Restoring/Common Beam and Flux Scaling in `tclean`

Jan-Willem Steeb and Urvashi Rau
8 Aug 2019

1 Summary

During the development of CASA 5.6, two issues where found that may leave errors in the flux scaling when using `tclean`. The first issue is that synthesized beams (point spread functions) that deviate from a perfect Gaussian may cause flux deviations from the true flux of sources and residuals when the signal in the data is not completely deconvolved. This "type-1" error is caused by logical algorithmic approximations used for the point-spread function in CASA and other radio astronomy software packages. It is present when using restoring/common beam in `tclean`, or when smoothing the image with `imsmooth`.

The second, more serious, issue involves a bug that was fixed in CASA 5.6, where residuals were scaled wrong when explicitly setting the *restoringbeam* parameter in `tclean` to a value different from the native resolution (including 'common' beam). The magnitude of this "type-2" error in CASA version $\leq$5.5 scaled with the difference in area between the native and specified beam, and did not affect smoothing images with `imsmooth`.

User Recommendation

- Use weighting schemes (via a uv-taper) to implement coarse resolution handling, instead of using restoring beams. This will create beams with more Gaussian shapes.
- When the point spread function (PSF) shape deviates from a Gaussian, only trust the restoration (this includes using a common beam or a restoring beam) on the fully-deconvolved flux.

2 Problem Description

A bug on the flux scaling of the residuals was reported in CASA Jira ticket CAS-12148, stating that when a restoring beam or a common beam is used in `tclean` the residual image is not scaled correctly. The missing scaling factor is $r_\Omega = \Omega_{\text{tar}}/\Omega_{\text{psf}}$, where Ω_{tar} and Ω_{psf} are the beam areas of the target beam (the restoring or common beam) and the Gaussian approximation of the point spread function, respectively. This scaling error becomes significant in an image when the image is not completely deconvolved (that is, source flux is still present in the residual) or there is low signal to noise. The root mean square of the noise will always be affected by the missing scaling factor.

While investigating this bug report, the CASA team also investigated a more subtle effect on the fluxes caused by the standard algorithm in tclean for signal that is not fully deconvolved, especially when the synthesized beam deviated substantially from a Gaussian. Both issues are explained in this CASA memo.

3 Background

3.1 The Elliptical Gaussian

The zero centered two dimensional elliptical Gaussian function or Gaussian beam (see figure 1c) used by CASA is given by

$$f(x,y) = A\exp\left\{-\left(\frac{4\ln(2)}{d_1^2}(\cos(\theta)x+\sin(\theta)y)^2+\frac{4\ln(2)}{d_2^2}(-\sin(\theta)x+\cos(\theta)y)^2\right)\right\}, \tag{1}$$

where A is the amplitude (usually set to unity) and θ is the anti-clockwise angle from the x axis to the line that lies along the greatest width of $f(x,y)$ (the line and the x axis must be coplanar). The factors d_1 and d_2 are respectively the semi-major and semi-minor axis of the ellipse (see figure 1b) which is formed by the cross-section that lies parallel to the x,y plane, at a height so that d_1 is equal to the FWHM (full width at half maximum) distance of the one dimensional Gaussian which lies on the plane formed by the z axis and d_1 (see figure 1c). Note that $d_1 \geqslant d_2 > 0$, since d_1 is the semi-major axis.

By grouping the x and y terms, equation 1 can be written as

$$f(x,y) = A\exp\left[-\left(\alpha x^2+\beta yx+\gamma y^2\right)\right], \tag{2}$$

where

$$\alpha = 4\ln(2)\left[\frac{\cos^2(\theta)}{d_1^2}+\frac{\sin^2(\theta)}{d_2^2}\right], \tag{3}$$

$$\beta = 8\ln(2)\left[\frac{1}{d_1^2}-\frac{1}{d_2^2}\right]\sin(\theta)\cos(\theta), \tag{4}$$

$$\gamma = 4\ln(2)\left[\frac{\sin^2(\theta)}{d_1^2}+\frac{\cos^2(\theta)}{d_2^2}\right]. \tag{5}$$

The form of the two dimensional elliptical Gaussian in equation 2 is useful in calculating its Fourier transform.

Converting from α,β,γ to d_1,d_2,θ can be done using the following set of equations (which are derived using equations 3 to 5):

$$d_1 = \sqrt{\frac{8\ln(2)}{(\alpha+\gamma)-\sqrt{\alpha^2-2\alpha\gamma+\gamma^2+\beta^2}}}, \tag{6}$$

$$d_2 = \sqrt{\frac{8\ln(2)}{(\alpha+\gamma)+\sqrt{\alpha^2-2\alpha\gamma+\gamma^2+\beta^2}}}, \tag{7}$$

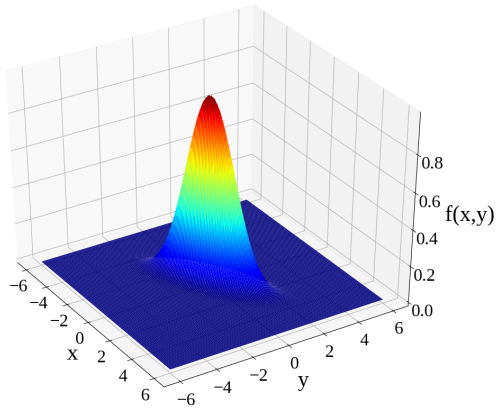
$$\theta = 0.5\ \arctan2(-\beta,\gamma-\alpha). \tag{8}$$

3.2 Fourier Transform of an Elliptical Two Dimensional Gaussian

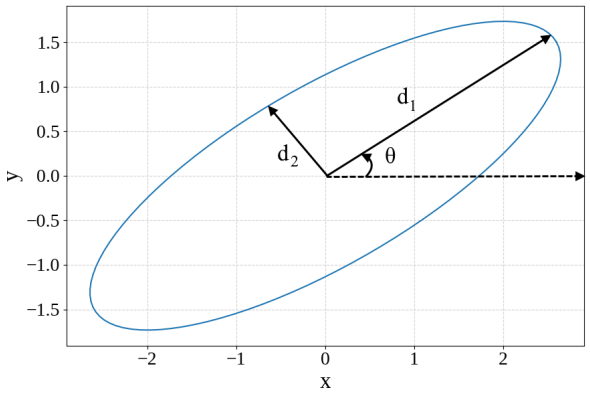
The following form of the Fourier transform and its inverse will be used:

$$F(u,v) = \int_{\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)e^{-i2\pi(ux+vy)}dxdy, \tag{9}$$

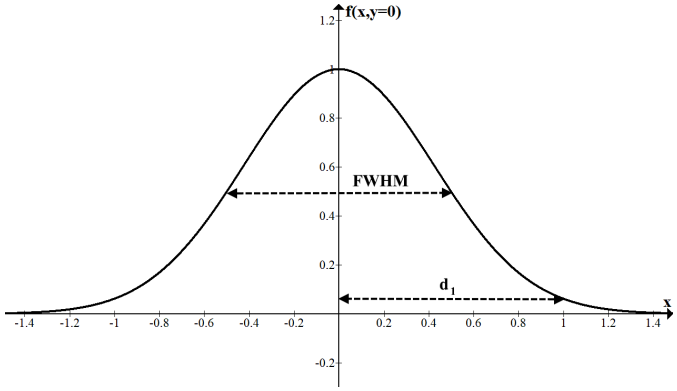
$$f(x,y) = \int_{\infty}^{\infty}\int_{-\infty}^{\infty} F(u,v)e^{i2\pi(ux+vy)}dudv. \tag{10}$$



(a) Surface plot of a two dimensional elliptical Gaussian with $A = 1$, $d_1 = 3$, $d_2 = 1$ and $\theta = 30°$.



(b) Cross-section parallel to the x, y plane of the two dimensional elliptical Gaussian from figure 1a where the resulting ellipse has a semi-major and semi-minor axis equal to d_1 and d_2, respectively.



(c) One dimensional Gaussian plot for $A = 1$, $y = 0$, $\theta = 0$ and $d_1 = 1 = \text{FWHM}$.

