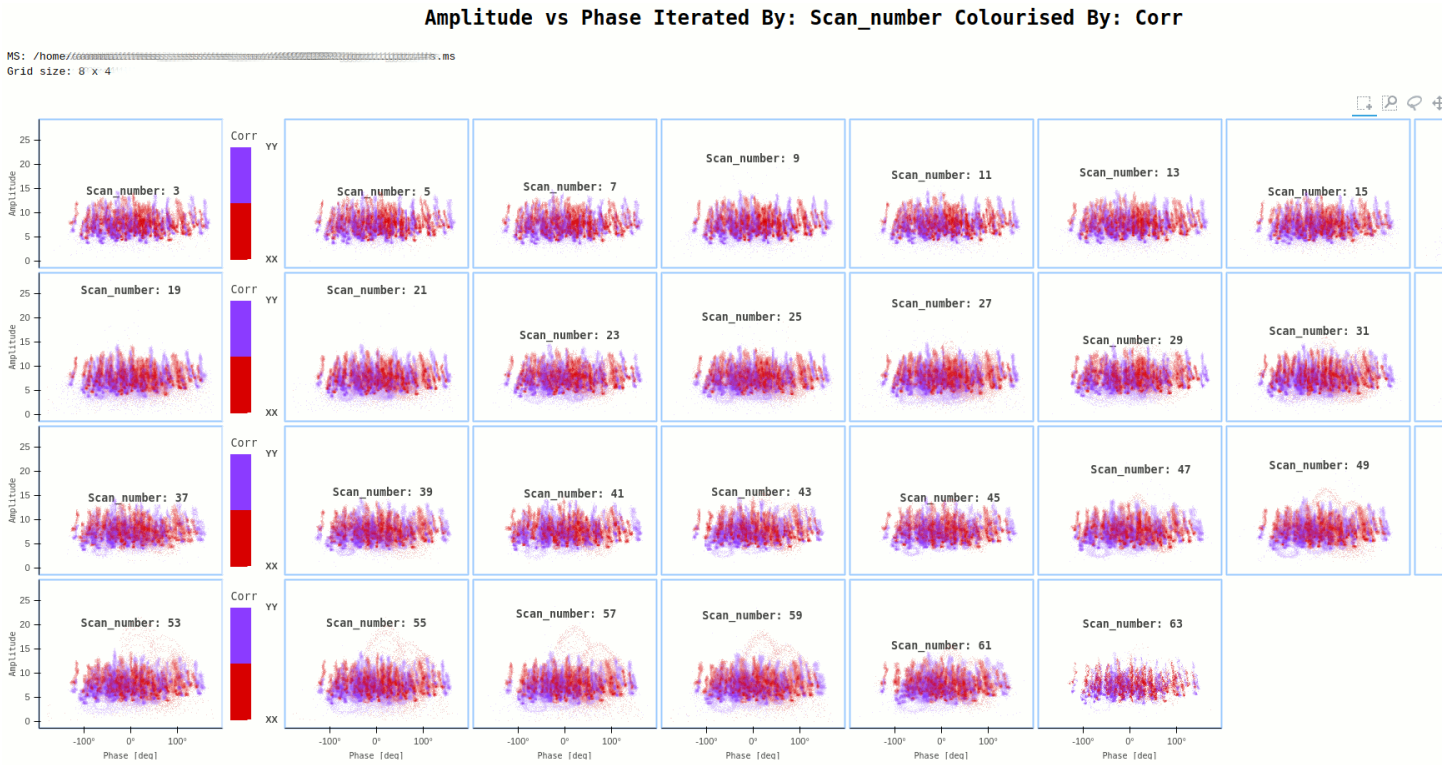


Command: `ragavi-vis --ms dir/test.ms --xaxis phase --yaxis amp --iter-axis scan`



