# SoFiA 2 User Manual

## v2.5.1

Tobias Westmeier

September 5, 2023

SoFiA
Source Finding Application

# Contents

# Introduction

With the new generation of large-scale, blind HI surveys to be carried out on the next generation of centimetre-wave radio telescopes, including ASKAP, MeerKAT, WSRT/Apertif and FAST, fully automated, accurate and reliable source finding is of fundamental importance to the scientific success of these surveys. The Source Finding Application, SoFiA (Serra et al. 2015), was designed to meet the source finding needs of the HI community and has become the de-facto standard source finding tool for spectral-line data.

While SoFiA introduced a large number of novel algorithms that have revolutionised HI source finding, there are a few issues that affect its applicability to large data volumes and its long-term maintainability:

- Large parts of SoFiA are written in Python, resulting in needlessly large execution times and memory consumption in addition to imposing a range of restrictions.
- SoFiA depends on a large number of third-party libraries which tend to break the code due to the lack of compatibility between different library versions.
- Different parts of SoFiA are written in different programming languages, including Python, C++ and Cython, making the software difficult to maintain.

In order to address these problems, most notably the speed and memory issues, the SoFiA team decided to reimplement the most critical and powerful components of the software in the C programming language. This new version has been named SoFiA 2 (Westmeier et al. 2021) and, like its predecessor, has been made available on GitLab.[1]

## Installation

### Standard installation procedure

Installation of the SoFiA 2 source code from GitLab is straightforward, as there is currently only a single dependency, WCSLIB,[2] which is required for converting between pixel-based and world coordinates. In addition, the GNU C compiler (GCC) must be available, although other GCC-compatible compilers supporting the C99 standard, including CLANG, might work as well. SoFiA 2 can then simply be compiled by executing the `compile.sh` shell script in the base directory. This will compile the source code and generate an executable file named `sofia`. Multi-threading support can be enabled by adding the `-fopenmp` flag to the `compile.sh` script call.[3]

It is important to ensure that no error messages are produced by the compiler during the compilation process, as otherwise SoFiA 2 will not have been compiled correctly and may not run. In addition, the instructions shown at the end of the installation process should be followed to set a global alias or symbolic link to the `sofia` executable.

SoFiA 2 makes extensive use of multi-threading using OpenMP. As OpenMP support is natively built into the GCC compiler, no additional libraries or dependencies are needed, and multi-threading is enabled by default. If not otherwise specified, SoFiA 2 will use the environment variable `OMP_NUM_THREADS` to control the number of threads used by OpenMP. If unset, this will normally default to using all CPU cores available on a machine, thereby minimising the runtime of the pipeline. The maximum number of cores utilised by SoFiA 2 can be explicitly set using the `pipeline.threads` option. This may be desirable in certain situations to reduce the CPU load caused by SoFiA 2. With a code parallelisation fraction of just over 80%, the use of about 8 cores is optimal, with no significant gains expected beyond that.

[1] SoFiA 2 on GitLab: `https://gitlab.com/SoFiA-Admin/SoFiA-2/`.

[2] WCSLIB on Mark Calabretta's website: `https://www.atnf.csiro.au/people/mcalabre/WCS/`.

[3] The `-fopenmp` flag is used by GCC. Other compilers may use a different flag for enabling OpenMP; please see your compiler documentation for more information.

