Modeling with ALMA

G.S. Mathews

While ALMA observations are revealing many new details of the millimeter universe, the interpretation of these data are still dependent on comparisons with models. In order to compare data with models, one must first have an idea of what information can be extracted from your data, and have a good route for extracting the same type of information from your models. This second step usually involves some approach to simulating an observation.

This document provides a walkthrough to the full process of comparing a model to an observation, starting with extraction of information from data, simulating an observation of a model, and then practicing a few methods of comparison between the two. This document does not address the production of a model in the first place.

As much as possible, we will use commands in CASA. However, there are some functions which are not yet enabled, and in these cases I present example code in IDL. I attempt to explain this code in a thorough enough fashion that it could be replicated in other languages.

1 Preparations

There are several items of data and sample models which are to be downloaded and used with this walkthrough. In addition, there are two custom CASA packages that have been developed by Attila Juhász, with the Allegro Alma Regional Center node. These are as follows:

- Observation: HD163296.spw2.CO32.split.ms
  - This is made from spw 2 from HD163296_Band7_concat.ms, in the publicly released data. Channels 1000 to 2600 selected for faster operations

  - Note that the central velocity of the source is at 7.01 km/s LSRK in this data set, rather than the actual central velocity of 5.8 km/s. This error is noted in a footnote in Rosenfeld et al. 2013.

- Model: image0.fits (J=3-2 image generated by Lime with 0.05" pixels, for a source at 122 pc)
- Script: simobs_custom
- Script: visbin
- Script: getBaselineLengths and buildConfigurationFile from ALMA analysisTools
  download here: http://casaguides.nrao.edu/index.php?title=Analysis_Utilities
  more documentation here: https://safe.nrao.edu/wiki/bin/view/Main/CasaExtensions
- IDL codes:
  - atv - http://www.physics.uci.edu/~barth/atv/instructions.html
  - mrdfits, from the Goddard library - http://idlastro.gsfc.nasa.gov
  - hist_nd, from the Coyote library - http://www.idlcoyote.com/documents/programs.php
  - uvfits.pro, by G. Mathews

1.1 Installing simobs_custom and visbin

1. place the script in a folder for holding new CASA commands (e.g. ~/home/scripts/CASA/)
2. follow the instructions in the README (summarized here)
   (a) cd to the directory containing the script
   (b) load CASA
   (c) os.system('buildmytasks')
   (d) Add the file mytasks.py, with path, to your /.casa/init.py file

2 Brief introduction to interferometer data

The raw data from an interferometer are *visibilities*. Briefly, these are data describing how much emission is present at a particular angular scale on sky. The angular scale and direction is given by the *(u,v)* position of the data point. The angular scale is inversely proportional to the uv-distance from the origin, and each axis corresponds to a direction on sky (u - east west, v - north south). At a given uv point, the interferometer reports a complex number. The amplitude corresponds to the flux at that particular size scale and direction, and the phase reflects shifts from being centered at the target of the observation.

For more overview, see the ALMA primer at http://almatelescope.ca/ALMA-ESPrimer.pdf.

For more in-depth information, see the notes from Allegro workshops, at http://www.alma-allegro.nl/allegroworkshops. In particular, see:

- The June 2012 workshop has a talk by Markus on proposal preparation, submission, and review that has a nice stepthrough on some interferometry basics. In particular, there is a nice sequence of images showing the build up of the response pattern as you add more and more antennae (his is the example case of filling in more and more of a giant antenna, but same idea)
- Dec. 2011 - Pam's talk on mm interferometry (first talk) gives a thorough presentation of the mathematics behind interferometry
- Dec. 2011 - Markus' talk on calibration and imaging theory does a step-through showing what happens with CLEAN, the most common algorithm for inverting data from the visibility to the image plane.
- Dec. 2011 - Markus gave a nice talk on Inversion & Imaging, including a full walkthrough of the self-calibration procedure for improving image quality
- Dec. 2011 - Pam gave a nice talk on how specific errors can be detected, "What would happen if."

2.1 Manual illustration of visibilities $\leftrightarrow$ image

To briefly illustrate the concept behind visibilities and the reconstruction of an image, here is a simple set of IDL commands that generate a set of visibilities and do a simple step by step reconstruction of an image.

1. create a pair of 2-dimensional arrays containing X and Y values. These are analogous to values of ΔRA and ΔDeclination in an astronomical image.

   ```
   x = reverse((findgen(1001) - 500) / 10)
   y = (findgen(1001) - 500) / 10
   ones = make_array(1001, value=1.0)
   ```
   ; now generate a 1001 × 1001 array where each element contains the X value, from 50 to -50
   ```
   xa = x # ones
   ```
   ; now generate a 1001 × 1001 array where each element contains the Y value, from -50 to 50
   ```
   ya = ones # y
   ```
   ; inspect by eye the structure arrays to ensure they show values from east-to-west, and north-to-south
   ```
   atv, xa
   ```

2. generate a single *(u,v)* point and see what structure it corresponds to in the image plane.

   ```
   uva = [2, 0.1, 1]
   im1 = uva[2] * COS(xa*uva[0] + ya*uva[1])
   atv, im1
   ```

   This shows that the object on sky has some structure about 15 units wide, angled just north of east-west. In this example, the phases are all set to zero indicating that the interferometer was aimed at the center of the structure, and that the emission was symmetric about the center.

3. Add a second point. Using '&' allows you to place multiple commands on the same line - I do that here to save space.

```
uva = [0.5,1.0,1] & im2 = uva[2] * COS(xa*uva[0] + ya*uva[1])
& atv, im2
```

This shows that the object also has an ∼25 unit wide structure angles along a direction about 30 degrees west of north.

4. Examine several more features.

```
uva = [-1.0,2.0,1] & im3 = uva[2] * COS(xa*uva[0] + ya*uva[1])
& atv, im3

uva = [-4.0,2.0,1] & im4 = uva[2] * COS(xa*uva[0] + ya*uva[1])
& atv, im4

uva = [-0.2,0.5,1] & im5 = uva[2] * COS(xa*uva[0] + ya*uva[1])
& atv, im5
```

Notice that larger *uv-distances* ($r_{uv} = \sqrt{u^2 + v^2}$) correspond to smaller physical scales.

