WSClean

Release git

André Offringa

Oct 21, 2022

CONTENTS

This is the official WSClean manual. The latest version of WSClean can be found at https://gitlab.com/aroffringa/wsclean/.

If you use WSClean for scientific work, please cite *the appropriate paper(s)*.

Contents:

CHAPTER

ONE

GETTING STARTED

1.1 Introduction

WSClean stands for "w-stacking clean". W-stacking is an alternative to the w-projection algorithm, in which the uv-samples are not convolved with a w-term correcting kernel before the FFT, but are instead corrected with a multiplication after the FFT. Multiplying each pixel is generally much faster than convolving an image, but the downside is that samples can not be added together on one grid until the w-term has been corrected. This means that in w-stacking, each w-layer needs to be stored and FFT-ed individually. The result is that w-stacking requires more memory and spends relative more time doing the FFTs, instead of gridding+convolving.

It turns out that for many telescopes, the WSClean approach is a very good trade-off, and WSClean is typically an order of magnitude faster than Casa's w-projection on MWA data. It can also handle full-sky 10k x 10k images on which Casa runs out of memory. WSClean with the same number of layers is at least as accurate as w-projection. In tests it seems even to be slightly more accurate, probably because there is no need to trim down a convolution kernel. See the *further information page* for more references.

Next chapter: *General usage*

1.2 Change logs

1.2.1 Version 3 releases

WSClean version 3 is the current major release number. It is in development since 2021. Stable releases:

WSClean version 3.2

Released 2022-10-21

New/changed dependencies:

- To make use of EveryBeam, WSClean 3.2 requires EveryBeam version 0.4.0.
- The requirements on IDG have not been changed; WSClean 3.2 requires IDG version 1.0 or newer.

Summary:

Major & visible changes

- Add support for direction dependent PSFs in deconvolution (parameter `-dd-psf-grid`).
- Support heterogenously polarized measurement sets (#130).
- Make use of the low-memory interface of EveryBeam. This reduces the amount of memory required for large image runs and additionally reduces the disk IO substantially.
- Include an experimental, potentially faster tuned wgridder implementation.
- Automatically determine right amount of memory and threads to use in parallel runs.
- The various `-use-...-gridder` are replaced by a single `-gridder <name>` option.
- Support spectral fitting when polarizations are joined.
- Better automatic determination of number of cores to be used.
- Fix point-scale auto-masking on data with missing channels (Radler #114).
- Calculate and save the MFS beam when relevant.
- In facetted imaging, replace NaN solutions by zeros.

Bug fixes

- Fix a crash when using local rms + multiscale.
- Fix a flux-scale issue when applying the primary beam on Stokes I imaging runs (see !465 and !441).
- Solve crash when compiling with `_GLIBCXX_ASSERTIONS`.
- Fix compilation on Macs.
- Allow custom HDF5 directory.
- Fix multi-polarization imaging without joining pols (#128).
- Allow facets to be defined outside the FOV, and skip these during imaging.
- Fix missing FITS file error when faceting without applying beam.
- Improve some error messages.
- Make sure exceptions are reported while gridding in parallel.
- Fix FITS `LONPOLE` issue causing incorrect image coordinates at the pole.
- Output correct source lists when using `-shift` (#98).
- Fixes for GCC-12.

Other significant code changes

- Format python files using black.
- Documentation improvements.
- Separate deconvolution code into separate library: Radler

WSClean version 3.1

Released 2022-04-01

New/changed dependencies:

- To make use of IDG, WSClean 3.1 requires IDG version 1.0 or newer.
- To make use of EveryBeam, WSClean 3.1 requires EveryBeam version 0.3.1.

Summary: *Facet-based imaging* is significantly improved, and now supports applying per-facet solutions (from H5Parm files) and beam corrections (from EveryBeam). Stokes-I-only imaging with IDG imaging is now significantly faster. WSClean 3.1 also adds experimental support for GMRT primary beam correction.

Major & visible changes

- Support time-direction *baseline-dependently averaged data from DP3*.
- Reduce disk-access during *IDG* Stokes I imaging by not storing XY/YX to disk, and enable the faster IDG Stokes I mode when possible.
- Automatically use the primary beam corrected image (`...-pb.fits`) file when predicting with beam correction.
- Cache average beam to disk in IDG mode to reduce memory usage.
- Add support for the GMRT beam (scientific validation still in progress).
- Add a beam-normalisation-mode option to control EveryBeam's normalisation mode.
- Support multiple h5 files in facetting mode.
- Introduce `-pb-grid-size` and deprecate `-pb-undersampling`.

Bug fixes

- Report error when user asks for incorrect channel selection.
- Fix certain selection parameters that no longer worked in reorder mode (e.g. even/odd timestep selection).
- Correctly normalize reported flux levels for squared channels.
- Fix a crash with the `-reuse-dirty` option in certain modes.
- Fix segmentation fault when using `-reuse-psf` with IDG.
- Fix masked parallel deconvolution.
- Fix beam + H5 solution application in facetting mode.
- Fix possible `-facet-regions` mismatch with H5Parms.
- Support differential lofar beam in facet-based imaging.
- Support scalar gain corrections in facet-based imaging.
- Do not apply beam or solutions to PSF in facet-based imaging.
- Fix unset threadcount value in imageweights class.
- Fix IUWT. Because of code clean-ups in Nov 2020, the IUWT deconvolution option was no longer producing correct results.
- Fix channel selection issue for non-uniform multi-SPW sets (#79).

- Fix channel selection issue for multi-SPW set with no-mf-weighting (#105).
- Fix rare crash caused by uninitialized array.
- Fix compilation on hurd-i386.

Other significant code changes

- WSClean now requires a compiler that supports C++17.
- Add support for gcc versions 11 and 12.
- Lots of refactoring.
- Preparations to split off deconvolution code as a separate reusable module.
- Several documentation improvements / updates.
- Better support and error reporting for missing git submodules.
- Many new unit and system tests.

WSClean version 3.0

Released 2021-08-26

New/changed dependencies:

- To make use of IDG, WSClean 3.0 requires IDG version 0.8 or newer.
- To make use of EveryBeam, WSClean 3.0 requires EveryBeam 0.2. Using EveryBeam 0.3 (or newer) results in compilation errors.

Summary: This is a major release containing many new features and bug fixes. The biggest new feature in this release is facetting, which runs the gridder for smaller parts of the image, stitches those parts and runs deconvolution on the resulting image. Not all facetting functionality is finished as of yet (see the *facetting documentation*). Upcoming releases will add more functionality to facetted imaging. Two other significant features are deconvolution with forced spectra and the ability to write deconvolution scripts in Python.

Major & visible changes

- *Facetted imaging*.
- Use EveryBeam for calculation of *all* instrumental primary beams. **This implies that EveryBeam is now required to apply beams from instruments like LOFAR, MWA, JVLA, ATCA, etc.**
- Make it possible to call Python deconvolution implementations (parameter `-python-deconvolution`).
- New feature to force spectra from an external image (parameter `-force-spectrum`).
- New option `-deconvolution-threads` to limit the number of threads (and associated memory) used in deconvolution.
- New option `-wgridding-accuracy` to tweak the accuracy of the wgridder.
- New option `-primary-beam-limit` to set the level at which to trim the beam.
- New option `-simulate-baseline-noise` to perform simulations with baseline-specific noise values.
- Support for SKA beams (through EveryBeam).

- Phase *shifting* (not rotating) is now done inside `wsclean` and does not require (or allow!) shifting with `chgcentre` (see *chgcentre documentation* for the changes).
- Allow distributed multi-polarization imaging with IDG (also fixes #56).
- Major speed improvements in deconvolution and various other places.
- `chgcentre` is now part of the wsclean package.

Bug fixes

- Various fixes to deconvolution channel weighting; improves stability of deconvolution when channels are missing (includes fixing #71).
- Fix openmpi distributed imaging of very large images (>2^31 bytes per image) (#13).
- Fix PSF problem when using tapers, as noted by P. Serra (#134).
- Fix horizon treatment in the W-gridder.
- Avoid an extra major iteration when using auto-masking and `niter` is reached.
- Make it possible to read measurement sets with a varying number of channels (#18).
- Fix writing of imaging weight spectrum column, reported by L. Bester.
- Improve allocation of IDG buffers, improves performance and memory usage on very large memory machines (#2).
- Make sure `wsclean-mp` is installed during `make install`.
- Avoid use of python2's `print a` for MWA beam finding (#23).
- Allow combining the w-gridder with `-parallel-gridding`.
- Fix hang during beam generation (#32).
- Improve console output in various ways (includes #60).
- Fix bug that causes aterms not to be correctly applied in `wsclean-mp`.
- Fix hanging `wsclean-mp` when only one node is available.
- Solve crash when using large RMS windows.
- Fix setting of MPI compile flags/options.
- When given an odd image size, increase it to an even size.
- Avoid uninitialized value when first channel has zero weight, fixing a possible image corruption.
- Erase reordered MS data BEFORE reordering next timestep. Not after. Reported by O. Smirnov (#55).
- Fix predict timer (#9).
- Solve crash in empty predict when using the wgridder (#66).
- Fix `-reuse-primary-beam`.

Other significant code changes

- Use of git submodules for common code (aocommon, pybind11, schaapcommon).
- Add Gitlab continuous integration script & large increase of automated tests.
- Format and check code with clang-format.
- Lots of restructuring and code cleaning.
- Update of wgridder to the most recent version.
- Remove use of boost thread.
- Let deconvolution use single precision (#29).
- Improve dijkstra's algorithm to improve deconvolution performance.
- Make channel interpolation use multiple threads to improve deconvolution performance.
- Add several dockerfiles to the repository to build containers.
- Use `-fvisibility=hidden` to disable pybind11 compilation warnings.
- Parellellize `RMSImage::SlidingMinimum` and calculating the image weights to improve start-up and deconvolution performance.
- Reimplement and parallelize FFTConvolver, improving speed of various tasks.
- As always, various documentation improvements.
- *Changelog for WSClean 3.2* (2022-10-21)
- *Changelog for WSClean 3.1* (2022-04-01)
- *Changelog for WSClean 3.0* (2021-08-26)

1.2.2 Version 2 releases

WSClean version 2 was in development between 2016 and 2021. Stable releases:

WSClean version 2.10

Released 2020-07-15

WSClean 2.10 depends on IDG version 0.7 or newer.

Summary of changes: WSClean can now run on multiple nodes using MPI. Several new methods for calculating a-terms were added: position-delta screens, better kernel windowing, gridding with the VLA beam and joint deconvolution of multibeam/PAF systems (which was demonstrated using Apertif).

This is the last version that is released via SourceForge. Future development will move to GitLab: https://gitlab.com/aroffringa/wsclean.

Major & visible changes:

- Support for distributed computing using MPI
- Support for "position shift screens" (dldm-screens) during a-correction gridding
- Support for joint deconvolution of PAF / multi-beam observations
- More windowing options for a-correction kernels, defaults now to "raisen Hann"
- The VLA beam has been implemented

- Support imaging of multiple fields
- Add parameter `-multiscale-max-scales`
- Restore source component models using `-restore-list`

Bug fixes & other changes:

- Properly weigh channels when averaging with user-set deconvolution channels.
- WSClean could iterate infinitely when first multiscale scale is not 0
- GCC 10 fixes
- Improved error reporting
- Fix inversion timing

WSClean version 2.9

Released 2020-03-27

WSClean 2.9 is released together with IDG version 0.7.

Summary of changes: WSClean+IDG is now much faster, in particular with fast-changing aterms. The combination of WSClean 2.9 + IDG 0.7 supports the efficient use of GPUs to grid large data volumes with complex a-terms. Furthermore, a new gridder that is more accurate compared to the default w-stacking gridder, and for some cases also faster! (in particular large images, many visibilities). The plan is also to move WSClean to Github in the next version of thereafter.

Major & visible changes:

- Add new gridder "*wgridder*", a stand-alone gridder written by Martin Reinecke that is now integrated in WSClean.
- Applying a second aterm to all antennas is possible by setting dimension of antenna axis to one in aterm fits file.
- Make source list output possible in IDG.
- The FFTs are now performed with floats instead of doubles, thereby descreasing memory usage by 2 and increasing the speed anywhere from 20-100%.
- LOFAR image-based beam correction (i.e. no aterm correction) now uses full Mueller matrix calculations instead of Jones, which removes some biases in beam correction that were seen.
- New options `-reuse-psf` and `reuse-dirty` to skip these steps when they already exist.
- WSClean makes use of IDG's API changes to allocate optimal buffer sizes, greatly increasing the speed of IDG, and solving allocation problems on low-mem platforms
- When parallel deconvolution is used, the logging is now much improved: only one thread reports its full progress.
- Correct smallest multi-scale scale when using Gaussian tapering. Multi-scale cleaning was using just the longest baseline for determining the smallest scale to clean with. This causes tapered imaging to almost never use the smallest scale.
- Add a 'dry run' mode, requested by J. Morgan. Fixes #162.

Bug fixes & other changes:

- Stokes V imaging with R/L pols results in zero image (reported by Chiara Stuardi)
- Don't allow direct MS access for multi spw sets, to prevent a deadlock.
- Output error when trying to combine parallelgridding with IDG.

- Don't incorrectly apply the average beam when using idg + apply-primary-beam.
- Support newer casacore versions.
- Remove dependency on boost thread.
- Image the gridding kernel as option was removed, as both IDG and WGridder don't support it.
- Fix beam calculation for 1 timestep mses
- Fix a crash when using pb correction + deconvolution channels
- Solve reported problem with HDF5/Intel compilation

WSClean version 2.8

Released 2019-09-16

WSClean 2.8 is released together with IDG version 0.6. Together, these two new versions bring a ton of bug fixes.

Summary of changes: The combination of WSClean 2.8 + IDG 0.6 allows for the first time all major imaging use-cases, including a-term corrections (screens of diagonal Jones matrices, TEC values or beam corrections), fast GPU and CPU gridding of really large images together with all deconvolution methods. In WSClean, much work was done to allow parallel deconvolution in all major modes, thereby speeding up the deconvolution by an order of magnitude. Fixing (SF #116) improved multi-scale convergence & stability.

Major & visible changes:

- Use a better multiscale convolution padding by default (SF #116).
- Implemented `-local-rms` in combination with parallel cleaning
- Implemented source list output in combination with parallel cleaning
- Add a `-horizon-mask parameter` to automatically mask out emission beyond the horizon
- Option `-direct-allocation` added for experimenting with skipping the image buffer allocator
- Allow a-terms to have a frequency axis

Bug fixes & other changes:

- Multi-channel deconvolution bug causing multi-channel imaging to use the automask of channel 0. Reported by P. Serra.
- Fixed issued when using the combination of multiscale + auto mask + paparallel deconvolution (many identified by F. De Gasperin).
- Fixed issue causing parallel deconvolution to stay in infinite loop when niter low.
- Fix predict crash when image size not specified
- Fix non-parallel cleaning in two cases (local rms and saving source lists wouldn't work properly anymore).
- Fixing bug when using indivisable nr of deconvolution channels
- Using `-channel-range` caused wrong behaviour or exception.
- Remove requirement on boost threads.
- Solve bug in reordering of multi-interval imaging, causing a tempfiles not found error.
- Improved some error messages and spelling.
- Solve crash in LOFAR beam creation
- Fix hang when creating a primary beam

WSClean version 2.7

Released 2019-04-18

WSClean 2.7 can be compiled together with IDG version 0.5.

Summary of changes: This is a major release with many new functions. Imaging with IDG has matured to a fully functional form, providing speed improvements and new features: It is now possible to grid with beams from several arrays, as well as with a-term screens and gain solutions by using IDG. This has shown excellent results to correct for DD effects. The MWA and ATCA beams have been implemented, gridding different channels can be parallelized which majorly speeds up multi-frequency imaging, a direct FT option was implemented, the default accuracy has been increased while at the same time imaging has become faster, and finally many small isues were solved.

Major & visible changes:

- WSClean now implements the AARTFAAC, the MWA and the ATCA (ATCA_16 band) beam.
- Imaging tasks can be run in parallel, which greatly speeds up multi-frequency imaging in normal *w*-stacking mode.
- Reordering can also be in parallel now, but remains experimental as casacore sometimes causes race bug.
- **Several ways to apply direction dependent effect during the gridding with IDG:**
  - LOFAR & MWA beam gridding
  - TEC screens
  - "Generic" solution gain screens
  - Any combination of the above.
- Calculate average IDG beam and output pb corrected images.
- Option to set IDG beam update rate (`-beam-aterm-update`).
- Option to set a-term kernel size (`-aterm-kernel-size`).
- Allow saving of a-term screens (`-save-aterms`).
- IDG settings can now be specified in a separate config file.
- Add countdown system to multiscale algorithm to prevent iterating too many times around the stopping criterium.
- The first residual image is no longer saved by default; new option `-save-first-residual` can enable old behaviour.
- Determine correct beam automatically from LOFAR beam keywords stored by DP3 in data column.
- Implemented option `-use-weights-as-taper` (after communication with V. Mahatma).
- Option `-direct-ft` implements a fully accurate direct FT.
- Several new gridding kernels (implemented for EoR gridding accuracy paper).
- New option to set location of MWA h5 file manually (`-mwa-path`).
- Add option `-nwlayers-factor` to make it easier to increase *w*-term accuracy.
- Increased default oversampling factor to 1023.

Bugfixes / technical changes:

- When using temporary directory, an ending slash in the ms name would lead to incorrect temporary filenames, causing multi-ms imaging to fail.
- Always resample tec/gain screens onto the correct a-term kernel resolution (reported by T. Hodgson, see IDG ticket 12).

- Output error when frequency axis is not correct (reported by F. De Gasperin).
- Fix by M. Mevius to be able to predict LOFAR beam at NCP.
- Store model files with Jy/px units.
- Fix several compilation errors, including when using newer gccs and Casacore versions
- LOFAR centroid gridding was removed.
- Increase size of bounding box for Gaussians in multiscale.
- Make sure that reversed channels work when using small inversion optimization (ATCA sets seem to (sometimes?) have the frequency channels in reverse (low index = high frequency)).
- Fix prediction issue when manually setting nwlayers.
- Clear cache between imaging of different intervals (SF #146), reported by J. Morgan).
- Don't restore NaN components to avoid corrupting pb mf image (Reported by C. Riseley).
- Foundations for multi-node distribution were implemented.

WSClean version 2.6

Released 2018-06-11

Summary: A "lots of little improvements" update, with no major changes at this time. Interesting new features are some very specific cleaning modes, like linked polarizations and parallel cleaning.

Major & visible changes:

- Support for "parallell cleaning" (option '`-parallel-clean`'), which splits up images using a path search algorithm, and allows faster and clean and larger images.
- Polarization-linked cleaning ('`-link-polarizations`'), which allows cleaning a set polarizations using information from another set of polarizations (e.g. clean QUV only using I information).
- Support for calculating AARTFAAC primary beam.
- Tapering is now taken into account in the initial value of the PSF, thereby improving fit stability on tapered runs (SF #134, reported by P. Serra)
- Circular and non-circular Gaussian fitter improvements (related to SF #134).
- Added option '`-beam-fitting-size`' to allow changing the fitting box size (useful for MSSS).
- Added option '`-divide-channels-by-gaps`', which changes how bandwidth is splitted in multiple-output-channels mode.
- Adding option '`-pb-undersampling`' to turn off or change undersampling.
- Added parameter '`-lofar-centroids`', which takes centroid information into account. This was an experiment, and seemed not to be very useful (it is very slow and does not improve accuracy).
- WSClean can now grid only the even or odd timesteps (parameters '`-even-timesteps`' and '`-odd-timesteps`'). Useful for certain experiments, e.g. estimating noise value (SF #133).
- Option '`-no-normalize-for-weighting`' was removed, as keywords can be used to scale images appropriately.
- Accept 3d fits masks that have velocity instead of frequency as 3rd axis (reported by M. Ramatsoku).
- Provide an error message when automasking threshold <= auto threshold.
- Removing `-force-dynamic-join` option, as it is no longer necessary to specify it.

- IDG command line parameters have been added to the command line help.

Bugfixes / technical changes:

- CMake findcasacore script updated to fix unresolved symbols with certain Casacore versions.
- Model column was not updated when using '`-continue`' (reported by R. van Weeren and G. Di Gennaro).
- One-off error in calculating of nr scans, and reporting proper error when requesting more intervals than available (SF #131, reported by S. Bourke)
- When writing model data back to MS, progress bar did not actually show progress.
- Threading and mutexes are now done with (C++11) the stdlib. This decreases the dependency on Boosts.
- Fixed gcc 7.3 compilation: pow10 was removed from gcc 7.3 (glibc 2.27).
- Fixed unit weighting to not include flagged/unavailable visibilities during imaging when used.
- Preparations for IDG A-term correction: no longer store visibilities preweighted, pass full weighting to IDG.
- Do output errors in quiet mode.
- Don't allow rectangular images when IDG is enabled.
- Improved readability of non-verbose mode a bit; be less verbose for sets with denormal phase centre and removed some other technical output.
- Reported nr of major iterations was one too high (on cmdline as well as in fits hdr).
- Invalid floating point cmd line parameters are reported as error instead of interpreting them as zero.
- Independent QU imaging with multiple channels would not clean U: weight was not initialized (reported by R. van Weeren).
- Also fixing bad keyword 'WSCMINOR' in dirty image of second polarization.
- Lofar beam correction with only-I imaging causes segfault (Identified by A. Shulevski).

WSClean version 2.5

Released 2017-12-01

Summary: The previous release was about half a year old, but the long period between these releases has resulted in many improvements: a few major and a very large number of minor improvements. Full support for the fast IDG GPU gridder is one of the major changes, making it possible to make enormous images quickly, and clean them with all of WSClean's cleaning options. Future releases will additionally support fast a-term correction. Spectral line users can now use multi-frequency masks. Users no longer have to think about padding (/trimming): padding space is now by default added to the requested image size. This has resulted in the removal of the `-trim` parameter, which I realise will unfortunately require changes to a few pipelines. Another visible change is that many parameters were slightly changed, to make their syntax/wording be consistent. The old parameters are still accepted, but will result in deprecation warnings.

To use IDG, the external IDG library is required. See the *installation instructions* for help on linking WSClean with IDG.

Full list of new features:

- WSClean has now full support for the "IDG" GPU gridder. It allows the making of huge images in a fraction of the time of imaging with the old w-stacking gridder. Moreover, in the future it will support a-term correction *without any additional computational costs* (SF #84, SF #122, SF #123).
- It is now possible to use a 3-dimensional (multi-frequency) mask, in particular useful for line/HI imaging (SF #47, requested by S. Makhathini and P. Serra).

- New option `-padding`, which replaced `-trim`. Default padding is 1.2. Changing meaning of `-size` to mean the trimmed size. This makes `-size` more intuitive, and will in most cases no longer require to think about padding (SF #100).
- Made parameters consistent: all parameters now use the same "dashing scheme" (SF #124).
- Write version information to fits file (SF #120).
- It is now possible to *store the image weights* that WSClean calculates (SF #126, requested by O. Smirnov).
- Allow 'mas' as unit for milliarcseconds.

Bugfixes:

- (LOFAR) primary beam correction works now with baseline-dependent averaging (SF #92)
- Fixing threshold & reported flux in XX/YY or XX/XY/YX/YY cleaning (SF #51).
- Use of auto-masking on sets that have fully flagged output channels would cause segfault. This occurs in particular in spectral line imaging (reported by S. MakHathini).
- Correcting *WSCNORMF keyword* with a factor of 4. This makes the meaning of `WSCNORMF` equal to the number of cells gridded (when uniformly weighted).
- Fixed a possible stability issue in IUWT (SF #118).
- Fixes for compilation on Mac (avoiding `sincos`, `M_PI1`, `exp10`).
- Documentation improvements.
- Make sure that values are not divided by zero when part of the image is zero and local rms is used.
- No longer forcing the restoring beam to be larger than 1 pixel, which could lead to bad flux scales with very exotic beams (SF #113).
- Speed improvement: the min/maxuv/ws values are now cached between major iterations, which saves going an extra time over the data each major iteration (SF #26).
- The PB corrected source list output works now together with reduced deconvolution channels (SF #125).

Some technical low-level fixes:

- Use of `boost::optional` to represent incorrect values, thereby avoid NaN values and allowing the use of `-ffast-math`, although no change in performance (SF #95).
- Use of `unique_ptr` instead of pointers everywhere relevant.
- Added r-value support to lane.
- Several small memory leaks were fixed, e.g. in polynomialfitter.
- Several new tests were added.
- Improvements to the Python interface; better data type validation.
- Improvements to buffer allocator, so that 0-sized allocation do not cause warning on WSClean exit.
- Work on applying the LOFAR beam a-terms with IDG. This is almost finished, but not yet available in this release.
- Speed improvement: Made sure that nr of threads to be used by nwlayer gridder is not more than nlayers.
- Fixed rare issue: exceptions could cause early destruction of image buffer allocator to segfault instead of show exception.
- Correct displaying of NaN value for flux density (Jy) values.

WSClean version 2.4

Released 2017-05-28

Summary: Version 2.4 mostly contains small improvements on various points. Improved source component output allows better integration with (DD and DI) self-calibration pipelines, and continued work on the integration with IDG is aiming towards a fast, full A-correction imager.

New features:

- Option `-save-source-list` to save a component list including Gaussians for multi-scale components. The output format is supported by DPPP. Old option `-save-component-list` has been removed.
- Source lists can be primary-beam corrected by combining `-save-source-list` with `-apply-primary-beam` (SF #111).
- Added keywords `TELESCOP`, `OBSERVER` and `OBJECT` to fits output (SF #112).
- Fitting a Gaussian to the PSF is now more stable, and deals better with border cases (reported by G . Heald).
- Multi-frequency cleaning will now properly take channel weights into account during spectral fitting and peak finding (SF #109)
- Flux density values are now reported with sensible units, and units can be given on the cmdline (SF #104).
- WSClean will by default now stop deconvolution after 20 major iterations, which can be changed with the new option `-nmiter` (SF #117, requested by N. Hurley Walker).
- Option `-save-psf-pb` to be able to get a corrected PSF.
- Option `-circularbeam` now properly performs a constrained fit (SF #55).
- Option `-multiscale-convolution-padding` to change the default padding.
- WSClean now reports the reason for stopping cleaning.
- Slight increase of gridding speed, because X and Y dimensions of gridding kernel are now stored separately. Larger oversampling rates are now also supported.

Bugfixes:

- Parameter `-stopnegative` was ignored since the new generalized cleaning algorithm was implemented (reported by N. Hurley-Walker and C. Tremblay).
- In multi-scale, joined polarizations mode, WSClean could not determine scale sizes when last channel has no unflagged visibilities (Reported by R. van Weeren).
- In joined polarization mode, WSClean could continuously perform further major iterations without improving the residuals any further (reported by N. Hurley-Walker).
- Deprecated parameter name `-rms-background`, which was renamed to `-local-rms`.
- Fixes for IDG gridding; CPU and GPU gridding & cleaning with IDG works.
- Version information now includes availability of IDG.
- More unit tests.
- Support proper displaying of negative angles.
- Refactoring of primary beam correction code.

WSClean Version 2.3

Released 2017-03-01

Summary: The most prominent new feature is *local RMS thresholding* using either an automatically generated RMS map or by providing an RMS map. When using multi-scale clean, a component list using Gaussians can now be requested. This release also fixes some larger bugs that had been introduced in the Hogbom/generalized Clark clean transition, so I recommend to upgrade.

New features:

- *Local RMS thresholding*, options `-rms-background`, `-rms-background-window` and `-rms-background-method` (SF #102, SF #105).
- Use channel weights when fitting spectral polynomials. This helps in particular to properly ignore missing channels, which would formerly cause bad fits (SF #107).
- Option `-save-component-list` for outputting the Gaussian component lists in multi-scale clean.
- Improve speed of multi-scale clean, by setting the clark masking threshold lower than the stopping threshold in the subminor loop optimization.
- Added a `-multiscale-shape` parameter to either select Gaussians or a tapered squared function.
- WSClean now ships with an example program `wsuvbinning` to produce a csv file with binned visibilities, properly normalized to values in Jy (requested by C. Spingola).
- Fits files now contain a sum-of-visibility-weights keyword `WSCVWSUM` which can be used to calculate the nr vis per uv bin (this is required to use the `wsuvbinning` tool).

