Astronomy & Astrophysics

Efficient wide-field radio interferometry response

Philipp Arras[1,2,3], Martin Reinecke[1], Rüdiger Westermann[3], and Torsten A. Enßlin[1,2]

[1] Max-Planck Institut für Astrophysik, Karl-Schwarzschild-Str. 1, 85748 Garching, Germany
e-mail: parras@mpa-garching.mpg.de
[2] Ludwig-Maximilians-Universität München (LMU), Geschwister-Scholl-Platz 1, 80539 München, Germany
[3] Technische Universität München (TUM), Boltzmannstr. 3, 85748 Garching, Germany

ABSTRACT

Radio interferometers do not measure the sky brightness distribution directly, but measure a modified Fourier transform of it. Imaging algorithms therefore need a computational representation of the linear measurement operator and its adjoint, regardless of the specific chosen imaging algorithm. In this paper, we present a C++ implementation of the radio interferometric measurement operator for wide-field measurements that is based on so-called improved w-stacking. It can provide high accuracy (down to $\approx 10^{-12}$), is based on a new gridding kernel that allows smaller kernel support for given accuracy, dynamically chooses kernel, kernel support, and oversampling factor for maximum performance, uses piece-wise polynomial approximation for cheap evaluations of the gridding kernel, treats the visibilities in cache-friendly order, uses explicit vectorisation if available, and comes with a parallelisation scheme that scales well also in the adjoint direction (which is a problem for many previous implementations). The implementation has a small memory footprint in the sense that temporary internal data structures are much smaller than the respective input and output data, allowing in-memory processing of data sets that needed to be read from disk or distributed across several compute nodes before.

Key words. instrumentation: interferometers – techniques: interferometric – methods: numerical – methods: data analysis

1. Introduction

The central data analysis task in radio interferometry derives the location-dependent sky brightness distribution $I(l, m)$ from a set of complex-valued measured visibilities d_k. In the noise-less case they are related by the expression (e.g. Richard Thompson et al. 2017)

$$d_k = \iint \frac{e^{2\pi i \lambda_k^{-1} \tilde{w}_k(n(l,m)-1)}}{n(l,m)} I(l,m) \, e^{-2\pi i \lambda_k^{-1}(\tilde{u}_k l + \tilde{v}_k m)} \, \mathrm{d}l \, \mathrm{d}m. \quad (1)$$

Here, l, m, and $n := \sqrt{1 - l^2 - m^2}$ are direction cosines with respect to the central observation axis, while $\tilde{u}_k$, $\tilde{v}_k$, and $\tilde{w}_k$ are the coordinates of the baselines in metres and λ_k are the observation wavelengths. When we assume that $I(l, m)$ is approximated by discretised values on a Cartesian (l, m) grid, the double integral corresponds to a discrete Fourier transform. The entries d_k of the data vector d correspond to delta peak readouts of the three-dimensional Fourier transformed sky brightness at Fourier location (u_k, v_k, w_k), which are commonly called 'visibilities'. It suffices to discuss the noise-less case here. While taking the noise into account is the task of the chosen imaging algorithm, all such algorithms need an implementation of Eq. (1).

Typical problem sizes range from 10^6 to beyond 10^9 visibilities, fields of view can reach significant fractions of the hemisphere, and image dimensions exceed $10\,000 \times 10\,000$ pixels. It is evident that naïve application of Eq. (1) becomes prohibitively expensive at these parameters; a single evaluation would already require $\approx 10^{17}$ calls to the complex exponential function.

Massive acceleration can be achieved by using 'convolutional gridding' (in other fields often called 'non-uniform fast Fourier transform'; Dutt & Rokhlin 1993). Here, the information contained in the d_k is transferred onto a regular Cartesian grid by convolving the delta peak readouts at (u_k, v_k, w_k) with an appropriately chosen kernel function, which is evaluated at surrounding (u, v) grid points. Transformation between u, v and l, m can now be carried out quickly by means of a two-dimensional fast Fourier transform (FFT; Cooley & Tukey 1965), and the smoothing caused by the convolution with the kernel is compensated for by dividing the $I(l, m)$ by the Fourier-transformed kernel.

When the term $e^{-2\pi i \lambda^{-1} \tilde{w}(n-1)}/n$ is very close to 1, no further optimisation steps are required. This criterion is not fulfilled for non-planar instruments and for wide-field observations. Therefore the visibilities need to be gridded onto several uv-planes with different w, which are Fourier-transformed and corrected separately. Perley (1999) has pointed out that Eq. (1) can be written as a three-dimensional Fourier transform. Based on this idea, Ye (2019) applied the convolutional gridding algorithm not only for the uv-coordinates, but also for the w-direction. Because this approach naturally generalises w-stacking (Offringa et al. 2014) to use gridding in the w-direction as well, we propose the term 'w-gridding' instead of the term 'improved w-stacking' (Ye 2019, Ye et al. 2021).

This paper does not present any new insights into the individual components of the radio interferometric measurement operator implementation (except for the introduction of a tuned gridding kernel in Sect. 3.2); our code only makes use of algorithms that are already publicly available. Instead, our main point is to demonstrate how significant advances in performance and accuracy can be achieved by appropriate selection of individual components and their efficient implementation. Our implementation has been integrated into the well-known imaging tool `wsclean`[1] (Offringa et al. 2014) since version 2.9, where it can be selected through the `-use-wgridder` flag, and the imaging

---

[1] https://gitlab.com/aroffringa/wsclean

toolkit `codex-africanus`[2]. Furthermore, the implementation presented here has been used in Arras et al. (2020, 2021), for instance.

Section 2 introduces the notation used in this paper and summarises the algorithmic approach to numerically approximate Eq. (1) and its adjoint. Section 3 describes all algorithmic components in detail from a computational point of view, and Sect. 4 lists the design goals for the new code, which influence the choice of algorithmic components from the set given in Sect. 3. Here we also list a number of additional optimisations to improve overall performance. The new code is validated against discrete Fourier transforms in Sect. 5, and an analysis of its scaling behaviour as well as a performance comparison with other publicly available packages is presented in Sect. 6.

2. Notation and formal derivation of the algorithm

The data that are taken by radio interferometers are called 'visibilities'. Equation (1) already shows that the operation that is to be implemented is similar to a Fourier transform modulated by a phase term. In the following, we introduce all notation that is required to describe the algorithm and present the three-dimensional (de)gridding approach from Ye (2019) in this notation.

Let $\lambda \in \mathbb{R}^{n_v}$ be the vector of observing wavelengths in metres and $(\tilde{u}, \tilde{v}, \tilde{w})$ the coordinates of the baselines in metres, each of which are elements of $\mathbb{R}^{n_r}$. In other words, n_v and n_r are the number of observing wavelengths and number of rows of the data set, respectively. Then, the effective baseline coordinates (u, v, w) are defined as

$$u := \tilde{u} \otimes \lambda^{-1}, \quad v := \tilde{v} \otimes \lambda^{-1}, \quad w := \tilde{w} \otimes \lambda^{-1}. \quad (2)$$

These are the effective locations of the sampling points in Fourier space. To simplify the notation, we view the above three coordinates as elements of a simple vector space, for example, $u \in \mathbb{R}^{n_d}$ with $n_d = n_r n_v$. Because the measurement Eq. (1) is to be evaluated on a computer, it needs to be discretised,

$$(R_0 I)_k := \sum_{l \in L} \sum_{m \in M} \frac{e^{-2\pi i [u_k l + v_k m - w_k(n_{lm} - 1)]} I_{lm}}{n_{lm}}, \quad k \in \{0, \ldots, n_d - 1\}, \quad (3)$$

where R_0 is the (accurate) response operator defined by the right-hand side of the equation, and L, M are the sets of direction cosines of the pixels of the discretised sky brightness distribution in the two orthogonal directions. $(\Delta l, \Delta m)$ are the pixel sizes, and (n_l, n_m) are the number of pixels. Then, formally, L and M can be defined as

$$L := \left\{ \left(-\tfrac{n_l}{2} + j\right) \Delta l \mid j \in \{0, \ldots, n_l - 1\} \right\}, \quad (4)$$

$$M := \left\{ \left(-\tfrac{n_m}{2} + j\right) \Delta m \mid j \in \{0, \ldots, n_m - 1\} \right\}. \quad (5)$$

It is apparent that computing Eq. (3) is prohibitively expensive because the whole sum needs to be performed for each data index k individually. As a solution, the convolution theorem can be applied in order to replace the Fourier transform by an FFT that can be reused for all data points. As it stands, Eq. (3) is not a pure Fourier transform because of the phase term $w_k(n_{lm} - 1)$. As discussed above, we follow Perley (1999) and introduce an

---
[2] `https://github.com/ska-sa/codex-africanus`

auxiliary Fourier integration in which w and $n_{lm} - 1$ are viewed as Fourier-conjugate variables,

$$(RI)_k = \sum_{l \in L} \sum_{m \in M} \int_{\tilde{n} \in \mathbb{R}} e^{-2\pi i [u_k l + v_k m + w_k \tilde{n}]} \frac{\delta(\tilde{n} - (1 - n_{lm}))}{n_{lm}} I_{lm} \, d\tilde{n}. \quad (6)$$

The next goal is to replace the above three-dimensional non-equidistant Fourier transform by an equidistant one. This can be done by expressing a visibility d_k as a convolution of a field defined on a grid with a convolution kernel. This convolution is undone by dividing by its Fourier transform in sky space.

For this, we need to define the convolution kernel. Let $\phi : \mathbb{R} \to \mathbb{R}^+$ be a function that is point-symmetric around 0 and has compact support $\text{supp}(\phi) = [-\frac{\alpha}{2}, \frac{\alpha}{2}]$ with the kernel support size $\alpha \in \mathbb{N}$. In other words, the kernel function is zero outside a symmetric integer-length interval around zero. In practice, this means that every visibility is gridded onto the $\alpha \times \alpha$ uv-grid points that are closest to it. We use ϕ as convolution kernel to interpolate all three grid dimensions. Let $\psi : [-\frac{1}{2}, \frac{1}{2}] \to \mathbb{R}$ be its Fourier transform: $\psi(k) := \int_{-\infty}^{\infty} \phi(x) e^{ikx} \, dx$. ψ needs to be defined only on $[-\frac{1}{2}, \frac{1}{2}]$ because ϕ is evaluated on a grid with pixel size 1.

