Advanced CASA

Joshua Marvil

7th VLA Data Reduction Workshop

Introduction

This presentation will contain both informational and interactive content (see headings)

You can run the interactive content in your own CASA session during the talk

interactive_commands.txt

lists each interactive command if you want to copy/paste (but please only one line at a time)

INTERACTIVE

Introduction

This slide has interactive content

Please run the following in your own terminal

Raise your hand if you need assistance

```
# In a terminal window

cd ~/data/DRW/Lectures/L8_advanced_casa

casa -r 5.4.2-5
```

Outline

Advanced CASA overview

The CASA toolkit

CASA scripts and custom tasks

CASA Command Line Interface (CLI)

Custom Ipython Shell

- Python interpreter
- standard library
- additional modules
- iPython extensions
- ~/.casa/init.py

Tasking system

- inp
- go
- default
- tget
- tput
- tasklist
- taskhelp

Python (2.7)

Python objects, (e.g., int, float, bool, str, list, tuple, set, dict), and their methods

Other programming elements, e.g., operators, expressions, statements, syntax

Built-in functions, e.g., help, open, print, format, eval, exec, type, input

Python standard library

All standard modules are available in CASA

E.g., os, sys, re, glob, shutil, pickle, time, datetime

Full list of Python 2.7 standard library:
docs.python.org/2.7/library

```
# In CASA

import os
os.path.isdir( '3C75.ms' )
```

Additional Python modules

Several additional Python modules are available in CASA

Details are platform and version specific, but the following are typically available (usually not the most recent version):
numpy, scipy, matplotlib, dateutil, pytz, pyfits

```
# In CASA

import matplotlib.pyplot as plt

import numpy as np

plt.plot( np.random.randn(5) )
```

IPython magic commands

IPython has many enhancements over the standard interpreter

Designed for interactive use, mostly incompatible with scripting

Magic commands, preceded by %, are one such enhancement; others include completion <tab>, introspection ?, shell access !

```
# In CASA

%lsmagic
```

IPython magic commands

Notable magic commands:

- automagic -- % not required
- autocall -- () not required
- pprint -- pretty printing
- cpaste -- paste an entire cell (block of commands)
- history -- view command line history

```
# In CASA

history

?history
```

(by default, help uses 'less' as the pager. press 'q' to quit)

IPython system shell access

Arbitrary system shell commands can be executed with !

Output can be returned as a Python variable

If your desired system command is not found, check the path that CASA is using (e.g., %env PATH or os.environ['PATH'])

```
# In CASA

!du -hs *

hostname = !echo $HOSTNAME

print( hostname )
```

The IPython alias command

%alias is an IPython magic command that defines an alias to a system shell command

An alias is treated like a new magic command

Aliases have lower precedence than magic functions and normal Python variables

```
# In CASA

alias

alias du du -hs

du *
```

Further customization

CASA can be further customized by creating a file ***init.py*** in the $HOME/.casa directory

You can put commands here that will run every time CASA starts up, e.g., import modules, set variables, modify the PATH

You can also customize many aspects of IPython using ***init.py***, e.g., the color scheme, magic commands, aliases, behavior of extensions

CASA Tasking System

Tasking system

- inp
- go
- default
- tget
- tput
- tasklist
- taskhelp

Python functions that facilitate access to task parameters

Task parameters are variables in the global namespace

Together with %autocall, they give CASA a look and feel that is more user friendly and less 'Pythonic'

INTERACTIVE

CASA Tasking System

```
# In CASA

go?
```

example output (not interactive)

```
Signature: go(taskname=None)
Docstring: Execute taskname:
File:      /home/casa/packages/RHEL7/release/casa-
            release-5.4.2-5/lib/python2.7/init_tasks.py
Type:      function
```

CASA Tasks

CASA Command Line Interface
(Python, Ipython)

Task interface (XML)

- parameters
- description
- defaults
- expansions
- validation

Task script (Python)

- CASA toolkit
- logging
- history
- exceptions

INTERACTIVE

CASA task 'scripts'

Each CASA task executes a Python script

It can be informative to read some of these scripts

Scripts are named '**task_*taskname*.py**' and are located in a folder inside the CASA distribution

```
# In CASA

task_path = casa['dirs']['python']

print(task_path)

ls $task_path/task*py
```