Bugfixes:

- Rectangular images (i.e., width != height) caused cleaning to fail.
- Crash when performing full XX/XY/YX/YY imaging, causing missing MFS files. Reported by T. Franzen.
- Bug in -intervalsout; interval index would not increase, causing each next index to overwrite the previous index.
- Segfault when chosing an odd image size and enabling beam correction, reported by M. Iacobelli. Caused by FFTW not properly supporting unaligned r2c fft.
- Segmentation fault when auto-masking with multiple output intervals.
- Some improvements to the IDG connection.
- Work on a polynomial channel fitter as described in the WSClean multi-scale clean paper, which in the end will allow better multi-frequency fitting. This is not finished, see ticket (SF #110).
- On some platforms, parsing any integer command line parameter resulted in an exception (fixed between 2.2.0 and 2.2.1).

WSClean Version 2.2

Released 2017-01-20

Summary: This version improves the speed of Högbom clean and joined clean significantly, improves the stability of multi-scale clean, and replaces several specialized cleaning methods by one generic algorithm that makes several features work in all clean modes.

New features:

- Use the Clark optimization (Clark 1980) in Högbom clean and joined clean, which typically speeds up cleaning by a factor of ~5 (SF #14).

- Faster multi-scale clean, by adding option `-multiscale-gain` and setting it to 0.2 by default.
- Turned the weight rank filter on by default with a value of 3.0.
- Auto-masking works now also in Högbom clean mode (SF #101).
- Refactored cleaning algorithms to use the same code for single-scale cleaning, which makes all generic parameters work now in all cleaning modes (e.g. `-deconvolution-channels` and `-squared-channel-joining`; (SF #58).

Bugfixes:

- Report an error when invalid characters are used in integer parameters, like `-niter 1e5`.
- Multi-scale clean will no longer put large-scale kernels through the image edge, which makes the algorithm more stable (#103).
- Perform FFT convolutions with padding in Multiscale fast subminor optimization and Clark optimization in normal clean, which improves stability.
- Make auto-masking work without specifying an auto-threshold.
- Fixing possible race condition in inversion code of the gridder.
- Always use C++11 versions of lane and uvector.
- Removed old code for old 'fast multi-scale algorithm' which did not work as well.
- Optimized clean functions in ImageSet class when only one image is cleaned.
- Adding test utilities for Högbom clean.
- Cleaning up of code using cppcheck.

WSClean Version 2.1

Released 2016-12-19

Summary: This version adds important enhancements to multi-scale clean. Deep cleaning (>=100k - 1M iterations) of large images with multi-scale clean is now 5-10 times faster in sets where deconvolution is the bottleneck. Furthermore, an *automatic masking feature* using `auto-mask` is now available in multi-scale clean, which allows automated deep cleaning. Together with `auto-threshold`, it makes deep cleaning quite easy. A typical run now looks something like this:

```
wsclean -auto-threshold 0.3 -auto-mask 3 \
  -niter 1000000 -mgain 0.8 -multiscale \
  -size 4096 4096 -trim 3072 3072 -scale 30asec \
  obs.ms...
```

This cleans to 3 sigma and keeps track of what components at what scales are cleaned. Once a maximum 3 sigma peak is reached, it will turn this information into a scale-dependent mask, such from then on, only components/scales are selected that were already found, and continues cleaning up to 0.3 sigma. This typically is good enough to only show residuals caused by calibration errors and noise.

New features:

- Automatic masking with `-auto-mask` parameter.
- Much faster multi-scale algorithm (can be disabled with `-no-multiscale-fast-subminor`).
- Parameter '`-restore`' can be used to restore a model image on a residual image (SF #71).
- Parameter '`-help`' was added.

- In prediction-only mode, the image size and scale is now read from the fits file, instead of requiring manually setting the size and scale (SF #97).

Bugfixes:

- For linear polarizations, the signs of Stokes Q and V have been flipped! The polarization labelled X now represents N-S instead of E-W. After much debate, it was agreed that this is the proper Casacore Measurement set convention, even though it is different in UVFits files and importuvfits will not rotate the polarization angle.
- Make sure compilation succeeds when no SSE is available
- When all *w*-values are set exactly to zero, *w*-stacking would fail (Reported by B. van der Tol).
- Major refactoring of fits writing code (SF #94).
- Improved FITS keywords of UV fits files (SF #96).
- Better input parameter checking and improved error messages.
- Added information about whether lofar beam lib was found to output of `-version`.
- WSClean now requires `-size` and `-scale` to be set, instead of using default values when they are not set.
- Improve output and cleaning up of WSClean when an exception occurs.

WSClean Version 2.0

Released 2016-11-04

Summary: This version adds *baseline-based averaging*, a feature that can improve the computational performance of imaging. Two other new features are `-auto-threshold`: using *a threshold relative to the noise level*; and `-continue` to *start off where a previous deconvolution finished*. Furthermore, quite a lot of small bugs were fixed.

New features:

- *Baseline-based averaging*) (SF #85).
- Option `-continue` to *continue a previous run* (SF #79).
- Option `-auto-threshold` to *calculate the threshold* from the residual noise level.
- Option `-spws` to select which bands to image (SF #83).
- Primary beam correction for MF image. Primary beam corrected versions of the residual and model images are saved too (SF #82, requested by T. Shimwell).
- Option `-simulate-noise` to perform a noise-only simulation.
- Connection to the Image Domain Gridder (IDG) is finished, testing continues.
- Cleaned up FITS keywords, added keyword 'WSCNORMF' to allow easier PS normalization (SF #75).
- WSClean unit tests are now shipped in the distribution and can be run with 'make check'.

Fixes:

- Use no cleaning border when a trimmed size is specified (SF #81).
- Frequency headers in FITS file were wrong for datasets with multiple spectral windows (reported by R. van Weeren).
- Channel width not set in fits if input has 1 channel (SF #87, reported by S. Bourke).
- Incorrect frequency headers in PSFs (SF #93).
- Incorrect frequency headers when imaging multi-spw sets (SF #90, reported by R. van Weeren).

- Fitting of MFS image fails when highest frequency is flagged (reported by K. Emig).
- Fix for imaging sets with reversed channels, which seems common for Miriad sets (Reported by L. Pratley).
- Adding error message when requesting joined channels but not requesting more than one output channel (suggested by F. de Gasperin).
- Solving bad deconvolution accuracy when single polarizations are flagged.
- Improved speed of convolving the model with the restoring beam.
- LOFAR beam did not use the right ra,dec in the first instance of beam creation.
- Update help of `-beamsize`, `-beamshape` and `-theoreticbeam`.
- Make sure that primary beam is not updated faster than the integration time (Reported by A. Nilsson).
- Fixed crash when applying beam on multi-channel set
- Fixed compilation of the 'operator' library interface for connecting with custom deconvolution algorithms.
- Enabled fast math for gridder (although no significant change in speed).
- Fixed compilation on a Mac.
- Better validation and error repporting of input.
- Some refactoring.
- *Changelog for WSClean 2.10* (2020-07-15)
- *Changelog for WSClean 2.9* (2020-03-27)
- *Changelog for WSClean 2.8* (2019-09-16)
- *Changelog for WSClean 2.7* (2019-04-19)
- *Changelog for WSClean 2.6* (2018-06-11)
- *Changelog for WSClean 2.5* (2017-12-01)
- *Changelog for WSClean 2.4* (2017-05-28)
- *Changelog for WSClean 2.3* (2017-03-01)
- *Changelog for WSClean 2.2* (2017-01-20)
- *Changelog for WSClean 2.1* (2016-12-19)
- *Changelog for WSClean 2.0* (2016-11-04)

1.2.3 Version 1 releases

WSClean Version 1.12

Released 2016-07-09

Summary: No major feature enhancements, but many small improvements and fixes. *Differential LOFAR beam* is probably the biggest new feature, while *some new deconvolution modes* can improve RM-synthesis imaging.

New features:

- Support *correction of the differential LOFAR beam* instead of full beam.
- Option `-squared-channel-joining` for [using the channel-squared values during peak finding](RMSynthesis), instead of channel-summed.

- Option to turn off dirty image creation (`-no-dirty`).
- Adding WSCNVIS and WSCENVIS keywords to fits files.
- During imaging, WSClean now outputs the 'effective visibility count'.
- Rewrote dft prediction algorithm, it is now possible to use NDPPP for prediction.
- Adding cleaned flux to output when performing multi-scale clean (requested by A. G. de Bruyn).
- Better determination of available number of CPUs, to make WSClean cooperate with cluster management tools like SLURM (Using CPU affinity code from J. D. Mol).
- Support for masks in the IUWT deconvolution method.
- Adding time to output when requested new parameter with `-log-time` (SF #76).

Fixes:

- Changed behaviour of SPWs to not count unused SPWs in calculation of output channels (Requested by R. van Weeren).
- Model data corrupted when using `-predict` because of bug in contiguous MS writer (SF #78).
- Issue with `-predict` mode – prediction no longer used the kaiser bessel kernel, because of an unset value.
- Use WEIGHT column if WEIGHT_SPECTRUM is not available (SF #73).
- Fully flagged output channels causes joined cleaning to malfunction (reported by K. Emig).
- Increased precision of fits keywords to have double precision for RA and DEC output, otherwise VLBI observations have slightly inaccurate positions (Reported by R. Dean).
- Issue with IUWT in combination with `-deconvolution-channels`, reported by O. Smirnov.
- Some usual refactoring and fixes for newer compilers.
- Some preparation to allow IDG as gridder.
- Imaging multiple polarizations in multi-frequency, multi-spw modes would lead to an error (reported by R. van Weeren).

WSClean Version 1.11

Released 2016-03-11

Summary: This version adds *primary beam correction for LOFAR* and uses a stronger default anti-aliasing kernel (SF #76). The new kernel decreases the chance of sources far out appearing as ghost sources, but increases the noise added near the borders of the image. It is recommended to trim the borders with the `-trim` option.

Full list of changes:

- New feature: Correct Stokes I or IQUV images with the LOFAR primary beam (SF #70).
- New feature: Option `-multiscale-scales` to select scales that are to be used in multiscale (SF #72).
- New feature: Rectangular images (i.e. non-square images) can now be made (SF #49).
- New feature: Report multiscale clean statistics (SF #69).
- New feature: Added option -make-psf-only and renamed -makepsf to -make-psf.
- New feature: Rectangular gridding function added, option `-gridmode rect`.
- Bugfix: Improved default anti-aliasing kernel, uses Kaiser-Bessel parameter of 8 instead of 3 (SF #67).

- Bugfix: Fix calculation of theoretic beam and max/min w-values when multiple measurement sets are specified (reported by R. van Weeren).
- Bugfix: PSF normalization bug, now always normalizing the PSF to be unitary in the centre. Only affected large/high-w-term images (reported by O. Smirnov).
- Bugfix: Improved Casacore CMake scripts to improve compatibility with different Casacore setups (reported by T. J. Dijkema).
- Bugfix: Non-normalized images for power spectrum creation should be multiplied by 2 for correct unit-level UV values.
- Bugfix: Delay opening of measurement set in PartionedMS to have less files open.
- Bugfix: Fixing possible valgrind warning in FitsReader.
- Bugfix: Adding error message when specifying invalid trim size (noted by M. Iacobelli).
- Bugfix: Better error check when something is wrong with the data selection and/or scale.
- Bugfix: Multi-frequency modes do not work when channels are reversed (SF #68).
- Fixing possible leaking of FITS cio file handlers during reading
- Some refactoring (SF #46).
- Textual: Some help fixes.

In the mean time, chgcentre 1.3 and 1.4 were also released. For version 1.3, chgcentre was converted to work with casacore 2. Version 1.4 solves an issue when trying to rotate a set with denormal phase centre (option `-shift-back`) for the second time (requested by M. Rioja).

WSClean Version 1.10

Released 2015-12-12

Summary: New features include *several tapering options*, new multi-frequency deconvolution strategies that impose spectral smoothness in a fast way. Furthermore, lots of small improvements were made and some bugs were solved.

Note that WSClean needs Casacore 2.0 or higher, with C++11 support turned on (since WSClean 1.9).

Full list of changes:

- Implemented a Gaussian taper (SF #62; option "-taper-gaussian").
- Implemented a "don't grid near the uv-plane-edge" taper ("-taper-edge").
- Implemented Tukey tapers: inner, outer (uv-distance; these have a circular shape) and edge (distance to uv-plane-border; these have a square shape) taper, options "-taper-tukey", "-taper-inner-tukey", "-taper-edge-tukey". See *tapering*.
- Support for fitting a spectral function during joined channel deconvolution in multiscale, IUWT and normal cleaning was added. Linear and power-law fitting is supported. (SF #36)
- Decrease frequency resolution during cleaning (option "-deconvolution-channels"), and interpolate up/down to imaging frequency resolution with a given spectral function (SF #57).
- New option "-trim" to trim the output image before cleaning & writing to fits, to avoid noisy borders.
- Added option "-nwlayers-for-size" to calculate automatic nwlayers for a different size.
- Some work on linearly-interpolated spectral fitting.
- The low-level Python "operator" API has been extended with more documentation, it is fully tested and has been tested with RESOLVE.

- Write the dirty MFS image when using -channelsout != 1.
- Added option "-subtract-model", to subtract the initial model from the data and allow re-imaging an already cleaned set (SF #53).
- The MFS PFS is now fitted with the beam, and beam keywords are added to the other MFS images (SF #42).
- Render the MFS model onto the MFS residual image to create the MFS restored image, such that the MFS restored image matches the fitted MFS beam.
- Allowing 'spectral moresane': The spectral fitting and deconvolution channel parameters can now be used to run Moresane (requested by O. Smirnov).
- Measurement sets with multiple SPWs are correctly handled (SF #37).
- Added options -quiet and -verbose to modify verbosity (SF #60).
- Bugfix: Major bug in multi-frequency multi-scale cleaning; peak finding was not always performed on sum of channels and reported fluxes were wrong.
- Bugfix: Updating the model data after reordering when -channelrange is specified would write to wrong channels (reported by O. Smirnov).
- Adding option "-theoreticbeam", never use the theoretic beam otherwise (SF #43)
- Bugfix: Multiscale crashes with multiple polarizations without joining (SF #52)
- Bugfix: Incorrect evaluation of angles specified in radians – WSClean would actually assume radians are asec.
- Bugfix: Joined cleaning would no longer honour mgain and continue cleaning to threshold when initial peak is negative.
- Bugfix: Bug in trimming ("-trim …") with CS iterations; untrim was broken.
- Bugfix: In the small inversion optimization, always pick an image size that is easily factorizable (SF #65).
- Bugfix: incorrect documentation for -predict (Reported by F. de Gasperin).
- Bugfix: Compilation errors with WSClean using older LOFAR beamresponse library (SF #56; reported by A. van Amersfoort and A. Horneffer).

WSClean Version 1.9

Released 2015-08-21

Summary: This release introduces some major changes, both under and above the hood. First of all, there are two new deconvolution methods: *IUWT deconvolution*, which is similar to *MORESANE*), and a *new multi-scale algorithm* that is much more accurate than the previous algorithm. The 'channelsout' multi-frequency option now treats multi-measurement-set imaging properly. Finally, all combinations of "-joinchannels", "-joinpolarizations", "-channelsout", "-pol ..", "-iuwt" and "-multiscale" that one can possibly wish are supported.

Note that WSClean now needs Casacore 2.0 or higher, with C++11 support turned on.

Full list of changes:

- *New multi-scale algorithm* which is more accurate, somewhat slower in most cases, but still fast. Enabled with option "-multiscale".
- The older algorithm is now available with option "-fast-multiscale", but should probably not be used for anything serious.
- The multi-scale mode now supports masks (SF #38).
- A new *IUWT compressed sensing algorithm* which is comparable to *MORESANE*, but allows multi-frequency deconvolution and is faster on CPU.

- Support dividing total bandpass using -channelsout when specifying multiple MSs of different frequencies. (SF #44)
- Rewrote the deconvolution code to use the generic ImagingTable. This allows jointly cleaning over less common polarizations combinations (e.g. Stokes I and Q) as well as cleaning channels jointly without jointly cleaning the polarizations. (SF #39)
- WSClean's gridding engine (class WStackingGridder) has been refactored, and it is now possible to reuse the gridder (API docs).
- Add -moresane-sl parameter to set MORESANE deconvolution depth levels per iteration (patch by O. Smirnov).
- Code was converted to Casacore 2: version 2 or higher compiled with C++11 support is now required.
- Bugfix: Add prefix to MORESANE input filenames, to avoid clashes with parallell imaging runs (SF #48, patch by O. Smirnov).
- Bugfix: Conversion from circular to Stokes Q, and from Stokes to circular was not implemented (/correctly). Cotton-Schwab iterations are now possible with circular feeds (e.g. JVLA). (SF #45)
- Bugfix: Report error when specifying invalid channel range.
- Bugfix: Removing -imaginarypart parameter from wsclean's help, as it has been replaced (way back) by combined real/imag imaging for xy/yx.
- Bugfix: Make it easier to build with platform independence (SF #50).
- Bugfix: WSClean could crash on zero-length selection (either because of no rows in the selection or no channels in selection).
- Bugfix: Better error message when mmap fails.
- Bugfix: Changing console output of prediction step to correspond with FTs/prediction stage, and removing redundant 'Rows that were required' output
- Lots of cleaning up and renaming.

WSClean Version 1.8

Released 2015-05-21

Summary: The new weight rank filter allows uniform weighting with better noise properties. Programming interfaces to use the inversion engine of WSClean and a simple Python interface to call WSClean were added. A lot of bugs were fixed; many small ones but also pretty big ones, including incorrect flux scaling for very wide images.

Full list of changes:

- Adding weighting rank filter options "`-weightingrankfilter`" and "`-weighting-rank-filter-size`". This decreases the weight of uv-samples with very few samples in it, thereby decreasing the noise in the image (with no effect on the beam).
- New option "`-saveuv`", which stores the gridded uv data & sampling function after w-correction .
- Added a C interface to use the WSClean inversion/prediction algorithm as an operator, useful for testing compressed sensing algorithms.
- Added a Python interface to enable calling WSClean from Python code, both as an operator (like the C interface) or the whole imaging algorithm.
- Setting default behaviour to prefer reordering also when mgain!=1.0.
- Support masks in Moresane deconvolution.

- Bugfix: Removing sqrt(1-l^2-m^2) normalization. This changes the flux scaling of wide field imaging (>~20deg), which used to be wrong. Reported by S. Bourke and M. Eastwood; see SF #35 for more info (SF #35)
- Bugfix: Crashes when cleaning near the horizon, caused by incorrect bounding box calculation during restoring. Reported by S. Bourke. (SF #33)
- Bugfix: With natural weighting, the uv-l cuts did not work. Uniform, Briggs and Natural weighting now follow the same code path for determing the weight.
- Bugfix: Issues with beam keywords and restoration in multi-channel runs. (SF #41, related to SF #42)
- Bugfix: Fixing polarization keyword in model FITS files when using joined polarization and/or channel cleaning (reported by D. Rafferty).
- Bugfix: Integrated "MFS" images have wrong phase centre keywords when using `chgcentre -shiftback`
- Bugfix: When no reordering is performed, NaNs in an MS would still be counted in calculating the total weight (but they should not).
- Bugfix: Making sure first threshold in multiscale cleaning is not negative, which would lead to a diverging clean
- Bugfix: A bug that could cause DATE-OBS to be set like 2013-08-18T16:60:01.0, reported by N. Hurley-Walker
- Optimization: Significantly increased speed of reordering when using interval range (`-interval` or `-intervalsout`).
- Some work on a minimal-noise deconvolution technique.
- Textfix: Some extra detail in `-circularbeam` help

André Offringa

WSClean Version 1.7

Released 2015-03-10

Summary: Deconvolution can be performed with a compressed sensing method (*MORESANE*) and clean masks are now supported. Also, determining the restoring beam size by fitting the psf is now the default; previously this was only done when `-fitbeam` was specified.

Besides new features, version 1.7 has a long list of bugfixes so I recommend everyone to update.

Full list of changes:

- *MORESANE Compressed sensing deconvolution* using J. Kenyon's python MoreSane implementation.
- Cleaning now supports cleanmasks, either with a casa file (`-casamask`) or a fits file (`-fitsmask`).
- Fitting a Gaussian to the PSF is now the default for determining the restoring beam size.
- Added convenience parameter `-intervalsout` to make fits snapshot movies.
- Added option `-no-update-model-required` to skip writing the model column when possible, which speeds up -mgain<1 runs that include reordering.
- Added option `-saveweights` to store the gridded weights as fits file.
- Bugfix: Samples which were gridded at v=0 were slightly downweighted in uniform/briggs weighting due to one-off bug in (inverse-symmetric) weighting grid. This could manifest itself in a PSF which was slightly less aesthetically pleasing, although in most cases it is hardly noticeable (especially in cleaned images).
- Bugfix: In multi-scale mode, there was half a pixel offset in restored structures.
- Bugfix: Rare time-rounding error, causing incorrect DATE-OBS value in output fits files. Reported by D. Kaplan.

- Bugfix: Bug causing infinite continuing cleaning in joined channel cleaning when ending on negative component.
- Bugfix: *Image weight keyword* is incorrectly converted to int, leading to possible MFS images with NaN values. Reported by R. van Weeren.
- Bugfix: WSClean used REFERENCE_DIR instead of PHASE_DIR for image centre, giving incorrect coordinates when NDPPP was used for phase shifting (SF #30). Reported by R. van Weeren.
- Bugfix: Crashes when cleaning near the horizon, caused by incorrect bounding box calculation during restoring (SF #33). Reported by S. Bourke.
- Bugfix: When specifying multiple measurement sets with different bands, frequency keyword now represents central frequency.
- Bugfix: Fixing FITS keyword bug; WSClean used to write 'Hz' in CUNIT4 (the 'Stokes' dimension). Reported by M. Iacobelli.
- Textual: Help on `-beamsize` parameter said default unit is arcmin, but since 2.5 it is actually in arcsec (when not specifying a unit). Reported by M. Iacobelli.

These bugs were fixed after internal releases:

- Bugfix: Fixing bug in masked cleaning causing spurious components.
- Bugfix: Honour cleaning border when using masks and in single pol mode (SF #32).

WSClean Version 1.6

Released 2015-01-23

Summary: This version has a few changes that are especially useful for reducing LOFAR data: it supports correcting the LOFAR beam during prediction, and the speed of major iterations on typical LOFAR measurement sets was increased.

When compiling version 1.6, cmake will now search for the LOFAR StationResponse library. If not found, the `-dft-with-beam` option will not work.

Full list of changes:

- Feature: It is now possible to do the prediction with a DFT, option `-dft-prediction`. This is very slow, but allows correcting for a beam.
- Feature: LOFAR beam can be computed, option `-dft-with-beam`.
- Performance fix: Normal prediction during major iterations is now buffered, which leads to ~3x faster prediction.
- Bugfix: Predicting with option `-predict` crashes (reported by N. Hurley-Walker).
- Bugfix: Joined-frequency cleaning with one polarization did not clean negative components.

WSClean Version 1.5

Released 2014-12-16

Summary: The PSF can now be fitted with an elliptical Gaussian, it is now possible to use "-scale 1asec" as well as set a uv-cut in lambda on top of in meters and several bugs were fixed.

Full list of changes:

- Beam fitting: option `-fitbeam` (/ `-nofitbeam`) enables fitting an elliptical Gaussian to the PSF, and restoring is subsequently also performed with this elliptical PSF. This is not yet the default. (SF #17)
- New options `-circularbeam` and `-ellipticalbeam` to control beam fitting.
- New option `-beamshape` to set major/minor/position angle for elliptical beam shape.

- Option `-beamsize` now takes arcseconds by default. It sets a manual circular beam, and understands units (e.g. `-beamsize 2arcmin`).
- Wide-band mode can now also be enabled for single-polarisation imaging (SF #11).
- The -scale parameter now accepts units, e.g. "-scale 1arcmin" (SF #21).
- New option `-tempdir` for specifying alternative temporary directory for reordered files (SF #25).
- No longer save gridding correction image by default, only save when new option `-savegridding` is given.
- New option `-minuv-l` and `-maxuv-l`, which set the maximum uv values that are still gridded.
- Options `-minuvw` and `-maxuvw` are renamed to `-minuvw-m` and `-maxuvw-m` to distinguish from above new options.
- Bug fix: WSClean keywords were missing in MFS images (SF #20). Reported by N. Hurley-Walker.
- Bug fix: parameters -multiscale-threshold-bias and -multiscale-scale-bias didn't have effect in some multiscale modi.
- Bug fix: Crash "Table column WEIGHT_SPECTRUM is unknown" when no weight column present. Reported by S. Makhathini (SF #22).
- Bug fix: -interval parameter didn't work correctly when using reordering. Reported by A. Rowlinson.
- Bug fix: DATE-OBS keyword in fits output should reflect -interval setting. Reported by A. Rowlinson (SF #23).

WSClean Version 1.4

Released 2014-10-06

This version is one of the smallest updates as of yet, but does have some important bug fixes. The small inversion is now on by default, there was something wrong with the stored XY dirty images in some cases (the 'image' restored images were not affected) which was fixed, and a big memory leak was fixed.

Full list of changes:

- Small inversion optimalisation is now on by default (SF #18)
- New option `-nosmallinversion` to disable the small inversion
- Bugfix: Fixing Briggs' MFS weighting in multi-channel mode, refactoring ImageWeights (SF #12)
- Bugfix: Corrected writing of residual dirty XY image and no longer write the dirty YX-i (SF #9).
- Bugfix: Don't crash when one of the fits keys is NaN or infinite (SF #10). This fixes a crash when creating a psf with an empty data selection.
- Bugfix: Report an error when trying to predict from an image with NaNs or other non-finite values
- Bugfix: Removing two memory leaks, one of which might cause some major leakage
- Some refactoring of the weighting calculations.

WSClean Version 1.3

Released 2014-08-05

In this version, it is now possible to select the new & supposedly fast multi-scale algorithm I've been working on. It works well without tweaking on some MWA data of Vela & Puppis A, but other telescopes / sets might require tweaking of the parameters (with `-multiscale-threshold-bias` and `-multiscale-scale-bias`). For those using Cotton-Schwab cleaning that includes XY/YX polarizations, I urge upgrading to this version for some of the bug fixes.

Changes:

- Feature: multi-scale cleaning can now be selected by using `wsclean -multiscale`. This is a new, supposedly fast algorithm for multi-scale cleaning and has shown good initial results. It's still somewhat experimental, and might require tweaking the threshold bias and scale bias parameters (see the help).
- Feature: `wsclean -version` will now report version info
- Bugfix: Fixing big bug in prediction for XY and YX – sign error in imaginary image values before prediction, on baselines with w>=0. This could potentially cause noisier/incorrectly deconvolved XY/YX images.
- Bugfix: Fixing writing of wrong imaginary residual image – the actual saved image in XYi-residual was the real value of XY
- Bugfix: Throw exception when cleaning XY/YX polarizations without joined pol
- Bugfix: Fixing help on -joinpolarizations and -pol
- Bugfix: Fixing copy constructor of fitsreader – causing segfault on complex image prediction
- Bugfix: Fixing error in -predict mode with XY/YX polarizations
- Bugfix: Fixing issue with -predict: wouldn't use Kaiser-Bessel gridding by default
- Bugfix: Fixing problem in -predict with uninitialized reader causing possibly undefined behaviour
- Bugfix: Making sure that uvw samples outside the field of view do not force more w-layers
- Bugfix: Fixing possible crash when imaging beyond horizon
- Bugfix: Extra safety check on input parameters in WSInversion

WSClean Version 1.2 and 1.2.1

Released 2014-05-29

This version fixes one major bug (cleaning didn't always work, SF #6) and adds support for dual-polarization cleaning.

Changes:

- New feature: Joined cleaning of two polarizations; previous version could only joinedly clean 4 polarizations. Particularly useful for cleaning XX,YY. Enabled by e.g. `-joinpolarizations -pol xx,yy`.
- New feature: Prediction-only mode (option `-predict`).
- New feature: Added -j option to limit nr of cores used.
- New feature: Added option `-cleanborder` to prevent cleaning of border, now defaults to 5%.
- Major bug fix: Fixing issue with restored images in single pol mode not being restored from residual properly (SF #6).
- Bug fix: Intel compiler didn't like certain C++11 features.
- Bug fix: Only one psf is saved in multi-channel mode (reported by A. Neben).

