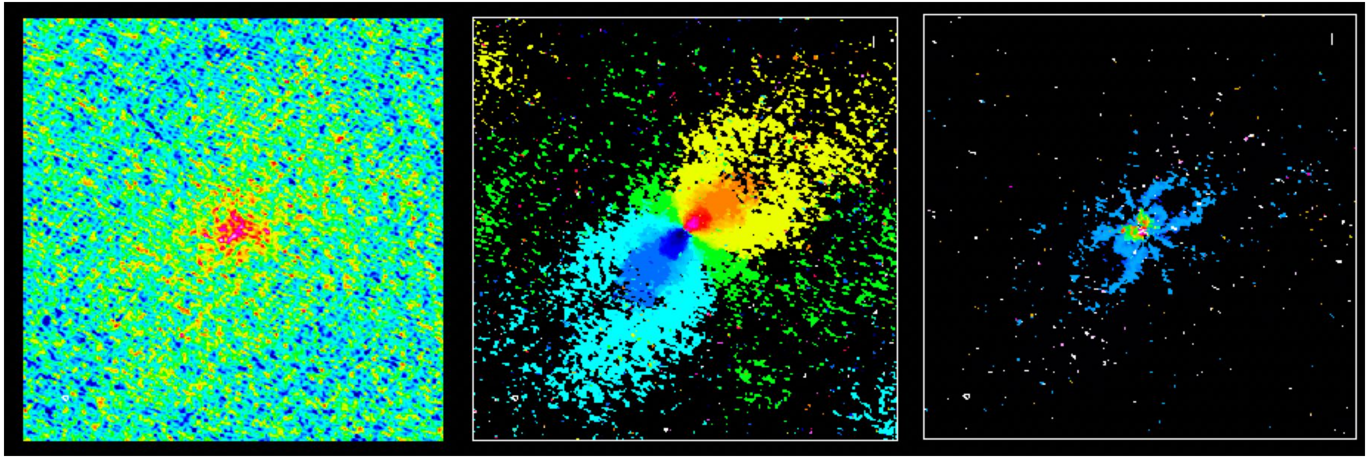
HD 163296 CASA
Spectral Line Reduction Tutorial
Imaging and Analysis



Alex Hygate and Katharine Johnston

CASA guides tutorial link

We will follow the script:

`HD163296_Imaging_Selfcal_withQs.py`

which can be downloaded from the ERIS website.

There is also a script with the answers included:

`HD163296_Imaging_Selfcal_answers.py`

The data can also be downloaded from the ERIS website:

`HD163296_data_new.tgz`

You can run the steps in the script in CASA as we go along

Which data to use

The calibrated data tar file:

`HD163296_data_new.tgz`

can be untarred/unzipped using: `tar -xvzf FILENAME`

It should contain:

`HD163296_CO_new.ms` and `HD163296_cont_all.ms`

You can recreate these data by:

- Running the script `scriptforPI.py` from the ALMA archive
- Averaging the calibrated data using the commented code at the start of `HD163296_Imaging_Selfcal_withQs.py`

Step 1: Listobs and save original flags

```
os.system('rm -rf HD163296_cont_all.ms.listobs’)
listobs(vis = 'HD163296_cont_all.ms',
        listfile = 'HD163296_cont_all.ms.listobs')

if not os.path.exists('HD163296_cont_all.ms.flagversions/
  flags.Original'):
  flagmanager(vis = 'HD163296_cont_all.ms',
              mode = 'save',
              versionname = 'Original')
```

Step 1: Listobs and save original flags

```
os.system('rm -rf HD163296_CO_new.ms.listobs’)
listobs(vis = 'HD163296_CO_new.ms’,
        listfile = 'HD163296.ms.listobs’)

if not
os.path.exists('HD163296_CO_new.ms.flagversions/
  flags.Original'):
  flagmanager(vis = 'HD163296_CO_new.ms',
              mode = 'save',
              versionname = 'Original')
```

Step 1: plotants

```
plotants(vis = 'HD163296_cont_all.ms', logpos=True,
            figfile='HD163296_cont_all.plotants.png')
```

Question: What’s the expected synthesized beam? (hint: use the output from plotants)

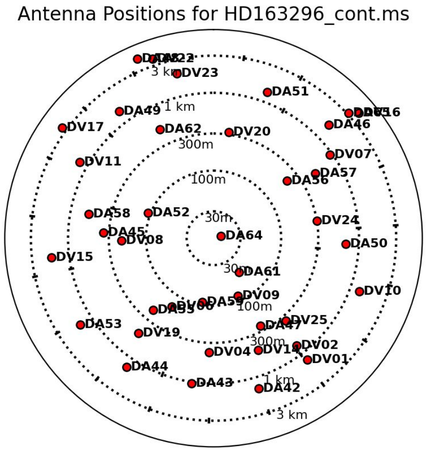
Step 1: plotants

```
plotants(vis = 'HD163296_cont_all.ms', logpos=True,
         figfile='HD163296_cont_all.plotants.png')
```

Question: What's the expected synthesized beam? (hint: use the output from plotants)

$$\Theta = \lambda / \mathbf{D}$$

D is the maximum baseline length



Step 1: plotants

```
plotants(vis = 'HD163296_cont_all.ms', logpos=True,
         figfile='HD163296_cont_all.plotants.png')
```

Question: What’s the expected synthesized beam? (hint: use the output from plotants)

$\Theta = \lambda / D$ where D is the maximum baseline length

$\lambda = c / \nu = 1.3$ mm $\quad\quad$ $D \sim 6$km

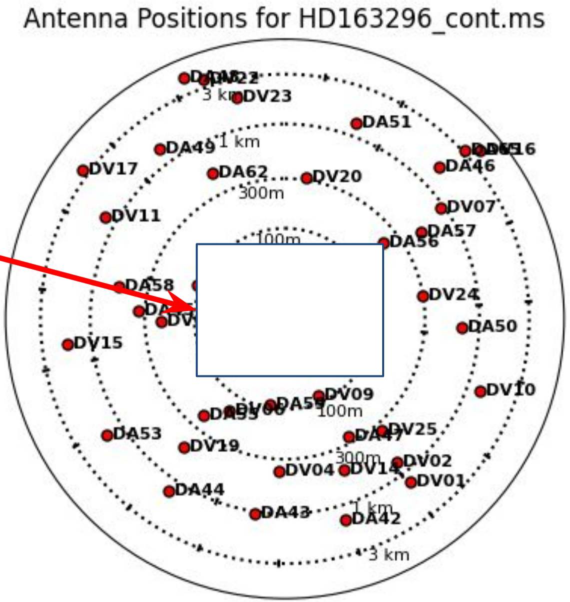
Get D from looking at `HD163296_cont_all.plotants.png`

Θ will initially be in radians, so need to convert to degrees/arcseconds

Step 1: plotants

```
plotants(vis = 'HD163296_cont_all.ms', logpos=True,
         figfile='HD163296_cont_all.plotants.png')
```

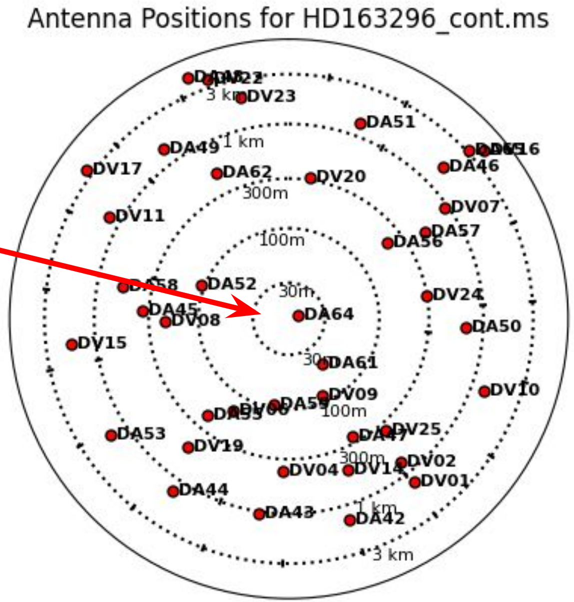
Question: Choose a good antenna in the centre as your reference antenna (hint: use the output from plotants)



Step 1: plotants

```
plotants(vis = 'HD163296_cont_all.ms', logpos=True,
         figfile='HD163296_cont_all.plotants.png')
```

Question: Choose a good antenna in the centre as your reference antenna (hint: use the output from plotants)



Set ref. ant. and cellsize for imaging

Set the reference antennas to be antennas close to centre of array with good data

antrefs= '***'

Set the cellsize so that there will be 4-5 pixels across the beam

cellsize='***arcsec'

Estimating the noise for imaging

Question:

Find out the total time on source using the weblog.

Can then use the ALMA sensitivity calculator to determine the expected noise (if have internet):

https://almascience.nrao.edu/proposing/sensitivity-calculator

Need: Declination, Obs. frequency, bandwidth of continuum, number of antennas, time on source (see listobs file!)

Estimating the noise for imaging

Question:

Find out the total time on source using the weblog.

Answer: Time on source ~ 54m 20s

Can then use the ALMA sensitivity calculator to determine the expected noise (if have internet):

https://almascience.nrao.edu/proposing/sensitivity-calculator

Need: Declination, Obs. frequency, bandwidth of continuum, number of antennas, time on source (see listobs file!)

Estimating the noise for imaging

Use the ALMA sensitivity calculator to determine the expected noise (if have internet connection):

https://almascience.nrao.edu/proposing/sensitivity-calculator

Need: Declination (-22deg), Obs. Frequency (~230.5GHz), bandwidth of continuum (6.938GHz), number of antennas (40), time on source (54.33min)

Expected sensitivity = ~20 microJy/beam

To do:

Find the sensitivity for the CO obs. for 2 km/s channels (Answer: ~1.2 mJy/beam)

You will use this later

Step 2: Make first continuum image

```
tclean(vis='HD163296_cont_all.ms',
        imagename='HD163296_cont.firstclean',
        specmode='mfs', deconvolver='multiscale',
        imsize=500, spw='0~3', scales=[0, 5, 10],
        cell=cellsize, weighting='briggs',
        robust=-0.5, interactive=interact,
        threshold='0.2mJy', niter=10000)
```

`mode='mfs'` – use multi-frequency synthesis algorithm for continuum imaging

`cell=,0.01arcsec'` – the synthesised beam at 350GHz should be ~0.05",
want 4-5 pixels across the beam

