Feathering

This talk (FeatheringPresentation.ipynb) can be found at http://small.cat/hat (http://small.cat/hat) until next week. Please follow along on your laptop or phone.

Goals:

- Describe and demonstrate feathering process
- Show some of the free parameters that can be selected when feathering
- Show various visualizations that can be used to assess goodness-of-merge
    - PSD plots
    - Residuals from theoretical
- Demonstrate limitations of image combination
- Show tools for simulating realistic synthetic data

What is feathering?

Fourier-space (UV-space) weighted averaging of two data sets.

The weight function is selected to match the Fourier transform of the single-dish telescope's beam.

Benefits & Costs of feathering

The good:

- Simple
- Applicable to non-interferometer (e.g., bolometer + other) data

The bad:

- *Deceptively* simple
- Ignores spectral information (see SDINT, TP2VIS talks)
- Does not fix interferometric image reconstruction problems
- Tricky to balance weights for correctness vs optimal noise

radio-astro-tools

http://radio-astro-tools.github.io (http://radio-astro-tools.github.io)

Includes tools for cubes (https://spectral-cube.readthedocs.io (https://spectral-cube.readthedocs.io)), beam metadata handling (https://github.com/radio-astro-tools/radio-beam (https://github.com/radio-astro-tools/radio-beam)), and feathering (https://github.com/radio-astro-tools/uvcombine/ (https://github.com/radio-astro-tools/uvcombine/))

Feathering (combination) tests

This notebook presents a series of experiments in single-dish + interferometer combination on "realistic" data.

We're "observing" at 2mm, so a 12m dish has a FWHM=40" Gaussian beam and a 9m baseline has a sharp cutoff at 56"

This presentation deals mostly with idealized cases; there is no primary beam degradation included in the simulations.

Requirements for this work: `turbustat` (https://github.com/Astroua/TurbuStat/) generates our synthetic data and helps with power-spectral-density (PSD) plotting. `astropy.convolution` provides access to convolution tools, and `uvcombine` is our python-only implementation of `feather`.

https://turbustat.readthedocs.io/en/latest/ (https://turbustat.readthedocs.io/en/latest/), especially https://turbustat.readthedocs.io/en/latest/generating_test_data.html (https://turbustat.readthedocs.io/en/latest/generating_test_data.html)

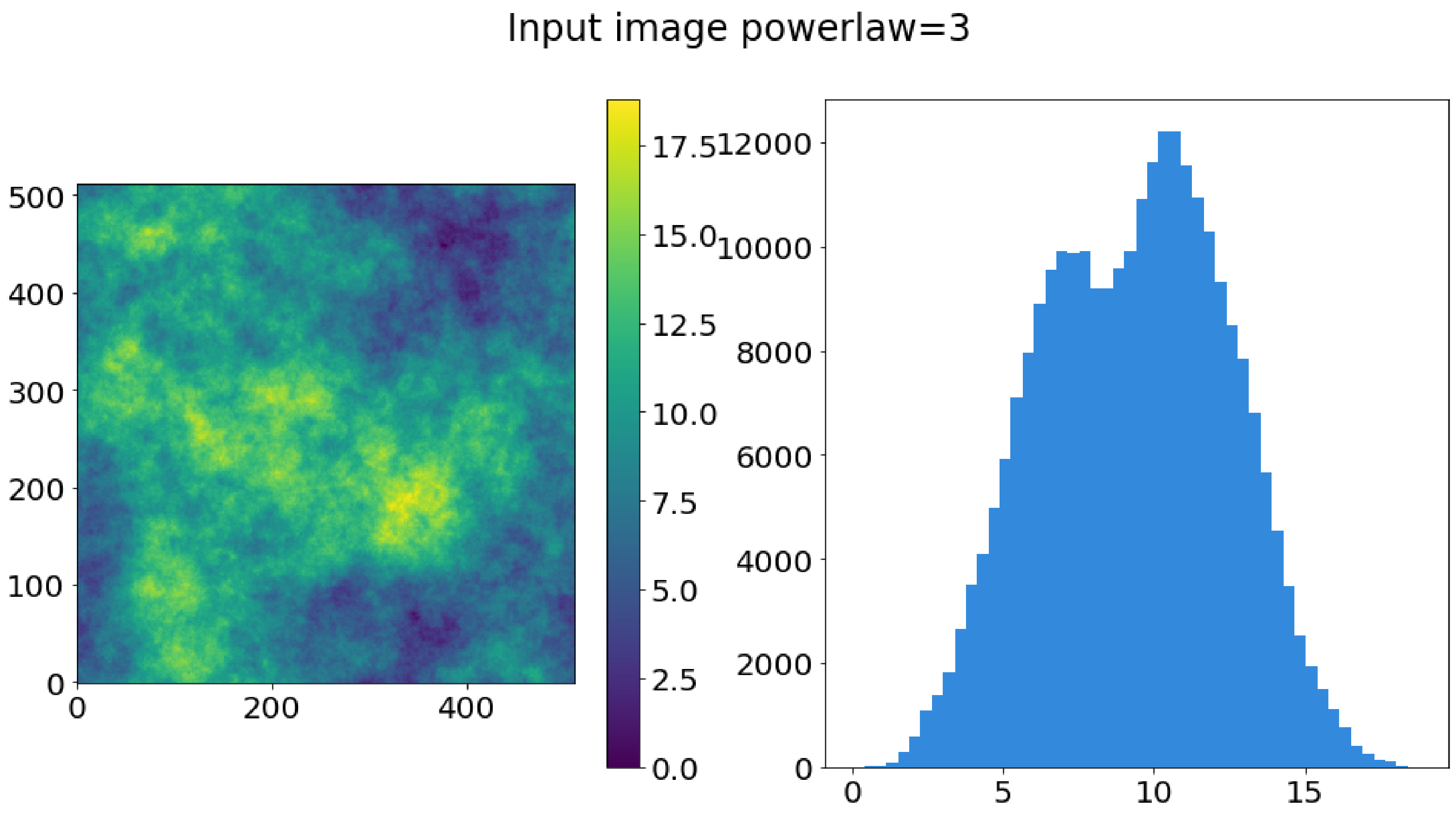
```
from turbustat.simulator.gen_field import make_extended
from turbustat.statistics import psds
from astropy import convolution, units as u
import numpy as np
from uvcombine.uvcombine import feather_kernel, fftmerge
```

We create a synthetic power-law power-spectrum image. This sort of image is typical of a dust image of the Galactic plane, for example.

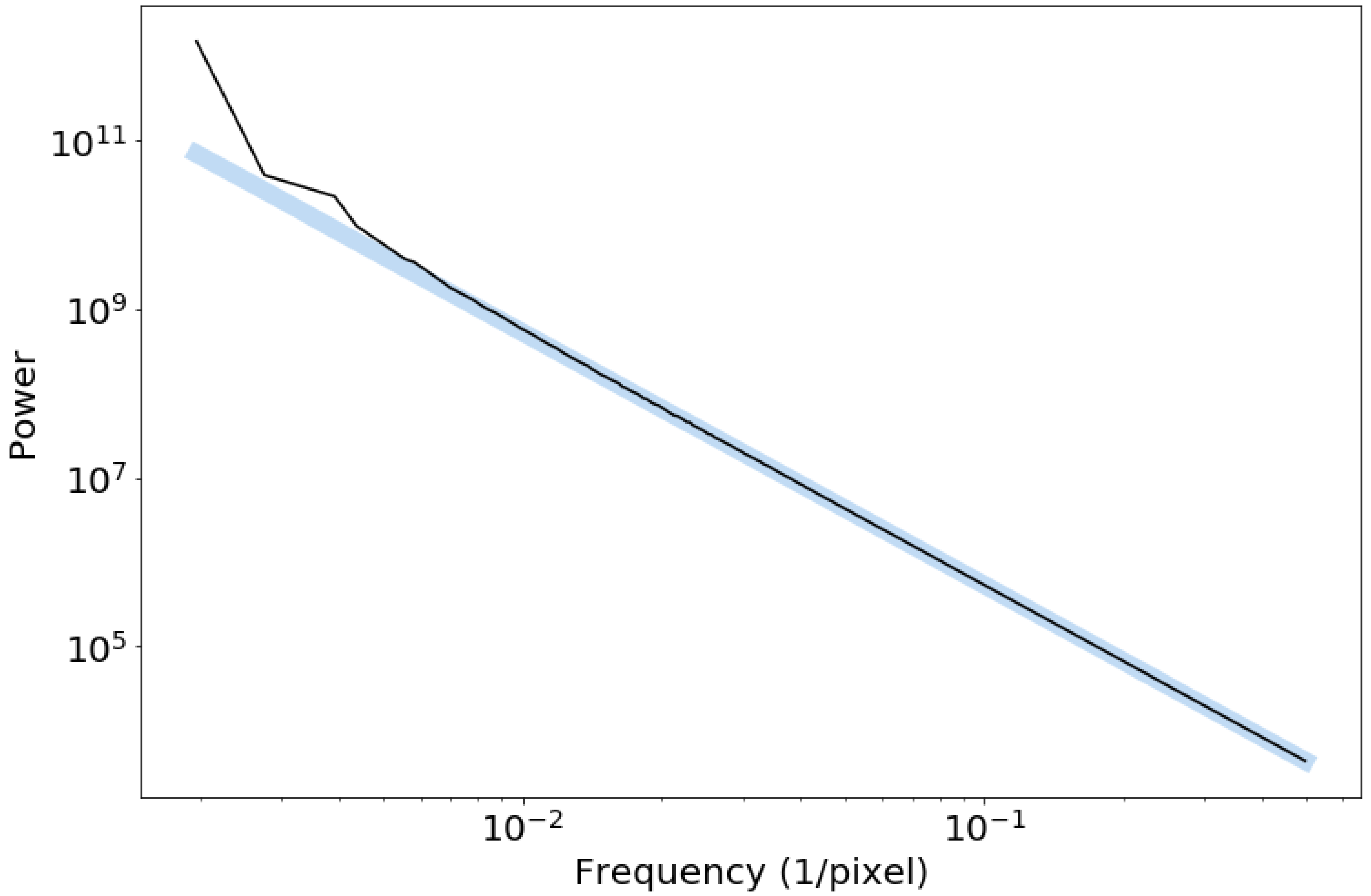
```
# create an input image with specified parameters
# (this can later be modified - it will be good to examine the effects of
# different power laws, different types of input...)
# We're assuming a scale of 1"/pixel for this example
imsize = 512
powerlaw = 3
im = make_extended(imsize=imsize, powerlaw=powerlaw, randomseed=0)
# the real sky is positive, so we subtract the minimum to force the overall image positive
im = im - im.min()
```

Input Image Visualization

This is the input image along with its histogram.



The power spectrum of the input image (set to be $\alpha = 3$), verifying that the turbustat code works



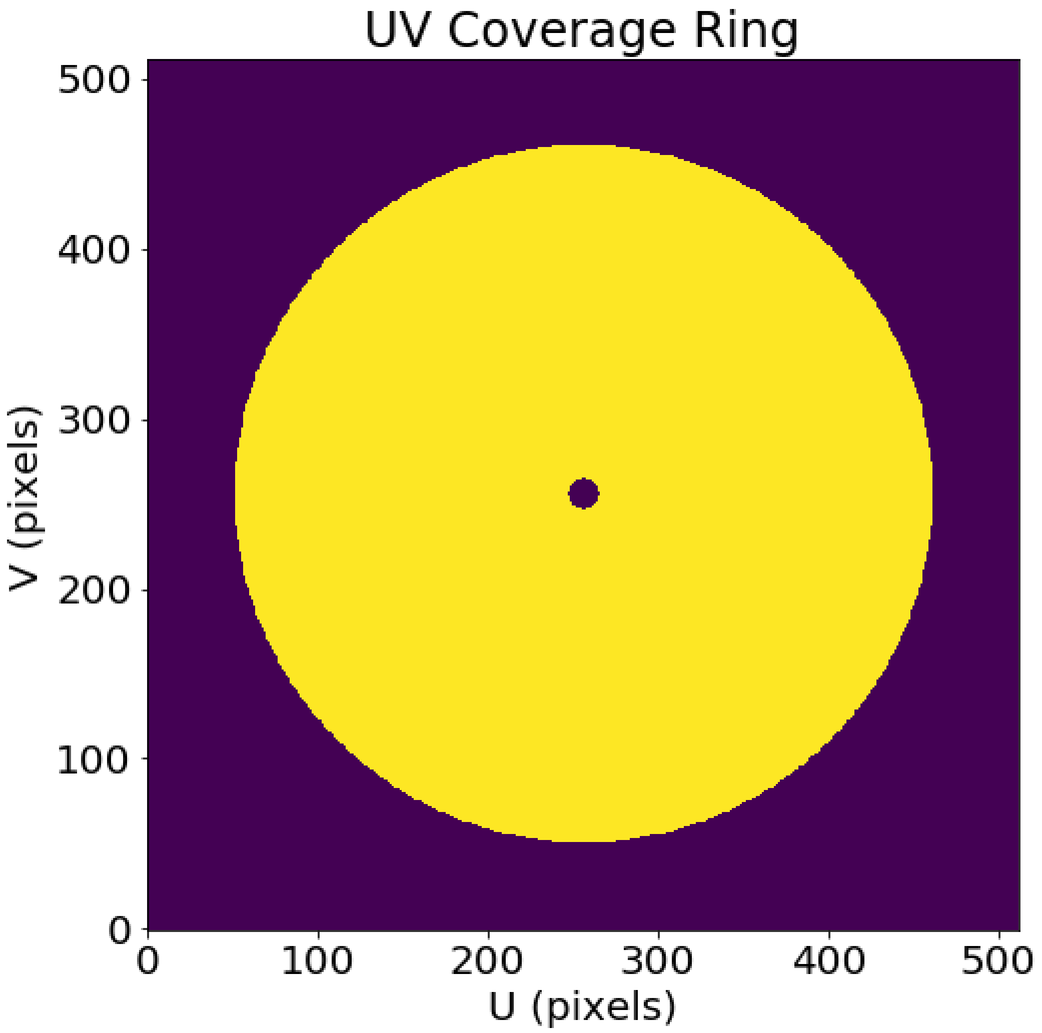
Next, we create our simulated interferometer by creating a UV domain and selecting which pixels in that domain will be part of our telescope. This process creates an ideal interferometer.

```
# set up the grid
ygrid, xgrid = np.indices(im.shape, dtype='float')
rr = ((xgrid-im.shape[1]/2)**2+(ygrid-im.shape[0]/2)**2)**0.5
```

```
# Create a UV sampling mask.
# This removes all large-angular scale (r<8) features *in UV space* and all
# small angular scales.
# In fourier space, r=0 corresponds to the DC component
# r=1 corresponds to the full map (one period over that map)
# r=256 is the smallest angular scale, which is 2 pixels
# We're assuming a pixel scale of 1" / pixel
# therefore 56" corresponds to 9m at 2mm (i.e., nearly the closest spacing possible for 7m)
# We cut off the "interferometer" at 2.5" resolution
largest_scale = 56.*u.arcsec
smallest_scale = 2.5*u.arcsec
pixscale = 1*u.arcsec
image_scale = im.shape[0]*pixscale # assume symmetric (default=256)
ring = (rr>=(image_scale/largest_scale)) & (rr<=(image_scale/smallest_scale))
```

Synthetic Perfect Interferometer

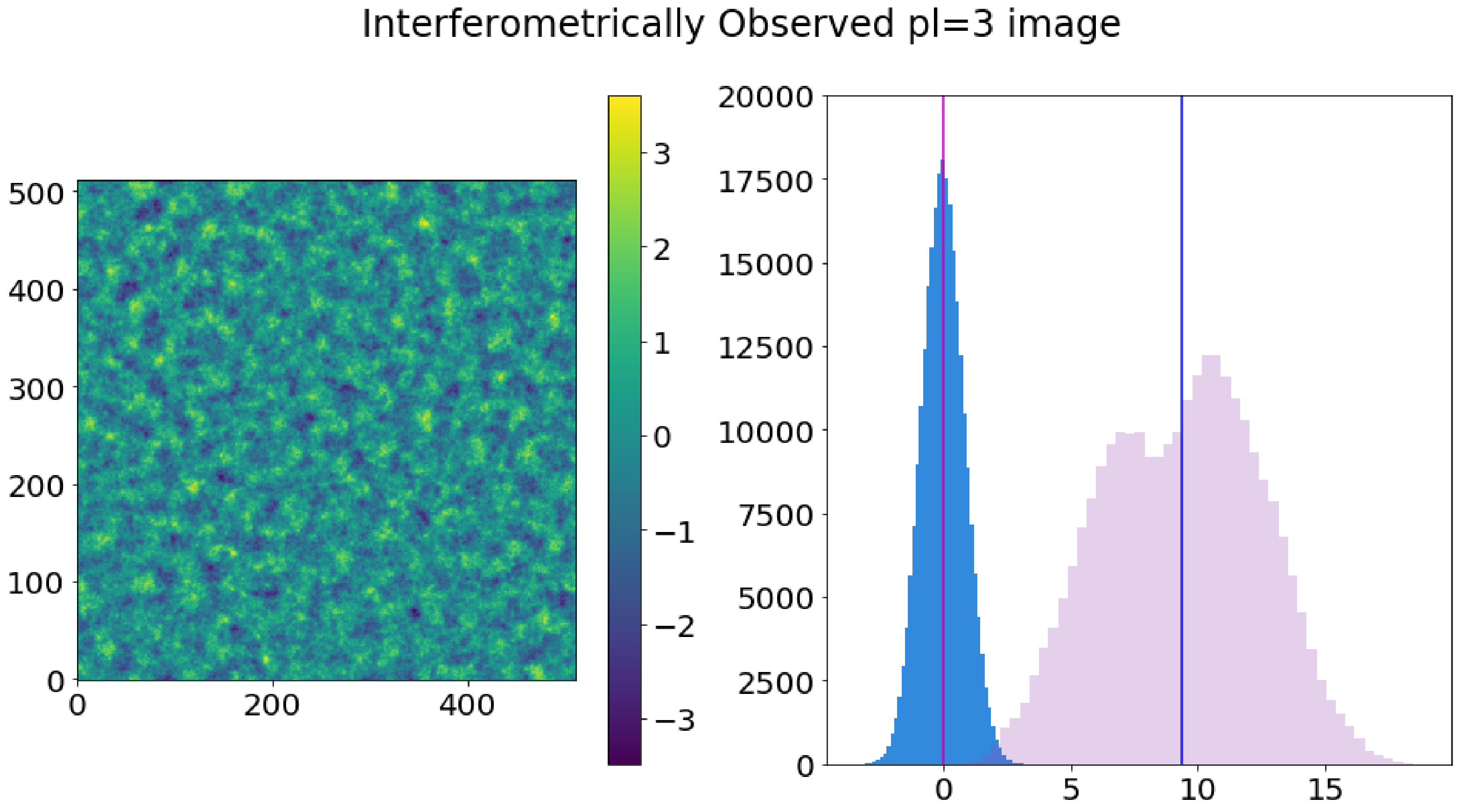
The synthetic interferometer's UV coverage map (it's a perfect interferometer)


UV Coverage Ring
V (pixels)
U (pixels)
0
100
200
300
400
500
0
100
200
300
400
500

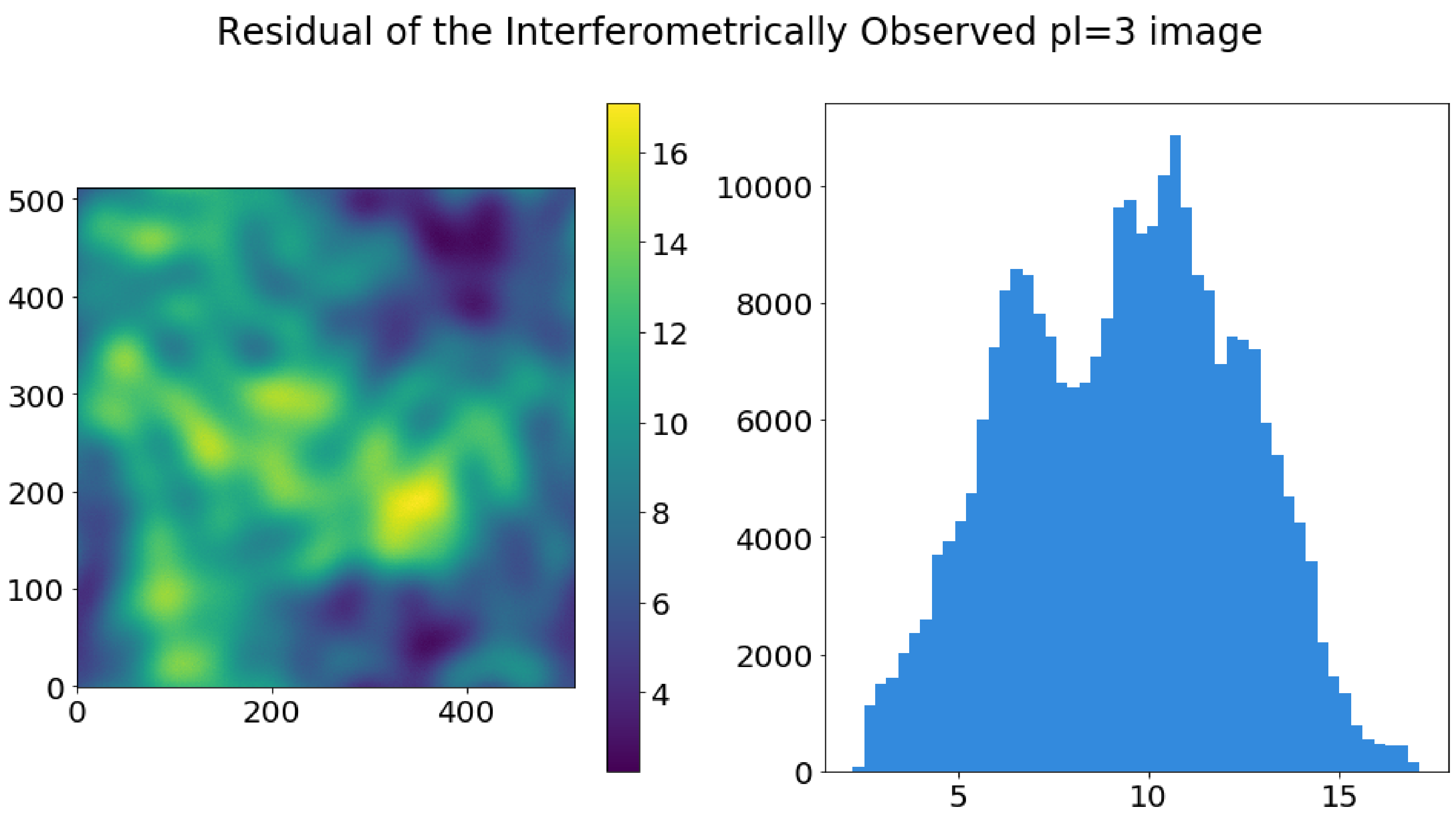
Next, we create the interferometric map by multiplying our interferometer mask by the fourier transform of the sky data

```
# create the interferometric map by removing both large and small angular
# scales in fourier space
imfft = np.fft.fft2(im)
imfft_interferometered = imfft * np.fft.fftshift(ring)
im_interferometered = np.fft.ifft2(imfft_interferometered)
```

The interferometric image does not preserve total flux, as expected. Note that the mean of the histogram is shifted.



