ALMA Data Reduction Tutorials
Synthesis Imaging Summer School
May 16, 2014
Outline

• Short introduction to CASA and the Python interface
  – Discussion of “tools” will be done separately
• The Flow of Calibration
• Key CASA tasks for data reduction/calibration
• Data Inspection and Flagging
• Basic Imaging
CASA (Common Astronomy Software Applications)

- CASA is the offline data reduction package for ALMA and the VLA (data from other telescopes usually work, too, but not primary goal of CASA)
- Code is C++ (fast) bound to Python (easy access and scripting) (plus some Qt or other apps)
- Import/export data, inspect, edit, calibrate, image, view, analyze
- Also supports single dish data reduction (based on ASAP)
- CASA has many tasks and a LOT of tool methods
- Easy to write scripts and tasks
- We have a lot of documentation, reduction tutorials, helpdesk, user forum
- CASA has some of the most sophisticated algorithms implemented (multi-scale clean, Taylor term expansion for wide bandwidths, W-term projection, OTF mosaicing, etc.)
- We have a active Algorithm Research Group, so expect more features in future versions…
CASA Startup

$ casapy (or simply “casa”)

bash-4.1$ casa

The start-up time of CASA may vary depending on whether the shared libraries are cached or not.

CASA Version 4.2.1 (r29948)
Compiled on: Tue 2014/04/01 19:49:27 UTC

For help use the following commands:
tasklist - Task list organized by category
taskhelp - One line summary of available tasks
help taskname - Full help for task
toolhelp - One line summary of available tools
help par.parame[ername] - Full help for parameter name

Activating auto-logging. Current session state plus future input saved.
Filename : ipython-20140512-125049.log
Mode : backup
Output logging : False
Raw input log : False
Timestamping : False
State : active

*** Loading ATNF ASAP Packages...
*** ... ASAP (4.2.0a rev#28887) import complete ***

CASA <2>: "My First Task"

5

5
CASA Interactive Interface

- CASA runs within python scripts or through the interactive IPython (ipython.org) interface
- IPython Features:
  - shell access
  - auto-parenthesis (autocall)
  - Tab auto-completion
  - command history (arrow up and “hist [-n]”)
  - session logging
    - ipython.log – ipython command history
    - casapyTIME.log – casa logger messages
  - numbered input/output
  - history/searching
Basic Python tips

• to run a python “.py” script:
  
  \[ \text{execfile}('\langle\text{scriptname}\rangle') \]

  \text{example: } \text{execfile('ngc5921\_demo.py')}  

Some python specialties:

• indentation matters!
  – indentation in python is for loops, conditions etc.
  – be careful when doing cut-and-paste to python
  – cut a few (4-6) lines at a time

• python counts from 0 to n-1!

• variables are global when using task interface

• tasknames are objects (not variables)
Tasks and tools in CASA

• **Tasks** - high-level functionality
  – function call or parameter handling interface
  – these are what you should use in tutorials
• **Tools** - complete functionality
  – `tool.method()` calls, they are internally used by tasks or can be used on their own
  – sometimes shown in tutorial scripts and CASAGuides
• **Applications** – some tasks/tools invoke standalone apps
  – e.g. `casaviewer, casaplotms, casabrowser, asdm2MS`
• **Shell commands** can be run with a leading exclamation mark
  `!du -ls` or inside `os.system("shell command")`
  (some key shell commands like “ls” work without the exclamation mark and we will use `os.system()` exclusively within this tutorial.)
Find the right Task

To see list of tasks organized by type:

tasklist
Find the right Task

To see list of tasks with short help:

taskhelp
Task Interface

examine task parameters with inp:

```
CASA <12>: inp
----------> inp()
# clean : Invert and deconvolve images with selected algorithm
vis = '' # Name of input visibility file
imname = '' # Pre-name of output images
outfile = '' # Text file with image names, sizes, centers for outliers
field = '' # Field Name or id
spw = '' # Spectral windows e.g. '0-3', '' is all
selectdata = False # Other data selection parameters
mode = 'channel' # Spectral gridding type (mfs, channel, velocity, frequency)
nchan = -1 # Number of channels (planes) in output image; -1 = all
start = 0 # Begin the output cube at the frequency of this channel in the MS
width = 1 # Width of output channel relative to MS channel (# to average)
interpolation = 'linear' # Spectral interpolation (nearest, linear, cubic). Use nearest for
chaniter = False # Clean each channel to completion (True), or all channels each cycle (False)
outframe = '' # velocity frame of output image

gridmode = '' # Gridding kernel for FFT-based transforms, default="None"
iter = 500 # Maximum number of iterations
gain = 0.1 # Loop gain for cleaning
threshold = '0.0mJy' # Flux level to stop cleaning, must include units: '1.0mJy'
psfmode = 'clark' # Method of PSF calculation to use during minor cycles
imagemode = '' # Options: 'csclean' or 'mosaic', '', uses psfmode
multiscale = [] # Deconvolution scales (pixels); [] = standard clean
interactive = False # Use interactive clean (with GUI viewer)
mask = [] # Cleanbox(es), mask image(s), region(s), or a level
imsiz = [256, 256] # x and y image size in pixels. Single value: same for both

cell = ['1.0arcsec'] # x and y cell size(s). Default unit arcsec.
phasedcenter = '' # Image center: direction or field index
restfreq = '' # Rest frequency to assign to image (see help)
stokes = 'I' # Stokes params to image (eg I,V,IQ,UV)
weighting = 'natural' # Weighting of uv (natural, uniform, briggs, ...)
modelimage = '' # Name of model image(s) to initialize cleaning
restoringbeam = [''] # Output Gaussian restoring beam for CLEAN image
pbcor = False # Output primary beam-corrected image
minpb = 0.2 # Minimum PB level to use
calready = True # True required for self-calibration
async = False # If true the taskname must be started using clean(...)
```

CASA <13>:
Task Interface

- standard tasking interface, similar to AIPS, MIRIAD, etc.
- parameter manipulation commands
  - inp, default, saveinputs, tget, tput
- use parameters set as global Python variables
  \[
  \text{<param>} = \text{<value>}
  \]
  (e.g. \text{vis} = \text{\texttt{ngc5921.demo.ms}})
- execute
  \[
  \text{<taskname>} \ \text{or go} \ (\text{e.g. clean()})
  \]
- return values (except when using “go”)
  - some tasks return Python dictionaries, assign a variable name to get them, e.g. \text{myval=imval()}
  - Very useful for scripting based on task outputs
Expandable Parameters

- Boldface parameters have *subparameters* that unfold when main parameter is set.
Expandable Parameters

- Boldface parameters have subparameters that unfold when main parameter is set.
Expandable Parameters

- Boldface parameters have subparameters that unfold when main parameter is set
Parameter Checking

sanity checks of parameters in inp:

CASA <17>: inp
---------> inp()

# clean :: Invert and deconvolve images with selected algorithm
vis = 'm51-center-contall.ms' # Name of input file
imagename = 'M51-cont-rob-1as-noninteractive-multipole' # Pre-name of output images
outlierfile = '' # Text file with input and output outlier parameters for outliers
field = '' # Field Name or id
spw = '' # Spectral windows e.g. '0~3', '' is all

selectdata = False # True for data selection parameters
mode = 'Monty Python' # Spectral gridding type (mfs, channel, velocity, frequency)
gridmode = '' # Gridding kernel for FFT-based transforms, default='' None
niter = 1000 # Maximum number of iterations
gain = 0.2 # Loop gain for cleaning
threshold = '12uJy' # Flux level to stop cleaning, must include units: '1.0mJy'
psfmode = 'clark' # Method of PSF calculation to use during minor cycles

imagermode = 'csclean' # Options: 'psf' or 'mosaic', '', uses psfmode

imagermode = 'csclean'
cyclefactor = 1.5 # Controls how often major cycles are done. (e.g. 5 for frequently)
cyclespeedup = -1 # Cycle threshold doubles in this number of iterations

multiscale = [0, 2, 5, 8, 15, 50, 100] # Deconvolution scales (pixels); [] = standard clean
negcomponent = -1 # Stop cleaning if the largest scale finds this number of neg components
smallscalebias = 0.6 # a bias to give more weight toward smaller scales

interactive = False # Use interactive clean (with GUI viewer)
mask = [] # Cleanbox(es), mask image(s), region(s), or a level
imsizex = 1280 # x and y image size in pixels. Single value: same for both
imsizey = 1280 # x and y cell size(s). Default unit arcsec.
cell = '1arcsec' # Image center: direction or field index
phasecenter = 'J2000 13h29m52.2s +47d12m30s' # Image center: direction or field index
restfreq = '' # Rest frequency to assign to image (see help)
Help on Tasks

In-line help:

help clean   (Or pdoc clean)

Help on clean task:

Invert and deconvolve images with selected algorithm
The clean task has many options:

1) Make 'dirty' image and 'dirty' beam (psf)
2) Multi-frequency-continuum images or spectral channel imaging
3) Full Stokes imaging
4) Mosaicking of several pointings
5) Multi-scale cleaning
6) Widefield cleaning
7) Interactive clean boxing
8) Use starting model (eg from single dish)

vis -- Name(s) of input visibility file(s)
    default: none;
    example: vis='ngc5921,ms'
    vis=['ngc5921a,ms','ngc5921b,ms']; multiple MSes

imagename -- Pre-name of output images:
    default: none; example: imagnname='m2'
output images are:
    m2.image; cleaned and restored image
    With or without primary beam correction
    m2.psf; point-spread function (dirty beam)
    m2.flux; relative sky sensitivity over field
    m2.flux.pbcoverage; relative pb coverage over field
    (gets created only for ft='mosaic')
    m2.model; image of clean components
    m2.residual; image of residuals
Task Execution

• In addition to typing in all variables in the task interface and executing with `go` one can write the full parameter set in a line:

```latex
taskname( arg1=val1, arg2=val2, ... )
```

e.g.

```latex
clean(vis='input.ms', imagename='galaxy', selectvis=T, robust=0.5, imsize=[200,200])
```

– unspecified parameters will be set to their default values (globals not used; i.e. not to previously set variables)

– Useful in scripts, but also in ‘pseudo-scripts’:

  • To keep a record it is frequently a good idea to write down the full line as above in an editor, then cut and paste into CASA.
  
