

# Cube Reprojection Tutorial

## Authors

Adam Ginsburg, Eric Koch

## Learning Goals

- reproject a cube spectrally
- smooth it spectrally
- reproject it spatially

## Keywords

spectral cube, radio astronomy, astroquery, units, dask

## Summary

Spectroscopic cube observations taken at different wavelength can trace the motion of gas or stars using spectral lines, but often lines at different wavelengths give different information. For example, one might observe a galaxy in the 21cm line of HI and the 115 GHz line of CO, or a protoplanetary disk in a line of N<sub>2</sub>H<sup>+</sup> and a line of CO, or a galactic disk in the H-alpha and H-beta lines (in absorption or emission). In order to compare these data sets pixel-by-pixel, they must be placed onto a common grid with common resolution.

This tutorial shows how to take two spectral cubes observed toward the same part of the sky, but different frequencies, and put them onto the same grid using [spectral-cube](#).

It uses [astroquery](#) to obtain line frequencies from [splatalogue](#); this example uses radio-wavelength data for which Splatalogue's molecular line lists are appropriate. Finally, it shows how to do the

reprojection using [dask](#) to enable parallelization.

# Index

- [Step 1: Download](#)
- [Step 2: Open files, collect metadata](#)
- [Step 3: Convert to velocity](#)
- [Step 4: Spectral Interpolation](#)
- [Step 5: Spatial Smoothing](#)
- [Step 6: Reprojection](#)

In this example, we do spectral smoothing and interpolation (step 4) before spatial smoothing and interpolation (step 5), but if you have a varying-resolution cube (with a different beam size for each channel), you have to do spatial smoothing first. For more information see the [spectral-cube documentation](#).

## Step 1: Download the data

(you might not have to do this step, since you may already have data)

```
import numpy as np
from astropy.utils.data import download_file
```

We download two example spectral cubes of a point in the Galactic center from a permalink on the ALMA archives. These are moderately large files, with sizes 18 MB and 337 MB.

If you have trouble with these downloads, try changing to a different ALMA server (e.g., [almascience.eso.org](https://almascience.eso.org)->[almascience.nrao.edu](https://almascience.nrao.edu)) or increase the timeout. See the [download\\_file documentation](#).

```
filename_1 = download_file(
    "https://almascience.eso.org/dataPortal/member.uid___A001_X1465_X3a33.BrickMaser"
    cache=True,
)
```

```
filename_2 = download_file(
    "https://almascience.eso.org/dataPortal/member.uid___A001_X87d_X141.a_sma1_sci.s
    cache=True,
)
```

## Step 2: Load the cubes

```
from spectral_cube import SpectralCube
```

```
cube1 = SpectralCube.read(filename_1)
cube1
```

WARNING: StokesWarning: Cube is a Stokes cube, returning spectral cube for I component

SpectralCube with shape=(75, 250, 250) and unit=Jy / beam:

n_x:	250	type_x:	RA---SIN	unit_x:	deg	range:	266.534072 deg:	266.554577
n_y:	250	type_y:	DEC--SIN	unit_y:	deg	range:	-28.713958 deg:	-28.695975
n_s:	75	type_s:	FREQ	unit_s:	Hz	range:	139434992275.503 Hz:	1395039423

```
cube2 = SpectralCube.read(filename_2)
cube2
```

WARNING: StokesWarning: Cube is a Stokes cube, returning spectral cube for I component

SpectralCube with shape=(478, 420, 420) and unit=Jy / beam:

n_x:	420	type_x:	RA---SIN	unit_x:	deg	range:	266.537002 deg:	266.551600
n_y:	420	type_y:	DEC--SIN	unit_y:	deg	range:	-28.711371 deg:	-28.698569
n_s:	478	type_s:	FREQ	unit_s:	Hz	range:	216957714464.027 Hz:	2171906390

The cubes are at different frequencies - 139 and 217 GHz.

The first cube covers the H2CS 4(1,3)-3(1,2) line at 139.483699 GHz.