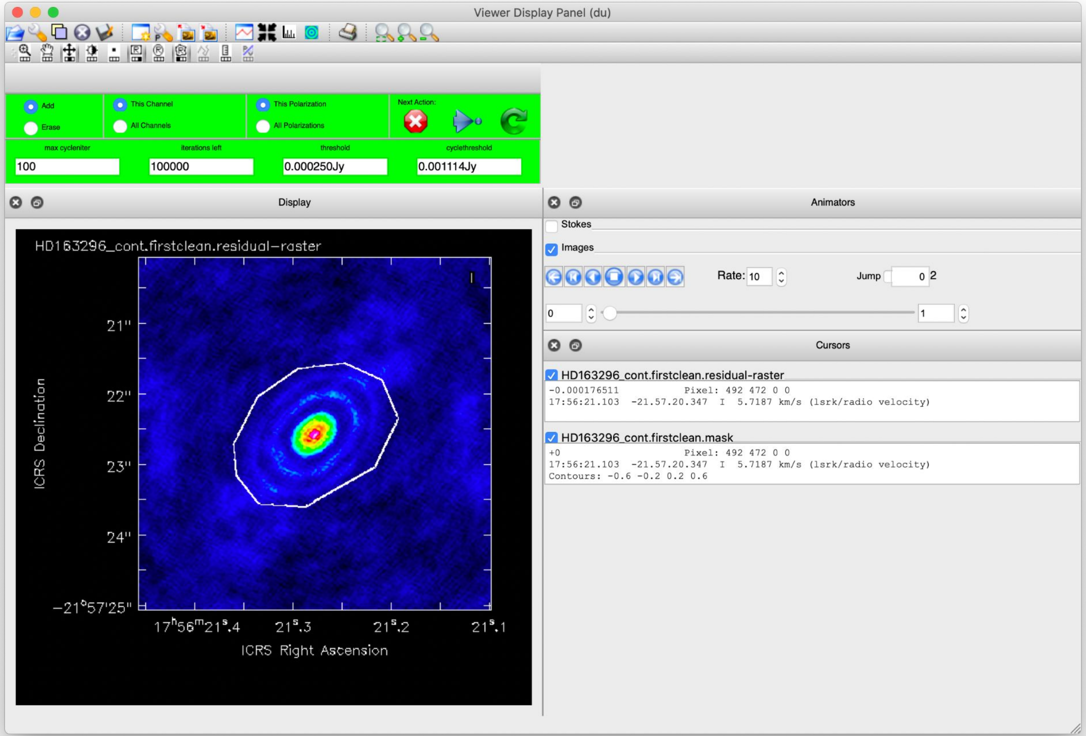
`imagesize=500` – emission is ~2.5" so 0.01"x 500 = 5" will cover it
if don't know in advance use previous obs. or trial and error (start small)

`weighting='briggs', robust=-0.5` – how you weight the data in uv-space
(taken from DSHARP papers here, find by experimentation)

`threshold='0.2mJy'` – a threshold for cleaning, several times noise (more if not self-cal'ed)

`niter=10000` – number of iterations, large so limit set by `threshold`

Step 2: Make first continuum image



Step 3: Make first continuum image for self-calibration

```
delmod(vis='HD163296_cont_all.ms', scr=True)
clearcal(vis='HD163296_cont_all.ms')

os.system('rm -rf HD163296_cont.clean*')
tclean(vis='HD163296_cont_all.ms',
       imagename='HD163296_cont.clean',
       specmode='mfs', deconvolver='multiscale',
       imsize=500, spw='0~3', scales=[0, 5, 10],
       cell=cellsize, weighting='briggs',
       robust=-0.5, interactive=interact,
       threshold='0.1mJy', niter=300)
```

This clears any previous models and calibrations from this ms before self-cal

Step 3: Make first continuum image for self-calibration

Print the Peak, RMS and S/N ratio and **note these down** to compare to later images we make during the self calibration process.

```
peak=imstat(imagename='HD163296_cont.clean.image',box='125,125,37
rms=imstat(imagename='HD163296_cont.clean.image',box='10,10,125,1
print('HD163296_cont.clean.image peak = %7.4f mJy/beam, rms  = %7
print('Make a note of these numbers to check for improvement afte
```

Calculate the minimum selfcal solution interval and set solintp1, solintp2

Noise on one baseline (over entire observation) is:

$$\sigma_{\rm baseline} = \sigma_{\rm array}\sqrt{N(N-1)/2}$$

Noise on one antenna (over entire observation) is:

$$\sigma_{\rm antenna} = \frac{\sigma_{\rm baseline}}{\sqrt{N-3}} \quad \textbf{=>} \quad \sigma_{\rm antenna} = \sigma_{\rm array}\sqrt{\frac{N(N-1)}{2(N-3)}}$$

Require S/N of 3 (where P is the peak emission on the longest baselines):

$$\sigma_{\rm antenna} \leq P/3$$

Calculate the minimum selfcal solution interval and set solintp1, solintp2

For the minimum solution time dt_{min} and one spw and polarization:

$$\sigma_{\mathrm{antenna}}(\mathrm{dt_{min}}) = \sigma_{\mathrm{array}}(\mathrm{t_{tot}})\sqrt{\frac{\mathrm{t_{tot}}}{\mathrm{dt_{min}}}}\sqrt{\frac{N(N-1)}{2(N-3)}}\sqrt{n_{\mathrm{pol}}n_{\mathrm{spw}}}$$

Rearranging for dt_{min} and including $\sigma_{\mathrm{antenna}} \leq P/3$:

$$\mathrm{dt_{min}} \geq \mathrm{t_{tot}}\left[\frac{\sigma_{\mathrm{array}}(\mathrm{t_{tot}})}{P/3}\right]^2 \frac{N(N-1)}{2(N-3)} n_{\mathrm{pol}}n_{\mathrm{spw}}$$

Step 4: First Phase self-calibration (p1)

```
ft(vis='HD163296_cont_all.ms',
   model='HD163296_cont.clean.model',
   usescratch=True)
```

This inserts the model image from the previous clean as the data model

```
os.system('rm HD163296_cont.clean.model.png')
plotms(vis='HD163296_cont_all.ms', xaxis='uvwave',
       yaxis='amp', ydatacolumn='model', showgui=interact,
       plotfile='HD163296_cont.clean.model.png')
```

Check model is in model column

Step 4: First Phase self-calibration (p1)

```
os.system('rm -rf HD163296_cont.p1')
gaincal(vis='HD163296_cont_all.ms',
        caltable='HD163296_cont.p1',
        solint='10000s',refant=antrefs,
        calmode='p', gaintype='G',
        combine='scan', minsnr=1.5,
        minblperant=3)
```

Long initial solution - all of observation!

Lower accepted SNR to increase no. of accepted solutions

Phase-only solution to begin with

Question: In general, if you don't have enough signal to noise in each solution, what can you do to improve this?

Step 4: First Phase self-calibration (p1)

Question: If you don’t have enough signal to noise in each solution, what can you do to improve this?

Answer: Combine correlations (gaintype=‘T’ after first iteration), scans (combine=‘scan’) or spws (combine=‘spw’), or increase solint

Step 4: Plot p1 gaincal solutions

```
plotms(vis='HD163296_cont.p1',xaxis='time', yaxis='phase',
        coloraxis=’spw’,gridrows=3, gridcols=3,
        iteraxis='antenna',plotrange=[0,0,-180,180.])
```

Plot results in 3 x 3 grid

Step 5: Apply solutions and reimage (p1)

```
applycal(vis='HD163296_cont_all.ms',
         gaintable='HD163296_cont.p1',
         calwt=False, applymode='calonly')

os.system('rm -rf HD163296_cont.p1.clean*')
tclean(vis='HD163296_cont_all.ms',
         imagename='HD163296_cont.p1.clean',
         specmode='mfs', deconvolver='multiscale',
         imsize=500, spw='0~3', scales=[0, 5, 10],
         cell=cellsize, weighting='briggs',
         robust=-0.5, interactive=interact,
         threshold='0.1mJy', niter=500,
         mask='HD163296_cont.clean.mask')
```

Do not flag data without solutions

Do not calibrate the weights

Slightly more iterations than before

Step 5: Apply solutions and reimage (p1)

The Peak, RMS and S/N ratio are printed to screen
Note these down and compare to previous values

```
peak=imstat(imagename='HD163296_cont.p1.clean.image',box='125
rms=imstat(imagename='HD163296_cont.p1.clean.image',box='10,1
print('HD163296_cont.p1.clean.image peak = %7.4f mJy/beam, rm
print('There should be an improvement')
```

Step 6: Second Phase self-calibration (p2)

```
ft(vis='HD163296_cont_all.ms',
   model='HD163296_cont.p1.clean.model',
   usescratch=True)
```

Insert the model image from the previous clean as the data model

```
os.system('rm HD163296_cont.p1.clean.model.png')
plotms(vis='HD163296_cont_all.ms', xaxis='uvwave', yaxis='amp',
       ydatacolumn='model', showgui=interact,
       plotfile='HD163296_cont.p1.clean.model.png')
```

Check model is in model column

Step 6: Second Phase self-calibration (p2)

```
os.system('rm -rf HD163296_cont.p2')
gaincal(vis='HD163296_cont_all.ms',
        caltable='HD163296_cont.p2',
        gaintable=['HD163296_cont.p1'],
        solint='1088s',
        refant=antrefs,
        calmode='p',gaintype='T',
        combine='scan',minsnr=1.5,
        minblperant=3)

plotms(vis='HD163296_cont.p2',xaxis='time', yaxis='phase',
       coloraxis='spw', gridrows=3, gridcols=3,
       iteraxis='antenna', plotrange=[0,0,-180,180.])
```

Derive solutions using new model

Use shorter solution interval

Plot solutions

Step 7: Apply solutions and reimage (p2)

```
applycal(vis='HD163296_cont_all.ms',
         gaintable=['HD163296_cont.p1', 'HD163296_cont.p2'],
         calwt=False, applymode='calonly')

os.system('rm -rf HD163296_cont.p2.clean*')
tclean(vis='HD163296_cont_all.ms',
       imagename='HD163296_cont.p2.clean',
       specmode='mfs', deconvolver='multiscale',
       imsize=500, spw='0~3', scales=[0, 5, 10],
       cell=cellsize, weighting='briggs',
       robust=-0.5, interactive=interact,
       threshold='0.1mJy', niter=1000,
       mask='HD163296_cont.clean.mask')
```

Slightly more iterations than before

Step 7: Apply solutions and reimage (p2)

The Peak, RMS and S/N ratio are printed to screen
Note these down and compare to previous values

```
peak=imstat(imagename='HD163296_cont.p2.clean.image',box='125,
rms=imstat(imagename='HD163296_cont.p2.clean.image',box='10,10
print('HD163296_cont.p2.clean.image peak = %7.4f mJy/beam, rms
print('More improvement?')
```

Step 8: Third Phase self-calibration (p3)

```
ft(vis='HD163296_cont_all.ms',
   model='HD163296_cont.p2.clean.model',
   usescratch=True)

os.system('rm HD163296_cont.p2.clean.model.png')
plotms(vis='HD163296_cont_all.ms', xaxis='uvwave', yaxis='amp',
       ydatacolumn='model', showgui=interact,
       plotfile='HD163296_cont.p2.clean.model.png')
```

Check model is in model column

Step 8: Thirds Phase self-calibration (p3)

```
os.system('rm -rf HD163296_cont.p3')
gaincal(vis='HD163296_cont_all.ms',
        caltable='HD163296_cont.p3',
        gaintable=['HD163296_cont.p1','HD163296_cont.p2'],
        solint='544s',refant=antrefs,
        calmode='p',gaintype='T',
        combine='scan', minsnr=1.5,
        minblperant=3)

plotms(vis='HD163296_cont.p3',xaxis='time', yaxis='phase',
       coloraxis='spw', gridrows=3, gridcols=3,
       iteraxis='antenna', plotrange=[0,0,-180,180.])
```

Derive solutions using new model

Use shorter solution interval

Plot solutions

Step 9: Apply solutions and reimage (p3)

```
applycal(vis='HD163296_cont_all.ms',
         gaintable=['HD163296_cont.p1','HD163296_cont.p2','HD163296_cont.p3'],
         calwt=False, applymode='calonly')

os.system('rm -rf HD163296_cont.p3.clean*')
tclean(vis='HD163296_cont_all.ms',
       imagename='HD163296_cont.p3.clean',
       specmode='mfs', deconvolver='multiscale',
       imsize=500, spw='0~3', scales=[0, 5, 10],
       cell=cellsize, weighting='briggs',
       robust=-0.5, interactive=interact,
       threshold='0.1mJy', niter=2000,
       mask='HD163296_cont.clean.mask')
```

More iterations than before

Step 9: Apply solutions and reimage (p3)

The Peak, RMS and S/N ratio are printed to screen
Note these down and compare to previous values

```
peak=imstat(imagename='HD163296_cont.p3.clean.image',box='125
rms=imstat(imagename='HD163296_cont.p3.clean.image',box='10,1
print('HD163296_cont.p3.clean.image peak = %7.4f mJy/beam, rm
print('More improvement?')
```

Step 10: chose solinta for Amplitude self-calibration

Set `solinta`, the solution interval for the amplitude solutions.