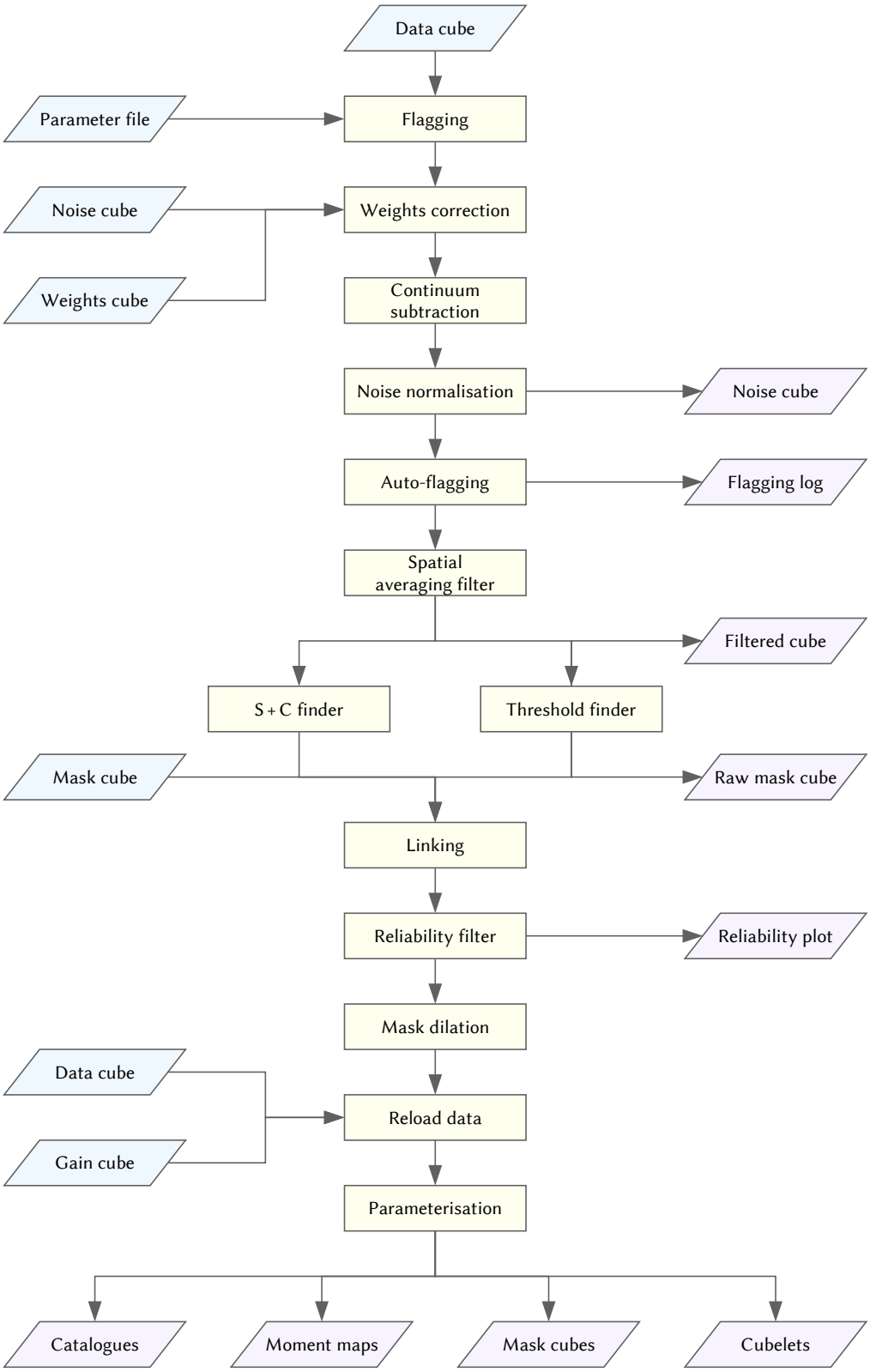


**Figure 1**: Flowchart of SoFiA 2 showing the individual steps in the pipeline (yellow), the input data cubes accepted by SoFiA 2 (blue) and the different output files that can be generated (purple).

### Container formats

In addition to the standard installation procedure from source code, all stable releases of SoFiA 2 are also supplied in the form of **Docker** containers. These are available for download from Docker Hub.[4] Due to the significant overhead imposed by the containerisation process, it is strongly advisable to install SoFiA 2 from source code by default, which is generally easier and will occupy significantly less disc space. Docker containers should only be used in special circumstances where installation from source is not possible or a container format is preferred for some other reason. As Docker images are natively supported by **Singularity**, the official SoFiA 2 Docker images can also be run in Singularity. Further information is available from the Singularity website.

