ngVLA Memo No. 59:
QAC: Quick Array Combinations with CASA

Peter Teuben
University of Maryland, College Park, MD
`teuben@astro.umd.edu`
January 3, 2020

Abstract

QAC is a simple python layer in CASA, developed to aid in writing scripts for array (single dish and interferometric) combinations and simulations. Although initially developed for TP2VIS, running simulations and comparing with other array combination methods, this package turned out to be useful for array design studies as well. Both ALMA and ngVLA simulations are already supported, but extending to more generic arrays is straightforward. This memo complements ngVLA memo 54, where QAC V0.3[1] was used for an array design study. QAC is probably less useful for real data, where more CASA flexibility might be needed, but does contain some useful functions that do not depend on the simulation environment.

1 Introduction

CASA ([1], [2]) is a general purpose python interface to radio astronomy software. It handles interferometric as well as single dish data, all the way from ingestion, calibration and imaging to analysis. Most ALMA and VLA data are now routinely processed with CASA using a custom built pipeline. CASA uses object oriented "tools", as well as the more classic python functions, called "tasks" in CASA. One can write very complex procedures this way, and in fact, the ALMA/VLA pipeline is an example of such an interface. The QAC interfaces we discuss in this memo were also designed with a specific goal of testing the combination of single dish and interferometric data. They are also a convenient method to study array design, in this case for the ngVLA. Some results have been published in ngVLA memo 54 [3].

The development of QAC started in 2017 with the TP2VIS project [4], to provide a more easily programmable interface, orchestrate simulations and provide a reproducable baseline using regressions. It can be obtained from `https://github.com/teuben/QAC` and is also registered in the ASCL as `ascl:1908.001`.

We first summarize the different methods how CASA can be extended by using your own custom built python code, then how QAC is installed, and a typical usage. We also give a short summary of the API, and publish a benchmark in the Appendix. Given that QAC is available in github, you will likely find updates to QAC (and this memo) in this repository.

Several other efforts have been going on wrapping CASA tasks and tools in a more convenient environments, e.g. casanova [2], ADMIT [3], and the calibration and imaging pipelines for ALMA and VLA [4]

[1]this memo describes QAC v0.6 as of December 2019, please update as versions change
[2]`https://github.com/kaspervd/casanova`
[3]`http://admit.astro.umd.edu/`
[4]Included with CASA via `https://casa.nrao.edu/casa_obtaining.shtml`

2 Running python code in CASA

CASA interacts with the user in an interactive python (ipython) session. For most users adding C++ code to CASA is a complex operation, but installing new python interfaces to ease writing CASA scripts is usually fairly straightforward, and nowadays most users are familiar with this language. Several methods (and hybrid between these) exist for CASA:[5]

1. `buildmytasks`

   This is the native CASA method of installing a real CASA task. The CASA Cookbook describes a procedure to install new CASA tasks, but at the same time warns this method may get deprecated. Nonetheless, this so-called "buildmytasks" has been used by other teams, most notably by the Nordic ARC node[6]. This is typically run once inside the directory where your `foo.py`, `foo.xml`, and other material is present, after which the code, and documentation, gets installed at the right place inside the CASA tree. Users can then run this task with the CASA command

   ```
   foo(1,'b',[1,2,3])
   ```

   Note that in this example `foo()` is now a true CASA task, with its `default / inp / go / tget / tput` interface.

2. `import foo`

   The traditional way a user includes software to a python based system would be the python `import` command. This is fine for stable software, and can be installed with python's `setuptools` in CASA. In the future CASA6 one should be able to use virtualenv to test out software like this without the need to write into CASA's personal space. One can also consider the use of using `$PYTHONPATH` to point to the directory where `foo.py` is present, but this method can easily conflict with other installation methods (in fact, is strongly discouraged in a CASA environment).

   ```
   foo.bar(1,'b',[1,2,3])
   ```

   In this, and all following examples below, `foo()` is just a python function, not a CASA task.