Now, the (discrete) convolution theorem can be applied to turn the sums in Eq. (6) into a discrete Fourier transform followed by a periodic convolution on an oversampled grid (with oversampling factor σ) and to turn the integral over $\tilde{n}$ into a regular convolution. Some degree of oversampling ($\sigma > 1$) is required to lower the error of the algorithm to the required levels; ultimately, the error depends on σ, α, and the kernel ϕ. Specifically for the w-direction, using the coordinate transform $c(x) = w_k + x\Delta w$ and the definition of ψ,

$$e^{2\pi i (n_{lm} - 1) w_k} \psi\big([n_{lm} - 1]\Delta w\big) = \int_{-\infty}^{\infty} e^{2\pi i (n_{lm} - 1)(w_k + \Delta w x)} \phi(x) \, dx \quad (7)$$

$$= \int_{-\infty}^{\infty} e^{2\pi i (n_{lm} - 1)c} \phi\left(\frac{c - w_k}{\Delta w}\right) \frac{dc}{\Delta w} \quad (8)$$

$$\approx \sum_{c \in W} e^{2\pi i (n_{lm} - 1)c} \phi\left(\frac{c - w_k}{\Delta w}\right), \quad (9)$$

with $W = \left\{ w_\circ + j\Delta w \mid j \in \{0, \ldots, N_w - 1\} \right\}$, it follows that

$$e^{2\pi i (n_{lm} - 1) w_k} \approx \frac{\sum_{c \in W} e^{2\pi i (n_{lm} - 1)c} \phi\left(\frac{c - w_k}{\Delta w}\right)}{\psi\big([n_{lm} - 1]\Delta w\big)}. \quad (10)$$

This expression replaces the w-term in Eq. (6) below. The idea of rewriting the w-term as a convolution was first presented in Ye (2019). $w_\circ$, N_w, and Δw denote the as yet unspecified position of the w-plane with the lowest w-value, the number of w-planes, and the distance between neighbouring w-planes, respectively. The approximation Eq. (9) is only sensible for all $l \in L, m \in M$ and all k if Δw is small enough. The proper condition is given by the Nyquist-Shannon sampling theorem (Ye 2019),

$$\max_{(l,m) \in L \times M} 2\Delta w \, \sigma |n_{lm} - 1| \leq 1. \quad (11)$$

The factor σ appears because the accuracy of a given gridding kernel ϕ depends on the oversampling factor. Therefore, the optimal, that is, largest possible, Δw is

$$\Delta w = \min_{(l,m) \in L \times M} \frac{1}{2\sigma |n_{lm} - 1|}. \quad (12)$$

For a given σ this determines Δw. $w_\circ$ and N_w are still unspecified. Combining Eqs. (6) and (10) leads to the final approximation of the measurement equation,

$$(RI)_k := \sum_{a \in U} \sum_{b \in V} \sum_{c \in W} \Phi_k(a,b,c) \sum_{l \in L} \sum_{m \in M} e^{-2\pi i[al+bm+c(n_{lm}-1)]} \frac{I_{lm}}{n_{lm}\Psi_{lm}},$$

(13)

where R is the linear map that approximates R_0 in our implementation, and Φ and Ψ are the threefold outer product of ϕ and ψ, respectively,

$$\Phi_k(a,b,c) = \phi(N_u \chi(a - u_k \Delta l)) \phi(N_v \chi(b - v_k \Delta m)) \phi\left(\frac{c-w_k}{\Delta w}\right),$$

(14)

$$\Psi_{lm} = \psi\left(\frac{l}{\sigma n_x \Delta x}\right) \psi\left(\frac{m}{\sigma n_y \Delta y}\right) \psi([n_{lm}-1]\Delta w),$$

(15)

with $\chi(a) = a - \lfloor a \rfloor - 0.5$ where $\lfloor a \rfloor := \max\{n \in \mathbb{Z} \,|\, n \le a\}$. To define the sets U, V, and W, the discretisation in uvw-space needs to be worked out. The number of pixels in discretised uv-space is controlled by the oversampling factor σ,

$$N_l = \lceil \sigma n_l \rceil, N_m = \lceil \sigma n_m \rceil, \text{for } \sigma > 1,$$

(16)

where $\lceil a \rceil := \min\{n \in \mathbb{Z} \,|\, n \ge a\}$. Thus, the set of pixels of the discretised uvw-space is given by

$$U = \left\{ -\frac{1}{2} + \frac{j}{N_u} \,\Big|\, j \in \{0, \ldots, N_u - 1\} \right\},$$

(17)

$$V = \left\{ -\frac{1}{2} + \frac{j}{N_v} \,\Big|\, j \in \{0, \ldots, N_v - 1\} \right\}.$$

(18)

For the w-dimension we can assume $w_k \ge 0$ for all k without loss of generality because the transformation

$$(u_k, v_k, w_k, d_k) \rightarrow (-u_k, -v_k, -w_k, d_k^*)$$

(19)

leaves Eq. (3) invariant individually for each k. Because of this Hermitian symmetry, only half of the three-dimensional Fourier space needs to be represented in computer memory.

For a given Δw, the first w-plane is located at

$$w_\circ = \min_k w_k - \frac{\Delta w (\alpha - 1)}{2},$$

(20)

that is, half of the kernel width subtracted from the minimum w-value, and the total number of w-planes N_w is

$$N_w = \frac{\max_k w_k - \min_k w_k}{\Delta w} + \alpha,$$

(21)

because below the minimum and above the maximum w-value, half a kernel width needs to be added in order to be able to grid the respective visibilities with extreme w-values.

In Eq. (13), we can view the sky brightness distribution I as element of $\mathbb{R}^{n_l n_m}$ and $d \in \mathbb{C}^{n_k}$. Then Eq. (13) can be written as $d = R(I)$ with $R : \mathbb{R}^{n_l n_m} \rightarrow \mathbb{C}^{n_k}$ being a $\mathbb{R}$-linear map. In imaging algorithms this linear map often appears in the context of functionals that are optimised, for example, a negative log-likelihood or a simple $\chi^2 = |d - R(I)|^2$ functional between data and expected sky response. To compute the gradient (and potentially higher derivatives) of such functionals, not only R, but also $R^\dagger$, the adjoint, is needed. It can be obtained from Eq. (13) by reversing the order of all operations and taking the conjugate of

the involved complex numbers. In the case at hand, it is given by

$$(R^\dagger d)_{lm} = \frac{1}{n_{lm}\Psi_{lm}} \sum_{a \in U} \sum_{b \in V} \sum_{c \in W} e^{2\pi i[al+bm+c(n_{lm}-1)]} \sum_k \Phi_k(a,b,c) \, d_k.$$

(22)

Here we can already observe that parallelisation over the data index k is more difficult in Eq. (22) than in Eq. (13). In Eq. (22), the grid in Fourier space is subject to concurrent write accesses, whereas in Eq. (13), it is only read concurrently, which is less problematic. In Sect. 4.4 we discuss this in more detail and present a parallelisation strategy that scales well in both directions.

All in all, the scheme Eq. (13), which approximates the discretised version Eq. (3) of the radio interferometric response function Eq. (1), has been derived. That it can be computed efficiently is shown in the subsequent sections. The choice of the gridding kernel function ϕ, the kernel support α, and the oversampling factor σ have not yet been discussed. Tuning these three quantities with respect to each other controls the achievable accuracy and the performance of the algorithm (see Sect. 3.2).

3. Algorithmic elements

Equation (13) prescribes a non-equidistant Fourier transform that is carried out with the help of the as yet unspecified gridding kernel Φ. Its choice is characterised by a trade-off between accuracy (larger kernel support α and/or oversampling factor σ) and computational cost. As a criterion for assessing the accuracy of a kernel, we use a modified version of the least-misfit function approach from Ye et al. (2020).

3.1. Gridding and degridding and treatment of the w-term

For the implementation, Eq. (13) is reordered in the following way:

$$(RI)_k = \sum_{c \in W} \left[\sum_{a \in U} \sum_{b \in V} \Phi_k(a,b,c) \sum_{l \in L} \sum_{m \in M} e^{-2\pi i[al+bm]} \tilde{I}_{lmc} \right]$$

(23)

$$\tilde{I}_{lmc} := e^{2\pi i c(1-n_{lm})} \tilde{I}_{lm}$$

(24)

$$\tilde{I}_{lm} := \frac{I_{lm}}{n_{lm}\Psi_{lm}}.$$

(25)

In other words, first the geometric term n and the gridding correction Ψ are applied to the input I_{lm} Eq. (25). Then, the w-planes are handled one after another. For every w-plane the phase term $e^{2\pi i c(1-n_{lm})}$, called w-screen, is applied to the image Eq. (24). This is followed by the Fourier transform and the degridding procedure with Φ (bracketed term in Eq. (23)). Finally, the contributions from all w-planes are accumulated by the sum over $c \in W$ to obtain the visibility d_k.

For the adjoint direction, Eq. (22) is reordered to

$$(R^\dagger d)_{lm} = \frac{1}{n_{lm}\Psi_{lm}} \sum_{c \in W} e^{-2\pi i c(1-n_{lm})} H_c,$$

(26)

$$H_c := \sum_{a \in U} \sum_{b \in V} e^{2\pi i[al+bm]} \sum_k \Phi_k(a,b,c) \, d_k.$$

(27)

In words, the w-planes are handled one after another again. First, the visibilities that belong to the current w-plane are gridded onto a two-dimensional grid with Φ and the two-dimensional Fourier transform is applied Eq. (27). Then, its result H_c is multiplied

with the complex conjugate w-screen and the contributions from w-planes to the image are accumulated by the sum over $c \in W$ Eq. (26). Finally, the gridding correction Ψ_{lm} and the geometric factor n_{lm} are applied.