- Bug fix: when using natural weighting and when imaging below the synthesized beam resolution, uv-samples would wrap around in the uv field, causing imaging artefacts (reported by A. Neben).
- Bug fix: Fixing imaging of sets with multiple spectral windows.
- Bug fix: Malfunctioning MFS weighting mode with continuous reader.
- Bug fix: Cleaning of images with only zeros does not end.
- Bug fix: Issue with FindPeak, causing incorrect indices and not proper cleaning in certain modi.
- Textual: Improved help message and some other output.
- New (disabled) feature: A fast, new multi-scale clean algorithm is implemented, but it is not compiled in at the moment pending further testing.

Version 1.2.1, released one day after 1.2, fixes a missing "`uvector_03.h`" error that made older (non-C++11) compilers fail.

André Offringa

WSClean Version 1.1

Released 2014-04-23

This version introduces new features including a wideband deconvolution method, and fixes quite a few bugs.

Changes:

- New dependency: GSL and CBLAS are now required to build WSClean.
- New feature: *Cleaning channels joinedly* with cmdline parameter: `-joinchannels`. This performs peak finding in the sum over all output channels, but subtracts components independently from the channels.
- New feature: Option to grid weights once for all output channels (`-mfsweighting`).
- New feature: When spectral imaging (with `-channelsout`), a weighted MFS image will be created.
- New feature: Direct imaging of Stokes IQUV, and possibility to clean joinedly over these too (requested by O. Smirnov, SF #2).
- New feature: Support imaging specific polarizations (rr / Q / ..) in measurement sets with circular polarizations.
- New feature: Specify absolute memory limit instead of relative limit (`-absmem`), requested by M. Bell.
- New feature: Implemented options -minuvw and -maxuvw to set range of baselines to be gridded.
- Syntax change: explicit option for performing joined-polarization cleaning (`-joinpolarizations`), no longer the default.
- Change: When imaging >= 4 polarizations, reorder by default.
- Bugfix: Fixing some valgrind errors and a possible segmentation fault.
- Bugfix: 'make install' does not work (reported by G. Molenaar, SF #1).
- Bugfix: With spectral imaging, the uvw are not correctly divided by lambda, leading to incorrectly scaled images.
- Bugfix: Problem with `-smallinversion` and used memory.
- Bugfix: PartitionedMS handler does not handle an invalid WEIGHT_SPECTRUM column (reported by I. Heywood).
- Bugfix: When reordering an MS, end with writing reordered model data back to measurement set, so that self-cal etc is still possible when using Cotton-Schwab cleaning.
- Bugfix: `-interval` parameter does not work properly.

- Bugfix: Documentation for `-pol` parameter is out-dated (reported by O. Smirnov, SF #3).
- Bugfix: Remove temporary image files

André Offringa

WSClean Version 1.0

Released 2014-03-15

This is the first public WSClean version. The features currently supported:

- Fast wide-field imaging
- Hogbom cleaning & Cotton-Schwab cleaning
- Joined-polarization cleaning (peak finding in $xx^2 + xy^2 + yx^2 + yy^2$)
- W-snapshot imaging
- Most common weighting modes (uniform, Briggs', natural, super and sub-pixel weighting)
- Adjustable antialising kernel
- All important operations are multi-threaded

A paper has been submitted that shows WSClean's accuracy is at least equal to the w-projection imaging in CASA, and sometimes even better, with a considerable faster imaging time when large w-values are involved.

André Offringa

WSClean version 1 was in development from 2014 to 2016. Stable releases:

- *Changelog for WSClean 1.12* (2016-07-09)
- *Changelog for WSClean 1.11* (2016-03-11)
- *Changelog for WSClean 1.10* (2015-12-12)
- *Changelog for WSClean 1.9* (2015-08-21)
- *Changelog for WSClean 1.8* (2015-05-21)
- *Changelog for WSClean 1.7* (2015-03-10)
- *Changelog for WSClean 1.6* (2015-01-23)
- *Changelog for WSClean 1.5* (2014-12-16)
- *Changelog for WSClean 1.4* (2014-10-06)
- *Changelog for WSClean 1.3* (2014-08-05)
- *Changelog for WSClean 1.2* (2014-05-29)
- *Changelog for WSClean 1.1* (2014-04-23)
- *Changelog for WSClean 1.0* (2014-03-15)

1.3 Installation instructions

1.3.1 Getting the source code

Unless you need specific new features, it is recommended to use a stable (tagged) version of WSClean. These can be downloaded from https://gitlab.com/aroffringa/wsclean/-/releases.

To retrieve the (experimental) master branch of WSClean, use git:

```
git clone -b master git@gitlab.com:aroffringa/wsclean.git
```

This will retrieve the *master* branch of WSClean.

1.3.2 Manual compilation

After downloading the source code, one will need to compile WSClean. WSClean requires:

- Casacore, for opening measurement sets. Version >=2.0 is required, not lower. Casacore is required even if you already have Casa installed. Casacore needs to be compiled with C++11 support, which is the default for the latest version.
- FFTW version 3.3.5 or newer, used to perform Fourier transformations.
- Boost, used for threading, date and time calculations and some other general functionalities.
- CFITSIO, for reading and writing FITS files.
- GSL, the GNU Scientific Library, used for certain computations.

WSClean uses C++11 features. Because of this, building WSClean with GCC requires at least GCC version 4.8.1.

To use the *image-domain gridder* (a fast GPU gridder), you will need to install the IDG libraries from https://gitlab.com/astron-idg/idg. To apply primary beams, the EveryBeam package is required from https://git.astron.nl/RD/EveryBeam. To use the distributed mode, OpenMPI is required.

After installing these dependencies, compile WSClean with the following commands:

```
mkdir -p build
cd build
cmake ../
make -j 4
sudo make install
```

If cmake reports errors, you might have to install or specify your libraries in the call to cmake if they are not in standard directories, e.g.:

```
cmake ../ -DCMAKE_PREFIX_PATH=/opt/cep/casacore/
```

to add that directory to the search path. To add multiple directories to the search path, put the paths between double quotes and separate them with a semicolon:

```
cmake ../ -DCMAKE_PREFIX_PATH="/path/to/casacore;/path/to/cfitsio"
```

1.3.3 Installed files

After succesfully running `make install`, the following programs will be installed:

wsclean
: The main executable

wsclean-mp
: The main executable for distributed runs

chgcentre
: A program to change the phase centre of measurement sets, see *chgcentre*.

1.3.4 On Ubuntu and Debian

Binary packages are available on Ubuntu and Debian, and can be installed with `sudo apt-get install wsclean`. In case that version is new enough for your purpose, you're all done. If you want to compile WSClean from source, the following packages need to be installed:

```
apt-get install  \
  casacore-dev libgsl-dev libhdf5-dev \
  libfftw3-dev libboost-dev \
  libboost-date-time-dev libboost-filesystem-dev \
  libboost-program-options-dev libboost-system-dev \
  libcfitsio-dev cmake g++
```

The LOFAR beam and IDG libraries are optional, but need to be installed manually from source if they are required (see elsewhere on this page).

1.3.5 On Red Hat

The following document lists some instructions that can be helpful for instaling WSClean on Red Hat and CentOS: *Installing WSClean on Red Hat and CentOS*.

1.3.6 Linking errors

If you get undefined reference errors in casacore-code when you compile WSClean, e.g. similar to "undefined reference to `casa::ArrayColumn<float>::get(unsigned int) const`", it probably means that you did not compile Casacore with C++11 support. When you compile Casacore, you need to turn this on explicitly:

```
anoko@leopard:~/casacore/build$ cmake ../ -DCXX11="ON"
```

after which cmake should respond with a list of enabled features, including:

```
-- C++11 support ......... = ON
```

Whether it is necessary to switch on C++11 depends on the version of the compiler – with newer compilers it is turned on by default.

1.3.7 Compiling platform independently / portability

By default, cmake will create a binary that is optimized for the machine that it is compiled on, and will only work on machines that contain the same instruction set as the compiling machine. In other words, the same binary might not work on other machines, because it might use advanced instructions such as AVX instructions that might not be available on another machine. It has been reported that this can e.g. lead to an "Illegal instruction" error (see [ticket 50](tickets:#50)).

If you want to make a binary that can be used on different platforms, you can add -DPORTABLE=True to cmake:

```
cmake ../ -DPORTABLE=True
```

You should note that this makes the binary in general slower, so you should not use it unless you have to.

1.3.8 Using the LOFAR/MWA/... beam

The latest versions of WSClean require the EveryBeam package to apply the LOFAR beam and other known beams (VLA, MWA, LWA, ATCA, ...). Older versions used the LOFAR beam from the LOFAR repository or (since *version 2.6*) the "LOFARBeam" package from Github instead.

To have these primary beams available in WSClean, CMake needs to find the EveryBeam installation on your computer. If you have installed EveryBeam in a custom directory, you can add it to your search path. For example, if EveryBeam been installed to `~/Software/EveryBeam-install`, this cmake command will use it:

```
cmake ../ -DCMAKE_PREFIX_PATH="~/Software/EveryBeam-install/"
```

CMake will tell whether the LOFAR tree was found:

```
cmake ../ -DCMAKE_PREFIX_PATH="~/Software/EveryBeam-install/"
[..]
EveryBeam beam library found.
```

Extra paths can be added to e.g. also include IDG. Paths can be separated with a ; (semicolon).

1.3.9 The MWA beam

The MWA beam requires that a `.h5` file with beam coefficients is present in your path, which you can find in the MWA software tools. This file needs to be in your Python search path. WSClean will iterate over your Python search paths by executing the following Python program:

```
from __future__ import print_function
import sys
for a in sys.path:
    print(a)
```

Installation of WSClean on Red Hat-based systems

This chapter contains some instructions for Red Hat based systems.

CentOS 8

The source repository contains a Docker file scripts/docker/Centos8. This is a script to build a Docker container based on CentOS 8. These same steps can be used as a reference when building the latest WSClean on a (non-virtual) CentOS system. Other Red Hat systems should be very similar.

WSClean 2.7 on Red Hat 7.6

The text below was written by Leonardo Saavedra from NRAO.

Note: Be aware that these instructions are not for the latest WSClean version.

Some WSClean dependencies are provided by RHEL 7.6, but you have to install the latest fftw and casacore.

- http://www.fftw.org/
- https://sourceforge.net/p/wsclean
- https://github.com/casacore/casacore

Download the packages

```
mkdir ~/wsclean
cd wsclean
wget -c http://www.fftw.org/fftw-3.3.8.tar.gz
wget -c https://github.com/casacore/casacore/archive/v3.1.1.tar.gz
wget -c https://sourceforge.net/projects/wsclean/files/wsclean-2.7/wsclean-2.7.tar.bz2
```

Note: Be aware that these instructions refer to the old SourceForge version of WSClean. It is highly recommended to use newer versions from GitLab.

For this example I am going to install under `/export/local`

Install fftw

```
cd ~/wsclean
tar xzvf fftw-3.3.8.tar.gz
cd fftw-3.3.8/
./configure --prefix=/export/local  --enable-threads --enable-openmp  --enable-shared
make
make install
```

Install Casacore

```
export LD_LIBRARY_PATH=/export/local/lib:$LD_LIBRARY_PATH
cd ~/wsclean
tar xzvf v3.1.1.tar.gz
cd casacore-3.1.1/
mkdir build
cd build
cmake ../ -DCMAKE_PREFIX_PATH=/export/local/
make -j `nproc`
vim cmake_install.cmake <-- modified CMAKE_INSTALL_PREFIX
make install
```

Install WSClean 2.7

```
cd ~/wsclean
tar xvfj wsclean-2.7.tar.bz2
cd wsclean-2.7/
mkdir build
cd build/
cmake ../ -DCMAKE_PREFIX_PATH=/export/local/
make -j `nproc`
vim cmake_install.cmake <-- modified CMAKE_INSTALL_PREFIX
make install
```

Check WSClean

```
pwd
/export/local/bin
./wsclean -version

WSClean version 2.7.0 (2019-04-19)
This software package is released under the GPL version 3.
Author: André Offringa (offringa@gmail.com).
```

1.4 General usage

WSClean is a command line program. The "wsclean" executable accepts many parameters. The "wsclean" program can be run without parameters to get a list of allowed parameters together with a short description. The general syntax of wsclean is as follows:

```
wsclean [-options] <obs1.ms> [<obs2.ms> ..]
```

Here's an example of a typical run for an MWA observation:

```
# Make a Stokes I image of size 3072 x 3072 with 0.7' pixels,
# do 10000 iterations, use CS cleaning and stop when the peak
```

(continues on next page)

(continued from previous page)

```
# flux has reached 3 level.
wsclean -size 3072 3072 -scale 0.7amin -niter 10000 \
  -mgain 0.8 -auto-threshold 3 obs.ms
```

This performs a Cotton-Schwab clean. The Cotton-Schwab algorithm is enabled with the "`-mgain 0.8`" parameter, which means that the peak flux is reduced by 80% until a new major iteration is started. Multiple measurement sets can be specified on the command line to image the integration of those observations.

For fast imaging with less accuracy, you can perform a Högbom clean and disable padding:

```
# Similar to above statement, but now only Högbom cleaning
# and no padding
wsclean -size 3072 3072 -scale 0.7amin -niter 10000 \
  -auto-threshold 3 -padding 1 obs.ms
```

Because the '`mgain`' parameter was left out, WSClean will not iteratively go back to the visibilities. This also implies that the MODEL_DATA column will not be filled (see *self-calibration with WSClean*) for more info).

A description of the basic cleaning parameters is given in the *basic cleaning chapter*.

1.4.1 Multiple measurement sets

When specifying multiple measurement sets on the command line, WSClean images the combined data from all measurement sets together. For example,

```
wsclean -size 1024 1024 -scale 1asec \
  -niter 1000000 -mgain 0.8 \
  -multiscale -auto-threshold 1 -auto-mask 5 \
  lofar-100mhz.ms vla-l-band.ms vla-c-band.ms
```

will make deconvolved images from the combination of the 3 specified measurement sets.

The measurement sets may be observations at different frequencies, at different times and/or from different instruments. They are required to have the same phase centre though. To image measurement sets with different phase centres, the measurement sets should be *phase rotated to the same direction* before imaging. If the measurement sets have different pointings and it is desirable to correct for the primary beam, it is possible to *combine pointings by gridding with the beam*.

When splitting the data into so-called output channels or when selecting a subset of channels using the `-channel-range` option, the full combined bandwidth (over all measurement sets) is split / divided. See the *making image cubes* and *wideband deconvolution* chapters for more information. This is different from selecting sub-intervals using `-interval` or `-intervals-out`, for which timestep indices are determined from the first measurement set only, and the resulting timestep index selection is then applied to all the measurement sets. More information about time selections can be found in the *snapshot imaging chapter*.

1.4.2 An advanced MWA example

As a more enhanced example, here is a command to clean MWA GLEAM data:

```
wsclean -name obs-1068210256 \
  -size 4000 4000 -niter 1000000 -mgain 0.95 \
  -weight briggs -1.0 -scale 0.75amin \
  -auto-threshold 1 -auto-mask 5 -multiscale \
  -channels-out 4 -join-channels \
  -pol xx,yy -join-polarizations \
  1068210256.ms
```

The explanation of this command:

- Briggs' weighting with robustness of -1 is used. For the MWA, this decreases the noise in single snapshots slightly. See *image weighting* for more info on supported weightings.
- The (instrumental) XX and YY polarizations are imaged separately and cleaned together. This allows more accurate primary beam correction for the MWA. See *polarimetric deconvolution* for more info.
- A high `mgain` value is used because the MWA synthesized beam is well behaved.
- A larger image is made because GLEAM includes lower frequencies, at which the primary beam is larger.
- For GLEAM, the W-snapshot algorithm is used by executing the `chgcentre` command prior to imaging, as described on the *w-snapshot algorithm*.

Next chapter: *Basic cleaning*

1.5 Basic cleaning

1.5.1 Image dimensions

The image size is set in pixels with the `-size` parameter, which takes a width and a height. An image is not required to be square shaped. It is however required to use even numbers for width and height. The `-scale` parameter takes an angle and sets the angular size of a single pixel. The pixel scale currently has to be square shaped. Together, the `-size` and `-scale` parameters set the angular size of the image.

1.5.2 Threshold and maximum number of iterations

The basic required parameters for cleaning are `threshold` and `niter`. Threshold defines where to stop cleaning: WSClean will continue cleaning until the peak residual flux is below the given threshold. It is given in Jy. The `niter` parameter sets the maximum number of minor iterations that are allowed to be used.

It is good practice to make sure cleaning has reached the threshold, and only use niter to make sure wsclean will not run for an excessively long time. One should also not clean deeper than the noise, unless a mask is used. A typical stopping criterium is 3 x stddev, where stddev is the standard deviation of the noise in the image. The easiest way of setting a stopping criterium based on the noise, is by using an automatic threshold. This is discussed next.

1.5.3 Automatic threshold

To set the deconvolution threshold, one would need to know the level of the noise before running the clean. For automated processing, this is undesirable. Therefore, WSClean can automatically set a threshold based on the residual noise level. The option for this is `-auto-threshold`. With this option, WSClean will calculate the standard deviation of the residual image before the start of every major deconvolution iteration, and clean up to the given factor times the found noise standard deviation. The standard deviation is calculated using the medium absolute deviation, which is a robust estimator that is not very sensitive to source structure still present in the image. When performing *wideband* and/or *polarized deconvolution*, the RMS is measured from the integrated image. An example:

```
# Clean to a 3 sigma noise level
wsclean -auto-threshold 3 -size 2048 2048 -scale 1amin \
  -mgain 0.8 -niter 50000 observation.ms
```

Note that the `-mgain` parameter is used, in order to enable the Cotton-Schwab style major iterations. While this is not necessary for the automatic threshold to work, if the image has a very high dynamic range, the initially computed standard deviation might not be a good estimate. By using Cotton-Schwab, the standard deviation is recalculated at the beginning of every major iteration, and this will be more accurate. The `-mgain` parameter is discussed in more detail in the next section.

One can also specify both an automatic threshold and a manual threshold. In this case, whenever one of the thresholds is reached, the cleaning stops.

1.5.4 Using Cotton-Schwab: the `-mgain` parameter

The `-mgain` parameter sets the major iteration gain: during every major iteration, the peak is reduced by the given factor. With an `mgain` of 0.8, the peak is reduced by 80%. This is quite a common and safe option for cleaning. With a reasonable good PSF, using 0.9 is possible without loss of accuracy, and a bit faster. With a very bad PSF, it might be necessary to lower the `mgain` parameter.

When `-mgain` is not given, or when it is set to 1, WSClean will never go back to the visibilities. It will therefore perform a simple image-based Högbom clean. While this is fast, it limits the accuracy of the image and the dynamic range that can be reached. WSClean will also not write to the MODEL column, as would be required for self-calibration (see the chapter on *self-calibration*). To use Högbom clean and still fill the model column, an mgain of e.g. 0.9999 can be used. Alternatively, the final model *can be predicted* in a second WSClean run.

Note that the `-mgain` parameter is not the same as the `-gain` parameter. The latter sets the minor loop cleaning gain, and is 0.1 by default. It is almost never required to change the `gain` parameter.

Next chapter: *Prediction*

1.6 Prediction

WSClean can be used to fill the `MODEL_DATA` column with the visibilities corresponding to an image. This is called 'predicting' visibilities (CASA's corresponding task is 'ft' – sometimes this is referred to as 'degridding').

Predicting can be performed by adding '`-predict`' to the command line. For this to work, the input image needs to be in the exact same projection as that WSClean would output it. If your image is in a different projection, you will have to regrid it first.

An example prediction run:

```
# Predict visibilities from a Stokes I image
wsclean -predict -name my-image obs.ms
```

Be aware that the specified name (here '`-name my-image`') still specifies the prefix of the file names in the same way as that it is normally used for imaging. That means that in this run, prediction will look for an image called 'my-image-model.fits'. If you predict for multiple polarizations, e.g. with "`-pol xx,xy,yx,yy`", the files should be named accordingly, so the XX name is "`my-image-XX-model.fits`", etc. The same applies when predicting with multiple frequency intervals with '`-channels-out`'.

Note: For predicts that include a beam while gridding (facet-based or using IDG), the beam-corrected model image is read. These images have a '-pb.fits' suffix, e.g. 'my-image-model-pb.fits'.

The normal use-case for using prediction is for self-calibration or subtraction. For these cases one should use a "model" image, which contains the clean components. If the input image is in absolute flux, it might be necessary to first apply the beam to the model image.

Prediction is supported since *WSClean version 1.2*. Since *WSClean 2.1*, it is no longer necessary to manually specify the image dimensions and pixel size. It is still allowed though, in which case the specified dimensions are checked against the image dimensions.

1.6.1 Frequency information during prediction

Some care should be taken when predicting at a different frequency than the model image or when predicting from image cubes made with WSClean (described in *making image cubes*), because this works slightly different than CASA. At this point, WSClean does not look at the frequency of the FITS file, and therefore won't interpolate or extrapolate the model to the right frequency. For example, if an image is made with

```
wsclean [...] ms-at-100MHz.ms
```

and the generated model is then predicted with

```
wsclean -predict [...] ms-at-200MHz.ms
```

the model is not extrapolated to the right frequency, so one will end up with the flux levels of the 100 MHz model.

WSClean will also not look at the frequency of the FITS files when you make use of the *wide-band modes* and create multiple model images. E.g. when you image with:

```
wsclean -channels-out 4 [...] ms-at-100MHz.ms ms-at-110MHz.ms ms-at-120MHz.ms ms-at-
↪130MHz.ms
```

and predict those four models into only one MS with:

```
wsclean -channels-out 4 -predict [...] ms-at-100MHz.ms
```

this will split the channels in the 100MHz MS into four groups and predict the four channel images into these groups. So one will get different results in this MS compared to what one would get with:

```
wsclean -channels-out 4 -predict [...] ms-at-100MHz.ms ms-at-110MHz.ms ms-at-120MHz.ms␣
↪ms-at-130MHz.ms
```

Which will split the full available bandwidth into four groups and thus predict the first channel image into the first MS and so on.

1.6.2 Some MWA specifics

Applying the beam to an MWA image is a bit tricky, because the feeds are not orthogonal for anything but zenith. You can use the 'pbcorrect' tool (in my MWA repository) to apply a beam to an image. The basic syntax is:

```
pbcorrect -uncorrect <image-prefix> <image-postfix> <beam-prefix> <stokes-prefix>
```

The input are absolute Stokes images and the output are apparent flux images with linear polarizations. The `-uncorrect` parameter specifies it should do the opposite of its normal operation, as it would normally make Stokes images out of wsclean's output. For example, if the `<stokes-prefix>` is "stokes", then `pbcorrect` will look for stokes-I.fits, stokes-Q.fits, stokes-U.fits and stokes-V.fits. If any of these is not present, it will be assumed zero (and a warning is issued). The 'beam' files are 8 files containing all real/imaginary components for the four linear polarizations. These can be created with the 'beam' tool in my MWA repository.

If you use `pbcorrect` to prepare an image for wsclean prediction, you should set `<image-postfix>` to "model.fits", and the image prefix is the same prefix you will specify to wsclean.

Next chapter: *WSClean and self-cal*

1.7 Self-calibration

WSClean can be used to perform self-cal. There are two main approaches, supporting a range of scenarios:

- Use WSClean to fill the `MODEL_DATA` with predicted visibilities from an imaging run. When using `-mgain` with a value less than 1, WSClean will fill the column. This column can then be used during calibration (e.g. with DP3). The advantage of this is that it is relatively fast and easy.
- Make WSClean output a source component list (see *Component lists*) and use this list during the calibration (DP3 supports this format). The advantage of this approach is that it is easier to apply the beam (and other effects) on a source list, it is very accurate, and it is possible to prune/edit the source list. A disadvantage can be that if the number of sources is very large, calibration will be very slow.

1.7.1 Image-based self-calibration

One scenario where image-based self-cal can be useful is when combined with the ability to do (almost-)full-sky imaging, since in some situations MWA's field of view (for example) might require including highly off-axis sources in the calibration model.

To perform self-cal, you need to use major iterations, as these will fill/update the `MODEL_DATA` column. When using Casa's tasks for calibration, the calibration tasks will use the `MODEL_DATA` and calibrate the `CORRECTED_DATA` using this column. Other calibration utilities, like MWA's `mitchcal` or LOFAR's `DP3` can similarly be instructed to calibrate using the `MODEL_DATA` column.

Filling the `MODEL_DATA` requires a setting of mgain < 1, e.g. an mgain of 0.9. As long as mgain is not 1, WSClean will end with a major iteration, and the `MODEL_DATA` column will be set to the "best" model from the cleaning model.

1.7.2 Self-calibration from existing image

The '`-predict`' option can be used to fill the `MODEL_DATA` column with a prediction from a pre-existing image (see *prediction*). After having predicted model visibilities, these visibilities can be used to calibrate the data (e.g. with DP3).

1.7.3 Polarized imaging & calibration

You can self-cal on Stokes I or on multiple polarizations, where the latter is more accurate (at least in the case of the MWA), but takes more time. If you run WSClean on the desired polarizations one by one, e.g. on XX and then on YY, or joinedly clean them, the `MODEL_DATA` column will have all imaged polarizations correctly filled in. The first run will create the `MODEL_DATA` column, set all values to zero and then fill the XX column, the second run will notice the `MODEL_DATA` already exists, and only update the YY column. B. McKinley has used this method and did a few self-cal loops to create very deep and well-calibrated Fornax A images. You can run all polarizations at once with `-pol xx,xy,yx,yy`, `-pol iquv`, or `-pol rr,rl,lr,ll`. For info on polarimetric deconvolution settings, see *polarimetry deconvolution*.

1.7.4 CASA on-the-fly mode

Certain CASA commands (e.g. ft) will put keywords in a measurement set that turn on the "on-the-fly" (otf) mode. In OTF mode, CASA will ignore the `MODEL_DATA` column and use other keywords to determine the model data. To make use of the `MODEL_DATA` afterwards, you can use the delmod CASA command to disable OTF mode:

```
delmod(vis='myobs.ms',otf=True,scr=False)
```

WSClean will never use or change OTF keywords in the measurement set.

Next chapter: *Image weighting*

CHAPTER

TWO

WEIGHTING

Contents:

2.1 Image weighting

WSClean supports uniform, natural and Briggs' weighting. These work exactly like in Casa and other imagers, and are further described below.

2.1.1 Uniform weighting

Uniform weighting will give each scale the same weight. If multiple visibilities fall in one uv cell, they are down-weighted by the number of visibilities in that cell. In this weighting mode, the image will have the highest resolution, but the system noise will not be optimal. This gridmode is the default, but can be explicitly selected with `-weight uniform`.

2.1.2 Natural weighting

In natural weighting, the visibilities are not weighted before gridding, causing scales that have more baselines to dominate the image. Arrays have more small baselines, which will mean that large scales will be more dominantly present. This mode will make sure that the system noise has the least effect in the image, but resolution and sidelobe noise are worse. This mode is activated with option `-weight natural`.

2.1.3 Briggs' weighting

Briggs' weighting is a compromise between uniform and natural weighting. A parameter, called the robustness parameter, selects the desired level between uniform and natural. Typical values are between -1 and 1. Lower values give more uniform-like weights, higher values give more natural-like weights. Note that values of -1 / 1 do not *equal* uniform / natural weighting. Briggs weighting mode is selected with `-weight briggs <robustness>`, for example `-weight briggs 0.5`.

2.1.4 Super-uniform and super-Briggs weighting

In some cases, slightly better results can be obtained by using super weighting. Super weighting refers to performing the counting of visibilities on larger grid cells, such that neighbouring uv-samples share their weight. This can be selected with `-super-weight <value>`. The value represents how much larger the weight-counting cell should be. CASA's super uniform weighting mode is equivalent to `-super-weight 3 -weight uniform`. Additionally, it is also possible to perform sub-uniform or sub-Briggs weighting, by giving a value less than one. The super-weighting can also be used in combination with Briggs' weighting. Super-weighting was implemented because many astronomers are familiar with the "super uniform" weighting option in CASA. In many cases, the weighting rank filter described below is a better option.

Super-pixel weighting can be nice on very large (full-sky) images, since the immense uv resolution might otherwise start to make your beam become natural-ish weighted. Dan Briggs' thesis describes sub/super-pixel weighting very well.