Amplitude changes more slowly and is less constrained than phase, so need to pick a longer solution interval for amplitude, e.g. 1088s.

Step 10: Amplitude self-calibration (a1)

```
ft(vis='HD163296_cont_all.ms',
   model='HD163296_cont.p3.clean.model',
   usescratch=True)

os.system('rm HD163296_cont.p3.clean.model.png')
plotms(vis='HD163296_cont_all.ms', xaxis='uvwave',
       yaxis='amp', ydatacolumn='model', showgui=interact,
       plotfile='HD163296_cont.p3.clean.model.png')
```

Step 10: Amplitude self-calibration (a1)

```
os.system('rm -rf HD163296_cont.a1')
gaincal(vis='HD163296_cont_all.ms',
        caltable='HD163296_cont.a1',
        gaintable=['HD163296_cont.p1','HD163296_cont.p2','HD163296_cont.p3'],
        solint=solinta, refant=antrefs,
        calmode='a', gaintype='T',
        combine='scan',
        minsnr=1.5, minblperant=4)
```

Apply all phase solutions on the fly

```
plotms(vis='HD163296_cont.a1',xaxis='time', yaxis='amplitude',
       gridrows=3, gridcols=3, iteraxis='antenna',coloraxis='spw')
```

Step 11: Apply phase and amp. solutions and image!

```
applycal(vis='HD163296_cont_all.ms',
         gaintable=['HD163296_cont.p1','HD163296_cont.p2',
         'HD163296_cont.p3', 'HD163296_cont.a1'],
         calwt=False, applymode='calonly')
```

Apply the phase and amplitude solutions

```
os.system('rm -rf HD163296_cont.p3a1.clean*')
tclean(vis='HD163296_cont_all.ms',
       imagename='HD163296_cont.p3a1.clean',
       specmode='mfs', deconvolver='multiscale',
       imsize=500, spw='0~3', scales=[0, 5, 10],
       cell=cellsize, weighting='briggs',
       robust=-0.5, interactive=interact,
       threshold='0.1mJy', niter=3000,
       mask='HD163296_cont.clean.mask')
```

Step 11: Apply solutions make final image

The Peak, RMS and S/N ratio are printed to screen
Note these down and compare to previous values

```
peak=imstat(imagename='HD163296_cont.p3a1.clean.image',box=
rms=imstat(imagename='HD163296_cont.p3a1.clean.image',box='
print('HD163296_cont.p3a1.clean.image peak = %7.4f mJy/beam
print('More improvement?')
```

Question: Did we reach the theoretical noise we found from the sensitivity calculator?

Step 11: Apply solutions make final image

Print the Peak, RMS and S/N ratio and **note these down** and compare to previous values.

```
peak=imstat(imagename='HD163296_cont.p3a1.clean.image',box='
rms=imstat(imagename='HD163296_cont.p3a1.clean.image',box='1
print('HD163296_cont.p3a1.clean.image peak = %7.4f mJy/beam,
print('More improvement?')
```

Question: Did we reach the theoretical noise we found from the sensitivity calculator?

Answer: No (actual is higher). This is likely due to residual sources of non-closing (not antenna based) errors in the data like baseline errors and baseline-dependent bandpass errors. Additionally, polarization errors (which we did not calibrate at all) can contribute. If there is sparse uv coverage (not really an issue here), this can also cause problems.

Step 12: Apply calibration to line data

```
applycal(vis='HD163296_CO_new.ms',
         gaintable=['HD163296_cont.p1','HD163296_cont.p2',
         'HD163296_cont.p3', 'HD163296_cont.a1'],
         calwt=False, applymode='calonly')
```

Line (full spec. res.) dataset

Plot calibrated line data (to determine line-free channels):

```
plotms(vis='HD163296_CO_new.ms',
       xaxis='channel', yaxis='amp',
       ydatacolumn='corrected',
       avgtime='99999', avgscan=True,
       coloraxis='corr')
```

Set the variable continum_fitspw using the plot above.

Step 13: Subtract continuum

```
os.system('rm -rf HD163296_CO_new.ms.contsub')

uvcontsub(vis='HD163296_CO_new.ms',
          spw='0', fitspw=continum_fitspw,
          excludechans=False, solint='int',
          fitorder=0, want_cont=False)
```

Select the line free channels to fit the continuum

e.g. fitspw='0:0~100;150~200' will exclude channels 101-149
(this is an example - not the correct answer!)

Set the rest frequencies and line rms threshold

```
co_21_restfreq = '230.538GHz'

linethresh = '***.mJy'
```
(set to 3 x line rms per channel)

You estimated this from the sensitivity calculator

```
chanstart= '***km/s'
chanwidth= '***km/s'
nchan=   '***'
```

Set channel parameters and velocity range

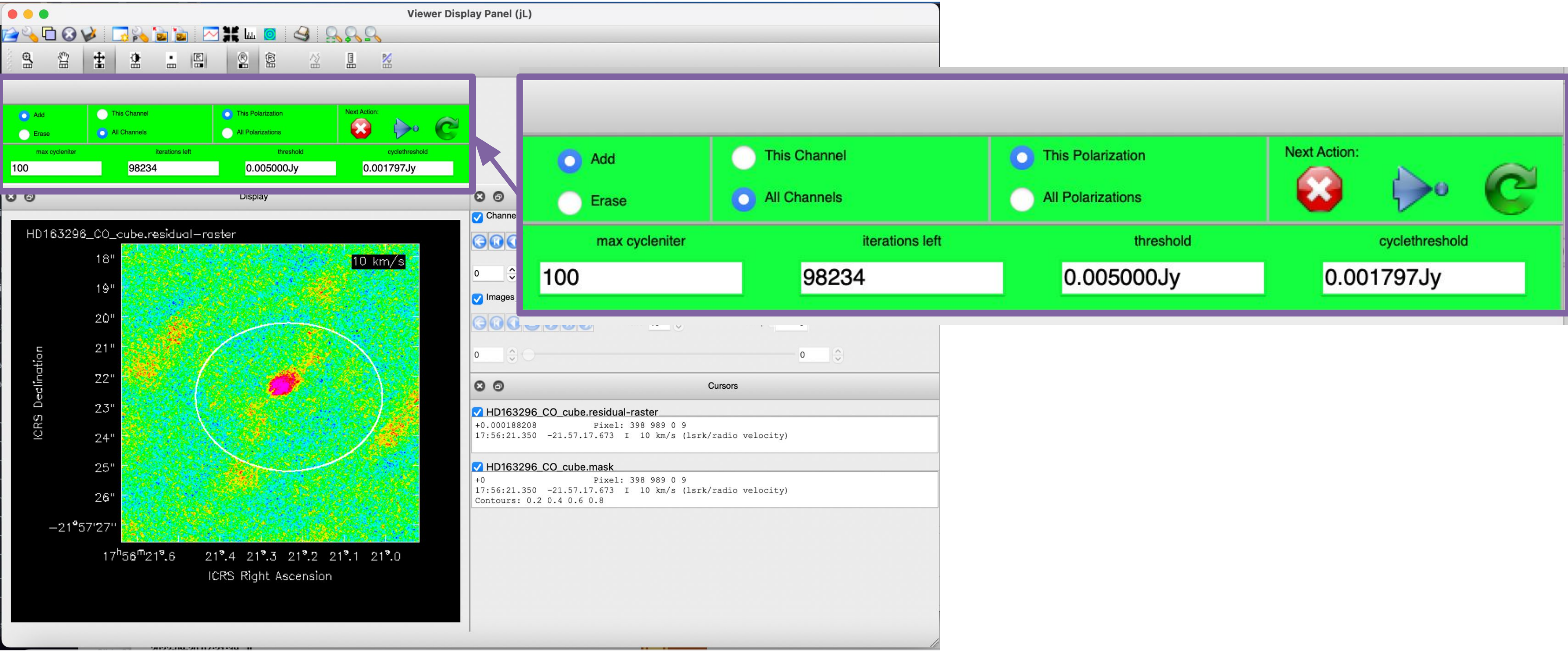
Step 14: image CO(2-1)

```
os.system('rm -rf HD163296_CO_cube*')

tclean(vis='HD163296_CO_new.ms.contsub',
       imagename='HD163296_CO_cube',
       specmode='cube', spw='0',
       imsize=1000, cell='0.01arcsec',
       deconvolver='hogbom',
       start=chanstart, width=chanwidth, nchan=nchan,
       outframe='LSRK', veltype='radio',
       restfreq=co_21_restfreq,
       weighting='briggs',robust=0.5,threshold=linethresh,
       niter=100000, cycleniter=500,
       restoringbeam='common', pbcor=True,
       interactive=interact)
```

Enough channels to cover the line

Max. no. of iterations per minor cycle

Step 14: image CO(2-1)



Step 14: image CO(2-1)

To do: determine the restoring synthesized beam sizes for the two images using the task **imhead**, e.g.

```
imhead('HD163296_CO_auto.image')
```

```
hdr=imhead('HD163296_CO_auto.image')
```