The number of iterations in the loop over the w-planes W can be reduced by up to a factor of two by restricting the w coordinate to $w \geq 0$ with the help of the Hermitian symmetry Eq. (19). The implementation scheme described above highlights that the choice of the kernel shape ϕ and its evaluation are crucial to the performance of the algorithm: The support α should be small in order to reduce memory accesses and kernel evaluations. At the same time, the oversampling factor σ needs to be small such that the Fourier transforms do not dominate the run time. Additionally, the kernel itself needs to be evaluated with high accuracy, while at the same time, its computation should be very fast.

3.2. Kernel shape

As already mentioned, the shape of the employed kernel function ϕ has a strong effect on the accuracy of the gridding and degridding algorithms. The historical evolution of preferred kernels is too rich to be discussed here in full, but see Ye et al. (2020) for an astronomy-centred background and Barnett et al. (2019) for a more engineering-centred point of view.

It appears that the kernel shape accepted as 'optimal' amongst radio astronomers is the spheroidal function as described by Schwab (1980). This function maximises the energy in the main lobe of the Fourier-transformed kernel compared to the total energy, which is essential to suppress aliasing artefacts.

However, this concept of optimality only holds under the assumption that gridding and degridding are carried out without any oversampling of the uv-grid and the corresponding trimming of the resulting dirty image. While this may have been the default scenario at the time this memorandum was written, most currently employed gridding algorithms use some degree of oversampling and trimming (i.e. $\sigma > 1$), which requires restating the optimality criterion: instead of trying to minimise the errors over the entire dirty image, the task now is to minimise the error only in the part of the dirty image that is retained after trimming, while errors in the trimmed part may be arbitrarily high. More quantitatively: Given a kernel support of α cells and an oversampling factor of σ, a kernel shape is sought that produces the lowest maximum error within the parts of the dirty image that are not trimmed.

Ye et al. (2020) demonstrated an approach to determine non-analytic optimal kernels. However, very good results can also be obtained with rather simple analytical expressions. Barnett et al. (2019) presented the one-parameter kernel called 'exponential of a semicircle kernel' or 'ES kernel',

$$\phi_\beta : \left[-\frac{\alpha}{2}, \frac{\alpha}{2}\right] \to \mathbb{R}^+, \ x \mapsto \exp\left(\alpha\beta\left[\sqrt{1-(2x/\alpha)^2}-1\right]\right), \quad (28)$$

for $\beta > 0$. In the following, we use a two-parameter version derived from this,

$$\phi_{\beta\mu} : \left[-\frac{\alpha}{2}, \frac{\alpha}{2}\right] \to \mathbb{R}^+, \ x \mapsto \exp\left(\alpha\beta\left[\left(1-(2x/\alpha)^2\right)^\mu-1\right]\right), \quad (29)$$

for $\beta > 0$ and $\mu > 0$ and call it 'modified ES kernel'.

To determine optimal values for the two parameters for given α and σ, we use the prescription described in Ye et al. (2020). The idea is to consider the squared difference between the outputs of the accurate and the approximate adjoint response operator R_0 and R. Without loss of generality, we restrict the following

analysis to the case of a one-dimensional non-equidistant Fourier transform. For readability, we define $\tilde{\psi}(x) := \psi\left(\frac{x}{\sigma n_x \Delta x}\right)$ and $\tilde{\phi}_k(a) := \phi\left(N_u \chi(a - u_k \Delta l)\right)$ and

$$(\tilde{R}_0^\dagger d)(x) := \sum_k d_k e^{2\pi i u_k x}, \quad (30)$$

$$(\tilde{R}^\dagger d)(x) := \tilde{\psi}(x)^{-1} \sum_k d_k \sum_{a \in U} \tilde{\phi}_k(a) \, e^{2\pi i a x}. \quad (31)$$

Using the Cauchy-Schwarz inequality, the squared error can be bounded from above with

$$\left|(\tilde{R}_0 d - \tilde{R} d)(x)\right|^2 = \left|\sum_k d_k e^{2\pi i u_k x}\left(1 - \sum_{a \in U} \frac{e^{2\pi i (a - u_k) x} \tilde{\phi}_k(a)}{\tilde{\psi}(x)}\right)\right|^2 \quad (32)$$

$$\leq \left(\sum_k |d_k|^2\right) \sum_k \left|1 - \sum_{a \in U} \frac{e^{2\pi i (a - u_k) x} \tilde{\phi}_k(a)}{\tilde{\psi}(x)}\right|^2. \quad (33)$$

The first term of the right-hand side of the inequality is purely data dependent and therefore not relevant in quantifying the (upper limit of the) approximation error of the linear map $R^\dagger$. The actual approximation error does depend on the data d, and for a given data vector, more accurate approximation schemes could be derived in principle. However, because generic statements about d are difficult to make and a data-independent generic kernel is desired here, we optimise the right-hand side of the inequality. If the number of visibilities is large (tests have shown that in generic setups already > 10 visibilities suffice), the values of $(a - u_k)x \mod 2\pi$ sample the interval $[0, 1)$ sufficiently uniformly. Then the second term is approximately proportional to the data-independent integral

$$l^2(x) := \int_0^1 \left|1 - \sum_{a \in U} \frac{e^{2\pi i (a - v) x} \tilde{\phi}_k(a)}{\tilde{\psi}(x)}\right|^2 \, dv. \quad (34)$$

Because the actual error is quantified by $l(x)$, we call $l(x)$ the 'map error function' in contrast to Ye et al. (2020), who used this name for $l^2(x)$. $l(x)$ depends on the choice of the functional form of ϕ, the kernel support α, and the oversampling factor σ. Ye et al. (2020) used Eq. (34) in a least-squares fashion to determine the 'optimal gridding kernel' for a given α and σ.

We propose to use Eq. (34) slightly differently. Instead of the L2-norm, we use the supremum norm to minimise it because the error should be guaranteed to be below the accuracy specified by the user for all x. Additionally, we use the two-parameter modified ES kernel. The parameters that result from a two-dimensional parameter search are hard-coded into the implementation. For explicitness, a selection of parameters is displayed in Appendix A.

As an example, Fig. 1 shows the map error function of the modified ES kernel in dependence on the oversampling factor σ and for fixed α. Increasing the oversampling factor allows a reduction of the convolution kernel support size while keeping the overall accuracy constant, which reduces the time required for the actual gridding or degridding step. At the same time, however, an increase in σ implies both a larger uv-grid and a higher number of w-planes. The former aspect leads to increased memory consumption of the algorithm, and both aspects increase the total cost of FFT operations. As a consequence, for a given number of visibilities, dirty image size, w range, and desired accuracy, it is possible to minimise the algorithm run-time by
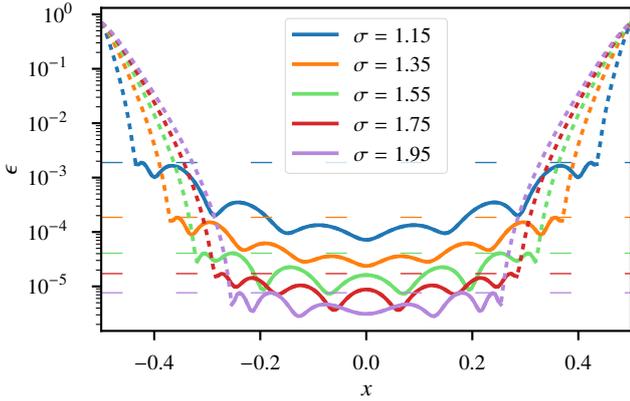
Fig. 1. Map error function for kernel support $\alpha = 6$ for a varying oversampling factor σ. The horizontal dotted lines display the advertised accuracy of the kernel.
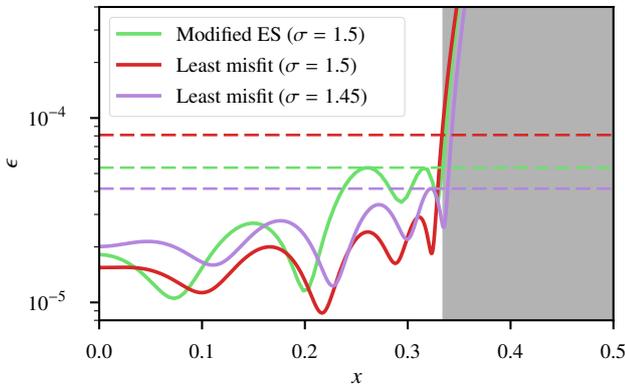

Fig. 2. Comparison of the map error function for least-misfit kernels with different oversampling factor and modified ES kernel. The kernel support size is $\alpha = 6$ for all three kernels. The dashed lines denote the supremum norm of the respective functions. We display only positive x (in contrast to Fig. 4). All map error functions are symmetric around $x = 0$.

finding the optimal trade-off between oversampling factor and kernel support size. The sweet spot for most applications lies in the range 1.2 to 2.0 for the oversampling factor. Our chosen functional form of the gridding kernel naturally leads to higher accuracy towards the phase centre, that is, $x = 0$.