  • When changes are needed, change in editor and cut and paste again. That is good practice to keep a record of the exact input.

  • But note that the logger is also repeating the full task command
**Measurement Set**

- CASA stores u-v data in directories called “Measurement Sets”
  TO DELETE THEM USE `rmtables("my_table.gcal")`

- These data sets store two copies of the data (called “columns”):

<table>
<thead>
<tr>
<th>“Data” Column</th>
<th>“Corrected” Column</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contains the raw, unprocessed measurements.</td>
<td>Usually created by applying one or more calibration terms to the data.</td>
</tr>
</tbody>
</table>

- Additionally a “model” may be stored separately.
  THIS IS USED TO CALCULATE WHAT THE TELESCOPE SHOULD HAVE OBSERVED.

- Each data point may also be “flagged,” i.e., marked bad.
  IN THIS CASE IT IS IGNORED (TREATED AS MISSING) BY CASA OPERATIONS.
Calibration Tables

- Calibration yields estimates of phase and amplitude corrections. E.G., AS A FUNCTION OF TELESCOPE, TIME, FREQUENCY, POLARIZATION.

- CASA stores these corrections in directories called “calibration tables.” TO DELETE THEM USE rmtables(“my_table.gcal”)

- These are created by calibration tasks: E.G., gaincal, bandpass, gencal

- Applied via “applycal” to the data column and saved as corrected.
Basic Calibration Flow

Define what the telescope SHOULD have seen.

- Measurement Set
  - Model (defaults to point source)

Define a model for the data (setjy)

- Measurement Set (with associated model)
Basic Calibration Flow

Derive the corrections needed to make the data match the model.

- Measurement Set (with associated model)
  - Calibration Task (e.g., gaincal, bandpass)
  - Calibration Table
Apply these corrections to derive the corrected (calibrated) data.

- Measurement Set
- Data Column
- Calibration Table

Apply Calibration
applycal

- Measurement Set
- Corrected column now holds calibrated data.
Basic Calibration Flow

Define what the telescope SHOULD have seen.

- Measurement Set
- Model (defaults to point source)
- Define a model for the data (setjy)
- Measurement Set (with associated model)

Derive the corrections needed to make the data match the model.

- Measurement Set (with associated model)
- Calibration Task (e.g., gaincal, bandpass)
- Calibration Table

Apply these corrections to derive the corrected (calibrated) data.

- Measurement Set
- Data Column
- Calibration Table
- Apply Calibration applycal
- Measurement Set
  Corrected column now holds calibrated data.
Outline

- Short introduction to CASA and the Python interface
  - Discussion of “tools” will be done separately
- The Flow of Calibration
- Key CASA tasks for data reduction/calibration
- Data Inspection and Flagging
- Basic Imaging
ALMA Online Calibration

- **System Temperature (Tsys)** – atmospheric emission/opacity
  - Key to gain transfer across elevation
  - Amplitude calibration, variable with frequency (observed in “TDM”)
  - System temperatures of order ~100 K at Band 3 to ~1000 K at Band 9

- **Water Vapor Radiometer (WVR)** – phase delay due to atmosphere
  - Key to correct short-timescale phase variations
  - Phase calibration, variable with time

These are provided by the observatory (eventually applied online).
  - Apply them as first step (or start with provided pre-applied versions)
  - In either case, inspect these tables to learn about data quality
  - *The datasets associated with this tutorial already have these corrections applied*
  - Make sure reference antenna is “well-behaved”
Possible Flagging and Calibration Recipe

- (Apply online calibrations for water vapor and Tsys)
- EXAMINE bandpass/flux calibrator(s)
- FLAG bandpass/flux calibrators
- APPLY bandpass/flux calibration to itself
- APPLY bandpass/flux cal to gain cal sources
- EXAMINE gain calibration sources
- FLAG gain calibration sources
- APPLY gain calibration to itself
- APPLY bandpass/flux/gain cal to targets
- EXAMINE targets
- FLAG targets

Iterate

Repeat as necessary
ALMA Online Calibration

uid__A002_X56a957_X63a ms tsys
DA42 DA44 DA45 DA46 DA49 DA50 DA51 DA53 DV02 DV04 DV05 DV07 DV08 DV10 DV13 DV16 DV17 DV18 DV19 DV20 DV22 DV23 PM01 PM02
spw9, field 0: J0522-364, t1/6 05:00:12

XX solid
YY dashed

uid__A002_X56a957_X63a ms tsys
spw11, field 0: J0522-364, t1/6 05:00:12

XX solid
YY dashed

uid__A002_X56a957_X63a ms tsys
spw13, field 0: J0522-364, t1/6 05:00:12

XX solid
YY dashed

uid__A002_X56a957_X63a ms tsys
spw15, field 0: J0522-364, t1/6 05:00:12

XX solid
YY dashed

uid__A002_X56a957_X63a.ms ObsDate=2012-12-02 plotbandpass3 v1.23 = 2013/02/14 18:24:44
ALMA Online Calibration

uid__A002_X56a957_X63a ms tsys
DA42  DA43  DA46  DA49  DA50  DA51  DA53  DV01  DV04  DV05  DV07  DV08  DV10  DV13  DV16  DV17  DV18  DV19  DV20  DV22  DV23
spw9, field 0: J0522-364, t1/6 05:00:12

uid__A002_X56ec89_X25c ms tsys
DA42  DA43  DA44  DA45  DA46  DA48  DA49  DA50  DV03  DV04  DV05  DV06  DV07  DV08  DV10  DV13  DV16  DV17  DV19  DV20  DV22  DV23
spw13, field 0: J0522-364, t1/6 05:00:12

uid__A002_X56ec89_X25c ms tsys
DA42  DA43  DA44  DA45  DA46  DA48  DA49  DA50  DV03  DV04  DV05  DV06  DV07  DV08  DV10  DV13  DV16  DV17  DV19  DV20  DV22  DV23
spw13, field 0: J0522-364, t1/6 05:00:12

uid__A002_X56ec89_X25c ms tsys
DA42  DA43  DA44  DA45  DA46  DA48  DA49  DA50  DV03  DV04  DV05  DV06  DV07  DV08  DV10  DV13  DV16  DV17  DV19  DV20  DV22  DV23
spw13, field 0: J0522-364, t1/6 05:00:12

uid__A002_X56ec89_X25c.ms ObsDate=2012-12-30 plotbandpass3 v1.20 = 2013/01/07 21:23:53
ALMA Online Calibration

Phase vs. Time
One 600m Baseline
~600 GHz
Before WVR, After WVR
Key Tasks for Calibration

Derive Calibration Tables
- **setjy**: set “model” (correct) visibilities using known model for a calibrator
- **bandpass**: calculate bandpass calibration table (amp/phase vs frequency)
- **gaincal**: calculate temporal gain calibration table (amp/phase vs time)
- **fluxscale**: apply absolute flux scaling to calibration table from known source

Manipulate Your Measurement Set
- **flagdata/flagcmd/flagmanager**: flag (remove) bad data
- **applycal**: apply calibration table(s) from previous steps
- **split**: split off calibrated data from your ms

Inspect Your Data and Results
- **plotms**: inspect your data interactively
- **plotcal**: examine a calibration table
Schematic Calibration

Calibrate the Amplitude and Phase vs. Frequency of Each Antenna  
Assume time & frequency response separable, remove time variability

Calibrate the Amplitude and Phase vs. Time of Each Antenna  
Assume time & frequency response separable, remove freq. variability

Set the Absolute Amplitude Scale With Reference to a Known Source  
Planet (modeled), monitored quasar, etc.

