# Skymodel

Alvaro Sanchez-Monge, Adam Ginsburg (U. Florida, NRAO)
Michael Rugel (Max-Planck-Institut für Radioastronomie, Bonn, Germany)
Timothy Braun (University of New Mexico, Albuquerque, NM, USA)
Vivien Chen (National Tsing Hua University, Hsinchu, Taiwan)
September 4, 2019

*This memo was prepared as part of the workshop "Improving Image Fidelity on Astronomical Data: Radio Interferometer and Single-Dish Data Combination," held on 12-16 Aug 2019 at the Lorentz Center in Leiden, The Netherlands.*

## 1 Dataset overview

We used the M100 data set that Peter Teuben downsampled as part of his QAC benchmark suite. We then split out a single channel from each of the 7m and 12m data sets. The image data are produced from a power-law noise spectrum with slope $\alpha = 4$ provided by Jin Koda (Figure 1), but equivalent to that produced by, e.g., `turbustat`'s `make_extended` code[1]. The image is 4096 pixels on a side, and we set a pixel scale of $0.1''$ pix$^{-1}$. The measurement set has a synthesized beam size $\sim 3$ (dependent on the selected weighting).

To populate the visibilities from our image data, we used two methods:

1. Using CASA's `simulate` (abbreviated `sm`) module, we run `sm.setvp()` then `sm.predict()`.
2. We run `tclean` with `startmodel` set to the image. The full parameters can be seen in the associated notebook[2].

We rebinned the sky image to $1024^2$ prior to running these commands. The `sm.predict`ed version does not include appropriate primary-beam tapering of the input image, while the `tclean` approach does. In both cases, we do not fully understand how the images were sampled into visibilities, since the input data highly oversample the beam.

The primary-beam-covered-area slices through the brightest point of emission in our power-law power-spectrum map, which causes severe problems with cleaning. Bright features outside the primary beam cause substantial cleaning artifacts when running tclean.

## 2 Combination Methods

### 2.1 Method 1: Feather and Start Model

We feathered data using both CASA's `feather` task and `uvcombine`'s `feather_simple` task. Both results are identical.

However, we explored a variety of different combinations of feathering with different input start models. The methods are:

1. Jointly deconvolve the 7m and 12m data with no start model, then feather the results togegther

[1] `https://turbustat.readthedocs.io/en/latest/api/turbustat.simulator.make_extended.html`
[2] `https://gist.github.com/keflavich/31a054566d25aa731d16a9908d8327ce`