5. Try making a few of your own sample *(u,v)* points. The variable *uva* contains the u, v, and amplitude, respectively.

6. Phase can be added by giving *uva* a fourth element, and changing the expression for the image to:

```
im5 = uva[2] * COS(xa*uva[0] + ya*uva[1] + uva[3] * !PI / 180)
```
(where the term !PI / 180 converts a phase from degrees to radians)

7. Few astronomical objects actually look like stripes across the sky, however. These points all represent structures that were detected by particular pairs of antennae at particular times (we use the rotation of the earth itself to have each pair of detectors in the interferometer sweep through u,v space). Summing these images, however, will begin to give a representation of how the object looks on sky. Notice that the spatial cosine functions have begun to interfere, leaving a bright point at the middle.

```
im = im1 + im2 + im3 + im4 + im5
atv, im
```

Of course, a typical observation will have much more than 5 u,v points. You can simulate having an approximately uniform u,v coverage with a simple function such as this:

```
ruv = (findgen(500)+1) / 10
theta = (findgen(500)+1) / 10
u = ruv * COS(theta)
v = ruv * SIN(theta)
a = make_array(500, value=1)
plot, u, v, psym=2 ; plot the u,v coverage
im[*] = 0.0
for n=0,499 do im = im + a[n] * COS((xa*u[n] + ya*v[n]))
atv, im
```

Notice that the u,v points in this example have a uniform amplitude. In lectures on interpreting interferometer data, we're often told that a point source will have a constant amplitude in u,v space. Here we see why that is - all the cosine functions are maximized at a position of 0,0 but elsewhere they destructively interfere. Try writing a function to make the amplitude, a, a function of u and v, and see how that changes the resulting image.

e.g. `a = 1.0 / (ruv^0.1) * cos(ruv * 1.0 / 5)`

3 Information extraction

This tutorial will start with an overview of different pieces of information which can be extracted directly from ALMA observations. If the same types of data are then extracted from models, then the two may be quantitatively compared, as with the calculation of χ^2 as part of a parameter search algorithm.

3.1 Preparation for analysis

With line data, you should work with a dataset that has been placed in a common reference frame, and which has had the continuum subtracted. This can be accomplished with the following tasks:

1. Check that data from multiple nights is in the same restframe. If you see the line at multiple frequencies, then use *cvel* to place the data into LSRK

   ```
   plotms(vis='HD163296.spw2.CO32.split.ms', xaxis='channel', yaxis='amp',
   averagedata = True, avgchannel = '', avgtime = '100000s')

   cvel(vis = 'HD163296.spw2.CO32.split.ms', outputvis =
   'HD163296.CO32.split.LSRK.ms', outframe = 'lsrk')
   ```

2. Select a subset of the data around the line frequency to speed up operations. In a sense, this tutorial is cheating on this step because the original data file has already been split from the full observation.

   ```
   split(vis='HD163296.CO32.split.LSRK.ms', outputvis='HD163296.CO32.LSRK.ms',
   spw='0:500~1200', datacolumn='data')
   ```

3. identify line free channels for the continuum subtraction. If the line is not clearly present, make a best-guess to channels that lie off of the line frequency.

   ```
   plotms(vis='HD163296.CO32.LSRK.ms', xaxis='channel', yaxis='amp',
   averagedata = True, avgchannel = '', avgtime = '100000s')
   ```

4. Subtract the continuum

   ```
   uvcontsub(vis='HD163296.CO32.LSRK.ms', fitspw='0:50~200;480~680',
   fitorder=1)
   ```

   This generates a new file, HD163296.CO32.LSRK.ms.contsub

5. Use *plotms* to examine the new dataset, setting the axes to *Amplitude* vs. *channel*. If the continuum region now has a rising slope or weird bend, try the following corrections to the preceding steps:

(a) expand the regions used in the continuum fits. If necessary, carry out continuum subtraction on the data set prior to the *split* command

(b) try setting fitorder = 0 (but only if you are highly certain there is no continuum slope)

6. Convert from the visibility plane to the image plane. This will also make a mask that will be useful in later analysis. A common algorithm for converting a visibility dataset to the image plane is CLEAN. While the details of this process are beyond the scope of this workshop, here is a sample CLEAN command for CASA:

```
clean(vis='HD163296.CO32.LSRK.ms.contsub', imagename = 'HD163296.CO32',
mode='velocity', nchan = 150, start='-10km/s', width='0.20km/s',
outframe='LSRK', imsize=[300,300], cell=['0.1arcsec'], restfreq=
'345.7959899GHz', stokes='I', threshold='150mJy', interactive=True)
```

It is also possible to first CLEAN the *cvel* corrected .ms, and then use *imcontsub* to subtract the continuum.

3.2 *(u,v)* coverage

Lay, Carlstrom, and Hills (1997, ApJ, 489, 917) has a nice illustration of the transform between the image and visibility planes.

Within CASA, *plotms* is a primary tool for directly examining visibility datasets. Opening a visibility dataset (.ms folder) and examining the distribution of points in u,v space will give you a qualitative idea as to whether your observation will have resolution and sensitivity differences along different axes. The r_{uv} extent along a given direction gives a sense of the resolution in that direction - greater r_{uv} indicates finer resolution. The density of points will somewhat reflect the fidelity of the image reconstruction, reflecting the ability to detect structure on many different size scales.

Open the sample observation in *plotms* and set the x and y axes to show the u and v values.

```
plotms(vis='HD163296.CO32.LSRK.ms.contsub', xaxis='u', yaxis='v',
averagedata = True, avgchannel = '', avgtime = '100000s', plotrange=[400,-400,
-400, 400])
```

Qualitatively, you can see that there is relatively uniform coverage at baselines up to about 200 kλ. At longer baselines (i.e. smaller physical scales) image fidelity may start to drop. Coverage extends to longer baselines in along the north-south direction (v), which will correspond to having a smaller beamsize, or better resolution, along that direction.