Apply all corrections to produce calibrated data
Schematic Calibration

1. Calibrate the Amplitude and Phase vs. Frequency of Each Antenna bandpass
   - Bandpass Calibration Table

2. Calibrate the Amplitude and Phase vs. Time of Each Antenna gaincal
   - Phase Calibration Table
   - Amplitude Calibration Table

3. Set the Absolute Amplitude Scale With Reference to a Known Source fluxscale
   - Flux Calibration Table

4. Apply all corrections to produce calibrated data applycal
   - Measurement Set
   - Corrected column now holds calibrated data.
“My first task” and Orientation to the Data

ALMA Data Reduction Tutorials
Synthesis Imaging Summer School
May 16, 2014
My first task…

Let’s get started with the data reduction…

Under the sis14 directory you will find a directory called /lessons. Inside /lessons there is a README file that outlines the directory structure for the entire tutorial.

We are going to work directly under the /lessons directory step by step so if a dataset gets corrupted, you can easily copy over a new copy from the /working_data directory. That is why every script starts with a:

```
   os.system("rm -rf sis14_twhya_uncalibrated.ms")
```

To begin, if you haven’t already done so…start casa:
```
casapy
```

Copy the data over from the working directory:

```
   os.system("cp -r ../working_data/sis14_twhya_uncalibrated.ms .")
```
My first task

View a list of available tasks

**tasklist**

Available tasks, organized by category (experimental tasks in parenthesis ()
deprecated tasks in curly brackets {}).

<table>
<thead>
<tr>
<th>Import/export</th>
<th>Information</th>
<th>Editing</th>
<th>Manipulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>exportasdm</td>
<td>imhead</td>
<td>fixplanets</td>
<td>concat</td>
</tr>
<tr>
<td>exportfits</td>
<td>imrefrme</td>
<td>fixvis</td>
<td>conjugatevis</td>
</tr>
<tr>
<td>exportuvtfs</td>
<td>imstat</td>
<td>flagcmd</td>
<td>cvel</td>
</tr>
<tr>
<td>importasdm</td>
<td>imval</td>
<td>flagdata</td>
<td>fixvis</td>
</tr>
<tr>
<td>importfits</td>
<td>listcal</td>
<td>flagmanager</td>
<td>hanningsmooth</td>
</tr>
<tr>
<td>importfitsidi</td>
<td>listfits</td>
<td>mview</td>
<td>imhead</td>
</tr>
<tr>
<td>importuvtfs</td>
<td>listhistory</td>
<td>plotms</td>
<td>msmoments</td>
</tr>
<tr>
<td>importvla</td>
<td>listobs</td>
<td>plotxy</td>
<td>partition</td>
</tr>
<tr>
<td>(importvla)</td>
<td>listpartition</td>
<td>plotms</td>
<td></td>
</tr>
<tr>
<td>(importgmrt)</td>
<td>listvis</td>
<td>plotxy</td>
<td></td>
</tr>
<tr>
<td></td>
<td>plotms</td>
<td></td>
<td>split</td>
</tr>
<tr>
<td></td>
<td>plotuv</td>
<td></td>
<td>testconcat</td>
</tr>
<tr>
<td></td>
<td>plotxy</td>
<td></td>
<td>uvcontsub</td>
</tr>
<tr>
<td></td>
<td>vishead</td>
<td></td>
<td>virtualconcat</td>
</tr>
<tr>
<td></td>
<td>visstat</td>
<td></td>
<td>vishead</td>
</tr>
<tr>
<td></td>
<td>(asdmsummary)</td>
<td></td>
<td>(mstransform)</td>
</tr>
<tr>
<td></td>
<td>.</td>
<td></td>
<td>.</td>
</tr>
<tr>
<td></td>
<td>.</td>
<td></td>
<td>.</td>
</tr>
</tbody>
</table>
My First Task

View available inputs to the listobs task

```
inp listobs

# listobs :: List the summary of a data set in the logger or in a file
vis = "" # Name of input visibility file (MS)
selectdata = True # Data selection parameters
field = "" # Field names or field index numbers: "+=all, field='0~2,3C286'
spw = "" # spectral-window/frequency/channel
verbose = True
listfile = "" # Name of disk file to write output: "+=to terminal
listunfl = False # List unflagged row counts? If true, it can have significant negative performance impact.
cachesize = 50 # EXPERIMENTAL. Maximum size in megabytes of cache in which data structures can be held.
async = False # If true the taskname must be started using listobs(…)
```

Set the visibility and review the modified inputs

```
vis = 'sis14_twhya_uncalibrated.ms'
inp listobs

# listobs :: List the summary of a data set in the logger or in a file
vis = 'sis14_twhya_uncalibrated.ms' # Name of input visibility file (MS)
selectdata = True # Data selection parameters
field = "" # Field names or field index numbers: "+=all, field='0~2,3C286'
spw = "" # spectral-window/frequency/channel
antenna = "" # antenna/baselines: "+=all, antenna='3,VA04'
timerange = "" # time range: "+=all,timerange='09:14:0~09:54:0'
My First Task

Run the listobs task
My First Task

Instead, send output to a file

```python
listobs(vis="sis14_twhya_uncalibrated.ms", listfile="my_listfile.txt")
```

System commands can be run from within casapy

```python
os.system(“ls”)
```

```bash
casapy-20140506-152315.log first_script.py ipython-20140506-152318.log listobs.last
my_first_task.py sis14_twhya_uncalibrated.ms
```

```python
Out[12]: 0
```

```python
os.system("more my_listfile.txt")
```

In this case, we will use os.system() instead of "!" because it is needed when scripting...
first_script.py

# This is a comment

# A python command
print "Hello CASA!"

# A call to a system command
os.system("rm -rf my_script_listfile.txt")

# A CASA command
listobs(vis="sis14_twhya_uncalibrated.ms", listfile="my_script_listfile.txt")

Run it with execfile

    execfile("first_script.py")

Hello CASA!
Writing output to file: my_script_listfile.txt
Getting Oriented

First, cd into the /orient directory and copy the data over for inspection:

```bash
os.system("cp -r ../../working_data/sis14_twhya_uncalibrated.ms .")
```

Run the listobs task