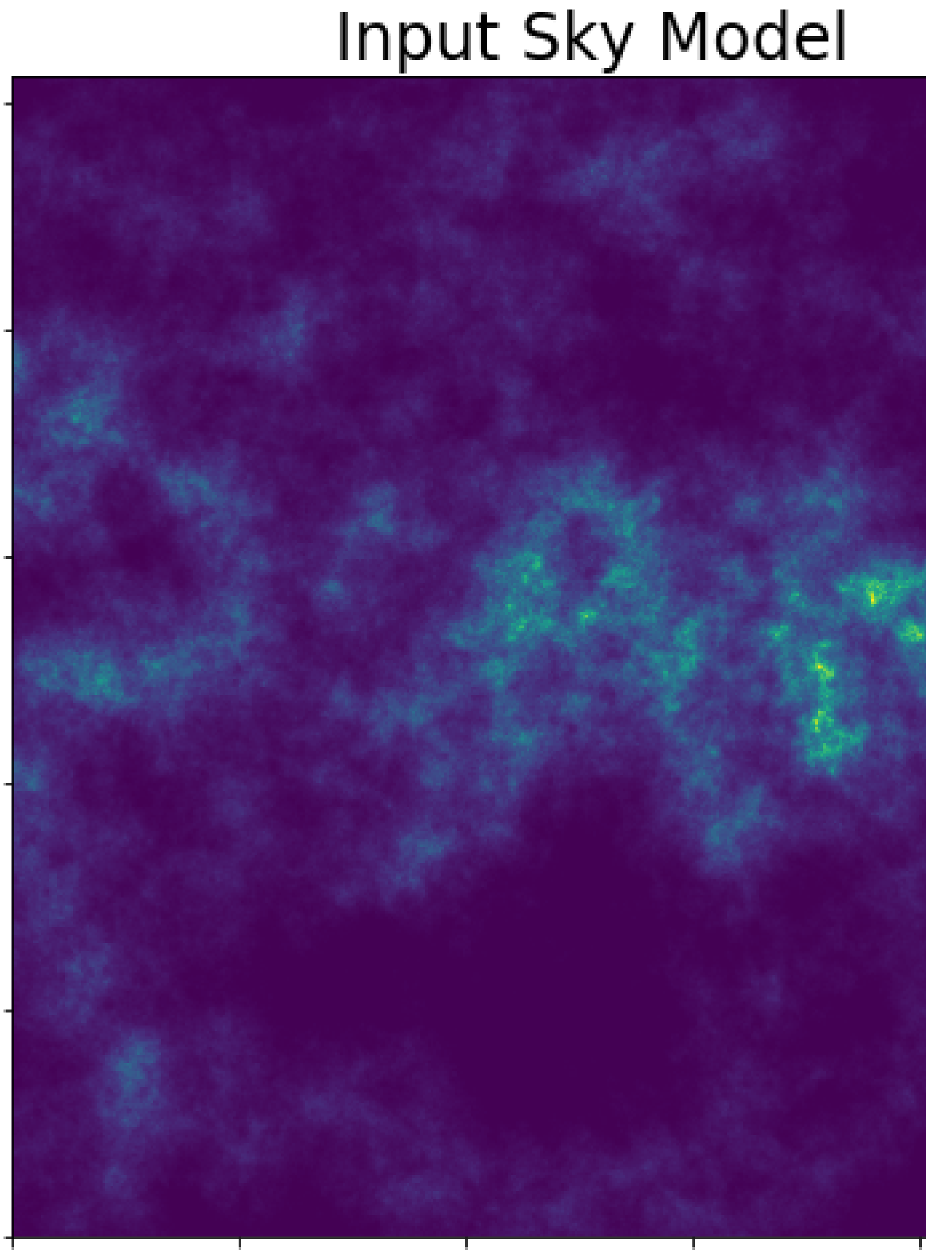

**Figure 1:** The input sky model (rebinned to 1024x1024).

2. Jointly deconvolve the 7m and 12m data with the full-map total power as a start model
3. Jointly deconvolve the 7m and 12m data with the primary-beam-tapered total power as a start model (i.e., the 7m+12m `pb` image is multiplied by the `tp` image prior to using it as a `startmodel`)
4. Deconvolve the 7m data with the total power as a start model, then use the 7m with TP start model as a start model for the 12m clean
5. 'Individual' in the figures below means cleaning the 12m and the 7m arrays independently. Then, featehr 7m and TP together. Finally, feather the 12m image with the 7m+TP.
6. 'Individual startmodel' in the figures below means cleaning the 7m array using TP as model. Then, feather 7m and TP. Next step, clean the 12m array using the TP+7m as model. Finally, feather the 12m image with the 7m+TP.