The second covers SiO v=5-4 at 217.104984 GHz

We use the `find_lines` tool to query [splatalogue](#) with [astroquery](#) over the spectral range covered by the cube. It returns a table of matching lines. Note that some line names will be repeated because Splatalogue includes several different databases and most chemical species are present in all of these.

```
cube1.find_lines(chemical_name=" H2CS ").show_in_notebook()
```

WARNING: ExperimentalImplementationWarning: The line-finding routine is experimental

key	...cies	name	..._nar	...d_Q	linelist	...tens	...ener	...ener	sijmu2	sij
0	210	H<sub>2</sub>CS	Thiofor...	4( 1, 3...	JPL	0.17	16.1465	20.799...	30.516...	11.223
1	210	H<sub>2</sub>CS	Thiofor...	4( 1, 3...	CDMS		16.1329	20.785...	30.596...	11.251
2	210	H<sub>2</sub>CS	Thiofor...	4( ...	SLAIM	0.17	16.133	20.785...	30.594...	3.75
3	210	H<sub>2</sub>CS	Thiofor...	4(1,3)-...	Lovas	0.17	0	0	0	0

```
cube2.find_lines(chemical_name="SiO").show_in_notebook()
```

WARNING: ExperimentalImplementationWarning: The line-finding routine is experimental

key	...cies	name	..._nar	...d_Q	linelist	...tens	...ener	...ener	sijmu2	sij
0	20	SiO <f...	Silicon...	5- 4	CDMS		14.4843	21.726...	47.991...	5
1	20	SiO <f...	Silicon...	5-4	JPL	1.6	14.4843	21.726...	48.146...	5
2	20	SiO <f...	Silicon...	5-4	Lovas	1.6	0	0	0	0
3	20	SiO <f...	Silicon...	5 ...	SLAIM		14.484	21.725...	47.6849	5

## Step 3: Convert cubes from frequency to velocity

To compare the kinematic structure of the target, we need to convert from the observed frequency (which must be in a common reference frame; in this case, it already is) to the doppler velocity.

```
from astropy import units as u
```

```
cube1vel = cube1.with_spectral_unit(  
    u.km / u.s, velocity_convention="radio", rest_value=139.483699 * u.GHz  
)  
cube1vel
```

SpectralCube with shape=(75, 250, 250) and unit=Jy / beam:

n_x:	250	type_x:	RA--SIN	unit_x:	deg	range:	266.534072	deg:	266.554577
n_y:	250	type_y:	DEC--SIN	unit_y:	deg	range:	-28.713958	deg:	-28.695975
n_s:	75	type_s:	VRAD	unit_s:	km / s	range:	-43.509	km / s:	104.

```
cube2vel = cube2.with_spectral_unit(  
    u.km / u.s, velocity_convention="radio", rest_value=217.104984 * u.GHz  
)  
cube2vel
```

SpectralCube with shape=(478, 420, 420) and unit=Jy / beam:

n_x:	420	type_x:	RA--SIN	unit_x:	deg	range:	266.537002	deg:	266.551600
n_y:	420	type_y:	DEC--SIN	unit_y:	deg	range:	-28.711371	deg:	-28.698569
n_s:	478	type_s:	VRAD	unit_s:	km / s	range:	-118.278	km / s:	203.

From the shape of the cube, we can see the H2CS cube is narrower in velocity, so we'll use that as the target spectral reprojection. However, the SiO cube is the smaller footprint on the sky.

## Create spatial maps of the peak intensity to quickly explore the cubes:

One way to quickly explore the structure in the data cubes is to produce a peak intensity map, or the maximum along the spectral axis (`axis=0`).