```bash
listobs("sis14_twhya_uncalibrated.ms")
```
Getting Oriented

Run the plotants task

plotants("sis14_twhya_uncalibrated.ms", figfile="plotants.png")
inp plotms

# plotms :: A plotter/interactive flagger for visibility data.
vis = ""    # input MS (or CalTable) (blank for none)
xaxis = ""  # plot x-axis (blank for default/current)
yaxis = ""  # plot y-axis (blank for default/current)
selectdata = True  # data selection parameters
  field = ""  # field names or field index numbers (blank for all)
  spw = ""  # spectral windows:channels (blank for all)
  timerange = ""  # time range (blank for all)
  uvrange = ""  # uv range (blank for all)
  antenna = ""  # antenna/baselines (blank for all)
  scan = ""  # scan numbers (blank for all)
  correlation = ""  # correlations (blank for all)
  array = ""  # (sub)array numbers (blank for all)
  observation = ""  # Select by observation ID(s)
  msselect = ""  # MS selection (blank for all)

averagedata = True  # data averaging parameters
  avgchannel = ""  # average over channel? (blank = False, otherwise value in channels)
  avgtime = ""  # average over time? (blank = False, other value in seconds)
  avgscan = False  # only valid if time averaging is turned on. average over scans?
  avgfield = False  # only valid if time averaging is turned on. average over fields?
  avgbaseline = False  # average over all baselines? (mutually exclusive with avgantenna)
  avgantenna = False  # average by per-antenna? (mutually exclusive with avgbaseline)
  avgspw = False  # average over all spectral windows?
  scalar = False  # Do scalar averaging?

transform = False  # transform data in various ways?
extendflag = False  # have flagging extend to other data points?
iteraxis = ""  # the axis over which to iterate
customsymbol = True  # set a custom symbol for unflagged points
  symbolshape = 'autoscaling'  # shape of plotted unflagged symbols
  symbolsize = 2  # size of plotted unflagged symbols
Getting Oriented

plotms(vis="sisl4_twhya_uncalibrated.ms", xaxis="time", yaxis="amp", averagedata=T, avgchannel="1e3", coloraxis="field")
plotms(vis="sis14_twhya_uncalibrated.ms", xaxis="time", yaxis="amp", averagedata=T, avgchannel="1e3", coloraxis="field")
Bandpass, Phase and Amplitude Calibration

ALMA Data Reduction Tutorials
Synthesis Imaging Summer School
May 16, 2014
What is Bandpass Calibration?

As we have seen all week, the goal of calibration is to find the relationship between the observed visibilities, $V_{\text{obs}}$, and the true visibilities, $V$:

$$V_{ij}(t,\nu)_{\text{obs}} = V_{ij}(t,\nu)G_{ij}(t)B_{ij}(t,\nu)$$

where $t$ is time, $\nu$ is frequency, $i$ and $j$ refer to a pair of antennas $(i,j)$ (i.e., one baseline), $G$ is the complex "continuum" gain, and $B$ is the complex frequency-dependent gain (the "bandpass").

**Bandpass calibration** is the process of measuring and correcting the frequency-dependent part of the gains, $B_{ij}(t,\nu)$.

$B_{ij}$ may be constant over the length of an observation, or it may have a slow time dependence.
Why is BP Calibration important?

Good bandpass calibration is a key to detection and accurate measurement of spectral features, especially weak, broad features.

Bandpass calibration can also be the limiting factor in dynamic range of continuum observations.

• Bandpass amplitude errors may mimic changes in line structure with $\nu$

• $\nu$-dependent phase errors may lead to spurious positional offsets of spectral features as a function of frequency, mimicking doppler motions

• $\nu$-dependent amplitude errors limit ability to detect/measure weak line emission superposed on a continuum source. Consider trying to measure a weak line on a strong continuum with $\sim$ 10% gain variation across the band.
Bandpass Calibration

• Determine the variations of phase and amplitude with frequency

• Account for slow time-dependency of the bandpass response

• We will arrive at antenna-based solutions against a reference antenna
  – In principle, could use autocorrelation data to measure antenna-based amplitude variations, but not phase
  – Most bandpass corruption is antenna-based, yet we are measuring $N(N-1)/2$ baseline-based solutions
  – Amounts to channel-by-channel self-cal
Bandpass Calibration:
What makes good calibrators?

• Best targets are bright, flat-spectrum sources with featureless spectra
  – Although point-source not absolutely required, beware frequency dependence of resolved sources
  – If necessary, can specify a spectral index using setjy

• Don’t necessarily need to be near science target on the sky
CASA Tasks for Bandpass Calibration

• We will use *gaincal* to measure time variation of phase

• Then use *bandpass* task
  – We will calibrate channel-to-channel variation (preferred method)
  – Alternatively, could fit a smooth function
  – Pay close attention to solutions; e.g. bright calibrators are rare, esp. at Band 9

• Use *applycal* to apply the bandpass solution to other sources
Orient yourself with a listobs

Run a listobs and note the bandpass calibrators. We have two, but will work with field 0 in this data set.

- listobs("sis14_twhya_uncalibrated.ms")

Gaincal is the general purpose task to solve for time-dependent amplitude and phase variations for each antenna. Here we carry out a short-timescale phase solution ("int") on the bandpass calibrator. This is saved as a calibration table "phase_int.cal".

- os.system("rm -rf phase_int.cal")
- gaincal(vis="sis14_twhya_uncalibrated.ms",
  caltable="phase_int.cal",
  field="0",
  solint="int",
  calmode="p",
  refant="DV22",
  gaintype="G"
Plot phase vs. time

Now we plot the calibration table, showing phase vs. time with a separate plot for each antenna. The two colors are the two correlations (i.e., polarizations).

- `plotcal(caltable="phase_int.cal", xaxis="time", yaxis="phase", subplot=331, iteration="antenna", plotrange=[0,0,-180,180])`
First bandpass solution

Now carry out a bandpass solution. This will solve for the amplitude and phase corrections needed for each channel for antenna. We use gaintable to feed the short-timescale phase solution to the task. This means that this table will be applied before the bandpass solution is carried out. We will deal with the overall normalization of the data later, for now we tell the task to solve for normalized (average=1) solutions via solnорм=True.

- os.system("rm -rf bandpass.cal")
- bandpass(vis="sis14_twhya_uncalibrated.ms",
  caltable="bandpass.cal",
  field="0",
  refant="DV22",
  solint="inf",
  combine="scan",
  solnорм=True,
  gaintable=["phase_int.cal"]
)
Plotbandpass

We inspect the phase and amplitude behavior of the calibration plotting the corrections for each antenna using plotbandpass. We tell it to plot both phase and amplitude for three antennas at a time. Cycle through the plots.

- plotbandpass(caltable="bandpass.cal", xaxis=chan", yaxis="both", subplot=32)
Calibrate the bandpass

Notice how noisy the solutions are. We can also calibrate the bandpass but average several channels at once, which is good if you think that signal-to-noise may be an issue and the solutions can be described as smoothly varying functions. We do this by setting a solution interval of 10 channels.

- os.system("rm -rf bandpass_10chan.cal")
- bandpass(vis="sis14_twhya_uncalibrated.ms",
  caltable="bandpass_10chan.cal",
  field="0",
  refant="DV22",
  solint="inf,10chan",
  combine="scan",
  solnorm=True,
  gaintable=["phase_int.cal"]
)
Plot solutions

Now plot these. There are less points and they are less noisy in absolute scale. Both tables seemed fine, but we will use these.

- `plotbandpass(caltable="bandpass_10chan.cal", xaxis="chan", yaxis="both", subplot=32)`
Apply the solutions

Apply the solutions - both in time and frequency - to the data using applycal. This creates a new corrected data column. Note that we will only apply these to field 0 at first and then look at the effects.

```python
applycal(vis="sis14_twhya_uncalibrated.ms",
         field="0",
         gaintable=
          ["bandpass_10chan.cal", "phase_int.cal"],
         interp=
          ["linear","linear"],
         Gainfield=
          ["0","0"]
         )
```

Plot the results of the calibration by comparing the dependence of phase and amplitude on channel before and after calibration.

**At this point, we are going to look at how the solutions have fixed the phase and amplitude variations vs. frequency. You can try the non-channel averaged data to see if there are any differences.**
Phase vs Channel before

```python
plotms(vis="sis14_twhya_uncalibrated.ms", xaxis="chan", yaxis="phase", ydatacolumn="data", field="0", averagedata=T, avgtime="1e3", coloraxis="corr")
```
Phase vs Channel after

```python
plotms(vis="sis14_twhya_uncalibrated.ms", xaxis="chan", yaxis="phase", ydatacolumn="corrected", field="0", averagedata=T, avgtime="1e3", coloraxis="corr")
```
Amp vs. Chan before

- plotms(vis="sis14_twhya_uncalibrated.ms", xaxis="chan", yaxis="amp", ydatacolumn="data", field=0, averagedata=T, avgtime="1e3", coloraxis="corr")
Amp vs. Chan after

- `plotms(vis="sis14_twhya_uncalibrated.ms", xaxis="chan", yaxis="amp", ydatacolumn="corrected", field="0", averagedata=T, avgtime="1e3", coloraxis="corr")`
Apply bandpass solutions to the whole data set

Note a couple things.

1. This will overwrite the previous corrected data for the bandpass calibrator.

2. Without the time-dependent gain factor applied the plotms plots above will not necessarily work as well (they do okay, but that's just lucky and good quality data).

We now thing that we have removed frequency dependent effects from the whole data set and will proceed with a time-dependent calibration.

Note that we use the non-standard "calonly" command, which tells applycal not to flag data for which the calibration has failed.

```
applycal(vis="sis14_twhya_uncalibrated.ms",
    field="",
    gaintable=['bandpass_10chan.cal'],
    interp=['linear'],
    gainfield=['0'],
    applymode='calonly')
```
Split out calibrated data

Now that we are satisfied with the bandpass calibration, we split out the bandpass calibrated data for further processing.

```python
• os.system("rm -rf sis14_twhya_bpcal.ms")
• split(vis="sis14_twhya_uncalibrated.ms",
    datacolumn="corrected",
    outputvis="sis14_twhya_bpcal.ms",
    keepflags=False)
```

This produces one of the supplied data products (sis14_twhya_bpcal.ms) - so you can restart from a successful version of this script anytime
Reminder: Basic Calibration Flow

Define what the telescope SHOULD have seen.

1. Define a model for the data (setjy)
2. Calibration Task (e.g., gaincal, bandpass)
3. Apply Calibration applycal

Derive the corrections needed to make the data match the model.

Apply these corrections to derive the corrected (calibrated) data.
Schematic Calibration

Calibrate the Amplitude and Phase vs. Frequency of Each Antenna
ASSUME TIME & FREQUENCY RESPONSE SEPARABLE, REMOVE TIME VARIABILITY

Calibrate the Amplitude and Phase vs. Time of Each Antenna
ASSUME TIME & FREQUENCY RESPONSE SEPARABLE, REMOVE FREQ. VARIABILITY

Set the Absolute Amplitude Scale With Reference to a Known Source
PLANET (MODELLED), MONITORED QUASAR, ETC.

Apply all corrections to produce calibrated data
Schematic Calibration

1. Calibrate the Amplitude and Phase vs. Frequency of Each Antenna bandpass
   - Bandpass Calibration Table

2. Calibrate the Amplitude and Phase vs. Time of Each Antenna gaincal
   - Phase Calibration Table
   - Amplitude Calibration Table

3. Set the Absolute Amplitude Scale With Reference to a Known Source fluxscale
   - Flux Calibration Table

4. Apply all corrections to produce calibrated data applycal
   - Measurement Set
     - Corrected column now holds calibrated data.
Key Tasks for Calibration

Derive Calibration Tables
- setjy: set “model” (correct) visibilities using known model for a calibrator
- bandpass: calculate bandpass calibration table (amp/phase vs frequency)
- gaincal: calculate temporal gain calibration table (amp/phase vs time)
- fluxscale: apply absolute flux scaling to calibration table from known source

Manipulate Your Measurement Set
- flagdata/flagcmd/flagmanager: flag (remove) bad data
- applycal: apply calibration table(s) from previous steps
- split: split off calibrated data from your ms (for imaging!)

Inspect Your Data and Results
- plotms: inspect your data interactively
- plotcal: examine a calibration table
Calibration Table
Later applied with applycal

Measurement Set
Data column holds observations.

gaincal
Solve for phase and amplitude response of each telescope as a function of time.
(Solutions derived to give best match of data to model once they are applied.)

Calibration Table
Later applied with applycal
gaincal

Measurement Set
Data column holds observations.

(Optional)
Associate a model (expected sky distribution) with the MS. (Else assume point source)

(Optional)
One or More Calibration Tables (applied on the fly before solution)

gaincal
Solve for phase and amplitude response of each telescope as a function of time. (Solutions derived to give best match of data to model once they are applied.)

Calibration Table
Later applied with applycal
gaincal

Measurment Set
Data column holds observations.

(Optional)
One or More Calibration Tables
(applied on the fly before solution)

gaincal
Solve for phase and amplitude response of each telescope as a function of time.
(Solutions derived to give best match of data to model once they are applied.)

(Optional)
Associate a model (expected sky distribution) with the MS.
(Else assume point source)

- What time interval to solve over?
- Requirements for a good solution.
  - Reference Antenna

Calibration Table
Later applied with applycal
Set Model for the Planet

First things first - we need to make sure that we have valid models in place for our data. We will work out the fluxes of the quasars later, for now it's good enough that we expect them to be point sources. However, the flux calibrator Ceres is somewhat resolved and we don't know the flux a priori. We need to read in a model from the solar system models that ship with CASA. We will use the task "setjy" and the library "Butler-JPL-Horizons 2012". With this call, we fill in the model column for Ceres.