If you want to make a figure of *(u,v)*-coverage using a different set of tools, you will need to export the visibilities as a uvfits file. For example:

```
exportuvfits(vis='HD163296.CO32.LSRK.ms.contsub', fitsfile=
'HD163296.CO32.vis.fits', datacolumn='data')
```

In the Appendix, I describe the structure of the resulting .fits file.

3.3 Size scales

To examine issues of image fidelity more quantitatively, the minimum resolvable size scales and largest observable scale can be calculated. These functions are taken from the *ALMA Primer*[1]

- Determine the shortest and longest baseline lengths using:

  ```
  au.getBaselineLengths('HD163296.CO32.LSRK.ms.contsub', sort=True)
  ```

  In this case, the shortest and longest wavelengths are 13.1 and 402.3 m

- The smallest resolvable size scale is based on the longest baseline in the observation (i.e. the longest distance between antennas in the array).

$$\theta_{min} \approx 0.2'' \times (300/\nu[\mathrm{GHz}]) \times (1[\mathrm{km}]/\text{max baseline}) \quad (1)$$

- The largest observable size scale (beyond which flux is negligible due to interferometric filtering) is

$$\theta_{\mathrm{LOS}} \approx 0.6'' \times (\lambda/b_{\mathrm{short}}) \times (180/\pi) \times 3600 \quad (2)$$

[1] https://almascience.nrao.edu/documents-and-tools/cycle-1/alma-es-primer

3.4 Simple model fits

Some tools are available for fitting basic morphology of the observed object. Such fits are most useful for identifying the center of emission, which is in turn useful for some comparisons of models to data. Within CASA, the *uvmodelfit* task allows you to fit a single point, elliptical Gaussian, or elliptical uniform disk model to the visibilities. This can take several minutes. e.g.,

```
uvmodelfit(vis='HD163296.CO32.LSRK.ms.contsub', spw='0:250~400',
comptype='G', sourcepar = [1.0, 0.0, 0.0, 2.0, 0.5, 45.0], varypar
= [T,T,T,T,T,T])
```

In addition to providing an estimate of the source flux, this can give an estimate of the source central position. However, remember that under ALMA's sharp-eyed gaze, fewer and fewer objects will look like simple gaussians or point sources. For example, the sample data resolves the three-dimensional structure of the CO emitting surface in the disk, including seeing the back side of the disk. A gaussian fit to such a structure will lead to an offset from the true central position. For more on this, see Rosenfeld et al. 2013.

An alternate way to use fits to line data may be to emulate the modeling approach of D. Harsono (REF) in finding the center of emission, channel by channel, to then construct a PV diagram, e.g.:

```
uvmodelfit(vis='HD163296.CO32.LSRK.ms.contsub', spw='0:300~301',
comptype='G', sourcepar = [1.0, 0.0, 0.0, 2.0, 0.5, 45.0], varypar
= [T,T,T,T,T,T])
```

For a ring morphology, or to fit a multi-component morphology, you will need to export the visibilities and fit in another code, e.g. MIRIAD.

3.5 Images and data cubes

In the image plane, an astronomer can carry out flux measurements and morphology analysis. The data must first be converted, however, from visibilities to the image plane. This should have already been done as part of the earlier preparations (§3.1).

3.5.1 Noise

Perhaps the most important statistic for describing a dataset is the noise level. This then places limits on what else is achievable with the data. Within CASA, load the cleaned .image dataset into the viewer, where you can then use built in tools to measure the noise.

1. scroll through the channels in the data cube. Find a channel that is relatively free of line emission.
2. left click on one of the region selection icons on the toolbar. Select a region that is also away from where the source lies.
3. Double-click on that region. The statistics for the region will appear in the terminal.
4. record the RMS noise (most commonly reported in units of Jy / beam)

In addition to visual inspection, this can be scripted using *imstat*

3.5.2 Flux measurement

There are several approaches to measuring the flux of the source.

- Make a 0th moment map, and do standard photometry.
  Use *immoment* with either a threshold (via includepix, set to 2 or 3σ, e.g., includepix = '15mJy') for selecting x, y, v points to include, or a channel-by-channel mask. Such a mask can be manually constructed in the viewer (this is a bit laborious) or by using the prototype boxit command. If one made a channel-by-channel mask at the CLEAN step in processing the observation, then this mask could be reused here.

  Then, in viewer, select a region and double click. Among other statistics, the total flux is reported. As an alternative, use *immath* to sum all the pixels within a target region.

- use the line fit feature in the spectra viewer
  Select a region in the image viewer that contains all the emission. Open the spectra viewer, and use the line fit feature. The integrated emission will be displayed among many other statistics in the terminal.

- export the spatially integrated spectrum and integrate.

4 Simulating observations

4.1 Files used

- HD 163296 CO 3-2 visibilities (HD163296.CO32.LSRK.ms.contsub)
- Sample Lime model of HD 163296 CO 3-2 line emission (image0.fits)

4.2 Approximating the uv-coverage with simobserve

1. Ensure the input model has the needed header information:

   RESTFREQ (in Hz), RADESYS = FK5, SPECSYS = LSRK, CRVAL1 (central RA, in degrees), CRVAL2 (central Dec, in degrees), CRVAL3 (in m/s), CUNIT3 = 'M/S'

   Example IDL code:

   ```
   im = mrdfits('image0.fits', 0, hd)
   sxaddpar, hd, 'RESTFREQ', 345.7959899E9
   sxaddpar, hd, 'RADESYS', 'FK5'
   sxaddpar, hd, 'SPECSYS', 'LSRK'
   sxaddpar, hd, 'CRVAL1', ten('17:56:21.2704')*15
   sxaddpar, hd, 'CRVAL2', ten('-21:57:22.205')
   sxaddpar, hd, 'CRVAL3', 7010.0
   sxaddpar, hd, 'CUNIT3', 'M/S'
   writefits, 'model.simobserve.fits', im, hd
   ```

2. Extract the antenna positions of the observations.

   ```
   au.buildConfigurationFile(vis='HD163296.CO32.LSRK.ms.contsub',
   dropTPpads=True)
   ```
   s

   This creates a file HD163296.CO32.LSRK.ms.contsub.cfg which should contain most of the antenna positions.