Figure 1

The Fourier transform of equation 2 is

$$\begin{aligned} F(u,v) &= \frac{2\pi A}{\sqrt{4\alpha\gamma - \beta^2}} \exp\left\{-\left(\frac{4\alpha\gamma\pi^2u^2 - 4\alpha\beta\pi^2uv + 4\alpha^2\pi^2v^2}{4\alpha^2\gamma - \alpha\beta^2}\right)\right\} \\ &= \frac{2\pi A}{\sqrt{4\alpha\gamma - \beta^2}} \exp\left\{-\left(\alpha^f x^2 + \beta^f xy + \gamma^f y^2\right)\right\}. \end{aligned} \tag{11}$$

Define $\mathbf{c} = [\alpha, \beta, \gamma]^T$ then the coefficients $\alpha^f, \beta^f, \gamma^f$ of the x and y terms in equation 11 can be written as

$$T(\mathbf{c}) = \mathbf{c}^f = \begin{bmatrix} \alpha^f \\ \beta^f \\ \gamma^f \end{bmatrix} = \frac{4\pi^2}{4\alpha\gamma - \beta^2} \begin{bmatrix} \gamma \\ \beta \\ \alpha \end{bmatrix}. \tag{12}$$

The function $T()$ is involutory (a function that is its own inverse), that is,

$$T(T(\mathbf{c})) = \mathbf{c}. \tag{13}$$

Therefore, $T()$ can be used for both the Fourier transform and the inverse Fourier transform. Note that the Fourier transform of a Gaussian yields another Gaussian.

The Fourier and inverse transformed amplitudes are given by

$$A^f = \frac{2\pi A}{\sqrt{4\alpha\gamma - \beta^2}} = \frac{\pi d_1 d_2 A}{4\ln(2)}, \tag{14}$$

$$A = \frac{4\ln(2)}{\pi d_1 d_2} A^f. \tag{15}$$

3.3 Smoothing of an Image to a Specified Resolution

A deconvolved image $\mathbf{I}$ can be smoothed to a target resolution by convolving it with a Gaussian beam $\mathbf{B}_{\rm tar}$. If the image is already convolved with another smaller beam $\mathbf{B}_{\rm cur}$ a correcting beam $\mathbf{B}_{\rm cor}$ can be calculated so that

$$\mathbf{B}_{\rm tar} * \mathbf{I} = \mathbf{B}_{\rm cor} * (\mathbf{B}_{\rm cur} * \mathbf{I}), \tag{16}$$

where $*$ is the convolution operator. The Fourier transform of equation 16 is

$$\mathbf{B}^f_{\rm tar}\mathbf{I}^f = \mathbf{B}^f_{\rm cor}(\mathbf{B}^f_{\rm cur}\mathbf{I}^f), \tag{17}$$

where the superscript f indicates the Fourier transform. The correcting beam can then be obtained by

$$\mathbf{B}_{\rm cor} = \mathcal{F}^{-1}\left(\frac{\mathbf{B}^f_{\rm tar}}{\mathbf{B}^f_{\rm cur}}\right). \tag{18}$$

Using the Fourier relationships derived in section 3.2, an analytical solution can be obtained for $\mathbf{B}_{\rm cor}$. The amplitude and coefficients used for equation 2 to generate $\mathbf{B}_{\rm cor}$ are

$$\mathbf{c}_{\rm cor} = \begin{bmatrix}\alpha_{\rm cor}\\ \beta_{\rm cor}\\ \gamma_{\rm cor}\end{bmatrix} = T(\mathbf{c}^f_{\rm tar} - \mathbf{c}^f_{\rm cur}), \tag{19}$$

$$\begin{aligned} A_{\rm cor} &= \frac{4\ln 2}{\pi d_{1,\rm cor} d_{2,\rm cor}} \frac{d_{1,\rm tar} d_{2,\rm tar}}{d_{1,\rm cur} d_{2,\rm cur}} \\ &= \frac{1}{\Omega_{\rm cor}} \frac{\Omega_{\rm tar}}{\Omega_{\rm cur}}, \end{aligned} \tag{20}$$

where $\mathbf{c}^f_{\rm tar} = T(\mathbf{c}_{\rm tar})$, $\mathbf{c}^f_{\rm cur} = T(\mathbf{c}_{\rm psf})$ and Ω is the beam area. By substituting $\mathbf{c}_{\rm cor}$ into equations 6 to 8 the semi-major axis $d_{1,\rm cor}$, semi-minor axis $d_{2,\rm cor}$ and position angle $\theta_{\rm cor}$ of the correcting beam $\mathbf{B}_{\rm cor}$ can be calculated

$$d_{1,\rm cor} = \sqrt{\frac{8\ln(2)}{(\alpha_{\rm cor} + \gamma_{\rm cor}) - \sqrt{\alpha^2_{\rm cor} - 2\alpha_{\rm cor}\gamma_{\rm cor} + \gamma^2_{\rm cor} + \beta^2_{\rm cor}}}}, \tag{21}$$

$$d_{2,\rm cor} = \sqrt{\frac{8\ln(2)}{(\alpha_{\rm cor} + \gamma_{\rm cor}) + \sqrt{\alpha^2_{\rm cor} - 2\alpha_{\rm cor}\gamma_{\rm cor} + \gamma^2_{\rm cor} + \beta^2_{\rm cor}}}}, \tag{22}$$

$$\theta_{\rm cor} = 0.5\arctan2(-\beta_{\rm cor}, \gamma_{\rm cor} - \alpha_{\rm cor}). \tag{23}$$

To ensure that $\mathbf{B}_{\rm cor}$ is Gaussian the following criteria must be met:

$$\alpha_{\rm cor} > 0, \tag{24}$$

$$\gamma_{\rm cor} > 0, \tag{25}$$

$$\alpha_{\rm cor}^2 - 2\alpha_{\rm cor}\gamma_{\rm cor} + \gamma_{\rm cor}^2 + \beta_{\rm cor}^2 \geq 0. \tag{26}$$

These criteria ensures that the target resolution is larger than the current resolution.

4 Errors in Smoothing an Image (when deconvolver != mtmfs)

For CASA < 5.6 there are two sources of error when smoothing an image. The type 1 error can not be removed (it can be attenuated by tapering the PSF), however the type 2 error has been corrected in CAS-12148.

Type 1 Error: When applying a target beam $\mathbf{B}_{\rm tar}$ (the target beam) in `tclean` the point spread function (PSF) $\mathbf{D}_{\rm psf}$ is approximated by a Gaussian beam $\mathbf{B}_{\rm psf}$ (current resolution) which is fitted to the main lobe of the PSF. This approximation produces an unavoidable error in the smoothed residual that will be added to the smoothed image model. This type of error will be present when using `tclean` (with a restoring beam or with a common beam) and when using `tclean` with `imsmooth`. The error can be represented by the following convolution kernel

$$\mathcal{F}^{-1}\left(\frac{\mathbf{D}_{\rm psf}^f}{\mathbf{B}_{\rm psf}^f}\right). \tag{27}$$

Type 2 Error: Before the fix in CASA 5.6.0 another source of error was the incorrectly scaled correcting beam $\mathbf{B}_{\rm cor}$. The factor

$$r_\Omega = \frac{\Omega_{\rm tar}}{\Omega_{\rm psf}} = \frac{d_{1,\rm tar} d_{2,\rm tar}}{d_{1,\rm psf} d_{2,\rm psf}} \tag{28}$$

was not included (see equation 20). The effect of the incorrect scaling becomes prominent when the signal to noise ratio is low or the image is not completely deconvolved. This error is only present when using `tclean` with a restoring beam or common beam and not when using `tclean` with `imsmooth`.