```python
setjy(vis="sis14_twhya_bpcal.ms",
     field="2",
     standard="Butler-JPL-Horizons 2012",
     usescratch=True)
```
Phase Calibration

First, we calibrate the phase for each antenna for each scan. This is the right cadence to transfer to the science target, which is visited only on a ~ every-other-scan timescale.

- os.system("rm -rf phase_scan.cal")
- gaincal(vis="sis14_twhya_bpcal.ms",
  caltable="phase_scan.cal",
  field="0,2,3",
  solint="inf",
  calmode="p",
  refant="DV22",
  gaintype="G")
Plot resulting phase calibration

plotcal(caltable="phase_scan.cal", xaxis="time", yaxis="phase", subplot=331, iteration="antenna", plotrange=[0,0,-180,180], markersize=10, figfile="sis14_phase_scan.png")
Flux Calibration

Flux calibration requires estimating the flux of the secondary calibrator. We will get there by bootstrapping from the flux of the primary calibrator (in this case the planet Ceres), which is known from a well understood model.

Before we begin, we want to remove any short timescale phase variation from the sources involved in the flux calibration. Do so using gaincal.

- `os.system("rm -rf phase_int.cal")`
- `gaincal(vis="sis14_twhya_bpcal.ms",
  caltable="phase_int.cal",
  field="0,2,3",
  solint="int",
  calmode="p",
  refant="DV22",
  gaintype="G")`
Plot on a short timescale phase calibration

```
plotcal(caltable="phase_int.cal", xaxis="time", yaxis="phase", subplot=331, iteration="antenna", plotrange=[0,0,-180,180], figfile="sis14_phase_int.png")
```
Check uv range of model

Our primary calibrator is a solar system body (Ceres). These can often be resolved, which can complicate any attempt to use them as calibrators. Best practice using these targets for flux calibration is to identify a subset of antennas or (more easily) a uv range over which the planetary disk shows a strong response. Look at the uv range of the model using plotms and try to identify such a range.

• plotms(vis="sis14_twhya_bpcal.ms",
  xaxis="uvdist",
  yaxis="amp",
  ydatacolumn="model",
  field="2",
  averagedata=T,
  avgchannel="1e3",
  avgtime="1e3")
Apply short timescale phase solutions

It looks like 0~150m is probably a good u-v range to be able to calibrate using Ceres. Now let's run an amplitude solution, first applying the short-timescale phase solution *only for this u-v range.

```
• os.system("rm -rf apcal_shortuv.cal")
• gaincal(vis="sis14_twhya_bpcal.ms",
  caltable="apcal_shortuv.cal",
  field="0,2,3",
  solint="inf",
  calmode="a",
  uvrange="0~150",
  gaintype="G",
  refant="DV22",
  gaintable="phase_int.cal")
```

Experiment and apply solutions over a longer or shorter uvrangle and see what happens…
Plot calibration, amplitude vs. time for each antenna

- `plotcal(caltable="apcal_shortuv.cal", xaxis="time", yaxis="amp", subplot=331, iteration="antenna", plotrange=[0,0,0,0])`
Correct flux of calibrators

The gaincal solved for the amplitude scaling to make the data match the current model. For Ceres, we have taken care to set the correct model using setjy. For the other two calibrators, however, we don't a priori know the flux. Those have been calibrated using the default model, which is a point source of amplitude 1 Jy at the middle of the field. We now use fluxscale to bootstrap from the (correct) flux of Ceres through the amplitude calibration table to estimates of the true flux of the other two calibrators. This will output both a new table and the flux estimates themselves.

- os.system("rm -rf flux_shortuv.cal")
- fluxscale(vis="sis14_twhya_bpcal.ms",
   caltable="apcal_shortuv.cal",
   fluxtable="flux_shortuv.cal",
   reference="2")
Plot the rescaled flux table, which now should contain the correct flux calibrations. It will not be our final amplitude table, though, because we only solved over short u-v distance baselines.

- `plotcal(caltable="flux_shortuv.cal", xaxis="time", yaxis="amp", subplot=331, iteration="antenna", plotrange=[0,0,0,0])`
Amplitude Calibration

From fluxscale, we see that the two quasars have fluxes of \( \sim 0.65 \) and \( \sim 8.4 \) Jy. Using the task `setjy`, we will adjust the model of these sources to reflect these flux estimates:

- `setjy(vis=\"sis14_twhya_bpcal.ms\", field=\"3\", fluxdensity = [0.65,0,0,0], usescratch=True)`

- `setjy(vis=\"sis14_twhya_bpcal.ms\", field=\"0\", fluxdensity = [8.43,0,0,0], usescratch=True)`
Amplitude solution

Now we have the model correct for the two quasars, which - as point sources - are useful calibrators for all u-v ranges. We can run another amplitude solution without restricting the u-v range (though note that this will push things a bit on Ceres).

- os.system("rm -rf flux.cal")
- gaincal(vis="sis14_twhya_bpcal.ms",
  caltable="flux.cal",
  field="0,2,3",
  solint="inf",
  calmode="a",
  gaintype="G",
  refant="DV22",
  gaintable="phase_int.cal")
Final flux calibration table

This is our final flux calibration table. Inspect the amplitude corrections for each antenna.

- plotcal(caltable="flux.cal", xaxis="time", yaxis="amp", subplot=331, iteration="antenna", plotrange=[0,0,0,0])
Apply Calibration

Apply our flux calibration and the (scan based) phase solution to all fields (including the science target).
Note that we use the non-standard "calonly" command, which tells applycal not to flag data for which the calibration has failed.