3. Determine the time on source

   ```
   au.timeOnSource('HD163296.CO32.LSRK.ms.contsub')
   ```

4. Load the model image into CASA format.

   ```
   importfits(fitsimage='model.simobserve.fits', imagename='CO32.model',
   zeroblanks=False)
   ```

5. Simulate observation.

   ```
   simobserve(project='sim', skymodel = '../model/CO32.model',
   indirection = 'J2000 17h56m21.2704 -21d57m22.205', antennalist
   =
   'HD163296.CO32.LSRK.ms.contsub.cfg', hourangle = '-2:30:00',
   totaltime = '4400s', direction = 'J2000 17h56m21.2704 -21d57m22.205')
   ```

6. After CLEANing the resulting .ms, you might adjust hourangle in order to better match the beam of the observation. Adjust date and start time to approximate uv coverage of the actual observations. For later comparison with the observation, you will need to subtract the continuum, as well.

7. One can improve the approximation by running simobserve several times, matching the several dates and times of actual observations. The resulting .ms files can then be catenated together.

4.3 Matching the exact uv-coverage using simobs_custom

1. Convert model .fits image to have the necessary features for simobs_custom:

   - Units: Intensity in Jy/px
   - Header: RA and Declination (CTYPE1, CRPIX1, CRVAL1, CDELT1, CUNIT1, and 2)
   - Header: frequency or velocity axis (CTYPE3, CRPIX3, CRVAL3, CDELT3, CUNIT3)
   - Header: Stokes axis (CTYPE4)

   If your data lacks velocity data (as with a continuum image) or stokes data (e.g. Lime models lack polarization), you will need to add "dummy" dimensions to your data. E.g.

IDL: Many functions and procedures, including the popular mwrfits, reform arrays to remove dimensions with a depth of 1. Therefore, if exporting a model from IDL, use something like the following:

```
im = mrdfits('image0.fits', 0, hd)
sxaddpar, hd, 'RESTFREQ', 345.7959899E9
sxaddpar, hd, 'RADESYS', 'FK5'
sxaddpar, hd, 'SPECSYS', 'LSRK'
sxaddpar, hd, 'CRVAL1', ten('17:56:21.2704')*15
sxaddpar, hd, 'CRVAL2', ten('-21:57:22.205')
sxaddpar, hd, 'CRVAL3', 7010.0
sxaddpar, hd, 'CUNIT3', 'M/S'
sz = size(im,/dim)
fits_write, 'model.simobs_custom.fits', reform(im, [sz[0],
sz[1], sz[2], 1]), hd
```

where image is the variable name for the image, and sz is a three dimensional array with the size of each dimension in the x and y directions and the velocity axis, respectively.

CASA: I attempted to force my model .fits image to have the proper format by importing the image to CASA and then exporting to .fits. This did not work, unfortunately - it did not add a Stokes dimension to the data. There may be a way to add a dummy Stokes dimension in CASA, but for now, we can just use a simple IDL script.

2. Create a simulated uv-dataset
Simobs_custom will create a new .ms with two columns: 'data' will contain the original observation, while 'model' will contain the model observations. It was setup this way to allow for the further creation of a residual uv data set, if so desired (this is discussed below, in Section REF). Here is an example call:

```
simobs_custom(vis='../observation/HD163296.CO32.LSRK.ms.contsub',
project = 'sim', skymodel = '../model/model.simobs_custom.fits',
mode = 'line')
```

To examine the simulated visibilities, open the new .ms in plotms.

Under Axes, select the Data Column: 'model'. Notice that the .ms contains the parent continuum-subtracted observation in the 'data' Data Column.

3. Make a continuum subtracted model dataset.

   (a) Extract the simulated observations of the model.

   ```
   split(vis='sim/sim.ms', outputvis='sim/modelOnly.ms', datacolumn='model')
   ```

   (b) Use *plotms* to identify continuum channels
   e.g. 0:110~150,450~525

   (c) carry out the continuum subtraction. If the continuum regions are short (as in this case) it may be better to use fitorder = 0. With observations, where there may be an actual continuum slope, it is best to use fitorder = 1

   ```
   uvcontsub(vis='sim/modelOnly.ms', fitspw='0:110~150;450~525',
   fitorder=0)
   ```

   this makes a new .ms (in this case, sim/modelOnly.ms.contsub) with the continuum subtracted model visibilities in the DATA column

   (d) As usual, check the resulting .ms using *plotms*.

4. Make a line image (data cube)
   Ideally, this will be done using the same clean mask used to make the final image of the observations. e.g.

   ```
   clean(vis='sim/modelOnly.ms.contsub', imagename='sim.CO32',
   mode='velocity', nchan = 150, start='-10km/s', width='0.20km/s',
   outframe='LSRK', imsize=[300,300], cell=['0.1arcsec'],
   restfreq='345.7959899GHz', stokes='I', threshold='150mJy',
   interactive=False, mask = '../observation/HD163296.CO32.mask')
   ```

5 Comparing simulated model observations to actual observations

5.1 Generate the visibility residuals

In order to create a uv-residual data set, you can do the following:

1. outside of CASA, copy the observation and simulated observation to a new .ms

   ```
   cp -r sim/sim.ms sim/resid.ms
   ```

2. replace the MODEL column in the new .ms with the continuum subtracted model.

   ```
   tb.open('sim/modelOnly.ms.contsub')
   tb.colnames
   model = tb.getcol('DATA')
   tb.close
   tb.open('sim/resid.ms', nomodify=False)
   tb.colnames
   tb.putcol('MODEL_DATA',model)
   tb.flush
   tb.close
   ```

3. Examine the DATA, CORRECTED, and MODEL columns of resid.ms in *plotms*. 'DATA' should match the continuum subtracted observation, 'MODEL' should match the continuum subtracted model, and 'CORRECTED' should be empty.