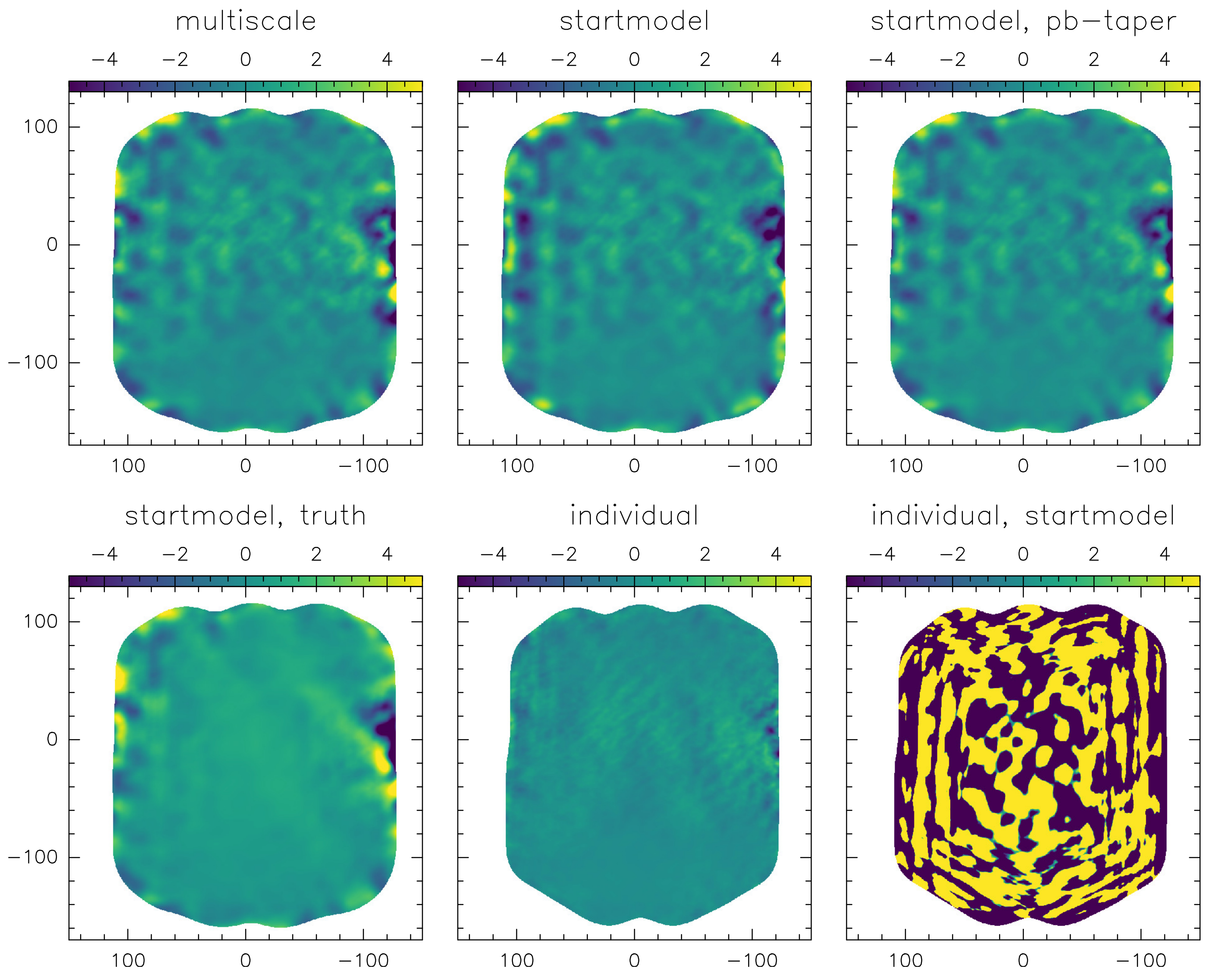

**Figure 2:** Fidelity plot corresponding to 'model − image', for different methods.

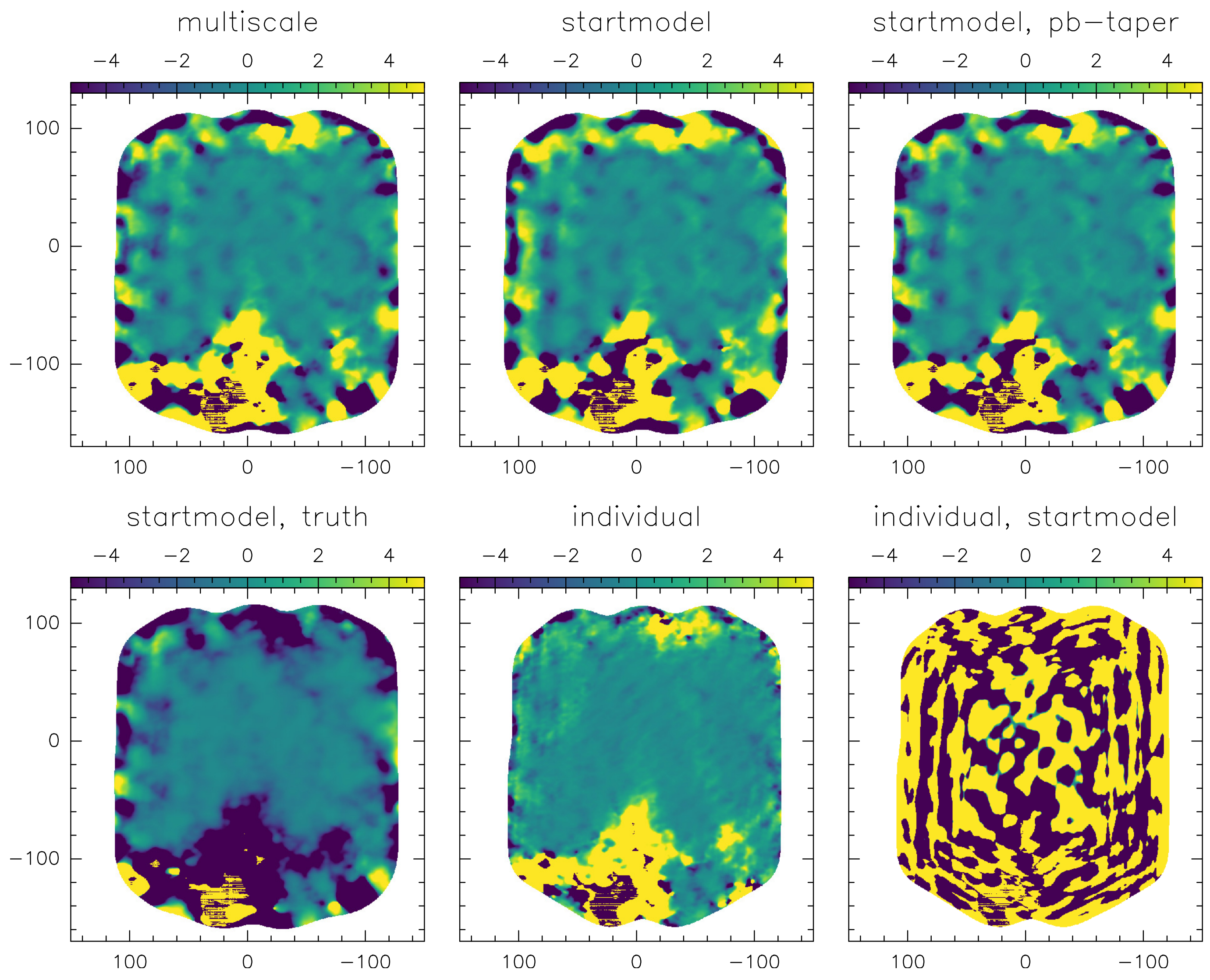

**Figure 3:** Fidelity plot corresponding to '(image − model) / model', for different methods.

## 2.2 Method 2: Joint Deconvolution (tp2vis)

We did not succeed in using the `tp2vis` method on the skymodel dataset since it was noiseless and an In figure ?? we show a few moment maps.

```
tp2vis(TPim,'tp_winpx9.ms','12.ptg',rms=TPrms)
```

## 2.3 Method 3: Joint Deconvolution (SDINT)

We use the `SDINT` package to jointly deconvolve single-dish and interferometry data.