```python
applycal(vis="sis14_twhya_bpcal.ms",
    field="",
    gaintable=["phase_scan.cal", "flux.cal"],
    interp="linear",
    applycal="calonly")
```
Inspect the Results

Look at amplitude vs. time first in calibrated data

- `plotms(vis="sis14_twhya_bpcal.ms", xaxis="time", yaxis="amp", ydatacolumn="corrected", field="0,2,3", averagedata=T, avgchannel="1e3", avgt ime="1e3", coloraxis="field")`
Inspect the Results

Look at amplitude vs. time first in model

- `plotms(vis="sis14_twhya_bpcal.ms", xaxis="time", yaxis="amp", ydatacolumn="model", field="0,2,3", averagedata=T, avgchannel="1e3", avgtime="1e3", coloraxis="field")`
Split out calibrated data

Now that we are satisfied with the gain calibration, we split out the gain calibrated data for further processing.

- os.system("rm -rf sis14_twhya_calibrated.ms")
- split(vis="sis14_twhya_bpcal.ms",
  outputvis="sis14_twhya_calibrated.ms",
  datacolumn="corrected",
  keepflags=False)

This produces one of the supplied data products (sis14_twhya_calibrated.ms) - so you can restart from a successful version of this script anytime.
Outline

• Short introduction to CASA and the Python interface
  – Discussion of “tools” will be done separately
• The Flow of Calibration
• Key CASA tasks for data reduction/calibration
• **Data Inspection and Flagging**
• Basic Imaging
Data Inspection and Flagging and End to End processing

ALMA Data Reduction Tutorials
Synthesis Imaging Summer School
May 16, 2014
Key Tasks for Data Inspection/Editing

Initial Inspection Tools
- **listobs**: list contents of a MS
- **plotant**: plot antenna positions

Inspect Your Data and Results
- **plotms**: inspect/flag your data interactively
- **plotcal**: examine a calibration table
- **listcal**: list calibration table data

Flagging
- **flagdata**: flag (remove) bad data
- **flagcmd**: batch flagging using lists/tables
- **flagmanager**: storage/retrieval of flagging state
Data Inspection and Flagging

- This next step goes through the basics of data inspection and flagging.
- Throughout the calibration process you will want to create a series of diagnostic plots and use these to identify and remove problematic data. This lesson steps through common steps in identifying and flagging problematic data.
- In the next lesson, we will see how this interplays with calibration in a typical iterative workflow.
- We will now use plotms to make a series of diagnostic plots. These plots have been picked because we have a good expectation of what the calibrators (fields 0, 2, and 3 here) should look like in each space. Before that however, let’s walk through the plotms GUI to familiarize ourselves with the interface.
**plotms**

A general-purpose graphical interface for plotting and flagging UV data

Can be started in the usual *casapy* interface

- `inp plotms`

Can be fully specified in the CASA command line (e.g.):

- `plotms(vis="sis14_twhya_bpcal.ms", xaxis="time", yaxis="amp", ydatacolumn="corrected", field="0,2,3", averagedata=T, avgchannel="1e3", avgtime="1e3", coloraxis="field")`

Also can be started directly from the unix prompt:

- `% casaplotms`
Data Review: plotms

Top Tabs

Side Tabs

Control Panel

Graphics Panel

Tools Panel
Data Review: plotms

Control panel: Data

The modification of certain parameters may not be applied if ‘Plot’ is clicked and ‘force reload’ is unchecked.
Data Review: plotms

Control panel: Axes

Drop down menus to select x and y axes: time, channel, frequency, velocity, amplitude, phase, uvdist, elevation, etc.
Data Review: *plotms*

**Iteration**

- Scan
- Field
- Spw
- Baseline
- Antenna

**Tool panel**
Data Review: *plotms*

**Display**

Colorize by:
- Scan
- Field
- Spw
- Antenna1
- Antenna2
- Baseline
- Channel
- Correlation
Data Review: plotms

Transformations

Frame: TOPO, GEO, BARY, LSRK, LSRD, etc..
Flagging: Locating Bad Data - plotms

Draw a box around the suspected bad data.
Flagging: Locating Bad Data - *plotms*

Click locate and CASA will send information about the data to the logger.
Bad data can be flagged by pressing this button or using the `flagdata` task at the CASA prompt.
Flagging: Locating Bad Data - *plotms*

Flagger’s remorse can be corrected by unflagging good data.
Flagging: Initial Flagging

• Shadowing
  – Issue at low elevations
  – Issue for compact arrays
  – In CASA: `flagdata(vis='my_data.ms', mode='shadow')`

• Observing Log
  – Many observatories will note weather or hardware problems that affect the data.

• Other obvious errors
Flagging: Initial Flagging

Tsys plots
NGC 3256 ALMA CASA Guide
Flagging: What to Look For

- Plots of amplitude and phase vs. time and frequency
- Iterate over
  - Antenna
  - Spectral window
  - Source
- Make plots of calibrators first
  - Easier to find problems in observations of bright point source
  - Harder to find problems in observations of a faint and extended source
Flagging: What to Look For

- Smoothly varying phases and amplitudes can be calibrated
- Discontinuities cannot be calibrated
- Features in the calibrators that may not be in the target data can cause problems
Flagging: What to Look For

Amp vs. Time

From TW Hydra ALMA Guide
Color: Polarization
One spectral window (spw) plotted
Flagging:
What to Look For
Amplitude vs. Frequency - Birdies

From TW Hydra CASA Guide
Brown and Green show phase calibrators
Flagging: What to Look For

Edge Channels

Amplitude vs. Channel

Amplitude vs. Channel
Flagging:
What to Look For

Data that should be flagged

Amplitude vs. Channel
Flagging:
What to Look For

From TW Hydra Band 7 Guide
Spectral line in Titan (Flux Calibrator)
Flagging: What to Look For
Phase vs. Time on Gain Calibrator

From Antennae ALMA CASA Guide
Gain calibrator observations on one antenna

First observation of data
Later observations of data
From Rick Perley:
“When in doubt, throw it out.”
Inspect your Data

In general, we will look through these plots one at a time and look for data that appears as outliers. Use the "locate" function, manipulate the plotted axes, and change the data selection and averaging to try to identify the minimum way to specify the problem data (antenna, scan, channel, etc.). Keep in mind here that the *science* data are not generally shown and will still need to be flagged.

Start with plots of amplitude and phase vs. uv distance. For point sources we expect flat amplitude and zero phases for these plots.
Amp vs. UVdist

plotms(vis="sis14_twhya_calibrated.ms", xaxis="uvdist", yaxis="amp", ydatacolumn="corrected", field="0,2,3", averagedata=T, avgchannel="1e3", avgtime="1e3", iteraxis="field", coloraxis="corr")
Phase vs. UVdis

We see some outliers. Using "locate" we clearly see that DV19 is having problems for the bandpass calibrator, showing low amplitudes. Let's have a look at amplitude vs. time

```python
plotms(vis="sis14_twhya_calibrated.ms", xaxis="uvdist", yaxis="phase", ydatacolumn="corrected", field="0,2,3", averagedata=T, avgchannel="1e3", avgtime="1e3", iteraxis="field", coloraxis="corr")
```
Amp vs. Time

plotms(vis="sis14_twhya_calibrated.ms", xaxis="uvdist", yaxis="phase", ydatacolumn="corrected", field="0,2,3", averagedata=T, avgchannel="1e3", avgtime="1e3", iteraxis="field", coloraxis="corr")
Amp vs. Time

Again, you see the same issues. We'll note that we want to flag DV19. Potentially we could flag it only on field 0, it's not totally clear that it's bad throughout the track. However, this means we will need to come up with an alternative calibration scheme for the bandpass (possibly using the other quasar). We will plan to flag DV19.

Next we will inspect phase and amplitude as a function of antenna. Each visibility (point) has two antennas associated with it, which are identified as Antenna1 and Antenna2 based on their number (the lower number is always 1). Remember here that we don't expect astrophysical signals to show up strongly as a function of antenna unless the antenna sits in a very unusual point in the array (check your plotants).
Amp vs. antenna

```
plotms(vis="sis14_twhya_calibrated.ms", xaxis="antenna1", yaxis="amp", ydatacolumn="corrected", field="0,2,3", averagedata=T, avgchannel="1e3", avgtime="1e3", iteraxis="field", coloraxis="corr")
```
Amp vs. antenna2

plotms(vis="sis14_twhya_calibrated.ms", xaxis="antenna2", yaxis="amp", ydatacolumn="corrected", field="0,2,3", averagedata=T, avgchannel="1e3", avgtime="1e3", iteraxis="field", coloraxis="corr")
You can see the problems with DV19 on field 0 here and you can also notice that DV01 has low amplitudes on field 3 (all amplitudes are low for this antenna). You may also notice that DV20 (antennae 22) shows some (but not all) low amplitudes, mostly on scan 30. We will note this and also plan to flag DV01.

Now look at the same plot in phase.
Phase vs. antenna

plotms(vis="sis14_twhya_calibrated.ms", xaxis="antenna1", yaxis="phase", ydatacolumn="corrected", field="0,2,3", averagedata=T, avgchannel="1e3", avgtime="1e3", iteraxis="field", coloraxis="corr")
The issue with DV 22 is particularly evident in these phase plots on Field 3, especially looking at Antennae2 (because 22 is a high number it's more commonly found as Antennae2).

```python
plotms(vis="sis14_twhya_calibrated.ms", xaxis="antenna2", yaxis="phase", ydatacolumn="corrected", field="0,2,3", averagedata=T, avgchannel="1e3", avgtime="1e3", iteraxis="field", coloraxis="corr")
```
You can see that likely DV19 is only really problematic in the first part of the track. Later on, scans 26 ~ 34, we see issues with DV20 (Antenna 22). We will plan to flag DV 20 over that range.

```
plotms(vis="sis14_twhya_calibrated.ms", xaxis="scan", yaxis="phase", ydatacolumn="data", field="3", antenna="", averagedata=T, avgchannel="1e6", iteraxis="field", coloraxis="corr", avgtime="1e6", avgscan=False)
```
Finally, we don't expect strong lines in the calibrators and sharp unexpected spikes anywhere are likely to be spurious. We will likely want to flag any lines or spikes. Plot the amplitude and phase as function of channel for the calibrators.

Field 2 (Ceres) shows an unexpected spike around channel 130. That seems spurious and we will want to flag this channel range.
plotms(vis="sis14_twhya_calibrated.ms", xaxis="channel", yaxis="amp", ydatacolumn="corrected", field="0,2,3", averagedata=T, avgtime="1e6", iteraxis="field", coloraxis="corr")
Phase vs. Channel

plotms(vis="sis14_twhya_calibrated.ms", xaxis="channel", yaxis="phase", ydatacolumn="corrected", field="0,2,3", averagedata=T, avgtime="1e6", avgchannel="10", iteraxis="field", coloraxis="corr")
Phase vs. Time

Finally, one can look at the continuity of the phase vs. time of each antenna paired with the reference. This is a good way to really dig into the data, but can be overwhelming. The previous plots (which show all data together) are probably your best first line of defense.