4. *uvsub* will calculate the residual, placing it in the CORRECTED_DATA column of resid.ms

   ```
   uvsub(vis='sim/resid.ms')
   ```

It is common to show the CLEAN image of the residual of the bestfit model.

```
clean(vis='sim/resid.ms', imagename='resid.CO32', mode='velocity',
```

```
nchan = 150, start='-10km/s', width='0.20km/s', outframe='LSRK',
imsize=[300,300], cell=['0.1arcsec'], restfreq='345.7959899GHz',
stokes='I', threshold='150mJy', interactive=False,
mask = '../observation/HD163296.CO32.mask')
```

5.2 unbinned, point-by-point comparison

Ideally, one would compare the least-processed data to the model on a point-by-point basis. The previous section has shown how to use CASA's *uvsub* command to generate the residual. With the large number of uv points in an observation (furthermore multiplied by the number of frequencies compared!), the reduced-χ^2 of the best-possible fit converges to 1, which can in some ways ease interpretation. On the other hand, if this data is combined with any other data set, it will overwhelm that other data's contribution (and hence swamp the ability to constrain parameters that primarily influence the other data set).

Unfortunately, it is unclear to me how to properly convert the weights that are assigned to each data point to an uncertainty. I have not been able to find documentation describing how the weights are calculated - assuming something along the lines of $\sigma = 1/w^2$ should work, at least for a minimization, but the resulting χ^2 contours will not necessarily map onto uncertainties (though that is generally not formally the case anyway with this sort of work).

Furthermore, there are no built-in functions for doing arbitrary math on visibility datasets, so this calculation will require doing calculations in Python or exporting to work with other languages e.g. IDL. To export the visibilities:

```
exportuvfits(vis = 'sim/sim.ms', fitsfile = 'observation.uv.fits',
datacolumn = 'data')
exportuvfits(vis = 'sim/resid.ms', fitsfile = 'residual.uv.fits',
datacolumn = 'corrected')
exportuvfits(vis = 'sim/modelOnly.ms.contsub', fitsfile = 'model.uv.fits',
datacolumn = 'data')
```

I have written an IDL function to load the visibility data from ALMA, *uvfits_read_ALMA* in the file *uvfits.pro*. There are then two followup procedures to deproject the visibilities, and to shift the phase center if necessary

(uvfits_deProject and uvfits_shiftPhaseCenter). To load the data, use e.g.,

```
obs = 1 & obshd = '1' & resid = 1 & residhd = '1' & stokes = 1 &
freq = 1 & u0 = 1 & v0 = 1 & vels = 1

uvfits_read_ALMA, 'observation.uv.fits', visstruc = obs, vishd =
obshd, stokes = stokes, freq = freq, u0 = u0, v0 = v0, vels = vels

uvfits_read_ALMA, 'residual.uv.fits', visstruc = resid, vishd = residhd
```

The variables stokes, freq, u0, v0, and vels will be filled with linear arrays of those parameters. The variables obs and resid will be arrays of structures, as described in the Appendix.

5.3 slightly binned, still capturing full *(u,v)* range

NOTE: This is the only code example where the coding approach is better in IDL than what I know can be achieved in Python. This is due to the *hist_nd* function used in the gridding, which includes a reverse indices that dramatically speeds up the calculation of the real and imaginary components in each bin by avoiding some degree of looping. Perhaps something clever can be written in Python (e.g., use a 2D histogram, then iterate through each occupied bin and use the Python equivalent of IDL's *where* function to speed the calculation).

This provides a solution to the problem of error estimation for the calculation of χ^2. It also eases the combination with other data sets by reducing the number of points (though additional relative weighting may still be needed), and allows for conceptually simpler estimation of uncertainties. $1\mathrm{k}\lambda \times 1\mathrm{k}\lambda$ bins will easily reduce a dataset to a few hundred points, while retaining coverage of the full *(u,v)* range of the observation. This cannot currently be done within CASA. Within IDL, the steps to use this would be:

1. load the observation

   ```
   uvfits_read_ALMA, 'observation.uv.fits', visstruc = obs, vishd
   = obshd, stokes = stokes, freq = freq, u0 = u0, v0 = v0, vels
   = vels
   ```

2. Create an array with the indices of the channels with line emission.

```
channels = findgen(110)+260
```

3. Grid the data in uv space, calculating the mean real and imaginary value as well as their uncertainty. *uvfits_make_gridded_uv* will by default split the uvspace into bins with a mean number of 20 integrations per bin. This can be changed by supplying a parameter *nbins*, which is the number of bins in both u and v. The ranges can also be set with 2-element arrays *urange* and *vrange*.

   ```
   gridded = uvfits_make_gridded_uv(obs, channels = channels)
   ```

4. The output will consist of a structure containing several arrays:

   - npts -
   - uArray -
   - vArray -
   - real -
   - real_Err -
   - imag -
   - imag_err -

5. Examine the arrays using *atv*, e.g.:

   ```
   atv, gridded.real
   atv, gridded.real / gridded.real_err
   atv, gridded.imag / gridded.imag_err
   atv, gridded.npts
   ```

5.4 a few bins in $R_{\rm uv}$, good for azimuthally symmetric data

Visibilities for an azimuthally symmetric source should have imaginary components of zero, in the case of the phase center being at the center of emission. In such a case, visibility profiles can be constructed in which the real and imaginary components are calculated as a function of $r_{\rm uv} = \sqrt{u^2 + v^2}$, as well as calculating the uncertainty in the mean in each bin.

These 1-dimensional profiles of the real component can then be interpreted in terms of simple gaussians or point sources. There has also been work dealing with the analytic interpretation of visibilities for rings (e.g. Hughes et al. 2007, ApJ, 664, 536). Non-zero values in the imaginary component reflect azimuthal asymmetries, and shifting the phase center to match the center of emission should minimize the imaginary components. Hughes et al. also outlines the transformations to deproject a set of visibilities (i.e., from inclined to face on, suitable for binning). Lay, Carlstrom, and Hills (1997, ApJ, 489, 917) Figure 3 is also a useful reference.