While initially struggling to get `SDINT` running with just a single channel in spectral line mode in tclean (setting `deconvolver='multiscale'`, `specmode='cube'` and `nchan=1` in `runsdint.py`), we managed to run `SDINT` on the simulated data with two methods. First, running `SDINT` in spectral line mode works once we simulated visibilities with three channels and using the same settings as above with `nchan=3`. Second, we succeeded in imaging the single-channel visibilities in continuum multi-

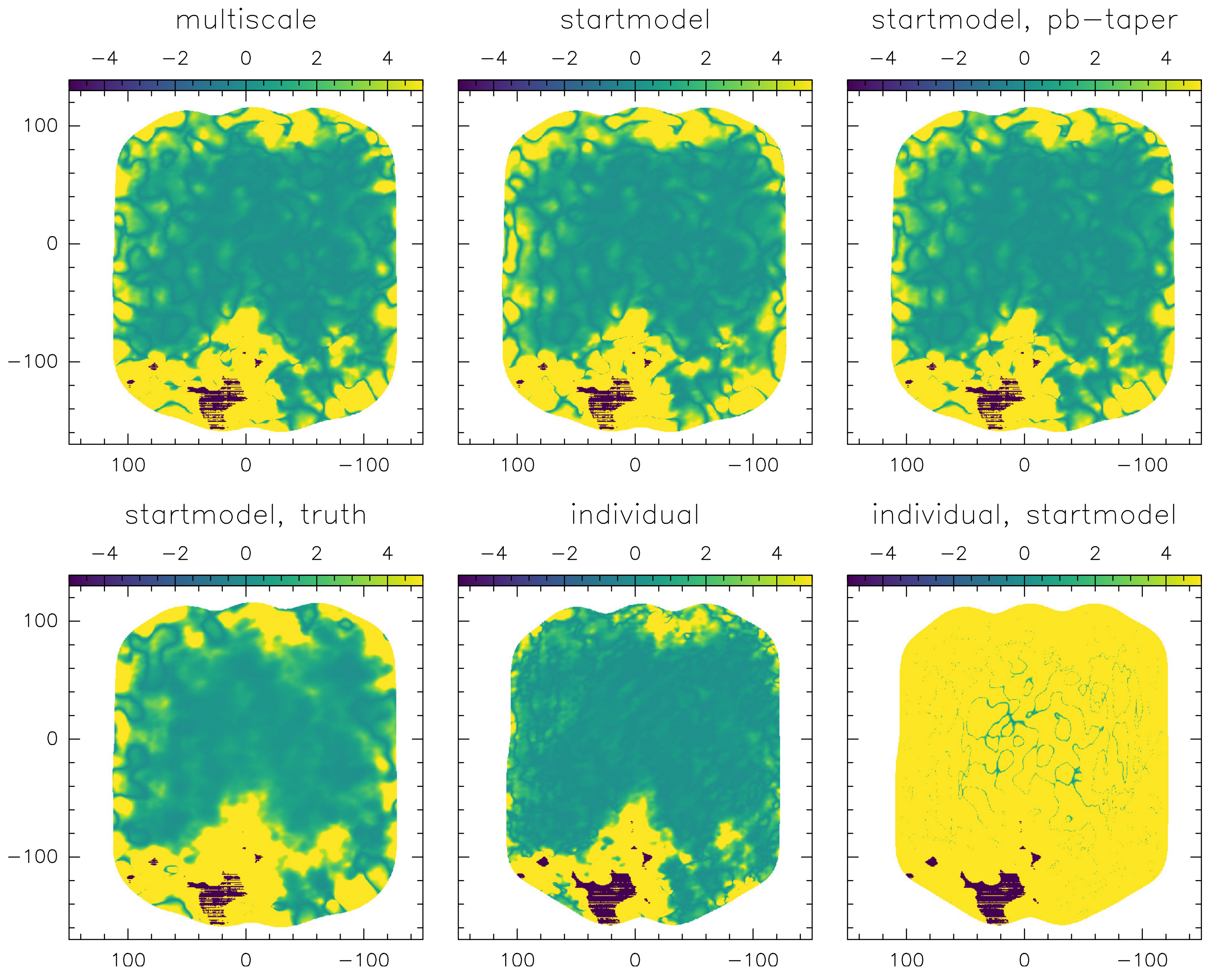

**Figure 4:** Fidelity plot corresponding to 'abs(image − model) / model', for different methods..

term multi-scale mode (setting `deconvolver='mtmfs'`, `specmode='mfs'`, `nchan=1` and `nterms=1` in `runsdint.py`). We will show the results of the latter method.