2.1.5 Weighting rank filter

The weighting rank filter works in combination with one of the other weighting modes, and suppresses visibilities that would receive a much larger weight than its uv-neighbouring visibilities. This is basically done by smoothing the weights – see the *weight rank filter page* for a detailed description. In uniform weighting, some visibilities can receive order of magnitudes more weight, causing these visibilities to dominate the image noise. By smoothing the weights, this issue is mitigated. The weighting rank filter does the smoothing in such a way that the PSF is not visibly affected.

2.1.6 Relation to visibility weights

In WSClean, image weighting normally works 'on top of' the visibility weights stored in the measurement set (the `WEIGHT_SPECTRUM` column). This in particular means that uniform weighting will weight all scales (baselines) the same independent of their visibility weight. One therefore cannot increase the weight on long baselines by using uniform weighting and increasing the visibility weights of those baselines. The visibility weights nevertheless still have a function: they determine the weight in their particular uv-cell with respect to other visibilities *in that same uv-cell*. All in all, this means that WSClean assumes that the visibility weights specify the inverse variance weight of the visibility. Up or downweighting of scales has to be done by the image weighting (which can include *tapering*). I've had reports that this behaviour is not consistent with how CASA treats the combination of visibility weights and imaging weights.

This behaviour can be altered by using the `-use-weights-as-taper` option. When specified, imaging weights will be determined without taking the visibility weights into account. Uniform weighting, for example, will make the visibilities have uniform weights based on the visibility *count* per uv-cell, without accounting for the visibility weights. The visibility weights are applied afterwards. This mode allows the visibility weights to be used to increase/decrease the weights of certain scales, just like an image taper.

An example: consider three visibilities, A, B and C. Visibilities A and B fall into the same uv-cell, and visibility C falls in a different UV cell. Assume the visibility weights of A, B and C are given by 1, 10 and 100. With normal uniform weighting, this implies that:

- Visibilities A and B are added together with weights, giving visibility B 10x more weight than visibility A.
- Visibility C is in a different cell, and since there are no other visibilities in this cell, its weight of 100 is irrelevant. In the image, the uv-cell of A and B receives the same weight as the uv-cell of C, as this is what uniform imaging weights implies.

Even though visibility C has a 10x higher weight than B, it will not be upweighted in the final image. Now, if `-use-weights-as-taper` would be specified:

- Again, visibilities A and B are added together with weights, giving visibility B 10x more weight than visibility A.

- The UV cell of A and B receives a weight of 11, whereas the UV cell of C receives a weight of 100.

2.1.7 Other weighting-related settings

- *MF weighting* – If the bandwidth is split in multiple subbands for imaging (e.g. with `-channels-out`), it is good to know about MF weighting.
- *tapers* – can be used to further shape the synthesized beam.
- *The weight rank filter* – used to decrease noise by avoiding giving too much weight to a single visibility.
- *Continuing a deconvolution* – can be used to deconvolve a model with one weighting setting, and image the final result with different weighting settings.

Next chapter: *Tapering*

2.2 Tapering

Tapering can be used to shape the synthesized beam, by tapering off the weight of visibilities in the uv plane. This page lists the tapering options supported by WSClean.

Tapering works cumulatively with weights set by the imaging weights (uniform, Briggs, natural, see *the image weighting chapter*) and the visibility weight. Tapering is available in WSClean since *version 1.10*.

2.2.1 Gaussian taper

A Gaussian taper multiplies the uv-weights with a Gaussian function, which therefore makes the synthesized beam approach a Gaussian function. A Gaussian taper is selected with *-taper-gaussian <beamsize>*. The beamsize is by default in arcseconds, but can be given with different units, e.g. "2amin". A Gaussian taper in uv space will be calculated such that the FWHM of the Gaussian in real space has the given beam size.

2.2.2 Tukey taper

Tukey tapers can be used to smooth edges with a *Tukey window*, also known as a *tapered cosine window*. WSClean allows tapering three edges:

- `-taper-tukey`, a circular taper that smooths the outer edge set by `-maxuv-l`
- `-taper-inner-tukey`, a circular taper that smooths the inner edge set by `-minuv-l`
- `-taper-edge-tukey`, a square-shaped taper that smooths the edges as set by the uv grid and `-taper-edge`.

The following plot shows how the Tukey-taper parameters influence the weights:

Next chapter: *MF image weighting*

2.3 Multi-frequency weighting

WSClean has a special weighting mode called 'MF weighting', which is turned on with the '`-mf-weighting`' or '`-join-channels`' parameter. This is a weighting mode that changes the way in which spectral imaging (i.e., making image cubes) is performed. MF weighting is therefore only relevant when using a '`-channels-out`' parameter larger than 1. This parameter is explained on the manual page *Making image cubes* page.

MF weighting improves the frequency-integrated (multi-frequency, "MF") images. If the individual (non-integrated) spectral images are to be used for science, **MF weighting is discouraged**. This is because MF weighting adjusts the weight of individual frequencies to make the integrated image look best. This however can create artificial spectral structures when the individual frequency images are inspected / used in science. It is fine to use `-join-channels` for those science cases, but make sure MF weighting is turned off with `-no-mf-weighting`.

If MF weighting is turned on, the weights of all imaged channels will be gridded on one grid. When turned off, the weights of each imaged channel are gridded on a separate grid.

Turning MF weighting on can be useful, for example, to get uniformly weighted image cubes with the MWA. When not using MF weighting, a single MWA channel creates such a fine track in uv space, that different tracks almost never share the same uv pixel, and hence all the weights tend to be closer to '1' in non-MF, but uniform, weighting mode. The effect is that each image is weighted more towards natural weighting compared to MFS imaging. In this case, when all images are summed later on, the integrated image does not look like a uniform image, because each channel has been weighted separately.

So, this is resolved by using MF weighting. In MF weighting, all weights are gridded on the same grid, and hence the sum of the image cube equals a MF image with the same weighting.

Note that MF weighting is off by default, but using the '`-join-channels`' automatically enables MFS weighting.

Example of using MF weighting to make a briggs -1 image cube:

```
wsclean -channels-out 768 -mf-weighting -weight briggs -1 myobservation.ms
```

A related option is the *weight rank filter* that is enabled with the '`-weighting-rank-filter`' option. The weight rank filter prevents outlier weights that might increase the noise of the image.

Next chapter: *Weight rank filter*

2.4 Weight rank filter

The weight rank filter can be used to decrease noise by avoiding giving too much weight to a single visibility.

To determine the *image weights*, the weights are gridded on a regular grid. With uniform or Briggs' weighting, the total weight of a visibility is determined by the number of visibilities that fall into that cell: the fewer visibilities fall into the same cell, the more weight this visilibity receives. Because the visibilities form a track in uv-space, it can happen that a particular visibility is the only visibility gridded in a particular uv-cell, while other uv-cells have on average hundred, thousands, ... of visibilities in them. This implies that this single visibility will receive a lot of weight.

The weight rank filter in WSClean avoids this by calculating a local weight RMS of the grid, and truncating all weights that are some factor above the local RMS. This effectively decreases the weight of those visibilities that would receive excessive weight otherwise. This results in very little change in the PSF, but can effect the noise in the image significantly.

2.4.1 Effect & improvement

The exact effect of this option changes very much on the type of observation, image size and uv-coverage. I've seen improvements of 20-30% reduction in noise in certain VLA observations, while I hardly saw any improvement in certain LOFAR and MWA observation.

To observe the effect of this filter, it can be useful to save the uv-weights with `-save-weights`, and perform an imaging run with and without the filter.

2.4.2 Syntax

The parameter to enable the weight rank filter is `-weighting-rank-filter`, which takes the factor relative to the RMS above which the values are truncated. For example:

```
wsclean -weighting-rank-filter 3 [other options] observation.ms
```

The default size over which the RMS is calculated, is 16 uv-pixels. During this calculation, only uv-cells with coverage are used in the calculation. If the visibility coverage is very sparse, the size of 16 uv-pixels might be too small. It can be changed with the `-weighting-rank-filter-size` option, for example:

```
wsclean -weighting-rank-filter 3 -weighting-rank-filter-size 64 \
    [other options] observation.ms
```

Since *WSClean 2.2*, the default in WSClean is to apply a weighting rank filter with a value of 3. It is recommended to always apply a weak rank filter, because it improves the noise with hardly any change on the PSF. In case there are no visibilities with excessive weight, this filter will do nothing.

The weight rank filter is available since *WSClean 1.8*. In versions before *WSClean 2.2*, the default setting was to not apply a weight rank filter, and since 2.2 it was turned on with a value of 3. As far as I'm aware, there's no weight rank filter option in CASA.

2.4.3 Related topics

- *Image weighting*, which discusses natural, uniform and Briggs' weighting, super-uniform weighting, etc.

CHAPTER

THREE

IMAGING MODES

3.1 Making image cubes

WSClean allows making image cubes, but it works slightly different compared to CASA. The following section describes how it works in WSClean.

To make image cubes with WSClean, the '`-channels-out`' parameter is used. This parameter specifies the number of different frequencies for which information will be present in the output.

Assume we have a measurement set 'myobservation.ms' with 100 channels. To image each channel seperately, '-channels-out 100' is added to the command line, for example:

```
wsclean -scale 1amin -size 3072 3072 -niter 1000 -threshold 1 \
  -channels-out 100 myobservation.ms
```

This will output 100 images, named wsclean-0000-image.fits, wsclean-0001-image.fits, ..., wsclean-0099-image.fits (and similarly for the PSF, residual and dirty). Additionally, images named like wsclean-MFS-image.fits will be outputted, which are the weighted sum over all 100 images.

WSClean does not output these images in a normal "imaging cube" like CASA does, i.e., a single fits file with several images in it. For now I've decided not to implement this (one of the reasons for this is that information about the synthesized beam is not properly stored in a multi-frequency fits file). One has of course the option to combine the output manually, e.g. with a simple Python script.

When you are interested in only a partial range of channels, the '`-channel-range`' parameter can be used to select that range. For example:

```
wsclean -scale 1amin -size 3072 3072 -niter 1000 -threshold 1 \
  -channel-range 60 70 -channels-out 10 myobservation.ms
```

This would image channels 60 to 70 and output separate images for each channel.

The channelsout parameter can be smaller than the available number of input channels. For example, a measurement set with 100 channels can be imaged with "`-channels-out 2`", which will divide the bandwidth in 2 and output 2 images. Furthermore, you can also specify multiple measurement sets at different frequencies. In such a case, the total bandwidth will be divided into the requested number of channels. For example:

Example:

```
wsclean -channels-out 3 -niter 10000 -mgain 0.8 -threshold 0.01 \
  band-100-MHz.ms band-110-MHz.ms band-145-MHz.ms \
  band-155-MHz band-190-MHz.ms band-200-MHz
```

This will output images at 105 MHz, 150 MHz and 195 MHz.

Some more notes:

- The '`-channels-out`' parameter can be used in combination with '`-join-channels`' to improve the deconvolution of MFS imaging in wideband scenarios, or to get better SNR during the deconvolution of spectral imaging. This is explained on the *Wideband deconvolution page*.
- WSClean normally weights each output channel separately. This is equivalent to how other imagers do this. However, this is not always desired, especially not with modern correlators with large number of channels. Therefore, a special weighting mode called 'MF weighting' has been implemented in WSClean to improve weighting of spectral cubes. This is further explained on the *MF weighting page*.

3.2 Snapshot imaging

WSClean accepts a convenience parameter to make it easy to separate a measurement set in small snapshots and image each one separately. The parameter to do this is:

```
-intervals-out <count>
```

This tells WSClean to split the measurement set in the given number of intervals, and image each interval separately. The file name of all the image products of an interval will have the interval number preceded by a '`t`'. For example:

```
wsclean -intervals-out 100 -scale 1asec -mgain 0.8 -threshold 1 \
  -niter 10000 -name cyga MyCygASet.ms
```

This will output images `cyga-t0000-dirty.fits`, `cyga-t0000-image.fits`, and all the other products for t0000, and so for t0001…t0099.

To image only part of the full time span, the `-interval` parameter can be used to select the range of timesteps that is to be imaged. So, with "`-intervals-out 100 -interval 200 300`", the first image will be made from timestep 200, the next one from timestep 201 and further until timestep 299. Units are in "number of timesteps" (so the integration time).

Note that the selection of timesteps with `-interval` is done before splitting into snapshots: if a measurement set has 10 timesteps, specifying `-intervals-out 10 -interval 2 4` is invalid, as it would first select the interval from timestep 2 to 4, and then split those 2 timesteps into 10 intervals. This is independent of the order in which these parameters are given: it does not matter whether `-interval` or `-intervals-out` is first on the command line.

You can use all other *cleaning parameters* together with `-intervals-out` (`-mgain`, `-auto-threshold`, `-niter`… etc.).

3.3 *w*-snapshot algorithm

WSClean provides an algorithm that has approximately the same performance as the W-snapshot algorithm suggested by Cornwell et al. (2012), although the algorithm in WSClean is slightly different. In WSClean, it consists of phase rotating the visibilities to zenith, and then recentring the image during w-stacking. Mathematical details are explained in the WSClean paper. It seems to be worthwhile for MWA snapshots of a few minutes with images of 3072 x 3072 pixels at zenith angles > 20 degree, and provides a speed-up of about a factor of 3 at zenith angles > 45 degrees. For larger images the speed-up will be greater. It is not useful for long integrations.

To use this method, the measurement set should be phase rotated to the phase centre that you want to image (as it normally would be). This chapter assumes this is `08h20m00.0s -42d45m00s` (which is Puppis A). To prepare for the *w*-snapshotting approach, first run *the chgcentre tool* with the following parameters:

```
chgcentre -minw copy.ms
```

This will calculate the optimal projection direction and perform the required phase rotations. Finally, during imaging the phases should be *shifted* to the original target, e.g.:

```
wsclean -shift 08h20m00.0s -42d45m00s ...
```

The `chgcentre` command *rotates* the coordinate system whereas WSClean's `-shift` parameter *shifts* the coordinate system along the tangent plane. The result is that imaging is done in a different projection that is still centered on the target location, but that has smaller *w*-terms. The *chgcentre page* provides further information about the `chgcentre` tool.

The `-shift` parameter is new in *WSClean 3.0*. Before 3.0, this could be achieved by using the `-shiftback` option of `chgcentre`. That approach is no longer supported in WSClean.

The *idg*, *w-gridding* gridder and *w*-stacking gridders all support this approach. Support for the *w-gridding* gridder was added in WSClean 3.0.

3.4 chgcentre

The 'chgcentre' tool can be used to change the phase centre of a measurement set. It will recalculate the *uvw*-values (from the antenna locations, phase centre and time) and phase-rotate the visibilities. We found that the casa task 'fixvis' has a bug (as of March 2014) that causes it to malfunction for some arrays (e.g. MWA, LOFAR).

Since *wsclean 2.10*, chgcentre is compiled and installed when wsclean is installed. See the *general installation instruction* for help.

Execute chgcentre without parameters to get help on the syntax:

```
A program to change the phase centre of a measurement set.
Written by André Offringa (offringa@gmail.com).

Syntax: chgcentre [options] <ms> <new ra> <new dec>

The format of RA can either be 00h00m00.0s or 00:00:00.0
The format of Dec can either be 00d00m00.0s or 00.00.00.0

Example to rotate to HydA:
        chgcentre myset.ms 09h18m05.8s -12d05m44s

Some options:
-geozenith
        Will calculate the RA,dec of zenith for each timestep, and moves there. This␣
↪make the set non-standard.
-flipuvwsign
        Flips the UVW sign. Necessary for LOFAR, for unknown reasons.
-minw
        Calculate the direction that gives the minimum w-values for the array.
-zenith
        Shift to the average zenith value.
-only-uvw
        Only update UVW values, do not apply the phase shift.
-shiftback
```

(continues on next page)

(continued from previous page)

```
        After changing the phase centre, project the visibilities back to the old phase␣
↪centre. This is useful
        in WSClean for imaging with minimum w-values in a different projection.
-f
        Force recalculation, even if destination is same as original phase direction.
-datacolumn <name>
        Only phase-rotate the visibilities in the given column. Otherwise, the columns
        DATA, MODEL_DATA and CORRECTED_DATA will all be processed if they exist.
-from-ms <ms>
        Rotate the measurement set to the same direction as specified
        in the provided measurement set.
```

When a measurement set contains multiple data columns (e.g., `DATA`, `MODEL_DATA`, `CORRECTED_DATA`), each column will be updated (as long as they have a standard name).

If you do not provide a new RA and dec, chgcentre will give you some info about the measurement set:

```
$ chgcentre myobs.ms
A program to change the phase centre of a measurement set.
Written by André Offringa (offringa@gmail.com).

Current phase direction:
  -00h59m31.7s -16d46m19s
Zenith is at:
  -01h00m00.8s -26d46m44s
  (-01h00m49.0s -26d46m44s - -00h59m12.7s -26d46m44s)
Min-w direction is at:
  -00h59m31.7s -26d46m19s
```

You can specify '`-zenith`' or '`-minw`' as option to rephase to the local array zenith or the direction orthogonal to the best-fit plane to the antennas. The latter is close to zenith, but provides slightly lower *w*-terms. This has not been tested on telescopes other than the MWA. The syntax for this is:

```
$ chgcentre -minw myobs.ms
```

This can be used in combination with WSClean's `-shift` parameter for *w*-snapshot imaging. In that case, the original phase centre should be specified with the `-shift` parameter. The net effect is that the measurement set is phase rotated to the sky direction with minimal *w*-terms, and shifted back along the tangent plane to the direction of interest.

In *WSClean 3.0*, this approach replaced the `-shiftback` option of `chgcentre` for shifting the visibilities along the tangent plane. See the *w-snapshot algorithm page* for more info.

3.4.1 Legacy data with `-shiftback` applied

Before WSClean 3.0, it was possible to prepare w-snapshotting with the `-shiftback` option, e.g.:

```
$ chgcentre -minw -shiftback myobs.ms
```

A shifted measurement set uses wsclean-specific keywords. Support for this was removed in WSClean 3.0 (see the *changelog* for details), and any observation for which these shifting keywords are detected will produce an error in WSClean 3.0.

In case archival data to which `-shiftback` is applied needs to be imaged with WSClean 3.0, the option should be undone. A shifted measurement set can be restored by phase rotating it to its original RA/dec: `chgcentre` will detect

the keywords in the measurement set, undo the shift and update the keywords.

3.4.2 A LOFAR bug

For unknown reasons, the uvw value needs to be flipped for LOFAR sets. As far as I know, this is not necessary for other telescopes, but LOFAR requires you to specify `-flipuvwsign`.

3.5 Baseline-dependent averaging

This page explains baseline-dependent averaging, a feature that can increase the performance of gridding. This feature is available in WSClean since *version 2.0*.

Baseline-dependent averaging is a technique that allows more efficient averaging of data. In normal averaging, long baseline are averaged at the same level as small baselines. However, since small baselines can be averaged much more, this is inefficient. In baseline-dependent averaging, short baselines are averaged more than the long baselines.

By decreasing the number of visibilities, the imaging will become faster. By chosing the proper averaging factor, this can be quite significant with negligible effects on the image quality. The total time it will save is dependent on many factors ; the image size, the size of the w-terms, the number of visibilities, etc. If the gridding is the dominant cost, then baseline-dependent averaging is probably a significant improvement. In cases where the image size is very large and/or the w-terms are very large, it might be that the Fourier transforms are the dominant cost, instead of the gridding, and the savings might be less.

In principle, the baseline-dependent averaging can be done both in time and in frequency. However, WSClean can only do this in the time direction so far.

3.5.1 Baseline-dependent averaging in WSClean

WSClean can perform baseline-dependent time averaging during its "reordering" step. In this mode, it reads in a normal measurement set and averages into an internal format. The baseline-dependent averaging happens thus all internally, and other programs don't handle the averaged data. There's currently no good definition for how to store baseline-averaged data in Measurement Sets, so one cannot e.g. perform a self-calibration loop that images the baseline-averaged data, and then calibrate the baseline-averaged data.

3.5.2 Setting the averaging factor

Baseline-dependent average is enabled with the option "`-baseline-averaging <nwavelengths>`", which takes a parameter that is the number of wavelengths it is allowed to average over. This value can be related to the averaging factor as follows:

nwavelengths = max baseline in nwavelengths * 2pi * integration time in seconds / (24*60*60)

E.g. if a baseline of 20 kLambda can be averaged to 12 seconds without decorrelation, a corresponding baseline-dependent averaging factor can be calculated with:

20000 * 2pi * 12 / (24*60*60) = 17 wavelengths.

Using `-baseline-averaging 17` would therefore average any baseline of 20 kLambda to 12 seconds. Smaller baseline will be averaged more, and longer baselines will be averaged less.

Of course, the number of wavelengths to average over can also be calculated from first principles by determining how much phase-change in one averaging cell is acceptable.

For LOFAR, we've seen speed ups of a factor of 4-8 and for MWA data likewise.

3.5.3 Further options

Cleaning and major iterations etc. are all possible like normal.

Currently, option '`-baseline-averaging`' requires option '`-no-update-model-required`', because no unaverager (extrapolating) method to write the model data back at full resolution is available. This means this option can not be trivally used inside a *self-cal loop* where the model data is required. This can be overcome by performing the imaging with baseline averaging, and *predicting* the final model image back into the measurement set with a separate WSClean run, that excludes baseline averaging.

This is a simple command to show the syntax of baseline-dependent averaging:

```
wsclean -size 1024 1024 -scale 5asec -mgain 0.8 \
   -niter 10000 -threshold 0.002 \
   -baseline-averaging 16 -no-update-model-required \
   -maxuvw-m 100000 observation.ms
```

Note that baseline-dependent averaging currently only works together with *primary-beam correction* since *version 2.5*.

3.5.4 Baseline-dependent averaging in DP3

Since *version 3.1* WSClean can read baseline-averaged data generated by DP3's BDAAverager. WSClean will automatically detect whether the data have been averaged by DP3. When using these data there are some restrictions.

- The data may only be averaged in the time direction.
- The following command line options are not supported:
  - `-baseline-averaging`
  - `-even-timesteps`
  - `-interval`
  - `-odd-timesteps`
  - `-simulate-baseline-noise`
  - `-simulate-noise`

3.5.5 Some LOFAR specifics

If a measurement set includes the international baselines, it can be helpful to remove these by a row-selection maximum uvw-m value. If one doesn't do this, the reordered file can still be very large because the international baselines won't be averaged at all. The `-maxuvw-m` option works before reordering (whereas `-maxuv-l` does not), and can thus remove those. Hence, adding a maxuvw-m that is just smaller than the first international (or unused) baseline can improve the speed. This even holds when those baselines are flagged.

Of course, even better is to explicitly remove the international stations from the measurement set when they are not used, e.g. using DP3.

3.6 Continuing a deconvolution

WSClean uses the option '`-continue`' to continue a previous deconvolution run, and `-reuse-psf` and `-reuse-dirty` to skip imaging of those two and directly clean existing images.

There are multiple scenarios why continuing can be useful, for example:

- To finish a previously unfinished deconvolution run when the initial deconvolution was prematurely stopped because of the stopping criteria (*not* because of a crash).
- To use different imaging settings at different stages; e.g. clean up to 3 sigma, then construct a mask manually and continue a deeper deconvolution with a mask. (note that *auto-masking* is often a better way to do this)
- If you want different image weighting settings with the same deconvolution. For example, one can run the deconvolution with uniform weights, and then reimage the set with the same deconvolution results but with different weighting.

There might be other use-cases for `-continue`. However, not that what `-continue` does is a bit complex, so you should understand the specific steps that `-continue` performs to make effective use of it.

3.6.1 Syntax

The syntax to continue a run is to simply add '`-continue`' to the command line:

First run:

```
wsclean -size 1024 1024 -scale 1amin -niter 1000 \
  -auto-threshold 10 -mgain 0.8                  \
  observations.ms
```

Continued run:

```
wsclean -continue -size 1024 1024 -scale 1amin -niter 1000 \
  -auto-threshold 3 -mgain 0.8                             \
  observations.ms
```

There are few considerations:

- The constructed model visibilities need to be stored in the measurement set. This implies that `-mgain` should be used (see the *Selfcal instructions* for more info on mgain) in the first run, or you have to manually predict the model image from the first run, before continuing. WSClean does not verify whether this assumption holds.
- As a result, it is not directly possible to use `-no-update-model-required` in the first run, because the second run requires the `MODEL_DATA` column to be filled. If `-no-update-model-required` was enabled, or only a model image without the corresponding predicted visibilities is available, it is still possible to continue the run by first *predicting* the model data (using `wsclean -predict ...`) from the model image before continuing.
- In a continued run, WSClean expects the model image to be there. It also expects it to have its normal name, i.e. something like *prefix-model.fits*. For predicts that include a beam while gridding (facet-based or using IDG), the beam-corrected model image is read, with a name such as *prefix-model-pb.fits*.
- One cannot change the image dimensions, phase center or pixel scale between the first and a continued run.
- The second run can not include more visibilities than the first run. E.g., if selection parameters such as `-maxuv-l` are used, the first run should not be more restrictive than the second run, because the `MODEL_DATA` column would not be set for those visibilities.

This option is available since *WSClean version 2.0*.

3.6.2 Example: change weighting

A common situation for using continued imaging is to image diffuse structure in the presence of point sources. In the first run, the point sources are subtracted, while in the second run, the weighting is changed to focus on the diffuse structure.

The following sequence of statements would do this:

```
# Run with uniform weighting to subtract point sources:
wsclean -size 2048 2048 -scale 1amin      \
  -niter 10000 -threshold 0.1 -mgain 0.8  \
  -weight uniform                         \
  -name diffuse-field observation.ms

# Since the images will be overwritten in the second run,
# here I copy the images so they can be inspected for
# debugging purposes.
cp diffuse-field-model.fits uniform-diffuse-field-model.fits
cp diffuse-field-image.fits uniform-diffuse-sources-image.fits

# Finally, the run is continued with different weighting
# settings to highlight the residual diffuse structure.
wsclean -size 2048 2048 -scale 1amin      \
  -continue                               \
  -weight natural -taper-gaussian 3amin   \
  -name diffuse-field observations.ms
```

See the *chapter on weighting* for more info on weighting.

3.6.3 What `-continue` really does

By adding `-continue` to the command line, WSClean will do the following things:

- The previous model image will be read.
- During the first inversion, WSClean will image the PSF (even thought it might already exist – since e.g. the weights might have been changed).
- During the first inversion, WSClean will immediately image the residual data (data - model data). This image is still named 'dirty image', even though it is actually the residual image of the first deconvolution.
- Any new components found during cleaning will be added to the previous model image.
- The previous model image will be overwritten.
- The final image will be restored with the full model.

The implication of this strategy, is that if you continue a first run with a second run with the same settings, both using `-niter N`, the resulting restored and model images are the same as when you would have run the full deconvolution with `-niter 2N` at once. However, during a continued run the PSF and residual image will be reimaged (to allow changing the weighting), so using the two runs will be slower. The bottom line is that one should only use `-continue` if there are good reasons for it.

3.6.4 Difference with `-subtract-model`

The `-subtract-model` option only makes WSClean subtract the model column from the data column during the first imaging iteration. The useful use-case for this is to make it directly image the residual without any extra cleaning. Any previous sources won't be restored. Also, if you enable cleaning something rather confusing happens: during the first iteration, the "current" residual is imaged. This image is cleaned and a new model is formed with a few residual sources. The models of the residual sources is written to the MODEL_DATA column, and in the next iteration, the new MODEL_DATA is subtracted from the DATA, making all sources that were previously subtracted appear. The bottom line is that this option should only be used to quickly reimage a residual image without extra cleaning. If you do want cleaning, you have to manually subtract the MODEL_DATA from the DATA, which is easy e.g. with this Taql statement:

```
taql update obs.ms set DATA=DATA-MODEL_DATA
```

3.6.5 Reusing PSF / dirty image

Existing PSF or dirty images can be reused to:

- run the deconvolution algorithms of wsclean without doing the inversion (i.e. without ever going back to the visibilities)
- speed up a second run of imaging when the PSF/dirty already exist, and no change in imaging settings (pixel scale, size, weighting, etc.) is made.