The residual of the original image minus the interferometrically observed image. The large scales and noise are preserved.



Not *quite* realistic

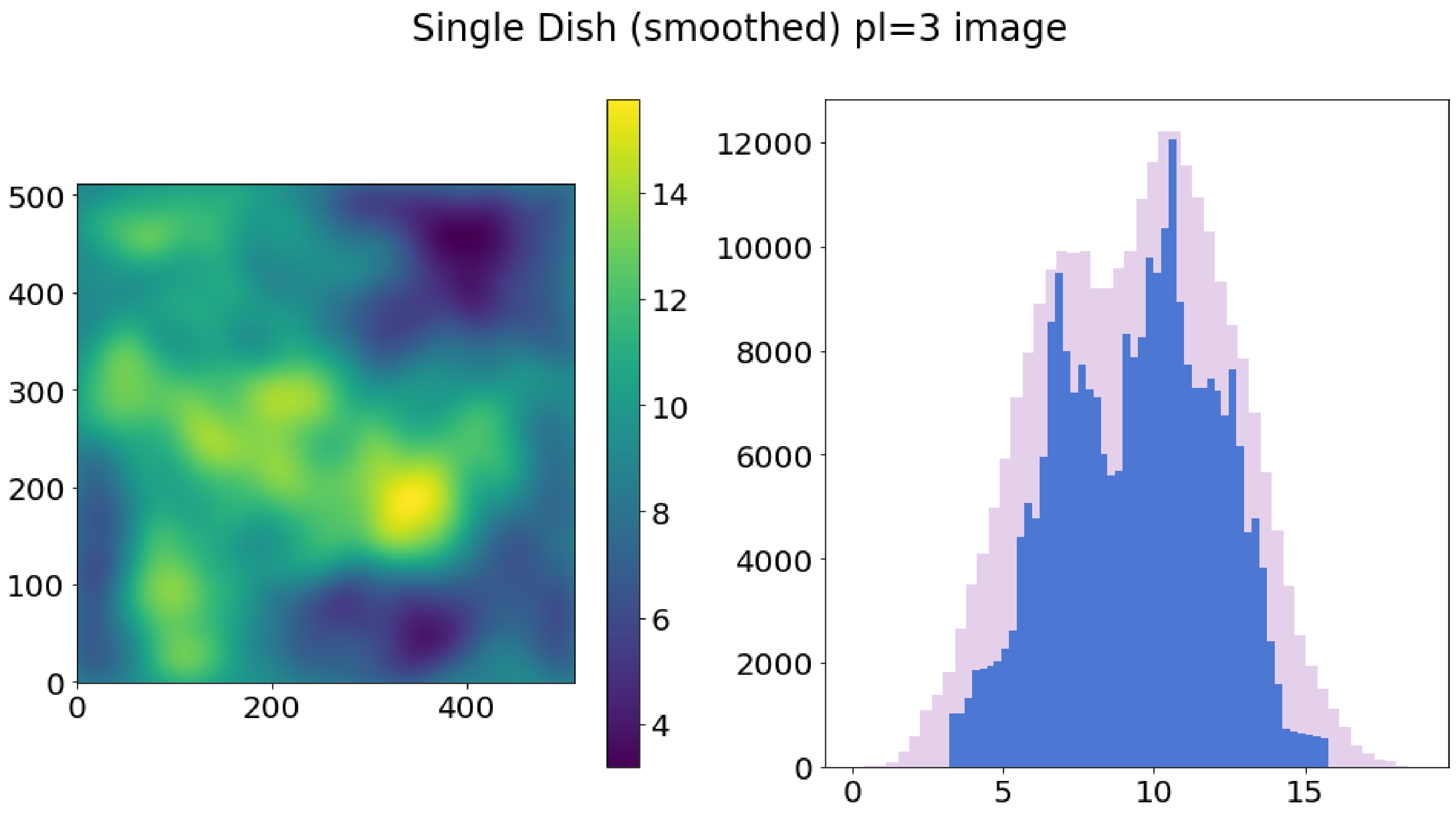
This synthetic interferometer map is a "perfect" interferometer image, which is not quite analogous to images produced by CASA, AIPS, etc. The CLEAN algorithm, while mostly intended to remove PSF artifacts from the data, also adds some power into the short spacings.

Synthetic Single Dish

The single dish map is just a convolution of the original data with a Gaussian beam. It preserves flux but loses small scales.

```
# create the single-dish map by convolving the image with a FWHM=40" kernel
# (this interpretation is much easier than the sharp-edged stuff in fourier
# space because the kernel is created in real space)
lowresfwhm = 40*u.arcsec
singledish_kernel = convolution.Gaussian2DKernel(lowresfwhm/pixscale/(8*np.log(2))**0.5,
                                                 x_size=im.shape[1], y_size=im.shape[0])
singledish_kernel_fft = np.fft.fft2(singledish_kernel)
singledish_im = convolution.convolve_fft(im,
                                         kernel=singledish_kernel,
                                         boundary='fill',
                                         fill_value=im.mean())
```

The single-dish image and its histogram

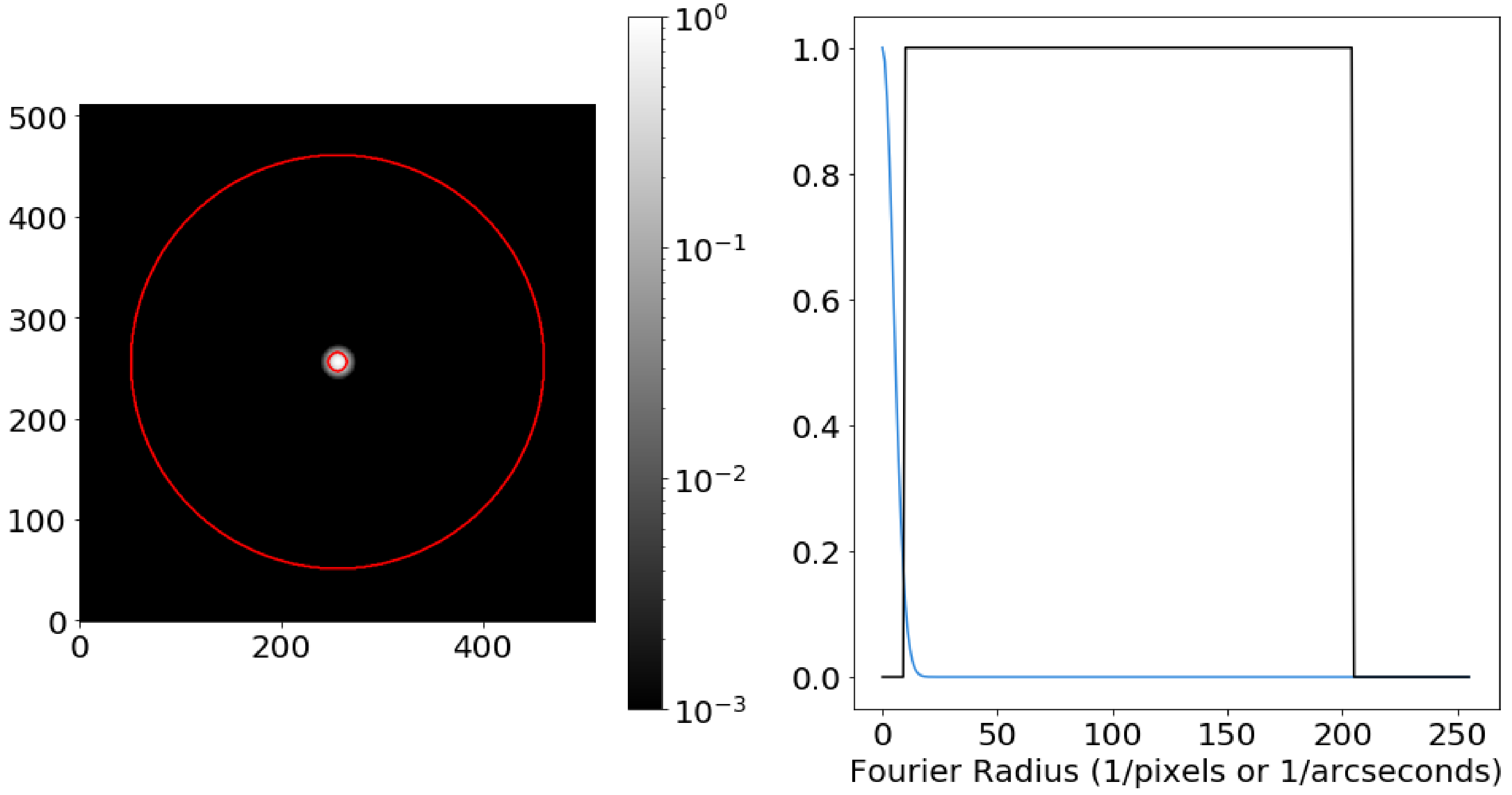


The single dish in Fourier space

We show the single dish beam in Fourier space with the interferometer coverage range overlaid

```
pl.cm.gray.set_under((0,0,0))
pl.cm.gray.set_bad((0,0,0))
```

Single Dish UV coverage map



Feathering

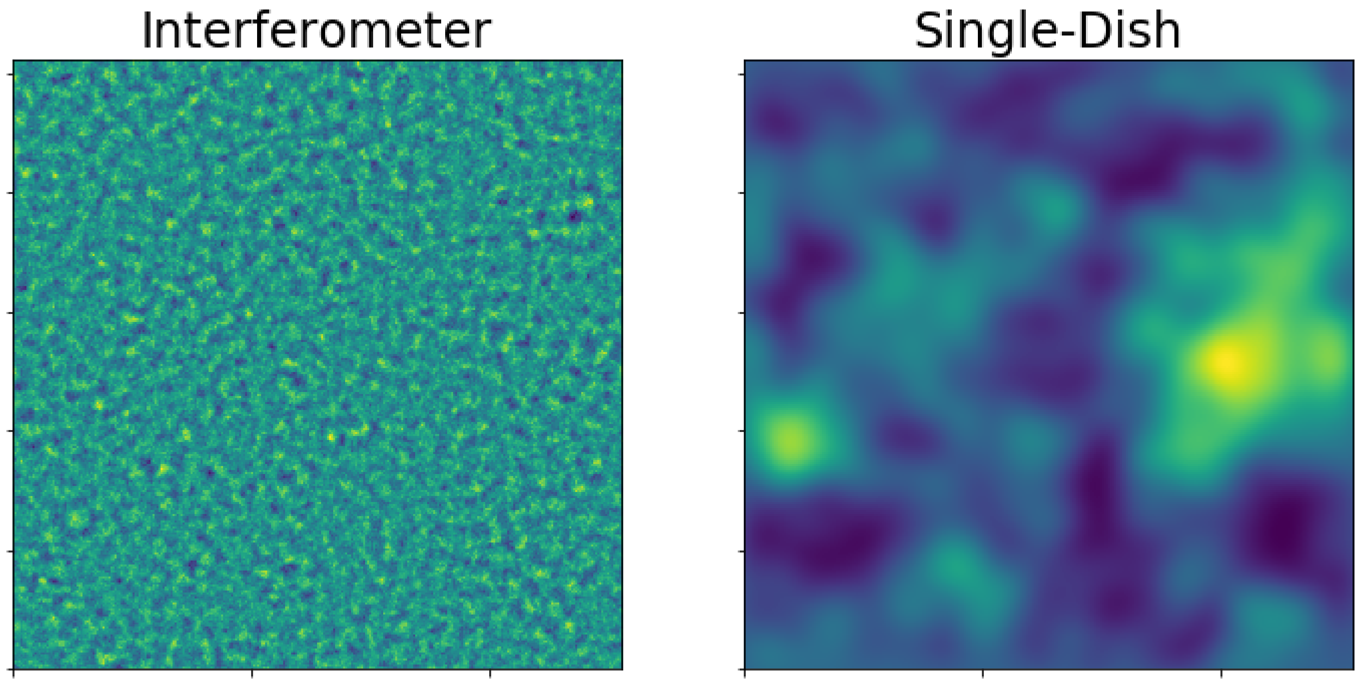
Feathering is the combination of the single-dish image with the interferometric image in the UV domain.

In the `uvcombine` package, this is handled by `uvcombine.feather_simple` (https://github.com/radio-astro-tools/uvcombine/blob/master/uvcombine/uvcombine.py#L751). However, we show the components of that function here.

For comparison, CASA's feather takes these inputs (https://casa.nrao.edu/casadocs/casa-5.6.0/global-task-list/task_feather/about (https://casa.nrao.edu/casadocs/casa-5.6.0/global-task-list/task_feather/about)):

```
#  feather :: Combine two images using their Fourier transforms
imagename           =         ''        #  Name of output feathered image
highres             =         ''        #  Name of high resolution (interferometer) image
lowres              =         ''        #  Name of low resolution (single dish) image
sdfactor            =        1.0        #  Scale factor to apply to Single Dish image
effdishdiam         =       -1.0        #  New effective SingleDish diameter to use in m
lowpassfiltersd     =      False        #  Filter out the high spatial frequencies of the S
D image
```

ASIDE: Proof that CASA’s feather and uvcombine’s feather do the same thing

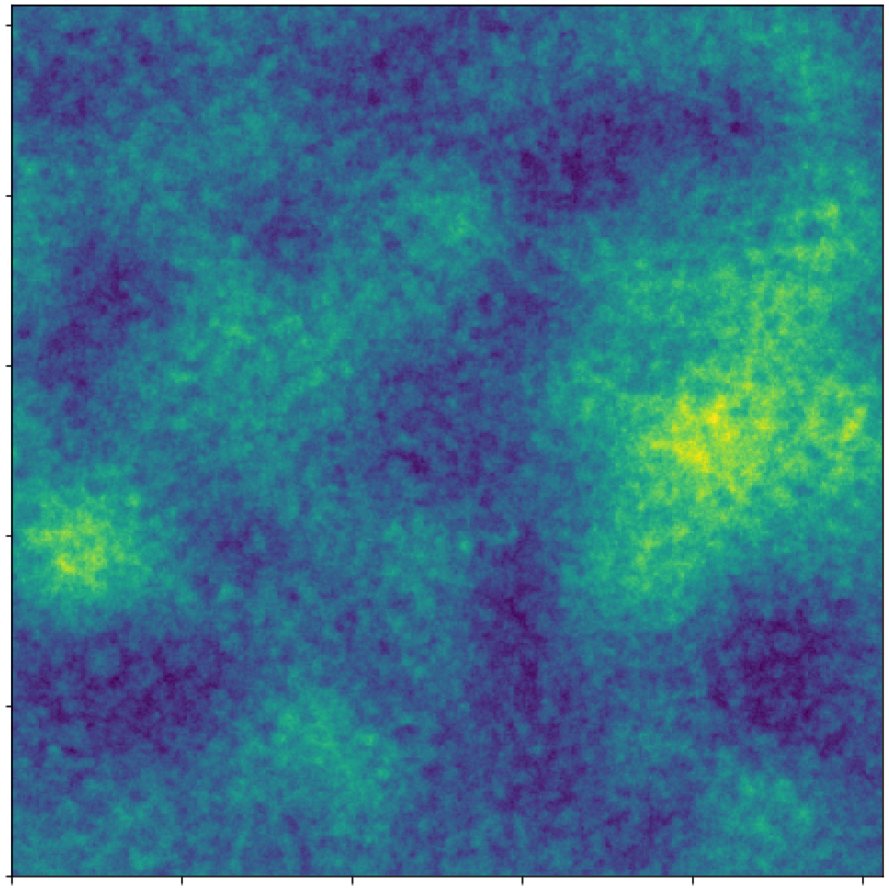

Interferometer
Single-Dish

```python
from casatasks import feather
from casatools import image
ia = image()
```

```
feather(imagename=output,
        highres=input_hires.replace(".fits",".image"),
        lowres=input_lores.replace(".fits",".image"),
        sdfactor=sdfactor,
        lowpassfiltersd=lowpassfilterSD,
       )

_ = ia.open(output)
_ = casa_feather_data = ia.getchunk()
_ = ia.close()
```

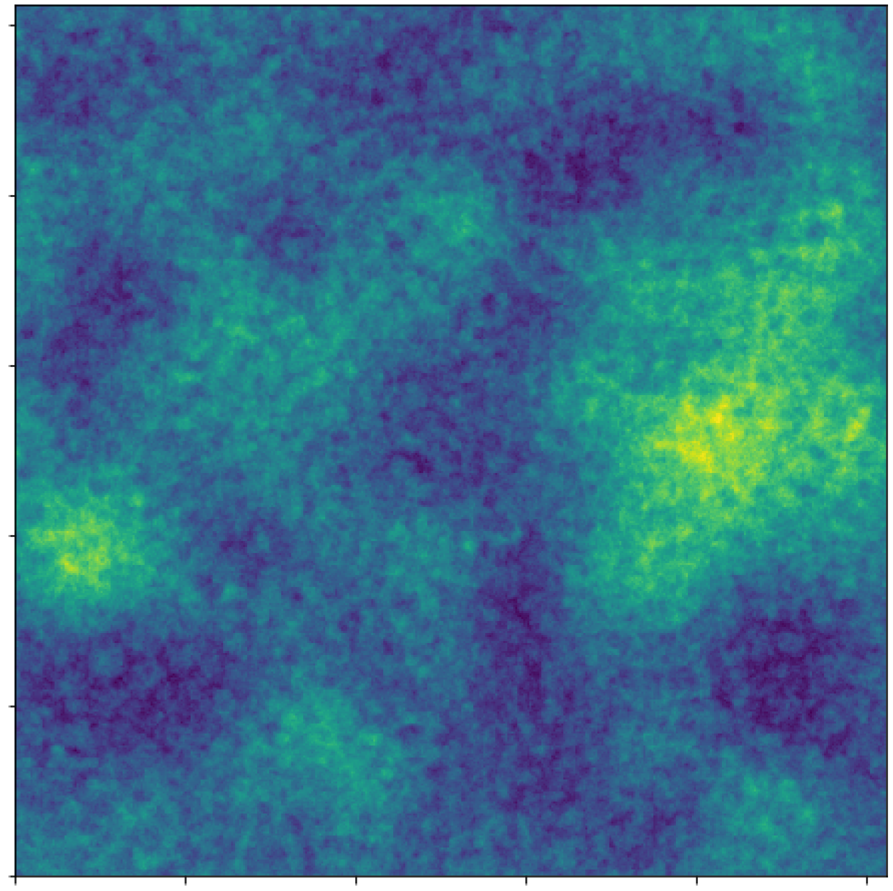
CASA Feather

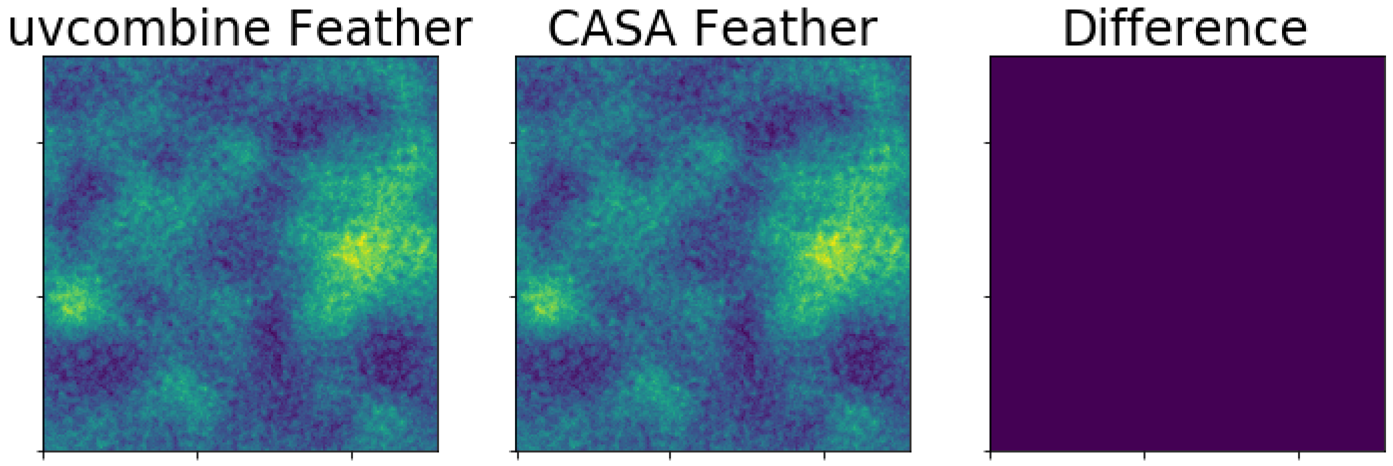


```python
from uvcombine import feather_simple
feathered_hdu = feather_simple(hires=input_hires,
                               lores=input_lores,
                               lowresscalefactor=sdfactor,
                               lowpassfilterSD=lowpassfilterSD,
                               return_hdu=True)
```

```
INFO: Low-res FWHM: 0.009166666666666667 deg [uvcombine.uvcombine]
INFO: Converting data from 34479262.37608193 Jy / sr to 9386979181.888325 Jy /
sr [uvcombine.uvcombine]
```

uvcombine Feather



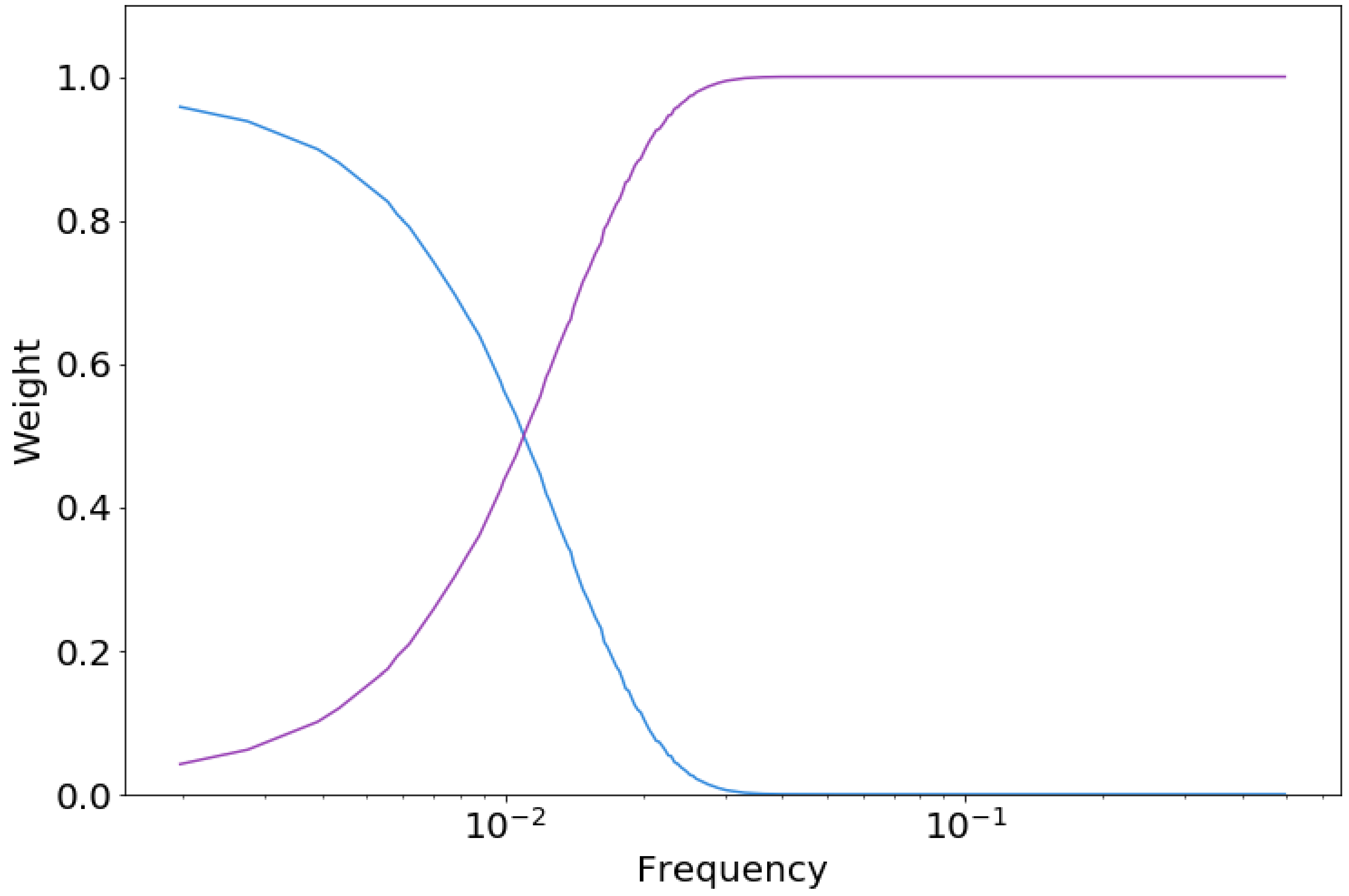

uvcombine Feather
CASA Feather
Difference

First, we define the FWHM of the low-resolution (single-dish) image, which defines the effective cutoff point between the interferometer and the single dish. `lowresfwhm` is equivalent to `effdishdiam` from CASA, albeit in different units.