```python
plotms(vis="sis14_twhya_calibrated.ms", xaxis="time", yaxis="phase", ydatacolumn="data", field="3", antenna="DV22&*", averagedata=T, avgchannel="1e6", iteraxis="antenna", coloraxis="corr", avgscan=True)
```
Flag your Data

We decided to flag DV19 and DV01 for all scans, DV20 for scans 26-34, and channels 124-130 on Ceres (Field 2). We do this using the flagdata command in its "manual" mode.

First flag the two antennas entirely.

•  flagdata("sis14_twhya_calibrated.ms", antenna="DV01,DV19")

Now specify a scan range for DV20.

•  flagdata("sis14_twhya_calibrated.ms",
             antenna="DV20",
             scan="27~34")

Finally, pick a field and a channel/spw range for Ceres.

•  flagdata("sis14_twhya_calibrated.ms",
             field="2",
             spw="0:124~130")
Flag your Data

We could split out the flagged data here, but we would rather take the knowledge of these flags back to the beginning of the calibration process. That is the next lesson.

For now, go repeat the commands above to see the effect of your data flagging and convince yourself that these commands will remove most of the problems that you see in the data.
Run through the calibration scheme now “end-to-end” in CASA

- This final lesson before we get to imaging will step you through a realistic calibration workflow. As such, there will be little in the way of plots in this part of the presentation.
- You will start with the uncalibrated data. Then you will execute a calibration script.
- After you inspect the data - as in the last tutorial - you will apply some flagging and then rerun the calibration.
- At the end, we will split out the calibrated and flagged data for further imaging.
Run Calibration Script

The next few slides on calibration are all steps inside “calibration_script.py”. You can run this script with the commands:

- `vis = "sis14_twhya_uncalibrated"
- `execfile("calibration_script.py")`
Calibrate without Flags

Start by removing previous calibrations

- \texttt{vis = "sis14\_twhya\_uncalibrated"}
- \texttt{clearcal(vis\+".ms")}
Bandpass Calibration

A short-timescale phase solution

- os.system("rm -rf phase_int_bp.cal")
- gaincal(vis=vis+".ms",
  caltable="phase_int_bp.cal",
  field="0",
  solint="int",
  calmode="p",
  refant="DV22",
  gaintype="G")
Bandpass Calibration

Calibrate the bandpass

- os.system("rm -rf bandpass_10chan.cal")
- bandpass(vis=vis+".ms",
  caltable="bandpass_10chan.cal",
  field="0",
  refant="DV22",
  solint="inf,10chan",
  combine="scan",
  gaintable=["phase_int_bp.cal"]
)
Bandpass Calibration

Apply!

- `applycal(vis=vis+.ms,`
  - gaintable=['bandpass_10chan.cal'],
  - interp=['nearest'],
  - gainfield=['0'])

- `os.system("rm -rf "+vis+_bpcal.ms")`
- `split(vis=vis+.ms,`
  - datacolumn="corrected",
  - outputvis=vis+_bpcal.ms",
  - keepflags=False)"
Set Calibrator Fluxes

Look up the model for Ceres

- setjy(vis=vis+"_bpcal.ms",
  field="2",
  standard="Butler-JPL-Horizons 2012",
  usescratch=True)

Set the model for the bandpass calibrator

- setjy(vis=vis+"_bpcal.ms",
  field="0",
  fluxdensity = [8.43,0,0,0],
  usescratch=True)

Set the model for the secondary calibrator

- setjy(vis=vis+"_bpcal.ms",
  field="3",
  fluxdensity = [0.65,0,0,0],
  usescratch=True)
Phase and Amplitude

Derive a short-timescale phase solution

- os.system("rm -rf phase_int.cal")
- gaincal(vis=vis+"_bpcal.ms",
  caltable="phase_int.cal",
  field="0,2,3",
  solint="int",
  calmode="p",
  refant="DV22",
  gaintype="G")
Phase and Amplitude

Calibrate the phase

- os.system("rm -rf phase_scan.cal")
- gaincal(vis=vis+"_bpcal.ms",
  caltable="phase_scan.cal",
  field="0,2,3",
  solint="inf",
  calmode="p",
  refant="DV22",
  gaintype="G")
Phase and Amplitude

Calibrate the amplitude

- os.system("rm -rf amp_scan.cal")
- gaincal(vis=vis+_bpcal.ms,
  caltable="amp_scan.cal",
  field="0,2,3",
  solint="inf",
  calmode="a",
  refant="DV22",
  gaintype="G",
  gaintable=["phase_int.cal"])
Application

scan based applied to everything

- `applycal(vis=vis+_bpcal.ms`,
  gaintable=["phase_scan.cal", "amp_scan.cal"],
  interp=["linear", "linear"],
  gainfield=[[],[]],
  applymode='calonly')

- Now inspect the data, following the previous lessons. Go back and review, or try a few of the same commands from those lessons.
Flagging

Here we apply the already worked-out flagging
First flag the two antennas entirely.

- `flagdata("sis14_twhya_uncalibrated.ms",
  antenna="DV01,DV19")`

Now specify a scan range for DV20.

- `flagdata("sis14_twhya_uncalibrated.ms",
  antenna="DV20",
  scan="27~34")`

Finally, pick a field and a channel/spw range for Ceres.

- `flagdata("sis14_twhya_uncalibrated.ms",
  field="2",
  spw="0:124~130")`
Rerun calibration with Flags

After flagging problematic data we want to rerun the entire calibration with the problem data removed (those data could affect the calibration of other antennas, so that removing them will improve the overall data quality).

- `vis = "sis14_twhya_uncalibrated"
- `execfile("calibration_script.py")`
Split Out Calibrated Data

Finally, let's split out the calibrated, flagged data. These should now be ready for imaging.

- os.system("rm -rf sis14_twhya_calibrated_and_flagged.ms")
- split(vis="sis14_twhya_uncalibrated_bpcal.ms",
  outputvis="sis14_twhya_calibrated_and_flagged.ms",
  datacolumn="corrected",
  keepflags=False)
Basic Imaging

ALMA Data Reduction Tutorials
Synthesis Imaging Summer School
May 16, 2014
Imaging in CASA

• Goals of this lesson:
  – Introduce deconvolution in CASA (clean)
  – Introduce various imaging methods available in CASA
How to analyze (imperfect) interferometer data?

- **image plane analysis**
  - dirty image $T^D(x,y) = \text{Fourier transform } \{V(u,v)\}$
  - deconvolve $b(x,y)$ from $T^D(x,y)$ to determine (model of) $T(x,y)$

visibilities  ➔  dirty image  ➔  sky brightness
Basic CLEAN Algorithm

① Initialize a *residual* map to the dirty map

1. Start loop
2. Identify strongest feature in *residual* map as a point source
3. Add this point source to the clean component list
4. Convolve the point source with $b(x,y)$ and subtract a fraction $g$ (the loop gain) of that from *residual* map
5. If stopping criteria not reached, do next iteration

② Convolve *Clean component* (cc) list by an estimate of the main lobe of the dirty beam (the “Clean beam”) and add *residual* map to make the final “restored” image
Basic CLEAN Algorithm (cont)

- **stopping criteria**
  - residual map max < multiple of rms (when noise limited)
  - residual map max < fraction of dirty map max (dynamic range limited)
  - max number of clean components reached (no justification)

- **loop gain**
  - good results for $g \sim 0.1$ to 0.3
  - lower values can work better for smoother emission, $g \sim 0.05$

- **easy to include a priori information about where to search for clean components (“clean boxes”)**
  - very useful but potentially dangerous!
Dirty Beam Shape and Weighting

• Each visibility point is given a weight in the imaging step

• Natural
  – Weights inversely proportional to noise variance
  – Best point-source sensitivity; poor beam characteristics

• Uniform
  – Weights inversely proportional to sampling density (longer baseline are given higher weight than in natural)
  – Best resolution; poorer noise characteristics

• Briggs (Robust)
  – A graduated scheme using the parameter robust
  – In CASA, set $\text{robust}$ from -2 (~ uniform) to +2 (~ natural)
  – $\text{robust} = 0$ often a good choice
Imaging Results

Natural Weight Beam

CLEAN image
Imaging Results

Uniform Weight Beam

CLEAN image
Imaging Results

Robust=0 Beam

CLEAN image
Clean in CASA:

```bash
casa clean

# clean :: Invert and deconvolve images with selected algorithm
vis = ''
imagename = ''
outlierfile = ''
field = ''
specline = ''
selectdata = False
mode = 'mfs'
iter = 1
reffreq = ''
gridmode = '
height = 100
psfmode = 'clark'
imagemode = 'csclean'
cyclefactor = 1.5
cyclespeedup = -1