3. `execfile('foo.py')`

   Will execute the code, after which an API is available (note this will not work in python3 anymore). This is the method we used in QAC. Notice that no command line parameters can be passed into the code. The code, and variables, defined through `execfile()` are immediately available in the CASA session.

   ```
   foo_bar(1,'b',[1,2,3])
   ```

   Incidentally, if these are combined and only one script needs to be executed and then analyzed outside of CASA, a very efficient way it to use could be to call casa from the command line, e.g. directly from bash (or via a Makefile):

[5]some of these methods are expected to become more common in CASA6
[6]`https://www.oso.nordic-alma.se/software-tools.php`

```
% casa --nogui -c foo.py a=1 b='"b"' c='[1,2,3]' > foo.log 2&>1
```

The overhead of setting up CASA before this script really starts work varies a lot depending on cashing and what's in the casa init files, but can be anywhere from 5 to 20 seconds. If many of these scripts are to be run, and each only takes a short time, the overhead will be too large, and alternative method will need to be employed (see below).

4. `run foo.py p1 p2 p3`

   Since CASA is essentially an ipython shell, the ipython `run` command can be used to execute a script, including conveniently parsing "command line arguments". This will need a parser for `p1=sys.argv[1]`, `p2=sys.argv[2]`, etc. note this is an ipython interface, not python, though it's similar to running in the unix shell `python foo.py p1 p2 p3`, but note this is different from the CASA method where local python variables can directly be set via the commandline without the need for a parser.

   ```
   run foo.py 1  'b'  [1,2,3]
   ```

5. `%run -m foo`

   Runs the foo module (from `sys.path`). In the current CASA manipulating `sys.path` is not recommended, the arguments similar to those of not using `$PYTHONPATH`

Miles Lucas made his summer-2018 toolkit Radio Imaging Combination Analysis (RIKA) available[7]. He uses the 4th (`run`) method. In QAC we decided to use the 3rd (`execfile`) method.

2.1 Installing QAC

QAC needs CASA to be installed, and the user can opt either to install a version of CASA from within QAC, or assume a version of CASA that is present on the system, i.e. there is a existing command called "casa". Since CASA startup can be controlled by `~/.casa/init.py` we choose this file to `execfile` the correct startup script, aptly named `casa.init.py` in the QAC distribution:

```
execfile(os.environ['HOME'] + '/.casa/QAC/casa.init.py')
```

in this example QAC was installed inside the `~/.casa` directory (or as a symlink to another location).
In our `casa.init.py` you will find a few examples of common packages loaded by CASA in `casa.init.py`.

3 Design Issues

QAC needs to be lightweight, and easy to install,

- Easy to install, ideally a one liner
- Easy to pass parameters into functions of scripts
- Consistent naming convention of functions and parameters

[7] `https://gitlab.com/mileslucas/RICA`

- Procedural. Although python has great support for a object oriented programming style, and plenty is used under the hood in CASA, for this simple interface a simple procedural path was chosen.
- clean vs. tclean. Although clean is formally not supported anymore, occasionally the two are compared, and this gives that option. See `qac_clean1(t=False)` to run the old clean.
- QAC works similar to CASA's `simobserve`, where all the work is done inside a designated directory. This is different from the CASA philosophy where users have more fine grained control over directories and filenames.

The use of a script with parameters is very useful for re-use, especially if the script also defines defaults. A drawback of the execfile approach is the lack to change these parameters. Add to this that execfile is not supported in python3, is a no-brainer not to use it, or at least switch to "run". There is another important difference, as was remarked before: `execfile()` causes code and variables to be shared (potentially overwritten), whereas `run` is the more pythonic approach, like `import`.