A first "regular" run to make the PSF and dirty image:

```
wsclean -make-psf -size 1024 1024 -scale 30asec -channels-out 4 \
  obs.ms
```

A run that reuses the PSF and dirty images from the previous run:

```
wsclean -reuse-psf wsclean -reuse-dirty wsclean \
  -no-reorder -name secondrun \
  -size 1024 1024 -scale 30asec -channels-out 4 \
  -niter 1000 -auto-threshold 5 obs.ms
```

The use of '`-no-reorder`' skips the reordering of visibilities by wsclean, which is useful when wsclean would never go back to the visibilities, as then the reordering is just overhead. It is allowed to keep the name between the runs the same (so to remove '`-name secondrun`' from the second run).

3.7 Distributed imaging

Multiple nodes can be used to speed up the imaging by running wsclean in distributed mode, which makes use of MPI to parallelize the imaging over multiple nodes. To use this, you need a recent WSClean version that provides the executable `wsclean-mp`.

A typical run:

```
mpirun --hostfile host_file -np 8 wsclean-mp -size 10000 10000 ...
```

And a host-file could look like this:

```
node100 slots=1
node116 slots=1
node118 slots=1
node119 slots=1
node122 slots=1
node123 slots=1
node124 slots=1
node125 slots=1
node126 slots=1
node127 slots=1
node128 slots=1
node129 slots=1
node130 slots=1
```

The host file should specify one slot per host, otherwise multiple wsclean's are executed on the same host, and that has (normally) no benefit, and would actually make those processes compete for memory and cpu. If you use wsclean-mp, all paths should be absolute for the hosts participating, and the paths should be reachable by all nodes (so `-name` should specify an absolute path name, and the input files should have an absolute path name).

`wsclean-mp` will distribute the different channels to different nodes. This implies that if you don't use `-channels-out` there's no benefit, whereas using `-channels-out 8` with `-np 8` gives you a speed-up of 8. If multiple output channels are not necessary for your science goal, one can use `-fit-spectral-pol 1 -deconvolution-channels 1`.

3.8 Primary beam correction

WSClean can calculate and apply the primary beams of several instruments (LOFAR and MWA since *version 2.7*, and later versions supported more). It uses the EveryBeam library for this. There are two ways to correct the beam: during gridding (using IDG) and as an image plane correction. This page discusses image plane correction, whereas beam correction during gridding is discussed on the following page: *Gridding with the beam using IDG*. Image plane correction is faster and less complex, but will not correct for time variation of the primary beam, and won't weight the data optimally. Beam correction during gridding is therefore superior to image plane correction, but is not always required to get good results.

Image-based primary beam correction is enabled with the option `-apply-primary-beam`. Primary beam correction can either be applied on Stokes I or on the four Stokes polarizations. In Stokes I-only mode, it is assumed that the field does not contain polarized flux (i.e., only a scalar correction is performed). For image-based correction, the beam's Jones matrix are summed as Mueller matrices. These are then inverted and applied:

$$\text{corrected} = B^{-1}\ \text{vec}(V)$$

Where B is the complex 4x4 Mueller matrix of the average beam, and V is the visibility matrix.

3.8.1 Supported telescopes

Primary beams are calculated using the EveryBeam library. This library, which was originally part of WSClean, supports numerous telescopes. See the EveryBeam README for a list of supported instruments. The EveryBeam will automatically determine what instrument is associated with the measurement set, and use the right beam (if available).

3.8.2 Beam sensitivity limit

By default, WSClean will clip the primary-beam-corrected image for which the response is smaller than 0.5%. This can be modified with the `-primary-beam-limit` parameter.

The reason for clearing pixels with low response is that the primary beam may have nulls or almost-nulls in them, and because the image is divided by those value, they may cause numerical issues. The non-primary-beam-corrected images will not be clipped, and can be inspected to analyze the full imaged area. The clipping can be turned off by setting `-primary-beam-limit 0`.

3.8.3 LOFAR specifics

It is common in LOFAR observations to correct the visibilities for the beam at the phase-centre. WSClean can also correct for this. This is referred to as "differential beam correction", while the normal situation is referred to as "full beam correction". Using a differential beam leads to better polarization correction.

Full beam correction

An example to perform Stokes I imaging and full beam correction:

```
wsclean -apply-primary-beam -size 1024 1024 -scale 20asec \
  observation.ms
```

WSClean outputs the normal uncorrected image (`wsclean-image.fits`), the 16 components of the beam Mueller matrix (`wsclean-beam-0.fits`, `wsclean-beam-1.fits`, ..., `wsclean-beam-15.fits`) and the primary beam corrected image (`wsclean-image-pb.fits`). Note that the dirty, residual and model images are not corrected.

Note: The single component beam images are not easy to interpret by themselves. Together, the 16 images form the complex Hermitian Mueller matrix for each pixel. See the *technical chapter on primary beam component images* for more info.

Example to make beam-corrected Stokes I, Q, U and V images:

```
wsclean \
  -apply-primary-beam \
  -pol iquv -size 1024 1024 -scale 20asec \
  observation.ms
```

WSClean outputs the 4 normal uncorrected images, the 16 components of the beam Mueller matrix and the 4 primary beam images (which have `pb` in them). Other combinations of polarizations, such as `xx,yy` or `iq` are currently not supported and will result in an error. If you need other modes let me know.

The beam correction can also be applied together with *multi-frequency output*, *joined channel mode* and/or the *snapshot mode*. In those cases, a `pb` image is saved for every output image. The channel-integrated "MFS" image is not corrected.

Beam correction works together with *baseline-dependent averaging*, but only since *WSClean 2.5*. Before that, WSClean would crash or give incorrect results when combining primary beam correction with baseline-dependent averaging.

Differential beam

To make primary-beam corrected images for observations in which the visibilities have already been (scalar) corrected for the beam at the phase-centre, the option '`-use-differential-lofar-beam`' can be added. ("`-apply-primary-beam`" is still required). In normal use-cases, this option should not be used, because WSClean determines itself what the correct beam is, and will make sure to output a correctly normalized image even if a scalar beam was applied previously. The combination "`-apply-primary-beam -use-differential-lofar-beam`" can be used to force application of the differential beam in cases the metadata of the measurement set does not contain the proper keys to force this.

Warning: This is an expert option that should rarely be used. Incorrect use of this feature will lead to an incorrect flux density values of the correct image.

The `REFERENCE_DIR` column is used for determining what phase centre the beam has been applied to. Mathematically, WSClean then applies the differential beam Di as derived below. The data *V* being imaged have been premultiplied with the central beam *C* for baseline *ij*, and we want to return a matrix that corrects the data for the full beam *B*. Given our data *R*:

$$V_{ij} = C_i^{-1} R_{ij} C_j^{-H}$$

we want to multiple data with a differential beam matrix *D* such that

$$D_i^{-1} V_{ij} D_j^{-H} = B_i^{-1} R_{ij} B_j^{-H}$$

With *B* the full beam matrix. We can solve for D_i:

$$\begin{aligned} D_i^{-1} C_i^{-1} &= B_i^{-1} \\ D_i^{-1} &= B_i^{-1} C_i \\ D_i &= C_i^{-1} B_i \end{aligned}$$

(The same could be achieved by solving for the D_j term in $C_j^{-H} D_j^{-H} = B_j^{-H}$).

3.8.4 MWA specifics

Version 2.7 and upwards can directly apply the MWA beam during imaging. This avoids having to separately image XX and YY if only Stokes I is needed.

As for the other telescopes, the option to make this happen is `-apply-primary-beam`. WSClean will determine from the telescope name stored in the measurement set that this is an MWA observation, and uses the MWA specific keywords that describe the pointing (antenna delays) of the tiles.

Usage of the MWA beam requires having installed the HDF5 file that is installed as part of the MWA repository, which will be searched at runtime. See also https://github.com/MWATelescope/mwa_pb.

3.8.5 Time-varying beams

When using image plane beam correction, WSClean calculates the time-integrated beam by summing snapshot beams; a beam is calculated for every 30 min and every *output* channel. Be aware that the beam correction is a single correction, and is not time-dependent. Hence, if the beam changes over time, information might smear out over the polarizations, leading to poor sensitivity. This is less of an issue when the (central) beam was taken out in the visibilities.

3.8.6 Installation information

LOFAR beam correction is available since *WSClean version 1.11*, AARTFAAC beam correction since *WSClean version 2.6*. To use either beam, you need to have compiled WSClean with the EveryBeam library. CMake reports whether it has found the library. If WSClean has been compiled without the library, and you ask to correct for the primary beam, WSClean will report an error and stop.

3.9 Facet-based imaging

WSClean provides experimental support for facet-based imaging, and applying direction-dependent effects (DDEs) per facet. A facet should be understood as a (polygonal shaped) subdomain of the full image, where both convex and concave polygons are supported. A first advantage of facet-based imaging is that DDEs can be applied per facet. Parallelization and reduction of the memory footprint can be other other advantages of facet-based imaging, because visibility gridding and predicting can largely be done for each facet independently, rather than requiring the full image.

3.9.1 Availability

Facetting is available in WSClean *version 3.0* and later.

3.9.2 Command-line options

To enable facet-based imaging in WSClean, a file containing the facet definitions should be provided via the `-facet-regions` option on the command line:

```
-facet-regions <MY_REGIONS_FILE>
```

in which `[MY_REGIONS_FILE]` is a file containing the facet definitions in the DS9 region file format. For a detailed explanation on the expected file format, see the explanation on *DS9 region file*.

Enabling the facet beam correction can be done with the option

```
-apply-facet-beam
```

The facet beam update interval (in seconds) can be defined by specifying:

```
-facet-beam-update <seconds>
```

The default value for the update interval is 120s.

Direction-dependent corrections per facet can also be read and applied from an H5Parm file - which in essence is a HDF5 file with some prescribed lay-out. This is done via the command-line option:

```
-apply-facet-solutions <path-to-h5parm> <name1[,name2]>
```

where `<path-to-h5parm>` is the path to the H5Parm solution file and `<name1[,name2]>` is a comma-separated list of strings specifying which "soltabs" from the provided H5Parm solution file are used. Acceptable names are `ampl000` and/or `phase000`.

In case WSClean uses multiple measurement sets as input, either one H5Parm solution file or a distinct H5Parm solution file per seasurement set can be specified. The correction that should be applied (`ampl000`, `phase000`, or both) is assumed to be identical for all H5Parm solution files. As an illustration, assume that `N` measurement sets are passed to WSClean, with corresponding solution files `h5parm1.h5, h5parm2.h5, ..., h5parmN.h5` containing a scalar

amplitude correction. The syntax for applying the facet solution files on its corresponding measurement set thus becomes:

```
-apply-facet-solutions h5parm1.h5,h5parm2.h5,...,h5parmN.h5 ampl000
```

Note: To find the matching direction in the solution file for the specified facets, the (RA, Dec) pointing of each facet is matched against the direction with the smallest (Euclidean) distance in the solution file. For further information on the (RA, Dec) pointing of a facet, see *DS9 region file*.

3.9.3 Example command

An example facet-based imaging command in WSClean, applying both a facet-based beam correction as well as a gain correction from an H5Parm file could be:

```
wsclean \
-apply-facet-solutions mock_soltab_2pol.h5 ampl000,phase000 \
-facet-regions ds9.reg \
-apply-facet-beam \
-facet-beam-update 120 \
-niter 1000000 -auto-threshold 5 -mgain 0.8 \
-size 1024 1024 -scale 1amin \
${ms}
```

3.9.4 Caveats

Facet-based imaging is currently an experimental feature, and therefore should be used with care. A (probably non-exhaustive) list of caveats is presented below:

- Parallel processing can be enabled with the `-parallel-gridding` option (multi-threaded) or the `wsclean-mp` (using MPI). Parallelization over facets is however barely tested, and may return unexpected errors or results, in particular when applying DDEs.
- Facet-based imaging in conjunction with the Image Domain Gridder (IDG) is only possible without applying DDEs.
- When applying solutions in WSClean for facetted imaging, only scalar solutions are currently applicable.

CHAPTER

FOUR

GRIDDING MODES

4.1 Image domain gridding

The image-domain gridder (IDG) is a new, fast gridder that makes *w*-term correction and *a*-term correction computationally very cheap. It performs extremely well on gpus.

To use IDG, the IDG library needs to be available during compilation. *WSClean 2.5* is compatible with IDG version 0.2. See *installation* for more information.

4.1.1 Considerations

IDG supports the same cleaning and data selections options that WSClean offers in normal mode (without IDG – so e.g. Cotton-Schwab, multi-frequency multi-scale cleaning, auto-masking, etc.). However there are a few points which require a bit of care to use IDG:

- *Baseline-dependent averaging* does not work together with IDG and should not be used together.
- Imaging multiple polarizations independently (without joining polarizations) is not possible with IDG, see the discussion on polarization settings below.
- IDG can only be used on measurement sets that have four polarizations (e.g. XX/XY/YX/YY).

4.1.2 Command-line options

There are several IDG related WSClean parameters:

```
-use-idg
```

This option turns on the IDG gridder. If IDG is not available, WSClean will show an error.

```
-idg-mode [cpu/gpu/hybrid]
```

IDG mode to use. The default is CPU, which works on any computer. GPU and Hybrid both make use of the GPU and require the availability of GPU libraries when compiling IDG (see the IDG manual). The "pure" GPU mode is however **not recommended** to be used, as it does not have all the features that the hybrid mode has, hence **hybrid is the recommended method when GPUs are available**.

4.1.3 Gridding with the beam

IDG allows gridding with a time-variable beam – currently (as of 2.6), this includes the LOFAR, AARTFAAC and MWA beam. This would increase the accuracy of with which sources are deconvolved (in theory, but it depends of course on the accuracy of the beam model). The option to do this is `-grid-with-beam`, and optionally, if the differential beam needs to be applied (which is common; see the LOFAR beam page for more info) one would also add `-use-differential-lofar-beam`.

In summary; IDG with full beam correction:

```
wsclean -use-idg -grid-with-beam ...
```

IDG with differential beam correction:

```
wsclean -use-idg -grid-with-beam -use-differential-lofar-beam ...
```

For more information about the differential lofar beam, see *primary beam correction*. Note that you do not have to add `-apply-primary-beam`: if you don't specify this parameter, WSClean will grid with the beam and output both the "flat noise" image and the pb corrected image.

NB: In earlier versions, specifying both `-apply-primary-beam` and `-use-idg` would apply the wrong beam!

4.1.4 Polarization settings for IDG

IDG always images all Stokes parameters. Therefore, it makes sense to set the polarizations to iquv (with `-pol iquv`) when using the IDG, as otherwise the other polarizations are imaged and then thrown away. IDG does however require joined or linked polarization cleaning: it is not possible to clean the polarizations individually with IDG in a single run. If this is really desired it can be done by running wsclean 4 times, imaging one polarization at a time, however most science cases will want to use one of the other options described below.

There are several distinct cases of interest:

- Total intensity science, i.e., not interested in QUV: only image I with `-pol I`.
- Total intensity science, but "nice to have QUV" for e.g. sensivity analysis (V is useful for that): image all polarizations with `-pol IQUV` and *link* the deconvolution on polarization I with `-link-polarization i`.
- Interested in rotation measure synthesis, not directly in I and V but nice to have: image all polarizations with `-pol IQUV` and link the deconvolution on polarization Q and U with `-link-polarization qu` (possibly with squared deconvolution etc., see [polarized cleaning page](PolarizedCleaning)).
- Interested in all stokes parameter, cleaning each polarization in a joined way: `-pol IQUV -join-polarizations`.

Some further explanation on this: IDG in combination with `-pol iquv -join-polarizations` can be the best choice in some cases, but if one is only interested in Stokes I, it has the downside that QUV are involved in the cleaning of I, and in case these are mostly empty, the net effect is that the noise in the clean-component finding step is increased slightly. Since [WSClean 2.6](Changelog-2.6) it is possible to "link" polarizations. Its use is explained on the [polarized cleaning page](PolarizedCleaning). To overcome the downside described above, the recommended setting for imaging with IDG is to use polarization linking on I, similar to:

```
wsclean -link-polarization i -use-idg [..]
```

This will clean Stokes I fully, and clean the components found in I also from the other polarizations. However, it will *not* clean structure from QUV that is not found in I.

4.1.5 Performance

On my 4-core home machine, the CPU version is not as fast as the wstacking gridder. However, it can apply *a*-terms and uses less memory, which can be advantageous in some cases. In contrast to the CPU gridder, the GPU gridder *is* considerably faster (up to an order of magnitude), but only for large images (>6k or so).

The exact performance benefits depend heavily on image size, bandwidth and number of cores available. Hence if performance is important, I recommend to make a careful comparison for the particular test case you are interested in.

IDG has been tested and shown to work on MWA data. It performs well, but does require a lot of memory, caused by the wide field-of-view of the MWA and therefore high w-terms.

4.1.6 Advanced *a*-term corrections

WSClean+IDG allows a combination of several direction-dependent corrections to be applied, including TEC screens, diagonal gain correction and position shifts ("dldm screens"). These are discussed on the [a-term correction page](ATermCorrection).

4.1.7 Information for older IDG versions

Before *WSClean version 2.9*, IDG's memory usage was highly dependant on the number of channels in the set. IDG can be made to use considerably less memory by splitting the bandwidth using `-channels-out` and wide-band deconvolution (see *making image cube* and *wideband deconvolution*). For example, splitting the bandwidth in 4 output channels has allowed imaging one of the MWA sets on a 32 GB machine:

> wsclean -grid-with-beam -beam-aterm-update 10 -channels-out 4 -join-channels -link-polarizations i -use-idg -size 1536 1536 -scale 1amin -niter 1000000 -auto-threshold 0.5 -auto-mask 4 -multiscale -mgain 0.8 observation.ms

This should no longer be necessary for WSClean 2.9 and later. In those versions, IDG should honour the requested memory settings and available memory. If you do expect memory issues, you can tweak the memory usage using the `-mem` and `-absmem` parameters of WSClean.

4.2 A-Term correction

A-term correction is available in WSClean when gridding with the Image Domain Gridder (IDG). Info on how to enable and effectively use IDG can be found on the *IDG page*.

4.2.1 Gridding with a-term screens

To use a-term correction or combine multiple corrections, a configuration file can be provided to WSClean to set up the a-term corrections to be applied by IDG. The configuration file can be specified with `-aterm-config <filename>`, and has a structure like this:

```
# This is a test parset. Comments are made by starting a line with a hash symbol
# The aterms option lists all the corrections that are made. For demonstration,
# this parset makes all possible corrections:
# The fourierfit, klfit aterms are new since 3.2
aterms = [ tec, dldm, diagonal, fourierfit, klfit, beam, paf ]

# A tec correction has parameters 'images' and 'window' :
```

(continues on next page)

(continued from previous page)

```
# See the WSClean help for a description of the image format used.
tec.images = [ aterms1-tec.fits aterms2-tec.fits ]
# The window parameter is new since 2.7
# It supports tukey, hann, raised_hann, rectangular or gaussian.
# If not specified, raised_hann is used, which generally performs best.
tec.window = raised_hann

# The dldm correction (source shift) has parameters 'images', 'window' and 'update_
↪interval'
dldm.images = [ aterms1-dldm.fits aterms2-dldm.fits ]
# How often (in seconds) to update the phases computed from the dl,dm values.
dldm.update_interval = 300
dldm.window = raised_hann

# The diagonal correction has parameters 'images' and 'window'.
diagonal.images = [ aterms1-diag.fits aterms2-diag.fits ]
diagonal.window = raised_hann

# The fourierfit (Fourier fitting) correction has only the parameter 'solutions'.
fourierfit.solutions = solutions.h5

# The klfit (Karhunen-Loève fitting) correction has the parameters 'solutions' and 'order
↪'.
# 'order' is the number of KL basis functions to be used for fitting.
klfit.solutions = solutions.h5
klfit.order = 3

# The beam correction has parameter 'update_interval'. It may also have
# telescope-specific options, e.g. the lofar beam supports 'differential', and
# 'usechannelfreq'.
beam.differential = true
beam.update_interval = 120
beam.usechannelfreq = true
beam.frequency_interpolation = true

# The paf correction is an easy wrapper for multi-beaming arrays. In this mode, measured
# beam gains are read from fits files. Each combination of antenna and beam  is stored in
# a separate fits file with total power (scalar) values, and each fits file can have
# multiple frequencies to support having a different beam at different frequencies.
# Each measurement set is assumed to contain the data for a single beam, and
# the order of the fits files specified here should match the order of the measurement␣
↪set.
paf.antenna_map = [ ant_0 ant_1 ant_2 ant_3 ant_4 ant_5 ant_6 ant_7 ant_8 ant_9 ant_10 ]
paf.beam_map = [ 00 01 ]
paf.beam_pointings = [ -09h25m00.0s 55d49m59.0s -09h35m33.0s 54d47m29.0s ]
paf.file_template = beammodels/$ANT/CygA_191120_$BEAM_$ANT_I.fits
paf.window = hann
```

The `aterms` keyword specifies a list of corrections to be applied, and each correction can have some parameters. The TEC and diagonal images are FITS image cubes with a special format; see the chapter below on TEC correction. Those parameters that are specified in the config file are no longer necessary on the command line (e.g. `-grid-with-beam` is no longer effective when a config file is specified with beam). The tec and diagonal image lists can take one or multiple fits files. The fits files are concatenated time-wise, so in an observation with 20 timesteps, the first one can specify the

first 10 and the second one the last 10. This allows timegaps; e.g. if two observations are imaged that have some gap in time, different fits files can be made so that it is not required to "zero pad" the time in between.

The `paf` correction is described in the chapter on *combining pointings*. The other corrections are discussed below.

4.2.2 Kernel size

When gridding with a-terms, it is important to set the `-aterm-kernel-size` parameter to an appropriate value. A-term images are resampled onto a small grid, and they are made as smooth as necessary to fit within the specified aterm kernel size. If your aterm screens (beam / ionospheric / ...) are smooth, the value can be low, but if they need to have a higher resolution, some tuning might be required. In WSClean 2.10, the default size is 16 when specifying a aterm config file. Otherwise / before WSClean 2.10, the default size is 5. A size of five is very small, but enough for the LOFAR beam. It is probably too low for most TEC or gain correction screens. You can check the result of resampling the aterms to the specified size with the `-save-aterms` option that is described elsewhere on this page.

4.2.3 TEC correction

IDG can apply a spatially varying time-variable TEC term that can additionally be different for different antennas and output channels. To use this, the `-aterm-config` option described above should be used to supply a tec image. The provided TEC image should have 5 dimensions, ordered as follows:

- RA
- DEC
- Antenna
- Frequency
- Time

The number of antennas should either match with the imaged measurement set, or should have a dimension of one, in which case the same aterm is used for all antennas. The time dimension is optional: when not specified, the same corrections are applied to all times. The RA and DEC dimensions are interpolated on the IDG sub-grid via a combination of low-pass filtering and nearest neighbour interpolation. This is typically around 64-256 pixels, so providing images that are larger is not necessary. The frequency and time axes are also interpolated. The RA and DEC dimensions should be in the standard radio imaging projection with appropriate `CRPIX`, `CRVAL` and `CDELT` settings. These parameters need also to be appropriately set for the FREQ and TIME axis. The frequency axis has values in Hz. The time axis should have AIPS/Casacore time values. These time are Modified Julian Dates (MJD), but *in seconds*, so they are MJD values multiplied by 86400. For example, the 8th of May in 1982 would be represented as 3.8965e+09. The times in the FITS file have the same meaning (and units) as values in the TIME column in the measurement set; so they represent the time at the centre of the timestep. The screen is selected whose time is nearest to that of the value in the TIME column.

Since TEC values are interpolated over frequency with its 1/ relation, it is normally not required to have more than one channel in the image, unless higher order terms need to be corrected. The correction is constant per output channel, so the output channels have to be chosen such that they are fine enough to achieve the desired accuracy. The values in a TEC file are applied as "delta TEC terms", meaning that a value of zero implies no change to the gain of the antenna. The phase of the gain (in radians) is evaluated as: `phase = image[pixel] * -8.44797245e9 / frequency`, with frequency in Hz.

This is an example header of an aterm TEC fits file:

```
SIMPLE  =                    T / file does conform to FITS standard
BITPIX  =                  -32 / number of bits per data pixel
NAXIS   =                    5 / number of data axes
NAXIS1  =                 1024 / length of RA axis
```

(continues on next page)

(continued from previous page)

```
NAXIS2  =                 1024 / length of DEC axis
NAXIS3  =                   48 / length of ANTENNA axis
NAXIS4  =                    1 / length of FREQ axis
NAXIS5  =                   10 / length of TIME axis
EXTEND  =                    T / FITS dataset may contain extensions
[..]
CTYPE1  = 'RA---SIN'           / Right ascension angle cosine
CRPIX1  =                513.
CRVAL1  =          123.4002825
CDELT1  =              -0.0125
CUNIT1  = 'deg     '
CTYPE2  = 'DEC--SIN'           / Declination angle cosine
CRPIX2  =                513.
CRVAL2  =     48.2173836111111
CDELT2  =               0.0125
CUNIT2  = 'deg     '
CTYPE3  = 'ANTENNA '
CRPIX3  =                  1.
CRVAL3  =                  0.
CTYPE4  = 'FREQ    '           / Central frequency
CRPIX4  =                  1.
CRVAL4  =     138475036.621094
CDELT4  =          183105.46875
CUNIT4  = 'Hz      '
CTYPE5  = 'TIME    '
CRPIX5  =                  1.
CRVAL5  =         5020582991.9 / MJD in seconds
CDELT5  =                32.0 / 32 seconds per aterm
```

4.2.4 dldm gain correction

"Dl-dm" gain correction can apply a positionshift to correct the position of sources. This kind of correction works almost the same as TEC correction. It also requires a FITS file with 5 dimensions:

RA, DEC, MATRIX, FREQ, TIME

Again, the TIME dimension is optional: when not specified, the same corrections are applied to all times. Like with TEC correction, the dimensions need to be given in this exact order. The dimension `MATRIX` should have 2 elements: one for the `dl` values, and one for the `dm` values. The other dimensions are as described for TEC correction.

4.2.5 Diagonal gain correction

Diagonal gain correction can correct the visibilities with a diagonal Jones matrix. Therefore, diagonal correction performs a correction with two complex values, one for XX and one for YY. Diagonal gain correction with IDG works almost the same as TEC correction. Instead of a FITS file with 5 dimensions, diagonal correction requires a FITS file with 6 dimensions:

RA, DEC, MATRIX, ANTENNA, FREQ, TIME

Like with TEC correction, the dimensions need to be given in this exact order. Compared to the TEC aterms file, there's one extra dimension: `MATRIX`. For diagonal gains, this matrix dimension has 4 elements: real XX, imaginary XX, real YY and imaginary YY. The other dimensions have their same use. The frequency axis is used to find the nearest image-frequency for each visibility (this works since *version 2.8*).

If you get images out with all NaNs, the gains might be all zero at some position. For TEC or dldm correction, this obviously is not a problem (zero phase=no correction), but for diagonal gains, a zero matrix leads to division by zero at some point. This can in particular happen because IDG pads the image – so if one makes TEC aterm images that are exactly the size of the output image, they won't cover the border.

4.2.6 Fourier fitting

The solutions of a calibration step are given via the "solutions" parameter in h5 format. From the solutions file, only the phases are used. The discrete set of solutions are fit to a screen using a Fourier based fitting technique.

Warning: This is an experimental feature as of May 2022. It has not been tested on real data, hence it is not sufficiently robust to outliers / NaNs that may be encountered there.

4.2.7 Karhunen-Loève fitting

The solutions of a calibration step are given via the "solutions" parameter in h5 format. From the solutions file, only the phases are used. The discrete set of solutions are fit to a screen using a Karhunen-Loève based fitting technique.

4.2.8 Analyzing / saving the a-terms

The `-save-aterms` can be useful for diagnostic output. It turns on saving of the TEC screen after resizing them to the IDG subgrid size and low-pass filtering them to the kernel size (see the kernel size section for more info). The output images are named "`aterm-ev0.fits`" and "`aterm-realxx0.fits`", with increasing numbers for the different aterms over time and counting further in subsequent cleaning iterations. Each image contains a mosaic of images, one image per antenna, starting counting in the bottom left. The images with "ev" in their name are the eigen value of the Jones matrix. These reflect e.g. the power of the beam when imaging with the beam. When imaging with only TEC aterm values, the values are all one, because a TEC change is just a phase change, and the eigenvalue of such a matrix is one: hence not very useful! The images with "realxx" in their names, are the real value of the first ("xx") element of the Jones matrix. These are more useful for assessing TEC aterm values.