Step 15: Clean CO(2-1) using auto-masking

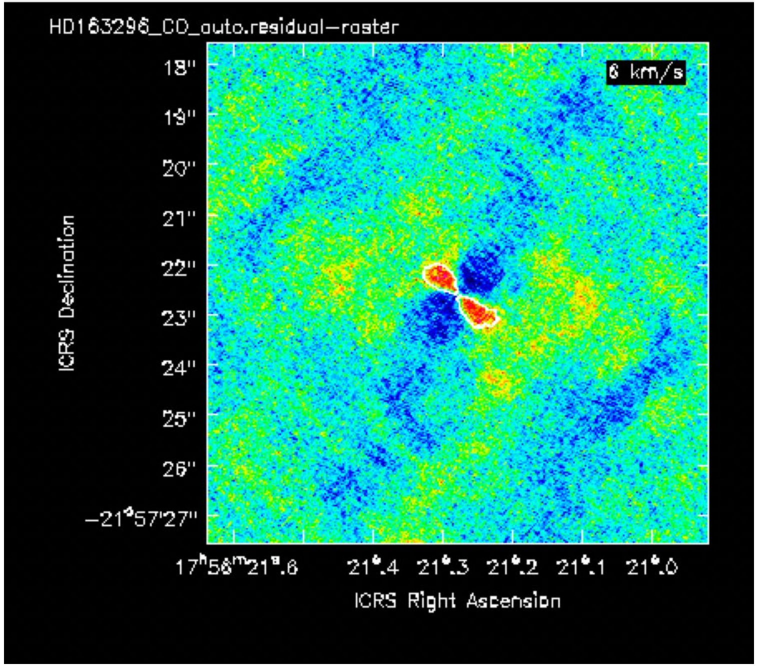
```
os.system('rm -rf HD163296_CO_auto*')
tclean(vis='HD163296_CO_new.ms.contsub',
       imagename='HD163296_CO_auto',
       specmode='cube', spw='0', imsize=1000, cell='0.01arcsec',
       deconvolver='hogbom',
       start=chanstart, width=chanwidth, nchan=nchan,
       outframe='LSRK', veltype='radio', restfreq=co_21_restfreq,
       niter=100000, cycleniter=500,
       weighting='briggs',robust=0.5,threshold=linethresh,
       usemask='auto-multithresh',
       noisethreshold=3.3, sidelobethreshold=1.8,
       growiterations=400, lownoisethreshold=1.2,
       minbeamfrac = 0.5,
       restoringbeam='common', pbcor=True,
       interactive=interact)
```

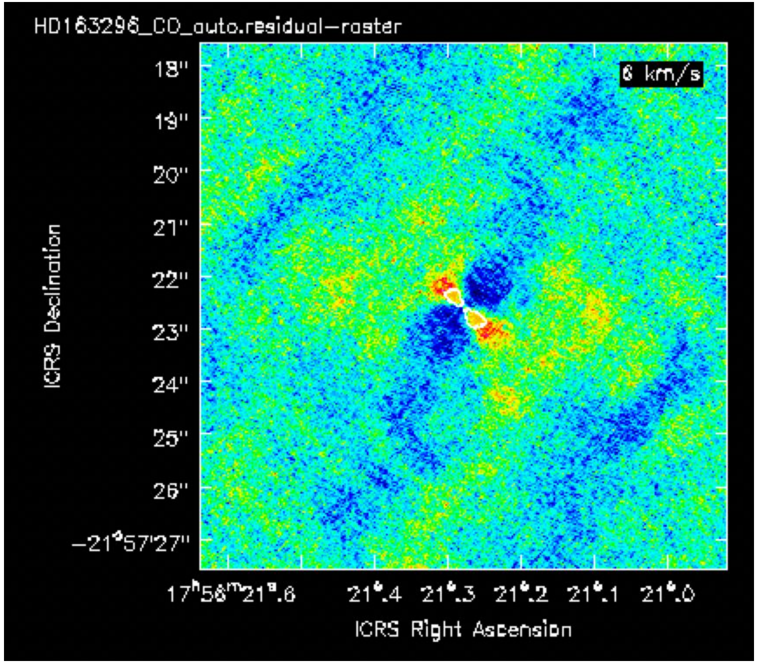
These parameters control automasking

For more info, see https://casaguides.nrao.edu/index.php/Automasking_Guide

Step 15: Clean CO(2-1) using auto-masking

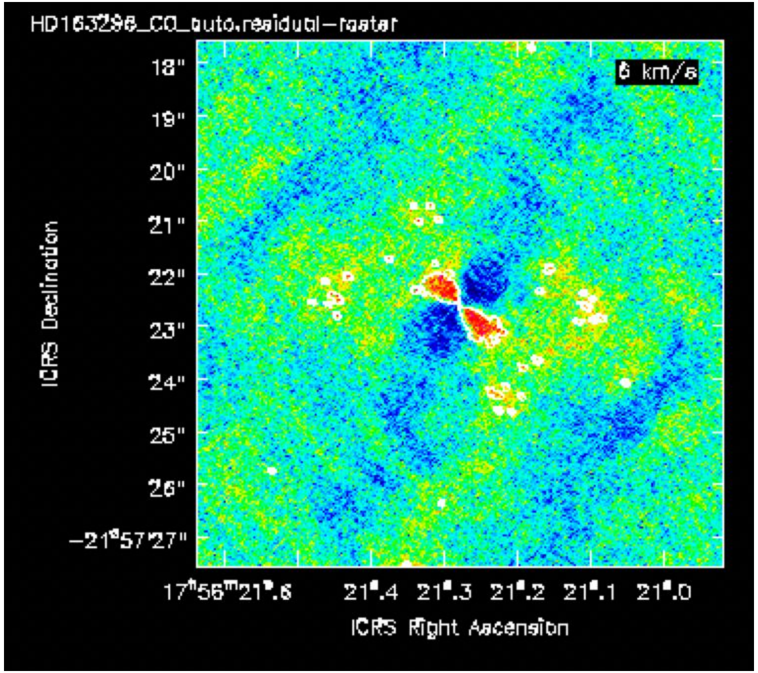
`Noisethreshold`, `sidelobethreshold` & `lownoisethreshold`: Set the noise threshold to enter into the mask

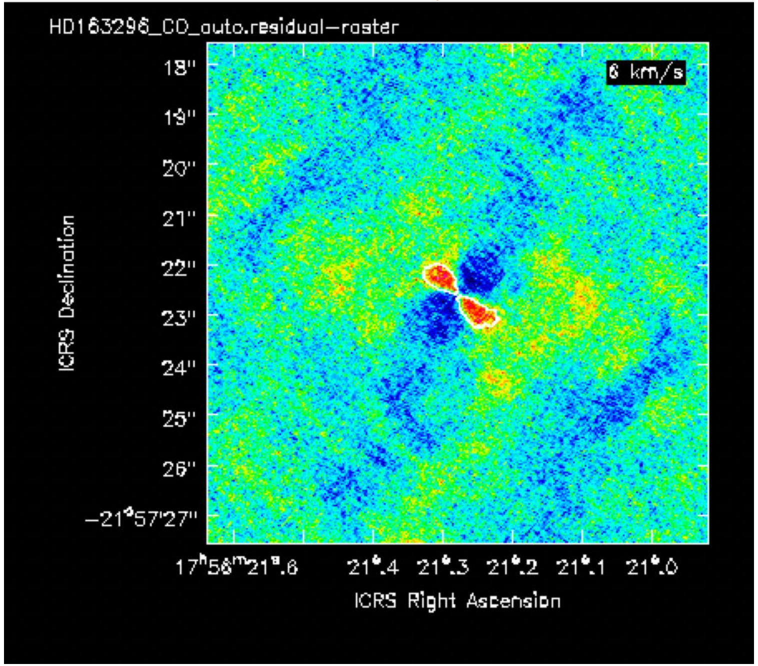




Step 15: Clean CO(2-1) using auto-masking

minbeamfrac: Controls the size of mask regions to prune. A bigger number sets a higher size threshold for acceptance





Step 15: Clean CO(2-1) using auto-masking

`noisethreshold=3.3`

This parameter sets initial mask based on noisethreshold x rms in residual image
The algorithm chooses the larger of the threshold produced by sidelobethreshold and noisethreshold.

`sidelobethreshold=1.8`

This parameter sets a threshold for making initial mask based on:
sidelobethreshold x fractional sidelobe level x peak in residual image

`lownoisethreshold=1.2`

This parameter sets threshold for low S/N expanded mask based on:
lownoisethreshold x rms in residual image
The algorithm chooses the larger of the threshold produced by sidelobethreshold and lownoisethreshold.

`growiterations=400`

The maximum number of iterations to spend expanding the mask into low-S/N regions

`minbeamfrac = 0.5,`

Controls the size of mask regions to prune. A bigger number sets a higher size threshold to be accepted

Step 15: Determine RMS noise

RMS noise can be determined using the task imstat for line-free channels, e.g.

```
results = imstat('HD163296_CO_auto.image', chans='1')
print (results)
co_image_sigma = results['sigma'][0]
print ("s.d. ", results['sigma'])
print ("RMS ", results['rms'])
```

Empty channel

This will be used for later analysis steps.

Step 15: Determining spectral extent

First estimate the spectral extent of the CO emission using the viewer (will be used in later steps):

```
imview('HD163296_CO_auto.image')
```

Then set this range in the parameter `moment_chans`:

```
moment_chans = '***~***'
```

Hints to determine `moment_chans`:

- Open the same image as a contour map in the same viewer
- Determine the range of channels which have flux > 5 sigma

Image analysis

- Step 16: Saving a spectrum to file and line fitting in CASA
- Step 17 & 18: Making moment maps
- Step 19: Primary beam correction
- Step 20: Fitting 2-D gaussians to emission
- Steps 21 and 22: Making PV diagrams
- Step 23: Reprojecting an image

Step 16: Save line spectrum to file using spectral profile tool

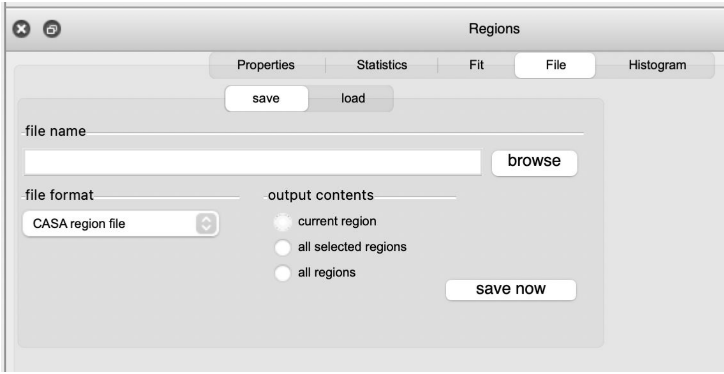
- Open line images in viewer, e.g.
  `imview('HD163296_CO_auto.image')`

- Draw a mask over a region with emission

- Use the Spectral Profile Tool (icon that looks like window with red line in it) to make a spectrum

- Save the spectrum to file (for creating a figure using your favourite software, e.g. python + matplotlib)

- Save region file, using the file tab in the regions box



Step 16: Line fitting in CASA

Can use specfit

```
spec_model = specfit(‘HD163296_CO_auto.image’,
           region='spectrum_region.crtf', poly=-1,
           ngauss = 1, logresults=True)
```