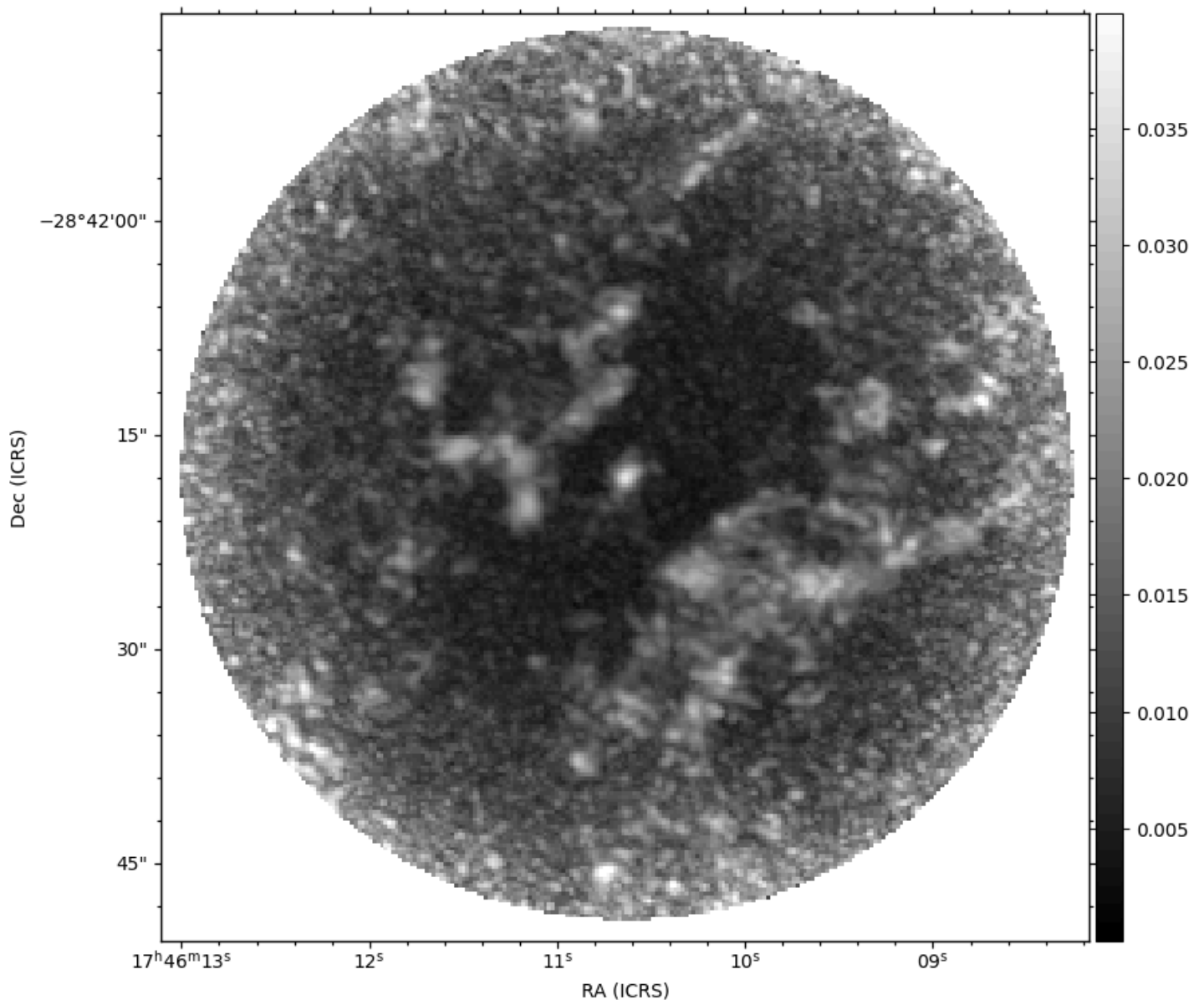
```
mx = cube1.max(axis=0)  
mx.quicklook()
```

```
WARNING: PossiblySlowWarning: This function (<function BaseSpectralCube.max at 0x7ff
/opt/hostedtoolcache/Python/3.12.11/x64/lib/python3.12/site-packages/spectral_cube/s
out = function(self._get_filled_data(fill=fill,
```

```
WARNING: AstropyDeprecationWarning: CoordinateHelper.ticks should not be accessed di
```

```
INFO: Auto-setting vmin to 1.944e-04 [aplpy.core]
```

```
INFO: Auto-setting vmax to 3.995e-02 [aplpy.core]
```



We can do the same thing all on one line (for the other cube this time):