4 Example

A typical command line usage would be

```
% casa --nogui -c myscript.py scale=2.0 nants=46 niter='[0,1000,10000]'
```

5 Scripts

A typical simulation script might look as follows.

```
qac_ptr(phasecenter,"test123.ptg")

qac_vla("test123","skymodel.fits", 4096, 0.01,ptg="test123.ptg",phasecenter=
   ↪ phasecenter)

qac_clean1("test123/clean1",phasecenter=phasecenter)
```

6 Timing and Regression

Because QAC deal almost exclusively with image type data, a regression test can be invoked automatically with the statistics report, if a regression string is given, viz.

```
r = "0.0038324084555372423␣0.021439742878458009␣-0.048513446003198624␣
   ↪ 0.41929447650909424␣383.60327838373536"
qac_stats(test+'/clean/tpint.image')
qac_stats(test+'/clean/tpint_4.tweak.image', r)
qac_stats(test+'/clean/tpint_4.tweak.image', r, eps=1e-6)
```

where in the first instance only the statistics are reported, the second instance will also flag any deviations from the expected string. The numbers represent the mean, std, min, max and total flux of the image. The last example regresses these values within a relative accuracy of `eps`.

For measurement set data it will use statistics on the amplitudes, but the flux is reported at 0 (which technically is correct for most interferometers).

7 Benchmarks

In QAC/test the command "make bench" will activate a benchmark, which should last about 2-3 mins. It uses a subset of the M100 casaguide data, but due to a variety of CASA problems, we don't expect the regression to be accurate until we release QAC V1.0.

8 API

Here we list the most important functions available in QAC, without further details except for arguably descriptive parameter names and defaults. The full and updated documentation can be seen online on `https://github.com/teuben/QAC/blob/master/docs/qac.md`.

```
  ## Adminstrativia

  qac_argv(sysargv)

  qac_begin(label='QAC', log=True, plot=False)

  qac_end()

  qac_log(message, verbose=True)

  qac_project(projectdir, chdir=False)

  qac_version()
```

```
## Simulation routines

  qac_vla(project, skymodel, imsize, pixel, phasecenter, freq, cfg, ptg, noise)

  qac_alma(project, skymodel, imsize, pixel, phasecenter, freq, cycle, cfg, ptg)

  qac_carma(project, skymodel, imsize, pixel, phasecenter, cfg, ptg, times)

  qac_noise(noise, *args, **kwargs)

  qac_clean1(project, ms, imsize, pixel, niter, weighting, startmodel, phasecenter,
      ↪  t, do_concat, **kwargs)

  qac_clean(project, tpms, mslist, imsize, pixel, weighting, startmodel,
      ↪ phasecenter, niter, do_concat, do_int, do_cleanup, **kwargs)

  qac_tp_otf(project, skymodel, dish, label, freq, template)

  qac_smooth(project, skymodel, label, niteridx)

  qac_tp_vis(project, imagename, ptg, pixel, niter, phasecenter, rms, maxuv, nvgrp,
      ↪  fix, deconv, **kwargs)
```

```
 qac_tweak(project, name = "dirtymap", niter = [0], **kwargs)

 qac_vp(vp=False, schwab=False)

 qac_tpdish(name, size=None)

 qac_feather(project, highres, lowres, label, niteridx, name="dirtymap")

 qac_ssc(project, highres, lowres)
```

```
## Analysis and Plotting routines

 qac_stats(image, test=None, eps=None, box=None, pb=None, pbcut=0.8, edge=False)

 qac_beam(im, normalized=True, chan=-1, plot=None)

 qac_phasecenter(im)

 qac_ptg(ptg, ptgfile=None)

 qtp_im_ptg(phasecenter, imsize, pixel, grid, im=[], rect=False, outfile=None)

 qac_ms_ptg(msfile, outfile=None, uniq=True):

 qac_summary(tp, ms=None, source=None, line=False)

 qac_math(outfile, infile1, oper, infile2)

 qac_mom(imcube, chan_rms, pb=None, pbcut=0.3, rms=None)

 qac_plot(image, channel=0, box=None, range=None, mode=0, title=None, plot=None)

 qac_flux(image, box=None, dv=1.0, plot='qac_flux.png')

 qac_niter_flux(dirname, box=None, flux=True, plot=None)

 qac_fidelity(model, image, figure_mode=5, diffim=None, absdiffim=None, fidelityim
    ↪ =None, absmodelim=None, interactive=False)

 qac_psd(image, plot='qac_psd.png', fit=False, pixel_s=None)

 qac_line(im)

 qac_fits(image,outfile=None,overwrite=True)

 qac_ds9(image, cleanup=False)
```

```
  qac_ingest(tp, tpout = None, casaworkaround=[1,3], ms=None, ptg=None)
```