Note: you’ll need to save a region in the viewer first

Steps 17 & 18: Moment maps of line emission

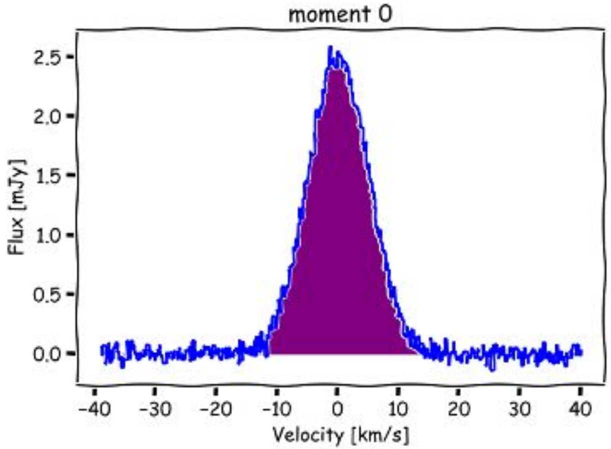
- **-1**: the mean value of the spectrum
- **0:** the integrated value of the spectrum
- **1:** the intensity weighted coordinate – used for velocity fields
- **2:** the intensity weighted dispersion of the coordinate – used for velocity dispersion fields
- **3:** the median of the spectrum
- **4:** the median velocity
- **5:** the standard deviation about the mean of the spectrum
- **6:** the root mean square of the spectrum
- **7:** the absolute mean deviation of the spectrum
- **8:** the maximum value of the spectrum
- **9:** the coordinate of the maximum value of the spectrum
- **10:** the minimum value of the spectrum
- **11:** the coordinate of the minimum value of the spectrum

Steps 17 & 18: Moment maps of line emission

- **-1**: the mean value of the spectrum
- **0:** the integrated value of the spectrum
- **1:** the intensity weighted coordinate – used for velocity fields
- **2:** the intensity weighted dispersion of the coordinate – used for velocity dispersion fields
- **3:** the median of the spectrum
- **4:** the median velocity
- **5:** the standard deviation about the mean of the spectrum
- **6:** the root mean square of the spectrum
- **7:** the absolute mean deviation of the spectrum
- **8:** the maximum value of the spectrum
- **9:** the coordinate of the maximum value of the spectrum
- **10:** the minimum value of the spectrum
- **11:** the coordinate of the minimum value of the spectrum

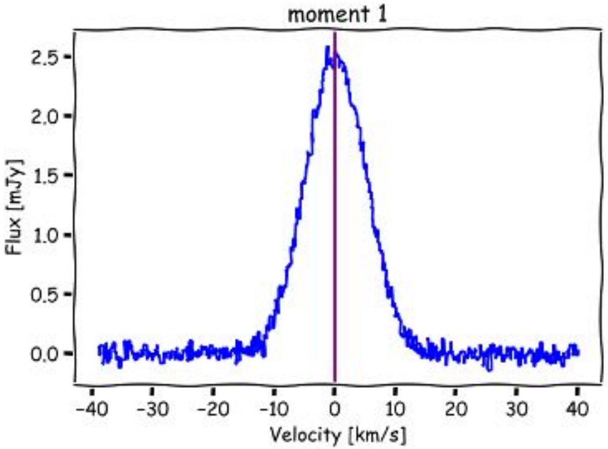
Steps 17 & 18: Moment maps of line emission

Integrated intensity



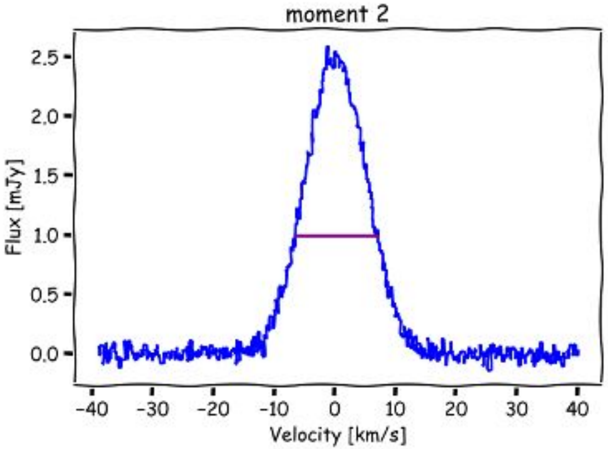
$$M_0 = \Delta v \sum I(v)$$

Velocity



$$M_1 = \frac{\sum v\, I(v)}{\sum I(v)}$$

Velocity dispersion



$$M_2 = \sqrt{\frac{\sum (v - M_1)^2\, I(v)}{\sum I(v)}}$$

Steps 17 & 18: Moment maps of line emission

Zero moment map = integrated flux map

First moment map = intensity-weighted velocity

Second moment map = intensity-weighted velocity dispersion about the mean

These are made using the task **immoments**

Step 17: zeroth moment map of line emission

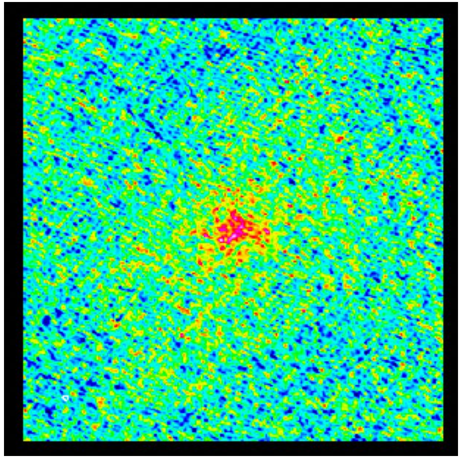
To do: Make zero moment maps for both lines using task immoments, e.g.

```
os.system('rm -rf HD163296_CO_auto.image.mom0')
immoments(imagename='HD163296_CO_auto.image',
          outfile='HD163296_CO_auto.image.mom0',
          moments=[0], chans=moment_chans)
```

Range of emission determined previously (channels that have flux > 5 sigma)

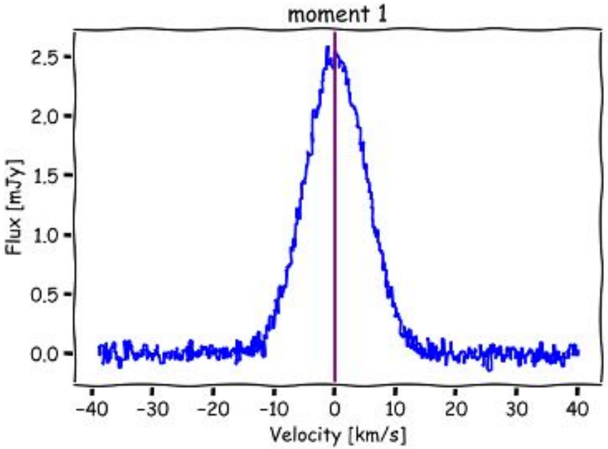
Steps 17 & 18: Moment maps of line emission

Integrated intensity



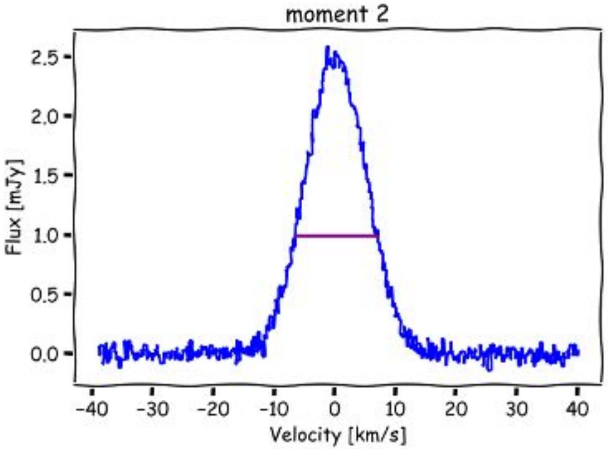
$$M_0 = \Delta v \sum I(v)$$

Velocity



$$M_1 = \frac{\sum v\, I(v)}{\sum I(v)}$$

Velocity dispersion



$$M_2 = \sqrt{\frac{\sum (v - M_1)^2\, I(v)}{\sum I(v)}}$$

Step 17: Viewing the moment map using task imview

```
imview(raster=[ {'file':'HD163296_CO_auto.image.mom0'}],
       contour={'file':'HD163296_cont.p3a1.clean.image',
                'base':0, 'unit':***,
                'levels':[3,5,10,50]})
```

Your rms noise for the continuum

Step 18: Making the first and second moment maps

First moment:

```
os.system('rm -rf HD163296_CO_auto.image.mom1')
immoments(imagename='HD163296_CO_auto.image', moments=[1],
          outfile='HD163296_CO_auto.image.mom1',
          chans=moment_chans, includepix=[4.0 * co_image_sigma, ***])
```

Set a high number to include all flux above the noise threshold

Second moment:

```
os.system('rm -rf HD163296_CO_auto.image.mom2')
immoments(imagename='HD163296_CO_auto.image', moments=[2],
           outfile='HD163296_CO_auto.image.mom2',
          chans=moment_chans, includepix=[4.0 * co_image_sigma, ***])
```

Channels with emission

4 - 5 x RMS noise in empty channel
(if there are image artifacts, may have to be higher)

Step 18: Viewing and exporting the moment maps

```
imview( raster=[ {'file':'HD163296_CO_auto.image.mom0'},
                 {'file':'HD163296_CO_auto.image.mom1'},
                 {'file':'HD163296_CO_auto.image.mom2'} ],
        contour={'file':'HD163296_cont.p3a1.clean.image',
                 'base':0, 'unit':***,
                 'levels':[3,5,10,50]} )
```

Your rms noise for the continuum

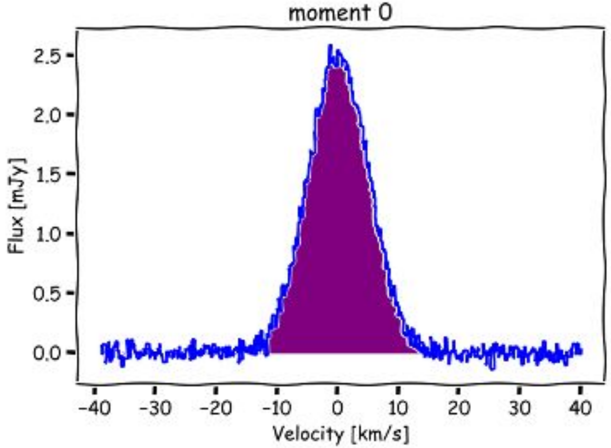
To do: Export your images using task exportfits, e.g.

```
os.system('rm -rf HD163296_CO_auto.image.fits')
exportfits(imagename='HD163296_CO_auto.image',
           fitsimage='HD163296_CO_auto.image.fits')
```

Extras: Check what parameters **velocity=True** and **dropstokes=True** do

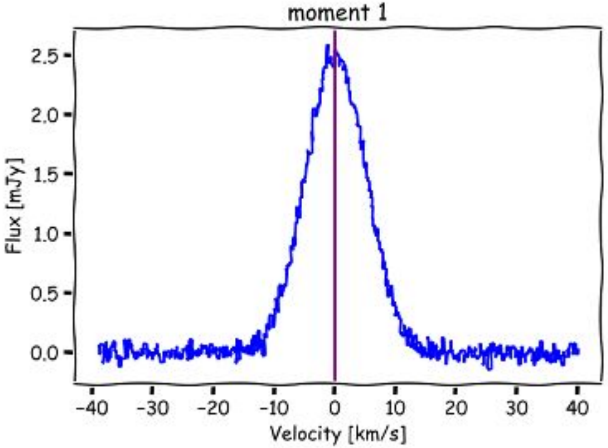
Steps 17 & 18: Moment maps of line emission