For the comparison of our modified ES kernel and the least-misfit kernel, we note that the kernels are designed to minimise the supremum norm and the L2-norm, respectively, of the map error function. All least-misfit kernels in the following were computed using the software released alongside Ye et al. (2020). For given α and σ, the least-misfit kernel is therefore not necessarily optimal in our metric and vice versa, and comparison becomes non-trivial. Figure 2 displays the map error function for the modified ES kernel and the least-misfit kernel with the same α and σ and compares it to the least-misfit kernel with $\sigma = 1.45$. The steep increase in the map error function of the least-misfit kernel for $\sigma = 1.5$ significantly affects the supremum norm but still leads to a lower value for the L2-norm because the function is considerably smaller for small x. For the following comparison we select the least-misfit kernel for $\sigma = 1.45$ by hand. It is optimal under the L2-norm for $\sigma = 1.45$, but still performs better than the modified ES kernel even at $\sigma = 1.5$ under the supremum norm. It is to be assumed that with a more systematic search, even better least-misfit kernels can be found, so that the selected one should be regarded only in a qualitative sense.
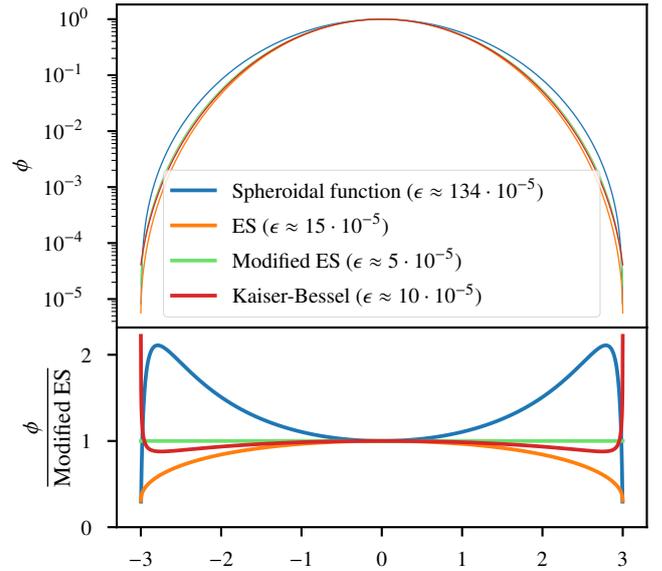

Fig. 3. Optimal kernel shapes for $\sigma = 1.5$ and $\alpha = 6$ with achieved accuracy ϵ.
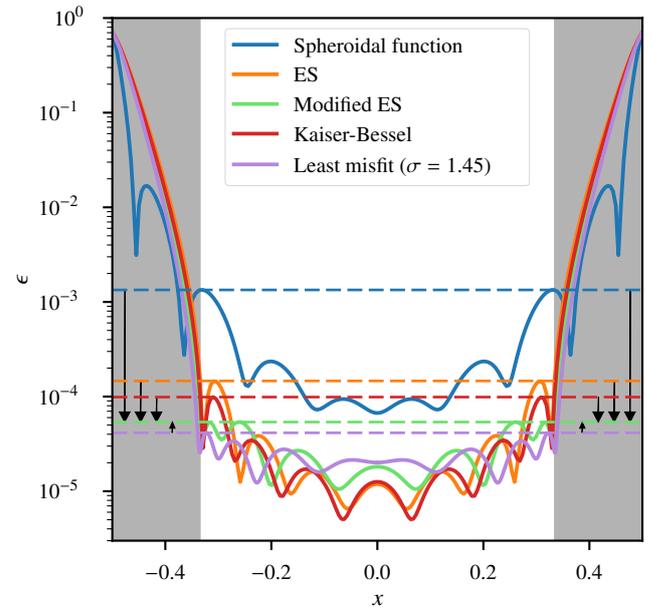

Fig. 4. Map error function of different kernel shapes for $\sigma = 1.5$ and $\alpha = 6$. A least-misfit kernel for a slightly lower oversampling factor is added for qualitative comparison (see the main text for a discussion of this choice), as well as the classic spheroidal function kernel. The arrows highlight the differences of the supremum norm of map error function of the different kernels with respect to our modified ES kernel.

Figures 3 and 4 display a comparison for given oversampling factor and kernel width of different gridding kernels. For all kernels (except for the least-misfit kernel) the same hyperparameter search for optimal parameters given α and σ was performed. The ES kernel (Barnett et al. 2019) is less accurate than the optimal Kaiser-Bessel kernel, while our modified ES kernel exceeds both other kernels in terms of accuracy. Figure 4 again shows that it is possible to find a kernel shape with this code that leads to more accurate transforms than our modified ES kernel. We also plot the spheroidal function kernel, which evidently performs much worse than the other kernels within the retained part of the image. The comparison with this particular error function

illustrates that the other kernels, which are chosen based on the knowledge that a part of the image will be trimmed, produce lower errors inside the final image in exchange for much higher errors in the trimmed regions.

Although the least-misfit kernel achieves a slightly more accurate gridding, we used the modified ES kernel for our implementation because only two real numbers are needed to specify the kernel for given α and σ in contrast to much larger tables for the least-misfit kernel. Additionally, it is non-trivial to minimise the supremum norm of Eq. (34) for a general least-misfit kernel. With only two parameters, a brute force parameter search is affordable, but this does not work for the many more degrees of freedom of the least-misfit kernels.

3.3. Kernel evaluation

In addition to choosing a kernel function that yields low errors, for the design of a practical algorithm it is also crucial to have a highly efficient way of evaluating this chosen function. Because for every visibility processed it is necessary to evaluate the kernel at least 3α times (α times each in u-, v-, and w-direction), this is definitely a computational hot spot, and therefore a single evaluation should not take more than a few CPU cycles.

From the candidate functions listed in Sect. 3.2, it is obvious that this rules out direct evaluation in most cases. The only exception here is the original ES kernel Eq. (28), which can be evaluated up to several hundred million times per second on a single CPU core using vector arithmetic instructions. To retain high flexibility with respect to the choice of kernel function, some other approach is therefore needed.

Traditionally, this problem is often addressed using tables of precomputed function values evaluated at equidistant points, from which the desired kernel values are then obtained by interpolation. Typically, zeroth-order (i.e. nearest-neighbour selection) and linear interpolation are used.

Interpolation at low polynomial degree soon leads to look-up tables that no longer fit into the CPU Level-1 and Level-2 caches when the required accuracy is increased, thus leading to high load on the memory subsystem, especially when running on multiple threads. To overcome this, we adopted an approach presented by Barnett et al. (2019) and approximated the kernel in a piece-wise fashion by several higher order polynomials. Barnett et al. (2019) reported that for a given desired accuracy ϵ, it is sufficient to represent a kernel with support α by a set of α polynomials of degree $\alpha + 3$. This means that a kernel evaluation can be carried out using only $\alpha + 3$ multiply-and-add instructions, and the total storage requirement for the polynomial coefficients is $\alpha(\alpha + 4)$ floating point numbers, which is negligible compared to the traditional look-up tables and much smaller than the CPU cache.

Because this approach is applicable to all kernel shapes discussed above, has sufficient accuracy (which can even be tuned by varying the degree of the polynomials), and has very low requirements on both CPU and memory, we used it in our implementation. Details on the construction of the approximating polynomials are given in Sect. 4.2.

4. Implementation

4.1. Design goals and high-level overview

In order to make our code useful (and easy to use) in the broadest possible range of situations, we aim for the library to have minimum external dependences (to simplify installation), have

a minimum simple interface and be easily callable from different programming languages (to allow convenient use as a plug-in for existing radio-astronomical codes), be well-suited for a broad range of problem sizes and required accuracies, have a very low memory footprint for internal data structures, and reach very high performance, but not at the cost of significant memory consumption. We decided to provide the functionality as a component of the ducc[3] collection of numerical algorithms. Because this package already provides support for multi-threading, SIMD data types, FFTs, and all other algorithmic prerequisites, the code does not have external dependences and only requires a compiler supporting the C++17 language standard. Its interface only consists of two functions (to apply the gridding operator and its adjoint), which take a moderate number of parameters (scalars and arrays). For illustration purposes, we list the interface documentation for the Python frontend of the library in Appendix B. Similar to many other gridder implementations, the interface allows specifying individual weights for each visibility, as well as a mask for flagging arbitrary subsets of the measurement set; both of these parameters are optional, however (see Appendix B). For an easy explicit understanding of the algorithm, we provide a compact Python and a slightly optimized Numpy and a Numba implementation of the w-gridding[4].

One important motivation for choosing C++ was its ability to separate the high-level algorithm structure from low-level potentially architecture-dependent implementation details. As an example, while the algorithm is written only once, it is instantiated twice for use with single-precision and double-precision data types. The single-precision version is faster, requires less memory, and may be sufficient for most applications, but the user may choose to use double precision in particularly demanding circumstances. Similarly, advanced templating techniques allow us to make transparent use of vector arithmetic instructions available on the target CPU, be it SSE2, AVX, AVX2, FMA3/4, or AVX512F; this is invaluable to keep the code readable and easy to maintain. The SIMD class of ducc supports the x86_64 instruction set, but could be extended to other instruction sets (such as ARM64) if needed.

Especially due to the necessity of having a low memory footprint, the w-planes are processed strictly sequentially. For the gridding direction (the degridding procedure is analogous), this means that for every w-plane, all relevant visibilities are gridded onto the uv-grid, weighted accordingly to their w-coordinate, the appropriate w-screen is applied to the grid, the grid is transformed into the image domain via FFT, and the resulting image is trimmed and added to the final output image. This approach is arguably suboptimal from a performance point of view because it requires re-computing the kernel weights in u- and v-direction for every visibility at each w-plane it contributes to: the number of kernel evaluations necessary to process a single visibility increases from 3α to $\alpha(2\alpha + 1)$. On the other hand, processing several planes simultaneously would increase the memory consumption considerably, and at the same time the speed-up would probably not be very significant because kernel computation only accounts for a minor fraction of the overall run time ($\lesssim 20\%$).

Overall, our approach requires the following auxiliary data objects: a two-dimensional complex-valued array for the uv-grid (requiring $2\sigma^2$ times the size of the dirty image), a temporary copy of the dirty image (only for degridding), and a data

---

[3] https://gitlab.mpcdf.mpg.de/mtr/ducc
[4] https://gitlab.mpcdf.mpg.de/mtr/ducc/-/blob/ducc0/python/demos/wgridder_python_implementations.py

structure describing the processing order of the visibilities (see Sect. 4.3 for a detailed description and a size estimate). Processing only a single w-plane at a time implies that for parallelisation the relevant visibilities need to be subdivided into groups that are gridded or degridded concurrently onto/from that plane by multiple threads. To obtain reasonable scaling with such an approach, it is crucial to process the visibilities in an order that is strongly coherent in u and v; in other words, visibilities falling into the same small patch in uv-space should be processed by the same thread and temporally close to each other. This approach optimises both cache re-use on every individual thread as well as (in the gridding direction) minimising concurrent memory writes. However, finding a close-to-optimal ordering for the visibilities in short time, as well as storing it efficiently, are nontrivial problems; they are discussed in Sect. 4.3.