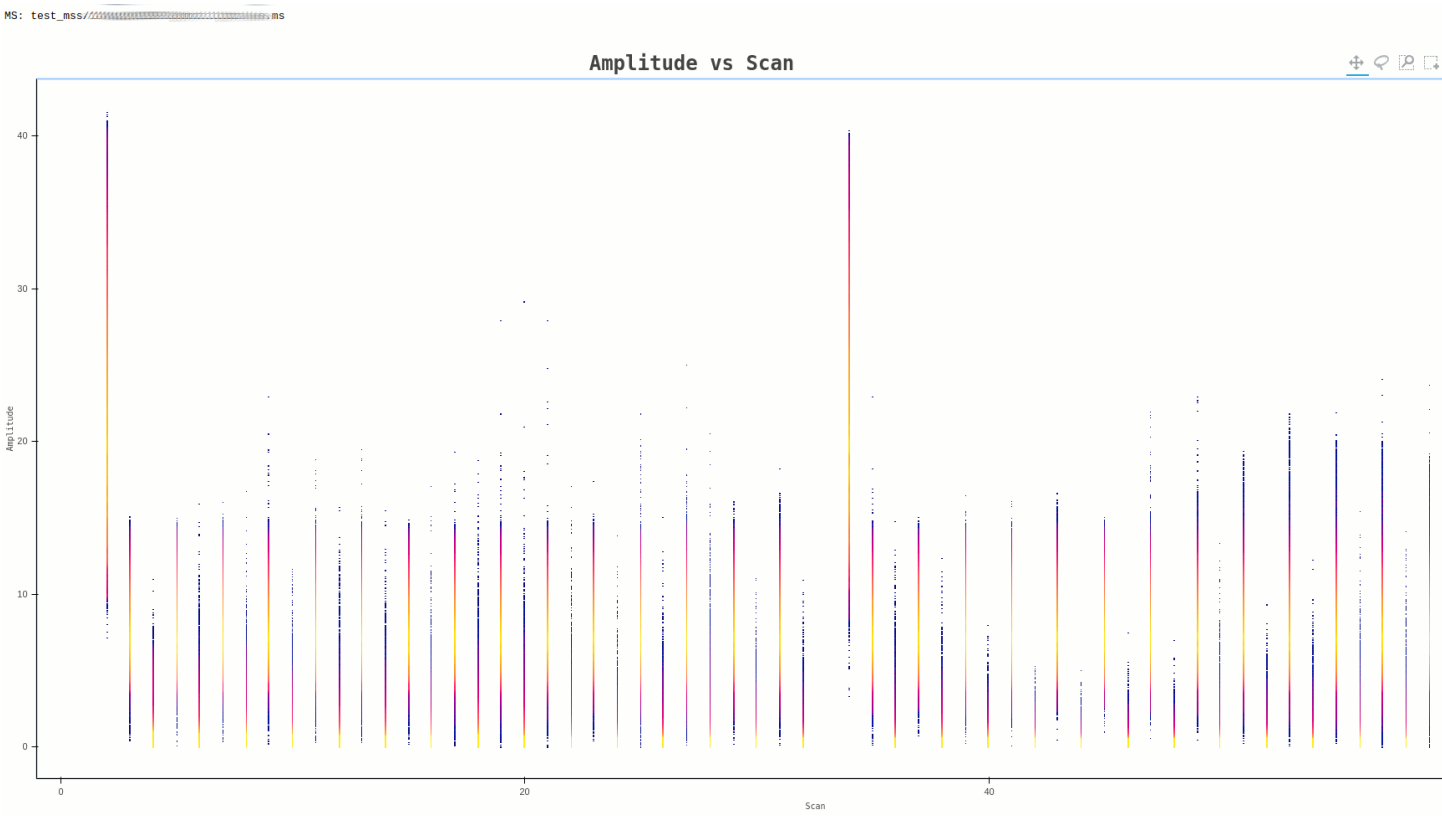
Command: `ragavi-vis --ms dir/test.ms --xaxis phase --yaxis amp --iter-axis scan`

```
--colour-axis corr
```