The kernels are weight arrays for the single-dish and interferometer data. They are the fourier transforms of the low-resolution beam and (1-that kernel), respectively.

```
# pixel scale can be interpreted as "arcseconds"
# then, fwhm=40 means a beam fwhm of 40"
pixscale = 1*u.arcsec
lowresfwhm = 40*u.arcsec
nax1,nax2 = im.shape
kfft, ikfft = feather_kernel(nax2, nax1, lowresfwhm, pixscale,)
```

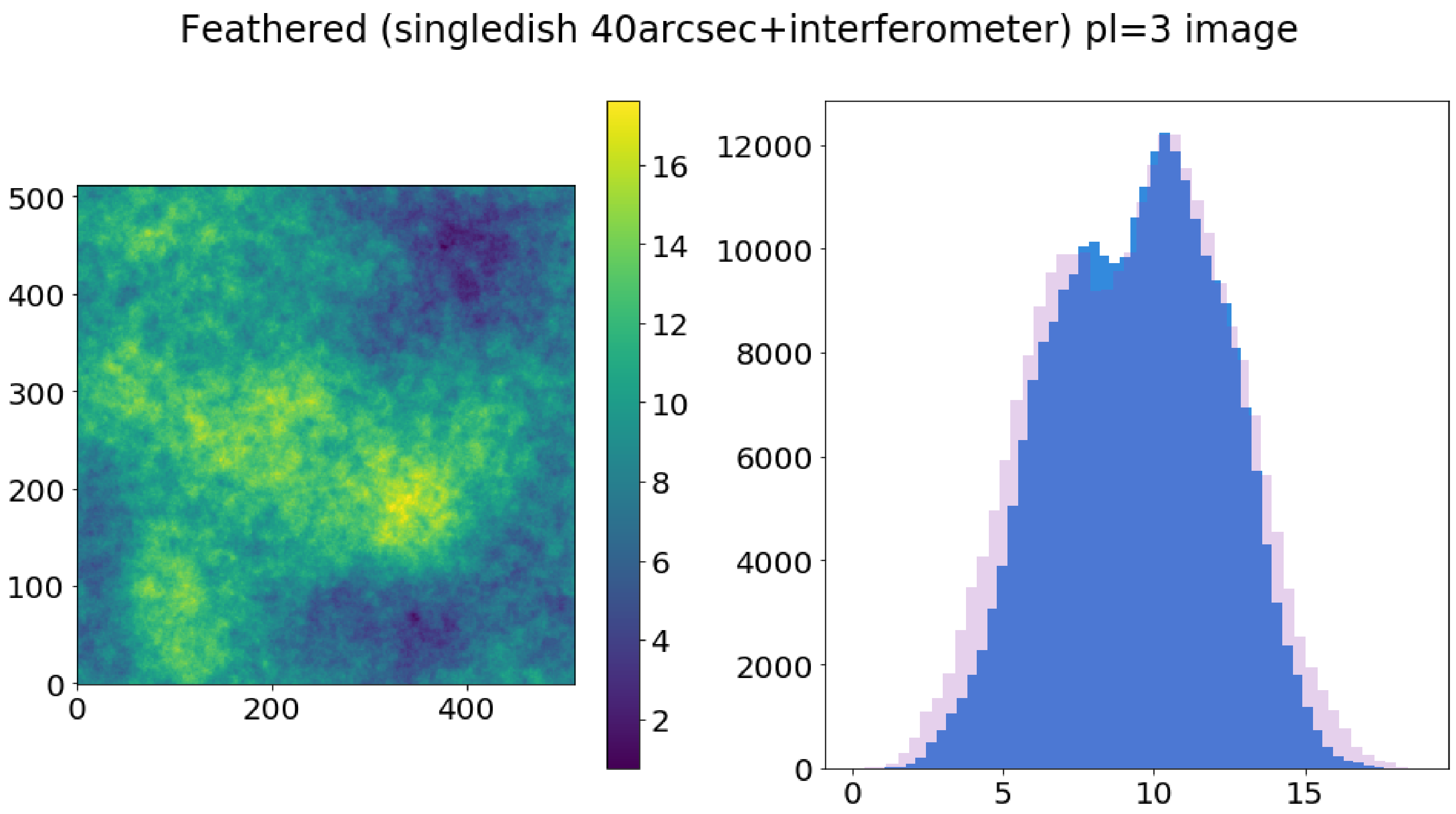
The weight functions for the single-dish image (blue; usually not applied) and the interferometer image (purple). The weighting function is defined entirely by the single dish beam profile.



Then we specify a few parameters that are not all available in CASA. CASA's `lowpassfiltersd` is equivalent to our `replace_hires`, and their `sdfactor` is our `lowresscalefactor`. The other parameters, `highresscalefactor`, `lowpassfilterSD`, and `deconvSD` are unavailable in CASA.

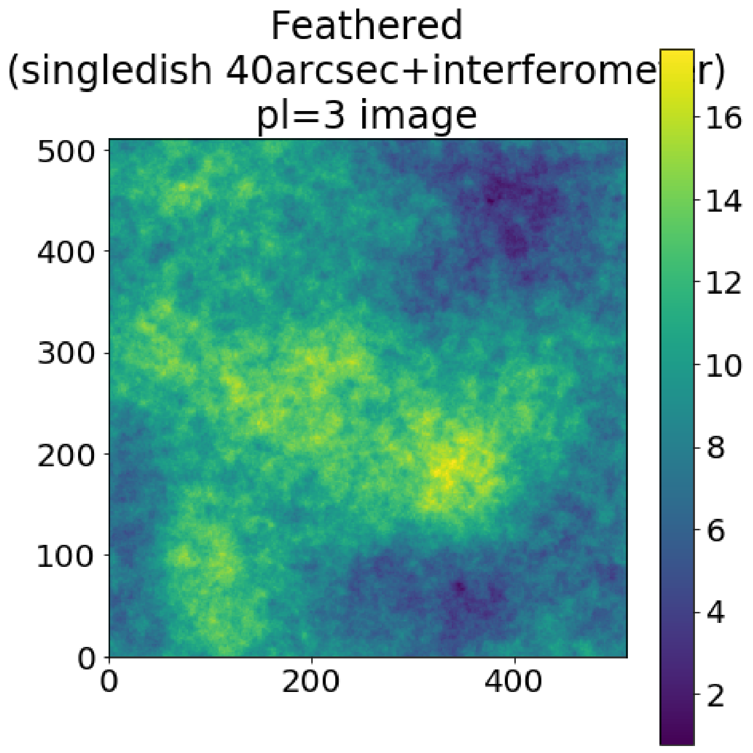
```
# Feather the interferometer and single dish data back together
# This uses the naive assumptions that CASA uses
# However, there are a few flags that can be played with.
# None of them do a whole lot, though there are good theoretical
# reasons to attempt them.
im_hi = im_interferometered.real
im_low = singledish_im
lowresscalefactor=1
replace_hires = False
lowpassfilterSD = False
deconvSD = False
highresscalefactor=1
fftsum, combo = fftmerge(kfft, ikfft, im_hi*highresscalefactor,
                         im_low*lowresscalefactor,
                         replace_hires=replace_hires,
                         lowpassfilterSD=lowpassfilterSD,
                         deconvSD=deconvSD,
                        )
```

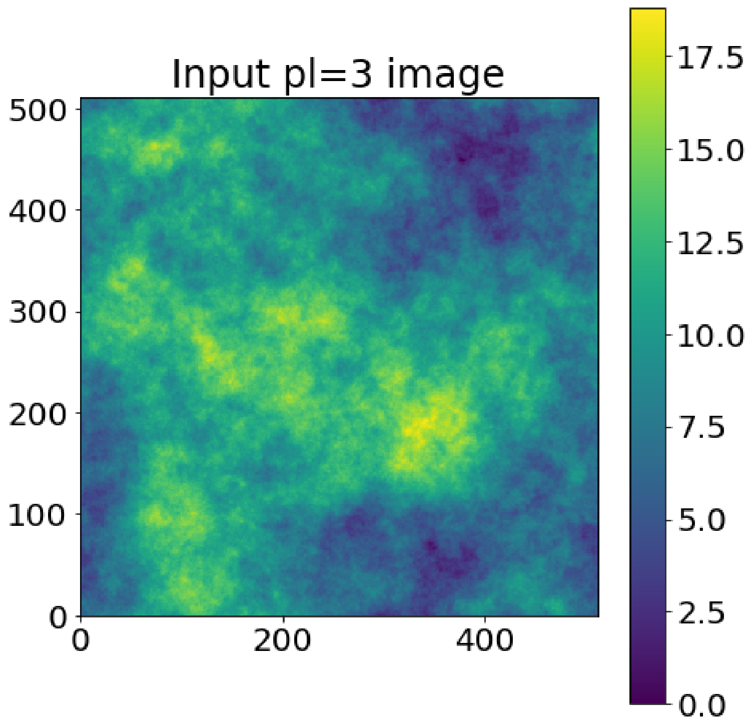
The feathered data set looks pretty good.



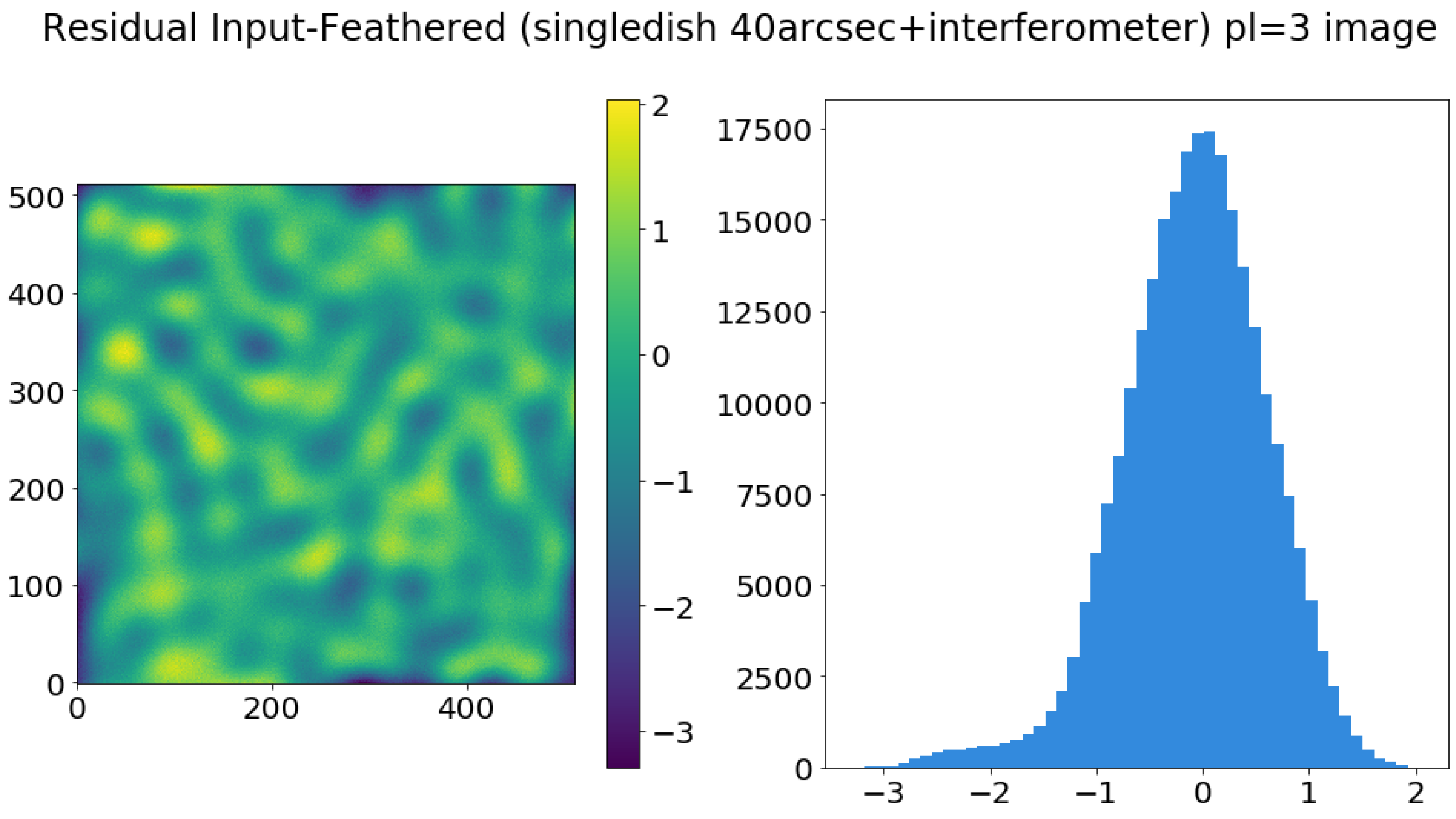
This image looks pretty close to the original, but the peaks and valleys are not recovered (the contrast is reduced compared to the original).

We then compare the feathered data to the input image.





The difference between the input image and the feathered image shows the remainder artifacts.



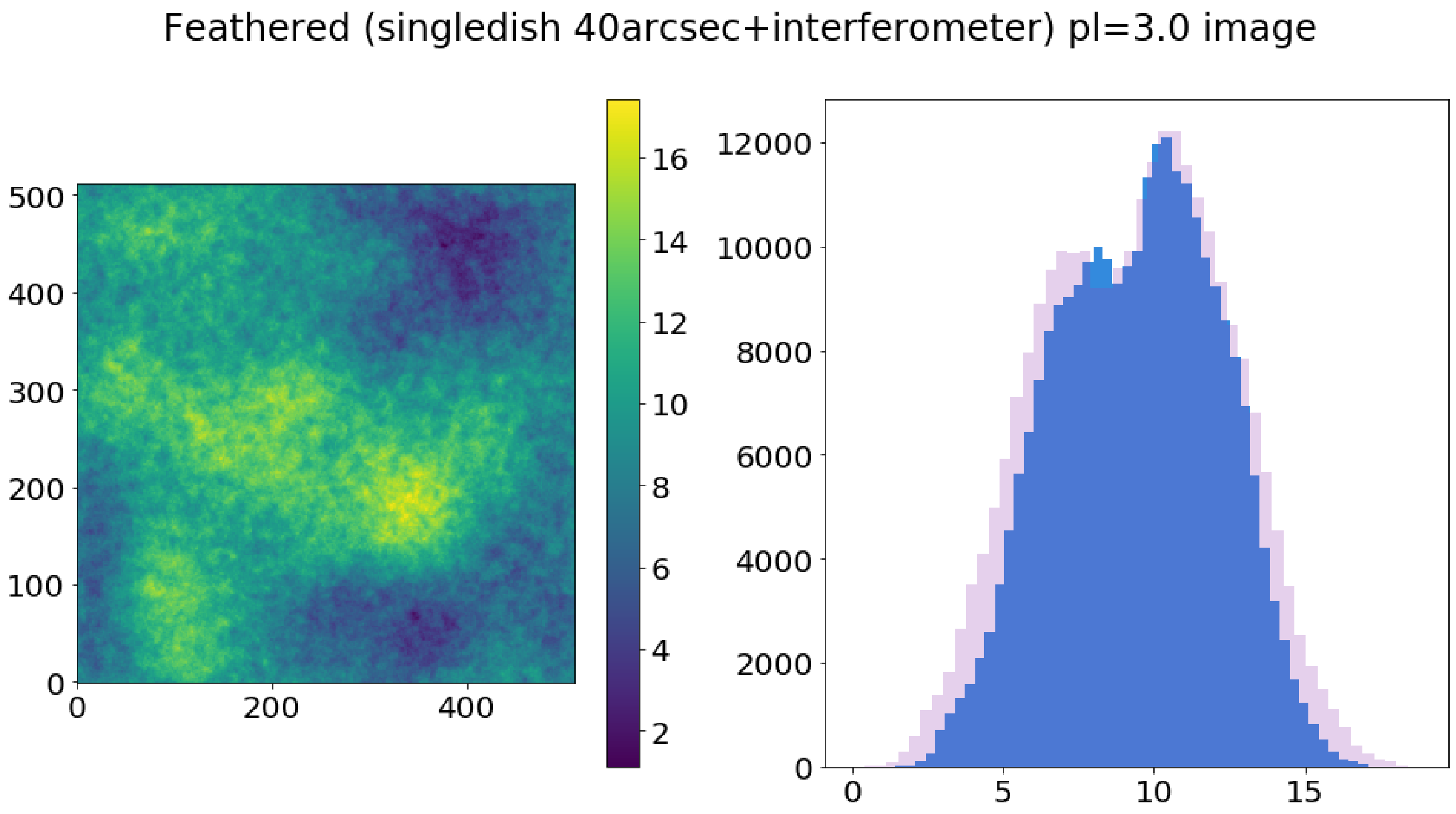
If we repeat the same experiment again, but with the shortest baseline at 12m instead of 9m, the results are noticeably worse:

```
largest_scale = 42.*u.arcsec # (1.22 * 2*u.mm/(12*u.m)).to(u.arcsec, u.dimensionless_angles
())
smallest_scale = 2.5*u.arcsec
image_scale = im.shape[0]*pixscale # assume symmetric (default=256)
ring = (rr>=(image_scale/largest_scale)) & (rr<=(image_scale/smallest_scale))
```

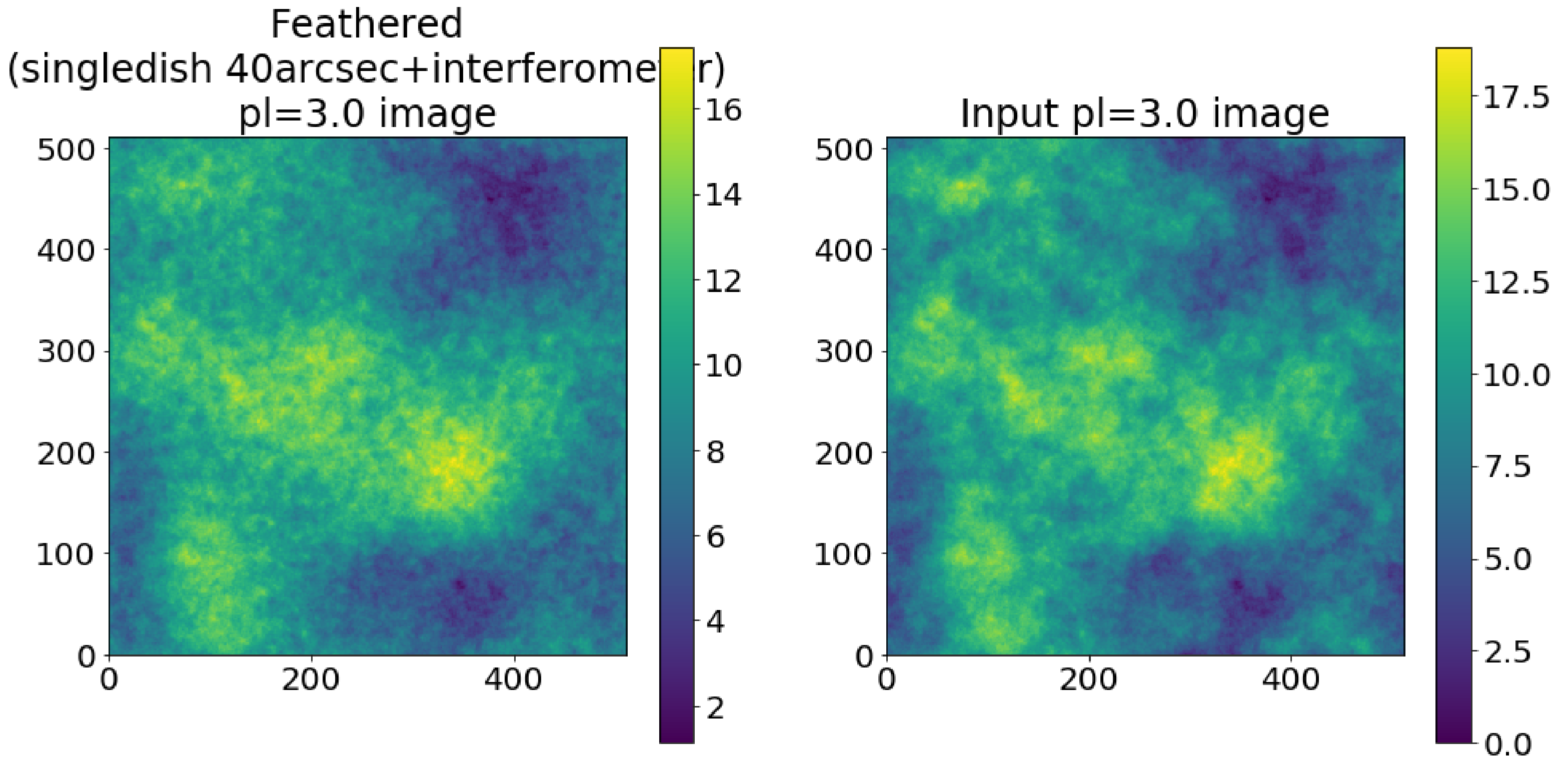
```
# create the interferometric map by removing both large and small angular
# scales in fourier space
imfft = np.fft.fft2(im)
imfft_interferometered = imfft * np.fft.fftshift(ring)
im_interferometered = np.fft.ifft2(imfft_interferometered)
```

```
im_hi = im_interferometered.real
lowresscalefactor=1
replace_hires = False
lowpassfilterSD = False
deconvSD = False
highresscalefactor=1
fftsum, combo = fftmerge(kfft, ikfft, im_hi*highresscalefactor,
                         im_low*lowresscalefactor,
                         replace_hires=replace_hires,
                         lowpassfilterSD=lowpassfilterSD,
                         deconvSD=deconvSD,
                        )
```