As mentioned initially, parameters for interferometric imaging tasks can vary extremely strongly: the opening angle of the field of view can lie between arcseconds and π, visibility counts range from a few thousands to many billions, and image sizes start below 10^6 pixels and reach 10^9 pixels for current observations, with further increases in resolution to be expected. Depending on the balance between these three quantities, the optimal choice (in terms of CPU time) for the kernel support α, and depending on this the choice, of other kernel parameters and the oversampling factor σ, can vary considerably, and choosing these parameters badly can result in run times that are several times slower than necessary. To avoid this, our implementation picks near-optimal α and σ depending on a given task's parameters, based on an approximate cost model for the individual parts of the computation. For all available α ($\alpha \in \{4, \dots, 16\}$ in the current implementation), the code checks the list of available kernels for the one with the smallest σ that provides sufficient accuracy and predicts the total run-time for this kernel using the cost model. Then the kernel, α, and σ with the minimum predicted cost are chosen.

4.2. Gridding kernel

Our code represents the kernel function by approximating polynomials as presented in Sect. 3.3. A kernel with a support of α grid cells is subdivided into α equal-length parts, one for each cell, which are approximated individually by polynomials of degree $\alpha + 3$. When the kernel is computed in u- and v-directions, evaluation always takes place at locations spaced with a distance of exactly one grid cell, a perfect prerequisite for using vector arithmetic instructions. As an example, for $\alpha = 8$ and single precision, all eight necessary kernel values can be computed with only 11 FMA (floating-point multiply-and-add) machine instructions on any reasonably modern x86 CPU.

We used the family of modified ES kernels introduced in Sect. 3.2. They are convenient because an optimised kernel for given α and σ is fully characterised by only two numbers β and μ, and therefore it is simple and compact to store a comprehensive list of kernels for a wide parameter range of α, σ and ϵ directly within the code. This is important for the choice of near-optimal gridding parameters described in the preceding section.

When a kernel has been picked for a given task, it is converted to approximating polynomial coefficients. For maximum accuracy, this should be done using the Remez algorithm (Remez 1934), but we found that evaluating the kernel at the respective Chebyshew points (for an expansion of degree n, these are the roots of the degree $(n+1)$ Chebyshev polynomial, mapped from $[-1;1]$ to the interval in question) and using the interpolating polynomial through the resulting values produces sufficiently accurate results in practice while at the same time being much simpler to implement. Chebyshew abscissas are used because the resulting interpolants are much less prone to spurious oscillations than those obtained from equidistant abscissas[5] (Runge 1901).

Even better accuracy could be obtained by switching from modified ES kernels to least-misfit kernels, but there is a difficult obstacle to this approach: determining a least-misfit kernel for a given α and σ, which is optimal in the supremum-norm sense instead of the L2-norm sense, may be possible only by a brute-force search, which may be unaffordably expensive. Because the obtainable increase in accuracy is probably modest, we decided to postpone this improvement to a future improved release of the code.

4.3. Optimising memory access patterns

With the highly efficient kernel evaluation techniques described above, the pure computational aspect of gridding and degridding no longer dominates the run time of the algorithm. Instead, most of the time is spent reading from and writing to the two-dimensional uv-grid. Processing a single visibility requires α^3 read accesses to this grid, and for the gridding direction, the same number of additional write accesses. While it is not possible to reduce this absolute number without fundamentally changing the algorithm (which in turn will almost certainly lead to increasing complexity in other parts), much can be gained by processing the visibilities in a cache-friendly order, as was already pointed out in Sect. 4.1. Making the best possible use of the cache is also crucial for good scaling behaviour because every CPU core has its own L1 and L2 caches, whereas there is only a small number of memory buses (with limited bandwidth) for the entire compute node. For multi-threaded gridding operations, this optimisation is even more important because it decreases the rate of conflicts between different threads trying to update the same grid locations; without this measure, $R^\dagger$ would have extremely poor scaling behaviour.

Reordering the visibility and/or baseline data is not an option here because this would require either creating a rearranged copy of the visibilities (which consumes an unacceptable amount of memory) or, in the gridding direction, manipulating the input visibility array in-place (which is fairly poor interface design). Consequently, we rather used an indexing data structure describing the order in which the visibilities should be processed.

For this purpose, we subdivided the uv-grid into patches of 16×16 pixels, which allowed us to assign a tuple of tile indices (t_u, t_v) to every visibility. The patch dimension was chosen such that for all supported α and arithmetic data types, the 'hot' data set during gridding and degridding fit into a typical Level-1 data cache. In w-direction, the index of the first plane onto which the visibility needs to be gridded is called t_w. For compact storage, we used the fact that the uvw-locations of the individual frequency channels for a given row of the measurement set tend to be very close to each other. In other words, it is highly likely that two visibilities that belong to the same row and neighbouring channels are mapped to the same (t_u, t_v, t_w) tuple.

The resulting data structure is a vector containing all (t_u, t_v, t_w) tuples that contain visibilities. The vector is sorted lexicographically in order of ascending t_u, ascending t_v, and finally ascending t_w. Each of the vector entries contains another vector, whose entries are $(i_{\text{row}}, i_{\text{chan,begin}}, i_{\text{chan,end}})$ tuples, where i_{row} is the row index of the visibility in question, and $i_{\text{chan,begin}}$ and $i_{\text{chan,end}}$

---

[5] https://en.wikipedia.org/wiki/Runge%27s_phenomenon

represent the first and one-after-last channel in the range, respectively. Each of these vectors is sorted lexicographically in order of ascending i_{row} and ascending $i_{chan,begin}$.

While fairly nontrivial, this storage scheme is extremely compact: for a typical measurement set, it consumes roughly one bit per non-flagged visibility and is therefore much smaller than the visibility data themselves (which use eight bytes for every visibility, even the flagged ones). In the most unfavourable case (which occurs, e.g., when the measurement set only contains a single channel or when every other frequency channel is flagged), the memory consumption will be around eight bytes per non-flagged visibility.

Processing the visibility data in this new ordering leads to a more random access pattern to the visibility array itself. This is only a small problem, however, because entries for neighbouring channels are still accessed together in most cases, and also because the number of data accesses to the visibility array is lower by a factor of α^2 than the one to the uv-grid in our algorithm.

4.4. Parallelisation strategy

Our code supports shared memory parallelisation by standard C++ threads, that is, it can be run on any set of CPUs belonging to the same compute node. To achieve good scaling, all parts of the algorithm that contribute noticeably to the run time need to be parallelised. In our case these parts are: building the internal data structures, performing the (de)gridding process, applying w-screens, evaluating Fourier transforms, and evaluating and applying kernel corrections.

For the construction of the data structures (discussed in Sect. 4.3), we subdivided the measurement set into small ranges of rows that are processed by the available threads in a first-come-first-serve fashion. The threads concurrently update a global sorted data structure (using mutexes to guard against write conflicts), which is finally converted into the desired index list in a single-threaded code section. While considerable speedups can be achieved by this approach compared to a purely single-threaded computation, this part of the algorithm does not scale perfectly and can become a bottleneck at very high thread counts.

With the list of work items in hand, parallelising the actual gridding and degridding steps is straightforward: first, the list is subdivided into a set of roughly equal-sized chunks with $n_{chunks} \gg n_{threads}$. Each thread fetches the first chunk that has not been processed yet, performs the necessary operations, and then requests the next available chunk, until all chunks have been processed. This kind of dynamic work balancing is required here because it is difficult to determine a priori how much CPU time a given chunk will require.

The way in which the list was constructed ensures that each chunk is confined to a compact region of the uv-plane and therefore reduces potential write conflicts between threads during gridding. Still, it might happen that different threads try to update the same pixel in the uv-grid simultaneously, which would lead to undefined program behaviour. To avoid this, each thread in both gridding and degridding routines employs a small buffer containing a copy of the uw-region it is currently working on, and when the gridding routine needs to write this back to the global uv-grid, this operation is protected with a locking mechanism. In practice, the amount of time spent in this part of the code is very small, so that lock contention is not an issue.

Furthermore, the application of the w-screens and the kernel correction factors are parallelised by subdividing the array

in question into equal-sized slabs, which are simultaneously worked on by the threads. The FFT component has a built-in parallelisation scheme for multi-dimensional transforms that we make use of.

As mentioned above, the provided parallelisation can only be used on a single shared-memory compute node. A further degree of parallelism can be added easily, for example by distributing the measurement set data evenly over several compute nodes, performing the desired gridding operation independently on the partial data sets, and finally summing all resulting images. Analogously, for degridding, the image needs to be broadcast to all nodes first, and afterwards, each node performs degridding for its own part of the measurement set. How exactly this is done will most likely depend on the particular usage scenario, therefore we consider distributed memory parallelisation to be beyond the scope of our library. A distribution strategy over several compute nodes will increase the relative amount of time spent for computing the FFTs. Still, our implementation partially compensates for this effect by picking a combination of α, σ, and kernel shape that is optimised for the changed situation.

5. Accuracy tests

This section reports the accuracy tests that we have performed to validate our implementation. The tests can be subdivided into two major parts: the accuracy with respect to the direct evaluation of the adjoint of Eq. (3),

$$(R_0^{\dagger} d)_{lm} = \frac{1}{n_{lm}} \sum_k e^{2\pi i [u_k l + v_k m - w_k (n_{lm}-1)]} d_k, \quad l \in L, m \in M, \quad (35)$$

and the adjointness consistency between the forward and backward direction of the different calls.