## Parameter files

SoFiA 2 is controlled through so-called parameter files that are used to select or deselect the different algorithms offered by SoFiA 2 and control their individual settings. As such, parameter files are simple text files made up of one or more parameter settings each of which must be on a separate line. Each parameter setting must be of the following form:

```
module.parameter = value
```

An arbitrary number of whitespace characters is allowed around the assignment operator (=) and at the beginning or end of each line; these will be ignored by the parser. Empty lines and any character sequence starting with a hash character (#) will also be ignored and be treated as a comment. A complete list of control parameters accepted by SoFiA 2 can be found in Appendix A. As unset parameters will automatically assume their default values, users may opt to specify only a subset of relevant settings in their parameter files, most notably the name of the input data set.

A basic example parameter file for a standard source finding run on a 3D extragalactic HI data cube is presented in Section 10.1.

## Running the pipeline

Once a parameter file has been created and saved, the pipeline can be launched from the terminal by running the following command:

```
sofia <parameter_file>
```

where <parameter_file> is the name of the SoFiA 2 parameter file. If the parameter file has been saved to the current directory, it will be sufficient to provide just its name, otherwise the full path will need to be included. A flowchart of the SoFiA 2 pipeline is presented in Fig. 1.

Note that it is not necessary to define all existing parameters in the user parameter file. Any parameters not defined will simply assume their default value. It is therefore usually sufficient to set only a few parameters that need to be adjusted, such as the name of the input data cube. It is also possible to specify individual settings on the command line in the following way:

```
sofia <parameter_file> <parameter>=<value>
```

which will first read the user parameter file and then update the specified parameter setting. Note that there must be no space in between <parameter> and <value> when specifying parameter settings directly on the command line. Any number of parameter files and/or individual settings can be passed on to SoFiA 2 on the command line in any order. All files and direct settings will simply be read and processed in the order in which they are provided, with later settings overwriting earlier ones. This could for example be used to define default parameter settings in a separate file while specifying the input data cube name on the command line, e.g.

[4] SoFiA 2 Docker images: https://hub.docker.com/r/sofiapipeline/sofia2/.

```
sofia default.par input.data=data.fits output.filename=test
```

would first read the parameter file `default.par` and then replace the input and output file names with `data.fits` and `test`, respectively. If a parameter setting supplied on the command line contains special terminal control characters or whitespace, then the setting must be enclosed in quotation marks, e.g. `input.data="sam's galaxy.fits"`.

# Input data sets

SoFiA 2 accepts several kinds of input files, including the data cube to be searched. Currently, only the Flexible Image Transport System (FITS) format (Pence et al. 2010) is supported by SoFiA 2, and all input and output imaging data files must be standard FITS files.

## Data cube

Data cubes are generally expected to be three-dimensional, with the first two axes containing the spatial coordinates (either equatorial or Galactic coordinates) and the third axis containing the spectral coordinates (either frequency or velocity). SoFiA 2 can also process two-dimensional images (e.g. radio continuum images), in which case the third axis is assumed to have a size of 1. Four-dimensional cubes will also be accepted as long as the fourth axis has a size of exactly 1. The fourth axis (e.g. Stokes I) will simply be dropped in this case to obtain a three-dimensional data structure.[5]

It is possible to process only a subregion of the full data cube with SoFiA 2. This can be useful if the full cube is too large to fit into memory, or if parts of the cube contain artefacts. The `input.region` option can be used in this case to specify a subregion in pixel coordinates (0-based), and only the specified region will then be read into memory and processed. It is important to note that all output, such as source centroids in the output catalogue, will be specified with respect to the subregion rather than the full cube unless `parameter.offset` is set to `true`.

To ease the processing of data cubes with negative signals, such as HI absorption lines, SoFiA 2 offers the possibility of inverting the data cube prior to processing. This can be controlled with the `input.invert` option. It should be noted that enabling this option will simply multiply the data cube by $-1$ such that positive signals become negative and vice versa. Hence, all flux-related output, such as the flux measurement or $0^{\text{th}}$ moment, will be inverted as well in this case and be reported as positive.