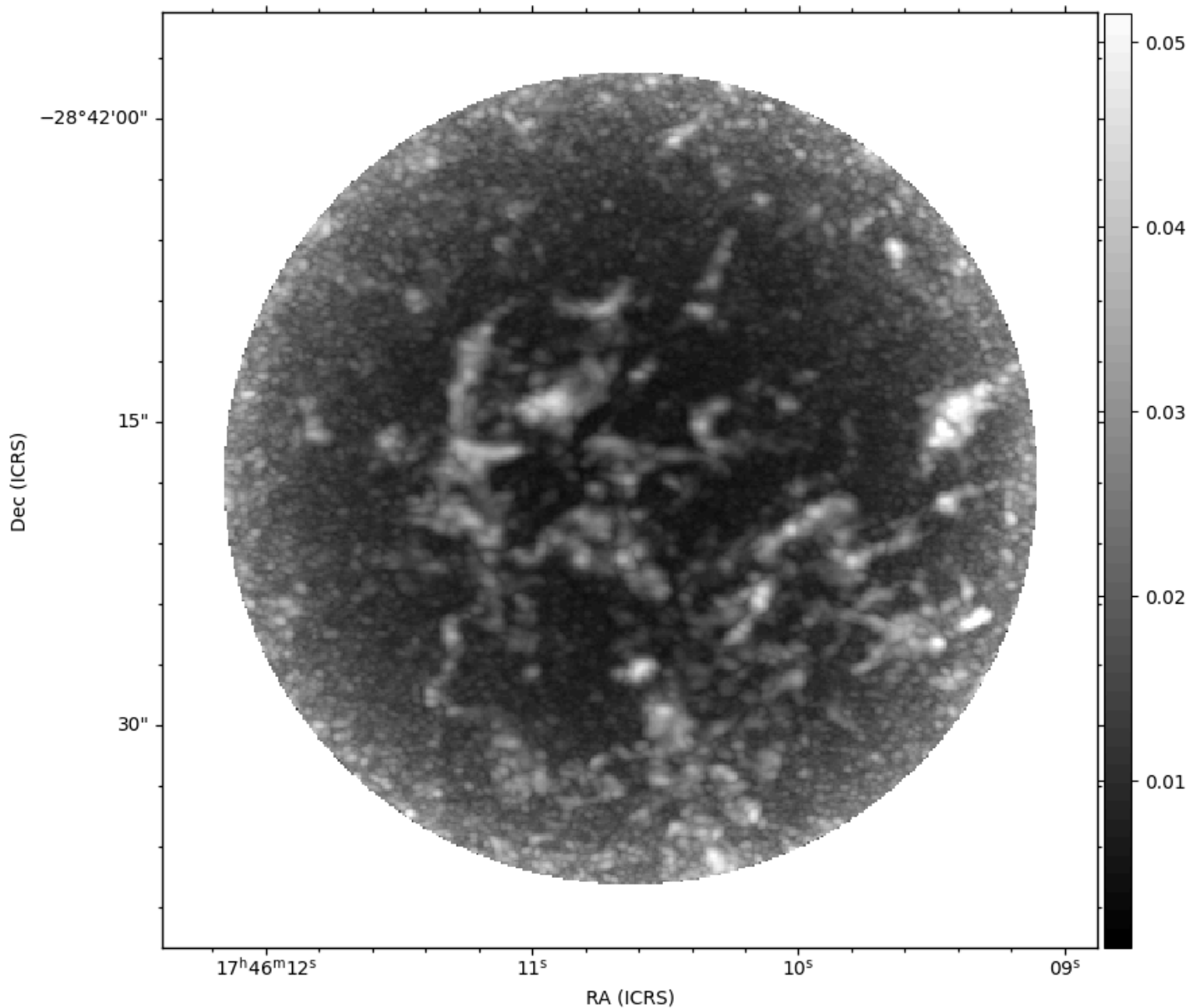
```
cube2.max(axis=0).quicklook()
```

```
WARNING: PossiblySlowWarning: This function (<function BaseSpectralCube.max at 0x7ff
```

```
/opt/hostedtoolcache/Python/3.12.11/x64/lib/python3.12/site-packages/spectral_cube/s  
out = function(self._get_filled_data(fill=fill,  
WARNING: AstropyDeprecationWarning: CoordinateHelper.ticks should not be accessed di
```

```
INFO: Auto-setting vmin to 1.002e-03 [aplpy.core]
```

```
INFO: Auto-setting vmax to 5.154e-02 [aplpy.core]
```



We can choose to do either the spatial or spectral step first.