CASA Tools

CASA Command Line Interface
(Python, Ipython)

CASA Tasks
(XML, Python, CASA tools)

Most tools are written in C++ and connected to Python using Simplified Wrapper and Interface Generator (SWIG)

Tool methods are more atomic than tasks; many tasks consist of calls to several tool methods

Tasks are meant to capture common use cases and be simple to use

Tools offer additional flexibility and functionality over tasks

Comparison: task vs. tool

imstat task

```
# In CASA

image = '3C75_initial.image.tt0'
imstat( image )
```

equivalent toolkit method (inside ***task_imstat.py***)

```
# In CASA

ia.open( image )
ia.statistics( robust=True )
ia.close()
```

CASA Misc

CASA Command Line Interface
(Python, Ipython)

CASA Tasks
(XML, Python, CASA tools)

CASA Tools
(C++, Python)

Data repo

- ephemerides
- geodetic
- ALMA calibration
- VLA calibration
- line catalogs
- support for simulations

Helper scripts

- recipes / utilities
- regression + other tests
- demonstrations
- parallelization

Outline

Advanced CASA overview

The CASA toolkit

CASA scripts and custom tasks

CASA Tools

af : Agent flagger	me : Measures
at : Atmospheric library	ms : MeasurementSet
ca : Calibration analysis	msmd : MS metadata
cb : Calibration	mt : MS transformer
cl : Component list	qa : Quanta
cp : Cal solution plotting	pm : PlotMS
cs : Coordinate system	po : Imagepol
cu : Class	rg : Region manipulation
dc : Deconvolver	sdms : Single-Dish MS
fi : Fitting	sl : Spectral line catalog
fn : Functional	sm : Simulation
ia : Image analysis	tb : Table
im : Imaging	tp : Table plotting
lm : Linear mosaic	vp : Voltage pattern

(also PySynthesisImager : tclean)

quanta tool

Values with units, unit conversion, string formatting

```
# In CASA

angle = qa.quantity( '1rad' )

print( angle )
```

example output (not interactive)

```
{'unit': 'rad', 'value': 1.0}
```

quanta tool

Values with units, unit conversion, string formatting

```
# In CASA

qa.convert( angle, 'arcsec' )

qa.time( angle, prec=10 )
```

example output (not interactive)

```
{'unit': 'arcsec', 'value': 206264.80624709636}

['03:49:10.9871']
```

measures tool

Reference frames, directions, conversions, measurements

```
# In CASA

dir1 = me.direction('J2000','06:18:00','+41.00.00')

from pprint import pprint as ppr

ppr( dir1 )
```

example output (not interactive)

```
{'m0': {'value': 1.6493361431346414, 'unit': 'rad'},
 'm1': {'value': 0.7155849933176751, 'unit': 'rad'},
 'refer': 'J2000',
 'type': 'direction'}
```

measures tool

Reference frames, directions, conversions, measurements

```
# In CASA

me.shift( dir1, '3arcmin', pa='45deg' )
```

```
# In CASA

me.doframe( me.observatory('VLA') )

me.doframe( me.epoch('utc','today') )

me.measure( dir1, 'azel' )
```

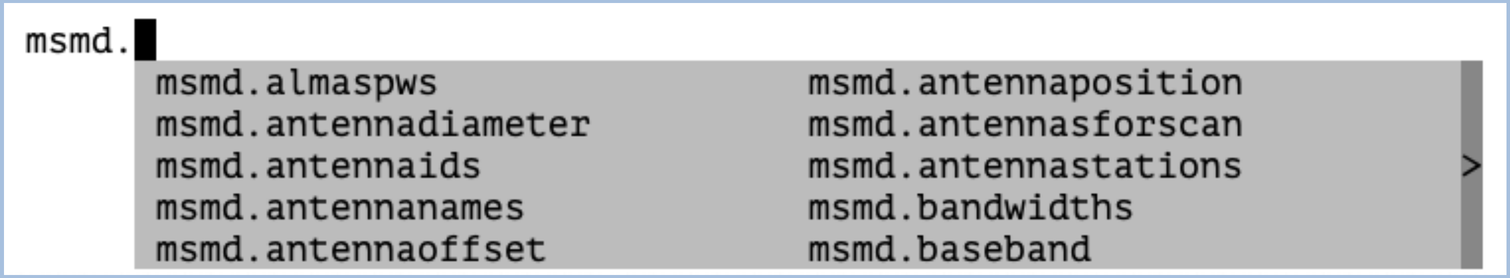
MS metadata tool

Helper functions to retrieve measurement set properties

```
# In CASA

vis = '3C75.ms'

msmd.open( vis )

msmd.<TAB>
```