## Mask cube

In addition to the input data cube, an input mask cube can be loaded into SoFiA 2. Mask cubes must have the same dimension as the input data cube and are used to mark with a non-zero value all pixels that are considered to be part of a source. SoFiA 2 will normally generate a fresh mask cube during source finding. However, the `input.mask` option can be used to load an existing mask cube instead, e.g. from a previous run of SoFiA 2. Any additional pixels detected by the source finder will then simply be added to the existing mask.

A useful application for supplying a mask cube would be to use SoFiA 2 simply as a parameteriser rather than a source finding pipeline. In this case the source finder can be disabled, but the linker must remain switched on, as otherwise SoFiA 2 would have no information on how many source there are within the mask and where they are located.

## Noise cube

Using the `input.noise` option, a noise cube can be provided to SoFiA 2 for the purpose of normalising the noise level across the data cube in situations where the noise varies either spatially or spectrally. SoFiA 2 will divide the data cube by the noise cube prior to source finding. A typical use case would be to enable local noise normalisation in SoFiA 2 (see Section 3.3) and write out the resulting noise cube. That noise cube can then be loaded in subsequent runs of SoFiA 2, alleviating the need to run the computationally expensive noise normalisation algorithm over and over again. Note that source parameterisation will be

[5] Currently, four-dimensional cubes where the third axis has a size of 1 while the fourth axis is larger than 1 are also supported; in this case, SoFiA 2 assumes that the fourth axis contains the spectral information, and the header elements for the third and fourth axis are simply swapped. This 'feature' may get removed in future releases.

carried out on the data cube without noise normalisation to ensure that the measured flux densities are correct.

### Weights cube

Using the `input.weights` option, a weights cube can be provided. SoFiA 2 will multiply the data cube by the square root of the weights cube prior to source finding. Note that source parameterisation will be carried out on the data cube without application of the weights cube to ensure that the measured flux densities are correct. It is possible to load and apply both a weights cube and a noise cube, but SoFiA 2 will issue a warning in this case.

### Gain cube

Some data cubes, such as radio-interferometric mosaics, may be affected by spatial or spectral gain variations. If uncorrected, these would result in parameterisation errors, in particular with respect to flux-based parameters. SoFiA 2 allows the user to specify a gain cube that can be used to correct for gain variations prior to source parameterisation. For this purpose, the input data cube will be divided by the gain cube to normalise the gain to 1 across the entire cube. Unlike noise variations, gain variations do not necessarily affect source finding as such, and the gain cube will therefore only be applied after source finding and before parameterisation.

# Preconditioning

Before a source finding algorithm can be applied to a data cube, it may be necessary to subject the cube to a range of preconditioning steps, e.g. to remove artefacts that would otherwise be picked up by the source finder, or to ensure that the noise level across the data cube is constant prior to applying a source finding threshold. SoFiA 2 currently provides several precondition options that are outlined in this section.

## Data flagging

Data affected by interference or artefacts may need to be flagged prior to source finding. SoFiA 2 currently provides three methods of flagging affected regions of the data cube. If the location of the interference or artefacts is known, the `flag.region` option can be used to provide SoFiA 2 with a list of spatial and spectral regions that need to be flagged. These need to be specified in units of pixels (0-based). The relevant flagging will then be applied once the data cube has been loaded. It is important to note that, if applicable, flagging regions are expected to be specified relative to the subregion being processed, not the full data cube.

Another manual flagging option offered by SoFiA 2 allows the user to provide a catalogue file using the `flag.catalog` option. The file must contain two columns with the coordinates of sky positions in the native coordinate system and units of the input data cube (e.g. right ascension and declination in decimal degrees). Columns can be separated by spaces, tabulators or commas, and entire lines can be commented out using the `#` character. A circular region around each position, if located within the area covered by the data cube, will then be flagged prior to source finding. The radius of the circular flagging region can be controlled with the `flag.radius` option. The catalogue-based flagging option is useful for masking the locations of residual continuum sources which can result from inadequate continuum subtraction.