4.1 Modeling the Errors

The restored image with all errors $\mathbf{I}_{\rm tar,\epsilon}$ generated by CASA (before the fix in branch CAS-12148), when using `tclean` with a restoring or common beam, can be modelled as

$$\begin{aligned}\mathbf{I}_{\rm tar,\epsilon} &= \kappa \mathbf{B}_{\rm tar} * \mathbf{I}_{\rm signal} + \mathbf{B}_{\rm cor} * \mathbf{D}_{\rm psf} * \mathbf{I}_{\rm residual} \frac{1}{r_\Omega} \\ &= \kappa \mathbf{B}_{\rm tar} * \mathbf{I}_{\rm signal} + \mathbf{B}_{\rm cor} * \mathbf{D}_{\rm psf} * ((1-\kappa)\mathbf{I}_{\rm signal} + \mathbf{I}_{\rm noise}) \frac{1}{r_\Omega},\end{aligned} \tag{29}$$

where κ is the fraction of the signal that was deconvolved, $\mathbf{I}_{\rm signal}$ is the deconvolved signal, $\mathbf{I}_{\rm noise}$ is the deconvolved noise. Using the convolution theorem equation 29 can be rewritten as

$$\mathbf{I}_{\rm tar,\epsilon} = \underbrace{\left\{\kappa\boldsymbol{\delta} + \frac{1-\kappa}{r_\Omega} \quad \mathcal{F}^{-1}\left(\frac{\mathbf{D}_{\rm psf}^f}{\mathbf{B}_{\rm psf}^f}\right)\right\}}_{\mathbf{E}_1} * \mathbf{B}_{\rm tar} * \mathbf{I}_{\rm signal} + \underbrace{\left\{\frac{1}{r_\Omega} \quad \mathcal{F}^{-1}\left(\frac{\mathbf{D}_{\rm psf}^f}{\mathbf{B}_{\rm psf}^f}\right)\right\}}_{\mathbf{E}_2} * \mathbf{B}_{\rm tar} * \mathbf{I}_{\rm noise}, \tag{30}$$

where $\boldsymbol{\delta}$ is a Dirac delta image centered at the origin and $\mathbf{E}_1$ and $\mathbf{E}_2$ are the error kernels that are convolved with the smoothed signal image and noise image, respectively. If $\mathcal{F}^{-1}\left(\frac{\mathbf{D}^f_{\rm psf}}{\mathbf{B}^f_{\rm psf}}\right) \approx \boldsymbol{\delta}$ then the error kernels simplify to scaling errors (only the type 2 error remains)

$$e_1 = \kappa + \frac{1-\kappa}{r_\Omega}, \tag{31}$$

$$e_2 = \frac{1}{r_\Omega}. \tag{32}$$

A plot of equation 31 is given in figure 2 as a function of κ and $1/r_\Omega$. The restoring beam has to be larger than the PSF, therefore $1/r_\Omega < 1$. Consequently, both e_1 and e_2 will attenuate the signal.

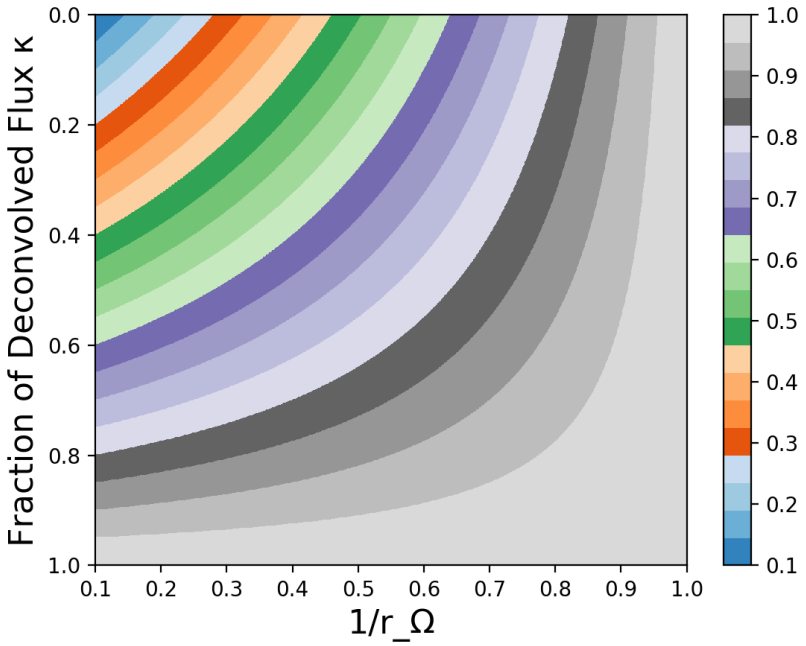


Figure 2: Plot of scaling error e_1 caused by the type 2 error (see equation 31). For example, in the limit of the PSF shape being close to a Gaussian, if both the semi-major axis and semi-minor axis of the restoring beam are twice as long as that of the Gaussian fitted PSF, a factor of four increase in area (i.e. $1/r_\Omega = 0.25$) when combined with niter=0 or κ=0 will cause a 1 $Jy/beam$ source to be attenuated to 0.25 $Jy/beam$.

With the fix in branch CAS-12148 the error kernels become (only the type 1 error remains)

$$\mathbf{E}_1 = \kappa\boldsymbol{\delta} + (1-\kappa)\ \ \mathcal{F}^{-1}\left(\frac{\mathbf{D}^f_{\rm psf}}{\mathbf{B}^f_{\rm psf}}\right), \tag{33}$$

$$\mathbf{E}_2 = \mathcal{F}^{-1}\left(\frac{\mathbf{D}^f_{\rm psf}}{\mathbf{B}^f_{\rm psf}}\right). \tag{34}$$

4.2 Simulation

To verify equation 30 the measurement set `refim_point.ms` was used. The measurement set contains the visibility data for a single point source which has 1.5 $Jy/beam$ flux in the first frequency channel

and no noise. Since there is no noise equation 30 simplifies to

$$\mathbf{I}_{\text{tar},\epsilon} = \underbrace{\left\{ \kappa\boldsymbol{\delta} + \frac{1-\kappa}{r_\Omega} \quad \mathcal{F}^{-1}\left(\frac{\mathbf{D}^f_{\text{psf}}}{\mathbf{B}^f_{\text{psf}}}\right)\right\}}_{\mathbf{E}_1} * \mathbf{B}_{\text{tar}} * \mathbf{I}_{\text{signal}}. \tag{35}$$

Using `tclean` and specifying a restoring beam $\mathbf{B}_{\text{tar}}$ should yield the same result as using `tclean` followed by `imsmooth` (if the restoring beam and the smoothing kernel are the same). This is not the case for CASA < 5.6, since there is the scaling error of $1/r_\Omega$ for `tclean` with a restoring beam. To simulate a type 1 error `tclean` with `imsmooth` is used and to simulate a type 2 error `tclean` with a restoring beam is used.

Type 1 Error: Note that even `tclean` with `imsmooth` does not produce a completely error free result. Using `tclean` with `imsmooth` on a noise free point source yields the following image

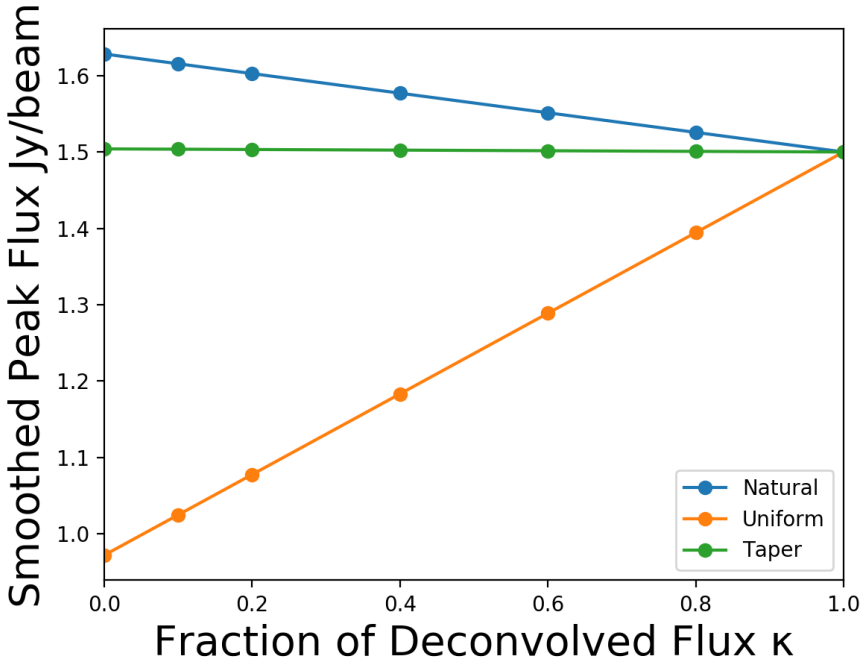
$$\mathbf{I}_{\text{smooth},\epsilon} = \underbrace{\left\{ \kappa\boldsymbol{\delta} + (1-\kappa) \quad \mathcal{F}^{-1}\left(\frac{\mathbf{D}^f_{\text{psf}}}{\mathbf{B}^f_{\text{psf}}}\right)\right\}}_{\mathbf{E}_1} * \mathbf{B}_{\text{tar}} * \mathbf{I}_{\text{signal}}. \tag{36}$$