4.3 Combining pointings

WSClean makes it possible to image and clean a combination of pointings and/or a drift-scan, also known as *joint deconvolution*. It is advisable to use *version 2.10* – older versions support some of the features, but in particular the 'measurementset multi-field' support was added only in 2.10.

What is discussed in this page, is to grid multiple pointings together on one grid, and applying the primary beam of each observation separately. It also allows to combine observations with different antenna responses (e.g. LOFAR international stations + LOFAR core stations; heterogenous VLBI arrays; interferometers with PAF like Apertif, etc.). It is somewhat similar to mosaicking a number of pointings together with the beam in image space, except that it additionally allows deconvolution on the full, combined image. This makes it possible to deconvolve images deeper.

4.3.1 Setting the phase centre

For an observation, the pointing centre can be different from the phase centre: the pointing centre is to what direction the dish is pointed and affects the primary beam. The phase centre direction is the direction at which the phases are zero, and is more or less a reference direction that affects what is actually imaged. To make it possible to combine different pointings, all pointings should be set to have the same phase centre. This can be done using the `chgcentre` tool that is shipped with wsclean. The phase centre of an observation can be set as follows:

```
chgcentre observation.ms 10h01m30s 02d26m00s
```

This implies your combined image will be centered on 10h01m30s 02d26m00s. If the observation contains multiple fields, all fields will be set to have this phase centre. See *chgcentre* for more info.

4.3.2 Gridding with the beam

To combine observations weighted with the beam, each visibility should be gridded with the corresponding beam. This is only possible with the IDG gridder, hence the IDG gridder has to be used (add `-use-idg` and optionally `-idg-mode hybrid` to use GPUs: see *image domain gridding*).

WSClean needs to know the beam shape. A few arrays are supported 'out of the box', such as LOFAR, MWA and VLA. In that case, simply adding `-grid-with-beam` is enough. Otherwise, a FITS file can be specified with the appropriate beams; see *a-term correction*.

WSClean needs the beam to have a certain level of smoothness, which results in a certain support of the kernel. In case the beam has unsmooth parts (e.g. near the horizon, or near the estimated response), the default settings might not be enough. Option `-aterm-kernel-size` can be used to change the default. Most beams are quite smooth, and don't require a kernel larger than 5..16 pixels. Hence, a safe start is to use a size of 16 and tweak later on for performance.

It is possible to apply other direction-dependent corrections during the imaging. To do so, the list of a-terms needs to be defined using a separate config file as explained on *the a-term correction page*. Such a config file can also specify the beam gridding settings, which means that in that case the parameters like `-grid-with-beam` do not have to be set on the command line.

4.3.3 PAF settings

WSClean has some specific features for joint convolution of multi-beam system, such as phased-array feeds (e.g. APERTIF or ASKAP). For such telescopes, it is often desirable to provide response images per beam, for which the individual beam images have also been centred on the pointing direction of each beam. This is normally simpler than providing response images that all have been gridded to the full field centre. What this implies is that each beam image only has to capture the part of the beam with significant gain.

To perform such 'paf' imaging, it is expected that the data for each beam is stored in a separate measurement set. Each beam and/or antenna may provide a separate beam fits file. The fits file may also have a frequency axis. Additionally, the beam can be corrected for the frequency of the visibilities.

This information can be provided by an a-term config file as described in the chapter *a-term correction*. Here is an example:

```
aterms = [ paf ]

paf.antenna_map = [ RT0 RT1 RT2 RT3 RT4 RT5 RT6 RT7 RT8 RT9 RT10 RT11 ]
paf.beam_map = [ 00 01 ]
paf.beam_pointings = [ -09h25m00.0s 55d49m59.0s -09h35m33.0s 54d47m29.0s ]
paf.file_template = observation_cbeam-$ANT-$BEAM.fits
```

(continues on next page)

(continued from previous page)

```
paf.window = raised-hann
paf.reference_frequency = 1355e6
```

To allow some flexibility, the filenames of the beams are specified with a file template. This template may use the special variables `$ANT` and `$BEAM`, which will be replaced by values from the `antenna_map` and `beam_map`, respectively. If `$ANT` and/or `$BEAM` is not used, the map still needs to be defined, but its values are not used. The `beam_map` maps each measurement set to the beam string that can be used in the file_template. The `antenna_map` links the antennas in the measurement set to an antenna string that can be used in the file_template. With the above example, visibilities from the first measurement set that need to be corrected for antenna 0 are corrected by the fits file `observation_cbeam-RT0-00.fits`.

The `beam_pointings` variable is required to hold the centres of the beam images (ra and dec values, separated by spaces). The `reference_frequency` is used as the central "unit scale" frequency of the fits file: for higher frequencies, the image size is shrunk by their ratios, and similarly stretched for lower frequencies. If this kind of frequency correction is not desired, the `reference_frequency` can be set to 0 or left out.

4.3.4 Other image settings

- The image dimensions should be set such that all pointings are included. For example, if you have 2x2 pointings that connect at the fwhm, and the fwhm corresponds to 1000 pixels, the phase centre should be set to the centre of the 4 pointings and by making an image of 2000x2000, all pointings are included up to their fwhm.
- When using 'multiple field' observations, that have been phase centered to the same direction, option `-field all` can be used to combine all fields, or a comma-separated numbered list can be specified if only a part of the fields are to be imaged, e.g. `-field 0,1,2` . Note that these option are only supported when all fields have the same phase center.
- Start with a low value for `-mgain`, like 0.6, or even lower when cleaning diverges or the uv-coverage is not so well to begin with. Cleaning a mosaick of pointings with different responses is not as stable as cleaning a homogenous observation.
- Because it is likely that the images are going to be very big when combining pointings, it is probably advisable to use the *parallel deconvolution option*.
- Because the noise probably changes over the image, it is likely useful to use the `-local-rms` option. See the *local RMS cleaning page* for more info.

Here is an example run:

```
wsclean \
    -use-idg -grid-with-beam -aterm-kernel-size 16 -multiscale \
    -field all \
    -mem 10 -temp-dir ~ -name fullfield -weight briggs 0 \
    -size 8192 8192 -scale 500masec \
    -niter 1000000 -nmiter 10 -mgain 0.5 -auto-threshold 1 \
    -auto-mask 5 -channels-out 4 -join-channels \
    -local-rms -parallel-deconvolution 4000 \
    vla-observations.ms
```

4.4 W-gridding

W-gridding is a wide-field gridding algorithm similar to W-stacking, but instead of assigning every visibility to its nearest w-plane, it extends the concept of uv-gridding to the w-direction and therefore grids each visibility to a small range of w-planes, weighted with a kernel function. The theoretical foundations of the algorithm are described in Ye, Gull, Tan & Nikolic (2021); a technical description of the implementation is given in Arras, Reinecke, Westermann & Enßlin (2020).

W-gridding is enabled by the command-line flag `-use-wgridder`, and its accuracy can be controlled via the parameter `-wgridder-accuracy`, which is set to `1e-4` by default and can be varied in the range from `1e-2` (very coarse, but fast gridding) down to `1e-6` (most accurate gridding achievable with single-precision floating point). This value specifies the maximum acceptable rms error of the result when compared to a direct Fourier transform. The algorithm will select appropriate parameters (like amount of padding, kernel shape and kernel support) automatically to reach the requested accuracy in the least amount of time. Therefore, many parameters accepted by `wsclean`'s w-stacking gridder (e.g. `-padding`, `-nwlayers*`, `-grid-mode`, `-kernel-size` and `-oversampling` and `-parallel-gridding`) will be ignored in that mode.

The algorithm has a very small memory footprint: it only requires storage for a single complex w-plane, a copy of the dirty image and some indexing data structures, which are typically much smaller than the visibility data.

Both gridding and degridding directions are available and support shared-memory parallelization that can be controlled using the `-j` parameter.

The w-gridder is available since *WSClean version 2.9*, and was further improved in speed in versions *2.10* and *3.0*.

CHAPTER

FIVE

DECONVOLUTION MODES

5.1 Masks and auto-masking

Masks are often used to be able to clean up to/below the noise level. Deeper cleaning leaves fewer undeconvolved residuals behind, hence the image quality is higher, and it decreases the self-cal bias when used inside a self-cal loop.

5.1.1 Providing a mask

WSClean accepts masks in CASA format and in FITS file format. A mask is a normal, single frequency, single polarization image file, where all zero values are interpreted as being not masked, and all non-zero values are interpreted as masked.

A basic example:

```
wsclean -fits-mask mymask.fits -niter 1000 -mgain 0.8 \
  -size 1024 1024 -scale 10asec myobservation.ms
```

A model image can be used as a mask file. In standard clean (i.e., not multi-scale clean) this can be used to limit the clean components to significant features. For example, one can first clean to 3 sigma (with the *-auto-threshold option*), then use the model as fitsmask and clean to e.g. 0.3 sigma. This does require two runs though, which is not necessary. Also, while this same method can be used for multi-scale cleaning, in general a multi-scale clean will make large parts of the model image non-zero, and cleaning will therefore not be very well constrained. To overcome both issues, WSClean implements a technique called auto-masking.

5.1.2 Auto-masking

Auto-masking allows automated deep cleaning and solves the two problems mentioned above:

- Only one run of wsclean is required;
- It maintains scale-dependent masks, which improves multi-scale cleaning.

Auto-masking works in all modes since *WSClean 2.2*. The general syntax is as follows:

```
wsclean -multiscale -auto-mask 3 -auto-threshold 0.3 \
  -niter 1000000 -mgain 0.8 -size 8000 8000 -scale 2asec \
  obs.ms
```

This will start multi-scale cleaning first towards a threshold of 3 sigma. During multi-scale cleaning, the positions and scale of each component is recorded and put in a scale-dependent mask. Only the center pixel of the kernel is placed in the mask. Once the 3 sigma level is reached, cleaning will continue towards the final threshold of 0.3 sigma. During that stage, the scale-dependent masks are used to constrain the cleaning.

A scale-dependent mask makes sure that when a certain scale-kernel size was cleaned, only that same scale is allowed at that position when the mask is used.

The combination `-auto-mask 3 -auto-threshold 0.3` seems like a good general setting, which normally leaves almost no residuals behind.

In cases where the RMS varies strongly over the field of view, for example because of calibration artefacts, it might be useful to use [local RMS thresholding](LocalRMSThresholding), instead of thresholding relative to the global RMS.

5.1.3 Combining auto-masking with a regular mask

Auto-masking works in combination with normal masking, so that WSClean can be forced to to find components only in a certain area, and after reaching the auto-masking threshold it would continue with the automask. Since the automask will only contain components within the fitsmask, no components are found outside the fitsmask. Auto-masking and regular masking can be combined by specifying both `-fits-mask <file>` and `auto-mask <threshold>` on the command line.

5.1.4 Availability

Auto-masking is available in multi-scale since *WSClean version 2.1*, and is available in all modes since *WSClean 2.2*.

5.1.5 References

The auto-masking algorithm is described and demonstrated in Offringa and Smirnov (2017).

5.2 Local RMS thresholding

Since *WSClean 2.3*, WSClean supports setting thresholding levels that are relative to the local RMS. This is useful when the RMS varies strongly over the field of view. A typical use-case for local RMS thresholding is for cleaning images that contain strong calibration artefacts. In this case, strong sources might generate strong artefacts. Those artefacts might be much stronger than faint sources elsewhere in the field of view. To be able to clean those faint sources, but not clean the artefacts, local RMS thresholding is required.

There are two ways of using local RMS thresholding:

- WSClean can automatically generate an RMS map; or
- An pre-made RMS map can be supplied to WSClean.

The two cases will be described in the sections below.

Note that when using local RMS thresholding, values reported during the minor cycles are no longer absolute fluxes, but have been scaled in some way to account for the local RMS.

5.2.1 Auto-generated RMS map

When WSClean is asked to use a local RMS without supplying an RMS image, WSClean will generate an RMS map. This is done every major iteration, before the clean loop is called. A typical run looks like this:

```
wsclean -size 1024 1024 scale 10asec -local-rms \
  -auto-threshold 3 -mgain 0.8 -niter 1000000 \
  [...] observation.ms
```

The local RMS thresholding is typically to be used together with the `-auto-threshold` parameter. While it is possible to combine it with a normal `-threshold` parameter, which will scale the threshold relatively to the minimum in the RMS map, this is not recommended. Local RMS thresholding can also be used together with [automatic masking](Masking) using the `-auto-mask` parameter. A typical run to do this looks like this:

```
wsclean -size 1024 1024 scale 10asec -local-rms \
  -auto-threshold 0.3 -auto-mask 3 -mgain 0.8 -niter 1000000 \
  -multiscale [...] observation.ms
```

Note that the `-local-rms` parameter was called `-rms-background` before *WSClean 2.4*.

The default configuration is to calculate the RMS over a Gaussian kernel with dimensions of 25 times the PSF. This value can be changed with the `-local-rms-window` option, for example:

```
wsclean -size 1024 1024 scale 10asec -local-rms \
  -local-rms-window 10 -auto-threshold 3 -mgain 0.8 \
  -niter 1000000 [...] \
  observation.ms
```

Instead of just using the RMS, it is also possible to use a combination of the local RMS and the negated minimum pixel. The formula used for this is max(window_rms, -1.5/5 x window_min) . This can be selected by adding `-local-rms-method rms-with-min`. We have not seen much difference between using the normal local RMS or this combined quantity.

5.2.2 Using a pre-existing RMS map

A pre-existing RMS map can be supplied with the `-rms-background-image` parameter, like this:

```
wsclean -size 1024 1024 -scale 10asec -local-rms-image rmsmap.fits \
  -auto-threshold 3 -mgain 0.8 -niter 1000000 [...] observation.ms
```

The input map (`rmsmap.fits` in this case) will have to have the size of the (trimmed) output image.

A typical use-case for this is to supply an RMS map created by a source detector. Generally, to use this method, it is required to image the field twice. Also, when an automatically-created RMS map is used instead of a fixed pre-made map, WSClean will adapt the RMS map each major iteration, and in most cases the automatic RMS map will therefore be more accurate compared to a partially cleaned RMS map created by (for example) a source detector. The pre-existing RMS map should therefore normally not be the first choice, but there are probably cases where it *is* useful.

5.3 Wideband deconvolution

WSClean has a 'wideband' multi-frequency deconvolution mode, which allows cleaning channels joinedly. This means that peak finding is performed in the sum of all channels, allowing deep cleaning, and the PSF is subtracted from each channel & polarization individually, scaled to the value of the peak in that image, which takes care of spectral variation and deconvolves each channel with its own PSF.

Wideband options are available since *WSClean version 1.1*.

5.3.1 Usage

A typical run in multi-frequency deconvolution would look like:

```
wsclean -join-channels -channels-out 4 [other parameters] \
  <measurement set>
```

This outputs 4 deconvolved images at the different frequencies and the weighted average of those 4. This mode can be combined with Cotton-Schwab imaging (with `-mgain ...`) which would fill the `MODEL_DATA` column with frequency dependent model info, so that it is possible to perform self-cal with the proper frequency information. Joining channels can also be combined with joining polarizations (`-join-polarizations`, see *polarimetric deconvolution*) to clean polarizations and channels both joinedly. This is available from *version 1.5*.

Joined channel deconvolution is computationally more expensive than bandwidth-integrated cleaning. If the processing time (or memory) becomes untractable, the speed and memory usage can be improved by using channel interpolation (see description of `-deconvolution-channels` below) or by using *parallel/subimage deconvolution*.

Note: Something to be aware of is that the `-join-channels` parameter turns on *MF weighting*.

5.3.2 Multiple measurement sets and subbands

WSClean calculates the full available bandwidth (over all measurement sets) and splits that into chunks with the same amount of input channels (or 1 off if the nr of input channels is not divisable by the nr of output channels). For example, if you have four measurement sets where each contains data from a different subband, e.g. 100 MHz, 110 MHz, 200 MHz and 210 MHz, and you specify "`-join-channels -channels-out 2`", the first images will be centred at 105 MHz and contain data from the first two MSes, and the other images will be centred at 205 MHz.

Example:

```
wsclean -join-channels -channels-out 3 -niter 10000 \
  -mgain 0.8 -threshold 0.01 \
  band-100-MHz.ms band-110-MHz.ms \
  band-145-MHz.ms band-155-MHz.ms \
  band-190-MHz.ms band-200-MHz.ms
```

This will perform multi-frequency deconvolution and output images at 105 MHz, 150 MHz and 195 MHz. This works since *WSClean version 1.9*.

In case the measurement sets have a different amout of channels *and* have gaps between them, the normal division of WSClean might divide the channels somewhat undesirably. For example, the data of the MSSS survey observations are stored in 10 measurement sets, each containing one 'band', but one of those 10 has fewer channels. The default WSClean division divides those "missing" channels out over the bandwidth, producing this imaging table:

```
.      # Pol Ch JG ²G In Freq(MHz) ch
| Independent group:
+-+-J- 0  I   0  0  0  0  119-121 (39)
  +-J- 1  I   1  0  1  0  121-126 (40)
  +-J- 2  I   2  0  2  0  126-130 (39)
  +-J- 3  I   3  0  3  0  130-136 (40)
  +-J- 4  I   4  0  4  0  136-146 (39)
  +-J- 5  I   5  0  5  0  146-150 (40)
  +-J- 6  I   6  0  6  0  150-152 (39)
  +-J- 7  I   7  0  7  0  156-158 (40)
```

There are two ways to remedy this. The first, most automated way is to use an option called `-gap-channel-division` (available since *WSClean 2.6*), which calculates the gaps between channels, and splits the input channels into output channels by splitting the largest gap until the number of output channels has been reached. For the above situation, this gives the following table:

```
.      # Pol Ch JG ²G In Freq(MHz) ch
| Independent group:
+-+-J- 0  I   0  0  0  0  119-121 (40)
  +-J- 1  I   1  0  1  0  124-126 (40)
  +-J- 2  I   2  0  2  0  128-130 (40)
  +-J- 3  I   3  0  3  0  134-136 (40)
  +-J- 4  I   4  0  4  0  142-144 (36)
  +-J- 5  I   5  0  5  0  146-148 (40)
  +-J- 6  I   6  0  6  0  150-152 (40)
  +-J- 7  I   7  0  7  0  156-158 (40)
```

Notice the different bandwidth per channel and output channel index 4 which now contains 36 channels.

The second option is to manually add splits by specifying frequencies. For this, the option `-channel-division-frequencies` can be used. The list doesn't necessarily contain all splits necessary. If fewer splits are given than required, WSClean will further subdivide the remaining parts as necessary. Be aware that splits are in Hz, but can be in scientific notation (e.g. 1.4e9 is 1.4 GHz). An example:

```
wsclean -size 512 512 -scale 1amin -channels-out 8 \
  -channel-division-frequencies 140e6,141e6,142e6 \
  observation.ms
[..]
=== IMAGING TABLE ===
      # Pol Ch JG ²G In Freq(MHz)
+-+-J- 0  I   0  0  0  0  134-137 (39)
+-+-J- 1  I   1  1  1  0  137-140 (39)
+-+-J- 2  I   2  2  2  0  140-140 (6)
+-+-J- 3  I   3  3  3  0  140-141 (7)
+-+-J- 4  I   4  4  4  0  141-142 (6)
+-+-J- 5  I   5  5  5  0  142-142 (6)
+-+-J- 6  I   6  6  6  0  142-153 (140)
+-+-J- 7  I   7  7  7  0  153-164 (141)
```

5.3.3 Fitting smooth spectra

The joined channel deconvolution method discussed above does not enforce a smooth spectra; each channel gets a separate solution. If one wants to image many spectral channels separately, while it is known that the sources have smooth behaviour, it is possible to enforce this during cleaning. Currently, WSClean supports fitting a polynomial and fitting a double-logarithmic polynomial. The command line parameters for this are `-fit-spectral-pol` and `-fit-spectral-log-pol`. Both require an extra parameter specifying the number of terms (degrees of freedom). For example, `-fit-spectral-log-pol 2` will fit a power law through all the output channels.

A simple – but very slow – example to perform cleaning with spectral fitting:

```
wsclean -multiscale -join-channels -channels-out 128 -niter 10000 \
  -mgain 0.8 -auto-threshold 3 -fit-spectral-pol 4 observations.ms
```

This will fit a polynomial with 4 terms (i.e., a third-order polynomial). This would be similar to CASA multi-term deconvolution with `nterms=4`. During each minor clean cycle, the 128 images at different frequencies will be added together, the pixel with the highest summed brightness is selected, the brightness for that pixel is found for each image, a 3rd order polynomial is fitted through those measurements and the smoothed "model" component is added to the model, as well as convolved with the PSF and subtracted from the residual dirty image. Spectral fitting works in all joined-channel modes (i.e., hogbom, multi-scale, iuwt, moresane).

As you might imagine, doing a clean with 128 images in memory is expensive, both in terms of memory and computing. Since it is not necessary to have that many images in memory when fitting only a few terms, it is also possible to decrease the number of output channels just during deconvolution. This is done with the `-deconvolution-channels` parameter, for example:

```
wsclean -join-channels -channels-out 128 -niter 10000 \
  -mgain 0.8 -auto-threshold 3 -fit-spectral-pol 4 \
  -deconvolution-channels 8 observations.ms
```

This will decrease the number of images from 128 to 8 before starting the deconvolution by averaging groups of 16 channels together. Cleaning is then performed with just 8 images. After cleaning, the requested function (3rd order polynomial in this case) is fitted to the model, and the model is interpolated using that function. This is much faster than the previous command, and equally precise. Setting the deconvolution channels is supported in all modes since *WSClean 2.2*. The spectral-fitting features were added in *WSClean version 1.11*.

5.3.4 Forced spectral indices

Spectral fitting is useful to reduce the degrees of freedom in deconvolution. It does not always produce accurate, physical spectral indices. This can either be because the bandwidth is too small, sources are very complex causing degeneracies, or both. WSClean version 2.12 has therefore a method called 'forced spectrum fitting'. In this mode, a pre-existing spectral index map is used during the deconvolution, and the resulting spectra of the model components are forced onto this spectral index map.

The spectral index map may be the result of earlier runs or from fitting between data from other bandwidths / telescopes. An added advantage is that those maps can be smoothed to limit the effect of noise and imaging artefacts. The mode is enabled by combining `-force-spectrum <fits filename>` with `-fit-log-pol 2`. The fits file should have the same dimensions and coordinate system as used for the imaging, and each pixel should hold a spectral index value. This mode should currently not be used together with the `-deconvolution-channels` option.

Together with *multiscale cleaning* and *source list output*, this mode allows building (text) models of sources with accurate spectral index information.

This method is currently being written up into an article – to be submitted somewhere in 2022.

5.3.5 Fit normal or logarithmic polynomials?

In general, fitting polynomials (`fit-spectral-pol`) works better than fitting logarithmic functions (`fit-spectral-log-pol`). The latter option fits every component to a power law or higher-termed logarithmic function. This works fine on strong sources, but when cleaning picks up some partly-negative artefacts, it can create bad results and fail. As an example, when one spectral value is negative and the others are positive, a power-law fit that minimizes the squared error, might create extremely high values at the edge channels.

As a side note, I believe that CASA also doesn't fit logarithmic polynomials, but rather fits normal polynomials (I'm not 100% sure from the papers about the technique, but from inspecting the CASA code, it seems to use normal polynomials). Hence, getting CASA's "nterms" behaviour is closest to `-fit-spectral-pol`.

5.3.6 Avoiding "steps" in the model data visibilities

When splitting up bandwidth with the `-channels-out` option, the output `MODEL_DATA` visibilities will have a step function. For example, when splitting up the bandwidth of a 256 channel set in 4 parts (`-channels-out 4`), there will be a step each 64 channels. When the `MODEL_DATA` is important, for example when removing the continuuum data from a HI set, these steps might be undesirable. Using `-fit-spectral-pol` does not remove these steps; that option only forces the individual images to obey a polynomial. To remove the steps, one would have to use `-fit-spectral-pol` and specify as many channels to `-channels-out` as that are present in the measurement set. This is of course very computationally and memory expensive, in particular during deconvolution. To make that possible in some cases, the above option `-deconvolution-channels` can be used in combination with the other options.

Another option to solve these step functions is to interpolate in the frequency during the prediction (i.e. when calculating the `MODEL_DATA`). WSClean can currently not do this, but will hopefully soon be implemented. Because of the already-available features mentioned above it has not been of very high priority, since the quality improvement on continuum imaging is rather small and the computational savings might only exist in some more exotic cases.

5.3.7 Relation to CASA's multi-term deconvolution

To avoid confusion, WSClean's wide-band mode is not the same as the multi-frequency deconvolution algorithm in CASA, i.e., the algorithm described in Sault & Wieringa (1994), further enhanced in Rau and Cornwell (2011). The multi-frequency deconvolution algorithm in WSClean is called "joined-channel deconvolution" and is described in Offringa and Smirnov (2017). As shown there, the multi-frequency implementation of WSClean is computationally *orders of magnitude* faster than MSMFS, which is the most important difference between the two algorithms. Accuracy wise, they produce similar results.

Although WSClean's mode and CASA's mode both mitigate the problem of spectral variation over the imaged bandwidth, the underlying algorithms are different and have different settings. Both CASA and WSClean can fit smooth functions over frequency. WSClean has additional functionality for the imaging of cubes that can have unsmooth spectral functions, such as for spectral lines or off-axis imaging (where the beam causes large/steep fluctuations over frequency). Smooth function fitting is implemented in *WSClean version 1.10*.

5.4 Polarimetric deconvolution

WSClean can perform deconvolution by combining polarizations. Similar to *joining channels*, this implies that peaks (and component shape) are searched in the integrated values over the selected polarizations, but the components strengths are determined from each polarization once its location is selected.

This method improves the quality of polarized images. It can also be used to perform self-cal to keep the polarized flux accurate, although it requires some flux to be present in the polarized images (instrumental leakage). In general, Stokes I quality is not improved by using this method. The method increases computational and memory cost.

Here are two examples for using joined polarization cleaning:

```
wsclean -pol xx,yy -join-polarizations -threshold 1 -mgain 0.8 -niter 10000 \
-scale 15amin -size 1024 1024 -weight briggs 0 galaxy.ms

wsclean -pol iquv -join-polarizations -channels-out 8 -join-channels \
-fits-mask 3c196-mask.fits -threshold 1 -mgain 0.8 -niter 10000 \
-scale 5asec -size 6000 6000 3c196.ms
```

Joined-polarization cleaning works in combination with *multi-scale cleaning*, *multi-frequency deconvolution* and *masked cleaning*.

If you are looking to do rotation measure synthesis with WSClean, have a look at the *RM-synthesis manual page* that describes some particulars for this.

5.4.1 Linking polarizations

Since *WSClean 2.6* it is possible to "link" polarizations. This has the effect of selecting a subset of polarizations for cleaning, but then subtracting components from all imaged polarizations. This results in that the subset of polarizations are cleaned as if they are the only polarizations being imaged, while the others polarizations are still saved, and sources that appear in the subset of polarizations are also cleaned in the other polarizations. Effectively, structure that appears mostly in the linked polarization(s) will be cleaned better. Hence, this is particularly useful for cleaning Stokes I or the XX,YY polarizations.

Note that this is not the recommended setting for RM synthesis; see above.

Linking polarizations is turned on with the `-link-polarizations` option, after which the set of polarizations (or single polarization) should be specified similar to the syntax for `-pol`. An example:

```
wsclean -pol xx,xy,yx,yy -link-polarizations xx,yy -auto-threshold 3 -mgain 0.8 \
-niter 10000 -scale 1amin -size 4096 4096 observation.ms
```

5.5 Rotation-measure (RM) synthesis

WSClean has some options which aid rotation-measure synthesis. The option `-squared-channel-joining` is in particular aimed at performing RM synthesis when combined with `-join-channels` and `-join-polarizations`. This is explained below.

Before reading this chapter, it is useful to understand the following concepts:

- *Making image cubes*
- *Wideband deconvolution*
- *Polarimetric deconvolution*

5.5.1 Basic RM-synthesis imaging

For RM-synthesis, one images the Stokes Q and Stokes U polarizations of a measurement set and produces images at many frequencies. After that, this QU-frequency cube can be Fourier transformed with an external tool to produce an RM cube.