In addition to manual flagging, SoFiA 2 also offers an automatic flagging mode for spectral channels and/or spatial pixels affected by interference. This mode can be controlled using the `flag.auto` option. The automatic flagging algorithm works by first measuring the RMS noise level, $\sigma_{\mathrm{rms}}(i)$, in each spectral channel or spatial pixel, $i$. It then calculates the median, $\mu$, of the RMS values and the median absolute deviation, $\tilde{\mu}$, from the median, which are used as proxies for the mean noise level and the standard deviation about that noise level, respectively. Lastly, a user-specified threshold, $\vartheta$, is applied (`flag.threshold`) such that all channels or pixels, $i$, where

$$|\sigma_{\mathrm{rms}}(i) - \mu| > \mathrm{C}\,\vartheta\,\tilde{\mu} \tag{1}$$

will be flagged. A constant of $\mathrm{C} \approx 1.4826$ is used in Eq. 1 to convert the median absolute deviation into a regular standard deviation under the assumption that the underlying scatter in the RMS values follows a Gaussian distribution. Flagged channels and pixels can be written to a log file by enabling the `flag.log` option.

Care should be taken to ensure that the data cube does not contain regions of extended emission (e.g. from the Milky Way) as channels or pixels with extended emission would potentially be flagged by the algorithm. Likewise, if there is significant spatial or spectral variation of the noise across the data cube, the noise normalisation filter (Section 3.3) must be enabled as well, as otherwise channels or pixels in regions of intrinsically higher noise level would potentially get flagged.

## Continuum subtraction

If the input data cube suffers from the presence of low-level residual continuum emission, then SoFiA 2 can help with its removal by switching on the `contsub.enable` option. SoFiA 2 will then use a robust algorithm to fit a low-order polynomial to the spectrum at each spatial position of the cube and subtract that polynomial from the data. The polynomial order can be controlled via the `contsub.order` parameter. Currently, only order 0 (constant offset) and order 1 (offset + slope) are supported. It should be noted that continuum subtraction will only occur prior to source finding, whereas source parameterisation will occur on the original data cube without continuum subtraction.

The algorithm will extract the spectrum at each spatial position in the data cube. It will then shift the spectrum symmetrically by ±`contsub.shift` channels and subtract the spectrum shifted in one direction from the spectrum shifted in the opposite direction. Next, the noise in the subtracted spectrum will be measured, and all channels in which the absolute value of the signal exceeds `contsub.threshold` times the noise level will be flagged and excluded from the polynomial fitting. An additional padding of ±`contsub.padding` channels around flagged channels can be applied as a safety margin. Finally, a polynomial is fitted to the remaining channels of the original spectrum and subtracted from the original data cube.

The polynomial fitting algorithm is fairly robust and should not normally be affected by spectral-line emission or other artefacts irrespective of how bright they are. However, it is important to ensure that a substantial fraction of the band is free from line emission and artefacts. As a general guideline, these should not extend across more than about 20% of the total bandwidth, as otherwise the continuum fit will start to be affected by their presence. Another problem could arise from emission near the edge of the spectrum or very broad, smooth emission that might not be filtered out by the algorithm. In such cases, increasing the values of `contsub.shift` and `contsub.padding` might help to improve the fit.

## Noise normalisation

Another important preconditioning filter offered by SoFiA 2 is noise normalisation, which is required in situations where the noise level varies across the data cube, either along the spectral axis or in the spatial domain. As SoFiA 2 will be applying a source finding threshold to the data, it will be necessary in such cases to first divide the data cube by the local noise level to remove any variation.

SoFiA 2 provides two methods for correcting noise variations. If the noise varies only as a function of frequency, but not spatially, SoFiA 2 can measure and correct the noise on a channel-by-channel basis by setting the option `scaleNoise.mode = spectral`. If the noise varies across the spatial plane as well, e.g. in an interferometric mosaic, then local noise scaling can be enabled by setting `scaleNoise.mode = local`.