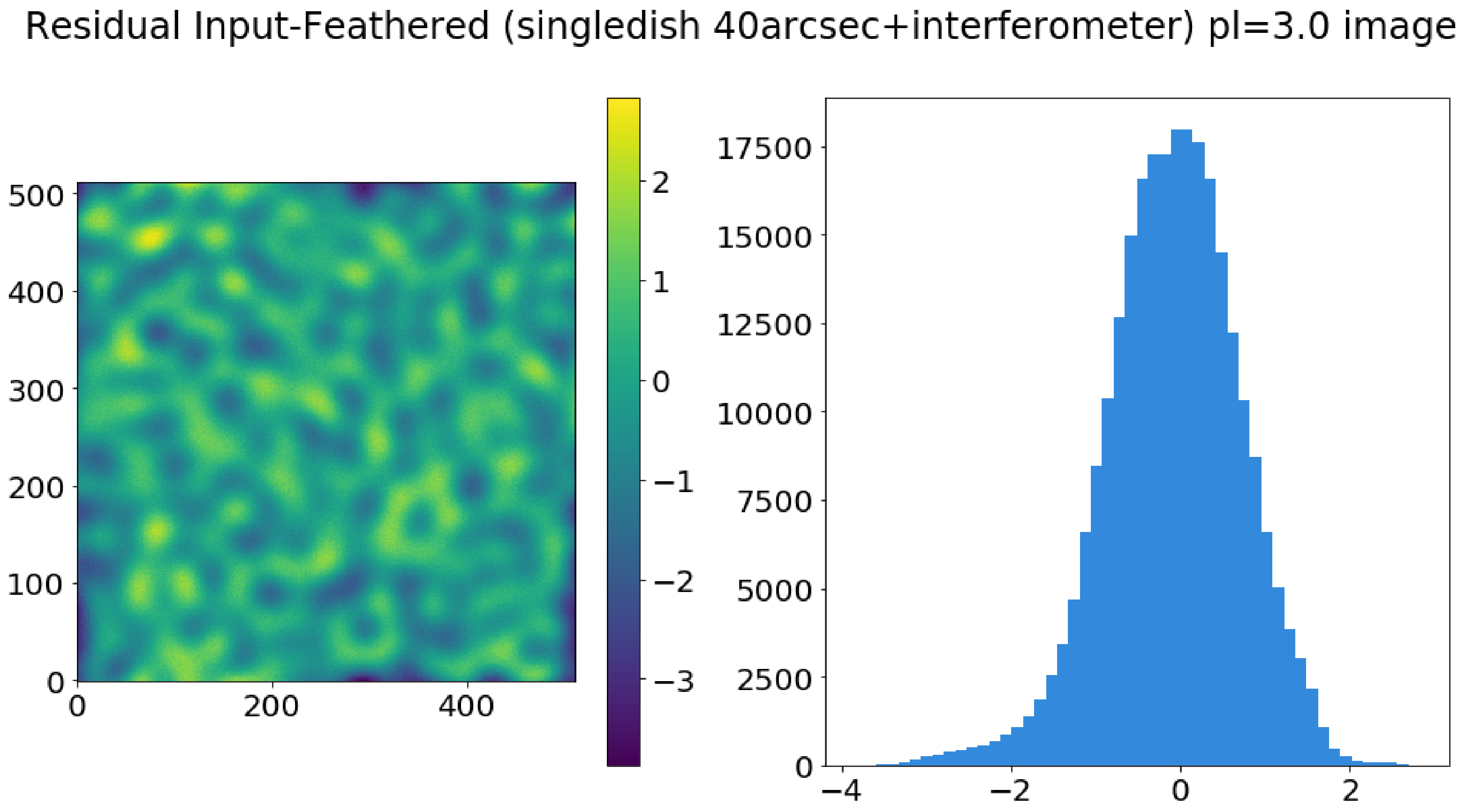
The feathered image with 12m shortest baselines instead of 9m



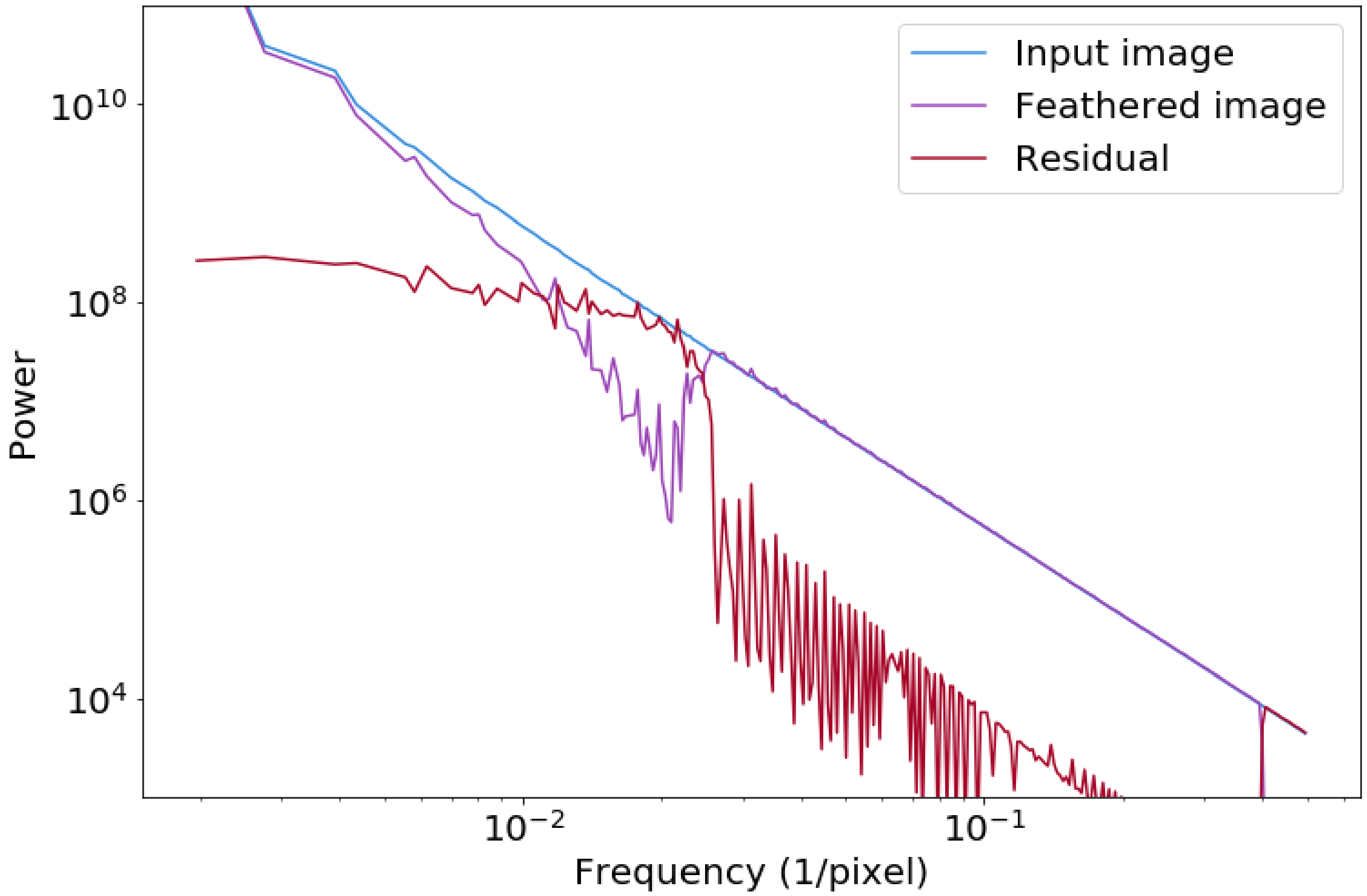
The side-by-side images comparison again, but with 12m instead of 9m shortest baselines



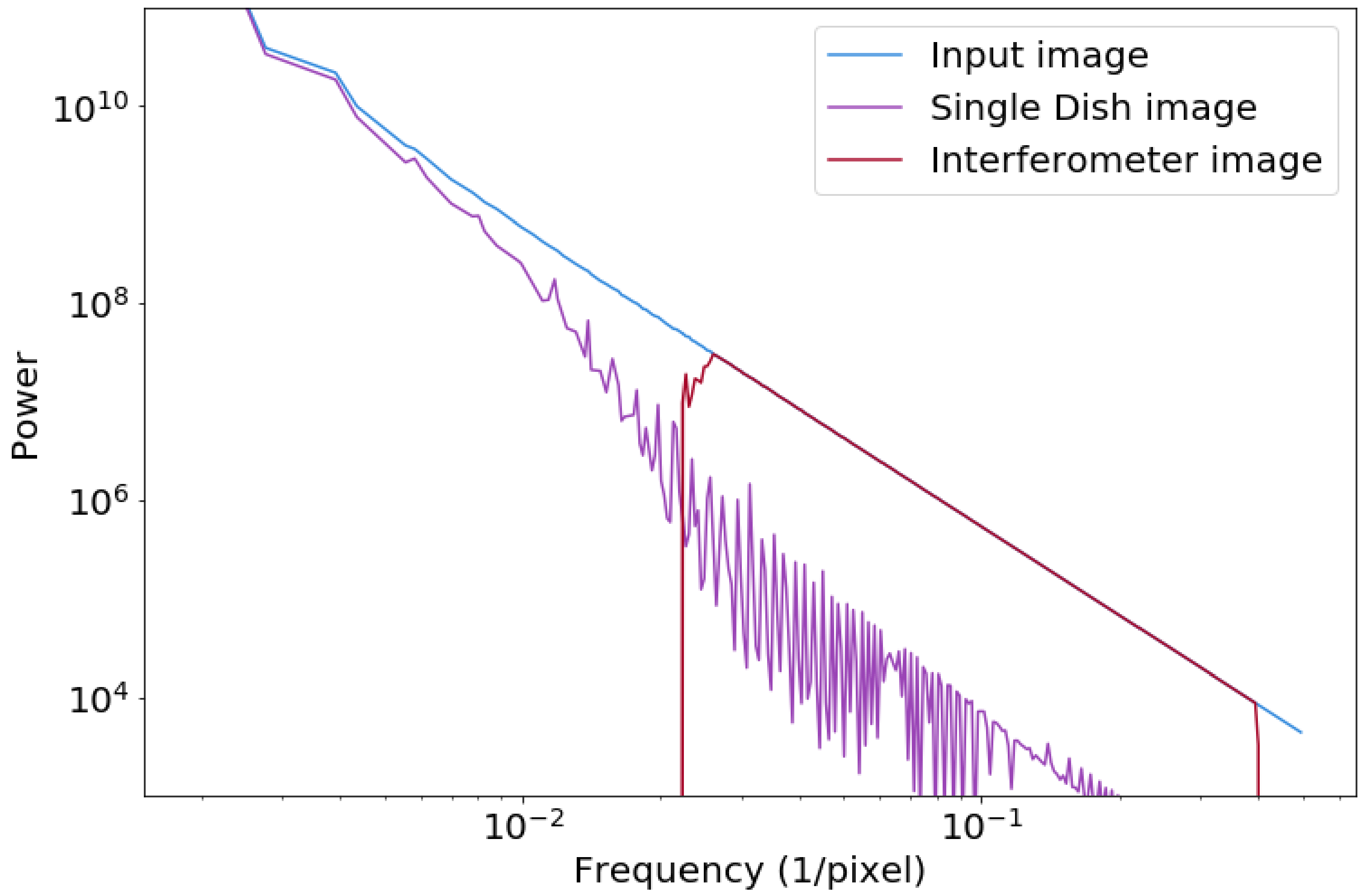
The difference image (original - feathered w/12m shortest baselines)



It is more helpful to look at the difference in power spectra. Note that the axes are log-scaled.



The components that go in to the feather can help clarify the picture



Different combinations of parameters can yield very different results.

We have 8 parameter combinations:

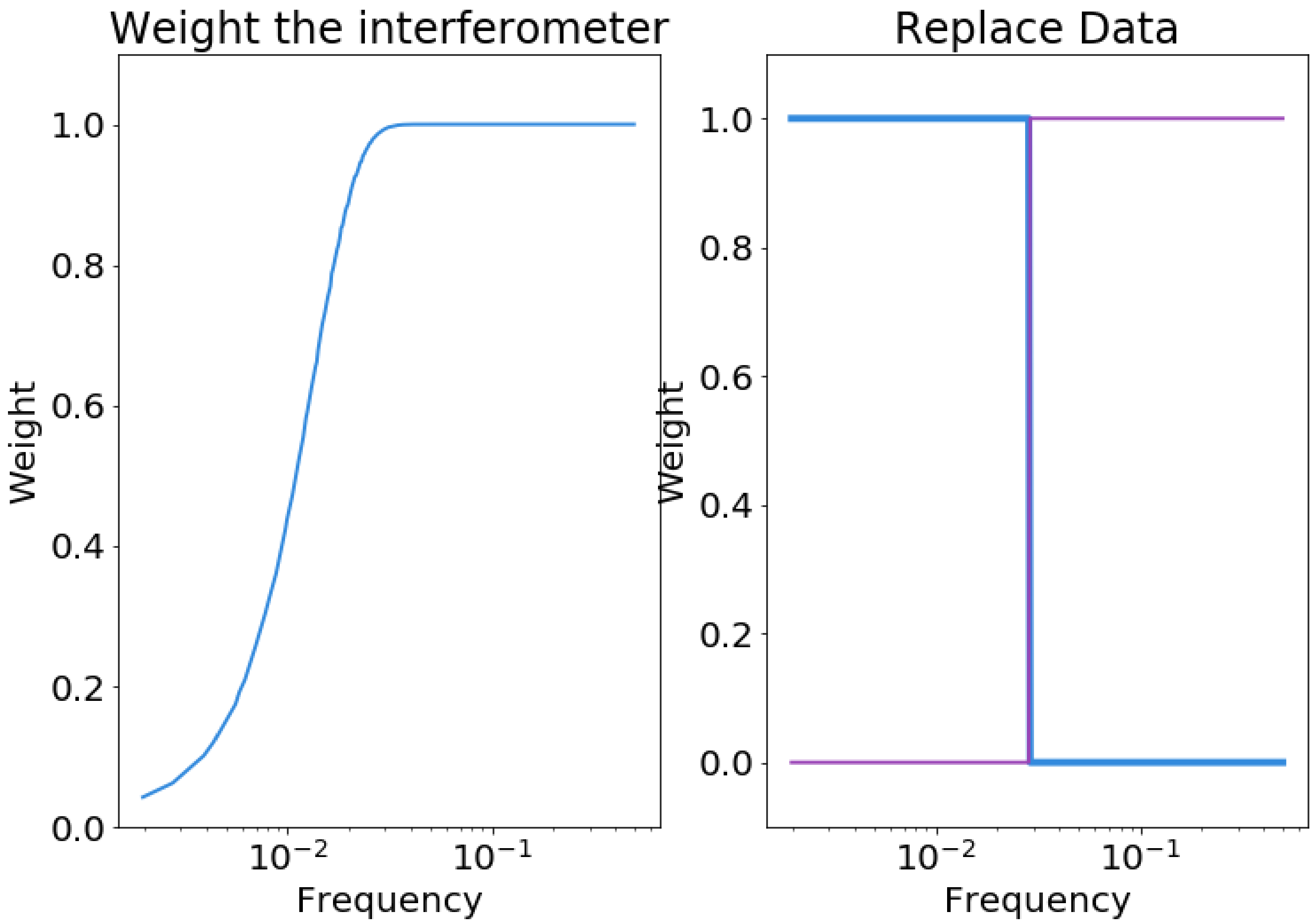
- Replace singledish-interferometer overlap w/interferometer, or average them
- Filter out the small angular scales from the single dish or don't
- Deconvolve the single dish (direct deconvolution) or don't

Replace vs Average

`replace_hires` in uvcombine, `lowpassfiltersd` in CASA

This approach may be useful if there is a lot of overlap between the interferometer & single-dish and the interferometer image is considered much more reliable.

Replace vs Average

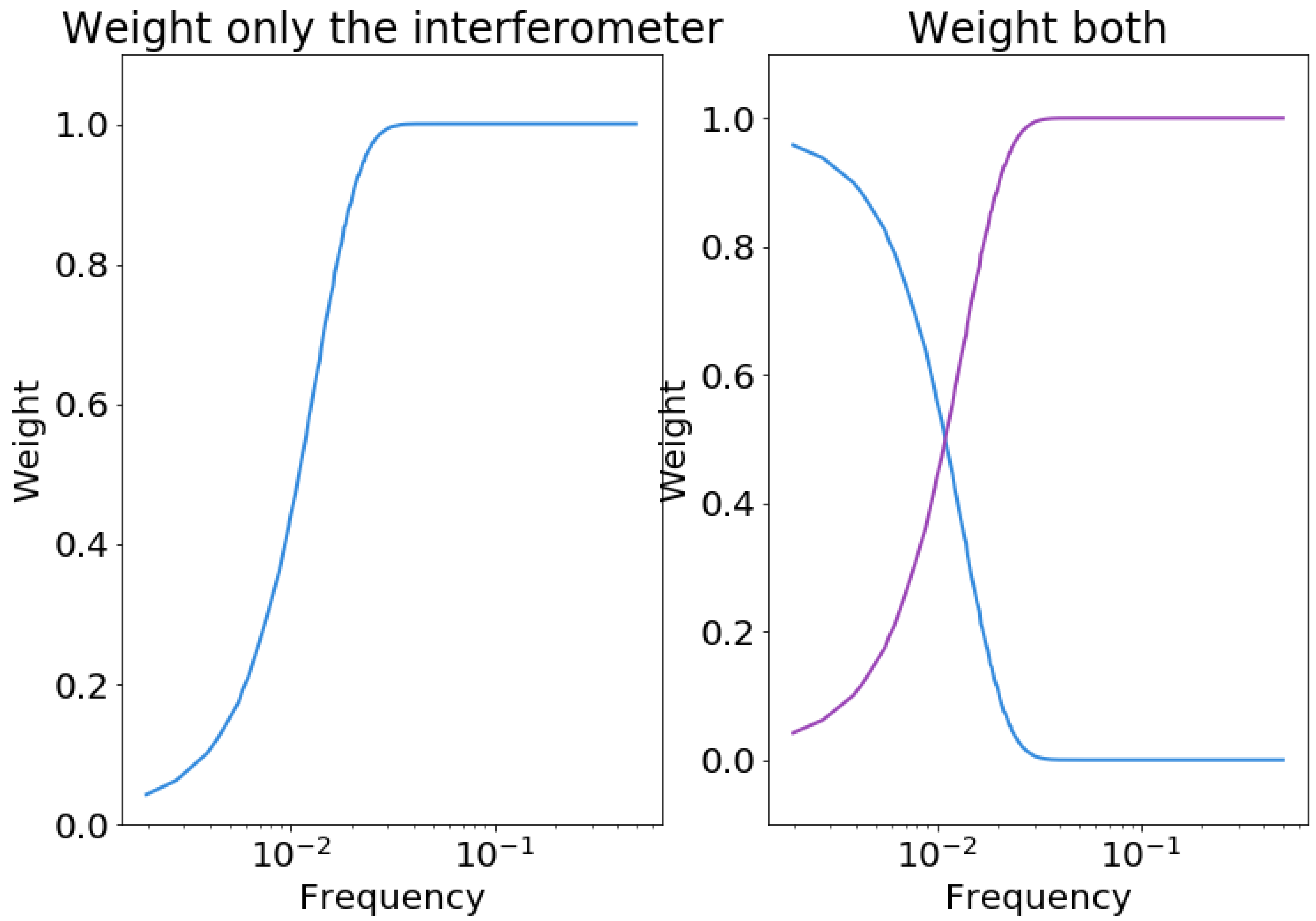


Filter out the small angular scales from the single dish or don't

`lowpassfilterSD` in uvcombine, unavailable in CASA.

The filter function for the single-dish is usually not applied, as it is assumed to have already been applied during the observation (the filter function *is* the single dish PSF / main beam).

Filter out the small angular scales from the single dish or don't



Deconvolve the single dish (direct deconvolution) or don't

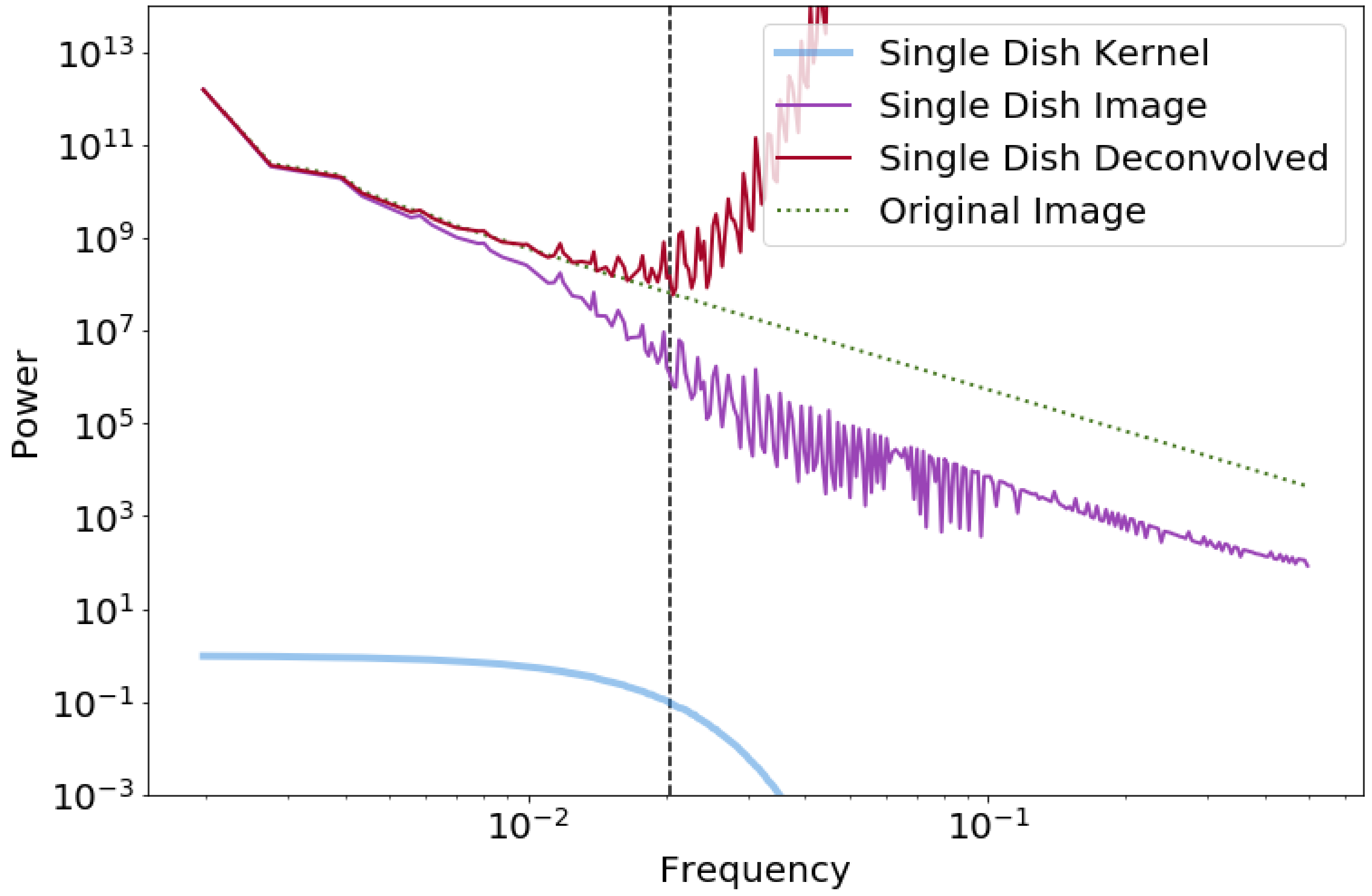
`deconvSD` in uvcombine, unavailable in CASA

Perhaps the most relevant parameter, one can deconvolve the single-dish data prior to deconvolution. This approach is a direct, frequency-space deconvolution, i.e., the data are divided by the kernel in the fourier domain.