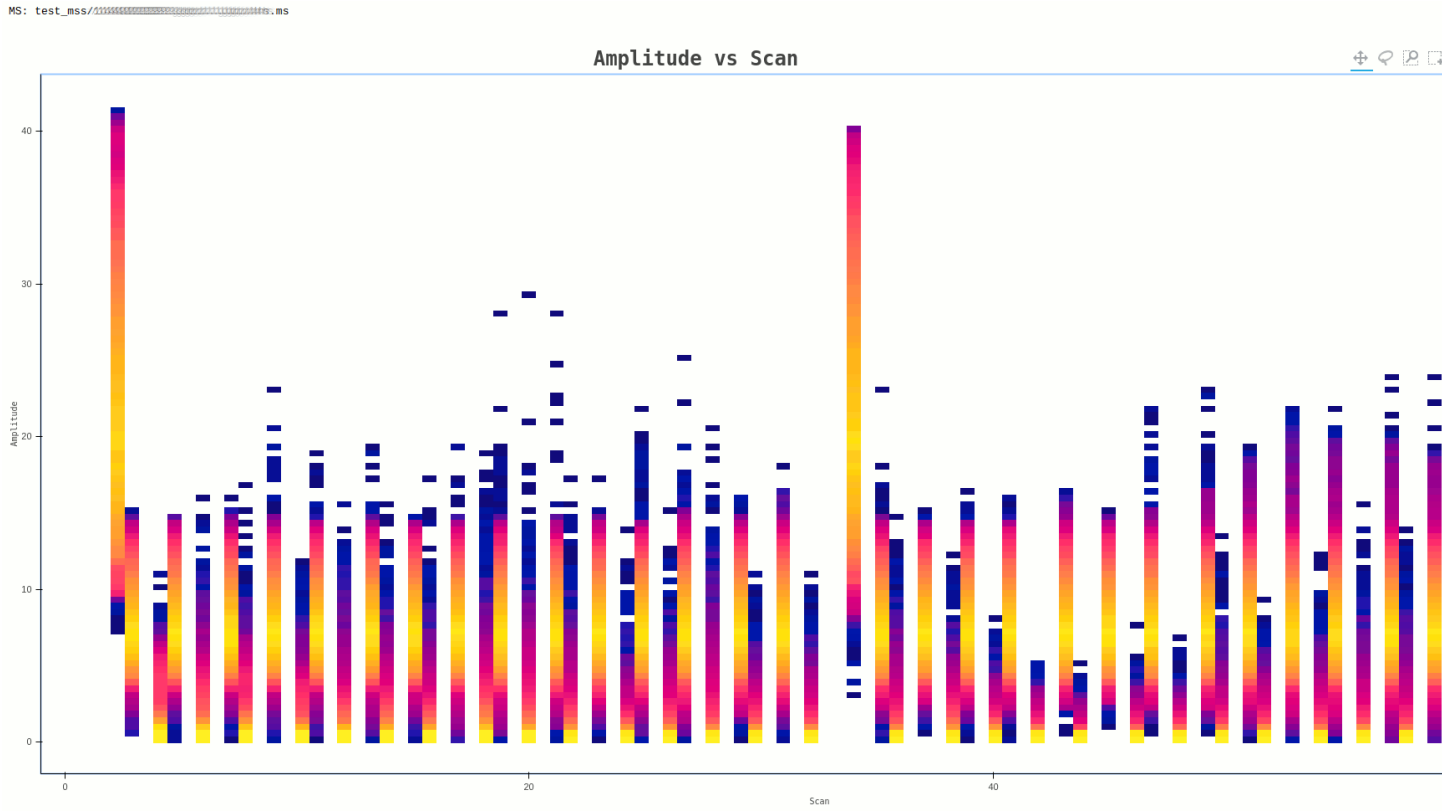
In some cases, the plot generated may have data points that are small and bordering invisible, such as the one below

Command: `ragavi-vis --ms dir/test.ms --xaxis scan --yaxis amp --cmap bmy`



A work around for such a case, is to reduce the resolution of the resulting image, by providing the `--canvas-width` and `--canvas-height` arguments. The image above, was generating using the default values 1080 and 720 respectively. If this is changed to 100 and 100, the resulting plot albeit coarse, is clearer as shown below

Command: `ragavi-vis --ms dir/test.ms --xaxis scan --yaxis amp --cmap bmy --canvas-width 100 --canvas-height 100`


MS: test_mss/
.ms
Amplitude vs Scan
Amplitude
Scan
0
10
20
30
40
0
20
40

CHAPTER 5

Utilities

Some utilities useful internally and externally from `ragavi`

`ragavi.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.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.utils.`**`calc_phase`**(*ydata*, *unwrap=False*)
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.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.utils.`**`calc_unique_bls`**(*n_ants=None*)
Calculate number of unique baselines :param n_ants: Available antennas :type n_ants: `int`