5.1. Adjointness consistency

First, the degridding and the gridding calls were checked for their consistency. This is possible because mathematically, the two calls are the adjoint of each other. Therefore

$$\Re\left(\left\langle R(I), d \right\rangle_{(1)}\right) \overset{!}{=} \left\langle I, R^{\dagger}(d) \right\rangle_{(2)}, \quad \forall I \in \mathbb{R}^{n_l n_m}, \forall d \in \mathbb{C}^{n_k}, \quad (36)$$

where $\langle a, b \rangle_{(1)} := a^{\dagger} b$ and $\langle a, b \rangle_{(2)} := a^T b$ are the dot products of $\mathbb{C}^{n_k}$ and $\mathbb{R}^{n_l n_m}$, respectively. On the left-hand side of the equation, the real part needs to be taken because R maps from an $\mathbb{R}$- to a $\mathbb{C}$-vector space. Still, $\Im(R(I))$ is tested by Eq. (36) because evaluating the scalar product involves complex multiplications. Therefore the real part of the scalar product also depends on $\Im(R(I))$.

For the numerical test, we chose $n_l = n_m = 512$ and a field of view of $15° \times 15°$. The observation was performed at 1 GHz with one channel. The synthetic uvw-coverage consisted of 1000 points sampled from a uniform distribution in the interval $[-a, a]$, where $a = \text{pixsize}/2/\lambda$, pixsize is the length of one pixel and λ is the observing wave length. The real and the imaginary parts of the synthetic visibilities d were drawn from a uniform distribution in the interval $[-0.5, 0.5]$. Analogously, we drew the pixel values for the dirty image I from the same distribution. We consider this setup to be generic enough for accuracy testing purposes.

As discussed above, our implementation supports applying or ignoring the w-correction and can run in single or double precision. This gives four modes that are tested individually in the
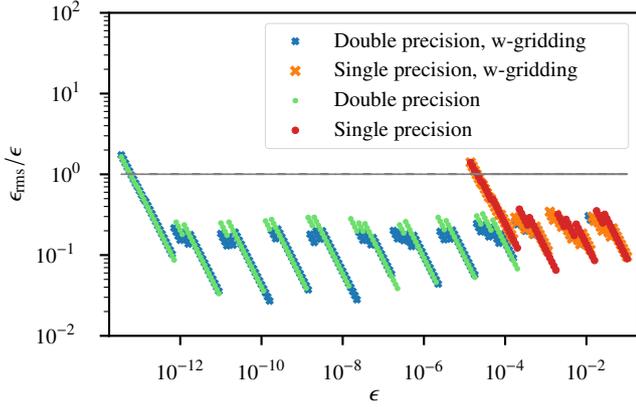
Fig. 5. Accuracy of $R^\dagger$. The ratio of measured root mean square error to the requested accuracy ϵ is plotted as a function of ϵ itself. The grey line denotes the identity function. Points lying in the region below the line represent configurations that are more accurate than specified by the user.

following. Moreover, the kernel sizes and the oversampling factor were chosen based on the intended accuracy ϵ, specified by the user. As a criterion for the quality of the adjointness, we use

$$\epsilon_{\text{adj}} := \frac{\left| \Re \left(\left\langle R(I), d \right\rangle_{(1)} \right) - \left\langle I, R^\dagger(d) \right\rangle_{(2)} \right|}{\min \left(\|d\| \cdot \|R(I)\|, \|I\| \cdot \|R^\dagger(d)\| \right)}. \tag{37}$$

For all four modes and for all tested ϵ in the supported range ($\geq 10^{-5}$ for single precision, $\geq 10^{-14}$ for double precision), this quantity lay below 10^{-7} and 10^{-15} for single and double precision, respectively.

5.2. Accuracy of $R^\dagger$

Second, we compared the output of our implementation to the output of the direct Fourier transform with and without w-correction. It suffices to test only $R^\dagger$ and not also R because the consistency of the two directions was already verified in Sect. 5.1. The error is quantified as rms error,

$$\epsilon_{\text{rms}}(d) = \sqrt{\frac{\sum_{lm} \left| (R_0^\dagger d)_{lm} - (R^\dagger d)_{lm} \right|^2}{\sum_{lm} \left| (R_0^\dagger d)_{lm} \right|^2}}. \tag{38}$$

As testing setup, the same configuration as above was employed. Figure 5 shows the results of the (approximate) gridding implementation against the exact DFT. It is apparent that single precision transforms reach the requested accuracy for $\epsilon \gtrsim 3 \times 10^{-5}$, while double precision transforms are reliably accurate down to $\epsilon \approx 10^{-13}$. We deliberately also show results for ϵ outside this safe range to demonstrate how the resulting errors grow beyond the specified limit due to the intrinsic inaccuracy of floating-point arithmetics. Inside the safe region, the achieved accuracy typically lies in the range between 0.03ϵ and ϵ, which indicates that the estimation in Eq. (33) is not overly pessimistic.

The saw-tooth pattern of the measured errors is caused by the dynamic parameter selection during the setup process of each gridding operation mentioned near the end of Sect. 4.1: Moving from higher to lower accuracies, a fixed combination of α, σ, and the corresponding kernel shape results in decreasing $\epsilon_{\text{rms}}/\epsilon$, which is indicated by the individual descending line segments. At some point, a new parameter combination (lower σ,
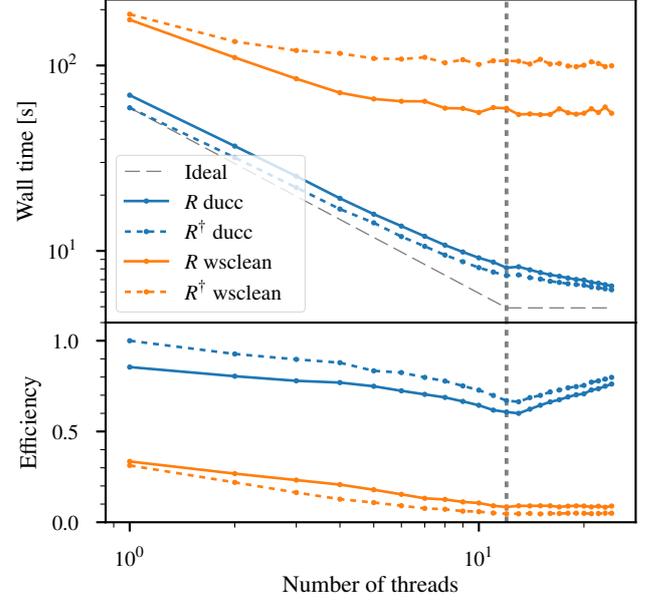

Fig. 6. Strong-scaling scenario. The vertical dotted gray line indicates the number of physical cores on the benchmark machine. Efficiency is the theoretical wall time with perfect scaling divided by the measured wall time and divided by the single-thread timing of '$R^\dagger$ ducc'.

or lower α with increased σ) with sufficiently high accuracy and lower predicted run time becomes available. This is then selected and the relative error jumps upwards, while still remaining well below the specified tolerance.

6. Performance tests

The tests in this section were performed on a 12-core AMD Ryzen 9 3900X CPU with 64GB main memory attached. `g++` 10.2 was used to compile the code, with notable optimisation flags including `-march=native`, `-ffast-math`, and `-O3`. The system supports two hyper-threads per physical CPU core, so that some of the tests were executed on up to 24 threads. As test data we used a MeerKAT (Jonas & MeerKAT Team 2016) L-band measurement set corresponding to an 11-hour synthesis with 8s integration time and 2048 frequency channels, using 61 antennas (824476 rows in total, project id 20180426-0018). We worked on the sum of XX and YY correlations only, ignoring polarisation, and after selecting only unflagged visibilities with non-vanishing weights, roughly 470 million visibilities need to be processed for each application of the gridding or degridding operator. The size of the dirty image was 4096×4096 pixels, and the specified field of view was $1.6° \times 1.6°$. Unless mentioned otherwise, computations were executed in single-precision arithmetic and with a requested accuracy of $\epsilon = 10^{-4}$. We compared the timings of our implementation to the standard radio software `wsclean` and the general-purpose library `FINUFFT`[6].

6.1. Strong scaling

First, we investigated the strong-scaling behaviour of our implementation. Figure 6 shows the timings of this problem evaluated with a varying number of threads. The ideal scaling would of course be $\propto n_{\text{threads}}^{-1}$, but this cannot be fully reached in practice.

---

[6] `https://github.com/flatironinstitute/finufft`, type 1, two-dimensional transform.

As mentioned in Sect. 4.4, the setup part of the algorithm does not scale perfectly, and the same is true for the FFT operations because of their complicated and not always cache-friendly memory access patterns.

Still, the implementation scales acceptably well, reaching a speed-up of roughly 8.0 when running on 12 threads. While the further improvements are much lower when scaling beyond the number of physical cores, as has to be expected, a total speed-up of around 9.6 is reached when using all hyper-threads available on the system.

In this test, degridding is slightly, but consistently slower than gridding, which appears counter-intuitive because degridding only requires roughly half the number of memory accesses. We assume that this is due to the horizontal addition of vector registers that has to be performed when a computed visibility value is written back to the measurement set. This kind of operation is notoriously slow on most CPUs, while the corresponding broadcast operation that is needed during gridding is much faster. If this interpretation is correct, it indicates that in the selected regime (single precision with an accuracy of 10^{-4}) memory accesses do not completely dominate computation. For higher accuracies this is no longer true, as shown in Sect. 6.3.

Figure 6 also shows analogous timings for the standard gridder in `wsclean`, but it is important to note that these cannot be directly compared to those of our code. While we tried to measure the timings with as little overhead as possible (we used the times reported by `wsclean` itself for the operations in question), the `wsclean` default gridder always interleaves I/O operations (which do not contribute at all to our own measurements) with the actual gridding and degridding, so there is always an unknown, non-scaling amount of overhead in these numbers. Additionally, the accuracy of `wsclean` cannot be set explicitly; based on experience, we expect it to be close to the target of 10^{-4} near the image center, but somewhat worse in the outer regions.