# Deconvolution scales (pixels); [] = standard clean
multiwave = []

# Use interactive clean (with GUI viewer)
interactive = False
mask = []
imsizex = 256
isizey = 256
cell = ['1.0arcsec']
phasecenter = ''
restfreq = ''
stokes = I
weighting = 'natural'
uvtaper = False
modelimage = ''
restoringbeam = ['']
pbcor = False
minpb = 0.2
usecsp = False
allowchunk = False
async = False
```

155
Basic Image Parameters: Pixel Size and Image Size

- **pixel size**
  - should satisfy $\Delta x < 1/(2 \ u_{\text{max}})$ $\Delta y < 1/(2 \ v_{\text{max}})$
  - in practice, 3 to 5 pixels across the main lobe of the dirty beam

- **image size**
  - Consider FWHM of primary beam (e.g. ~ 20” at Band 7)
  - Be aware that sensitivity is not uniform across the primary beam
  - Use mosaicing to image larger targets
  - Not restricted to powers of 2

*if there are bright sources in the sidelobes, they will be aliased into the image (need to make a larger image)*
**Maximum Angular Scale**

<table>
<thead>
<tr>
<th>Band</th>
<th>Frequency (GHz)</th>
<th>Primary beam (“”)</th>
<th>Range of Scales (“”)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>C32-1</td>
</tr>
<tr>
<td>3</td>
<td>84-116</td>
<td>72 - 52</td>
<td>4.2 - 24.6</td>
</tr>
<tr>
<td>6</td>
<td>211-275</td>
<td>29 - 22</td>
<td>1.8 - 10.7</td>
</tr>
<tr>
<td>7</td>
<td>275-373</td>
<td>22 - 16</td>
<td>1.2 - 7.1</td>
</tr>
<tr>
<td>9</td>
<td>602-720</td>
<td>10 – 8.5</td>
<td>0.6 - 3.6</td>
</tr>
</tbody>
</table>

- **Range** from synthesized beam to maximum angular scale (MAS)
- **Smooth** structures larger than MAS begin to be resolved out.
- All flux on scales larger than $\lambda/B_{\text{min}}$ ($\sim$2 x MAS) completely resolved out.
Copy the calibrated and flagged data from the working directory. This is our best version of the data.

```
   os.system("cp -r ../../../working_data/sis14_twhya_calibrated_flagged.ms .")
```

Orient yourself

```
   listobs('sis14_twhya_calibrated_flagged.ms')
```

<table>
<thead>
<tr>
<th>INFO listobs</th>
<th>Fields: 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Code Name</td>
</tr>
<tr>
<td>0</td>
<td>none</td>
</tr>
<tr>
<td>2</td>
<td>none</td>
</tr>
<tr>
<td>3</td>
<td>none</td>
</tr>
<tr>
<td>5</td>
<td>none</td>
</tr>
<tr>
<td>6</td>
<td>none</td>
</tr>
</tbody>
</table>

INFO listobs Spectral Windows: (1 unique spectral windows and 1 unique polarization setups)

<table>
<thead>
<tr>
<th>INFO listobs</th>
<th>SpwID Name</th>
<th>#Chans</th>
<th>Frame</th>
<th>Ch0(MHz)</th>
<th>ChanWid(kHz)</th>
<th>TotBW(kHz)</th>
<th>BBC</th>
<th>Num</th>
<th>Corrs</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ALMA_RB_07#BB_2#SW-01#FULL_RES</td>
<td>384</td>
<td>TOPO</td>
<td>372533.086</td>
<td>610.352</td>
<td>234375.0</td>
<td>2</td>
<td>XX</td>
<td>YY</td>
</tr>
</tbody>
</table>
plotms(vis='sis14_twhya_calibrated_flagged.ms', xaxis='u', yaxis='v',
avgchannel='10000', avgspw=False, avgtime='1e9', avgscan=False,
coloraxis="field")

Plot the u-v coverage
Basic Imaging

clean(vis='sis14_twhya_calibrated_flagged.ms', imagename='secondary', field='3', spw='', mode='mfs', nterms=1, imsize=[128,128], cell=['0.1arcsec'], weighting='natural', threshold='0mJy', interactive=True)
Basic Imaging

Clean Residuals
Output of `clean`

Minimally:

<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>my_image.flux</code></td>
<td>Relative sky sensitivity - shows the primary beam response</td>
</tr>
<tr>
<td><code>my_image.image</code></td>
<td>Cleaned and restored image</td>
</tr>
<tr>
<td><code>my_image.mask</code></td>
<td>Clean “boxes” shows where you cleaned</td>
</tr>
<tr>
<td><code>my_image.model</code></td>
<td>Clean components - the model used by clean (in Jy/pixel)</td>
</tr>
<tr>
<td><code>my_image.psf</code></td>
<td>Dirty beam - shows the synthesized beam</td>
</tr>
<tr>
<td><code>my_image.residual</code></td>
<td>Residual shows what was left after you cleaned (the &quot;dirty&quot; part of the final image)</td>
</tr>
</tbody>
</table>
Basic Imaging

`imview("secondary.image")`
Basic Imaging

clean(vis='sis14_twhya_calibrated_flagged.ms', imagename='secondary_robust', field='3', spw='', mode='mfs', nterms=1, imsize=[128,128], cell=['0.1arcsec'], weighting='briggs', robust=-1.0, threshold='0mJy', interactive=True)
Basic Imaging

Clean Residuals
Basic Imaging

`imview("secondary_robust.image")`
Basic Imaging

clean(vis='sis14_twhya_calibrated_flagged.ms', imagename='primary_robust', field='2', spw='', mode='mfs', nterms=1, imsize=[128,128], cell=[0.1arcsec], weighting='natural', threshold='0mJy', interactive=True)
Basic Imaging

Field 2

Clean Residuals
imview("primary_robust.image")
clean(vis='sis14_twhya_calibrated_flagged.ms', imagename='secondary_bigpix', field='3', spw='', mode='mfs', nterms=1, imsize=[32,32], cell=['0.5arcsec'], weighting='natural', threshold='0mJy', interactive=True)
Basic Imaging

Clean Residuals
Basic Imaging

imview("secondary_bigpix.image")
Basic Imaging
Basic Imaging - Aside

```python
import clean(vis='sis14_twhya_uncalibrated.ms', imagename='secondary_uncalibrated', field='3', spw='', mode='mfs', nterms=1, imsize=[128, 128], cell=['0.1arcsec'], weighting='natural', threshold='0mJy', interactive=True)
```

Dirty Image
Basic Imaging - Aside

```
clean(vis='sis14_twhya_calibrated.ms', imagename='secondary_unflagged', field='3', spw='',
mode='mfs', nterms=1, imsize=[128,128], cell=['0.1arcsec'], weighting='natural',
threshold='0mJy', interactive=True)
```

Dirty Image
Basic Imaging

Split out the science data and smooth using width='10'

```python
split(vis='sis14_twhya_calibrated_flagged.ms', field='5', width='10',
      outputvis='twhya_smoothed.ms', datacolumn='data')
```

Orient yourself

```python
listobs('sis14_twhya_calibrated_flagged.ms')
```

```
INFO listobs   Date        Timerange (UTC)          Scan  FldId FieldName  nRows SpwIds Average Interval(s)    ScanIntent
INFO listobs   19-Nov-2012/07:56:23.5 - 08:02:11.3    12      0 TW Hya      8514  [0]  [6.05] [OBSERVE_TARGET#ON_SOURCE]
INFO listobs   08:08:09.6 - 08:13:57.3    16      0 TW Hya    10360  [0]  [6.05] [OBSERVE_TARGET#ON_SOURCE]
INFO listobs   08:19:53.9 - 08:25:41.7    20      0 TW Hya    10321  [0]  [6.05] [OBSERVE_TARGET#ON_SOURCE]
INFO listobs   08:32:00.5 - 08:37:48.2    24      0 TW Hya    10324  [0]  [6.05] [OBSERVE_TARGET#ON_SOURCE]
INFO listobs   08:43:45.6 - 08:49:33.4    28      0 TW Hya     9462  [0]  [6.05] [OBSERVE_TARGET#ON_SOURCE]
INFO listobs   09:05:15.6 - 09:07:31.6    36      0 TW Hya     4180  [0]  [6.05] [OBSERVE_TARGET#ON_SOURCE]
INFO listobs               (nRows = Total number of rows per scan)
INFO listobs Fields: 1
INFO listobs   ID   Code Name                RA               Decl           Epoch   SrcId      nRows
INFO listobs   0    none TW Hya              11:01:51.796000 -34.42.17.36600 J2000   0          53161
INFO listobs Spectral Windows:  (1 unique spectral windows and 1 unique polarization setups)
INFO listobs   SpwID  Name                           #Chans   Frame   Ch0(MHz)  ChanWid(kHz)  TotBW(kHz) BBC Num  Corrs
INFO listobs   0      ALMA_RB_07#BB_2#SW-01#FULL_RES     39   TOPO  372535.833      6103.516    234375.0       2  XX  YY
```

NRAO
clean(vis='twhya_smoothed.ms', imagename='twhya_cont', field='0', spw='', mode='mfs', nterms=1, imsize=[250,250], cell=['0.08arcsec'], weighting='briggs', robust=0.5, threshold='0mJy', interactive=True)
Basic Imaging

One cycle of cleaning
Basic Imaging

Three cycles of cleaning
Basic Imaging

`imview("twhya_cont.image")`
Basic Imaging

```python
clean(vis='twhya_smoothed.ms', imagename='twhya_cont_auto', field='0', spw='', mode='mfs', nterms=1, imsize=[250,250], cell=['0.08arcsec'], mask='box [ [ 100pix , 100pix] , [150pix, 150pix ] ]', weighting='briggs', robust=0.5, threshold='15mJy', niter=5000, interactive=False)
```