In this case, we choose the spectral step first because the H<sub>2</sub>CS cube is narrower in velocity (`cube1vel`) and this will reduce the number of channels we need to spatially interpolate over in the next step.



We need to match resolution to the cube with the largest channel width:

```
velocity_res_1 = np.diff(cube1vel.spectral_axis)[0]
velocity_res_2 = np.diff(cube2vel.spectral_axis)[0]
velocity_res_1, velocity_res_2
```

```
(<Quantity 2.00262828 km / s>, <Quantity 0.67429189 km / s>)
```

Next, we will reduce `cube2vel` to have the same spectral range as `cube1vel`:

```
cube2vel_cutout = cube2vel.spectral_slab(
    cube1vel.spectral_axis.min(), cube1vel.spectral_axis.max()
)
cube1vel, cube2vel_cutout
```

```
(SpectralCube with shape=(75, 250, 250) and unit=Jy / beam:
  n_x:    250  type_x: RA---SIN  unit_x: deg    range:   266.534072 deg:   266.554577
  n_y:    250  type_y: DEC---SIN  unit_y: deg    range:  -28.713958 deg:  -28.695975
  n_s:     75  type_s: VRAD      unit_s: km / s  range:   -43.509 km / s:    104
SpectralCube with shape=(221, 420, 420) and unit=Jy / beam:
  n_x:    420  type_x: RA---SIN  unit_x: deg    range:   266.537002 deg:   266.551600
  n_y:    420  type_y: DEC---SIN  unit_y: deg    range:  -28.711371 deg:  -28.698569
  n_s:    221  type_s: VRAD      unit_s: km / s  range:   -43.432 km / s:    104
```

Note that it is important for the to-be-interpolated cube, in this case `cube2`, to have pixels bounding `cube1`'s spectral axis, but in this case it does not. If the pixel range doesn't overlap perfectly, it may blank out one of the edge pixels. So, to fix this, we add a little buffer:

```
cube2vel_cutout = cube2vel.spectral_slab(
    cube1vel.spectral_axis.min() - velocity_res_2, cube1vel.spectral_axis.max()
)
cube1vel, cube2vel_cutout
```

```
(SpectralCube with shape=(75, 250, 250) and unit=Jy / beam:
  n_x:    250  type_x: RA---SIN  unit_x: deg    range:   266.534072 deg:   266.554577
  n_y:    250  type_y: DEC---SIN  unit_y: deg    range:  -28.713958 deg:  -28.695975
  n_s:     75  type_s: VRAD      unit_s: km / s  range:   -43.509 km / s:    104
SpectralCube with shape=(222, 420, 420) and unit=Jy / beam:
  n_x:    420  type_x: RA---SIN  unit_x: deg    range:   266.537002 deg:   266.551600
  n_y:    420  type_y: DEC---SIN  unit_y: deg    range:  -28.711371 deg:  -28.698569
  n_s:    222  type_s: VRAD      unit_s: km / s  range:  -44.106 km / s:    104
```

Our H2CS cube (`cube1vel`) has broader channels. We need to first smooth `cube2vel` to the broader channel width before doing the spatial reprojection.

To do this, we will spectrally smooth with a Gaussian with width set such that smoothing `cube2vel` will result in the same width as `cube1vel`. We do this by finding the difference in widths when deconvolving the `cube1vel` channel width from `cube2vel`. For further information see the [documentation on smoothing](#).

Note that if we did not do this smoothing step, we would under-sample the `cube2vel` data in the next downsampling step, reducing our signal-to-noise ratio.