Attila has written a task called *visbin* to examine radial visibility profiles in CASA. This uses the information in the data column; therefore, to look at the profile of the model or residual, use *split* to place them in their own .ms files. e.g.

```
split(vis = 'sim/sim.ms', outputvis = 'observation.ms', datacolumn
= 'data')
split(vis = 'sim/resid.ms', outputvis = 'residual.ms', datacolumn
= 'corrected')
split(vis = 'sim/modelOnly.ms.contsub', outputvis = 'model.ms', datacolumn
= 'data')
```

The following is an example of the use of visbin to deproject in inclination and position angle, and bin the channels containing line emission.

```
visbin(vis='observation.ms', spw='0:260~370', incl = 44.0, posang=133.0,
binwidth = 10000, nbins = 40, binfreq=True, fit_phasecenter=False,
outfile='observation', overwrite=True)

visbin(vis='model.ms', spw='0:260~370', incl = 44.0, posang=133.0,
binwidth = 10000, nbins = 40, binfreq=True, fit_phasecenter=False,
outfile='model', overwrite=True)

visbin(vis='residual.ms', spw='0:260~370', incl = 44.0, posang=133.0,
binwidth = 10000, nbins = 40, binfreq=True, fit_phasecenter=False,
outfile='residual', overwrite=True)
```

derotates the observation from a position angle of 133 degrees, and de-inclines the visibilities from an inclination on sky of 44 degrees (which happen to be

the values from the literature for this particular disk). If there is a known position offset, it may be better to have a preceding step using *fixvis* to compensate (Note: visbin has an internal phaseshift capacity. However, I have not yet tested it, and therefore recommend for the moment the use of *fixvis*).

Otherwise, set *fit_phasecenter* to True. Note that the fitting routine will take a long time, and scales as the multiple of the number of offset points to test in RA and in Declination.

visbin will output a text file which contains the visibility profile, which can then be used for further calculations.

NOTE that program looks for multiple spws, and may display error messages if none are found. These can be safely ignored.

5.5 image plane comparisons

5.5.1 Multi-channel

The simplest way to make comparisons in the image plane is to use the viewer to overlay model contours on a raster map of your data. In order for this to work properly, you need to have CLEANed the simulated observation of the model with the same settings as the observations. It is also helpful to show contours for the observation (in a different color) and scale both sets of contours to show the same multiples of the uncertainty in the observation (e.g., 2, 3, 6, 9σ).

The comparison can be quantified and automated using *immath*, after using *immoments* with *moments=6* to construct a noise map.

```
immoments(imagename='myData.image', moments=6, axis='spectral',
chans='10~40;120~150', outfile='myData.noise')

immath(imagename = ['myData.image', 'myModel.image', myData.noise'],
mode=evalexpr, expr='((IM0 - IM1)*(IM0 - IIM1)) / (IM2 * IM2)', outfile
= 'chisqContribs')

imcollapse(imagename='chisqContribs', function='sum', axis = 1, outfile='temp')

chisq = imcollapse(imagename='temp', function='sum', axis = 1, outfile='temp2')
```

5.5.2 Spatially integrated spectra

With both datacubes being displayed in the viewer, the spectrum viewer will also display both lines simultaneously. The line fitting function will not fit both, however. It will fit the first one loaded first. You should then hide that image to fit the other. The properties of the fit will display in your terminal window. There may be a way to access those values programmatically, but I do not know it.

Also note that by default, the mean intensity is displayed in the spectrum viewer. There is a toggle under the display to change to showing the summed flux. Unfortunately, you will need to reset this everytime you open a new spectral viewer.

5.5.3 Position-velocity diagram

Position-velocity diagrams are common tools for examining kinematic information from data cubes. The ARTIST package includes a GUI tool for investigating PV diagrams, IVAN. CASA has a built in task, *impv*. With the center of emission and object position angle identified, one may examine the velocity profile to test e.g., Keplerian motion. This can provide additional constraints on e.g. the inclination of a disk determined using elliptical fits to the continuum or, if the inclination is assumed correct, constraints on the stellar mass.

To interactively examine the PV diagrams, use IVAN (a subset of ARTIST). On a Sterrewacht computer, do the following:

```
source /software/ARTIST/artisrc.csh (or .sh if using bash)
ivan
```

The final position-velocity diagram can be saved as a .fits file for other calculations, if desired.

The generation of a PV diagram can also be scripted, e.g.:

```
impv(imagename='myModel.image', outfile='myModel.pv', start= [200,195],
end=[318, 317])
```

Once PV diagrams are generated for both data and model, they can be compared as described above using *immath*.

6 Strategies for using and interpreting models

6.1 By-hand fits

At first, one should probably generate models and make comparisons manually, developing some intuitions for how different parameters of the models affect the resulting simulated observations and continually improving the steps in the process. There are many pieces that go into this sort of work, from the codes used to develop the models, to the processing of them and comparisons to data. Examining the output from each step is essential.

However, once one has finalized the approach to simulating observations and extracting metrics, and has developed a sense of the interesting parameters in the models, one should develop an automated way of exploring many models. There are two primary approaches to this task - a grid, and a parameter search algorithm. Below, I briefly discuss each, and outline an example approach.

6.2 A grid

With a grid, one predetermines a set of values to explore for the parameters of interest. Then, models are generated and examined. If the models change slowly enough across the grid, then intermediate values may be interpolated - this is, however, rare and risky.

The models that are produced by a grid can be examined by eye or by the calculation of metrics. Ideally, you will do both! Grids also provide the opportunity for uniform sampling of the parameter space, which can be useful for simplifying statistical analysis of the parameter uncertainties.

6.3 Parameter search algorithm

Numerous algorithms exist for automatically exploring a parameter space. These require that you develop a scriptable metric calculation which the algorithm will use in determining the parameter combinations for which it generates models.

One of the more easy-to-use extant tools is the MPFIT package for IDL (Markwardt, C.B., 20009, PASP 411, 251). To use this, one must first write a function that will generate a model given a set of parameters and calculate

it's χ^2 value. The MPFIT code then systematically varies the parameters and follows $\nabla\chi^2$ down to a minimum.