In figure 3a the peak flux $\mathbf{I}_{\text{smooth},\epsilon}$ is given as a function of the amount of flux that is deconvolved (κ). As κ increases, less of the flux is distorted by $\mathcal{F}^{-1}\left(\mathbf{D}^f_{\text{psf}}/\mathbf{B}^f_{\text{psf}}\right)$. How the visibilities are weighted has a significant effect, because it changes the shape of $\mathbf{D}^f_{\text{psf}}$. When a uv taper is applied to make the $\mathbf{D}^f_{\text{psf}}$ Gaussian the correct peak flux is recovered.

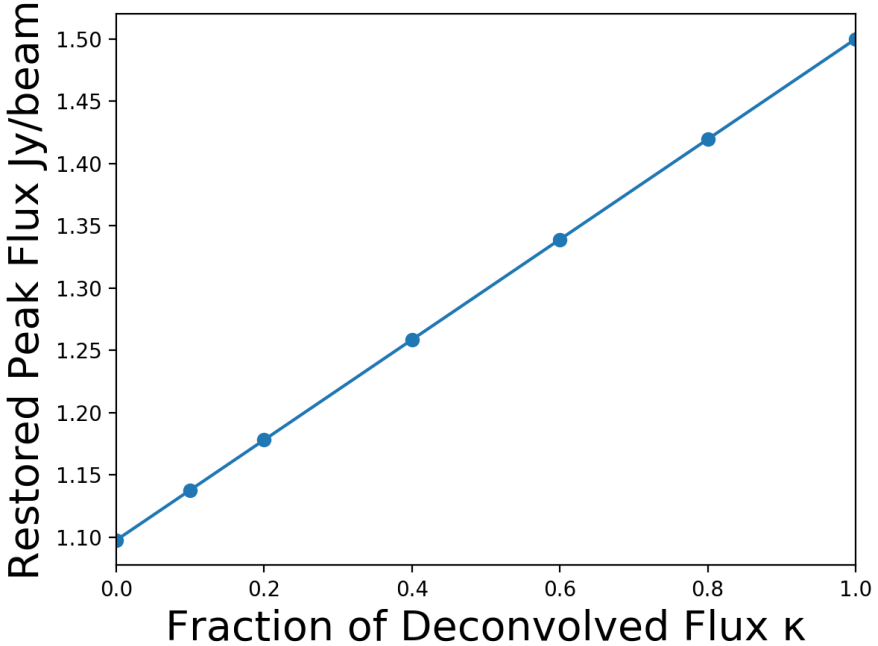
Type 2 Error: When `tclean` with the restoring beam parameter and a uv taper, so that $\mathbf{D}_{\text{psf}}$ is very close to a Gaussian, is applied the following approximation of the image is obtained (for CASA < 5.6)

$$\mathbf{I}_{\text{tar},\epsilon} \approx \underbrace{\left(\kappa + \frac{1-\kappa}{r_\Omega}\right)}_{e_1} \mathbf{B}_{\text{tar}} * \mathbf{I}_{\text{signal}}. \tag{37}$$

The taper used is the same as in figure 3a and attenuates the type 1 error. In figure 3b the peak flux $\mathbf{I}_{\text{tar},\epsilon}$ is given as a function of the amount of flux that is deconvolved (κ). As κ increases less of the flux is distorted by $1/r_\Omega$. Solving for r_Ω in e_1 from the simulated data gave the same result as calculating r_Ω from equation 28.



(a) Peak flux from images generated using `tclean` and `imsmooth` for varying fraction of flux deconvolved for different weightings. The correct flux is $1.5\ Jy/beam$. However, due to the type 1 error the flux for natural and uniform weighting is proportional to the fraction of deconvolved flux (see equation 36). When uv tapering is used to make the PSF more Gaussian, there is no error in the peak flux.



(b) Peak flux from images generated using `tclean` with the restoring beam parameter for varying fraction of flux deconvolved (see equation 37). A uv taper is applied to remove the type 1 error so that only the type 2 error is present due to the incorrect $1/r_\Omega$ scaling. The correct flux is $1.5\ Jy/beam$.

Figure 3

5 Errors in Smoothing an Image (when deconvolver == 'mtmfs')

When using the multi-term multi-frequency algorithm (deconvolver='mtmfs') the same type 1 and type 2 errors appear in the Taylor term images. However, the tt1 (Taylor term 1) to ttN_t (last Taylor term) images have a variant of the type 1 error that will be called the type 1b error. The restored tt0 and tt1 images with all errors generated by CASA (before the fix in branch CAS-12148) are given by

$$\begin{aligned}
\mathbf{I}_{\text{tt0,tar},\epsilon} &= \kappa \mathbf{B}_{\text{tar}} * \mathbf{I}_{\text{tt0,signal}} + \mathbf{B}_{\text{tt0,cor}} * \mathbf{D}_{\text{tt0,psf}} * \mathbf{I}_{\text{tt0,residual}} \frac{1}{r_\Omega} \\
&= \kappa \mathbf{B}_{\text{tar}} * \mathbf{I}_{\text{tt0,signal}} + \mathbf{B}_{\text{tt0,cor}} * \mathbf{D}_{\text{tt0,psf}} * \left((1-\kappa)\mathbf{I}_{\text{tt0,signal}} + \mathbf{I}_{\text{tt0,noise}}\right) \frac{1}{r_\Omega} \\
&= \underbrace{\left\{ \kappa \boldsymbol{\delta} + \frac{(1-\kappa)}{r_\Omega} \mathcal{F}^{-1} \left(\frac{\mathbf{D}^f_{\text{tt0,psf}}}{\mathbf{B}^f_{\text{tt0,psf}}} \right) \right\}}_{\mathbf{E}_{\text{tt0},1}} * \mathbf{B}_{\text{tar}} * \mathbf{I}_{\text{tt0,signal}} + \underbrace{\left\{ \frac{1}{r_\Omega} \mathcal{F}^{-1} \left(\frac{\mathbf{D}^f_{\text{tt0,psf}}}{\mathbf{B}^f_{\text{tt0,psf}}} \right) \right\}}_{\mathbf{E}_{\text{tt0},2}} * \mathbf{B}_{\text{tar}} * \mathbf{I}_{\text{tt0,noise}},
\end{aligned} \tag{38}$$

$$\begin{aligned}
\mathbf{I}_{\text{tt1,tar},\epsilon} &= \kappa \mathbf{B}_{\text{tar}} * \mathbf{I}_{\text{tt1,signal}} + \mathbf{B}_{\text{tt0,cor}} * \mathbf{D}_{\text{tt1,psf}} * \mathbf{I}_{\text{tt1,residual}} \frac{1}{r_\Omega} \\
&= \kappa \mathbf{B}_{\text{tar}} * \mathbf{I}_{\text{tt1,signal}} + \mathbf{B}_{\text{tt0,cor}} * \mathbf{D}_{\text{tt1,psf}} * \left((1-\kappa)\mathbf{I}_{\text{tt1,signal}} + \mathbf{I}_{\text{tt1,noise}}\right) \frac{1}{r_\Omega} \\
&= \underbrace{\left\{ \kappa \boldsymbol{\delta} + \frac{(1-\kappa)}{r_\Omega} \mathcal{F}^{-1} \left(\frac{\mathbf{D}^f_{\text{tt1,psf}}}{\mathbf{B}^f_{\text{tt0,psf}}} \right) \right\}}_{\mathbf{E}_{\text{tt1},1}} * \mathbf{B}_{\text{tar}} * \mathbf{I}_{\text{tt1,signal}} + \underbrace{\left\{ \frac{1}{r_\Omega} \mathcal{F}^{-1} \left(\frac{\mathbf{D}^f_{\text{tt1,psf}}}{\mathbf{B}^f_{\text{tt0,psf}}} \right) \right\}}_{\mathbf{E}_{\text{tt1},2}} * \mathbf{B}_{\text{tar}} * \mathbf{I}_{\text{tt1,noise}},
\end{aligned} \tag{39}$$

where the same notation is used as in equation 30 including the $tt0$ and $tt1$ subscripts. Note that both images are being smoothed to the same resolution $\mathbf{B}_{\text{tar}}$.