We have adopted a width equal to the channel width; the [line spread function](#) is actually a Hanning-smoothed tophat. We are making a coarse approximation here.

```
fwhm_gaussian = (velocity_res_1**2 - velocity_res_2**2) ** 0.5
fwhm_gaussian
```

$$1.8856963 \frac{\text{km}}{\text{s}}$$

```
from astropy.convolution import Gaussian1DKernel

fwhm_to_sigma = np.sqrt(8 * np.log(2))
# we want the kernel in pixel units, so we force to km/s and take the value
spectral_smoothing_kernel = Gaussian1DKernel(
    stddev=fwhm_gaussian.to(u.km / u.s).value / fwhm_to_sigma
)
```

We then smooth with the kernel. Note that this is doing  $420 \times 420 = 176400$  smoothing operations on a length-221 spectrum: it will take a little time

```
cube2vel_smooth = cube2vel_cutout.spectral_smooth(spectral_smoothing_kernel)
```

```
WARNING: nan_treatment='interpolate', however, NaN values detected post convolution.
```

Now that we've done spectral smoothing, we can resample the spectral axis of `cube2vel_smooth` to match `cube1vel` by interpolating `cube2vel_smooth` onto `cube1vel`'s grid:

```
cube2vel_spectralresample = cube2vel_smooth.spectral_interpolate(
    cube1vel.spectral_axis, suppress_smooth_warning=True
)
cube2vel_spectralresample
```

Spectral Interpolate: 100%

176400/176400 [00:09<00:00, 52225.57it/s]

SpectralCube with shape=(75, 420, 420) and unit=Jy / beam:

```
n_x:    420  type_x: RA--SIN  unit_x: deg    range:    266.537002 deg:    266.551600
n_y:    420  type_y: DEC--SIN unit_y: deg    range:    -28.711371 deg:    -28.698569
n_s:     75  type_s: VRAD     unit_s: km / s  range:      -43.509 km / s:    104.
```

Note that we included the `suppress_smooth_warning=True` argument. That is to hide this warning:

```
WARNING: SmoothingWarning: Input grid has too small a spacing. The data should be smoothed
```

which will tell you if the operation will under-sample the original data. The smoothing work we did above is specifically to make sure we are properly sampling, so this warning does not apply.

Now that we've done spectral smoothing, we also need to follow a similar procedure of smoothing then resampling for the spatial axes.

The `beam` is the resolution element of our cubes:

```
cube1vel.beam, cube2vel_spectralresample.beam
```

```
(Beam: BMAJ=1.29719604986604 arcsec BMIN=1.04247149438736 arcsec BPA=82.95313553702
Beam: BMAJ=0.8935712308515601 arcsec BMIN=0.6649610689789199 arcsec BPA=85.81119797
```

`cube1` again has the larger beam, so we'll smooth `cube2` to its resolution

## Aside: mixed beams

If `cube1` and `cube2` had different sized beams, but neither was clearly larger, we would have to convolve *both* to a [common beam](#).

In this case, it's redundant and we could have just used `cube1`'s beam, but this is the more general approach:

```
import radio_beam

common_beam = radio_beam.commonbeam.common_2beams(
    radio_beam.Beams(beams=[cube1vel.beam, cube2vel.beam])
)
common_beam
```

*Beam : BMAJ = \$1.29719604986604''\$BMIN = \$1.04247149438736''\$BPA = \$82.953:*

We then convolve:

```
# for v<0.6, we convert to Kelvin to ensure the units are preserved:
# cube2vel_spatialspectralsmooth = cube2vel_spectralresample.to(u.K).convolve_to(common_beam)
# in more recent versions, the unit conversion is handled appropriately,
# so unit conversion isn't needed
cube2vel_spatialspectralsmooth = cube2vel_spectralresample.convolve_to(common_beam)
cube2vel_spatialspectralsmooth
```

SpectralCube with shape=(75, 420, 420) and unit=Jy / beam:

```
n_x:    420  type_x: RA--SIN  unit_x: deg    range:    266.537002 deg:   266.551600
n_y:    420  type_y: DEC--SIN  unit_y: deg    range:    -28.711371 deg:   -28.698569
n_s:     75  type_s: VRAD      unit_s: km / s range:      -43.509 km / s:    104.
```