Returns **pq** (`int`) – Number of unique baselines

`ragavi.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.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.utils.**ctext**(*text*, *colour='green'*)
Colour some terminal output

ragavi.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.utils.**get_cmap**(*cmap*, *fall_back='coolwarm'*, *src='bokeh'*)
Get Hex colors that form a certain cmap. This function checks for the requested cmap in bokeh.palettes and colorcet.palettes. List of valid names can be found at: https: // colorcet.holoviz.org / user_guide / index.html https: // docs.bokeh.org / en / latest / docs / reference / palettes.html

ragavi.utils.**get_diverging_cmap**(*n_colors*, *cmap1=None*, *cmap2=None*)
Produce n_colors that diverge given two different colourmaps. This function depends on pl.diverging_palettes whose doc can be found at: https: // docs.bokeh.org / en / latest / docs / reference / palettes.html cmap1: : obj: *str*

Name of the first colourmap to use

cmap2: [obj: *str*] Name of the second colourmap to use

n_colors: [obj: *int*] Number of colours to generate.

Returns **colors** (: obj: *list*) – A list of size n_colors containing the diverging colours

ragavi.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.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.utils.`**`get_frequencies`**(*ms_name*, *spwid=None*, *chan=None*, *cbin=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
- **`cbin`** (`int`) – Number of channels to be binned together. If a value is provided, averaging is assumed to be turned on
- **`ms_name`** (`str`) – Name of MS or table
- **`spwid`** (`int` of `slice`) – Spectral window id number. Defaults to 0. If slicer is specified, frequencies from a range of spectral windows will be returned.

Returns **frequencies** (`xarray.DataArray`) – Channel centre frequencies for specified spectral window or all the frequencies for all spectral windows if one is not specified

`ragavi.utils.`**`get_linear_cmap`**(*cmap*, *n*, *fall_back='coolwarm'*)

Produce n that differ linearly from a given colormap This function depends on pl.linear_palettes whose doc be found at: https: // docs.bokeh.org / en / latest / docs / reference / palettes.html cmap: : obj: *str*

The colourmap chosen

n: [obj: *int*] Number of colours to generate

Returns **colors** (: obj: *list*) – A list of size n containing the linear colours

`ragavi.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.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.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[9]

`ragavi.utils.`**`slice_data`**(*inp*)

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

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

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.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.utils.**time_wrapper**(*func*)
A decorator function to compute the execution time of a function

ragavi.utils.**update_log_levels**(*in_logger*, *level*)
Change the logging level of the parent plotter

ragavi.utils.**update_logfile_name**(*in_logger*, *new_name*)
Change the name of the resulting logfile

CHAPTER 6

Appendix

6.1 Gains

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.**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 containing renderers sharing the same labels grouped together.

ragavi.ragavi.**corr_select_callback**()
Correlation selection callback

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 for plot

Returns **lo_ax1** (`dict`) – Dictionaries with legend objects for a plot

ragavi.ragavi.**create_stats_table**(*stats*, *yaxes*)

Create data table with median statistics :param stats: List of lists containing data stats for each iterations from

ragavi.ragavi.stats_display()

Parameters **yaxes** (`list`) – Contains y-axes for the current plot

Returns *Bokeh column layout containing data table with stats*

ragavi.ragavi.**errorbar**(*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
- **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=[]*, *group_cols=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** (list, optional) – TAQL where clause to be used within the table.

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 initial and final TIME column before selections

Returns **init_time, f_time** (tuple) – Containing initial time and final time available in the ms

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

Parameters

- **xaxis** (str) – Current xaxis
- **xds_table_obj** (*xarray.Dataset*) – xarray-ms table object
- **freqs** (np.array) – An array containing frequencies for the current SPW

Returns

- **spw_id** (numpy.ndarray) – Spectral window ids
- **scan_no** (numpy.ndarray) – scan ids

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

Returns code (str)

ragavi.ragavi.**link_plots**(*all_figures=None*, *all_fsources=None*, *all_ebars=None*)
Link all plots generated by this script. Done to unify interactivity within the plots such as panning, zoomin etc.