Local noise scaling operates by shifting a running window of a given size (`scaleNoise.windowXY` and `scaleNoise.windowZ`) across all three dimensions of the data cube on a grid that can also be specified by the user (`scaleNoise.gridXY` and `scaleNoise.gridZ`). If no grid size is specified, then SoFiA 2 will use half the window size by default. It will then measure the RMS noise level in each window and divide the data in the corresponding grid cell by the noise value. In addition, interpolation can be enabled (option `scaleNoise.interpolate`) to linearly interpolate the noise values in between the grid centres, thereby avoiding the sharp boundaries between grid cells that would otherwise occur.

The resulting noise measurement can be exported by setting `output.writeNoise = true`. In the case of local noise scaling, a FITS cube with the suffix "`_noise.fits`" will be created that contains the measured local noise level across the cube. If spectral noise scaling is enabled, then the measured noise per channel will be written to a plain text file with the suffix "`_noise.txt`".

Note that it is in principal also possible to achieve noise normalisation in each spectral channel using the local noise scaling algorithm by simply setting the spectral window and grid size to 1 and the spatial window and grid size to a very large number greater than the cube size, e.g. 99999. In this case the local noise scaling algorithm will behave in the same way as the spectral noise scaling algorithm, except that a noise cube rather than a noise spectrum would be produced. Abusing the local noise scaling algorithm in this way is not recommended, though, as the spectral noise scaling algorithm will always be more efficient and thus much faster.

## Ripple filter

Occasionally, data cubes are affected by low-level artefacts that are spatially and spectrally extended, but somewhat variable across the sky or with frequency, for example a spectral ripple caused by solar interference. Such weak features can get elevated above the sensitivity threshold after spatial and spectral

smoothing in the S+C finder, potentially resulting in false detections. SoFiA 2 offers a special ripple filter that can be applied to remove such artefacts, if necessary, by activating `rippleFilter.enable`.

The filter operates by calculating either the mean or the median (`rippleFilter.statistic`) across a running window, the spatial and spectral size of which can be set by the user (`rippleFilter.windowXY`, `rippleFilter.windowZ`). Likewise, the spatial and spectral step by which the window is moved can be controlled (`rippleFilter.gridXY`, `rippleFilter.gridZ`); by default it will be set to half the window size. The mean or median is then subtracted from every individual pixel within the grid cell, unless interpolation is enabled (`rippleFilter.interpolate`), in which case the values will first be linearly interpolated in between grid cells before being subtracted from the data.

While the filter is fairly robust against outliers and the presence of genuine sources by using the median in the averaging process by default, it should be used with great caution, as it has the potential to remove genuine astronomical signal from the data cube. There is a particularly high risk in the presence of spatially or spectrally extended astronomical sources such as nearby galaxies. The user is responsible for defining sufficiently large spatial and spectral window sizes to ensure that even the most extended sources expected to be present in the data cube do not affect the median value within the window.

The effect of the ripple filter on the data can in general be assessed by writing out the filtered cube using the `output.writeFiltered` option. Comparing the filtered cubes with and without the ripple filter enabled would allow the contribution from the filter to each pixel of the data cube to be exactly quantified if necessary.

# Source finding

SoFiA 2 comes with two source finding algorithms: a simple threshold finder and the smooth + clip finder (S + C finder). The threshold finder is rather basic and simply adds all those pixels to the source mask that have an absolute flux density in excess of the specified threshold. Note that this explicitly includes significant negative flux densities to avoid creating a positive bias. The threshold can be either absolute or relative to the global RMS noise level in the data cube. The threshold finder is useful if a pre-filtered cube is to be searched for signal exceeding a certain level of significance.

## The S + C finder

The second algorithm, the S + C finder, is the most sophisticated source finding algorithm in SoFiA 2 and the one that should normally be applied to data cubes for the purpose of blind source finding. The S + C finder works by spatially and spectrally smoothing the original data cube on multiple scales as defined by the user. In each smoothing iteration, a user-specified flux threshold relative to the global RMS noise level after smoothing is applied to the data, and all pixels with an absolute flux density exceeding that threshold will be added to the source mask. Note that this will extract pixels with both positive and negative flux density to remove the bias that would otherwise result from just considering positive flux densities.