Direct deconvolution is problematic, as we are trying to undo a multiply-by-zero operation with a divide-by-zero operation; as a result, the deconvolution diverges at high frequencies (small scales). We therefore do not include data below the `min_beam_fraction=0.1` by default.

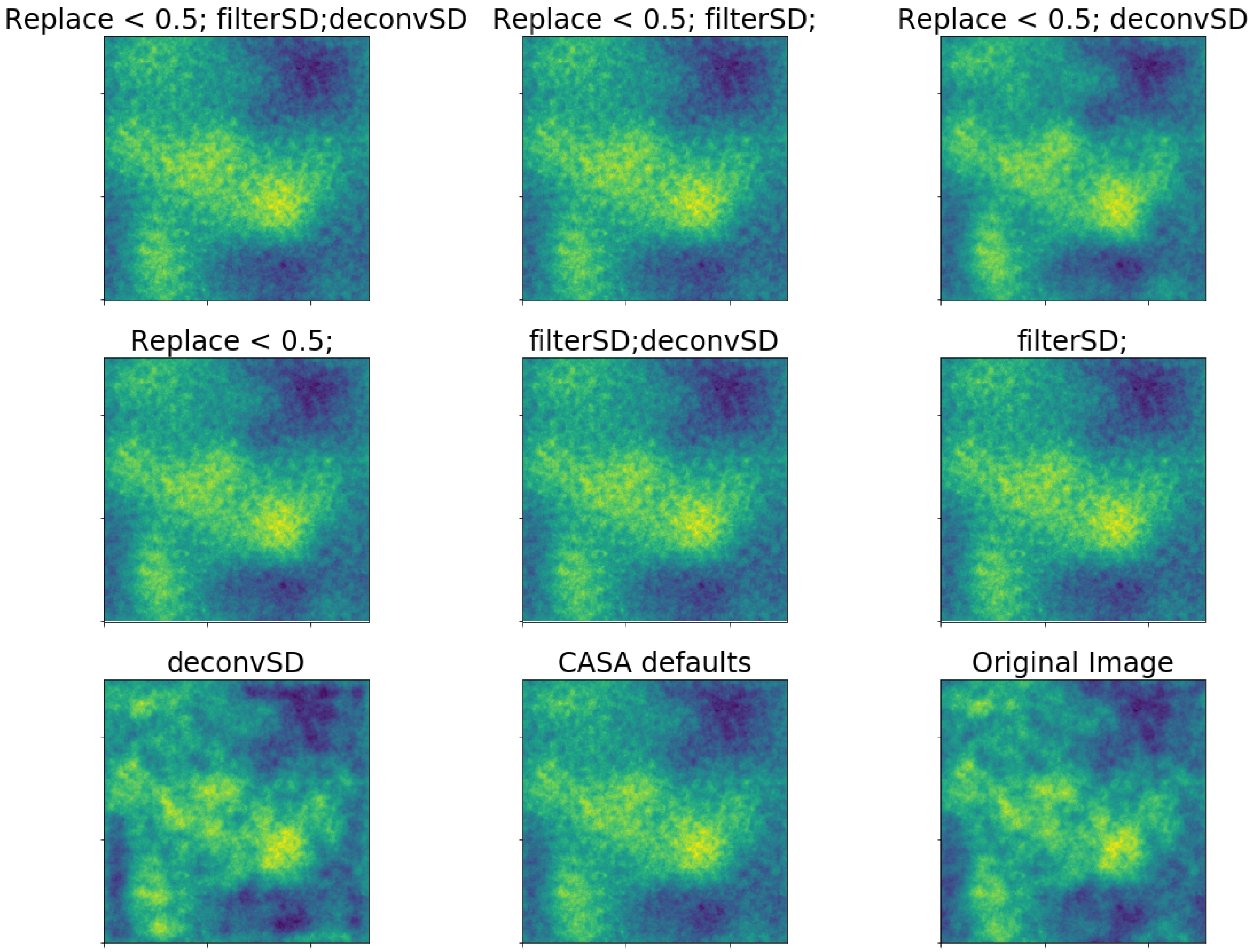
AFAIK, this method is unavailable in CASA's `feather` task, but it appears to be available in the C code using something called `GaussianDeconvolver` (see an old blog post, http://keflavich.github.io/blog/what-does-feather-do.html (http://keflavich.github.io/blog/what-does-feather-do.html)).

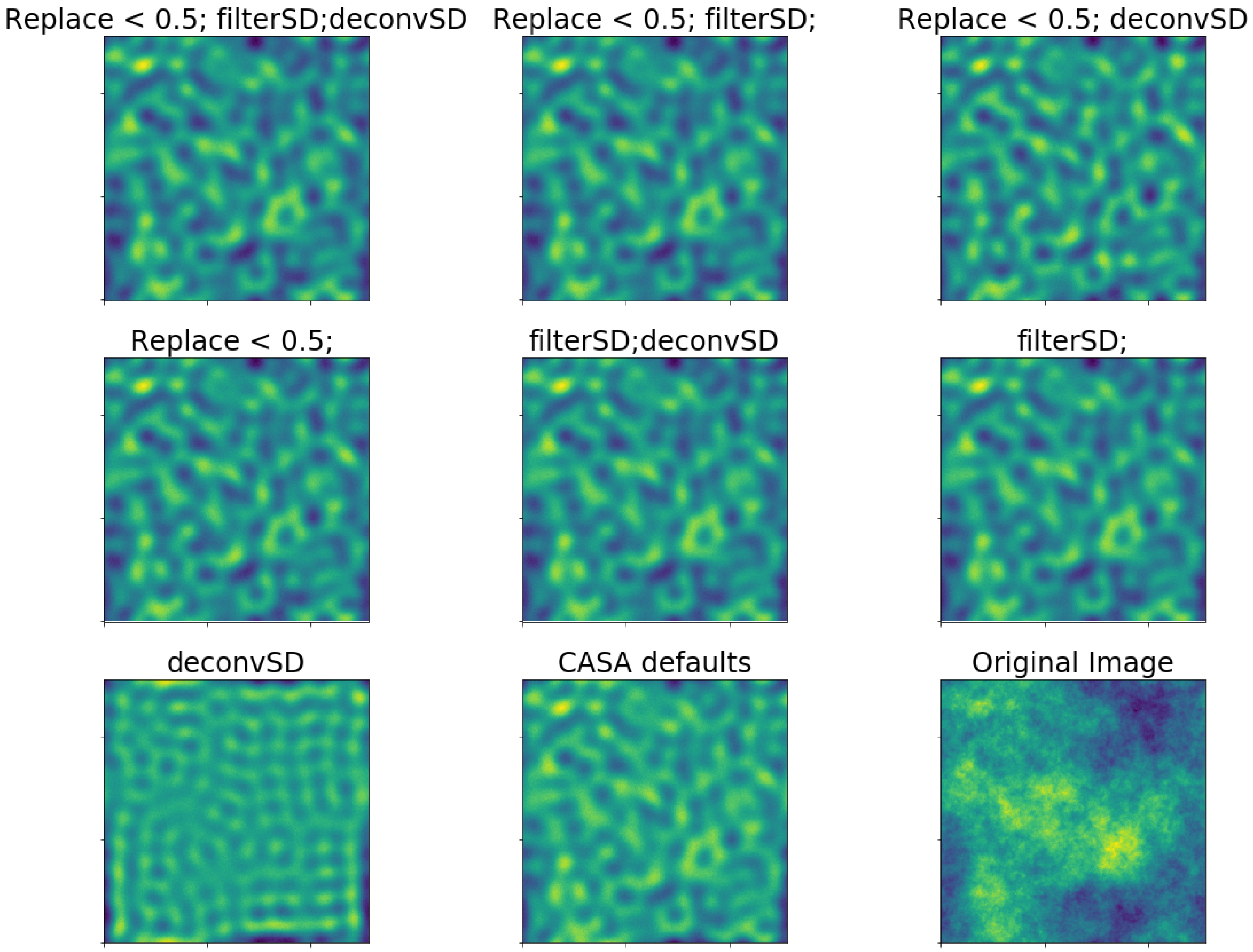
Deconvolve the single dish (direct deconvolution) or don’t



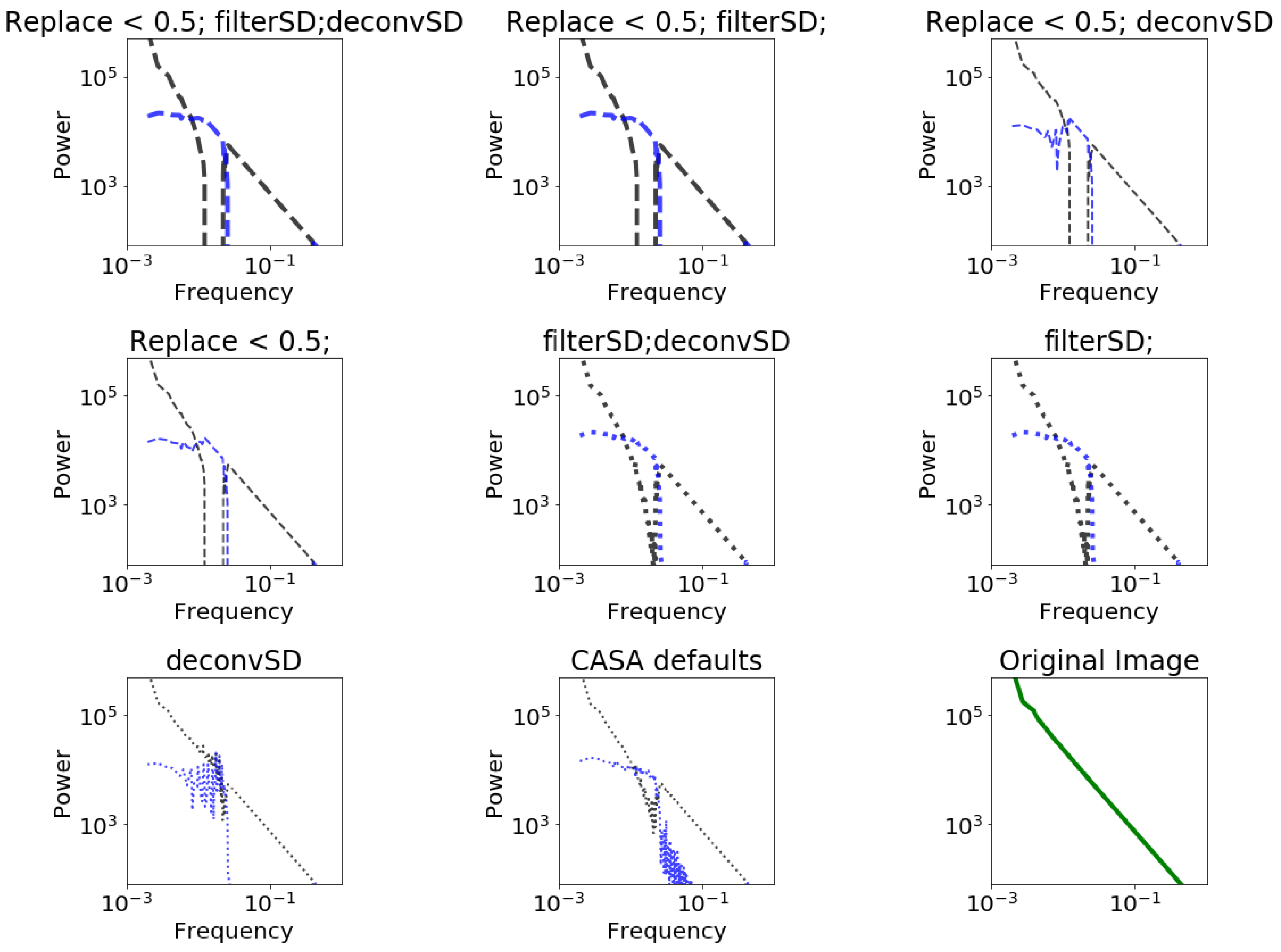
The next slides are a walkthrough of parameter exploration with different effective dish sizes.

First, we have a 12m single-dish combined with an interferometer with shortest baseline length of 12m.


Replace < 0.5; filterSD;deconvSD
Replace < 0.5; filterSD;
Replace < 0.5; deconvSD
Replace < 0.5;
filterSD;deconvSD
filterSD;
deconvSD
CASA defaults
Original Image


Replace < 0.5; filterSD;deconvSD
Replace < 0.5; filterSD;
Replace < 0.5; deconvSD
Replace < 0.5;
filterSD;deconvSD
filterSD;
deconvSD
CASA defaults
Original Image

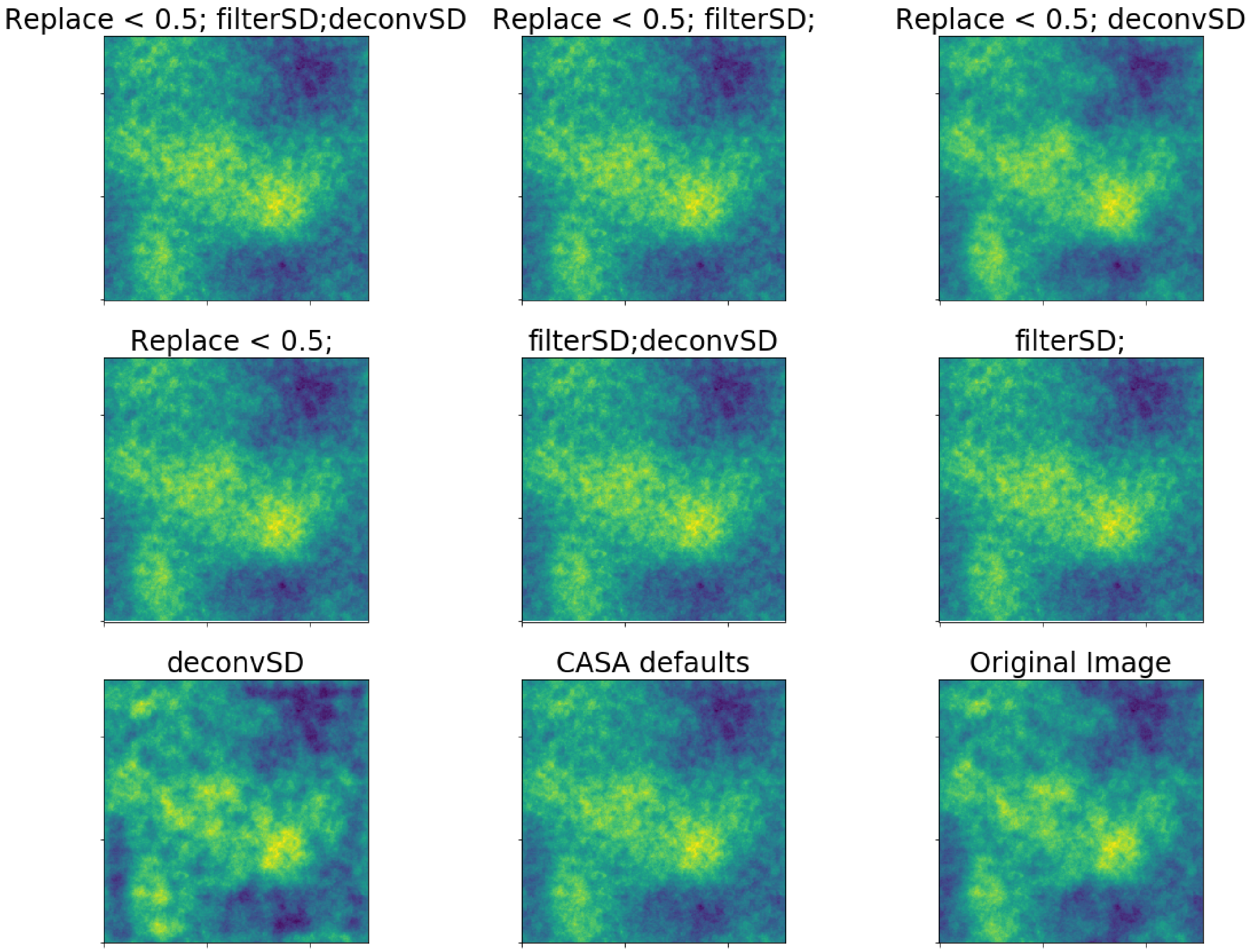
Residuals are in blue.

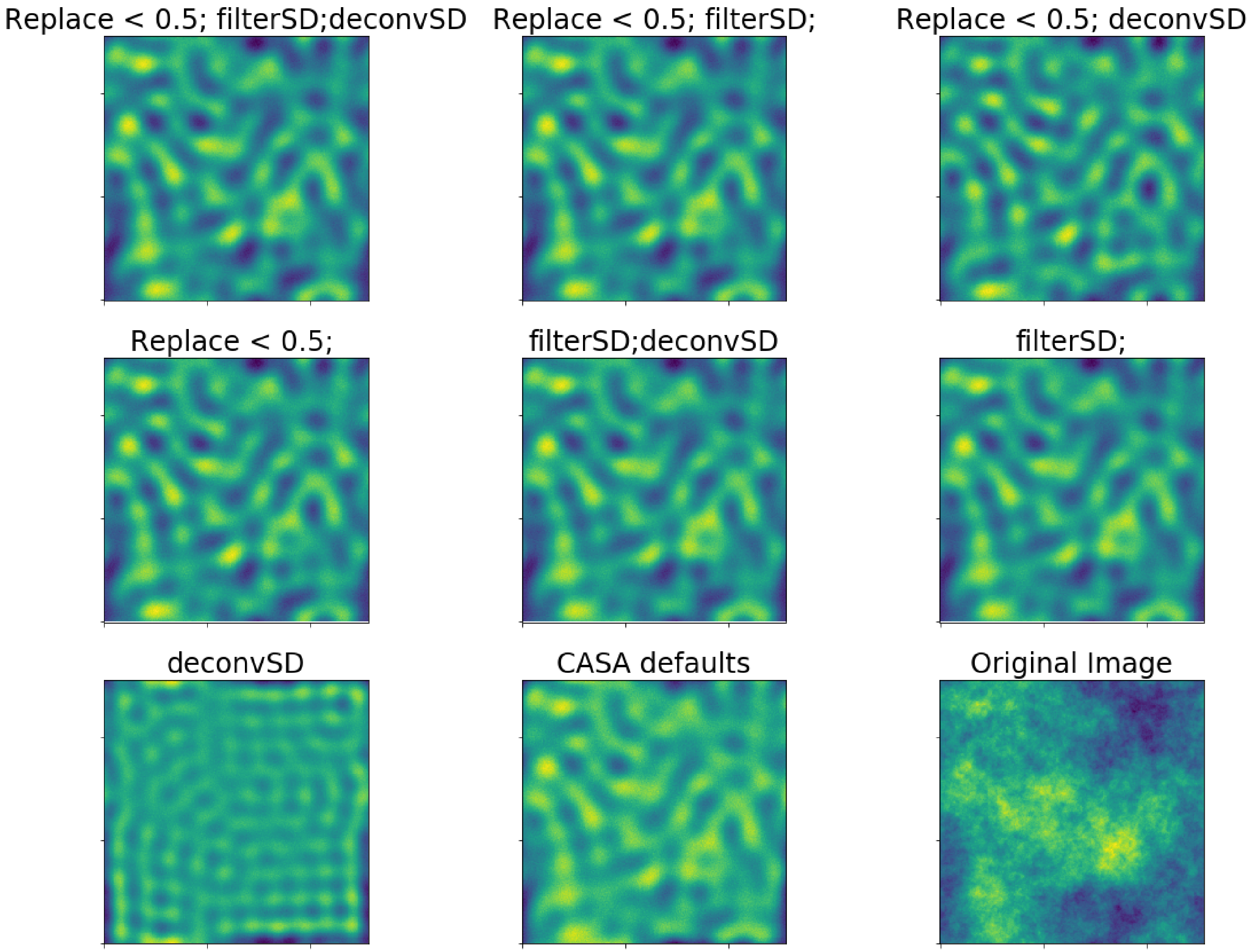


A short-spacing limit of 12m for the interferometer and dish size of 12m is a bad case, but not the worst.

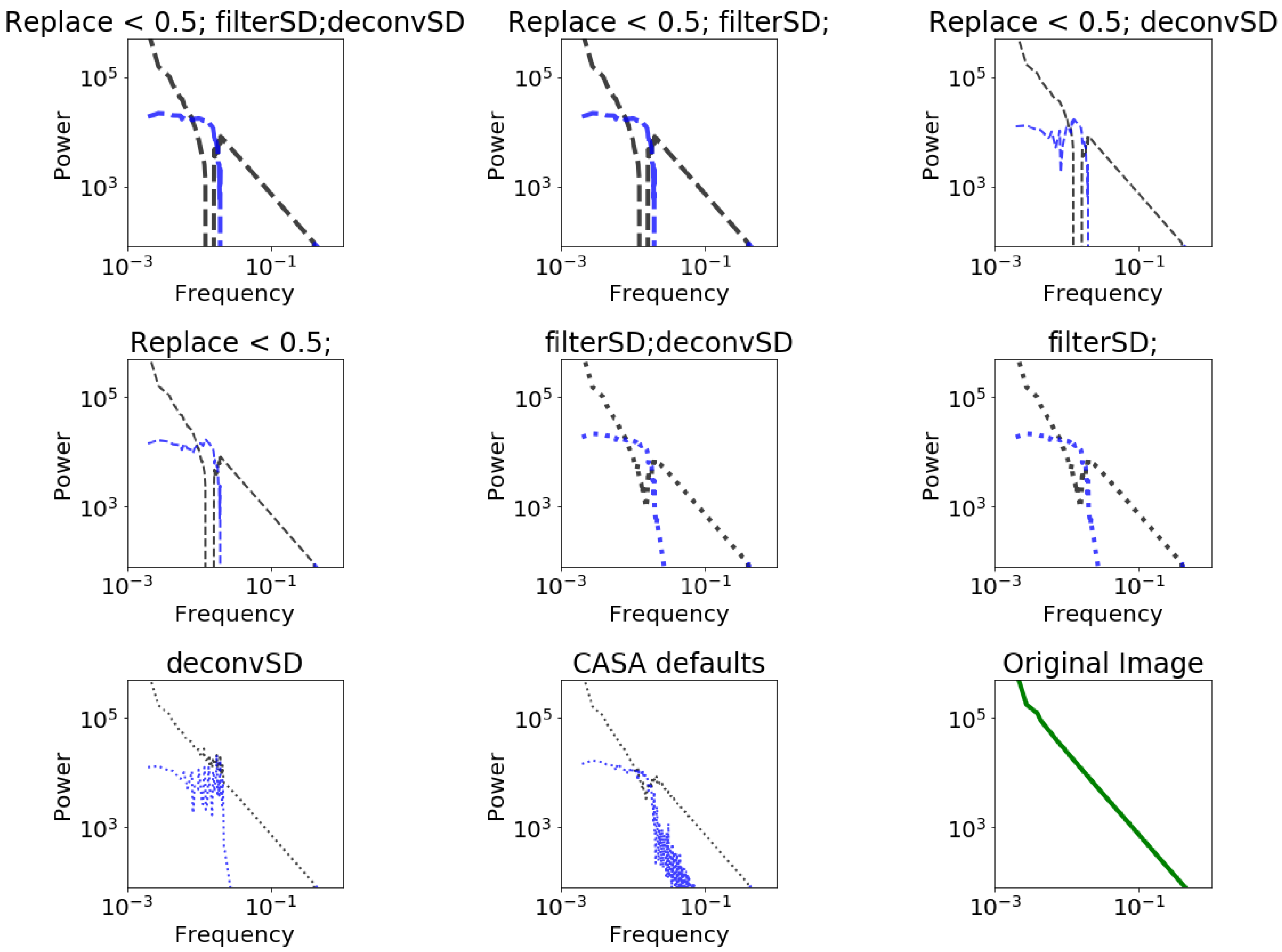
ALMA's shortest baselines are 14.6m (main array) and 8.7m (7m array). L05, the 5th percentile baseline length, is 21.4m (9.1m) for the 12m (7m) array.