9 Pros and Cons

As was outlined above, using QAC should result in shorter and simpler scripts, but can easily orchestrate large sets of simulations from the commandline or inside another python script. However, users would have to install QAC and learn to use this interface. With the complexity of CASA this is another hurdle. Also, we found CASA bugs that show up in QAC can be harder to explain to the CASA developers. Certainly giving a code example using QAC is not an accepted practice. Finally, QAC is great for simulations, but does not always expose all the rich parameters that full CASA tasks have.

10 Future

CASA is a development project, with frequent updates. The next release (V6) will have a major overhaul how python and the C++ libraries are integrated, and this will likely have some effect how QAC is installed, although less on its API. Ideally we like to switch to the `import` or `run` method once the CASA imports are standardized. I expect QAC will avoid software rot and adopt the new style. Look out for QAC V2.0

Acknowledgements: Jordan Turner and Sara Negussie have been patient contributers and users. Part of QAC was developed under the ALMA development study "TP2VIS" (PI: Jin Koda) and the "ngVLA" array combination study (ngVLA memo 54).

References

[1] J. P. McMullin, B. Waters, D. Schiebel, W. Young, and K. Golap. CASA Architecture and Applications. In R. A. Shaw, F. Hill, and D. J. Bell, editors, *Astronomical Data Analysis Software and Systems XVI*, volume 376 of *Astronomical Society of the Pacific Conference Series*, page 127, October 2007.

[2] B. Emonts. CASA Architecture and Applications. In P. J. Teuben, M. W. Pound, B. Thomas, and E. M. Warner, editors, *Astronomical Data Analysis Software and Systems XXVIII*, volume 376 of *Astronomical Society of the Pacific Conference Series*, page 127, October 2019.

[3] Jordan Turner, Peter Teuben, and Daniel Dale. Short Spacing Issues for Mapping Extended Emission: Milky Way Case Study. Technical Report 54, Jan 2019.

[4] Jin Koda, Peter Teuben, Tsuyoshi Sawada, Adele Plunkett, and Ed Fomalont. Total Power Map to Visibilities (TP2VIS): Joint Deconvolution of ALMA 12m, 7m, and Total Power Array Data. *Publications of the Astronomical Society of the Pacific*, 131(999):054505, May 2019.

Appendix A: Sample Code

To run a large suite of simulations, it can be very useful to call CASA from the Unix command line, and loop over many parameters, or organize them in a Makefile (see e.g. `QAC/test/Makefile`). For example:

```
casa --nogui -c vla1.py pixel_m=0.05 niter='[0,5000,15000]' dish=45 pdir='"exp102
   ↪ "'
```

As one of the products of the "tp2vis" ALMA development study ([4]) we continued the development of the QAC toolkit that simplifies writing some of these complex scripts. It also allow us to use a different combination method (feather, tp2vis, ssc etc.) with minimal changes to the simulations scripts.

As an example, consider the `simplenoise` procedure[8] to add a given noise to a simulation. Here is the example calling `qac_vla()` twice, in the end generating a Measurement Set with the correct 1 mJy/beam noise:

```
rms = 0.002 # request 2 mJy/beam RMS noise (NA)
ms1 = qac_vla(pdir,model, noise=-rms) # noise<0 triggers it to compute the rms
sn0 = qac_noise(noise,pdir+'/noise', ms1) # get scaling factor from rms in ms1
ms2 = qac_vla(pdir,model, noise=sn0) # MS that with correct "rms" in Jy/beam
```

In the first Measurement Set a noise level is computed for a fixed 1 Jy noise per visiblity on a zero model. The noise in the resulting dirty map, computed in `qac_noise()`, is then the scaling factor (`sn0` that needs to be applied to get the correct requested noise level in the second Measurement Set.