Integrated intensity



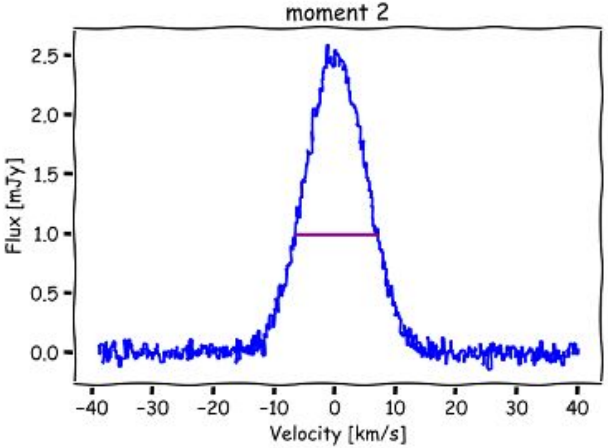
$$M_0 = \Delta v \sum I(v)$$

Velocity



$$M_1 = \frac{\sum v\, I(v)}{\sum I(v)}$$

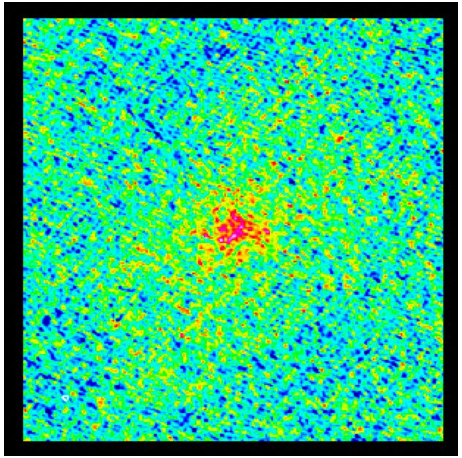
Velocity dispersion



$$M_2 = \sqrt{\frac{\sum (v - M_1)^2\, I(v)}{\sum I(v)}}$$

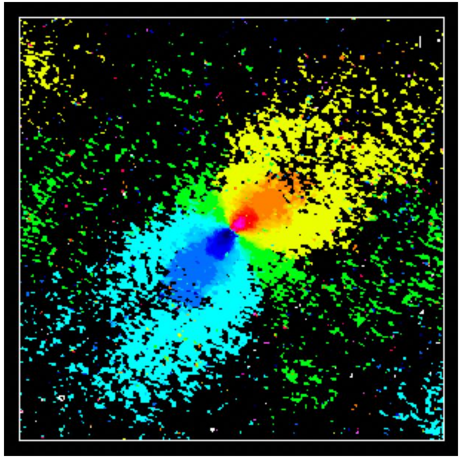
Steps 17 & 18: Moment maps of line emission

Integrated intensity



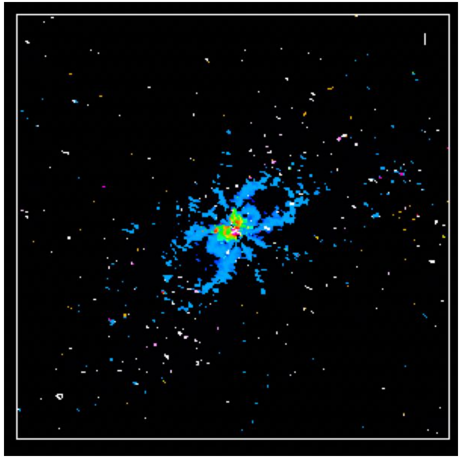
$$M_0 = \Delta v \sum I(v)$$

Velocity



$$M_1 = \frac{\sum v\, I(v)}{\sum I(v)}$$

Velocity dispersion



$$M_2 = \sqrt{\frac{\sum (v - M_1)^2\, I(v)}{\sum I(v)}}$$

Step 19: Primary beam corrections

- Without correction for the primary beam response (default), images should have roughly constant noise across them…
- …but the flux is incorrect everywhere except the field centre
- To measure fluxes in your images, make sure to correct for the primary beam response first!

Step 19: Primary beam corrections

You can use the task impbcor:

```
impbcor(imagename='HD163296_cont.p3a1.clean.image',
        pbimage='HD163296_cont.p3a1.clean.pb',
        mode='divide',
       outfile='HD163296_cont.p3a1.clean.image.pbcor')
```

Step 20: Fitting a gaussian to the continuum using imfit

Fit the continuum emission with a 2D gaussian:

```
imfit(imagename="HD163296_cont.p3a1.clean.image",
    region="imfit_region.crtf", logfile="contin_fit.log",
    model="HD163296_cont.p3a1.clean.image.imfit",
    residual="HD163296_cont.p3a1.clean.image.fitresid")
```

To do:

- Check the residual image to see if the fit was good…
- Look at the log file and determine the integrated flux and deconvolved size

Step 21: Making position-velocity diagrams in the viewer

- Open one of the image cubes in the viewer
- Click on the P/V tool button 
- Draw a slice across the source (blue to red shifted)
- Go to menu => view => Regions => pV tab
- Click “Generate P/V”
- Change the averaging width and generate again
- Save the image
- (Note down the length and position angle!)

Step 21-22: Making position-velocity diagrams using task impv

- Set `pv_centre` in pixel coordinates by inspecting continuum or moment maps or using results from imfit to continuum
- Set `pv_length` and `pv_pa` using the length and position angle determined in Step 21 above

Step 22: Making position-velocity diagrams using task impv

Can also generate pv diagrams using the task impv, e.g.

```
os.system('rm -rf HD163296_CO_auto.image.pv')
impv(imagename='HD163296_CO_auto.image',
     outfile='HD163296_CO_auto.image.pv',
     mode='length', center=pv_centre,
     length=pv_length,pa=pv_pa)
```

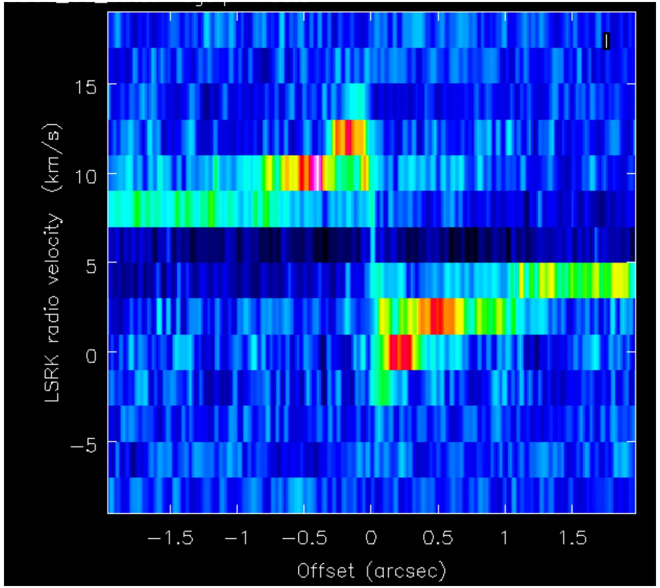
Centre of source [xpix, ypix] (Hint: find from inspection or imfit to continuum or moment maps)

Cover the whole source (can use results from step 21)

Position angle determined from step 21 (could also fit mom0 or look at mom1 velocity structure)

Step 21 & 22: Making position-velocity diagrams in viewer and using task impv

The resulting PV diagrams should look something like this:



Velocity (km/s)

Offset

Step 23: Reprojecting an image using task imregrid

For example, to reproject to Galactic coordinates:

```
imregrid(imagename='HD163296_CO_auto.image',
         template='GALACTIC',
         output='HD163296_CO_auto.image.galactic')
```

Or to reproject to another image header - useful for matching images from different telescopes (only an example!):

```
regrid_dict = imregrid(imagename="target.image",
                       template="get")
imregrid(imagename="input.image",
        output="output.image",
        template=regrid_dict)
```

parameters in blue are not real images, just example entries

More analysis tasks…

Can be found by typing “tasklist” in CASA:

```
Analysis
--------------------
imcollapse
imcontsub
imdev
imfit
imhead
imhistory
immath
immoments
impbcor
```

And many more!

CASA documentation

https://casadocs.readthedocs.io/en/stable

» API » casatasks » impv

Edit on GitHub

impv

impv(*imagename, outfile='', mode='coords', start='', end='', center='', length='', pa='', width=1, unit='arcsec', overwrite=False, region='', chans='', stokes='', mask='', stretch=False*) [source]

Construct a position-velocity image by choosing two points in the direction plane.

[Description] [Examples] [Development] [Details]

Parameters

- imagename (path) - Name of the input image
- outfile (string='') - Output image name. If empty, no image is written.