A realistic case for ALMA is then a 9m shortest baseline and a 12m effective single dish.

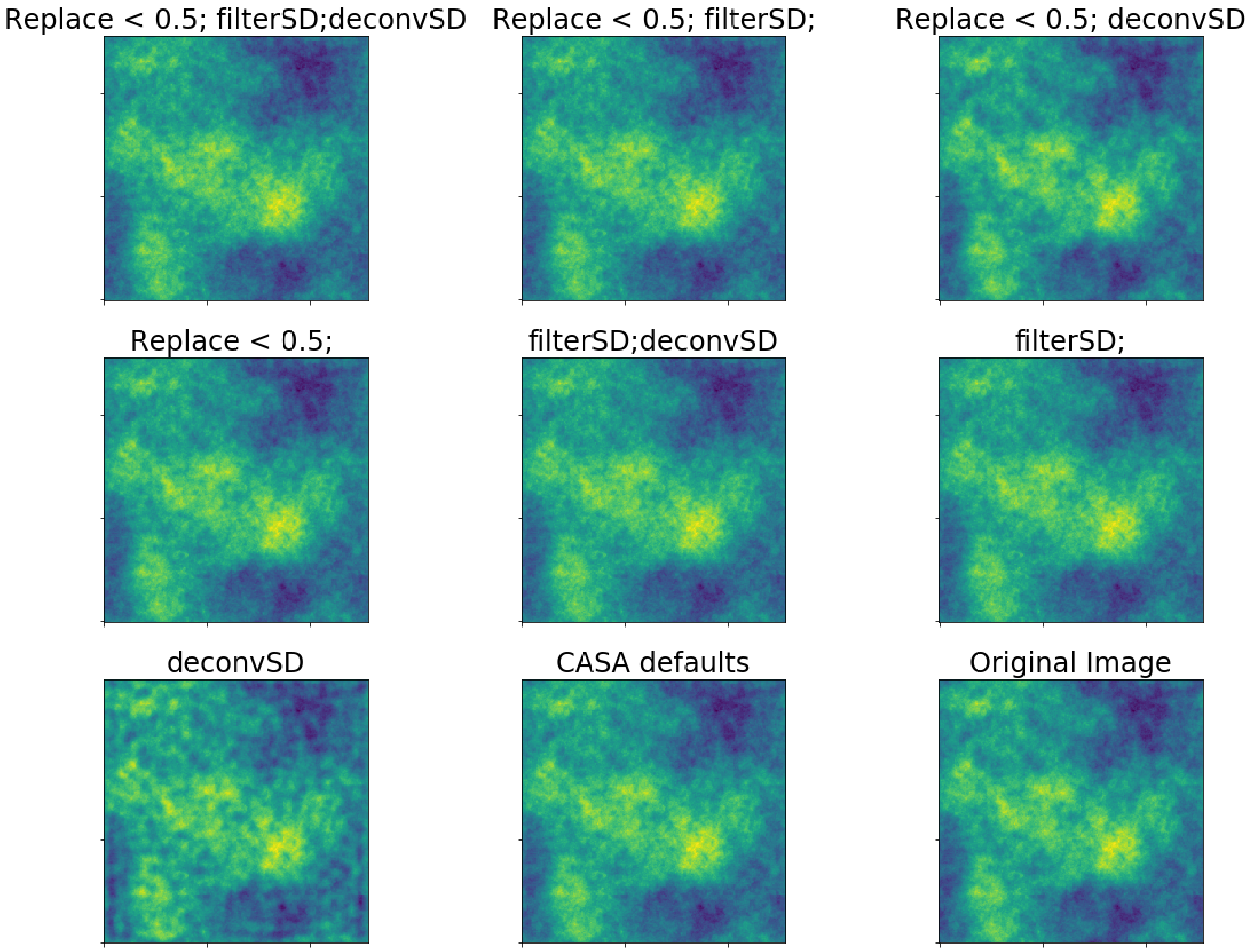

Replace < 0.5; filterSD;deconvSD
Replace < 0.5; filterSD;
Replace < 0.5; deconvSD
Replace < 0.5;
filterSD;deconvSD
filterSD;
deconvSD
CASA defaults
Original Image

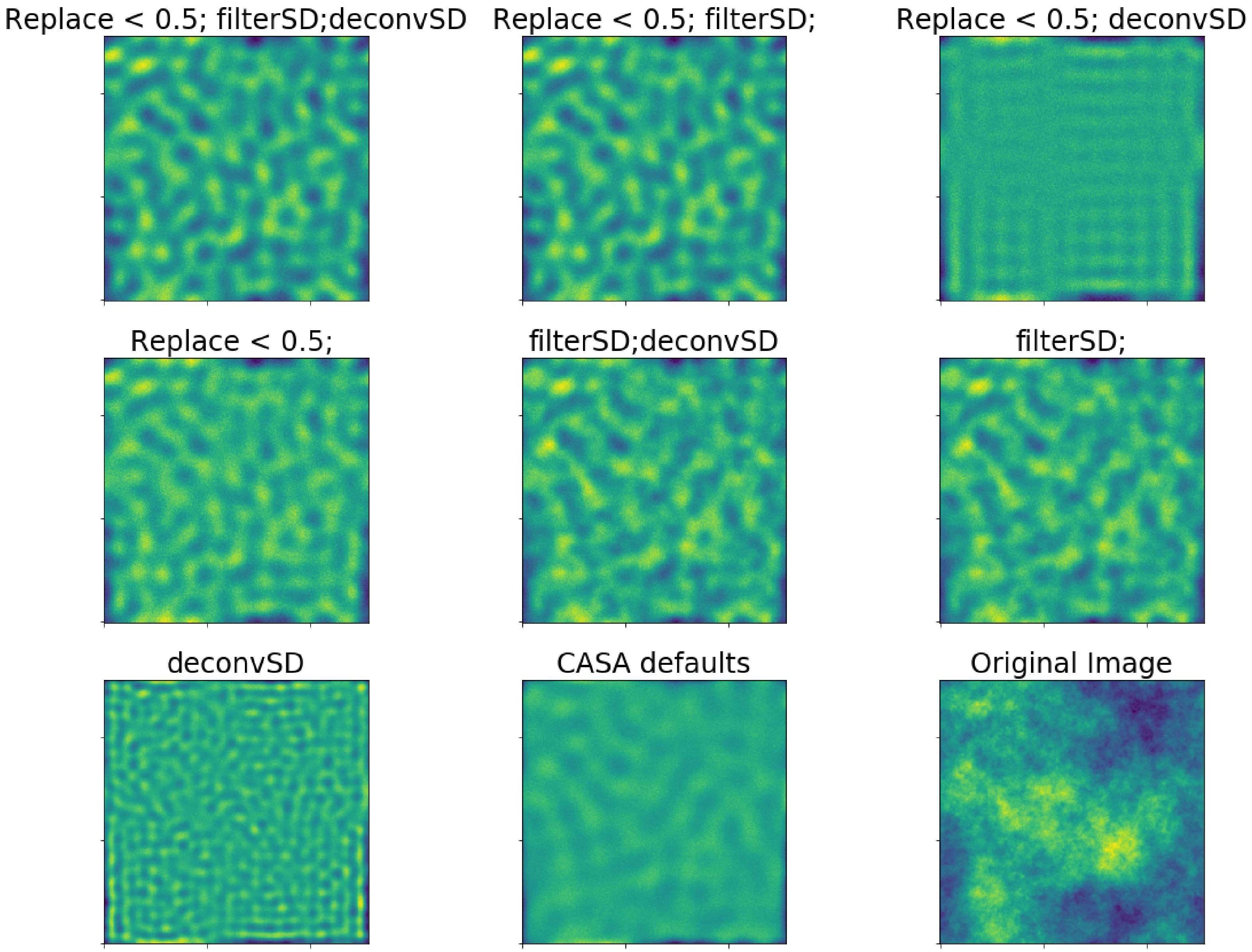

Replace < 0.5; filterSD;deconvSD
Replace < 0.5; filterSD;
Replace < 0.5; deconvSD
Replace < 0.5;
filterSD;deconvSD
filterSD;
deconvSD
CASA defaults
Original Image

Residuals are in blue.

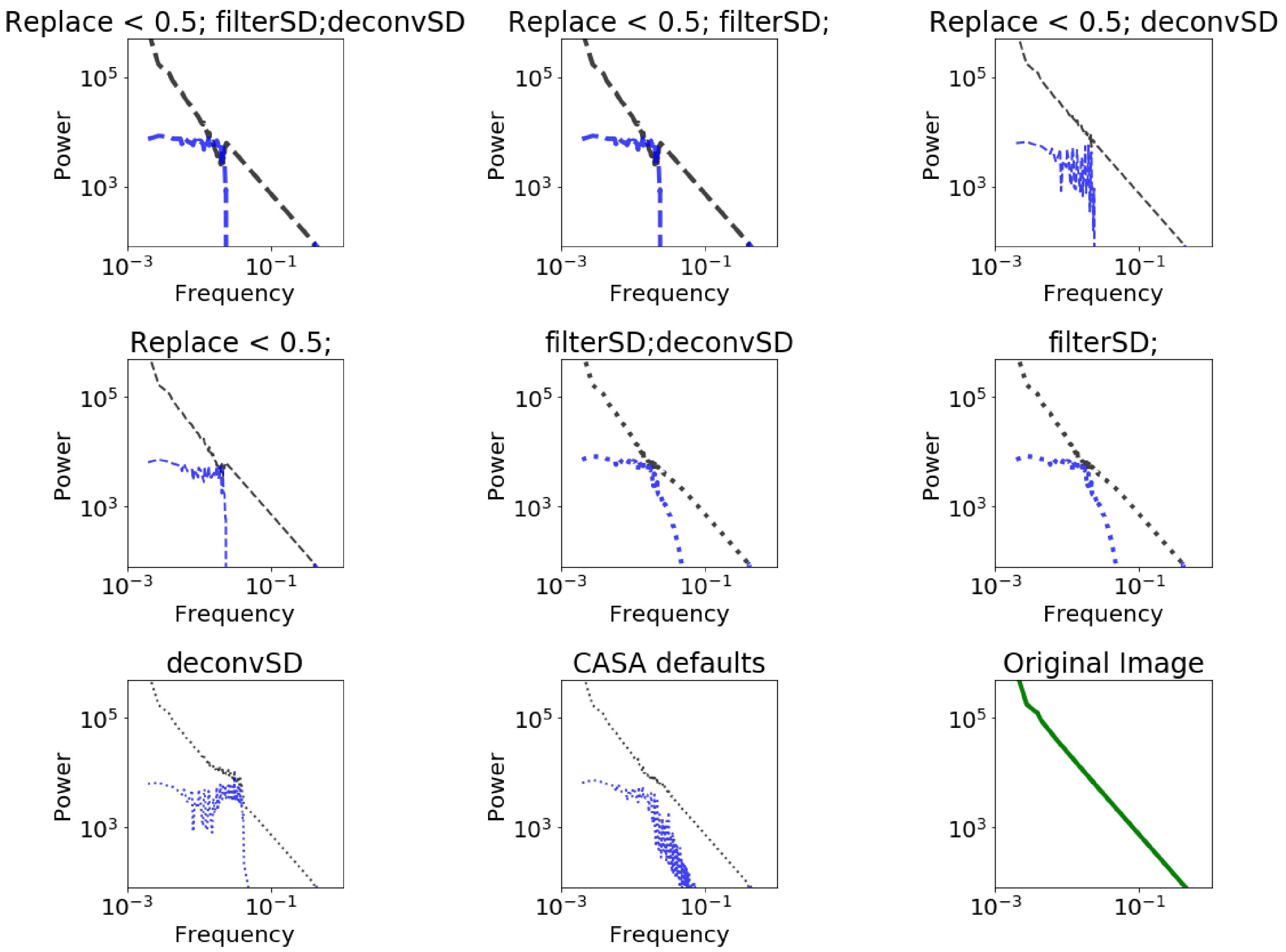


This realistic case is still bad. The "best case", in which we have good UV overlap (e.g., a 24m dish instead of a 12m), actually can get good theoretical recovery


Replace < 0.5; filterSD;deconvSD
Replace < 0.5; filterSD;
Replace < 0.5; deconvSD
Replace < 0.5;
filterSD;deconvSD
filterSD;
deconvSD
CASA defaults
Original Image


Replace < 0.5; filterSD;deconvSD
Replace < 0.5; filterSD;
Replace < 0.5; deconvSD
Replace < 0.5;
filterSD;deconvSD
filterSD;
deconvSD
CASA defaults
Original Image

Residuals are in blue.



CASA's default parameters are fine, but for best performance, the single-dish image should be deconvolved.

The appropriate choice of feather parameters depends (mildly) on the UV overlap:

- If the single dish is substantially bigger than the shortest baseline, CASA's defaults or replacing short-spacing with deconvolved single-dish both work well
- If the single dish is comparable to the shortest baseline, the best results come from deconvolving the single-dish data and weighted-averaging them with the interferometric

These feather experiments represent the absolute best-case scenario. They should be used as references for comparison with any other combination technique.

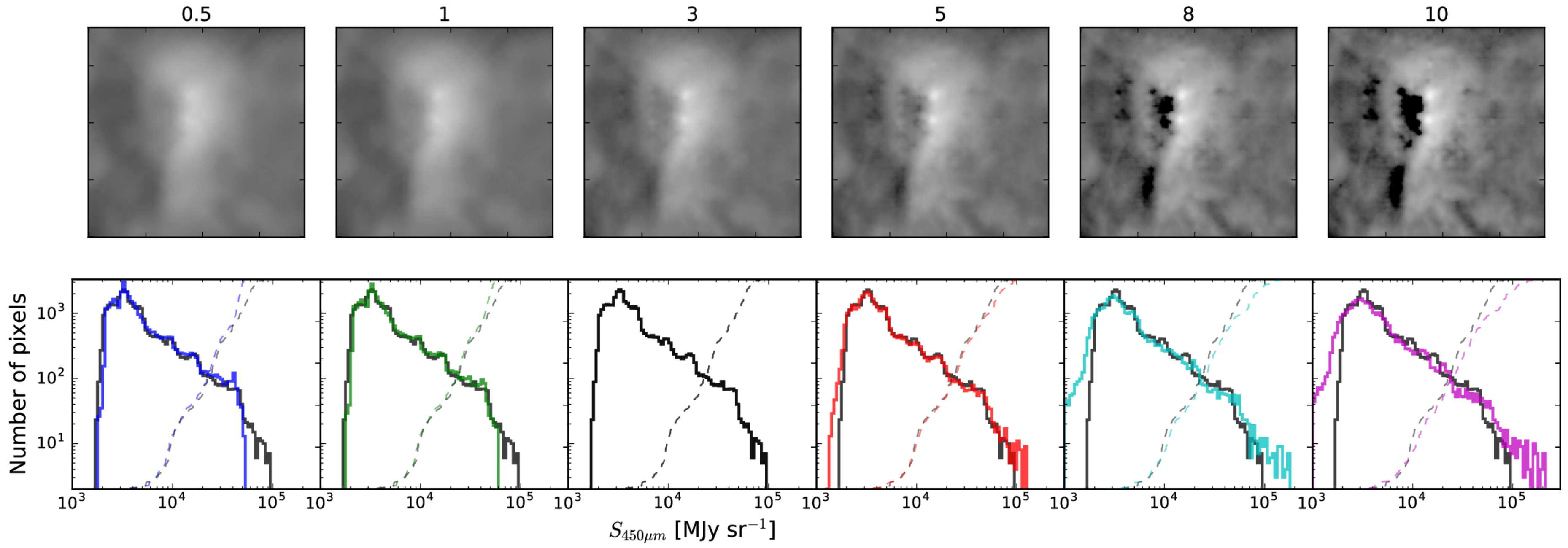
Non-ideal cases

Besides simple UV coverage problems (which are expensive to fix and usually out of our control), there are other issues:

- Relative calibration
- Relative sensitivity

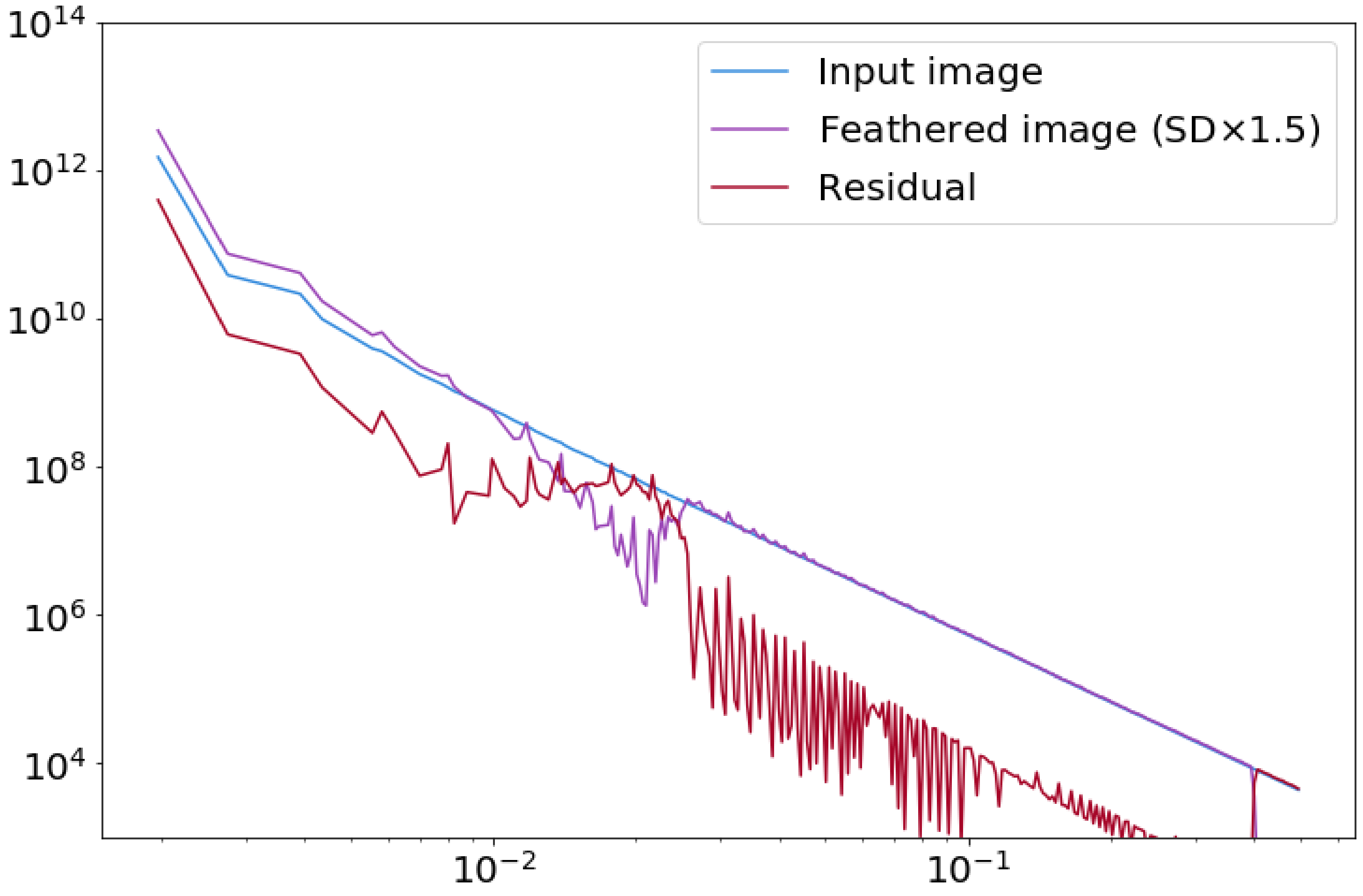
Relative calibration is an issue if the data simply aren't calibrated perfectly (systematic uncertainties are usually 5-15%) or if the single-dish data come from a different frequency range (e.g., using wide-band bolometer data to fill in the continuum zero-spacing).

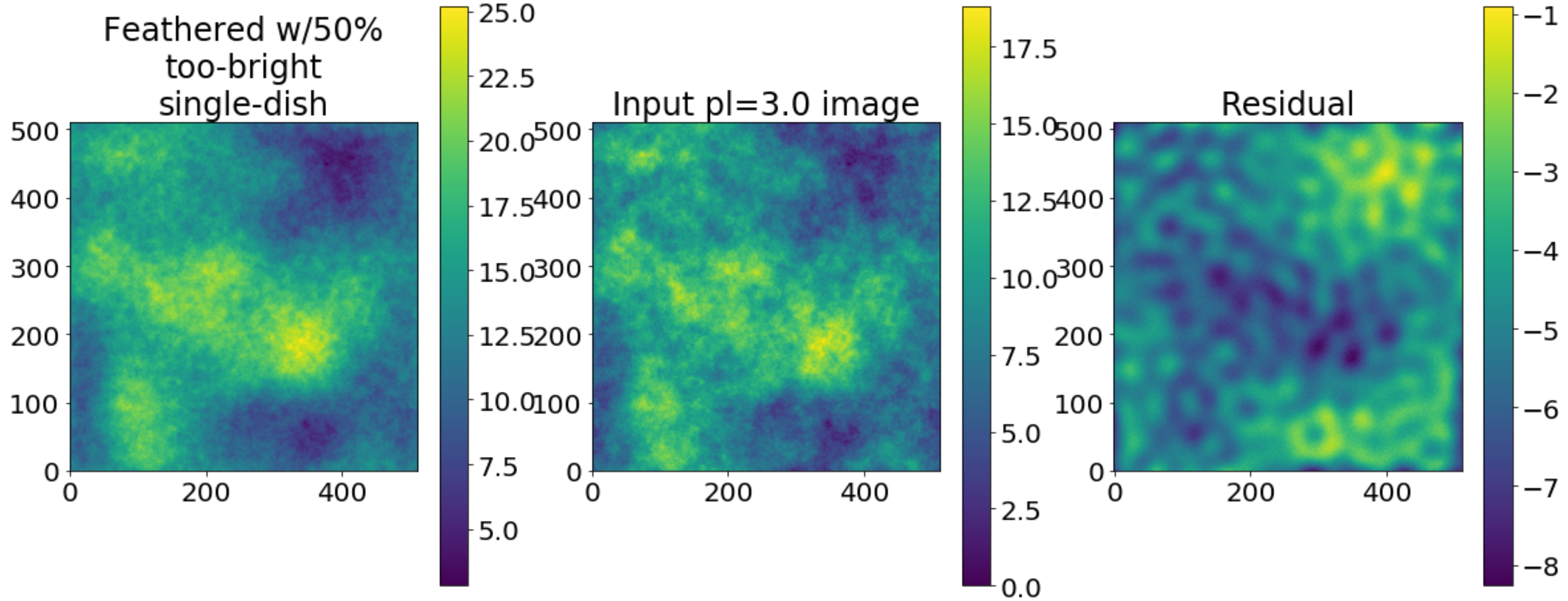
Uncertainties in the "scale factor" and in the beam size of factors of 2-3 are *realistic*: for example, SCUBA on the JCMT at 450 μm has a very uncertain error beam (http://adsabs.harvard.edu/abs/2018ApJ...853..171G (http://adsabs.harvard.edu/abs/2018ApJ...853..171G)).