6.2. Comparison to non-equidistant FFT

As mentioned in the introduction, gridding or degridding without the w-term can be interpreted as a special case of the non-uniform FFT, where the uv coordinates of the individual points are not independent, but vary linearly with frequency in each channel. For this reason we also performed a direct comparison of our implementation with the `FINUFFT` library (Barnett et al. 2019). We still used the same measurement set as above, but performed a gridding step without the w term, using double precision and requiring $\epsilon = 10^{-10}$.

Because a general non-uniform FFT algorithm cannot be informed about the special structure of the uv coordinates, we supplied it with an explicit coordinate pair for every visibility. This implies that a much larger amount of data is passed to the implementation, and it also increases the cost of the preprocessing step. To allow a fairer comparison, we also ran `ducc` on an equivalent flattened data set, which only contained a single frequency channel and therefore as many uv coordinates as there are visibilities. We verified that both implementations returned results that are equal to within the requested tolerance. The performance results are shown in Fig. 7. In contrast to our implementation, `FINUFFT` features a separate planning phase that can be timed independently, so we show `FINUFFT` timings with and without the planning time, in addition to `ducc` timings for processing the original and flattened measurement set.

To a large extent, the results confirm the expectations. `FINUFFT` is always slower than `ducc` when `ducc` works on the un-flattened data. This can be attributed to the slightly higher
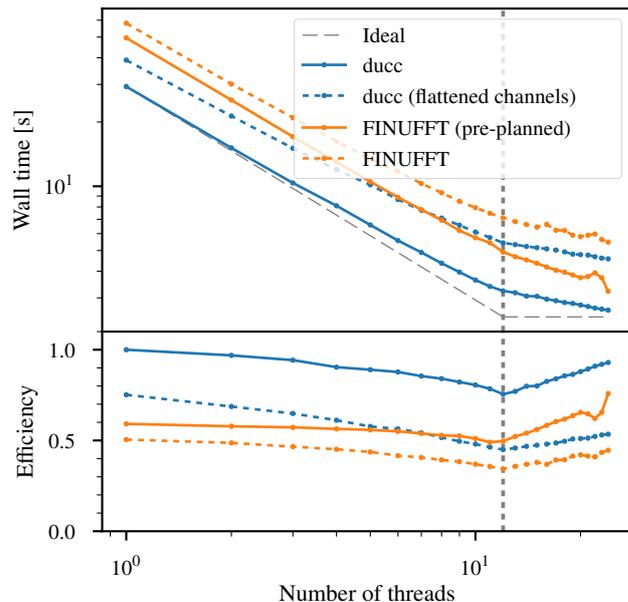

Fig. 7. Comparison to FINUFFT. The vertical dotted grey line indicates the number of physical cores on the benchmark machine. Efficiency is the theoretical wall time with perfect scaling divided by the measured wall time and divided by the single-thread timing of 'ducc'.

accuracy of the `ducc` kernels and/or to its advantage of knowing the internal structure of the uv data, which reduces setup time and the amount of memory accesses considerably. Furthermore, `ducc` performs rather poorly on the flattened data compared to its standard operation mode, especially with many threads. Here it becomes obvious that the index data structure, which has many benefits for multi-channel data, slows the code down when it is not used as intended by providing only a single channel. Finally, pre-planned `FINUFFT` performs worse than `ducc` with flattened data at low thread counts, but has a clear speed advantage on many threads; again, this is probably due to the `ducc` data structures, which are suboptimal for this scenario.

Memory consumption also behaves as expected, meaning that `ducc` without flattening requires the least amount of memory (because it does not need to store the redundant uv data), followed by both `FINUFFT` runs, while `ducc` with flattening consumes the most memory because it stores the full uv coordinates as well as a really large index data structure. Overall, we consider it very encouraging that despite differences in details, the performance and scaling behaviour of these two independent implementations are fairly similar to each other.

6.3. Run time vs. accuracy

For the following tests, we again used the setup described at the beginning of this section, but we fixed the number of threads to six and varied the requested accuracy ϵ as well as the data type of the input (single or double precision). Figure 8 shows the expected decrease in wall time for increasing ϵ, that is, lower accuracy. In single-precision mode the evaluation is indeed slightly faster than in double precision, most probably because more visibilities and grid points can be communicated per second between CPU and RAM for a given memory bandwidth. Moreover, the number of elements in the CPU vector registers is twice as large for single-precision variables.

In analogy to the observations in Sect. 6.1, degridding is slightly slower than gridding for these measurements. For double
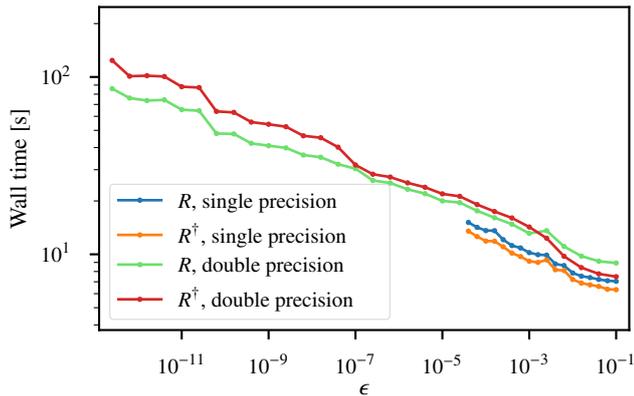
Fig. 8. Wall time vs. specified accuracy ϵ measured with six threads.

precision, the same is only true at very low accuracies; for $\epsilon \lessgtr 10^{-3}$, gridding becomes the more expensive operation, and this trend becomes very pronounced at the lowest reachable ϵ values. In these runs, the kernel support α is quite large and most of the run-time is presumably spent on data transfer from/to main memory. The results also show that while certainly attainable, high accuracy comes at a significant cost: going from a typical ϵ of 10^{-4} to 10^{-12} increases the run-time by about an order of magnitude.

7. Discussion

We have presented a new implementation of the radio interferometry gridding and degridding operators, which combines algorithmic improvements from different sources: an accurate and efficient treatment of the w-term for wide-field observations published by Ye (2019) and Ye et al. (2021), an easy-to-use, high-accuracy, functional form for the gridding kernels presented by Barnett et al. (2019), with some slight improvements, a piecewise polynomial approximation method for arbitrary kernels (also published by Barnett et al. 2019), which is very well suited for the task at hand), a parallelisation strategy, dynamic parameter selection, and indexing data structure of our own design. To the best of our knowledge, the resulting code compares favourably to other existing Fourier-domain gridders (both for wide- and narrow-field data) in terms of accuracy, memory consumption, single-core performance, and scalability. Our implementation is designed to have minimum dependences (only a C++17 compiler is needed), and it is free and open-source software. Therefore it may be advantageous to add it as an alternative option to existing radio interferometry imaging packages, as was already done in the `wsclean` code.

Compared with the fairly recent image-domain gridding approach (IDG, van der Tol et al. 2018), it appears that our implementation currently has a performance advantage when both algorithms are run on CPUs, but the GPU implementation of IDG easily outperforms all competitors on hardware of comparable cost. Furthermore, IDG can incorporate direction-dependent effects (DDEs) in a straightforward manner, which are difficult and costly to treat with Fourier-domain gridding algorithms.

However, it may be possible to address this within the w-gridding framework. The A-stacking algorithm (Young et al. 2015) might be combined with w-gridding, for instance. This would imply approximating all possible DDE patterns as linear combinations of a small set of N_b basis functions $f_b(l, m)$, computing (for every visibility) the projection of its particular DDE pattern onto this set of functions, running the w-gridder N_b times with the appropriate sets of weights, multiplying each result with the corresponding basis function, and finally adding everything together. Investigating the actual feasibility and performance of such an approach is left for future studies.

References

Arras, P., Frank, P., Haim, P., et al. 2020, ArXiv e-prints [arXiv:2002.05218]
Arras, P., Perley, R. A., Bester, H. L., et al. 2021, A&A, in press, https://doi.org/10.1051/0004-6361/202039258
Barnett, A. H., Magland, J., & af Klinteberg, L., 2019, SIAM. J. Sci. Comput., 41, C479
Cooley, J. W., & Tukey, J. W. 1965, An algorithm for the machine calculation of complex fourier series, 19, 297
Dutt, A., & Rokhlin, V. 1993, SIAM J. Sci. comput., 14, 1368
Jonas, J., & MeerKAT Team 2016, MeerKAT Science: On the Pathway to the SKA, 1
Offringa, A., McKinley, B., Hurley-Walker, N., et al. 2014, MNRAS, 444, 606
Perley, R. A. 1999, Synthesis imaging in radio astronomy II, Vol. 180, 383
Remez, E. Y. 1934, Compt. Rend. Acade. Sc., 199, 337
Richard Thompson, A., Moran, J. M., & Swenson, Jr., G. W. 2017, Interferometry and Synthesis in Radio Astronomy (Springer Nature)
Runge, C. 1901, Zeitschrift für Mathematik und Physik, 46, 224
Schwab, F. R. 1980, VLA Scientific Memoranda, 132
van der Tol, S., Veenboer, B., & Offringa, A. R. 2018, A&A, 616, A27
Ye, H. 2019, Accurate Image Reconstruction in Radio Interferometry, 139, https://www.repository.cam.ac.uk/handle/1810/292298
Ye, H., Gull, S. F., Tan, S. M., & Nikolic, B. 2020, MNRAS, 491, 1146
Ye, H., Gull, S. F., Tan, S. M., & Nikolic, B. 2021, arXiv e-prints [arXiv:2101.11172]
Young, A., Wijnholds, S. J., Carozzi, T. D., et al. 2015, A&A, 577, A56

Appendix A: Kernel parameters

Optimal kernel parameters and associated accuracy ϵ for the modified exponential semicircle kernel Eq. (29) given the oversampling factor σ and the kernel support size α. Larger σ and larger α lead to smaller ϵ. Larger σ and smaller α increase the fraction of the FFT of the total computation time. FFT and gridding costs are represented in our implementation with a simple cost model such that the algorithm can choose optimal α and σ automatically. For brevity, we display only the tables for $\alpha \in \{4, 7, 8, 12, 16\}$. The rest can be looked up in the ducc code repository. The least-misfit kernels (Ye et al. 2020) achieve an accuracy $\epsilon = 10^{-7}$ for $\alpha = 7$ and $\sigma = 2$.