Parameters

- **all_figures** (list) – Containing all the figures generated in the plot
- **all_fsources** (list) – All data sources containing flagged data for all figures
- **all_ebars** (list) – All error bars for all figures

Returns all_fsrc, all_ufsrc (tuple) – Unified flagged and un-flagged data sources

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

ragavi.ragavi.**make_plots**(*source*, *fid=0*, *color='red'*, *yerr=None*, *yidx=None*)
Generate a pair of plots

Parameters

- **fig** (bokeh.plotting.figure) – First figure
- **color** (str) – Glyph color[s]
- **fid** (int) – field id number to set the line width
- **source** (bokeh.models.ColumnDataSource) – Data source for the renderer
- **yerr** (numpy.ndarray, optional) – Y-axis error margins
- **yidx** (int) – Current enumerated y-axis number. Used for keeping track of the figures.

Returns

- **p_glyph, ebars** ((bokeh.models.Glyph, bokeh.models.Whisker))
- Tuple of containing glyphs and error bars to be used (from *ragavi.ragavi.errorbar()*).

ragavi.ragavi.**make_table_name**(*tab_name*)
Create div for stats data table

ragavi.ragavi.**plot_table**(***kwargs*)
Plot gain tables within Jupyter notebooks. Parameter names correspond to the long names of each argument (i.e those with –) from the *ragavi-vis* command line help

Parameters

- **table** (str or list) – The table (list of tables) to be plotted.
- **ant** (str, optional) – Plot only specific antennas, or comma-separated list of antennas.
- **corr** (int, optional) – Correlation index to plot. Can be a single integer or comma separated integers e.g "0,2". Defaults to all.
- **cmap** (*str, optional*) – Matplotlib colour map to use for antennas. Default is coolwarm

- **ddid** (int) – SPECTRAL_WINDOW_ID or ddid number. Defaults to all
- **doplot** (str, optional) – Plot complex values as amp and phase (ap) or real and imag (ri). Default is "ap".
- **field** (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.
- **k-xaxis** (str) – Choose the x-xaxis for the K table. Valid choices are: time or antenna. Defaults to time.
- **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**(*name*, *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_static_image**(*fname*, *figs=None*, *batch_size=16*, *cmap='viridis'*, *dpi=None*)
Save plots in png, ps, pdf, svg format

Parameters

- **name** (str) – Desired image name
- **figs** (list) – A list containing bokeh.plotting.Plot objects (The figures to be plotted.)

ragavi.ragavi.**set_tempdir**(*name*)
Set the current dir as the temp dir also

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

Returns code (str)

ragavi.ragavi.**spw_select_callback**()
SPW selection callaback

ragavi.ragavi.**stats_display**(*tab_name*, *yaxis*, *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
- **f_names** (list) – List with all the available field names
- **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().

- **flag** (bool) – Whether to flag data or not
- **spwid** (int) – Spectral window to be selected
- **yaxis** (str) – Can be amplitude, phase, real, imaginary or delay

Returns *List containing stats*

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.**antenna_iter**(*ms_name*, *columns*, ***kwargs*)
Return a list containing iteration over antennas in respective SPWs :param ms_name: Name of the MS :type ms_name: str :param columns: Columns that should be present in the dataset :type columns: list

Returns **outp** (list) – A list containing data for each individual antenna. This list is ordered by antenna and SPW.

ragavi.visibilities.**append_cbar**(*cats*, *category*, *cmap*, *ax*, *x_min*, *y_min*, *labels=None*)
Add a colourbar 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 :param labels: Labels containing names for the iterated stuff :type labels: list :param ax: Figure to append the color bar to :type ax: bokeh.models.figure

ragavi.visibilities.**corr_iter**(*subs*)
Return a list containing iteration over corrs in respective SPWs

Parameters **subs** (list) – List containing subtables for the daskms grouped data

Returns

- **outp** (list) – A list containing data for each individual corr. This list is ordered by corr and SPW.
- **# NOTE** (*can also be used for chan iteration. Will require name change*)

ragavi.visibilities.**create_bl_data_array**(*xds_table_obj*, *bl_combos=False*)
Make a dataArray containing baseline numbers

Parameters

- **xds_table_obj** (xarray.Dataset) – Daskms dataset object
- **bl_combos** (Bool) – Whether to return only the available baseline combinations