To simulate the visibilities, we used method (1) in Sect. 1, i.e., using CASA's `simulate`[3]. `SDINT` then needs as input an image of the single-dish point spread function (`SDcube.psf`). We use the script `make_gauss_beam_cube.py` to create a map with a 2D-Gaussian (peaked at 1) at its center with the width of the Single-dish beam. For mosaicked data, the SD PSF must be placed at the phase center of the central pointing (or whatever is used normally when setting the *phasecenter* parameter in `tclean` for your data). The script needed to be slightly modified, as it expects an input model with a beam table, whereas the simulated total power image contained the beam information in the header. This script will also need modifying if a data cube only has a common beam in the header instead of an individual beam per channel.

After these steps, we could successfully ran `runsdint.py`, with the scripts `sdint_helper.py` and

[3]For reference, we also attempted to image visibilities created with `tclean` according to method (2) in Sect. 1 with `SDINT` in spectral line mode. While `SDINT` ran through, the final image did not reproduce the initial model well. Further investigation is needed to explore this method with `SDINT`.

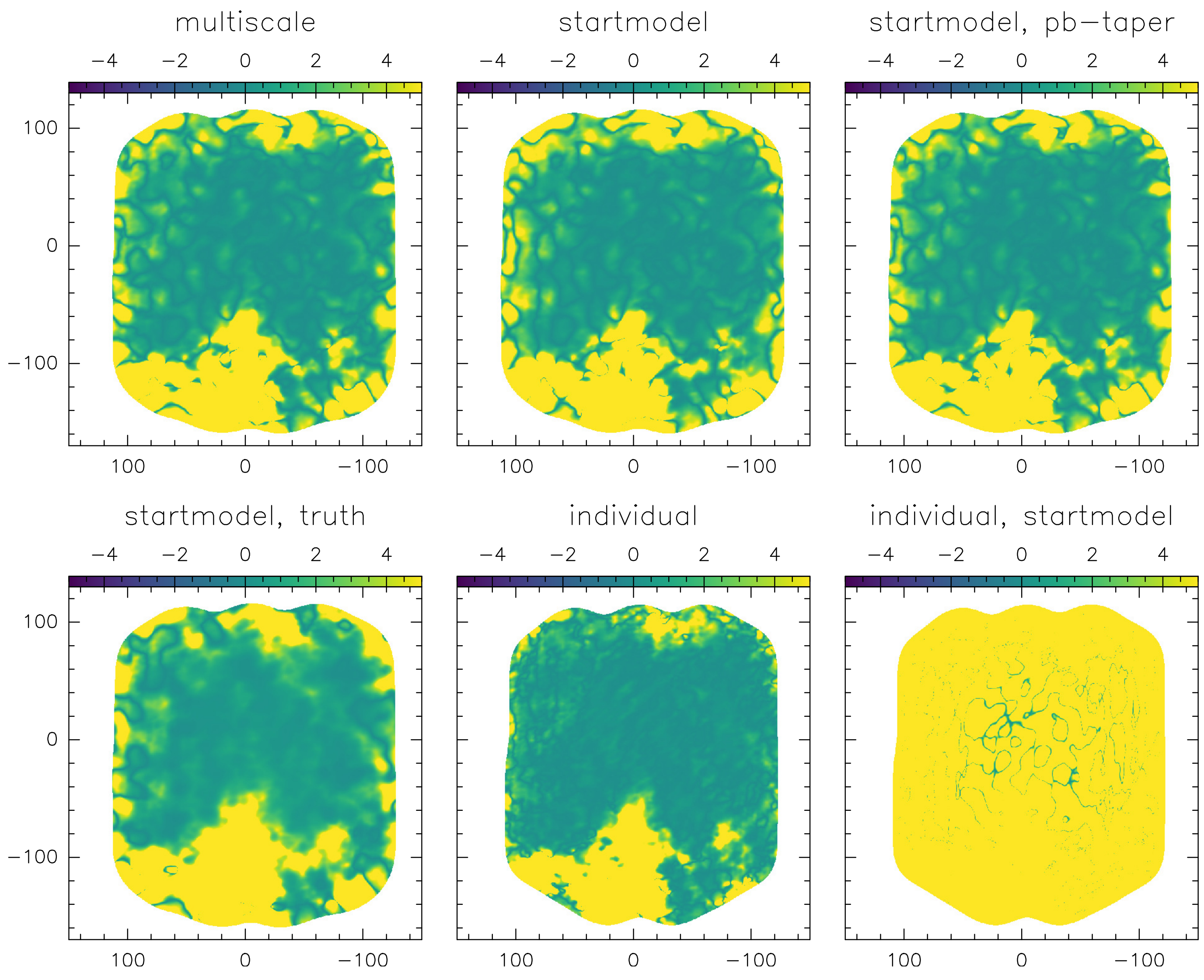

**Figure 5:** Fidelity plot corresponding to 'abs[(image − model) / model]', for different methods..

`sdint_imager.py` placed in the same directory. We choose 100k iterations and a clean threshold of 0.5 Jy. The final results are shown in Fig. 7, including the final image, the model at the same angular resolution, the residual, the difference between image and model, as well as the map of the 'Q'-parameter (fidelity plot).