Table A.1. Optimal parameters for $\alpha = 4$.

σ	ϵ	β	μ
1.15	0.025654879	1.3873426689	0.5436851297
1.2	0.013809249	1.3008419165	0.5902137484
1.25	0.0085840685	1.3274088935	0.5953499486
1.3	0.0057322498	1.3617063353	0.5965631622
1.35	0.0042494419	1.384549988	0.5990241291
1.4	0.0033459552	1.4405325088	0.5924776015
1.45	0.0028187359	1.4635220066	0.5929442711
1.5	0.0023843943	1.5539689162	0.5772217314
1.55	0.0020343796	1.5991008653	0.5721765215
1.6	0.0017143851	1.6581546365	0.5644747137
1.65	0.0014730848	1.7135331415	0.5572788589
1.7	0.0012554492	1.7464330378	0.5548742415
1.75	0.0010610904	1.7887326906	0.5509877716
1.8	0.00090885567	1.8122309426	0.5502273972
1.85	0.0007757401	1.8304451327	0.550396716
1.9	0.0006740398	1.8484487383	0.5502376937
1.95	0.00058655391	1.8742215688	0.5489738941
2.0	0.00051911189	1.90694363	0.5468009434

Table A.2. Optimal parameters for $\alpha = 7$.

σ	ϵ	β	μ
1.15	0.00078476028	1.5248706519	0.5288306317
1.2	0.00027127166	1.5739348793	0.5287992619
1.25	0.00012594628	1.6245240723	0.527921777
1.3	7.0214545e−05	1.6835745981	0.5257484101
1.35	4.1972457e−05	1.7343424414	0.5239793844
1.4	2.378019e−05	1.7845017738	0.5224266045
1.45	1.3863408e−05	1.8180597789	0.5221834768
1.5	9.1605353e−06	1.868082272	0.5206277502
1.55	6.479159e−06	1.9188980015	0.5183134674
1.6	4.6544571e−06	1.9536166143	0.5178695891
1.65	3.5489761e−06	1.9786267068	0.5178430252
1.7	2.7030348e−06	2.0027666534	0.5178577604
1.75	2.0533894e−06	2.0289949199	0.5176300336
1.8	1.6069122e−06	2.0596412946	0.5167551932
1.85	1.2936794e−06	2.0720606842	0.5178747891
1.9	1.0768664e−06	2.090898174	0.5181009847
1.95	9.0890421e−07	2.1086185697	0.5184537843
2.0	7.7488775e−07	2.1278284187	0.5186377792

Table A.3. Optimal parameters for $\alpha = 8$.

σ	ϵ	β	μ
1.15	0.00026818611	1.568124649	0.5223052481
1.2	7.8028732e−05	1.620926145	0.5219287175
1.25	2.7460918e−05	1.6851585171	0.519925059
1.3	1.3421658e−05	1.7442373315	0.5182155619
1.35	7.5158217e−06	1.7876782642	0.5176319503
1.4	4.2472384e−06	1.8294321912	0.5171860211
1.45	2.5794802e−06	1.871691821	0.5161733611
1.5	1.6131994e−06	1.9213040541	0.5145350888
1.55	1.0974814e−06	1.9637229131	0.5134005827
1.6	7.531955e−07	2.0002761373	0.5128849282
1.65	5.5097346e−07	2.0275645736	0.5127082324
1.7	4.0136726e−07	2.0498410409	0.5130237662
1.75	2.906467e−07	2.073158517	0.5131757153
1.8	2.1834922e−07	2.0907418726	0.5136046561
1.85	1.6329905e−07	2.1164552354	0.5133333878
1.9	1.2828598e−07	2.126157016	0.5143004427
1.95	1.0171134e−07	2.1363206613	0.515235491
2.0	8.1881369e−08	2.1397013368	0.5166895497

Table A.4. Optimal parameters for $\alpha = 12$.

σ	ϵ	β	μ
1.15	2.7535895e−06	1.6661837519	0.5098172147
1.2	5.2570038e−07	1.7294557459	0.5089239596
1.25	1.378658e−07	1.7698182384	0.5099240718
1.3	4.4329167e−08	1.8092042442	0.510607427
1.35	1.7038991e−08	1.8619112597	0.5093832337
1.4	6.5438748e−09	1.9069147481	0.5089479889
1.45	2.9874764e−09	1.9318398074	0.5098082325
1.5	1.4920459e−09	1.9628483155	0.5100985753
1.55	8.0989276e−10	2.0129847811	0.5085327805
1.6	4.1660575e−10	2.0517921747	0.5079102398
1.65	2.3539727e−10	2.06983884	0.5085131064
1.7	1.3497289e−10	2.0887365361	0.5090417146
1.75	8.3256938e−11	2.106955733	0.5095920671
1.8	5.8834619e−11	2.1359415217	0.5091887069
1.9	2.6412908e−11	2.2006369514	0.5075889699
1.95	1.7189689e−11	2.2146741638	0.5080017404
2.0	1.2174796e−11	2.2431392199	0.5075191177

Table A.5. Optimal parameters for $\alpha = 16$.

σ	ϵ	β	μ
1.3	1.1509596e−10	1.7892839755	0.5122877693
1.35	3.2440049e−11	1.8914441282	0.5063521839
1.4	8.4329616e−12	1.9296369098	0.5065170208
1.45	3.1161739e−12	1.9674735425	0.5063244338
1.5	1.2100308e−12	2.0130787701	0.5055587965
1.55	4.6082202e−13	2.0438032614	0.5056309683
1.6	1.7883238e−13	2.0329561822	0.5089045671
1.65	9.2853815e−14	2.0494514743	0.5103582604
1.7	5.6614567e−14	2.0925119791	0.5083767402
1.75	2.875391e−14	2.1461524027	0.5062037834
1.8	1.6578982e−14	2.1490040175	0.508272183
1.85	1.1782751e−14	2.1811826814	0.5072570059
1.9	8.9196865e−15	2.1981176583	0.5075840871
1.95	6.6530006e−15	2.234001135	0.5060133105
2.0	5.0563492e−15	2.2621631913	0.5056924675

Appendix B: Python interface documentation

```
def ms2dirty(uvw, freq, ms, wgr, npix_x, npix_y,
             pixsize_x, pixsize_y, nu, nv, epsilon,
             do_wstacking, nthreads, verbosity, mask):
""" Converts an MS object to dirty image.

Parameters
----------
uvw: numpy.ndarray((nrows, 3), dtype=numpy.float64)
    UVW coordinates from the measurement set
freq: numpy.ndarray((nchan,), dtype=numpy.float64)
    channel frequencies
ms: numpy.ndarray((nrows, nchan),
    dtype=numpy.complex64 or numpy.complex128)
    the input measurement set data.
    Its data type determines the precision in which
    the calculation is carried out.
wgt: numpy.ndarray((nrows, nchan), float with same
    precision as 'ms'), optional
    If present, its values are multiplied to the
    input before gridding.
npix_x, npix_y: int
    dimensions of the dirty image (must both be even
    and at least 32)
pixsize_x, pixsize_y: float
    angular pixel size (in radians) of the dirty image
nu, nv: int
    obsolete, ignored
epsilon: float
    accuracy at which the computation should be done.
    Must be larger than 2e-13. If 'ms' has type
    numpy.complex64, it must be larger than 1e-5.
do_wstacking: bool
    if True, the full w-gridding algorithm is carried
    out, otherwise the w values are assumed to be zero
nthreads: int
    number of threads to use for the calculation
verbosity: int
    0: no output
    1: some output
    2: detailed output
mask: numpy.ndarray((nrows, nchan),
                    dtype=numpy.uint8),
    optional
    If present, only visibilities are processed
    for which mask!=0

Returns
-------
numpy.ndarray((npix_x, npix_y), dtype=float of same
    precision as 'ms')
    the dirty image

Notes
-----
The input arrays should be contiguous and in C memory
order. Other strides will work, but can degrade
performance significantly.
"""
```

```
def dirty2ms(uvw, freq, dirty, wgr, pixsize_x,
             pixsize_y, nu, nv, epsilon, do_wstacking,
             nthreads, verbosity, mask):
""" Converts a dirty image to an MS object.

Parameters
----------
uvw: numpy.ndarray((nrows, 3), dtype=numpy.float64)
    UVW coordinates from the measurement set
freq: numpy.ndarray((nchan,), dtype=numpy.float64)
    channel frequencies
dirty: numpy.ndarray((npix_x, npix_y),
    dtype=numpy.float32 or numpy.float64)
    dirty image
    Its data type determines the precision in which
    the calculation is carried out.
    Both dimensions must be even and at least 32.
wgt: numpy.ndarray((nrows, nchan), same dtype as
    'dirty'), optional
    If present, its values are multiplied to the
    output.
pixsize_x, pixsize_y: float
    angular pixel size (in radians) of the dirty image
nu, nv: int
    obsolete, ignored
epsilon: float
    accuracy at which the computation should be done.
    Must be larger than 2e-13.
    If 'dirty' has type numpy.float32, it must be
    larger than 1e-5.
do_wstacking: bool
    if True, the full w-gridding algorithm is carried
    out, otherwise the w values are assumed to be zero
nthreads: int
    number of threads to use for the calculation
verbosity: int
    0: no output
    1: some output
    2: detailed output
mask: numpy.ndarray((nrows, nchan),
                    dtype=numpy.uint8),
    optional
    If present, only visibilities are processed
    for which mask!=0

Returns
-------
numpy.ndarray((nrows, nchan), dtype=complex of same
    precision as 'dirty')
    the measurement set data.

Notes
-----
The input arrays should be contiguous and in C memory
order. Other strides will work, but can degrade
performance significantly.
"""
```