Of course, making those QU images can be done with WSClean as expected, but there are some extra options which can be useful for RM-synthesis. The basic approach to make the cube is described in the manual page *Making image cubes*, where it is described how to use the `-channels-out` option to partition the bandwidth. The `-pol` option can be added to do this for Q and U:

```
wsclean -pol QU -channels-out 100 \
  -scale 1amin -size 1024 1024 observation.ms
```

This produces an *uncleaned* image cube for the Q and U polarizations. Standard cleaning parameters can be added to clean each polarization and each channel individually. However, by doing so one is limited by the noise in a single channel. This can be slightly improved by using the `-join-polarizations` option, and perform peak finding in QU or IQUV space. It is however beneficial to use the full bandwidth for cleaning. This is described below.

5.5.2 Using entire bandwidth for cleaning QU cubes

WSClean has multi-frequency deconvolution options as described on the *multi-frequency deconvolution chapter*. This mode allows using all images together for the peak finding, while still subtracting the peaks from the individual images. The default is to combine the images by summing over the channels. For example, if one is imaging Stokes Q over 100 channels and `-join-channels` is used, peak finding is performed on the integrated bandwidth in Stokes Q. This is undesirable when expecting flux with non-zero RM values, because the flux will average out over the bandwidth. Therefore, the option `-squared-channel-joining` was added. When added, WSClean will perform peak finding in the sum of squares of the images. Values with high RM values will thus not average out. This option can be combined with `-join-polarizations` to take the sum over $Q^2 + U^2$. This is therefore the recommended way to perform RM-synthesis:

```
wsclean -pol QU -join-polarization \
  -join-channels -squared-channel-joining -channels-out <nr> \
  [-niter/-mgain/-scale/... etc.] observation.ms
```

This mode allows one to clean deeper and more accurately compared to per-channel QU cleaning.

When using `-squared-joining`, an MFS image will be outputted as normal in addition to the channel images, and the MFS image will still have the normal average value (not the sum of squares). A cleaning threshold can be given as normal in Jy, and cleaning stops when the sqrt of the average of squares is below that value. The statistics of this value is thus slightly different as normal, and in general one will start cleaning the noise quicker. Some experimentation with the threshold might be required.

The squared channel joining also works together with the multi-scale mode. I have noticed though that the multi-scale algorithm regularly gets stuck when it is asked to search on squared values. This is because the sum of squares might show a large structure, while it is actually just a composition of small structures in the individual channels/polarizations, so that the fit to each individual channel will not remove anything, and WSClean will continue to find that large structure. If this happens, I would first suggest to clean the Q and U polarizations independently: the squared sum over one polarization is much less likely to behave this way. Otherwise, you could stop cleaning before the problem occurs, or turn multi-scale off altogether. There is a somewhat trivial fix for wsclean to git rid of this issue (namely to first convolve each channel and then integrate instead of integrating and then convolving), which I hope to implement in the future.

5.5.3 What is calculated?

Here are a few examples, along with a description of how peak finding is performed with the given settings:

```
wsclean -pol QU -channels-out 100 ...
```

Cleaning is performed indepently for each polarization, peaks are found in each image.

```
wsclean -pol QU -join-polarizations -channels-out 100 ...
```

Cleaning is performed in Q^2 + U^2, but independently for each channel.

```
wsclean -pol Q -join-channels -channels-out 100
```

Cleaning is performed in sum over channels of Q_ch: Pixels with non-zero RM values will average out and will not be cleaned (`-squared-channel-joining` should be added).

```
wsclean -pol QU -join-channels -channels-out 100
```

Cleaning is performed in sum over channels of Q_ch and separately for Q and U. Again, pixels with non-zero RM values will average out and will not be cleaned (`-squared-channel-joining` should be added).

```
wsclean -pol QU -join-polarizations -join-channels -squared-channel-joining -channels-
↪out 100 ...
```

Cleaning is performed in sum over channels of $Q_{ch}^2 + U_{ch}^2$. **When doing RM-synthesis, this is the most sensible option.**

```
wsclean -pol QU -join-polarizations -join-channels -channels-out 100 ...
```

Cleaning is performed in sum over channels of $\sqrt{Q_{ch}^2 + U_{ch}^2}$. Note that in this mode, flux with non-zero RM-values also does not get averaged out, hence squaring is not stricly necessary. The only difference between this example and the above example including `-squared-channel-joining` is the noise properties during peak finding: the square root makes the noise behave slightly worse, hence squaring is preferred (albeit that the difference is probably minor).

Note that these examples only differ in how cleaning is performed, they do not affect the output images otherwise.

5.5.4 Availability

`-squared-channel-joining` is available since *WSClean 1.12*.

5.6 Multi-scale cleaning

Multi-scale cleaning is useful for accurate deconvolution of resolved sources.

5.6.1 Introduction

WSClean supports multi-scale deconvolution with a new algorithm. Like Casa, WSClean's multi-scale deconvolution selects the highest peak and subtracts it with the best fitting 'scale', although internally it works somewhat different.

In WSClean, the scales to fit for do not have to be specified; WSClean will automatically use as many scales as necessary. The delta scale is always present. The next scale is calculated relative to the synthesized beam. Further scales are added by continuing to multiply by two until the scale is larger than the image size.

Multi-scale deconvolution can be turned on by added `-multiscale` to the command line. The other clean parameters that are also used for "normal" clean runs, work in multisscale mode in the same way. Here's an MWA example of a typical multi-scale deconvolution:

```
wsclean -multiscale -mgain 0.8 -niter 50000 -auto-threshold 5 \
  -size 2048 2048 -scale 0.8amin vela.ms
```

This is a simple cleaning run, using Cotton-Schwab iterations subtracting 80% of the flux in each major iteration. The output is a normal dirty, residual, etc. image, but the model will be composed of the combination of several scales.

5.6.2 Bias parameter

There's one specific multi-scale parameter to tweak the results: "`-multiscale-scale-bias`". This parameter balances between how sensitive the algorithm is towards large scales compared to smaller scales. Lower values will clean larger scales earlier and deeper. It's default is 0.6, which means something like "if a peak is 0.6 times larger at a 2x larger scale, select the larger scale". (Peaks are normalized by their scale size, so the actual selection is a bit more complex). This implies that a smaller value favours selection of larger scales. The value 0.6 seems to generally work well, and was also shown to have favourable properties (Offringa & Smirnov 2017). If you use Briggs or Natural weighting to accentuate larger scales, it might be necessary/better to increase the value. In a LOFAR imaging with Briggs weighting and robustness of 0.5 (somewhat towards natural), I noticed slightly better results with a value of 0.7. With 0.6, the delta scale was almost never used. The following command runs WSClean with a scale bias of 0.7:

```
wsclean -multiscale -multiscale-scale-bias 0.7 -mgain 0.8 \
   -niter 50000 -auto-threshold 5 -size 2048 2048 -scale 0.8amin \
   vela.ms
```

CASA's MSMFS and Moresane determine automatic multiscale bias values based on the strength of the PSF and/or the SNR at a given scale. This however causes image weighting to no longer have any effect on the deconvolution, which makes it impossible to tweak and/or focus on larger scales first. Therefore, WSClean uses the bias value.

5.6.3 Multi-scale with multi-frequency or polarimetric cleaning

The multi-scale algorithm works well in combination with the parameters "`-join-channels -channels-out`". This woul perform peak finding and scale selection on the integrated image, and fit the found scale to each output frequency. This mode is explain in the *wide-band imaging* chapter. In this mode, corrections are made for a wide band with a changing flux over frequency. Therefore, this mode is somewhat similar to CASA's MSMFS mode, but it is a different algorithm. The following statement would split the total bandwidth in four parts, and perform multi-frequency deconvolution.

```
wsclean -multiscale -channels-out 4 -join-channels -mgain 0.8 \
  -niter 50000 -auto-threshold 5 -size 2048 2048 -scale 0.8amin \
  vela.ms
```

Multi-scale clean also works with *polarimetrc cleaning* in a similar way. The combination of multi-frequency and multi-polarization imaging is also possible. For example, the following statement makes eight images; four images of both the xx and yy polarization, and cleaning is performed on the full integrated image:

```
wsclean -multiscale -pol xx,yy -join-polarizations \
  -channels-out 4 -join-channels -mgain 0.8 \
  -niter 50000 -auto-threshold 5 -size 2048 2048 -scale 0.8amin \
  vela.ms
```

5.6.4 Masks

Multi-scale cleaning works in combination with *masks*. If a mask is used, WSClean will only consider peaks that are in selected areas. For non-delta-scales, this implies that the centre of the structure has to lie in a selected area.

Auto-masking is also possible in combination with multi-scale cleaning, and is in fact one of WSClean's major improvements over other multi-scale algorithms, in terms of deconvolution quality (as was shown in Offringa & Smirnov 2017), and does even better than compressive sensing algorithms algorithms in terms of residual RMS (but not always in terms of model image quality). Because thresholds are calculated automatically, a cleaning mode such as the following command works often well without further tweaking:

```
wsclean -multiscale -auto-threshold 1 -auto-mask 5 \
  -niter 1000000 -mgain 0.8 \
  -scale 1amin -size 4096 4096 obs.ms
```

For more info about masking, see the chapter on *masks and auto-masking*.

5.6.5 References

WSClean uses a multi-scale algorithm that is an optimized version of Cornwell (2018). The full algorithm is described in Offringa and Smirnov (2017).

5.6.6 History

This section will describe the multi-scale algorithm introduced in *WSClean 1.9*. Previous WSClean versions had a different multi-scale implementation, which is now deprecated.

5.7 IUWT compressed sensing

IUWT stands for isotropic undecimated wavelet transform, which is a wavelet transform suitable for astronomical imaging. The IUWT is a way to decompose an image into different scales, similar to multi-scale clean. The IUWT approach is combined with a conjugate gradient deconvolution step. This is fundamentally different to the CLEAN approach that removes a single peak at a time.

The IUWT and *MORESANE* methods are somewhat experimental algorithms. They have shown good results on well calibrated data, but at the same time have shown rather bad results for data that (still) contain calibration errors. In the latter case, normal cleaning performs better. Also, with the advancement of *multi-scale cleaning*, compressed sensing methods have become relevant only to corner cases.

5.7.1 Relation to MORESANE

WSClean's IUWT deconvolution method is an extension/rewrite of the MORESANE deconvolution algorithm. PyMORESANE also uses IUWT and the conjugate gradient method, and is originally implemented by Jonathan Kenyon. It has shown great results on some obervations. It is implemented in Python, and can be used inside WSClean as described on the MORESANE page. The native Python implementation is quite slow, although it was specialized for GPUs, and when these could be used, the algorithm was faster. Reasons to implement a new algorithm are: to have a fast implementation in native C++ even without GPUs; to support multi-frequency and joined polarization deconvolution; and to support masks.

5.7.2 Usage

The IUWT deconvolution method is enabled with option "`-iuwt`". This method accepts all of the normal parameters. IUWT supports masking since *WSClean 1.12*. Like the `-multiscale` parameter (see *multiscale*), it is enabled with a single option and has reasonably optimized default settings. However, iterations are much slower. At the same time, it subtract much more flux per iteration, hence also needs fewer iterations. Therefore, `-niter` should normally be much lower than for cleaning; typical values are ~200 for complex images. Also, "`-gain`" can be somewhat higher, because each minor iteration performs a subminor iteration to deconvolve selected scales. A typical value for `-gain` is 0.2. Altogether, a typical IUWT run looks like this:

```
wsclean -iuwt -gain 0.2 -mgain 0.6 -niter 200 \
  -size 1024 1024 -scale 2.4amin cyga.ms
```

Wideband cleaning and *polarimetric deconvolution* are supported, and implemented in the same way as for *multi-scale cleaning*.

5.8 MORESANE deconvolution

WSClean supports deconvolution with the MORESANE algorithm, which is a compressed sensing method. The MORESANE algorithm is described by Dabbech et al. (2015).

WSClean also implements a newer *IUWT deconvolution algorithm* that supersedes MORESANE in most cases. The new IUWT mode supports more features, such as multi-frequency and multi-polarization deconvolution, and the CPU version is faster. A full proper comparison between MORESANE and IUWT has yet to be performed.

5.8.1 Using MORESANE

WSClean uses the Python implementation of MORESANE, written by Jonathan Kenyon. To use the MORESANE algorithm, you have to manually install the PyMORESANE source code. See https://github.com/ratt-ru/PyMORESANE.

Once installed, you can run wsclean with the -moresane-ext parameter. Directly after the -moresane-ext parameter, the full path of the '`runsane`' (older versions: '`pymoresane.py`') file needs to be specified, e.g.:

```
wsclean -moresane-ext /usr/local/bin/runsane [other parameters] \
    obs.ms
```

Most normal Clean parameters do not have effect on the moresane deconvolution. The following parameters influence MORESANE deconvolution:

- The `-mgain` parameter turns on a major loop, thereby correcting w-terms in the model. In general, w-terms cause the PSF not to be invariant and moresane does by itself not correct for this. At this point, its exact value does not matter, except that setting it to 1 (the default) implies no major iterations, while setting it lower than one turns on major iterations. Setting it thus lower than 1 is recommended for good imaging.

- The `-niter` parameter sets the number of major (!) iterations to be performed. Setting it to 1 will run MoreSane once, and (with `mgain`!=1) it will subsequently immediately finish after a prediction-imaging round. Setting it higher will call MoreSane more often. In later MoreSane rounds, WSClean will provide MoreSane with the residual image to which the model convolved with the invariant static PSF has been added. This improves accuracy somewhat, because intuitively the image now consists of a PSF that is "less varying". In images where w-values are relevant, I notice significant improvement in the second and third iteration, after which the image seems to have been converged and further iterations do not noticably change the image.
- Specifying '`-no-negative`' will constrain the model to not have negative values. This turns on the '`-ep`' parameter of PyMORESANE. So far, I've not seen this make the model better; in general, false negative components are replaced by other false positive components.
- Parameter "`-moresane-arg`" allows forwarding extra arguments to Moresane.
- Masks are also forwarded to Moresane.
- Parameter "-moresane-sl" sets the sigma depth level for different iterations.

NB: According to Jonathan, PyMORESANE only works with image sizes that are a power of two. Apparently this is solved in newer versions, but only in CPU mode.

NB2: The `-niter` has a significantly different meaning when using MoreSane deconvolution. Specifying hundreds of iterations will not improve quality and will take forever!

I have tested the MORESANE algorithm on images with the (highly resolved) Vela and Puppis A supernova remnants and got very good deconvolution results. My best results were imaged with these settings:

```
wsclean -niter 3 -mgain 0.9 -moresane `which runsane` \
  -size 2048 2048 -scale 1.2amin \
  -name vela-and-puppis vela-and-puppis.ms
```

Cleaning large images can be very expensive; 1-2k pixel images is doable; 1k takes tens of minutes, 2k takes a few hours. It goes up by more than the square, so you probably won't want to try this on 16k LOFAR images, but rather on small images centred on the diffuse sources.

Finally, the MORESANE algorithm does not support joined deconvolutions, so it will not work with the `-join-polarizations` or `-join-channels` modi. The *IUWT deconvolution mode* can be used for this.

5.8.2 History

WSClean supports deconvolution with the MORESANE algorithm since *version 1.7*. *WSClean 1.8* added parameters "`-moresane-arg`" and added support for passing masks to Moresane. *Version 1.9* added parameter "`-moresane-sl`".

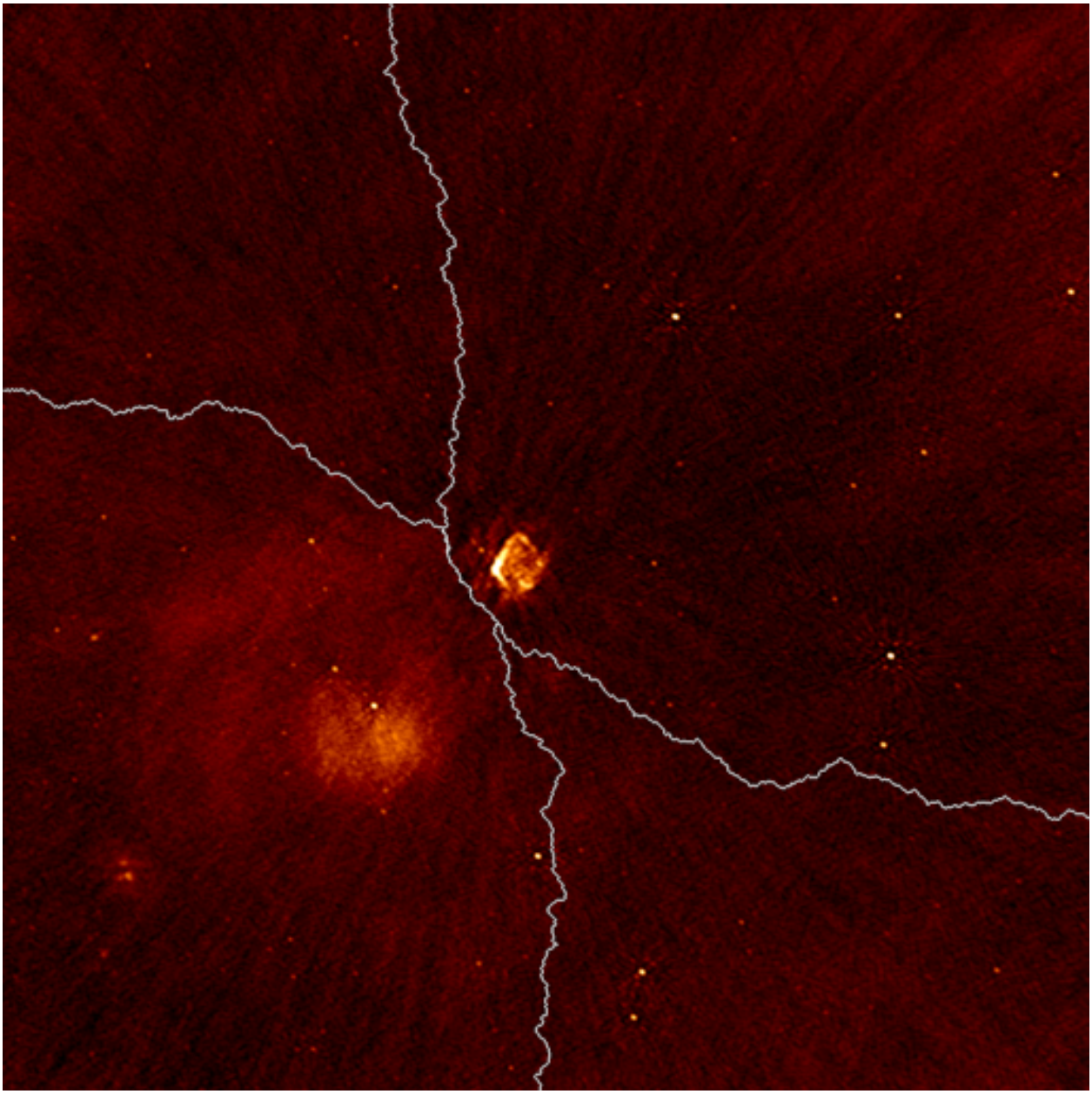
5.9 Parallel deconvolution

WSClean can speed up the deconvolution process by separately deconvolving the sub-images. This is highly desirable for cleaning large images, e.g. 15K x 15K LOFAR images, and this became necessary because the *IDG gridder* can easily produce such images. Despite that WSClean has very fast multi-scale multi-frequency algorithms, IDG has shifted the computational bottleneck back to the deconvolution process.

The parallel cleaning process is fully automated and is turned on with one parameter: `-parallel-deconvolution <maxsize>`, which requires one parameter giving the maximum size of the sub-images. Values of 1024-4096 work well for large images. Smaller values would split the image in very many subimages, increases the computational cost of splitting, whereas larger values might lead to too few subimages, and this might therefore lead to lesser parallelization.

5.9.1 Implementation

Images are split using a (constrained) path-finding algorithm called Dijkstra's algorithm, which calculates the smallest absolute path through the image, constrained to be near a desired splitting boundary. The smallest absolute path is re-determined before every major deconvolution iteration. Here is an example on a small image:



5.10 Direction-dependent Point Spread Functions (PSFs)

With large fields of view, the approximation of considering the dirty image as a convolution of the sky and the PSF yields a higher error. To mitigate this error, WSClean can calculate multiple PSFs in different directions on a grid, using the parameter `-dd-psf-grid`: the user can specify the number of psfs which should be fit horizontally and vertically in the image. The `-dd-psf-grid` argument only works together with the `-make-psf-only` or `-parallel-deconvolution` arguments:

- If `-parallel-deconvolution` is used, and the number of cells in the PSF grid exceeds the number of parallel sub-images in one or more dimensions, WSClean prints a warning and reduces the PSF grid size.
- If `-make-psf-only` is used without `-parallel-deconvolution`, WSClean uses the PSF grid as supplied by `-dd-psf-grid`.
- If both `-make-psf-only` and `-parallel-deconvolution` are not used, WSClean resets the PSF grid to 1x1 and thus disables using multiple PSFs, too.

Warning: This feature is still under developement as of June 2022. If triggered, WSClean will fall back to the existing single PSF implementation.

CHAPTER

SIX

RESULTS & PROBLEMS

6.1 Imaging artefacts

Making a more accurate image is often more expensive, hence the default settings of WSClean are balanced between speed and accuracy, such that average imaging scenarios show accurate fluxes and no visible artefacts. Nevertheless, in high dynamic range scenarios or strong off-axis sources near the edge of the image, artefacts may appear. In those cases, you might want to increase some of the accuracy settings. Here is a short description of the most common problems.

6.1.1 *W*-term correction

WSClean uses by default the *w*-stacking quite a low number of *w*-layers. This is because *w*-layers can increase memory consumption and speed quite significantly, and in most cases the default doesn't introduce any artefacts. The default settings that WSClean use correspond to a decorrelation over 1 radian at the image edge. This translates to quite a small error on sources not near the image edge. However, in the case of strong off-axis sources, those sources might show errors. In the case of snapshot imaging (typical for the MWA), faint "aliased" sources might show. In most cases, these average down when integrating more data.

If you suspect your image is affected by *w*-term corrections, you could: * Use the *w-gridding gridder*; * Increase the number of *w*-layers. The easiest way to do that is by setting the *-nwlayers-factor* option. The default is 1. With a value of 3, the number of *w*-layers will be 3 times higher, implying that the decorrelation between *w*-layers is over 1/3 radian. (The *-nwlayers-factor* option is available since *WSClean version 2.7*

6.2 Diverging clean

It may occassionally happen that CLEAN diverges. This might cause it to produce extremely high or NaN model flux values, or it might get into a 'feedback' loop and keep doing major cleaning iterations without progressing (in this case, 'not converging' is probably a better term than 'diverging').

The problem is more common during *multi-scale clean*, but can ocassionally occur during single-scale cleaning.

During multi-scale cleaning, in the command line output of WSClean this typically looks like the following:

```
Iteration 5978, scale 761 px : 923.32 mJy at 2148,2892
Iteration 6100, scale 1521 px : -930.26 mJy at 4238,2897
Iteration 708620, scale 48 px : -nan Jy at 0,0
```

What this means is that **before** iteration 708620, the deconvolution process has diverged. At the beginning of iteration 708620, the peak in the image is "`-NaN`", according to the log. When, instead of reporting NaN, it reports suddenly a very high value, e.g. 1e7 KJy, this has the same cause. Be aware that the reported position (in this case 0,0) is not the

source of the problem; the iteration that caused the problem happened before 708620, but is not in the output. These components somehow became coupled and diverged.

6.2.1 Causes & solutions

Here are possible causes:

1. **When using multi-scale clean, a common cause is that (very) large scales cause problems. As can be seen in the above output, this is likely here the case, since the iteration that caused problems (Iteration 6100) used a large scale (1521 px). You can use these methods to avoid the issue:**
    A. Don't clean with large scales. Using `-multiscale-max-scales` the number of scales can be limitted. Alternatively (but less preferred), using `-multiscale-scales` a custom list can be provided. If the image does not contain flux at these scales, you can leave the largest scales out without loss in image fidelity. Fewer scales will also speed op multi-scale.
    B. Stop before the problem occurs, by adding one or more stopping criteria. `-nmiter` can especially be helpful for this, but using `-niter`, `-auto-threshold` etc can help as well.
2. Cleaning too deep. It occasionally occurs that cleaning into the noise causes divergence. This is simply fixed by using an appropriate auto-threshold level (e.g. `-auto-threshold 5`, or `-auto-threshold 1 -auto-mask 5`).
3. Having bright emission near/against the edge of the image. This can cause a ripple at the edge, and produce weird results. If this is a likely cause, then it is best to increase the image size. Be aware that when using the `parallel-deconvolution` option, the image has many more edges. Make sure that the parallel deconvolution "size" parameter is not too small.
4. Using a value of `mgain` that's too high. The default of 0.8 is quite stable, but if your PSF has high sidelobes, this value might still be too large. In that case, try lowering it to 0.7 or 0.6. Note that this increases the nr of major iterations, so it can slow down the imaging considerably.

CHAPTER

SEVEN

TECHNICAL DETAILS

Contents:

7.1 Component lists

The benefit of a component list is that it allows representing the components of multi-scale cleaning without 'pixelizing' the larger kernels, thereby decreasing the number of required components in a sky model compared to representing all components with delta scales.

The option is enabled by adding `-save-source-list` on the command line

7.1.1 File format

The component list file format is compatible with the Blackboard Self-cal DP3 sky model formats, but be aware that versions after April 2017 of DP3 support the polynomial spectral function that is used.

When `-save-source-list` is added on the command line, a text file named `<prefix>-sources.txt` is saved. The file contains comma-separated values and looks as follows:

```
Format = Name, Type, Ra, Dec, I, SpectralIndex, LogarithmicSI, ReferenceFrequency=
↪'125584411.621094', MajorAxis, MinorAxis, Orientation
s0c0,POINT,08:28:05.152,39.35.08.511,0.00074881065040047 5,[-0.00695379313004673,-0.
↪0849693907803257],false,125584411.621094,,,
s0c1,POINT,08:22:27.658,39.37.38.353,-0.000154968071120503,[-0.000898135869319762,0.
↪0183710297781511],false,125584411.621094,,,
s0c2,POINT,08:18:44.309,39.38.37.773,0.000233552686127518,[-0.000869089801859608,0.
↪0828587947079702],false,125584411.621094,,,
s0c3,POINT,08:03:07.538,39.37.02.717,0.000919058240247659,[0.001264109956439,0.
↪0201438425344451],false,125584411.621094,,,
[..]
s1c0,GAUSSIAN,08:31:10.37,41.47.17.131,0.000723326710524984,[0.00344317919656096,-0.
↪115990377833407],false,125584411.621094,83.6144111272856,83.6144111272856,0
s1c1,GAUSSIAN,07:51:09.24,42.32.46.177,0.000660490865128381,[0.00404869217508666,-0.
↪011844732049232],false,125584411.621094,83.6144111272856,83.6144111272856,0
[..]
```

Each source (i.e., clean component) is one line in the file. The first line is a header starting with "`Format = ...`" that describes the columns. The header is allowed to specify a default value for this column, as is done above for the reference frequency. When a field is left empty, it should take the default value. See the BBS / DP3 documentation for more info on default values.

7.1.2 The columns

The **Name** contains a unique identifier for this component. In the current format, it consists of the letter 's' followed by the scale index of the component (0 meaning the smallest scale).

The **Type** column is either `POINT` or `GAUSSIAN`. For a point component, the axes and orientation fields are empty. A Gaussian is a standard Gaussian as often used in sky models. When asking WSClean to output a source list, it will automatically enable Gaussian shaped components, instead of the default tapered quadratic.

RA and **dec** are the central coordinates of the component, in notation of "hh:mm:ss.sss" and "dd.mm.ss.sss".

The **I** column represents the flux density in Jy at the reference frequency.

The **SpectralIndex** column is an array of numbers surround by square brackets, that represent the coefficients of the logarithmic or ordinary polynomial (see **logarithmicSI**), when normalized to the reference frequency. The logarithmic polynomial function is given by

$$\log S(\nu) = \log(S_0) + c_0 \log\left(\frac{\nu}{\nu_0}\right) + c_1 \log\left(\frac{\nu}{\nu_0}\right)^2 + ...$$

The logarithms are "base 10" logarithms, such that it is equivalent to say $S(\nu) = 10^{(\log(S_0)+...)}$. Also note that c_0 represents the spectral index term.