Now we can do the spatial resampling as the final step for producing two cubes matched to the same spatial and spectral pixel grid:

```
# first we make a copy of the target (cube1vel) header and set its rest frequency
# to that of the cube we're reprojecting (cube2vel_spatialspectralsmooth)
# (see https://github.com/radio-astro-tools/spectral-cube/issues/874)
tgthdr = cube1vel.header
tgthdr["RESTFRQ"] = cube2vel_spatialspectralsmooth.header["RESTFRQ"]

# now we continue with the reprojection
cube2vel_reproj = cube2vel_spatialspectralsmooth.reproject(tgthdr)
cube2vel_reproj
```

```
SpectralCube with shape=(75, 250, 250) and unit=Jy / beam:
n_x:    250  type_x: RA---SIN  unit_x: deg    range:    266.534072 deg:   266.554577
n_y:    250  type_y: DEC--SIN  unit_y: deg    range:    -28.713958 deg:  -28.695975
n_s:     75  type_s: VRAD      unit_s: km / s  range:      -43.509 km / s:    104.
```

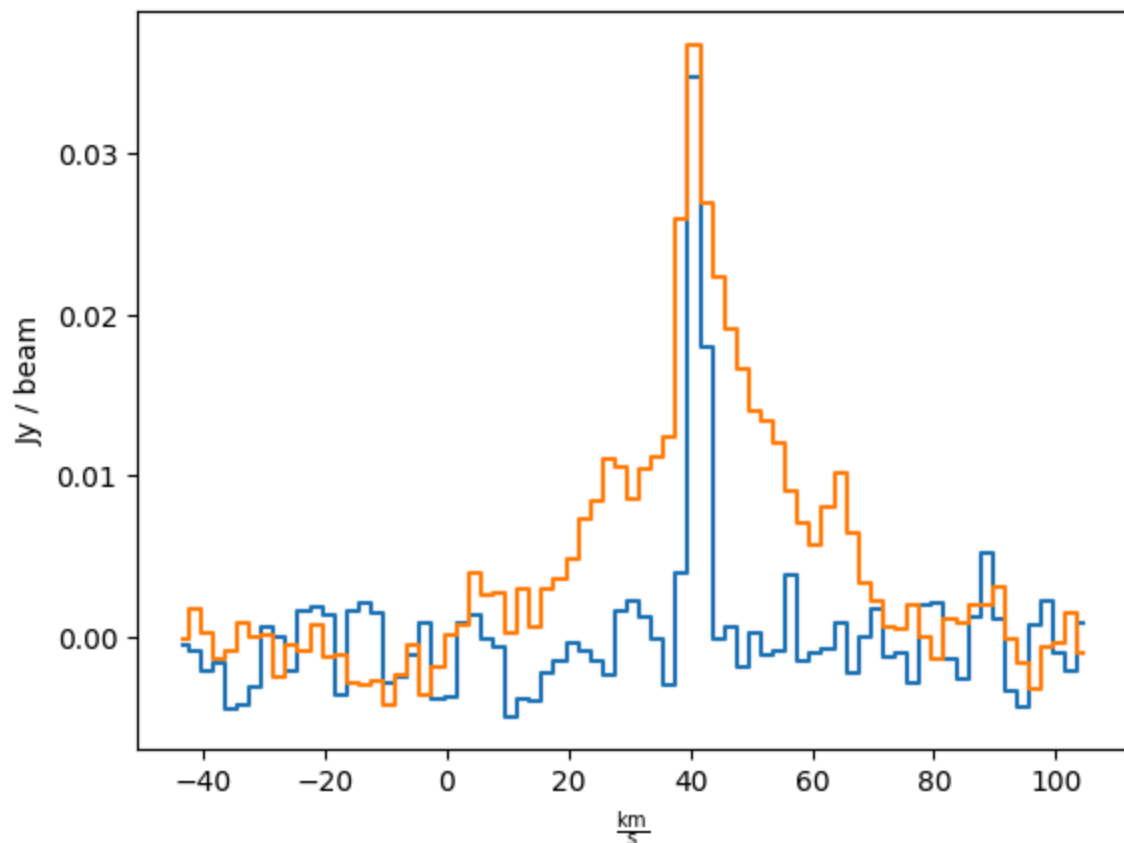
These two cubes are now on an identical grid, and can be directly compared:

```
cube2vel_reproj, cube1vel
```

```
(SpectralCube with shape=(75, 250, 250) and unit=Jy / beam:
n_x:    250  type_x: RA---SIN  unit_x: deg    range:    266.534072 deg:   266.554577
n_y:    250  type_y: DEC--SIN  unit_y: deg    range:    -28.713958 deg:  -28.695975
n_s:     75  type_s: VRAD      unit_s: km / s  range:      -43.509 km / s:    104
SpectralCube with shape=(75, 250, 250) and unit=Jy / beam:
n_x:    250  type_x: RA---SIN  unit_x: deg    range:    266.534072 deg:   266.554577
n_y:    250  type_y: DEC--SIN  unit_y: deg    range:    -28.713958 deg:  -28.695975
n_s:     75  type_s: VRAD      unit_s: km / s  range:      -43.509 km / s:    104
```