[8]See `simulator.setnoise`

Appendix B: Sample Simulation

Here we show how Figure 1 was made. Using a complex molecular cloud structure, what is the fidelity of the combined image when a single dish measurement was feathered into the interferometric array data

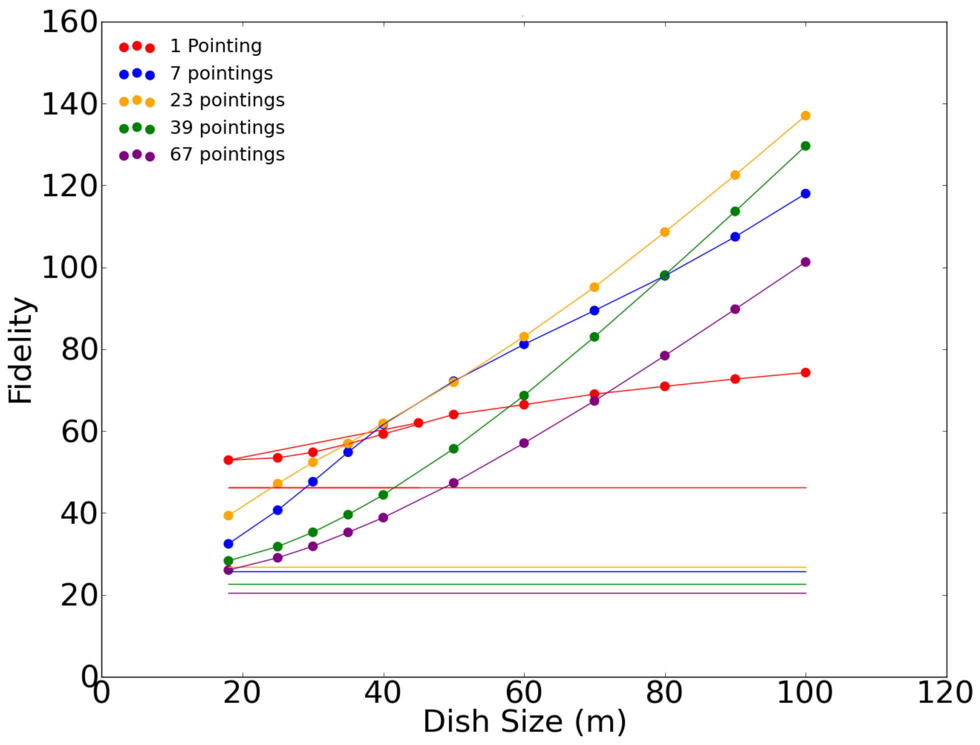

Figure 1: Fidelity (higher is better) of feathering the single dish image of given Dish Size with the simulated ngVLA array.

```
casa -c mapping2.py niter='[0,10000,20000]' dish=50 pixel_m=.02 pdir='"exp1"' >
    ↪ exp1.log 2>&1
casa -c mapping2.py niter='[0,10000,20000]' dish=50 pixel_m=.01 pdir='"exp2"' >
    ↪ exp2.log 2>&1
casa -c mapping2.py niter='[0,10000,20000]' dish=50 pixel_m=.015 pdir='"exp3"' >
    ↪ exp3.log 2>&1
casa -c mapping2.py niter='[0,10000,20000]' dish=50 pixel_m=.026 pdir='"exp4"' >
    ↪ exp4.log 2>&1
```