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

ragavi.visibilities.**create_categorical_df**(*it_axis*, *x_data*, *y_data*, *xds_table_obj*)

it_axis: **str** Column over which to iterate / colourise

x_data: **xr.DataArray** x-axis data

y_data: **xr.DataArray** y-axis data

xds_table_obj: `xr.Dataset` Daskms partition for this chunk of data

Returns

- **xy_df** (`dask.DataFrame`) – Dask dataframe with the required category
- **cat_values** (`np.array`) – Array containing the unique identities of the iteration axis

`ragavi.visibilities.`**`create_dask_df`**(*inp*, *idx*)

Parameters

- **`inp`** (`dict`) – A dictionary containing column name as the key, and the dictionary value is the dask array to be associated with the column name.
- **`ids`** (`da.array`) – A dask array to form the index of the resulting dask dataframe

Returns **ddf** (`dd.Dataframe`) – Dask dataframe containing the presented data

`ragavi.visibilities.`**`create_df`**(*x*, *y*, *iter_data=None*)

Create a dask dataframe from input x, y and iterate columns if available. This function flattens all the data into 1-D Arrays

Parameters

- **`x`** (`dask.array`) – Data for the x-axis
- **`y`** (`dask.array`) – Data for the y-axis
- **`iter_ax`** (`dask.array`) – iteration axis if possible

Returns **new_ds** (`dask.Dataframe`) – Dataframe containing the required columns

`ragavi.visibilities.`**`gen_image`**(*df*, *x_min*, *x_max*, *y_min*, *y_max*, *c_height*, *c_width*, *cat=None*, *c_labels=None*, *color=None*, *i_labels=None*, *ph=1080*, *pw=1920*, *x_axis_type='linear'*, *x_name=None*, *x=None*, *xlab=None*, *y_axis_type='linear'*, *y_name=None*, *ylab=None*, *title=None*, *xds_table_obj=None*, ***kwargs*)

Generate single bokeh figure

`ragavi.visibilities.`**`get_ms`**(*ms_name*, *ants=None*, *cbin=None*, *chan_select=None*, *chunks=None*, *corr_select=None*, *colour_axis=None*, *data_col='DATA'*, *ddid=None*, *fid=None*, *iter_axis=None*, *scan=None*, *tbin=None*, *where=None*, *x_axis=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.
- **`ants`** (`str`) – Values for antennas in ANTENNA1 whose baselines will be selected
- **`cbin`** (`int`) – Number of channels binned together for channel averaging
- **`colour_axis`** (`str`) – Axis to be used for colouring
- **`iter_axis`** (`str`) – Axis to iterate over
- **`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
- **x_axis** (:obj:‘) – The chosen x-axis
- **where** (str) – TAQL where clause to be used with the MS.
- **chan_select** (int or slice) – Channels to be selected
- **corr_select** (int or slice) – Correlations to be selected

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

ragavi.visibilities.**link_grid_plots**(*plot_list*, *ncols*, *nrows*)
Link all the plots in the X and Y axes

ragavi.visibilities.**massage_data**(*x*, *y*, *get_y=False*, *iter_ax=None*)
Massages x-data into a size similar to that of y-axis data via the necessary repetitions. This function also flattens y-axis data into 1-D.

Parameters

- **x** (xr.DataArray) – Data for the x-axis
- **y** (xr.DataArray) – Data for the y-axis
- **get_y** (bool) – Choose whether to return y-axis data or not

Returns

- **x** (dask.array) – Data for the x-axis
- **y** (dask.array) – Data for the y-axis

ragavi.visibilities.**mod_unlinked_grid_plots**(*plot_list*, *nrows*, *ncols*)

- Move tick marks into the plots
- Reduce frame size of plots
- Switch on only axis label for first item in row

ragavi.visibilities.**validate_axis_inputs**(*inp*)
Check if the input axes tally with those that are available :param inp: User’s axis input :type inp: str :param choices: Available choices for a specific axis :type choices: list :param alts: All the available altenatives for the various axes :type alts: dict