In order to avoid the possibility of settling on a local rather than globabl χ^2 minimum, one should run the search several times, starting in different regions of the parameter space. This has the added benefit of making the search more uniform, which is important for examining e.g. 2D χ^2 plots.

The user function may contain any number of operations. It is typical to write the function to facilitate later exploration of the models. For example, a code running with the goal of identifying the primary gas parameters of a circumstellar disk, the gas mass, scale height, and characteristic radius ($M_{\rm gas}$, $H_{\rm C}$, and $R_{\rm C}$, may have the following outline:

1. Load previously determined dust structure ($\rho_{\rm dust}$(R,h) and $T_{\rm dust}$(R,h)) and opacity information ($\kappa_{\rm dust}$)

2. Create a new folder to save diagnostics and intermediate products

3. Calculate $\rho_{\rm gas}$(R,h) using $M_{\rm gas}$, $H_{\rm C}$, and $R_{\rm C}$

4. Use $\rho_{\rm gas}$(R,h) to calculate the hydrogen column depth from infinity and from the star ($N_{\rm H_2}$), an important constraint on the formation of other molecules

5. Adopt $T_{\rm dust}$(R,h)) as a reasonable approximation of the gas temperature, $T_{\rm gas}$(R,h)). Save a diagnostic figure showing the gas density, temperature, and hydrogen column depth as a function of radius and height.

6. Use the hydrogen column depth and the gas temperature to determine the region in which CO is present (e.g. at $N_{\rm H_2} > 1.5 \times 10^{21}$ cm^{-2} and $T_{gas} > 20$ K)

7. Output the gas density and temperature in a format for loading to a radiative transfer code, e.g. LIME

8. Run the radiative transfer code for several lines and/or isotopologues of CO

9. Simulate observations of these models (see Section 4, or ALMA CASA documentation of simobserve)

10. Calculate a χ^2 value for each model. If several techniques are used (e.g. 1kλ $\times$ 1kλ binned visibilities for CO J=3-2, for which an interferometer observation is available, but just an integrated line flux for CO J=2-1, for which only a single dish spectrum is available in the literature), then each contribution to the χ^2 needs to be weighted (in this example case, perhaps the χ^2 from the binned visibilities, which have 100 times as many points as the 1D spectrum, is down weighted by a factor of 100). This is all that is necessary for the search algorithm, but perhaps the code also includes:

11. Create a CLEAN image of the model and generate a channel map showing the model overlain on the observation, to ease later examination of the models and parameter space. This can add time to the search and take up extra disk space.

It is usually best to follow-up such an automated parameter search with the production of a small uniform grid around the best-fit parameter, in order to provide a better estimate of the uncertainties that avoids biases in sampling that occur with automated searches.

A Structure of uv fits files from CASA

When a uvfits file output from CASA is read with the *mrdfits* function in the Goddard IDL library, a structure will be created. This will actually be a one-dimensional array of structures, each with two substructures. Each of the structures is a single integration. Below, I list the elements in these subarrays as well as the items added by my various functions and procedures.

array $0 \times 3 \times 2 \times n_{channel} \times 1 \times 1 \times 1$ array. The 0 and 1 element dimensions are reformed by most reader programs, leave a $3 \times 2 \times n_{channel}$ array. Each of these remaining columns selects for:

1. first column, with the elements selecting for:
   (a) The real component,
   (b) imaginary component, and
   (c) the weight.
   (d) amplitude (added by *uvfits_read_ALMA*)
   (e) phase (added by *uvfits_read_ALMA*)

(f) real components for the data with the phasecenter shifted. (added by *uvfits_shiftPhaseCenter*)

(g) imaginary components for the data with the phasecenter shifted. (added by *uvfits_shiftPhaseCenter*)

2. The second column selects between the XX and YY polarization components (which together add to twice Stokes I, e.g. see Synthesis Imaging in Radio Astronomy II, pg. 311).
3. The last column selects for the channel (frequency or velocity). Actually generating the frequencies or velocities requires the use of the corresponding header values (CRPIX4, CRVAL4, & CDELT4)

params contains the following information, in a 1D array:

1. UU - in seconds. SIN projection (i.e. along North)
2. VV - in seconds. SIN projection (i.e. along East)
3. WW - in seconds
4. DATE - day number
5. DATA - day fraction
6. BASELINE - baseline number
7. FREQSEL - frequency setup ID?
8. SOURCE - source ID number
9. INTTIM - integration time in seconds
10. u - in kλ (added by *uvfits_read_ALMA*)
11. v - in kλ (added by *uvfits_read_ALMA*)
12. R - in kλ (added by *uvfits_deProject*)
13. PHI - radian (added by *uvfits_deProject*)
14. DMAJ - (added by *uvfits_deProject*)
15. DMIN - (added by *uvfits_deProject*)
16. RUV - (added by *uvfits_deProject*)

The procedure *uvfits_read_ALMA* adds two dimensions to *array* - amplitude and phase, the alternative representation of the complex numbers. It also adds two new elements

uvfits_deProject adds 5 new elements to *params* - the original uv radius and angle on sky, $R_{uv,raw}$ and ϕ, as well as the deprojected uv distance along the major and minor axes and the deprojected uv-radius, d_{maj}, d_{min}, and R_{uv}.

uvfits_shiftPhaseCenter adds two more elements to the first column of *array*, the real and imaginary components for the data with the phasecenter shifted.

Here are some examples of plotting data that has been loaded into IDL with *uvfits_read_ALMA*.