Appendix C: Installing QAC

Here is a step-by-step example of installing QAC, but assuming CASA has been installed before. The file `QAC/docs/install.md` contains an example how to install CASA within QAC, and in the directory `QAC/casa/` several examples exist how to install CASA plugins from other groups (e.g. SD2VIS, SDINT, TP2VIS, au)

```
    git clone https://github.com/teuben/QAC
    cd QAC
    make install
    casa
```

you should now see (amongst) in the screen logs:

```
QAC: Root  /home/teuben/.casa/QAC
QAC: Load  src/qac.py
QAC: Load  src/ssc.py
QAC: Load  src/plot.py
QAC: Load  contrib/tp2vis.py
QAC: Skip  distribute/tp2vis.py
QAC: Skip  tp2vis/tp2vis.py
QAC:  qac: version 14-mar-2019
qac_root: /home/teuben/.casa/QAC
casa:5.4.1-32
data:/home/teuben/QAC/casa/casa-release-5.4.1-32.el7/data
```

notice that only the experimental `contrib/tp2vis` is loaded here. To activate the formal public release of `tp2vis`, the following step are needed:

```
    cd QAC
    make tp2vis
    casa
```

and now you should see that `distribute/tp2vis.py` is loaded. if you are a developer, you can also load the latest developers release via QAC:

```
    cd QAC
    make dev
    casa
```

These versions are clumsily maintained via a symlink to the existing `.git` repos. Break the symlink by removing the version which you don't want to see:

```
    cd QAC
    rm distribute
    rm tp2vis
```

Appendix D: Benchmark

It is very useful to have a quick demonstration of the QAC software, and at the same time serve as a benchmark to see how fast you current machine measures. The following example also provides a regression test, although it should be noted within CASA they often will vary in small values.

The standard benchmark should take about 2-3 minutes to finish and produce about 300MB of data in the `QAC/test/bench/` directory. The `bench.log`

```
cd QAC
make data tp2vis
make bench
```

and you should now see something like this: [9]

```
running bench.py ...
time casa --nogui -c bench.py > bench.log 2>&1
tail -4 bench.log
QAC_STATS: bench/clean/tpint_2.tweak.image 0.003761955025079131 0.021406407411194268
  -0.047620095312595367 0.41968712210655212 375.9275617537877 FAILED regression
                                       0.0038472646829610813 0.021499494640955678
  -0.047635149210691452 0.4208904504776001 384.45247992991648 EXPECTED
840.24user 20.54system 2:33.76elapsed 559%CPU (0avgtext+0avgdata 1021020maxresident)k
61416inputs+3457328outputs (261major+644633minor)pagefaults 0swaps
```

[9]ignore “FAILED regression” until QAC is at version 1.0

Appendix E: Data Naming Conventions

Although the user is free to choose a naming convention, there is some structure imposed on certain QAC commands. As a result of using `tclean()` the extensions are also often normalized (i.e. not needed)

The typical order is first running `tp2vis`, then optionally running a `qac_clean1`, creating the `clean0` hierarchy. Then `qac_clean` in a Joint Deconvolution, creating the `clean1` hierarchy. Once INT and TPINT images have been produced, feather images can be created via `qac_feather` etc. This includes multiple iterations if the `niter` keyword was given as a python list of niter values, e.g. `niter='[0,10000,20000]'`.[10]

```
 M100/
      tp.ms                                 # default name of qac_tp_vis()
      clean0/dirtymap.model                 # default name of qac_clean1()
             dirtymap.image
             dirtymap.image.pbcor
             dirtymap.psf
             dirtymap.pb
             dirtymap_2.image
             dirtymap_2.image.pbcor
      clean1/int.image.pbcor                # default names in qac_clean()
             int_2.image.pbcor
             tpint.image.pbcor
             tpint.image.psf
             tpint.image.pb
             otf.image                      # see qac_tp_otf
             otf.image.pbcor
             feather.image                  # see qac_feather()
             feather_2.image.pbcor
             skymodel.smooth.image
```

[10]this is one of the QAC features. Not to be confused with the major and minor cycles of clean, this uses the last model as startmodel in the next entry for the niter list. Thus it runs `tclean` 3 times in this example.