Returns **oup** (str) – Validated string

CHAPTER 7

Changelog

7.1 0.5.2

ragavi-gains

- Return -g/–gaintype and -kx/–k-xaxis for backward compatibility

7.2 0.5.1

ragavi-gains

- Gives default x-axis to unknown tables
- Fixes partially hidden widget box sizes
- Changes default x-axis for D-tables to time

7.3 0.5.0

ragavi-vis

- Fix corr selector again

ragavi-gains

- Remove the *–gaintype* argument
- Adds polcal tables Kcross, Xf and Df to allowed plots
- Changes *-kx/–k-xaxis* to *-x–xaxis* to allow for arbitrary specification of desired x-axis
- Adds *-y/ –yaxis* as an alternative to *-d/–doplot*

- Forces the temp directory ragavi uses to be the same as where ragavi is being ran. Attempt to fix "Out of memory error"

7.4 0.4.3

ragavi-gains

- Fix bug in flagged data display
- Fixes link between batch selection and other selections

7.5 0.4.2

ragavi-vis

- Fixes correlation selection using corr label bug
- Fixes mismatched colour map labels for coloured plots
- Improved appearance of colour bar
- Prevents crash if all data in a plot is flagged
- Iterated plots now not linked by default. Added argument `-lp / --link-plots` to allow plots to be linked. All plots thus have independent axes by default.

7.6 0.4.1

ragavi-gains

- Supports all `matplotlib` type static outputs
- Generate additional static plot for interactive plots with more than 30,000 points
- Multiple tables also supported with static plots
- Adds option `all` to plot all amplitude, phase, real and imaginary plots in a single plot

ragavi-vis

- Added colour bar to coloured iterated plots

7.7 0.4.0

general

- Added arguments `--debug` and `-lf / --logfile` to allow the enabling of debug messages and specification of desired log file names respectively
- Added argument `-v` / `--version` to display the current `ragavi` version.
- Some improvements in logging

ragavi-gains

- Fixed issue #70

- Fixed issue #69

ragavi-vis

- Changed argument `-nf / --no-flagged` to `-if / --include-flagged` for clarity.
- Better handle of RAM usage

7.8 0.3.7

ragavi-gains

- Fixed legend display bug
- `--gain-type` argument is now optional
- Static and interactive files can be generated simultaneously by specifying both `--plotname` and `--htmlname`

general

- Fixed bug in phase wrapping

7.9 0.3.6

ragavi-vis

- Fixed bug in correlation selection

7.10 0.3.5

ragavi-vis

- Added information on read the docs
- Fix error due to all null plots

7.11 0.3.4

ragavi-gains

- Fixes data selection bugs

7.12 0.3.3

- **ragavi-vis**
  - Both `--colour-axis` and `--iter-axis` can now be used simultaneously
  - Added `--cols` to specify number of columns when `--iter-axis` is supplied.
  - Allowed selection of correlations via respective labels
  - All valid y-axis names are also valid in the x-axis

  - Changed UVWave scale from Lambda to Kilo Lambda
  - Improved averaging speed
  - Added `--ant` for antenna selection
- **ragavi-gains**
  - Deprecated the use of the name `ragavi`. Now uses `ragavi-gains`
  - Slightly improved layout

7.13 0.3.2

- **ragavi-vis**
  - Added `--iter-axis` argument to generate grid plots over the iteration axis
  - `--colour-axis` generates a single plot coloured by the specified axis
  - Added `--canvas-width` and `--canvas-height` option to set resulting image resolution
  - Improved RAM management
  - Available X and Y axes names can be specified in short forms like CASA

7.14 0.3.1

- 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 x-axis to UTC time
  - Added new download selected data button
  - All available times displayed for bandpass plots

7.15 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

7.16 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)

7.17 0.2.1

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

7.18 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.

7.19 0.1.0

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

7.20 0.0.9

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

7.21 0.0.8

- Fixed bug due to string encoding for python2.7

7.22 0.0.7

- Updated version number

7.23 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
- Re-organise selector panel
- Added title and axis label size selectors
- Add field symbols alongside field names on check-boxes
- Allow automatic plot scaling
- Medians now shown in plot titles

7.24 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

7.25 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

7.26 0.0.3

- Travis release on tag
- Now plotting Flux calibration tables
- Extra frequency axis for bandpass plot

7.27 0.0.2

- Module importable
- Table parameter option

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

r

Index