- Initialize variables and read the observations

```
obs = 1 & obshd = '1' & model = 1 & modelhd = '1' & resid =
1 & residhd = '1' & stokes = 1 & freq = 1 & u0 = 1 & v0 = 1
& vels = 1

uvfits_read_ALMA, 'observation.uv.fits', visstruc = obs, vishd
= obshd, stokes = stokes, freq = freq, u0 = u0, v0 = v0, vels
= vels
```

- Also read in the model and residuals.

```
uvfits_read_ALMA, 'model.uv.fits', visstruc = model, vishd =
modelhd
uvfits_read_ALMA, 'residual.uv.fits', visstruc = resid, vishd
= residhd
```

- Display the u,v coverage

```
plot, obs[*].params[9], obs[*].params[10], psym=3
```

- display amplitudes of the XX components as a function of velocity

```
; make an array of velocities to match the dimensions of the nVis by
nFreq array ones_nUV = make_array(n_elements(obs))
plot, vels # ones_nUV, obs[*].array[3,0,*], psym=3
```

To see what I did here with the # operation, examine the variables:

```
help, vels, ones_nUV, vels # ones_nUV, obs[*].array[3,0,*]
```

- Show the weighted mean scalar-average amplitude as a function of velocity
```
weights = obs[*].array[2,*,*]
amps = obs[*].array[3,*,*]
stokes_I = REFORM(TOTAL(weights * amps, 2, /double) / TOTAL(weights,
2, /double)) / 2
weight_I = REFORM(1.0 / SQRT((1.0 / weights[*,0,*,*])^2 + (1.0
/ weights[*,1,*,*])^2))
avg = TOTAL(weight_I * stokes_I, 2, /double) / TOTAL(weight_I,
2, /double)
plot, vels, avg
```

- Show the weighted mean scalar-average amplitude as a function of velocity, for baselines longer than 200 kλ

```
use = where(SQRT(obs[*].params[9]^2 + obs[*].params[10]^2) LT
200)
weights = obs[use].array[2,*,*]
amps = obs[use].array[3,*,*]
stokes_I = REFORM(TOTAL(weights * amps, 2, /double) / TOTAL(weights,
2, /double)) / 2
weight_I = REFORM(1.0 / SQRT((1.0 / weights[*,0,*,*])^2 + (1.0
/ weights[*,1,*,*])^2))
avg = TOTAL(weight_I * stokes_I, 2, /double) / TOTAL(weight_I,
2, /double)
plot, vels, avg
```

- Show the weighted mean scalar-average amplitude as a function of velocity, for baselines longer than 200 kλ and for velocities from -15 to 15 km/s

```
useUV = where(SQRT(obs[*].params[9]^2 + obs[*].params[10]^2)
GT 200)
useChan = where(vels GT -15 and vels LT 15)
weights = obs[useUV].array[2,*,useChan]
amps = obs[use].array[3,*,useChan]
```

```
stokes_I = REFORM(TOTAL(weights * amps, 2, /double) / TOTAL(weights,
2, /double)) / 2
weight_I = REFORM(1.0 / SQRT((1.0 / weights[*,0,*,*])^2 + (1.0
/ weights[*,1,*,*])^2))
avg = TOTAL(weight_I * stokes_I, 2, /double) / TOTAL(weight_I,
2, /double)
plot, vels, avg
```

- Show the weighted mean vector-average amplitude as a function of velocity, for baselines longer than 200 kλ and for velocities from -15 to 15 km/s

```
useUV = where(SQRT(obs[*].params[9]^2 + obs[*].params[10]^2)
GT 200)
useChan = where(vels GT -15 and vels LT 15)
weights = obs[useUV].array[2,*,useChan]
reals = obs[use].array[0,*,useChan]
imags = obs[use].array[1,*,useChan]

stokes_I_real = REFORM(TOTAL(weights * reals, 2, /double) /
TOTAL(weights, 2, /double)) / 2
stokes_I_imag = REFORM(TOTAL(weights * imags, 2, /double) /
TOTAL(weights, 2, /double)) / 2
weight_I = REFORM(1.0 / SQRT((1.0 / weights[*,0,*,*])^2 + (1.0
/ weights[*,1,*,*])^2))
avg_real = TOTAL(weight_I * stokes_I_real, 2, /double) / TOTAL(weight_I,
2, /double)
avg_imag = TOTAL(weight_I * stokes_I_imag, 2, /double) / TOTAL(weight_I,
2, /double)
amp = SQRT(avg_real^2 + avg_imag^2)
plot, vels[useChan], avg
```

B Script outline

Either of the modeling approaches discussed in the main text, model grid or parameter search, will benefit from having your operations running from a script. In the following, I describe an example set of files and scripts for

generating a grid of models.

1. Model parameters file
   `Define the parameters of the models that will be generated`
2. Model iteration file
   `This is the script that is called to create the models. This may be combined with the 'Model parameters file'.`
   (a) Load observation metrics
   `Load metrics from the observations, to use repeatedly in calculating fit metrics for the models (e.g. the radial visibility profile of the observed object)`
   (b) Iterate through the models (grid), or calculate the parameters of the next model to run (algorithmic parameter search)
      i. Setup basic file structure
      `Make a folder for the new model, with subfolders for models, simulated observations, fit-metrics, and diagnostic figures`
      ii. Setup subsidiary parameter files
      `Each modeling code has its own formats for taking input. An example of this would be changing parameters in RADMCs problem_params.pro`
      iii. Save a list of the varied parameters in plain text in the model folder
      iv. Generate the model
      `Call a script to run the modeling code, e.g. from IDL, calling '.r problem_setup.pro' to generate a RADMC continuum model from dust emission`
      v. Simulate observation of the model
      `Call a script which has the simulation commands as shown in the main body of this document.`
      vi. Extract information
      `Call a script, or several scripts, to extract information from the simulated observation. It is a good idea for these scripts to also save figures for later examination.`

vii. Calculate metrics
`Use the extracted information from the model and the pre-loaded information from the observation to calculate e.g.` χ^2.

(c) Generate summaries
`Compiling the` χ^2 `of the generated models in a single table, or creating a file with diagnostic figures from all models (e.g. SEDs for dust modeling) can make later examination easier.`

C Shifting the phase-center

In the document, we recommend using *fixvis* to adjust the phasecenter when carrying out an analysis that requires centered data (e.g. visibility profiles for azimuthally symmetric sources). The underlying math for this operation can be seen in the function *uvfits_shiftPhaseCenter*, and the derivation is carried out below courtesy of Markus Schmalzl.