The error kernels $\mathbf{E}_{\text{tt0},1}$ and $\mathbf{E}_{\text{tt0},2}$ in $\mathbf{I}_{\text{tt0,tar},\epsilon}$ both contain the type 1 and type 2 errors that were discussed in section 4.

Type 1b Error: The type 1 error is modified in $\mathbf{E}_{\text{tt1},1}$ and $\mathbf{E}_{\text{tt1},2}$ to

$$\mathcal{F}^{-1} \left(\frac{\mathbf{D}^f_{\text{tt1,psf}}}{\mathbf{B}^f_{\text{tt0,psf}}} \right) \neq \boldsymbol{\delta}. \tag{40}$$

This is due to the tt1 image residual $\mathbf{I}_{\text{tt1,residual}}$ being smoothed with the same correcting beam $\mathbf{B}_{\text{tt0,cor}}$ as tt0 (this also applies to all Taylor term images greater than 0). The tt1 PSF $\mathbf{D}_{\text{tt1,psf}}$ is never a Gaussian. Therefore, the type 1b error can not be attenuated by using a uv taper. For the simplest case with only two channels (ν_0, ν_1 and center frequency $\nu_c = (\nu_0 + \nu_1)/2$) and two Taylor terms the tt0 PSF and tt1 PSF are

$$\mathbf{D}_{\text{tt0,psf}} = \frac{\nu_0}{\nu_c} \mathbf{D}_{\text{psf},\nu_0} + \frac{\nu_1}{\nu_c} \mathbf{D}_{\text{psf},\nu_1}, \tag{41}$$

$$\mathbf{D}_{\text{tt1,psf}} = \frac{\nu_0^2 - \nu_0\nu_c}{\nu_c^2} \mathbf{D}_{\text{psf},\nu_0} + \frac{\nu_1^2 - \nu_1\nu_c}{\nu_c^2} \mathbf{D}_{\text{psf},\nu_1}, \tag{42}$$

where $\mathbf{D}_{\text{psf},\nu_0}$ and $\mathbf{D}_{\text{psf},\nu_1}$ are the respective PSFs of the frequency channels ν_0 and ν_1, respectively. If $\mathbf{D}_{\text{psf},\nu_0}$ and $\mathbf{D}_{\text{psf},\nu_1}$ are Gaussian like, then $\mathbf{D}_{\text{tt0,psf}}$ will be Gaussian like. However, $\mathbf{D}_{\text{tt1,psf}}$ will not be Gaussian like because the $\mathbf{D}_{\text{psf},\nu_0}$ is subtracted ($\nu_0^2 < \nu_0\nu_c$).

5.1 Simulation

The measurement set `refim_point.ms` was used which contains the visibility data for a single point source and 20 channels. Only the first and last channel were used. Since there is no noise, equations 38 and 39 simplify to

$$\mathbf{I}_{\mathrm{tt0,tar},\epsilon} = \underbrace{\left\{\kappa\boldsymbol{\delta} + \frac{(1-\kappa)}{r_\Omega}\mathcal{F}^{-1}\left(\frac{\mathbf{D}^f_{\mathrm{tt0,psf}}}{\mathbf{B}^f_{\mathrm{tt0,psf}}}\right)\right\}}_{\mathbf{E}_{\mathrm{tt0},1}} * \mathbf{B}_{\mathrm{tar}} * \mathbf{I}_{\mathrm{tt0,signal}}, \tag{43}$$

$$\mathbf{I}_{\mathrm{tt1,tar},\epsilon} = \underbrace{\left\{\kappa\boldsymbol{\delta} + \frac{(1-\kappa)}{r_\Omega}\mathcal{F}^{-1}\left(\frac{\mathbf{D}^f_{\mathrm{tt1,psf}}}{\mathbf{B}^f_{\mathrm{tt0,psf}}}\right)\right\}}_{\mathbf{E}_{\mathrm{tt1},1}} * \mathbf{B}_{\mathrm{tar}} * \mathbf{I}_{\mathrm{tt1,signal}}. \tag{44}$$

For the simulation the code is executed using CASA before and after the fix in 12148 (that is, the type 2 error is first included and then removed).

Before 12148: If the native resolution is used, the peak flux in tt0 and tt1 is correct. Consequently, the spectral index at the peak flux is also correct (see figures 4a and 4b). Note that the spectral index is not correct off peak if the fraction of deconvolved flux is less than 1.0.

When a restoring beam (target resolution) is used with a uv taper, equations 43 and 37 simplify to

$$\mathbf{I}_{\mathrm{tt0,tar},\epsilon} \approx \underbrace{\left\{\kappa\boldsymbol{\delta} + \frac{(1-\kappa)}{r_\Omega}\right\}}_{e_{\mathrm{tt0},1}} \mathbf{B}_{\mathrm{tar}} * \mathbf{I}_{\mathrm{tt0,signal}}, \tag{45}$$

$$\mathbf{I}_{\mathrm{tt1,tar},\epsilon} = \underbrace{\left\{\kappa\boldsymbol{\delta} + \frac{(1-\kappa)}{r_\Omega}\mathcal{F}^{-1}\left(\frac{\mathbf{D}^f_{\mathrm{tt1,psf}}}{\mathbf{B}^f_{\mathrm{tt0,psf}}}\right)\right\}}_{\mathbf{E}_{\mathrm{tt1},1}} * \mathbf{B}_{\mathrm{tar}} * \mathbf{I}_{\mathrm{tt1,signal}}. \tag{46}$$

The error in the peak flux of the target resolution tt0 image is due to the type 2 error in $e_{\mathrm{tt0},1}$ and the error in the peak flux of the target resolution tt1 image is due to the type 1b and type 2 error in $\mathbf{E}_{\mathrm{tt1},1}$ (see figure 4a). Consequently, this creates an error in the spectral index of the image peak (see figure 4b). The errors are inversely proportional to the fraction of deconvolved flux.

After 12148: The peak values are still correct when the native resolution is used (see figures 4c and 4d).

When a restoring beam (target resolution) is used with a uv taper, equations 43 and 37 simplify to

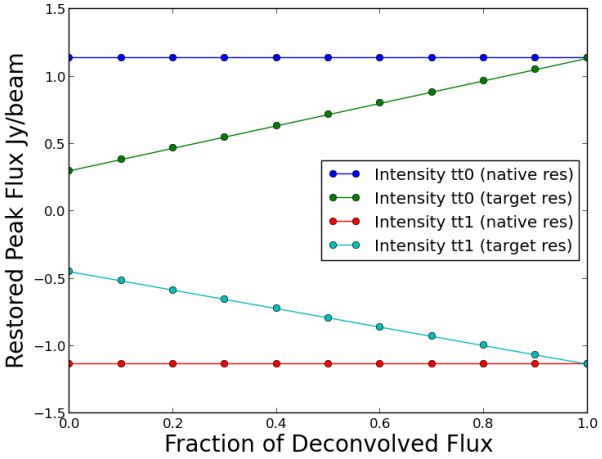
$$\mathbf{I}_{\mathrm{tt0,tar},\epsilon} \approx \underbrace{\{\kappa\boldsymbol{\delta} + (1-\kappa)\}}_{e_{\mathrm{tt0},1}} \mathbf{B}_{\mathrm{tar}} * \mathbf{I}_{\mathrm{tt0,signal}}, \tag{47}$$

$$\mathbf{I}_{\mathrm{tt1,tar},\epsilon} = \underbrace{\left\{\kappa\boldsymbol{\delta} + (1-\kappa)\mathcal{F}^{-1}\left(\frac{\mathbf{D}^f_{\mathrm{tt1,psf}}}{\mathbf{B}^f_{\mathrm{tt0,psf}}}\right)\right\}}_{\mathbf{E}_{\mathrm{tt1},1}} * \mathbf{B}_{\mathrm{tar}} * \mathbf{I}_{\mathrm{tt1,signal}}. \tag{48}$$