(browse functions with **TAB** or ↑ ↓, select with **RETURN**)

example output (not interactive)



```
msmd.
      msmd.almaspws                 msmd.antennaposition
      msmd.antennadiameter          msmd.antennasforscan
      msmd.antennaids               msmd.antennastations       >
      msmd.antennanames             msmd.bandwidths
      msmd.antennaoffset            msmd.baseband
```

MS metadata tool

Helper functions to retrieve measurement set properties

```
# In CASA

fields = msmd.fieldnames()

msmd.close()

print( fields )
```

example output (not interactive)

```
['3C75']
```

INTERACTIVE

measurement set tool

Measurement set inspection and manipulation

```
# In CASA

ms.open( vis )

results = ms.getdata( ['time','uvw'] )

ms.close()
```

Note: this will try to retrieve data from all rows of the MS

Additional selection and / or iteration may be required for large data sets to avoid memory issues

INTERACTIVE

measurement set tool

Measurement set inspection and manipulation

```
# In CASA

type(results)

results.keys()

results['uvw'].shape
```

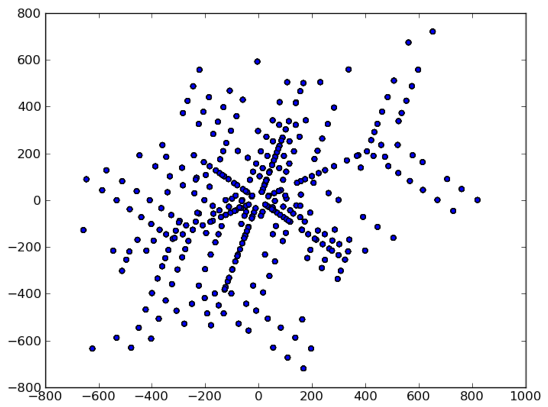
example output (not interactive)

```
dict

['uvw', 'time']

(3, 2808)
```

measurement set tool

Measurement set interaction and manipulation

```
# In CASA

u,v = results['uvw'][:2]

plt.scatter( u, v )

plt.scatter( -u, -v )
```



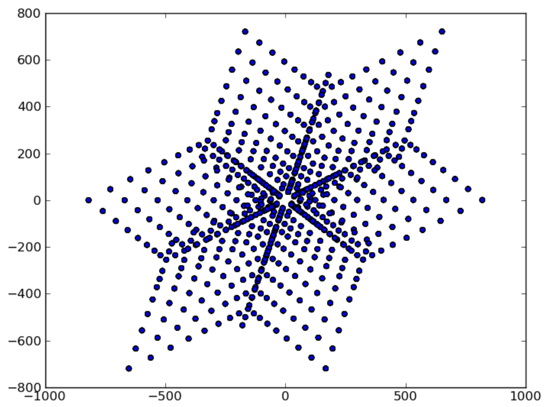


image analysis tool

Image inspection and manipulation

```
# In CASA

ia.open( image )

image_data = ia.getchunk()

ia.close()

image_data.shape

np.max( image_data )
```

table tool

Read, write and filter CASA tables. Works on measurement sets, ancillary tables of the MS, calibration tables, images, component lists, etc.

```
# In CASA

tb.open( vis+'/ANTENNA' )

tb.colnames()

tb.getcol( 'NAME' )

stb = tb.query( "NAME == 'ea12'" )

stb.getcol( 'POSITION' )

stb.done()

tb.close()
```

Outline

Advanced CASA overview

The CASA toolkit

CASA scripts and custom tasks

Getting started with CASA scripts

A CASA script is just a file that contains a sequence of commands, e.g., tasks, tools, general Python

Name your script almost anything you want, e.g. myScript.py

Run your script in CASA:

```
CASA>>  execfile( 'myScript.py' )
```

Run your script from the terminal:

```
bash$  casa -c myScript.py
```

Why write a script?