These spectra can now be overplotted as they are in the same unit with the same beam.

```
cube1vel[:, 125, 125].quicklook()
cube2vel_reproj[:, 125, 125].quicklook()
```



All of the above can be done using `dask` as the underlying framework to parallelize the operations. See [the spectral-cube documentation on dask integration](#) or the [dask documentation](#) for further details.

The dask approach can be made more memory-efficient (avoid using too much RAM) by writing intermediate steps to disk. The non-dask approach used above will generally need to read the whole cube into memory. Depending on the situation, either approach may be faster, but `dask` may be needed if the cube is larger than memory.

We repeat all the operations above using dask. We use a `ProgressBar` so you can see how long it takes. We also suppress warnings to make the output look cleaner (we already saw all the important warnings above).

```
from dask.diagnostics import ProgressBar
import warnings
```

```

with warnings.catch_warnings():
    warnings.simplefilter("ignore")
    with ProgressBar():
        cube2dask = SpectralCube.read(filename_2, use_dask=True)
        cube2daskvel = cube2dask.with_spectral_unit(
            u.km / u.s, velocity_convention="radio", rest_value=217.104984 * u.GHz
        )
        cube2daskvel_cutout = cube2daskvel.spectral_slab(
            cube1vel.spectral_axis.min() - velocity_res_2, cube1vel.spectral_axis.max() + velocity_res_2
        )
        cube2daskvel_smooth = cube2daskvel_cutout.spectral_smooth(
            spectral_smoothing_kernel
        )
        cube2daskvel_spectralresample = cube2daskvel_smooth.spectral_interpolate(
            cube1vel.spectral_axis, suppress_smooth_warning=True
        )
        cube2daskvel_spatialspectralsmooth = cube2daskvel_spectralresample.convolve(
            common_beam
        )
        cube2daskvel_reproj = cube2daskvel_spatialspectralsmooth.reproject(
            tgthdr
        ) # as above, tgthdr is altered cube1vel header
cube2daskvel_reproj

```

[	]		0% Completed		344.44 us
[	]		0% Completed		102.44 ms
[	]		0% Completed		289.59 ms
[	]		0% Completed		564.77 ms
[	]		0% Completed		792.41 ms
[	]		0% Completed		898.84 ms
[	]		0% Completed		1.70 s
[	]		0% Completed		1.81 s
[	]		0% Completed		2.04 s
[	]		0% Completed		2.32 s
[	]		0% Completed		2.42 s
[	]		0% Completed		2.52 s
[	]		0% Completed		2.62 s
[	]		0% Completed		3.39 s



```
[ ] | 0% Completed | 3.49 s

[ ] | 0% Completed | 3.59 s

[ ] | 0% Completed | 3.69 s

[ ] | 0% Completed | 154.01 s

[ ] | 0% Completed | 154.11 s

[ ] | 0% Completed | 154.34 s

[ ] | 0% Completed | 154.58 s

[ ] | 0% Completed | 154.68 s

[ ] | 0% Completed | 155.44 s

[ ] | 0% Completed | 155.54 s

[ ] | 0% Completed | 155.64 s

[#####] | 100% Completed | 155.74 s
```

DaskSpectralCube with shape=(75, 250, 250) and unit=Jy / beam and chunk size (75, 25  
n\_x: 250 type\_x: RA---SIN unit\_x: deg range: 266.534072 deg: 266.554577  
n\_y: 250 type\_y: DEC---SIN unit\_y: deg range: -28.713958 deg: -28.695975  
n\_s: 75 type\_s: VRAD unit\_s: km / s range: -43.509 km / s: 104.