An ordinary polynomial function is evaluated as

$$S(\nu) = S_0 + p_0\left(\frac{\nu}{\nu_0} - 1\right) + p_1\left(\frac{\nu}{\nu_0} - 1\right)^2 + ...$$

Note that the value 1 is subtracted in the base. This makes sure that S_0 (the "Stokes I" value) represents the flux density at the reference frequency.

LogarithmicSI is *true* or *false*, denoting that the spectral index column uses logarithmic or ordinary polynomials, respectively. Note that in absense of this column, logarithmic polynomials are used.

ReferenceFrequency gives the frequency in Hz at which the polynomial or logarithmic polynomial is normalized.

The **MajorAxis**, **MinorAxis** and **Orientation** columns define the shape of the Gaussian. The axes are given in units of arcseconds, and orientation is in degrees. Note that currently the major and minor axis of scales are always the same, and thus the orientation has no effect (and is therefore always zero).

7.1.3 History

WSClean supports output of a component list since *version 2.4*. Note that *version 2.3* supported a different format which is now deprecated.

7.2 Weighting and gridding

I got a few questions about weighting and gridding, so I decided to write down a technical description of the way the *w*-stacking algorithm of WSClean performs weighting and gridding. Note that this does not hold for the *w-gridding* and *idg* gridders.

7.2.1 Weighting

Uniform, natural and Briggs' weighting work exactly like in Casa and other imagers:

- Uniform weighting will grid the weights on one "uv" plane (not accounting for w), and each sample will be weighted by this value before being gridded on one of the w-layers. Because it needs to grid the weights prior to filling the wlayers, an extra pass through the measurement set is required. Therefore, this option is slightly slower (~a percent) than the natural and mwa weighting modes, but normally looks the best.
- Natural weighting won't weight the samples before gridding, causing scales that have more baselines to dominate the image. For the case of the MWA, it means that because of the many small baselines, the large scales will dominate the image.
- I've also implemented Briggs' weighting (use with e.g. "`-weight briggs 0.5`") and super-pixel weighting (e.g. "`-super-weight 3`"). Briggs weighting works exactly like Casa's Briggs weighting, although I've heard that the same WSClean Briggs robust value is not exactly identical to the CASA robust value.

Super-pixel weighting is similar to Casa's "super-uniform" weighting, but is more flexible and can also be applied to other weighting schemes, for example to get "super-Briggs'" weighting. You can also set the super-pixel weighting factor to less than one to get sub-pixel weighting, which changes the image more towards naturally weighted. Super-pixel weighting might be nice on very large (full-sky) images, since the immense uv resolution might otherwise start to make your beam become natural-ish weighted (Dan Briggs' thesis) describes sub/super-pixel weighting very well). I think a superweight of "image resolution / 2.5k" should be good to keep the beam uniform/Briggs' weighted.

The weighting scheme is multiplicative to the normal sample weight, i.e., prior to the weighting scheme, samples will be weighted by their weight as specified in the measurement set, and receive zero weight if they are flagged.

7.2.2 Gridding

WSClean can use several Gridding functions. Options include:

- Nearest-neighbour gridding
- A truncated sinc (rectangular function)
- The Kaisser-Bessel function
- A sinc function windowed by a Kaisser-Bessel function
- A Gaussian
- The Blackman-Nuttall window
- The Blackman-Harish window

The KB window is the default, and for most purposes accurate enough. By default, the function is 1023x supersampled with a kernel of size 7. With NN, a sample is placed at the nearest uv-pixel. Especially for small images, that might create aliasing and inaccurate fluxes. The supersampling places samples more accurately on the uv-grid, giving slightly more accurate fluxes. A windowed low-pass filter attenuates aliased sources. Sources that are just outside the field of view might otherwise be aliased back into the image. A Kaiser-Bessel function is very similar, but is considered slightly less optimal compared to the optimized prolate spheroidal function.

A 7 pixel low-pass filter is not very strong, and strong off-axis sources might show up as aliases ("ghost sources"). The best method to deal with those is to increase the field of view; that will also allow proper cleaning of those. Image accuracy is not so sensitive to the size or type of the windowing function, but is much more controlled by the oversampling rate. WSClean uses by default a quite large oversampling rate, because this is trivial to do with the w-stacking algorithm. This is also the reason why WSClean's output images are more accurate than when using a w-projection implementation (e.g. as implemented in CASA).

When making an image cube, it is possible to either grid the weights for each channel separately, or all on the same grid. The latter is called 'MFS weighting'; this is explained in more detail on the *MF weighting* page.

7.3 Restoring beam size

This chapter describes WSClean's algorithm for determining the synthesized beam major and minor axes and the position angle (PA).

The input to the fitting algorithm is the PSF size and an estimate of the PSF FWHM that is determined from the PSF size. Given this first estimate of the beamsize, a box width and height is calculated to be $10\times$ the estimated FWHM of the PSF around the central pixel. A 3-parameter least-squares fit is then performed to find the beam's major, minor and PA values inside that box, to minimize the sum-of-squares difference between the fitted model and the real PSF over all pixels in that box of $10\times$ the size of the PSF. So this includes values out to a few lobes.

Using the full image to do the fit is too expensive – the factor of 10 is to make the fitting go fast. The fit is generally equal to a fit using a larger area.

There are some added heuristical things when fits are very different from the initial estimated PSF: WSClean will iteratively recalculate the fitting box and refit.

7.3.1 Highly elliptical restoring beams

When the fit is highly elliptical and when this is undesirable, the option `-circular-beam` can be added. This constrains the fit to be circular. When forcing a circular beam, WSClean makes still sure that the resulting flux scale of restored sources is correct. This may improve the esthetics of produced images, but it might of course give a false sense of a circular beam.

7.3.2 History

- In *WSClean 2.6*, the option `-beam-fitting-size` was added. It sets the box size to be used during the fit. If the beam that is fitted is different from what is expected, changing this value might help. It seems that in particular lowering the value to 1-3 can solve certain (rare) issues.
- In *WSClean 2.5*, an improved fit for small and forced circular beams was added.

7.4 Polarizations and weights

7.4.1 Calculating Stokes I values

Forming Stokes I from linear/circular polarizations (XX/YY or LL/RR) is not straightforward when visibilities are weighted. When imaging Stokes I, WSClean combines XX and YY (or LL and RR). If either of the polarizations is flagged, the visibility is completely flagged. This means for example that an observation that has all LL visibilities flagged, will result in an empty Stokes I image. Some imagers "solve" this by being able to image "psuedo Stokes I", which is defined as the average over the unflagged polarizations. In case sources in the target image are polarized, this obviously leads to incorrect Stokes I values. This is often not such a concern, although with instrumental polarization leakage caused by fixed dipoles (LOFAR, MWA, etc.) this clearly is undesirable. WSClean therefore does not implement this directly, but can achieve the same effect: by imaging both XX and YY (or LL and RR) polarizations, and using the `-join-polarizations` option, you will be able to image and clean on the combination of the two polarizations. An example command:

```
wsclean -join-polarizations -pol xx,yy -niter 1000000 \
    -auto-threshold 3 -mgain 0.8 myobservation.ms
```

7.4.2 Visibility weights

Visibility weights are the weights that are read from the `WEIGHT_SPECTRUM` column: each visibility has an associated weight value (do not confuse these with *image weights*). When imaging one instrumental polarization, such as XX, just the XX weights are used, so no weights are averaged. When imaging Stokes I, the minimum value of XX and YY (or LL and RR) weights are used. A weight of 0 therefore prevents imaging the average I value altogether, as it should (because a missing XX value means that Stokes I has a completely undetermined value). It's not the statistically optimal, which would be:

$$w_I = \frac{2}{\frac{1}{w_{XX}} + \frac{1}{w_{YY}}},$$

but it is easier/faster, almost as good and has some implementation advantages.

7.5 FITS keywords

7.5.1 Meaning of frequency keywords

WSClean uses the standard `CRVAL` and `CDELT` keywords for the frequency axis. These specify the central frequency and frequency width of the output channel, respectively. Because the frequency axis is the third dimension (the default dimensions are ra, dec, frequency, polarization), this specifically means:

```
CRVAL3   -- centre frequency of the output channel
CDELT3   -- width of the output channel
```

Note: CDELT3 is sometimes used differently: in image cubes with multiple spectral images, CDELT3 is not the channel width, but the channel distance between adjacent images. The width of a channel might be different from the frequency distance, for example because two observations with partially overlapping channels are imaged together. If the first observation has a channel at 1.0 GHz of width 0.2 GHz, and the second observation has a channel at 1.1 GHz of width 0.2 GHz, and they fall into the same output channel, the output image will show CDELT3 = 0.3 GHz, because it covers 0.9 GHz up to 1.2 GHz.

7.5.2 WSClean-specific keywords

WSClean adds the following non-standard keywords to its output FITS files:

```
WSCVERSI -- Version string for WSClean, e.g. "2.5.1"  (since WSClean 2.5).
WSCVDATE -- Version date of WSClean, e.g. "2017-05-08"  (since WSClean 2.5).
WSCNWLAY -- nwlayers
WSCDATAC -- data column
WSCWEIGH -- weighting (textual description, e.g. uniform or Briggs'(0.5))
WSCGKRNL -- gridding (antialiasing) kernel size
WSCCHANS / WSCCHANE -- channel range (only when specified)
WSCTIMES / WSCTIMEE -- time range (only when specified)
WSCFIELD -- imaged field number in MS
WSCNVIS  -- gridded visibility count
WSCENVIS -- effective nr of visibilities (only for natural weighting)
WSCVWSUM -- sum of visibility weight (since WSClean 2.3)
WSCIMGWG -- Weight to be used when averaging images together
```

(continues on next page)

(continued from previous page)

```
WSCNORMF -- Normalization factor that was applied to the image. The factor is
            useful to undo the normalization for e.g. conversion to Kelvins.
            WSCNORMF is available since WSClean 2.
WSCTHRES -- Manually applied threshold
```

Cleaning

```
WSCNITER -- max number of iterations specified
WSCGAIN -- clean gain (minor iteration gain)
WSCMGAIN -- major cleaning gain
WSCNEGCM -- whether negative components were allowed to clean
WSCNEGST -- whether stopping on first negative component during clean
WSCMAJOR -- number of major iterations actually used before reaching stopping criteria
WSCMINOR -- number of minor iterations actually used before reaching stopping criteria
```

The actually used number of major/minor iterations are stored in `WSCMAJOR` and `WSCMINOR`, whereas `WSCMGAIN` and `WSCNITER` describe the input parameters.

No longer used

```
WSCSMPSF -- whether the PSF is made smaller prior to cleaning
```

7.6 Computational performance

This chapter explains some of the ideas to speed up imaging. If you are looking to speed up imaging, you should first determine if your imaging is dominated by gridding or by deconvolution. WSClean outputs the speed of gridding (prediction + inversion) and deconvolution at the end of a run.

7.6.1 Gridder

The biggest choice to make that typically affects computational performance the most is what gridder to use. The default gridder, the *w*-stacking gridder, is very fast for relatively small images (up to about 5k), in particular when it can be combined with the `-parallel-gridding` option. When parallel gridding is not possible, or when image sizes are reasonably large (say 5k-10k), the *w-gridder* might be a better choice. That gridder has the additional advantage that it is (much!) more accurate compared to the default gridder, although the level of accuracy of the *w*-stacking gridder is good enough for most science cases to not be a concern. Finally, for big images the *idg* can be even faster, in particular when gpus are available.

Each gridder performs different for different imaging setups: image size, number of visibilities, size of *w*-terms, resolution, these all impact the performance of the various gridders in different ways. Therefore, to really know which gridder is most efficient I recommend timing each of them.

7.6.2 General improvements

A few generic tips to speed up processing:

- Use *parallel deconvolution*.
- If the deconvolution is a bottleneck, don't use *multi-scale* if you don't need it.
- Use the parallel reordering option.
- When using *w*-stacking, use the parallel gridding option.
- Or even better, irregardless of the gridder, parallellize over multiple nodes using *wsclean-mp*.
- When using *a*-terms, make sure to keep your *a*-term kernel as small as possible (see *a-term correction*).
- Do not include baselines that provide a resolution higher than the imaging resolution. Despite that these baselines fall outside of the *uv*-plane, and therefore are discarded and don't affect performance, at lower elevations these baselines might still fall inside the *uv* plane and might have very large *w*-terms. Yet they don't contribute to imaging quality or the resolution. The best way to filter these baselines is using a `-maxuvw-m` limit. One use-case where this is important is when the international LOFAR baselines are present in the observation, but imaging is done at lower resolution (e.g. up to 2").
- Don't split your data in too many measurement sets. For example, keeping each LOFAR subband in a separate measurement set and imaging the full bandwidth is not efficient. Better is to use DP3 to concatenate the frequencies into only a few measurement sets and image those.

7.6.3 *w*-stacking

The *w*-stacking gridder of WSClean has very different performance characteristics to for example the *w*-projection algorithm. When wsclean is executed on a machine that does not have enough memory to store all *w*-layers at once in memory, the program will make several passes over your measurement set; in the first pass it will grid and FFT only the layers with lowest w terms, in the second pass it will process the data with second-lowest *w*-terms, etc. A few passes typically do not slow down the imaging much, but if on the order of ten or more passes are executed, the performance might no longer be acceptable. In that case you can:

- Decrease the number of *w*-layers (possibly by change the phase centre to zenith); or
- Decrease the size of your image; or
- Use a machine with more memory.

WSClean makes by default use of the *small inversion optimalization* <*small_inversion*>.

In the *w*-projection algorithm, the number of *w*-layers hardly affects the speed of the algorithm. However, in *w*-stacking it is almost linear in the number of *w*-layers. Hence, you should not make this number larger than necessary. If the observation has large *w*-values, because it is far off-zenith, a large number of *w*-layers might be required. Note that the *w*-projection algorithm will in such a case have a very large *w*-kernel as well, and becomes also extremely slow. In such cases it might be better to change the phase centre before imaging: see the *chgcentre* documentation.

7.7 *w*-stacking interface

WSClean's gridder is inside a single C++ unit file and is written to be modular. Therefore, it is not hard to integrate and reuse it in other software. The gridder can do both prediction and inversion.

The gridder is only a small part of the WSClean imager, and handles the low-level placing of visibilities on the uv-grid and the involved FFTs (inversion), or the other way around; predicting visibilities from a model image. There are also higher-level interfaces to WSClean, which support calling the whole WSClean program, including cleaning and handling measurement sets etc. If you need cleaning and/or your data is in the measurement set format, it might be useful to use other interfaces instead. However, if you would like to use the low-level interface, the following information might be useful.

Since WSClean 1.9, the gridder is contained in the unit file `wsclean/wstackinggridder.cpp` and its header `wsclean/wstackinggridder.h`. The header has extensive code documentation that should give enough information about how to call the gridder. You can create the Doxygen documentation for this class with a "make doc" within the build dir (which places the info in build/doc/html/index.html, etc). The code documentation can also be found here:

- WStackingGridder class API

The gridder can be compiled with two external libraries: FFTW and Boost. To avoid a Casacore dependency, you need to define AVOID_CASACORE while compiling the gridder. There's a prediction example in wsclean/examples called "wspredictionexample.cpp)", which can be compiled with:

```
g++ -o wspredictionexample -O3 -march=native -pthread \
  -I ../../external/aocommon/include/ \
  -DAVOID_CASACORE \
  wspredictionexample.cpp \
  ../wstackinggridder.cpp \
  -lfftw3f -lfftw3 -lfftw3f_threads -lfftw3_threads \
  -lboost_date_time -lboost_system
```

This assumes you are in the `wsclean/examples` subdirectory. There's a separate makefile for this: *wsclean/examples/Makefile*.

The prediction example shows how to initialize the gridder and get visibility samples from it and includes inline documentation, so should be a good starting point for using the gridder.

7.8 Storing imaging weights

WSClean offers the option `-store-imaging-weights` (since *WSClean 2.5*) to store the imaging weights in a column named `IMAGING_WEIGHT_SPECTRUM`. Its use is very similar to the `IMAGING_WEIGHT` column that CASA uses, but WSClean stores the imaging weights per polarization and channel, instead of only per channel as CASA does. The weights are written only when this parameter is given, and are not written by default (for reasons of performance).

7.8.1 Meaning of weight values

The values in `IMAGING_WEIGHT_SPECTRUM` are relative unnormalized weights, and exclude the visibility weights that are stored in `WEIGHT_SPECTRUM`. Hence, the desired weighting is achieved by multiplying the values in `IMAGING_WEIGHT_SPECTRUM` by values in `WEIGHT_SPECTRUM`. To convert a sum of weighted values back to proper units, the weighted sum should be divided by the sum of weights. The `IMAGING_WEIGHT_SPECTRUM` values will be set to unity when requesting natural weighting (without tapers). Any kind of weighting, tapering and weighting rank-filter is taken into account in the weights.

7.8.2 Polarizations and writing

The values are written/updated during the first inversion. Currently, **imaging weight values are only updated when reordering is disabled** with `-no-reorder`. This is hopefully fixed in a later version. Because no-reordering is often slower, if you need imaging weights the best approach is to only make a dirty image with the appropriate weighting.

Weights can only be stored when either imaging a one-to-one polarization in the measurement set, or when imaging Stokes I in a circular or linearly polarized measurement set. For example, in the case the measurement set contains XX/XY/YX/YY polarizations, one can request weights when imaging any of those polarizations, or when imaging Stokes I, but not when imaging stokes Q/U/V or LL/LR/.. polarizations. Polarizations that are not imaged will not be changed. Imaging Stokes I in a set with LL/RR or XX/YY polarizations will set the weights of the two polarizations (i.e. both LL/RR or both XX/YY, respectively).

7.9 DS9 region file

The tessellation of an image into polygon-shaped geometries — named facets — can be controlled with a DS9 region file. The generic region file format is documented at http://ds9.si.edu/doc/ref/region.html, this pages describes which subset of the region file can be parsed by WSClean and DP3 for the specific use case of tessellating the sky for direction-dependent calibration.

This page provides user documentation. For in-depth technical details, please consult the class responsible for parsing the facet file.

7.9.1 Defining a facet

Facets are specified using "polygons", where the (ra,dec) coordinates of the polygon vertices are specified sequentially, i.e.

```
polygon(ra_0, dec_0, ..., ..., ra_n, dec_n)
```

Please bear in mind:

- the `(ra,dec)` coordinates should be given in degrees;
- a `polygon` can have an arbitrary number of vertices;
- the start and end vertices of the `polygon` need not to coincide;
- the vertices can be placed in arbitrary order (i.e. clockwise or counter-clockwise);

7.9.2 Adding a text label

Optionally, a polygon can be equipped with a text label and/or a specific point of interest. The text label should be specified on the same line as the `polygon` to which it should be attached and should be preceded by a `#`. A valid polygon + label definition thus looks like:

```
polygon(ra_0, dec_0, ..., ..., ra_n, dec_n) # text="ABCD"
```

In DP3, this label can be used to refer to the facet. For example, `ddecal.modelnextsteps.ABCD=[applybeam]` implies that the beam is only applied to direction `ABCD`.

7.9.3 Adding a facet point of interest

By default the facet point of interest is the centroid of the facet. To override this behaviour, a `point` can be attached to a polygon to mark a specific point of interest. This can be useful to explicitly mark the position on which direction dependent effects (DDEs) should be evaluated.

Analogous to the `polygon` definition, the `point` coordinates are provided in `(ra,dec)` in degrees. A `point` should be placed on a new line, following the polygon definition, i.e.:

```
polygon(ra_0, dec_0, ..., ..., ra_n, dec_n) # text="ABCD"
point(ra_A,dec_A)
```

Only one `point` can be attached per `polygon`. In case multiple points were specified, the last one will be used. So in the following example:

```
polygon(ra_0, dec_0, ..., ..., ra_n, dec_n) # text="ABCD"
point(ra_A,dec_A)
point(ra_B,dec_B)
```

the `point(ra_A,dec_A)` is ignored and `point(ra_B,dec_B)` will be used.

7.9.4 Example facet file

Below a full example of a facet file in DS9 region file format along with its graphical representation is shown. A user-defined point as well as a label is attached to three of the facets, the upper left facet has neither of these. Obviously, the facet definitions serve illustration purposes only and are otherwise impractical, as they are disjoint and do not cover the full image domain.

```
# Region file format: DS9 version 4.1
global color=green dashlist=8 3 width=1 font="helvetica 10 normal roman" select=1␣
↪highlite=1 dash=0 fixed=0 edit=1 move=1 delete=1 include=1 source=1
fk5

# The polygons below were created manually. They create areas around each of the image␣
↪points.
# The comments below contain the corresponding 0-based pixel coordinates.

# The 'point' values are artifical values, for testing DS9FacetFile only.
# These points are selected such that they coincide with one of the vertices that
# are not located on the boundary.

# Ra+dec@(400,64). Polygon is (379,-9) (377,91) (548,95) (551,-5).
```

(continues on next page)

(continued from previous page)

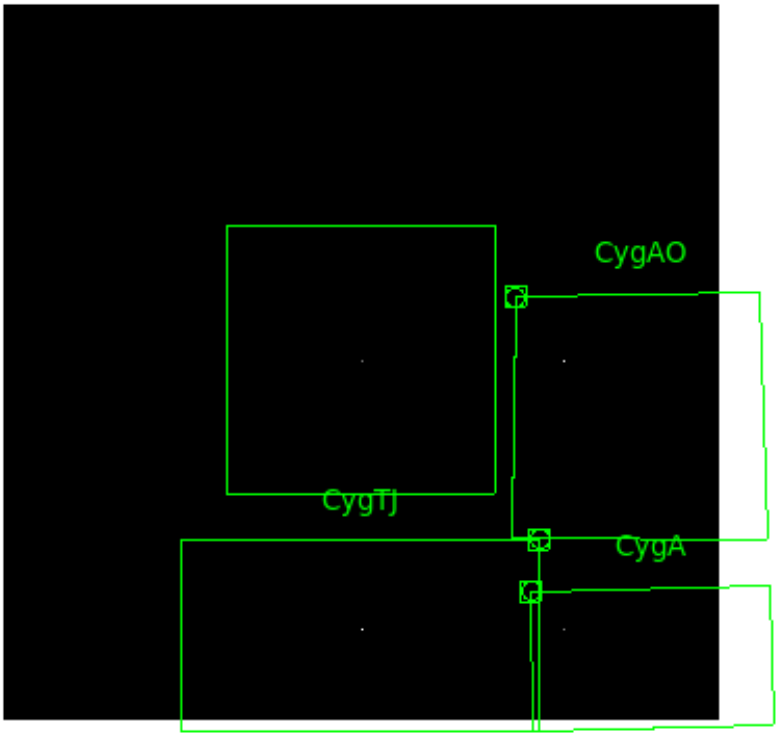
```
polygon(23.0,30.5, 23.0,31.5, 21.0,31.5, 21.0,30.5) # text=CygA
point(23.0,31.5)

# Ra@(400,256). Polygon is (364,129) (366,301) (541,305) (547,128).
polygon(23.15,31.88, 23.10,33.6, 21.0,33.6, 21.0,31.83) # text=CygAO
point(23.10,33.6)

# Dec@(256,64). Polygon is (128,-9) (128,128) (384,128) (384,-9).
polygon(25.91,30.5, 25.93,31.87, 22.92,31.87, 22.94,30.5) # text=CygTJ
# Test that the reader uses the last point when supplying multiple.
point(0, 0)
point(22.92,31.87)

# Center@(256,256). Polygon is (160,160) (160,352) (352,352) (352,160).
polygon(25.56,32.19 25.58,34.11, 23.26,34.11, 23.29,32.19)
# No explicit point -> RA() and Dec() should be nan.

# For debugging: Region that covers everything.
#polygon(33.0,27.0, 33.0,40.0, 17.0,40.0, 17.0,27.0)
```



7.10 Primary beam (component) images

As described in the *primary beam correction chapter*, in most primary-beam correcting modes, WSClean will store 16 separate images for each Mueller-matrix component (e.g. named `wsclean-beam-0.fits`, `wsclean-beam-1.fits`, ..., `wsclean-beam-15.fits`). These are difficult to interpret, as e.g. the shape that any one of these images indicates might not reflect the shape of the beam.

To turn a Mueller matrix into something that can be interpreted / validated, one option is to use the 16 images to form the Mueller matrix for each pixel (see below) and multiply this with the vector `(0.5; 0.0; 0.0; 0.5)`. This results in the full Jones sensitivity of the beam. Starting *WSClean 3.2*, only the components of the Mueller matrix that are necessary for the correction are stored, e.g. with Stokes I imaging, only element 0 and 15 are stored.

7.10.1 Mueller matrix structure

The images form the lower triangle of the Hermitian matrix. For the diagonal elements, only the real value is stored, and off-diagonal elements contain two images: the real and imaginary parts. They're counted from left to right, top to bottom, so their indices correspond to the following positions:

```
[0]
[1]+[ 2]i   [ 3]
[4]+[ 5]i   [ 6]+[ 7]i   [ 8]
[9]+[10]i   [11]+[12]i   [13]+[14]i   [15]
```

This implies for example that the diagonal elements are stored in images named like `wsclean-beam-0.fits`, `wsclean-beam-3.fits`, `wsclean-beam-8.fits`, `wsclean-beam-15.fits`, and the bottom-left entry of the matrix is split into a real part (`wsclean-beam-9.fits`) and an imaginary part (`wsclean-beam-10.fits`).

CHAPTER

EIGHT

FURTHER INFORMATION

8.1 Citing WSClean

If you use WSClean for scientific work, please cite the Offringa et al. (2014) WSClean paper. The multi-scale, multi-frequency and auto-masking algorithms are described in a second paper: Offringa & Smirnov (2017). When using *IDG*, please also cite Van der Tol, Veenboer & Offringa (2018).

8.1.1 bibtexs

bibtexs are given below.

Offringa et al. 2014

```
@article{offringa-wsclean-2014,
  author = {Offringa, A. R. and McKinley, B. and Hurley-Walker and others},
  title = {{WSClean: an implementation of a fast, generic wide-field imager for radio␣
↪astronomy}},
  volume = {444},
  number = {1},
  pages = {606-619},
  year = {2014},
  doi = {10.1093/mnras/stu1368},
  journal = {MNRAS}
}
```

Offringa & Smirnov 2017

```
@article{offringa-wsclean-2017,
  author = {Offringa, A. R. and Smirnov, O.},
  title = {{An optimized algorithm for multiscale wideband deconvolution of radio␣
↪astronomical images}},
  volume = {471},
  number = {1},
  pages = {301-316},
  year = {2017},
  doi = {10.1093/mnras/stx1547},
```

(continues on next page)

(continued from previous page)

```
  journal = {MNRAS}
}
```

Van der Tol, Veenboer & Offringa (2018)

```
@article{vandertol-2018,
  author = {Van der Tol, Sebastiaan and Veenboer, Bram and Offringa, Andr\'e R.},
  title = {Image Domain Gridding: a fast method for convolutional resampling of␣
↪visibilities},
  DOI= "10.1051/0004-6361/201832858",
  url= "https://doi.org/10.1051/0004-6361/201832858",
  journal = {A\&A},
  year = 2018,
  volume = 616,
  pages = "A27",
}
```

Other resources:

- Debian WSClean tracker

  Lists the current status of the WSClean Debian package.

- LOFAR Imaging Cookbook

  Provides LOFAR-specific info.

- Offringa et al. 2014

  The original WSClean paper.

- Offringa & Smirnov 2017

  The MF multi-scale clean & auto-masking algorithms.

- Van der Tol, Veenboer & Offringa (2018)

  IDG overview paper.

- Veenboer, Petschow & Romein (2018)

  IDG on Graphics Processors.

- Offringa et al. (2019)

  Assessment of gridding accuracy for 21-cm Epoch of Reionization experiments.

See also: *Citing WSClean*

To contact me, mail me on offringa at gmail dot com.

Many people have greatly helped the development of WSClean by submitting bug reports, feature requests, ideas and patches, for which I am very grateful. Bas van der Tol and Bram Veenboer have in particular contributed a lot to WSClean.

CHAPTER

NINE

NAVIGATE

- genindex
- search