Reproducible - an executable record of your processing

Efficient - launch a sequence of several commands

Inspectable - easy to edit, expand, debug

Shareable - distribute to colleagues / helpdesk / etc.

Transportable - run on different resources

Generalizable - compatible with other similar datasets

Tips for CASA scripts

It is generally advised to run CASA tasks as functions

Use Python fundamentals where appropriate,
e.g., variables, loops, conditionals, exception handling

Learn from the examples in the documentation,
e.g., casadocs, casaguides, the toolkit reference manual

Ipython magic commands will not work, e.g., autocall

Use a persistent session when running remotely,
e.g., vnc, screen, nohup

Work with a Python-aware text editor
- highlighting, block indent, block comment

Example CASA scripts

Example Script: G55.7 CASAguides tutorial

```
setjy(vis='G55.7+3.4_10s.ms', field='0542*',
      spw='2~3,5~6', modimage='3C147_L.im')

gaincal(vis='G55.6+3.4_10s.ms',
        caltable='G55.6+3.4_10s.G0',spw='2~3,5~6',
        solint='int', calmode='p', field='0542*')
```

Example CASA scripts

Example Script: G55.7 CASAguides tutorial

```
vis = 'G55.7+3.4_10s.ms'
field = '0542*'
spw = '2~3,5~6'
modimage = '3C147_L.im'

setjy(vis=vis, field=field, spw=spw,
      modimage=modimage)

gaincal(vis=vis, caltable=vis.replace('.ms','.G'),
        field=field, spw=spw, solint='int',
        calmode='p')
```

Example CASA scripts

Example Script: *taskname.last*

```
# In CASA

inp setjy

tput

cat setjy.last
```

example output (not interactive)

```
...
#setjy(vis="TDRW0001 ... )
```

Example CASA scripts

For loops:

```
spws = ['2','3','4','5']

for spw in spws:
    tclean( spw=spw, imagename='image_spw'+spw ... )
```

Flow control:

```
do_polcal = True

if do_polcal:
    polcal( ... )
```

3rd Party tasks and tools

There are many CASA tasks and tools written by members of the user community that can be imported into CASA

These are available from various locations, e.g., github, observatory websites, personal websites

Many of these are listed on the following CASAguide page:

casaguides.nrao.edu/index.php/Contributed_CASA_Tasks_and_Scripts

Writing your own CASA tasks

You can turn your own code into a CASA task in 3 easy steps:

- Create your own ***task_taskname.py***
- Create a xml file ***taskname.xml***
- Run !***buildmytasks*** and **execfile('mytasks.py')**

More info here:

casaguides.nrao.edu/index.php/Writing_a_CASA_Task

Documentation

Python: docs.python.org/2/

iPython: ipython.readthedocs.io/en/5.x/

CASA Guides: casaguides.nrao.edu

CASA Docs: casa.nrao.edu/casadocs

CASA Toolkit: casa.nrao.edu/docs/CasaRef/CasaRef.html

Measurement Set: casa.nrao.edu/Memos/229.html

Table Query Language: casa.nrao.edu/aips2_docs/notes/199

Python modules and CASA

You can write Python functions and import them into CASA

```
# my_CASA_functions.py
from casa import gaincal
from casac import casac
tb=casac.table()

def function1 ...
```

Then import them into CASA or use them in your scripts

```
os.path.append( '/path/to/my_functions_dir' )
import my_CASA_functions
```

PySynthesisImager

Tool-like library that interfaces with C++ imaging routines.
See 'task_tclean.py' for a complete example.

```
from imagerhelpers.imager_base import PySynthesisImager

imager = PySynthesisImager( params )

imager.initializeImagers()
imager.initializeNormalizers()
imager.setWeighting()
imager.initializeDeconvolvers()
imager.initializeIterationControl()

imager.makePSF()
imager.makePB()
```

PySynthesisImager

Tool-like helper functions that interface with C++ imaging routines. See 'task_tclean.py' for a complete example.

```
imager.runMajorCycle()

while ( not imager.hasConverged() ):

     imager.runMinorCycle()
     imager.runMajorCycle()
     imager.updateMask()


imager.predictModel()
imager.restoreImages()
imager.pbcorImages()
```

www.nrao.edu
science.nrao.edu
public.nrao.edu