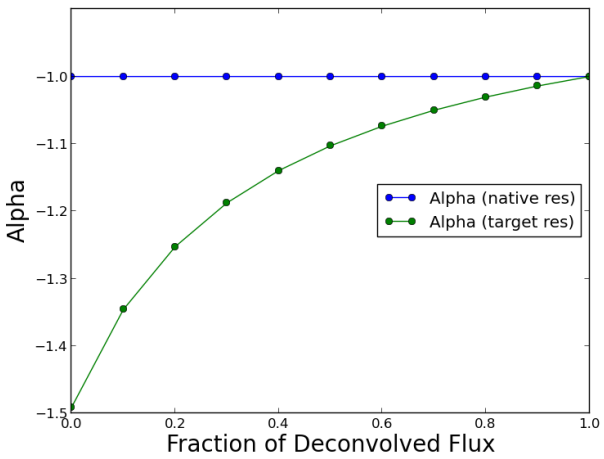
There is now no type 2 error. The peak flux of target resolution tt0 is now correct (see figure 4c). However, due to the type 1b error the flux of the target resolution tt1 image is not correct. Consequently, there is still an error in the spectral index at the image peak (see figure 4d).

6 Future Work

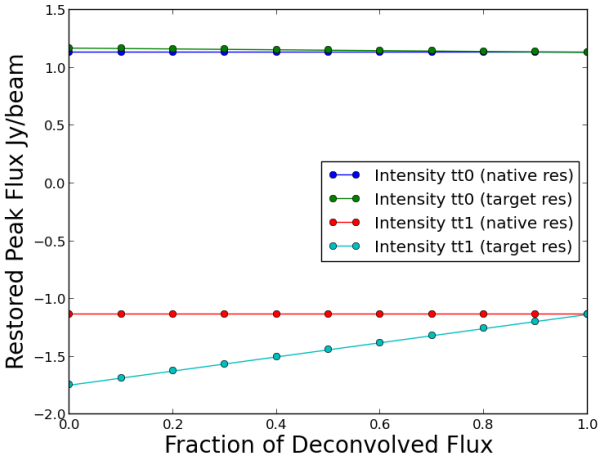
Development is under way to correct the type 1 and type 1b errors at the peak of the flux components, so that flux is conserved.



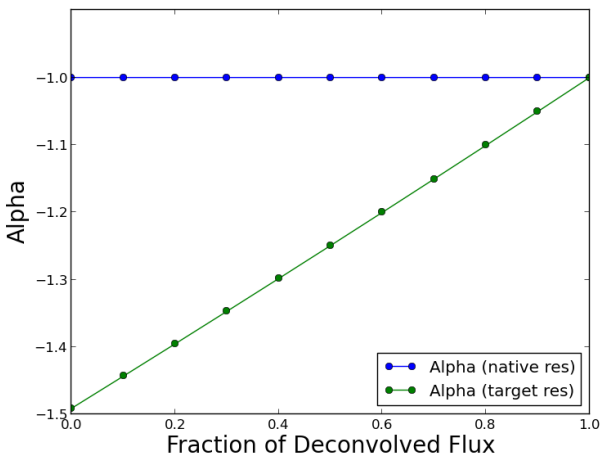
(a) Before the fix in 12148. Peak flux from tt0 and tt1 images generated using `tclean` with deconvolver = 'mtmfs' and a uv taper (no type 1 error). At the native resolution (no restoring beam and no type 1b or type 2 errors in the peak flux) the tt1 and tt0 flux is correct and constant for any fraction of deconvolved flux. When a restoring beam (target resolution) is used the type 2 error is present in the tt0 image and both the type 1b and type 2 errors are present in the tt1 image.



(b) Before the fix in 12148. The data from figure 4a is used to calculate the spectral index. For the native resolution there is no error in the spectral index α and it is constant for all fractions of deconvolved flux . When a restoring beam (target resolution) is used the type 1b and type 2 errors cause an error in the spectral index which varies with the fraction of deconvolved flux.



(c) After the fix in 12148 (no type 2 error). Peak flux from tt0 and tt1 images generated using `tclean` with deconvolver = 'mtmfs' and a uv taper (no type 1 error). At the native resolution the tt1 and tt0 peak fluxes are correct (see figure 4a) . As there is no type 2 error present, the peak flux of the tt0 image with restoring beam is now correct for all fractions of deconvolved flux. The tt1 image with restoring beam still has the type 1b error present.



(d) After the fix in 12148 (no type 2 error). The data from figure 4c is used to calculate the spectral index. For the native resolution there is no error in the spectral index α and it is constant for all fractions of deconvolved flux. When a restoring beam is used the type 1b error in the tt1 image causes an error in the spectral index which varies with the fraction of deconvolved flux.