  ▸ *outfile != ''*

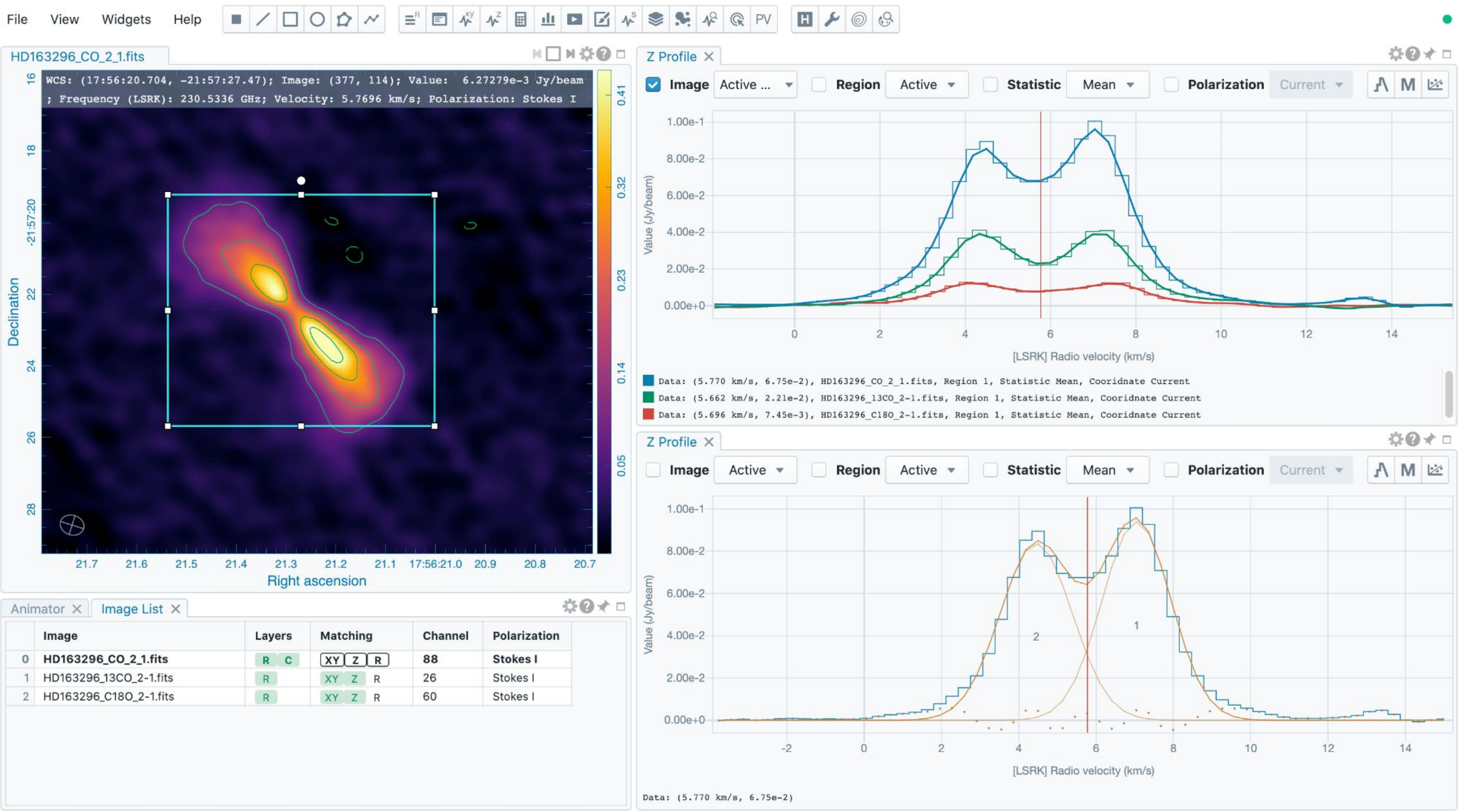
CASA documentation

https://casadocs.readthedocs.io/en/stable

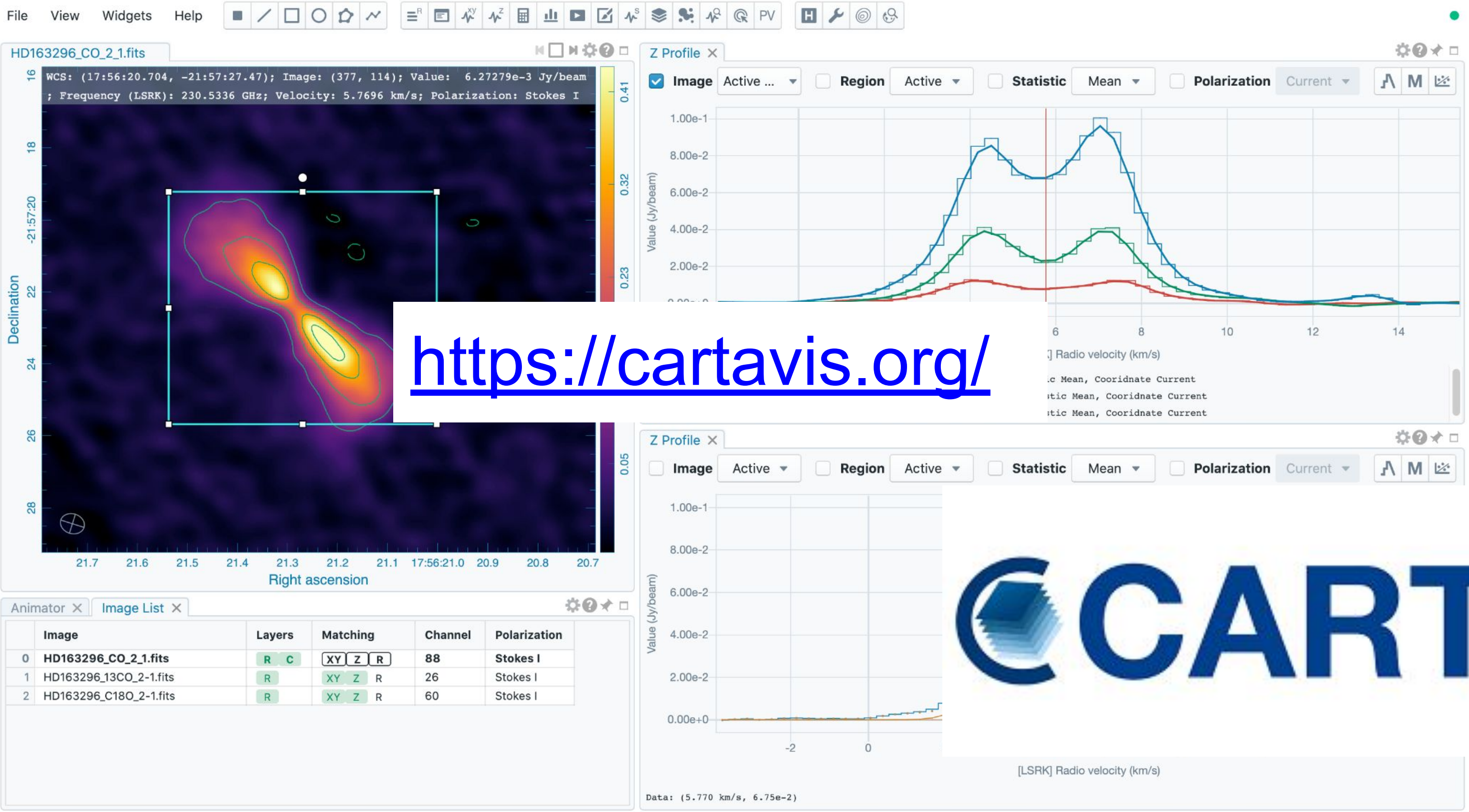
Examples

```
# create a pv image with the position axis running from ra,
# dec pixel positions of [45, 50] to [100, 120] in the input image
impv(imagename="my_spectral_cube.im", outfile="mypv.im", start=[45,50], end=[100,120])

# analyze the pv image, such as get statistics
pvstats = imstat("mypv.im")

# get the alternate coordinate system information
tb.open("mypv.im")
alternate_csys_record =
tb.getkeyword("misc")["secondary_coordinates"]
tb.done()
```

CARTA


File
View
Widgets
Help
PV
HD163296_CO_2_1.fits
WCS: (17:56:20.704, -21:57:27.47); Image: (377, 114); Value: 6.27279e-3 Jy/beam
; Frequency (LSRK): 230.5336 GHz; Velocity: 5.7696 km/s; Polarization: Stokes I
Declination
Right ascension
Z Profile
Image
Active
Region
Statistic
Mean
Polarization
Current
Value (Jy/beam)
[LSRK] Radio velocity (km/s)
Data: (5.770 km/s, 6.75e-2), HD163296_CO_2_1.fits, Region 1, Statistic Mean, Cooridnate Current
Data: (5.662 km/s, 2.21e-2), HD163296_13CO_2-1.fits, Region 1, Statistic Mean, Cooridnate Current
Data: (5.696 km/s, 7.45e-3), HD163296_C18O_2-1.fits, Region 1, Statistic Mean, Cooridnate Current
Data: (5.770 km/s, 6.75e-2)
Animator
Image List
Image
Layers
Matching
Channel
Polarization
0 HD163296_CO_2_1.fits R C XY Z R 88 Stokes I
1 HD163296_13CO_2-1.fits R XY Z R 26 Stokes I
2 HD163296_C18O_2-1.fits R XY Z R 60 Stokes I