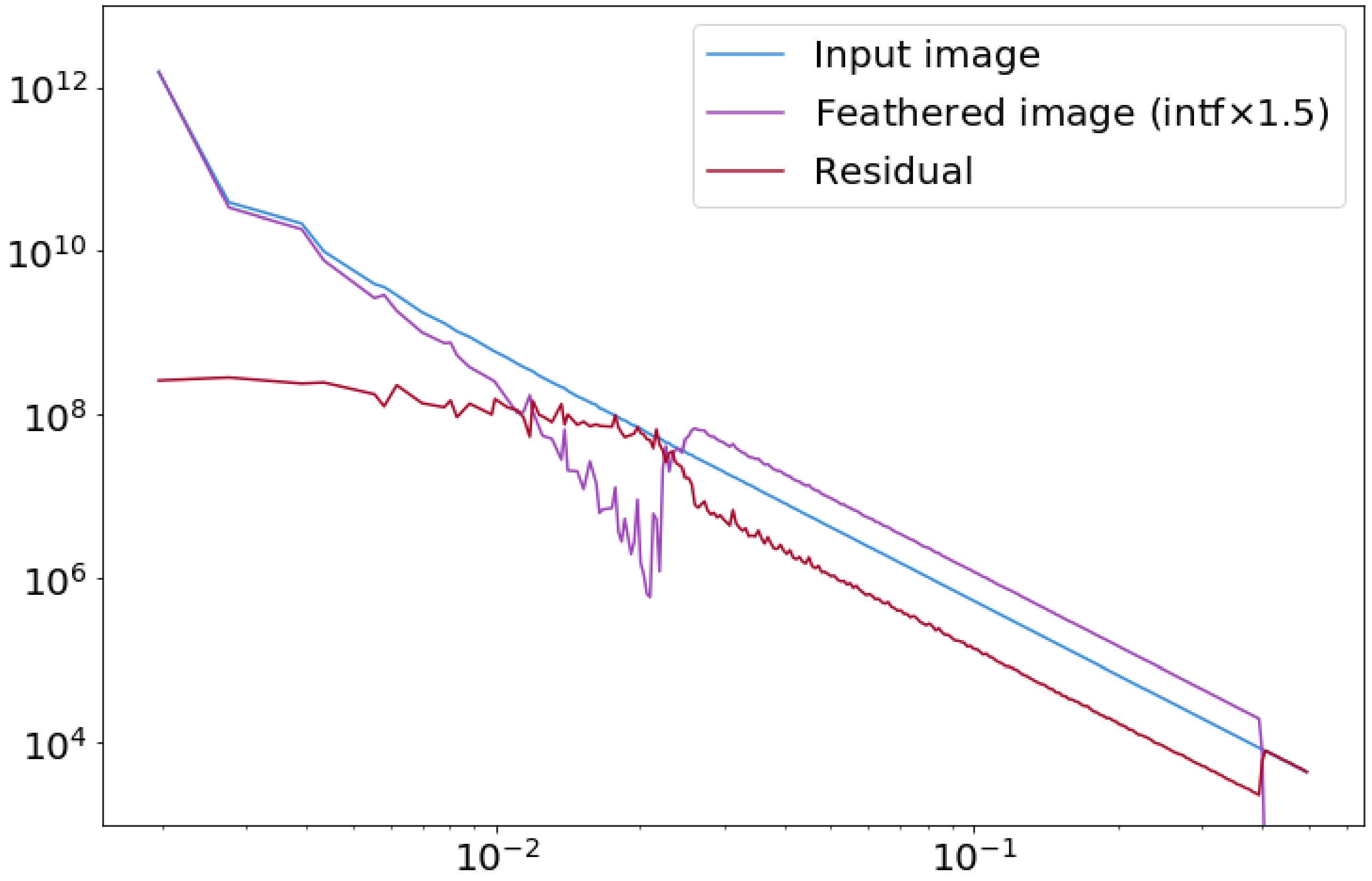
We can simulate calibration error using the `highresscalefactor` and `lowresscalefactor` parameters, which are included to correct for these errors

```
lowresscalefactor = 1.50
highresscalefactor = 1.0
fftsum, combo = fftmerge(kfft/lowresscalefactor, ikfft, im_hi*highresscalefactor,
                         im_low*lowresscalefactor,
                         replace_hires=replace_hires,
                         lowpassfilterSD=lowpassfilterSD,
                         deconvSD=deconvSD,
                        )
```

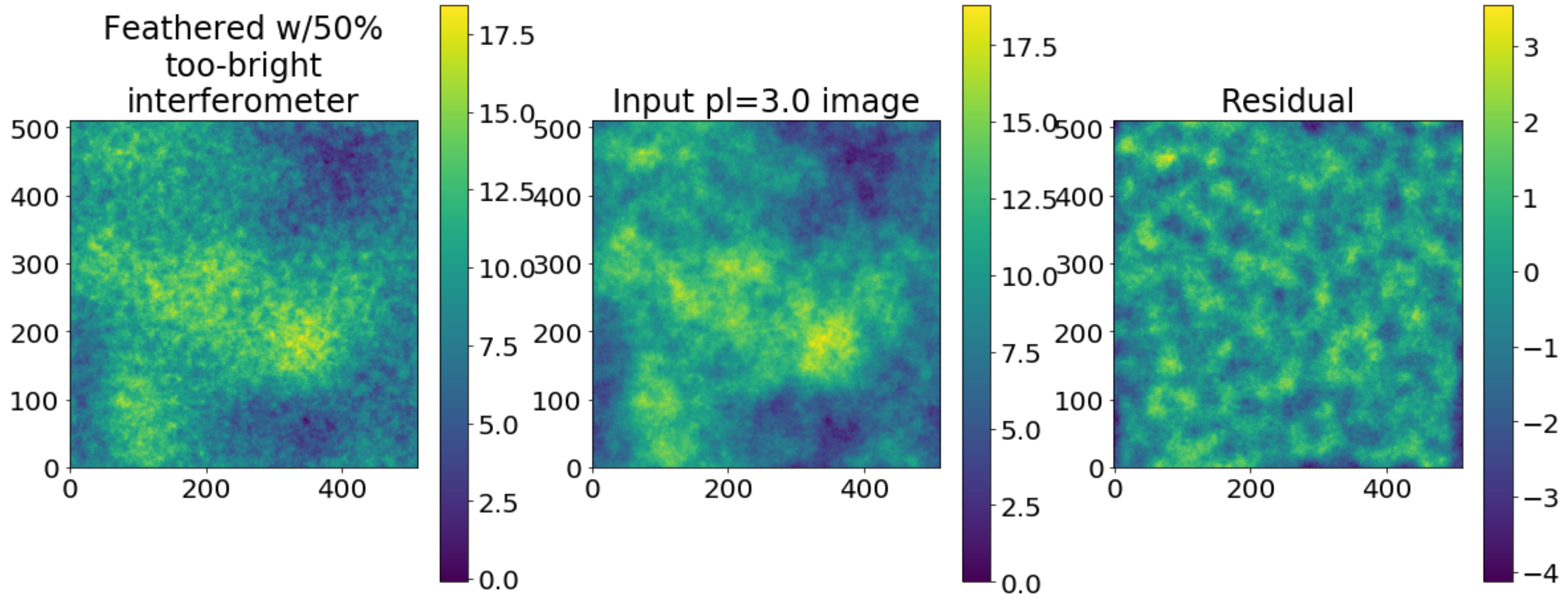

Input image
Feathered image (SD×1.5)
Residual
10^{14}
10^{12}
10^{10}
10^{8}
10^{6}
10^{4}
10^{-2}
10^{-1}


Feathered w/50% too-bright single-dish
Input pl=3.0 image
Residual
0
100
200
300
400
500
0
200
400
25.0
22.5
20.0
17.5
15.0
12.5
10.0
7.5
5.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
−1
−2
−3
−4
−5
−6
−7
−8

```
lowresscalefactor = 1.0
highresscalefactor = 1.50
fftsum, combo = fftmerge(kfft, ikfft, im_hi*highresscalefactor,
                         im_low*lowresscalefactor,
                         replace_hires=replace_hires,
                         lowpassfilterSD=lowpassfilterSD,
                         deconvSD=deconvSD,
                        )
```



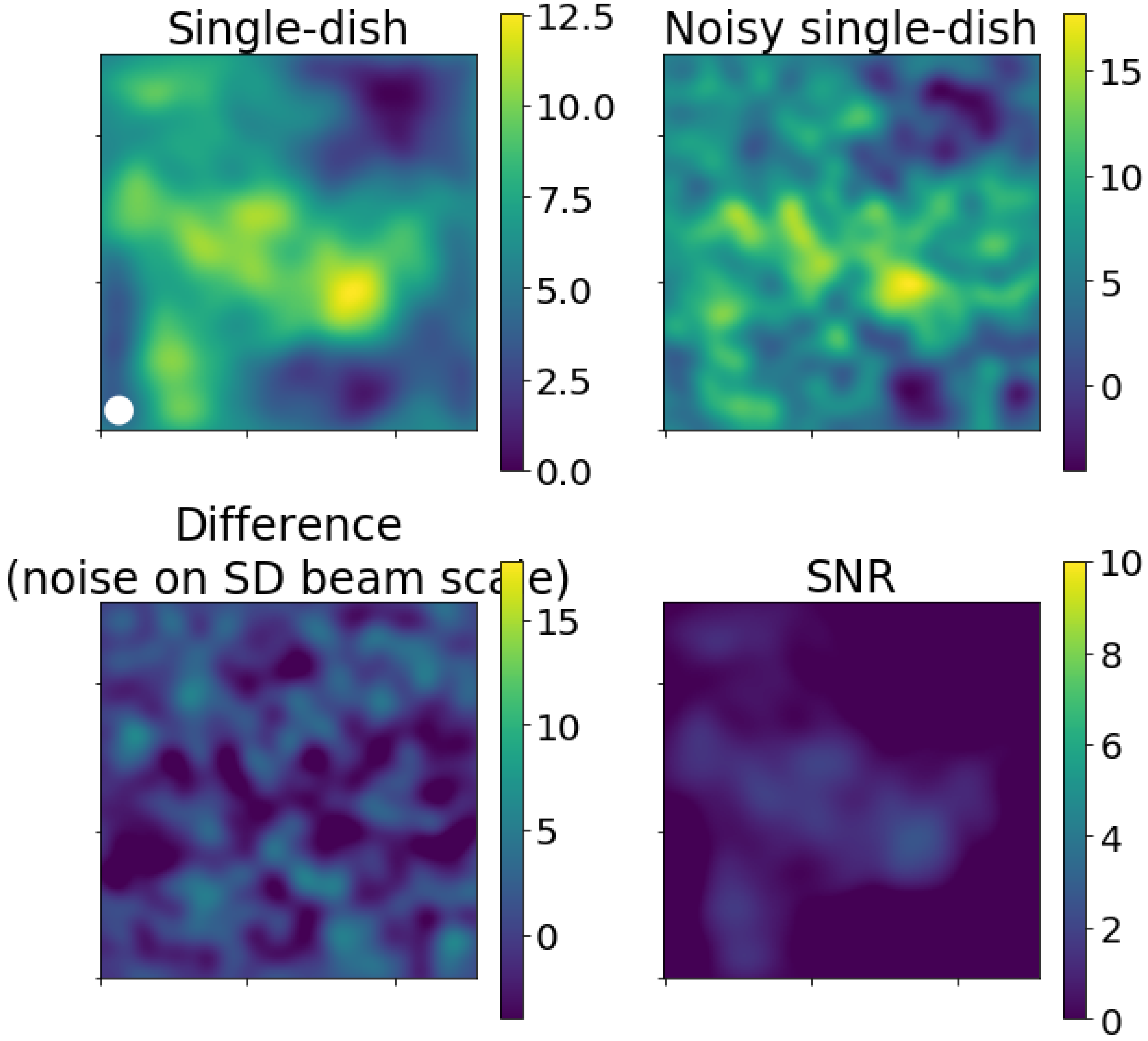
Input image
Feathered image (intf×1.5)
Residual
10^{12}
10^{10}
10^{8}
10^{6}
10^{4}
10^{-2}
10^{-1}

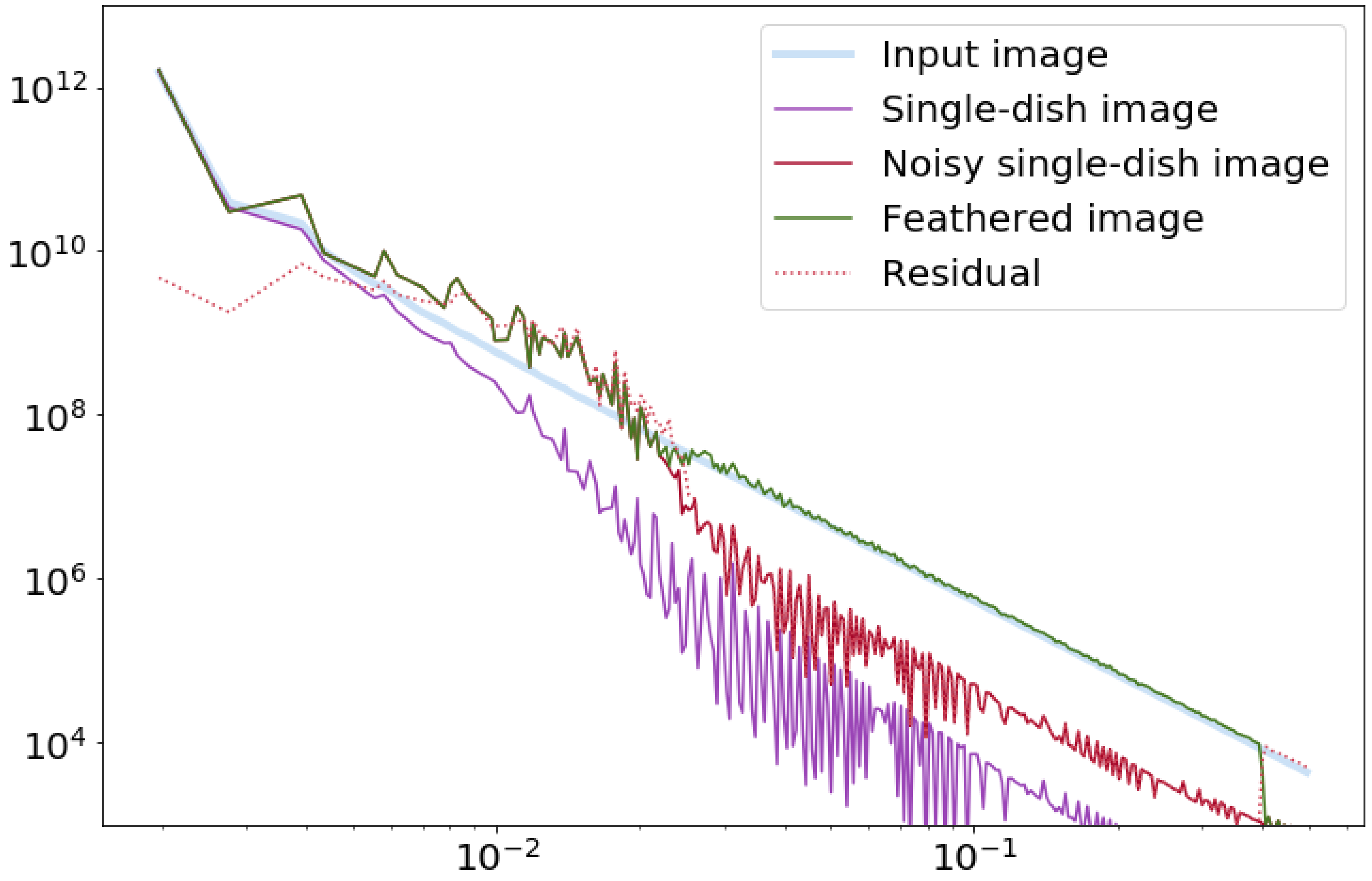

Feathered w/50% too-bright interferometer
Input pl=3.0 image
Residual
0
100
200
300
400
500
0
200
400
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
−4
−3
−2
−1
0
1
2
3

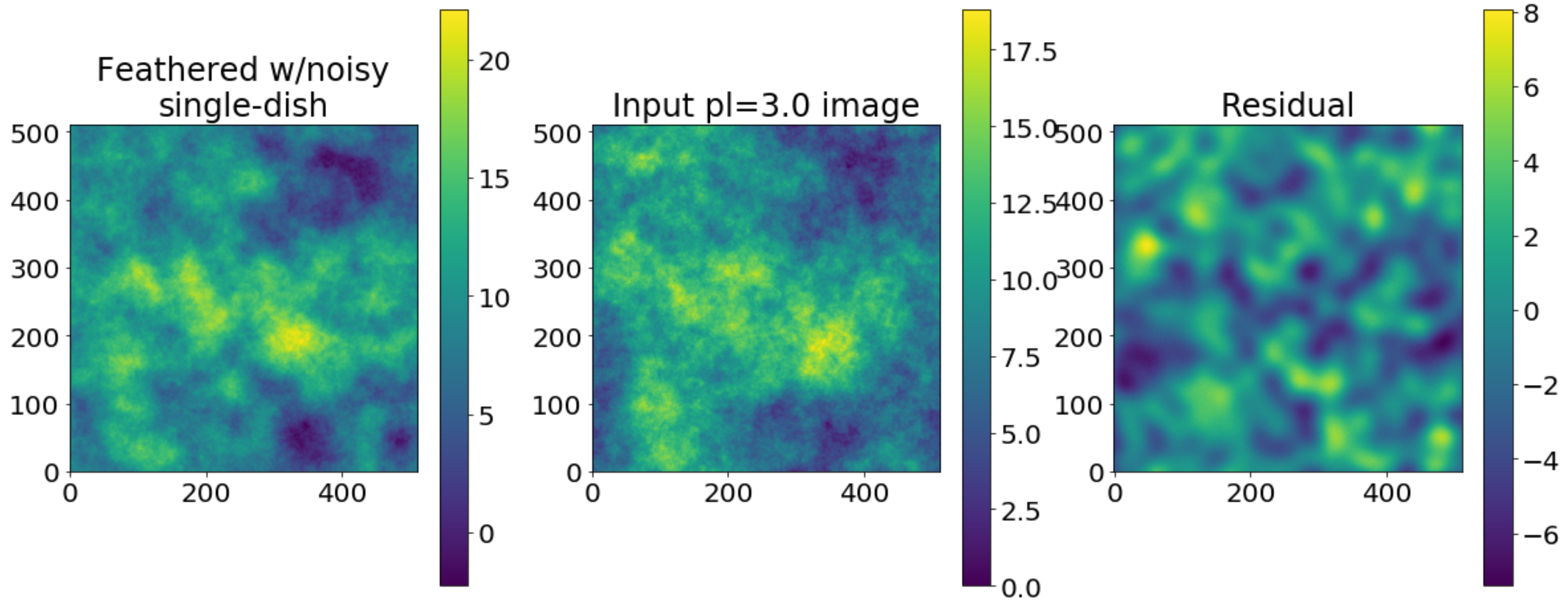
Relative sensitivity

The cases we've treated above assume that the interferometer and single-dish telescope have infinite sensitivity; the "noise" in the image is actually part of the sky. So, what happens if we add observational (as opposed to astrophysical) noise?

```
import radio_beam
import astropy.utils.misc
lowresscalefactor = 1.0
highresscalefactor = 1.0
sd_beam = radio_beam.Beam(lowresfwhm)
sd_beam_volume = (sd_beam.sr / pixscale**2).decompose()
noise_amplitude = 0.005
with astropy.utils.misc.NumpyRNGContext(0):
    noise_sd = convolution.convolve_fft(
        singledish_im.max() * noise_amplitude * np.random.randn(*singledish_im.shape) * sd_b
eam_volume,
        sd_beam.as_kernel(pixscale))
noisy_sd = (singledish_im + noise_sd)
fftsum, combo = fftmerge(kfft, ikfft, im_hi*highresscalefactor,
                         noisy_sd*lowresscalefactor,
                         replace_hires=replace_hires,
                         lowpassfilterSD=lowpassfilterSD,
                         deconvSD=deconvSD,
                        )
```


Single-dish
12.5
10.0
7.5
5.0
2.5
0.0
Noisy single-dish
15
10
5
0
Difference
(noise on SD beam scale)
15
10
5
0
SNR
10
8
6
4
2
0