## 3 Comparison and evaluation

The feather-only results appeared to fill in negative bowls, but didn't recover most of the more compact emission. We saw improvements to the maps when we used a SD starting model in `tclean` and then feathering the 7m+12m image with the TP image. Cleaning with a SD start model should help set the flux density scale for the more compact brighter emission and then feathering with the TP data afterwards should fill in the negative bowls.

Two other important methods involve jointly deconvolving (JD) the TP and interferometric (INT) data at the same time. The two methods we explored in this work are `tp2vis` and `SDINT`. The images/tp2vispl.png

**Figure 6:** Output of tp2vispl for the 12m, 7m, and total power visibilities. Red indicates TP; green 7-m array; and blue 12-m array.

main difference between `tp2vis` and `SDINT` is that `SDINT` combines SD and INT images and PSFs via feathering in the major cycle of tclean prior to any cleaning in the minor cycle. `tp2vis` creates TP/SD psuedo-visibilities and then combines the INT and SD visibilities into one measurement set prior to running tclean.

Currently, it isn't clear to us which JD method better or worse. Our skymodel synthesized data didn't have any noise added to the data, which is an important parameter in `tp2vis` when creating the TP psuedo-visibilities that controls relative weighting between INT and SD data. `SDINT` appeared to work well (reproduces the smoothed model image well) without having to determine noise levels