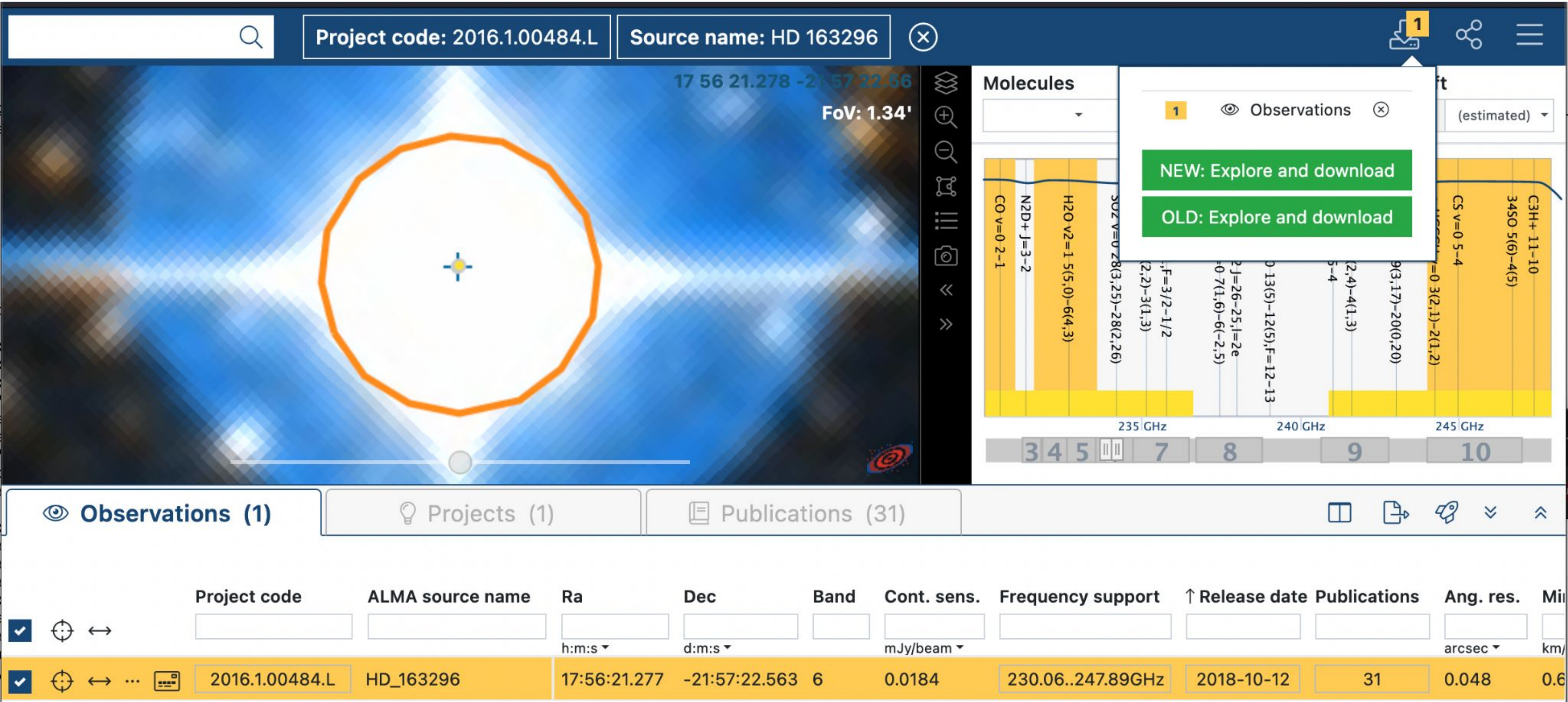
CARTA



https://cartavis.org/

CARTA

ALMA Archive



ALMA Archive

https://almascience.eso.org/aq/

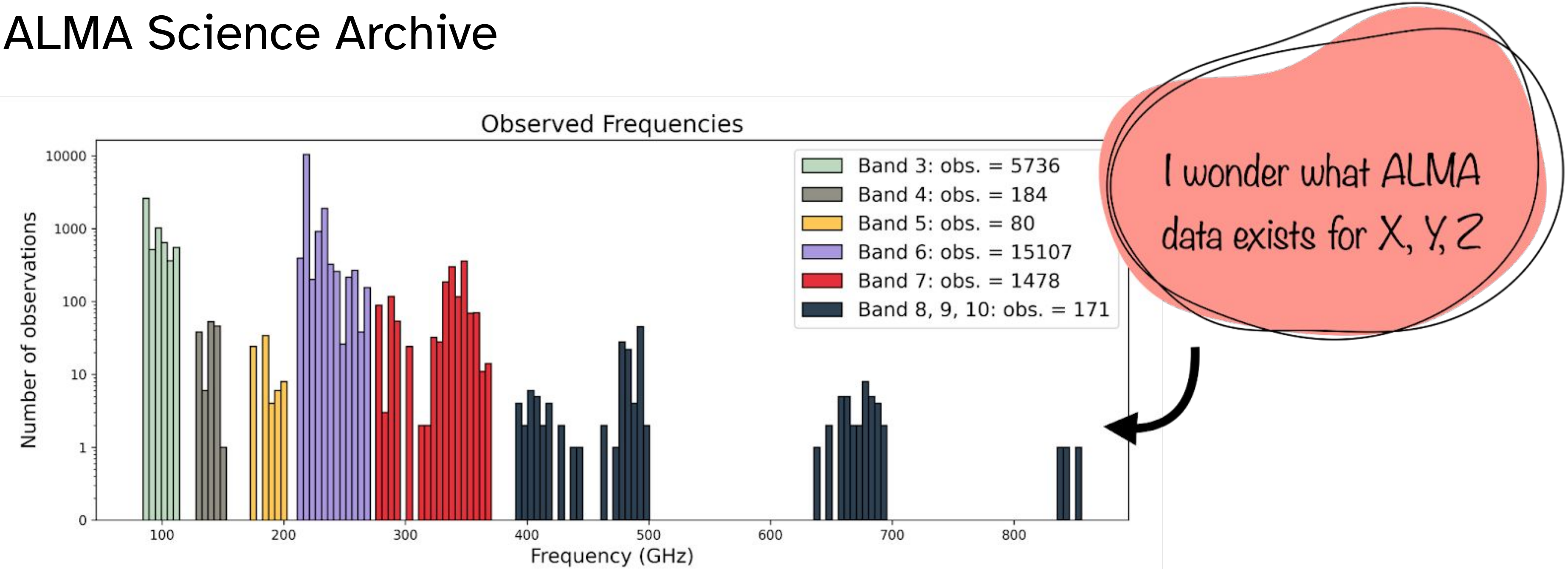
ALminer

Python-based code to effectively **query**, **analyse**, and **visualise** the ALMA Science Archive

Bonus: **download** ALMA data products and/or raw data

ALminer

Python-based code to effectively **query**, **analyse**, and **visualise** the ALMA Science Archive



<u>Bonus:</u> **download** ALMA data products and/or raw data

ALminer

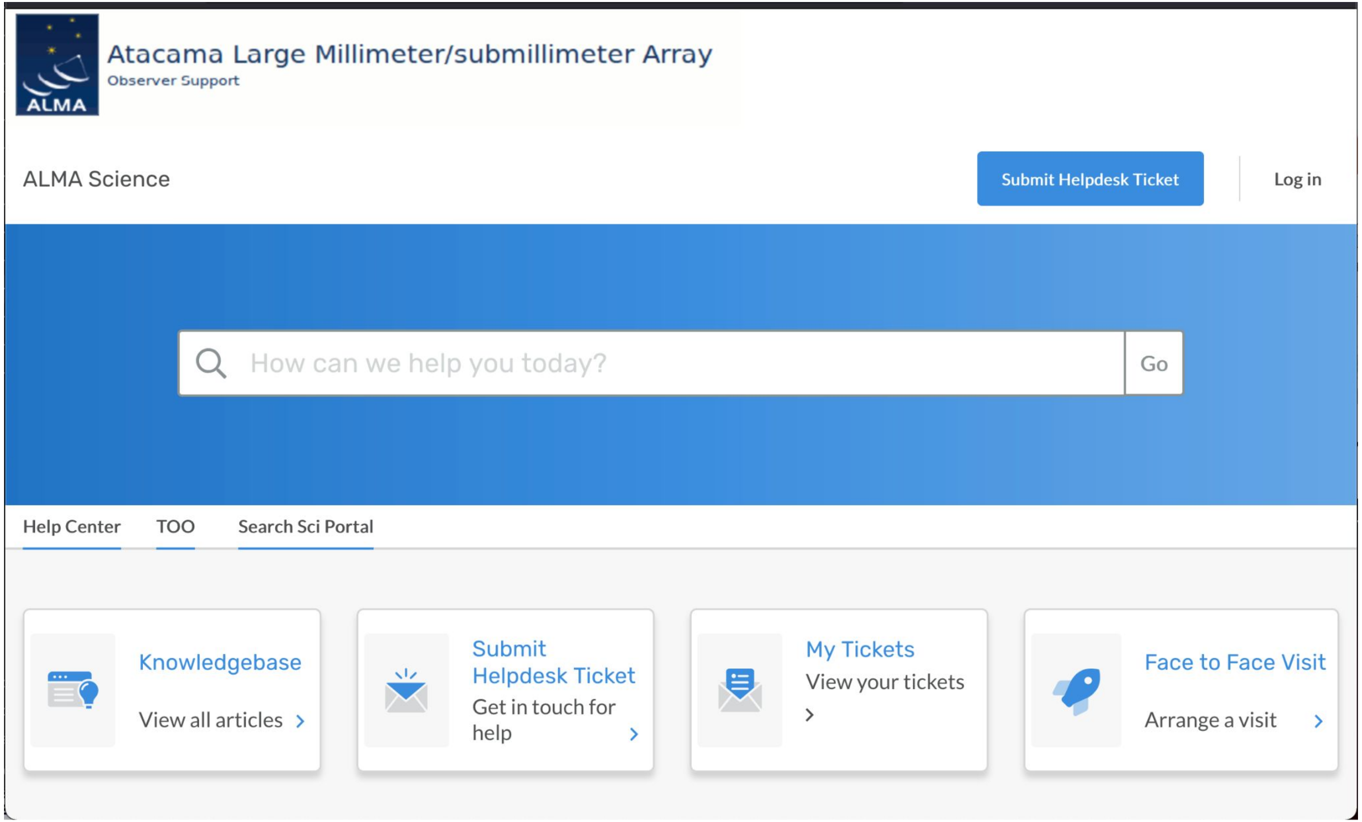
Documentation:

https://alminer.readthedocs.io/en/latest/?badge=latest

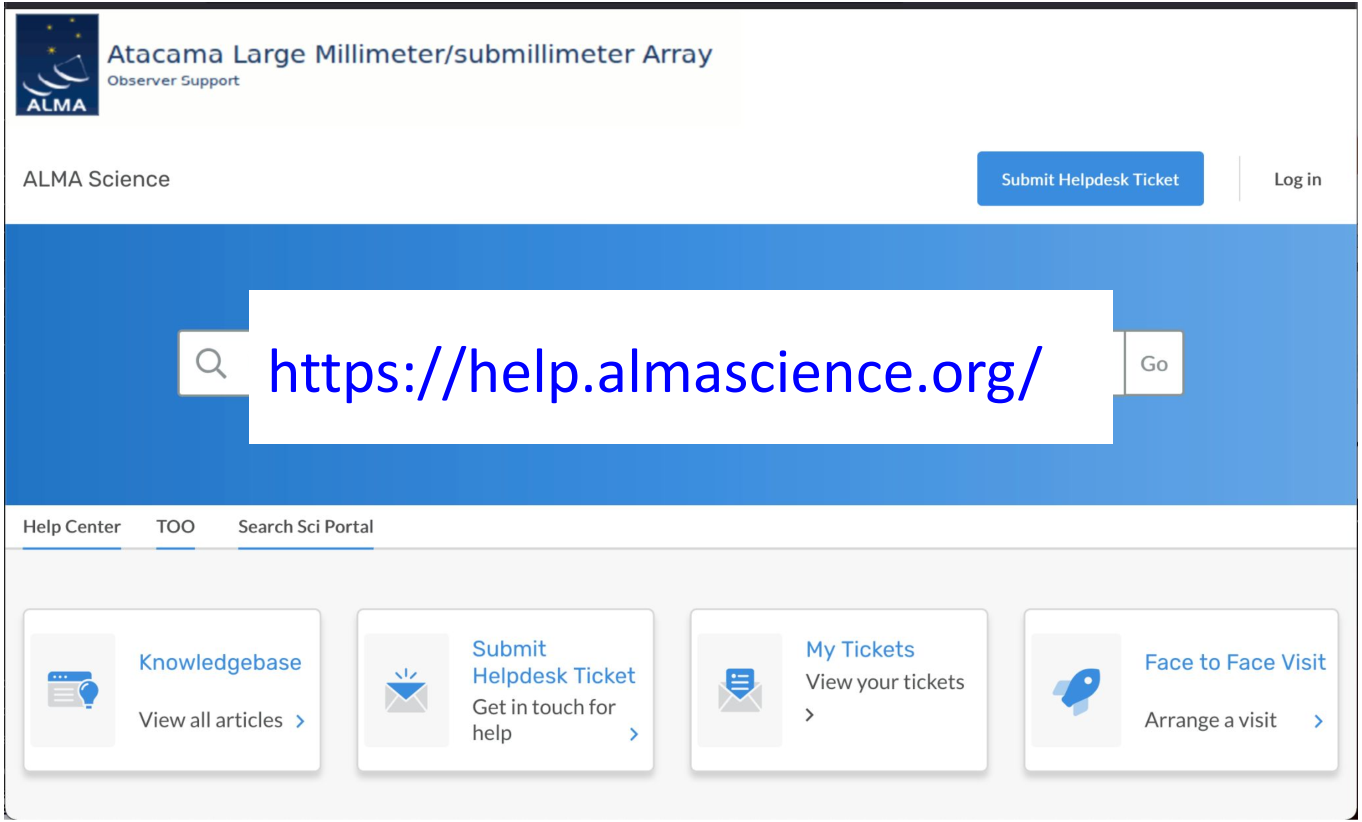
Tutorial notebook:

https://nbviewer.jupyter.org/github/emerge-erc/ALminer/blob/main/notebooks/tutorial/ALminer_tutorial.ipynb?flush_cache=True

ALMA helpdesk



ALMA helpdesk



EU arc network

- In-person and remote assistance with using ALMA, e.g.
  - Searching the ALMA archive
  - Proposal preparation
  - Data processing and analysis
- Events:
  - Training
  - Workshops
- High-powered computing facilities and data storage

https://www.eso.org/sci/facilities/alma/arc.html

I-TRAIN with the European ARC Network

ALMA
EUROPEAN ARC
ALMA Regional Centre

- Online trainings from the EU ARC with scripts, example data and video explanations.
- Topics include:
  - CARTA
  - ALminer
  - ALMA simulations
  - ALMA polarisation observations
  - And more!

https://almascience.eso.org/tools/eu-arc-network/i-train