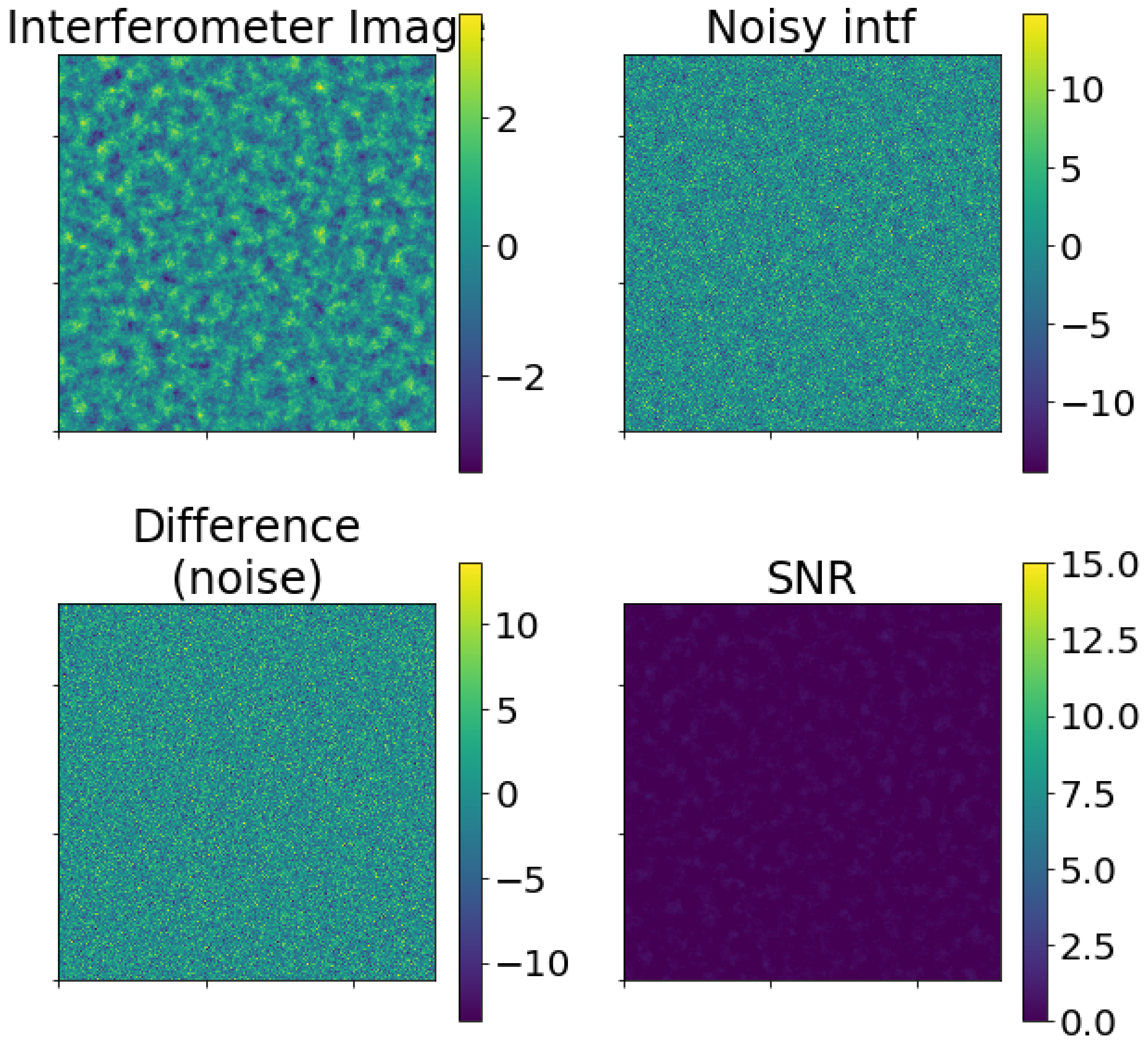

Input image
Single-dish image
Noisy single-dish image
Feathered image
Residual
10^{12}
10^{10}
10^{8}
10^{6}
10^{4}
10^{-2}
10^{-1}

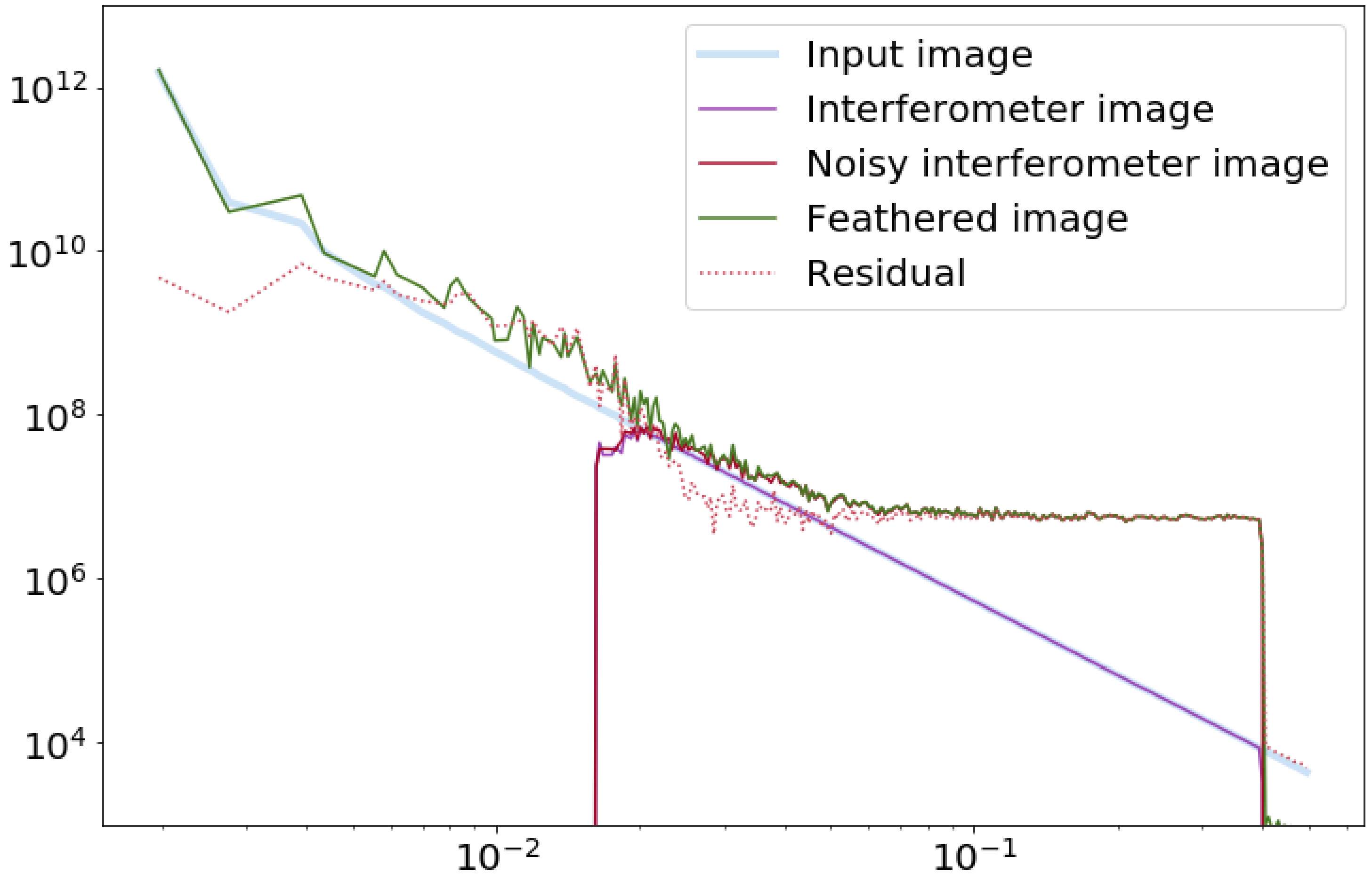

Feathered w/noisy single-dish
Input pl=3.0 image
Residual

Now we'll do the same experiment with noise added to the interferometric image too.

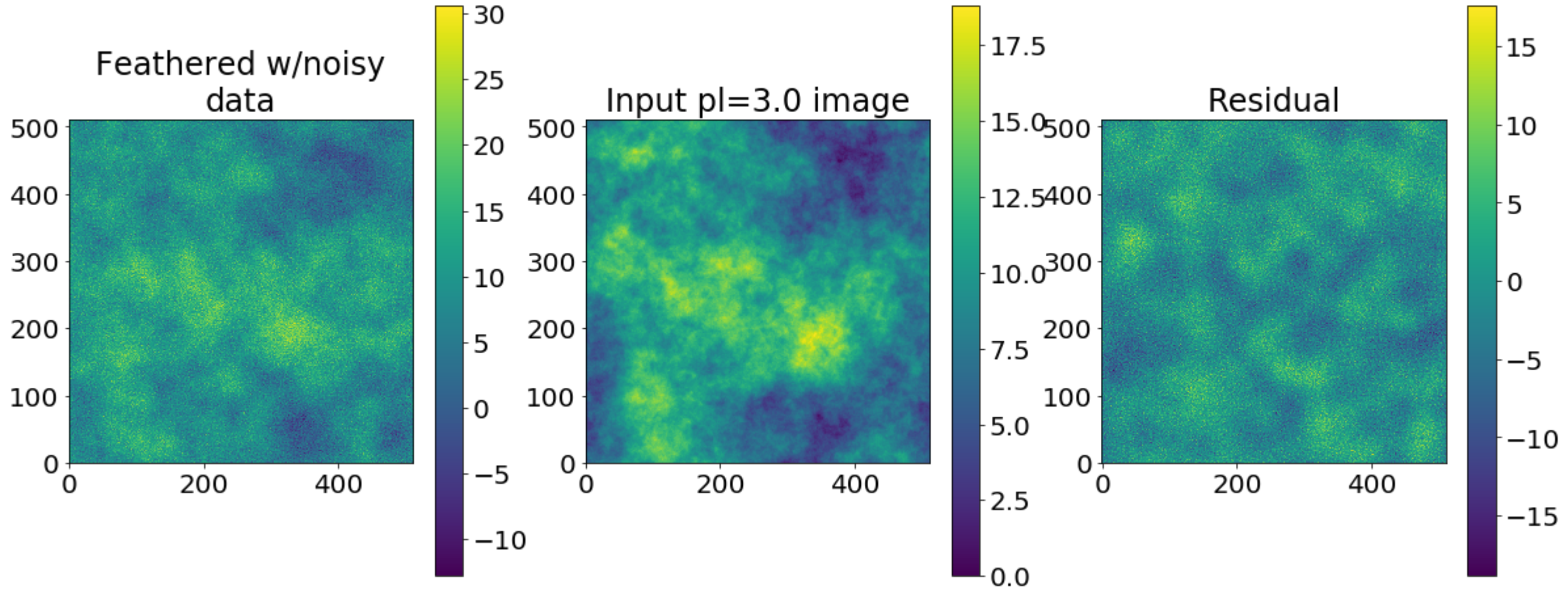
In this case, we're adding Gaussian (white) noise evenly in baseline length, which is perhaps slightly unrealistic.

```
lowresscalefactor = 1.0
highresscalefactor = 1.0
intf_beam = radio_beam.Beam(u.Quantity(smallest_scale, u.arcsec))
intf_beam_volume = (intf_beam.sr / pixscale**2).decompose()
assert intf_beam_volume < 100
noise_amplitude_intf = 0.0005
with astropy.utils.misc.NumpyRNGContext(0):
    noise_intf = np.fft.fftshift(np.fft.fft2(
        im_interferometered.real.max() * noise_amplitude_intf * np.random.randn(*im_interfer
ometered.shape) * intf_beam_volume
        * np.fft.fftshift(ring)).real)
noisy_intf = (im_interferometered.real + noise_intf)
fftsum, combo = fftmerge(kfft, ikfft, noisy_intf*highresscalefactor,
                         noisy_sd*lowresscalefactor,
                         replace_hires=replace_hires,
                         lowpassfilterSD=lowpassfilterSD,
                         deconvSD=deconvSD,
                        )
```


Interferometer Image
2
0
−2
Noisy intf
10
5
0
−5
−10
Difference
(noise)
10
5
0
−5
−10
SNR
15.0
12.5
10.0
7.5
5.0
2.5
0.0

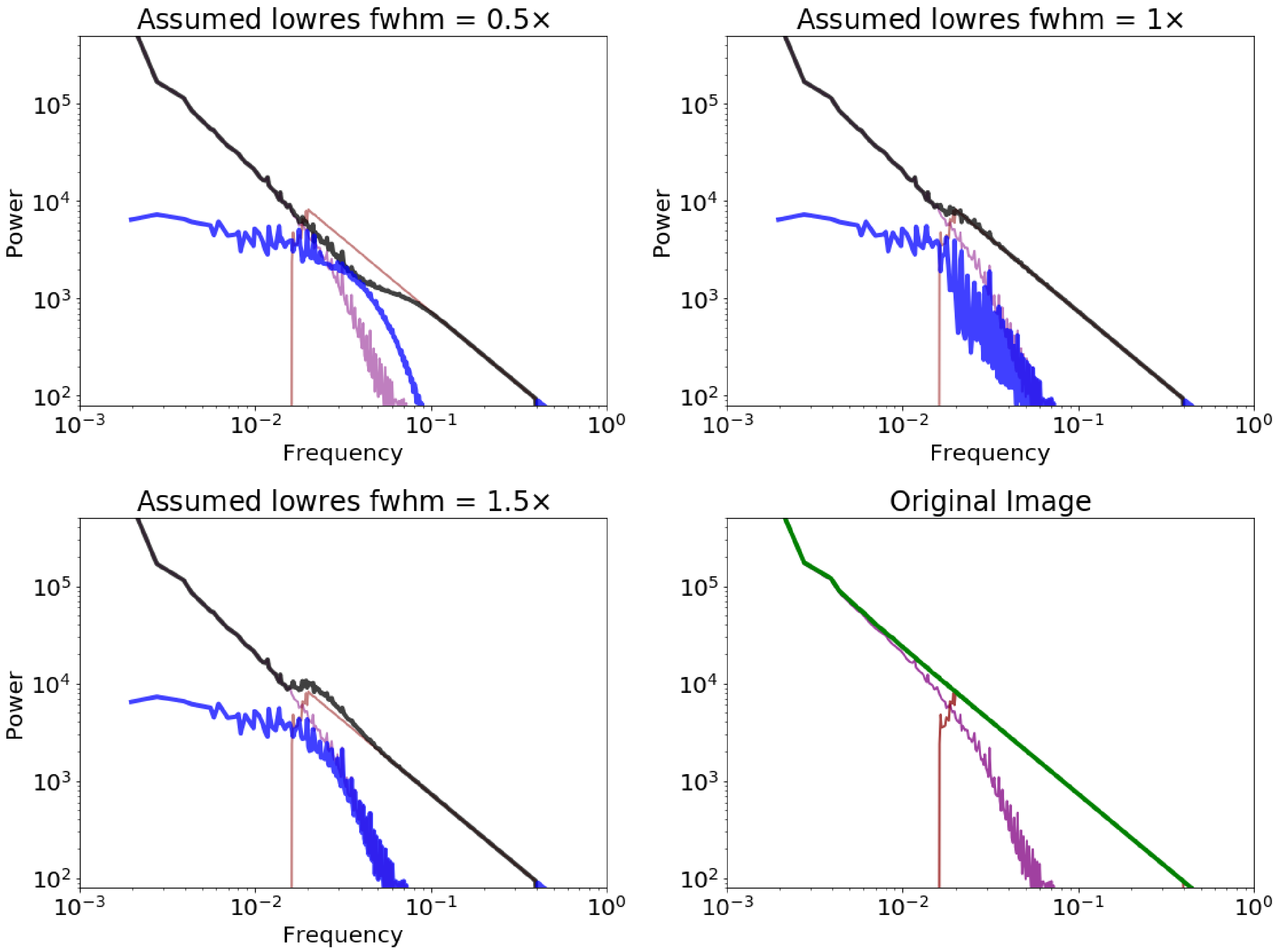


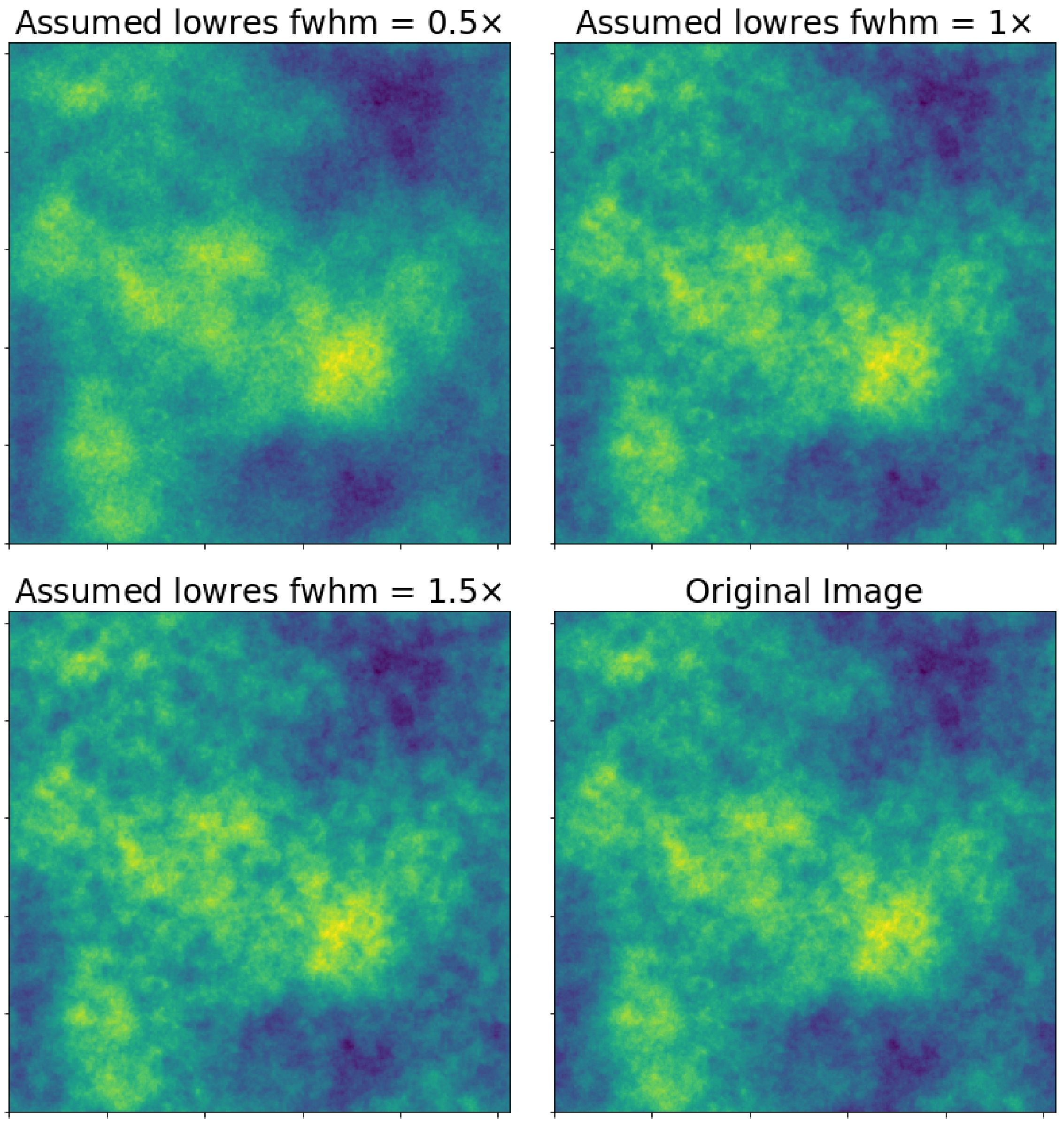
Input image
Interferometer image
Noisy interferometer image
Feathered image
Residual
10^{12}
10^{10}
10^{8}
10^{6}
10^{4}
10^{-2}
10^{-1}

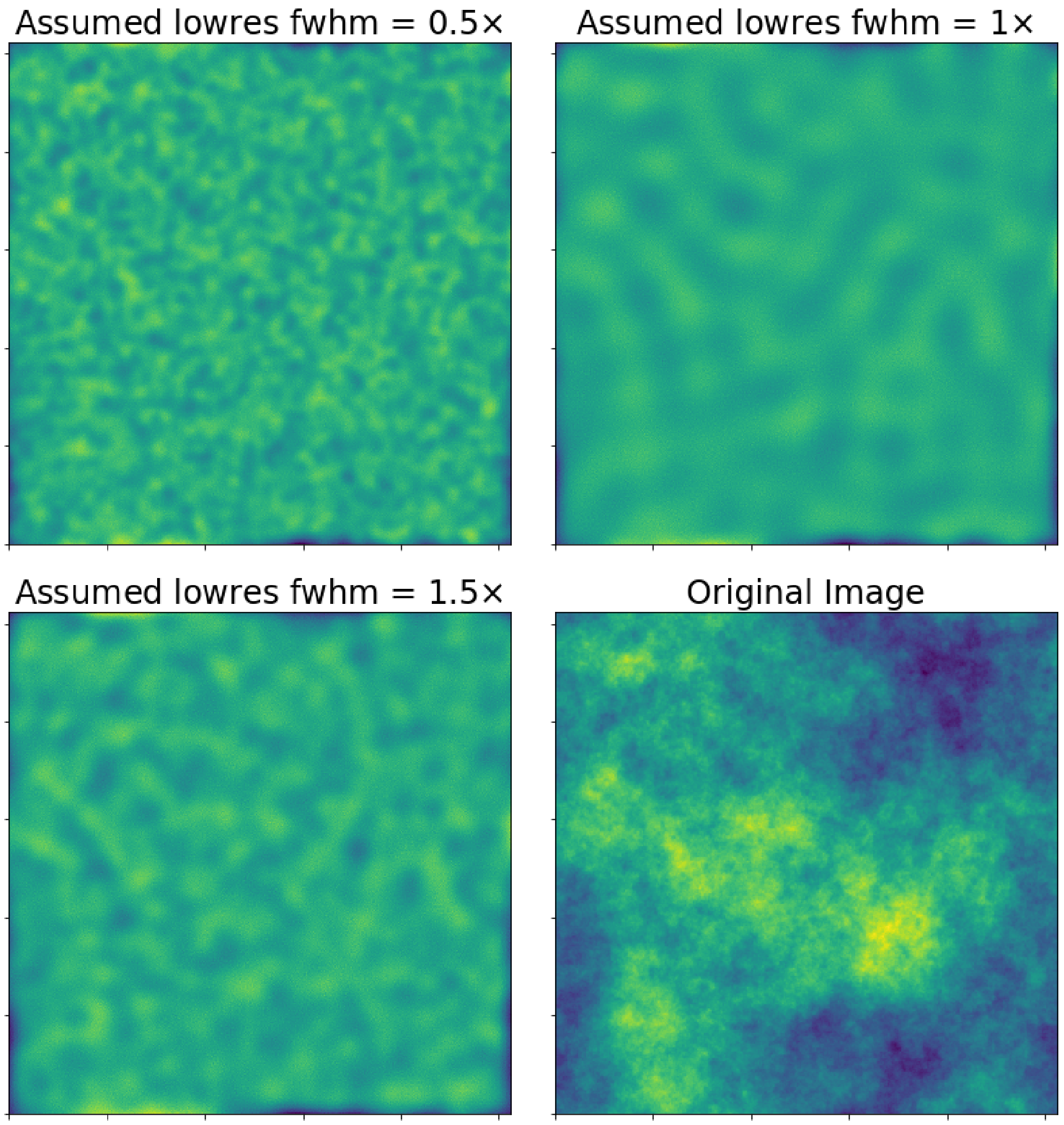

Feathered w/noisy data
Input pl=3.0 image
Residual

Finally, what happens if you assume your single-dish FWHM is smaller or larger than it really is?

In other words, what happens when you over/under weight the single-dish data?


Assumed lowres fwhm = 0.5×
Assumed lowres fwhm = 1×
Assumed lowres fwhm = 1.5×
Original Image
Power
Frequency
10^{-3}
10^{-2}
10^{-1}
10^{0}
10^{2}
10^{3}
10^{4}
10^{5}


Assumed lowres fwhm = 0.5×
Assumed lowres fwhm = 1×
Assumed lowres fwhm = 1.5×
Original Image


Assumed lowres fwhm = 0.5×
Assumed lowres fwhm = 1×
Assumed lowres fwhm = 1.5×
Original Image

Conclusion

Feathering is the most basic form of image combination, but it still has many adjustable parameters.

Feathering should be used as a reference when testing other techniques.

These slides, and others, can be found at https://keflavich.github.io/talks/FeatheringPresentation/ (https://keflavich.github.io/talks/FeatheringPresentation/)