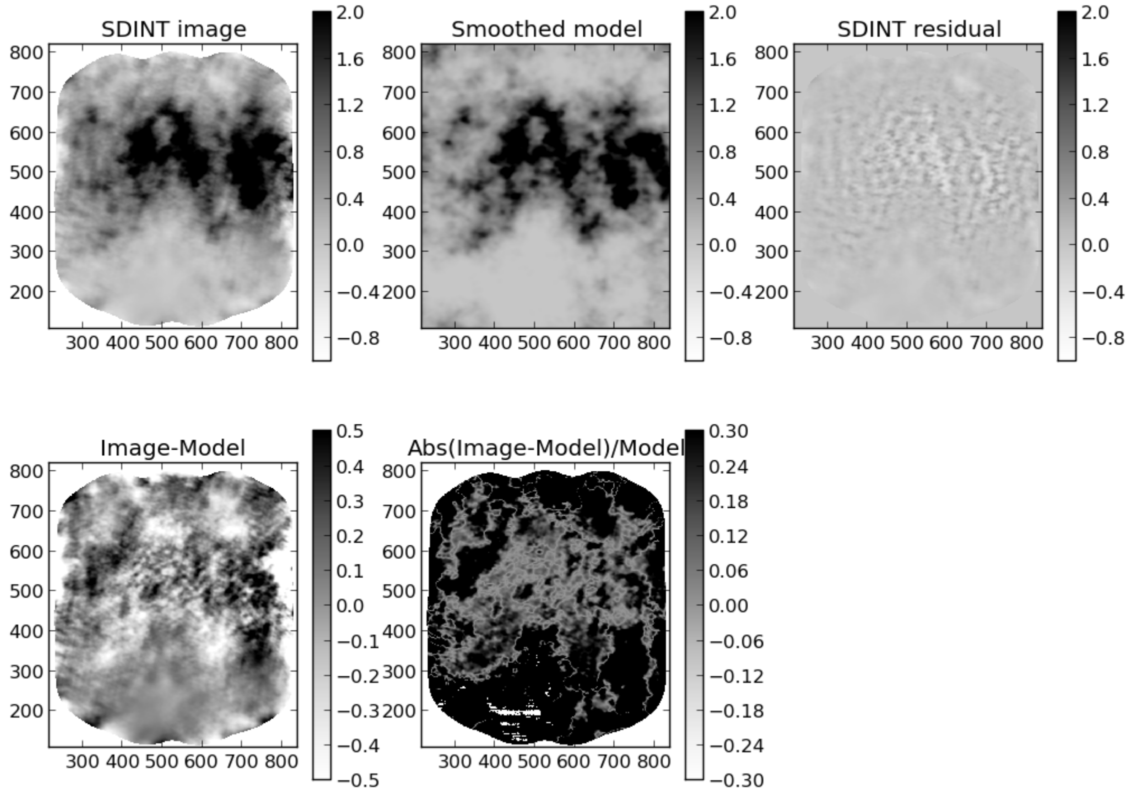

**Figure 7:** Results of the SDINT method (*from left to right, top to bottom)*: Final image, the model at the same angular resolution, the residual, the difference between image and model, as well as the map of the 'Q'-parameter (fidelity plot).

in the SD data. Our simulated data without noise added is not sufficient for a good comparison between SDINT and tp2vis due to the weighting issue described above.

## 4 Additional notes, comments, challenges encountered

We found two potential problems with tp2vis when we used it on our skymodel dataset. The first problem was that after we converted the TP map to psuedo-visibilities and then ran tclean, the total integrated flux was around half the total flux when compared to the original TP map. Other groups may have had this issue as well. The second issue we ran into with tp2vis is with the selection of the rms when creating psuedo-visibilities. Our data was simulated without any noise and when we set rms=0 in tp2vis, the weights of the TP visibilities blew up to infinity. We also used a more sensible value of rms=0.02 that was at the level of the more diffuse emission in our map.However, the 7m+12m+TP cleaned map showed residuals of the 12m pointings (the pointing table used for the TP data). These residuals appeared as columns of diffuse emission with widths similar to the TP PSF size. Moreover, our data is essentially continuum data without noise, so we can't properly determine a line-free channel rms as the tp2vis documentation suggests. We found that the choice of rms in `tp2vis` is extremely important since it controls the relative weights between the 7m, 12m, and TP visibilities.

Organize as you wish. Please, if you discovered an issue during this workshop, document it as an issue in the Github (`https://github.com/teuben/dc2019/issues`). You can briefly describe it here. But, even if you did not find any serious issues, you can comment on challenges, tips, tricks, anything you think would be important for a future user (or you) to recall.

## 5 Appendix: Code

```python
def gauss_beam_abg(amp,parms_abg,x,y):
    a = parms_abg[0]
    b = parms_abg[1]
    g = parms_abg[2]
    r = a*(x**2) + b*x*y + g*y**2
    B = amp*np.exp(-r)
    return B
```