Figure 4

7 Appendix: Code

7.1 Code for Section 3

The code in this script should be executed in python 3 (only numpy and matplotlib is required).

```
# demonstrate_gaussian_deconv.py
# This script demonstrates that B_cor = convolution(B_res , B_psf)
import numpy as np
import matplotlib.pylab as plt
from gaussian_beam_funcs import *

#Gaussian beam paramters d1, d2, t (semi-major (rad), semi-minor (rad), theta (
    ↪ rad))
psf_parms_ddt = np.array([0.7956633567810059*np.pi/(180*3600), 0.6768985390663147*
    ↪ np.pi/(180*3600), 56.512718200683594*np.pi/180])#Parameters for Gaussian
    ↪ beam fitted to PSF
A_psf = 1 #Amplitude of Gaussian beam fitted to PSF
restore_parms_ddt = np.array([1.05*np.pi/(180*3600), 0.83*np.pi/(180*3600), 88.7*
    ↪ np.pi/180]) #Restoring/Common beam
A_res = 1 #Amplitude of restoring/common beam

#Generate xy coordinate grid
Npix = 512
pix = 0.05*np.pi/(180*3600) #length of a pixal
fs = 1/pix
max_val = ((Npix-1)*0.05*np.pi/(180*3600))/2
x_vals = np.arange(-256,256)*pix
y_vals = np.arange(-256,256)*pix
X_vals = np.array([x_vals,]*Npix)
Y_vals = (np.array([y_vals,]*Npix)).T

#Calculate amplitude and and parameters for correcting beam
A_cor, correcting_parms_ddt = gaussian_deconvolve(restore_parms_ddt,psf_parms_ddt,
    ↪ A_res,A_psf)

#Generate beams
B_psf = gauss_beam_ddt(A_psf,psf_parms_ddt,X_vals,Y_vals)
B_res = gauss_beam_ddt(A_res,restore_parms_ddt,X_vals,Y_vals)
B_cor = gauss_beam_ddt(A_cor,correcting_parms_ddt,X_vals,Y_vals)

#If corrected beam was calculated correctly B_res = B_cor*B_psf
B_res_from_conv = fft_conv(B_cor,B_psf,pix)


plt.figure(101)
plt.title("Gaussian␣beam␣fitted␣to␣PSF")
plt.imshow(B_psf,aspect='auto')
plt.colorbar()
```

```
plt.show(block=False)

plt.figure(102)
plt.title("The␣Restoring␣Beam")
plt.imshow(B_res,aspect='auto')
plt.colorbar()
plt.show(block=False)

plt.figure(103)
plt.title("The␣Correcting␣Beam")
plt.imshow(B_cor,aspect='auto')
plt.colorbar()
plt.show(block=False)

plt.figure(104)
plt.title("Difference␣B_res␣-␣B_res_from_conv")
plt.imshow(B_res_from_conv - B_res,aspect='auto')
plt.colorbar()
plt.show()
```

```
 # gaussian_beam_funcs.py
import numpy as np
from scipy.fftpack import fft2, ifft2, fftshift

#Performs convolution using fft
def fft_conv(x,y,pix):
    return pix*pix*np.real(fftshift(ifft2((fft2(x))*(fft2(y)))))

#Convert Gaussian beam paramters from d1, d2, t (semi-major, semi-minor, theta)
    ↪ to a, b, g (alpha, beta, gamma)
def convert_to_abg(parms_ddt):
    d1 = parms_ddt[0]
    d2 = parms_ddt[1]
    t = parms_ddt[2]
    m = (4*np.log(2))/(d1**2)
    n = (4*np.log(2))/(d2**2)
    a = m*(np.cos(t)**2) + n*(np.sin(t)**2)
    b = 2*(m-n)*np.sin(t)*np.cos(t)
    g = m*(np.sin(t)**2) + n*(np.cos(t)**2)
    return np.array([a, b, g])

#Convert Gaussian beam paramters from a, b, g (alpha, beta, gamma) to d1, d2, t (
    ↪ semi-major, semi-minor, theta)
def convert_to_ddt(parms_abg):
    a = parms_abg[0]
    b = parms_abg[1]
    g = parms_abg[2]
    d1 = np.sqrt((8*np.log(2))/(a + g - np.sqrt(a**2 - 2*a*g + g**2 + b**2)))
```

```
    d2 = np.sqrt((8*np.log(2))/(a + g + np.sqrt(a**2 - 2*a*g + g**2 + b**2)))
    t = 0.5*np.arctan2(-b,g-a)
    return np.array([d1, d2, t])

#Calculates the abg parameters for a Fourier Transformed Gaussian
def fft_parms_abg(parms_abg):
    a = parms_abg[0]
    b = parms_abg[1]
    g = parms_abg[2]
    a_ft = (4*a*g*np.pi**2)/(4*(a**2)*g-a*(b**2))
    print(a_ft)
    b_ft = (-4*a*b*np.pi**2)/(4*(a**2)*g-a*(b**2))
    g_ft = (4*(a**2)*np.pi**2)/(4*(a**2)*g-a*(b**2))
    return np.array([a_ft, b_ft, g_ft])

#Find the amplitude and ddt parameters for the correcting beam
def gaussian_deconvolve(new_beam_parms_ddt,old_beam_parms_ddt,A_new_beam,
    ↪ A_old_beam):
    nb_abg = convert_to_abg(new_beam_parms_ddt)
    ob_abg = convert_to_abg(old_beam_parms_ddt)
    ft_nb_abg = fft_parms_abg(nb_abg)
    ft_ob_abg = fft_parms_abg(ob_abg)
    corrected_parms_abg = fft_parms_abg(ft_nb_abg-ft_ob_abg)
    corrected_parms_ddt = convert_to_ddt(corrected_parms_abg)

    A_nb_ft = ft_amp(new_beam_parms_ddt, A_new_beam)
    A_ob_ft = ft_amp(old_beam_parms_ddt, A_old_beam)
    A_cor = ift_amp(corrected_parms_ddt, A_nb_ft/A_ob_ft)

    return A_cor, corrected_parms_ddt

#Fourier transformed amplitude of beam
def ft_amp(beam_parms_ddt, A):
    return A*(np.pi*beam_parms_ddt[0]*beam_parms_ddt[1])/np.log(16)

#Inverse Fourier transformed amplitude of beam
def ift_amp(beam_parms_ddt, A):
    return A*np.log(16)/(np.pi*beam_parms_ddt[0]*beam_parms_ddt[1])

#Generating beams from parameters
def gauss_beam_ddt(amp,parms_ddt,x,y):
    d1 = parms_ddt[0]
    d2 = parms_ddt[1]
    t = parms_ddt[2]
    x_r = np.cos(t)*x + np.sin(t)*y
    y_r = -np.sin(t)*x + np.cos(t)*y
    r = 4*np.log(2)*(x_r**2)/(d1**2) + 4*np.log(2)*(y_r**2)/(d2**2)
```

```
    B = amp*np.exp(-r)
    return B

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
```

8 Code for Section 3

This script should be executed using CASA < 5.6 with the command `execfile('main_section_2_code.py')`.

```
#main_section_2_code.py
import os
import sys
import shutil
import commands
import numpy
from __main__ import default
from tasks import *
from taskinit import *
import matplotlib.pylab as plt

##This script should be used with CASA < 5.6 .
##The purpose of the script is twofold:
## 1. To show the effect of having a non Gaussian psf and not completely
   ↪ deconvoling a source.
## 2. To show the effect of the scaling error when using the restoring beam
   ↪ paramter in tclean. This scaling error has been corrected in CAS_12148
##The simulated measurement set consists of a single 1.5 Jy/beam point source at
   ↪ spectral channel 0.
##The target resolution is 120 arcsec x 120 arcsec with positionangle of 0 deg.

runim=True # run imaging code if true. Set to false if images have already been
   ↪ generated

msfile = 'refim_point.ms' #measurement set https://open-bitbucket.nrao.edu/
   ↪ projects/CASA/repos/casa-data/browse/regression/unittest/clean/refimager/
   ↪ refim_point.ms

gains = np.linspace(0,1,11) #loop gains (only 1 iteration)
spw = '0:0'

if runim==True :
    #clear previous generated images
```

```
os.system('rm␣-r␣results/restore_*')
os.system('rm␣-r␣results/native_*')

for g in gains:
    print('Generating␣Images␣for␣loop␣gain␣' + str(g))

    ###Images at native resolution (Gaussian beam fitted to PSF) that are then
       ↪  smoothed using imsmooth###
    #Natural Weighting#
    outfile = 'results/native_natural' + str(g)
    tclean(vis=msfile,imagename=outfile,
       imsize=100,cell='10.0arcsec',interpolation='nearest',
       interactive=0,gain=g,niter=1,deconvolver='clark', spw=spw)
    outfile_smooth = outfile + '.smoothed.image'
    imsmooth(outfile+'.image', targetres=True, major='120.0arcsec', minor='
       ↪ 120.0arcsec',pa='0deg',outfile=outfile_smooth,overwrite=True)

    #Uniform Weighting#
    outfile = 'results/native_uniform' + str(g)
    tclean(vis=msfile,imagename=outfile,
          imsize=100,cell='10.0arcsec',interpolation='nearest',
          interactive=0,gain=g,niter=1,weighting='uniform',deconvolver='clark',
             ↪ spw=spw)
    outfile_smooth = outfile + '.smoothed.image'
    imsmooth(outfile+'.image', targetres=True, major='120.0arcsec', minor='
       ↪ 120.0arcsec', pa='0deg',outfile=outfile_smooth,overwrite=True)

    #Natural Weighting with uvtaper (makes the PSF very close to a Gaussian)
    outfile = 'results/native_taper' + str(g)
    tclean(vis=msfile,imagename=outfile,
          imsize=100,cell='10.0arcsec',interpolation='nearest',
          interactive=0,gain=g,niter=1, weighting='natural', uvtaper=['1.5
             ↪ klambda'], deconvolver='clark',spw=spw)
    outfile_smooth = outfile + '.smoothed.image'
    imsmooth(outfile+'.image', targetres=True, major='120.0arcsec', minor='
       ↪ 120.0arcsec', pa='0deg',outfile=outfile_smooth,overwrite=True)

    ###Images Generated using Restoring beam with Natural Weighting and
       ↪ UVtaper###
    #The natural weighting along with UVtaper is chosen such that the psf is
       ↪ very close to a Gaussian, so that only the effect of the scaling
       ↪ error is present.
    outfile = 'results/restore_' + str(g)
    tclean(vis=msfile,imagename=outfile,
       imsize=100,cell='10.0arcsec',interpolation='nearest',
       interactive=0,gain=g,niter=1,restoringbeam='120.0arcsec', weighting='
          ↪ natural', uvtaper=['1.5klambda'],deconvolver='clark',spw=spw)
```

```
A = 1.5 #Deconvolved amplitude of source
gains = np.array([0.0,0.1,0.2,0.4,0.6,0.8,1.0])
kappas = np.zeros(gains.shape) #The fraction of the source flux that has been
    ↪ deconvoled
peak_smoothed_natural = np.zeros(gains.shape)
peak_smoothed_uniform = np.zeros(gains.shape)
peak_smoothed_taper = np.zeros(gains.shape)
peak_restored = np.zeros(gains.shape)

#Calculate r_Omega
#All units are in arcsec. d1 and d2 are the semi-major axis and semi-minor axis,
    ↪ respectively.
outfile = 'results/restore_0.0.psf'
psf_restored = imhead(imagename=outfile)
d1_psf = psf_restored['restoringbeam']['major']['value']
d2_psf = psf_restored['restoringbeam']['minor']['value']

d1_res = 120
d2_res = 120

r_Omega = (d1_res*d2_res)/(d1_psf*d2_psf)

print('r_Omega␣calculated␣using␣the␣parameters␣of␣the␣restoring␣beam␣and␣the␣
    ↪ Gaussian␣fitted␣psf␣' + str(r_Omega))


for i in range(gains.shape[0]):
    ###Calculate the percentage of flux deconvolved. Should be the same as the
        ↪ gain.
    outfile = 'results/native_natural' + str(gains[i])
    I_residual = np.array(imval(imagename=outfile+'.residual',chans='0')['data'])
    kappas[i] = (A-I_residual)/A

    ###The peak flux for the native resolution images that have been smoothed (
        ↪ natural weight)
    outfile_smooth = outfile + '.smoothed.image'
    peak_smoothed_natural[i] = np.array(imval(imagename=outfile_smooth,chans='0')['
        ↪ data'])

    ###The peak flux for the native resolution images that have been smoothed (
        ↪ uniform weight)
    outfile = 'results/native_uniform' + str(gains[i])
    outfile_smooth = outfile + '.smoothed.image'
    peak_smoothed_uniform[i] = np.array(imval(imagename=outfile_smooth,chans='0')['
        ↪ data'])

    ###The peak flux for the native resolution images that have been smoothed (uv
        ↪  tapering)
```

```
    outfile = 'results/native_taper' + str(gains[i])
    outfile_smooth = outfile + '.smoothed.image'
    peak_smoothed_taper[i] = np.array(imval(imagename=outfile_smooth,chans='0')['
        ↪ data'])

    ###The peak flux for the image generated using restoringbeam parameter in
        ↪ tclean (uv tapering).
    ###This is where the scaling error appears.
    outfile = 'results/restore_' + str(gains[i]) + '.image'
    peak_restored[i] = np.array(imval(imagename=outfile,chans='0')['data'])

plt.figure(0)
plt.plot(kappas,peak_smoothed_natural,label='Natural')
plt.plot(kappas,peak_smoothed_uniform,label='Uniform')
plt.plot(kappas,peak_smoothed_taper,label='Taper')
plt.xlabel('Fraction␣of␣Deconvolved␣Flux',fontsize=20)
plt.ylabel('Smoothed␣Peak␣Flux',fontsize=20)
plt.xlim([0,1])
plt.legend(loc=4)
plt.show()

plt.figure(1)
plt.plot(kappas,peak_restored)
plt.xlabel('Fraction␣of␣Deconvolved␣Flux',fontsize=20)
plt.ylabel('Restored␣Peak␣Flux',fontsize=20)
plt.xlim([0,1])
plt.show()

#Calculate r_Omega by using equation 28
r_Omega_restored = np.mean(((1-kappas[:-1])*A)/(peak_restored[:-1]-kappas[:-1]*A))
    ↪  #the last term is skipped since 1-kappa = 0, see equation 27.

print('Value␣for␣r_Omega␣calculated␣using␣image␣data␣from␣CASA␣' + str(
    ↪ r_Omega_restored))
```

9 Code for Section 4

```
import os
import sys
import shutil
import commands
import numpy
from __main__ import default
from tasks import *
from taskinit import *
import pylab as pl
```

```
##This script should be used with CASA < 5.6 and CAS_12148 branch.
##The purpose of the script is twofold:
## 1. To show the effect of having a non Gaussian psf and not completely
   ↪ deconvoling a source.
## 2. To show the effect of the scaling error when using the restoring beam
   ↪ paramter in tclean. This scaling error has been corrected in CAS_12148
##The simulated measurement set consists of a single 1.5 Jy/beam point source at
   ↪ spectral channel 0.
##The target resolution is 120 arcsec x 120 arcsec with positionangle of 0 deg.

runim=False # run imaging code if true. Set to false if images have already been
   ↪ generated.
impre='12148' #impre='5.4', set version of CASA being used.
#impre='5.4'

msfile = '../refim_point.ms' #measurement set https://open-bitbucket.nrao.edu/
   ↪ projects/CASA/repos/casa-data/browse/regression/unittest/clean/refimager/
   ↪ refim_point.ms

def getpix(imname):
    ia.open(imname)
    val = ia.pixelvalue( [50,50,0,0] )['value']['value']
    ia.close()
    return val

def plotit(xs):
    pl.figure(121)
    pl.plot( xs, im_nativeres, 'o-', label='Intensity tt0 (native res)' )
    pl.plot( xs, im_targetres, 'o-', label='Intensity tt0 (target res)' )
    pl.plot( xs, im1_nativeres, 'o-', label='Intensity tt1 (native res)' )
    pl.plot( xs, im1_targetres, 'o-', label='Intensity tt1 (target res)' )
    pl.xlabel('Fraction of Deconvolved Flux',fontsize=20)
    pl.ylabel('Restored Peak Flux Jy/beam',fontsize=20)
    #pl.title('Taylor Coefficients of Spectral Flux Model',fontsize=20)
    pl.legend(loc='center right')
    pl.figure(122)
    pl.plot( xs, al_nativeres, 'o-', label='Alpha (native res)' )
    pl.plot( xs, al_targetres, 'o-', label='Alpha (target res)' )
    pl.legend(loc='center right')
    pl.ylim(ymin=-1.5, ymax=-0.9)
    #pl.title('Spectral Index',fontsize=20)
    pl.xlabel('Fraction of Deconvolved Flux',fontsize=20)
    pl.ylabel('Alpha',fontsize=20)
    plt.legend(loc=4)

if runim==True:
    os.system('rm -rf '+impre+'.*')
```

```
N = 11 #number of gains to test
gains = np.linspace(0,1,N)
spw='0:0,0:19'

im_nativeres = np.zeros(gains.shape)
im_targetres = np.zeros(gains.shape)
im1_nativeres = np.zeros(gains.shape)
im1_targetres = np.zeros(gains.shape)
al_nativeres = np.zeros(gains.shape)
al_targetres = np.zeros(gains.shape)
mod_nativeres = np.zeros(gains.shape)
mod_targetres = np.zeros(gains.shape)

k = 0
for g in gains:
    print('Generating␣Images␣for␣loop␣gain␣' + str(g))

    #Needed due to iter control problem
    if g==0.0:
        niter=0
    else:
        niter=1

    lab = str(g)

    imname = impre+'.nativeres.'+lab
    if runim==True:
        print('doing␣image')
        ret = tclean(vis=msfile,imagename=imname,spw=spw, imsize=100,cell='10.0
            ↪ arcsec', interactive=0,gain=g,niter=niter,threshold=1e-04,deconvolver
            ↪ ='mtmfs',specmode='mfs', weighting='uniform',uvtaper=['1.5klambda'])
    im_nativeres[k] = getpix(imname+'.image.tt0')
    im1_nativeres[k] = getpix(imname+'.image.tt1')
    al_nativeres[k] = getpix(imname+'.alpha')

    imname = impre+'.targetres.'+lab
    if runim==True:
        ret = tclean(vis=msfile,imagename=imname,spw=spw, imsize=100,cell='10.0
            ↪ arcsec', interactive=0,gain=g,niter=niter,threshold=1e-04,
            ↪ restoringbeam='120.0arcsec',deconvolver='mtmfs',specmode='mfs',
            ↪ weighting='uniform',uvtaper=['1.5klambda'])
    im_targetres[k] = getpix(imname+'.image.tt0')
    im1_targetres[k] = getpix(imname+'.image.tt1')
    al_targetres[k] = getpix(imname+'.alpha')

    k = k+1

plotit(gains)
```