By smoothing the data on the relevant scales, the signal-to-noise ratio of sources with sizes matching the smoothing scales will be maximised, making it possible to detect even faint, extended emission that would remain below the noise level in the non-smoothed data cube. The signal-to-noise ratio of any source will be maximal when the spatial and spectral convolution filter size matches the spatial and spectral extent of the source, and it is the user's responsibility to select the appropriate filter sizes for the kind of sources expected to be present in the data.

SoFiA 2 will apply a Gaussian filter in the spatial domain and a boxcar filter in the spectral domain to accommodate the fact that typical extragalactic HI sources, such as distant galaxies, have an exponential radial surface brightness profile combined with a double-horn spectral profile with steep flanks. The FWHM of the spatial Gaussian kernels are specified with the `scfind.kernelsXY` option and are in units of pixels. The Gaussian filter is assumed to be symmetric in the two spatial dimensions. The widths of the spectral boxcar filters are specified with the `scfind.kernelsZ` option and define the full width of the boxcar filter in units of channels. Spectral boxcar filter sizes must be odd (3, 5, 7, ...).

All filters from the spatial kernel list will be combined with all of the filters from the spectral kernel list, and $N$ spatial filters combined with $M$ spectral filters will therefore result in $N \times M$ different combinations of filters being applied. For each combination of spectral and spatial smoothing kernels, the algorithm will operate as follows:

1. Create a copy of the original data cube.
2. In the data cube copy, replace the values of all pixels already detected in a previous iteration with `scfind.replacement` times the RMS in the original cube. This is to ensure that smoothing on large scales does not smear out the emission over too large a region, as the resulting source mask would otherwise extend far beyond the edge of the source.
3. Set all blanked pixels to a value of 0 prior to smoothing.
4. Convolve the data cube copy with the next set of spatial and spectral filters.
5. Change all originally blanked pixels from 0 back to blank.
6. Measure the RMS noise level in the smoothed copy.
7. In the source mask, mark all pixels as detected that have a flux density in the smoothed copy of more than `scfind.threshold` times the smoothed RMS noise level.

8. Delete the smoothed copy of the data cube again and continue with item 1 until all combinations of spatial and spectral smoothing kernels have been applied.

At the end of this process, the source mask will contain all pixels that exceeded the source finding threshold on at least one of the applied smoothing scales. The next steps will be to group the detected pixels into coherent sources and discard those that are deemed spurious based on simple size thresholds.

A few important things should be noted in relation to the S+C finder. First of all, it is important to specify all smoothing kernel sizes, both spatial and spectral, in the order of increasing size. This is crucial, as otherwise the replacement mechanism for previously detected pixels would fail to prevent the S+C finder from smearing out the source emission over too large an area. In particular, it is advisable to always specify a kernel size of 0 as the first spatial and spectral kernel.

Another thing to note is that Gaussian spatial smoothing in SoFiA 2 is approximated by a series of boxcar filters for reasons of speed. The algorithm can therefore in principle not approximate spatial filters that are smaller than 3 pixels in size, and the smallest sensible spatial filter size (apart from 0) should therefore be $\geq 3$.

# Linking of detected pixels

Before a source catalogue can be produced, the pixels recorded in the source mask will need to be grouped into individual sources. This is achieved in SoFiA 2 by applying a basic friends-of-friends algorithm based on merging lengths chosen by the user.

The algorithm will loop over the mask cube until it detects a pixel that has been marked as a detection, but not yet associated with a source. That pixel will be assigned a new source ID, and the algorithm will then recursively search all of the neighbouring pixels within a certain merging length for additional detected pixels, all of which will be assigned the same source ID. Once the recursion ends, all pixels belonging to that source will have been correctly labelled, and the algorithm will move on to link the next source.

The user can control the merging lengths in the spatial and spectral dimensions of the cube by setting the `linker.radiusXY` and `linker.radiusZ` options. The actual merging volume is assumed to be an ellipsoid of the form

$$\frac{(x-x_0)^2}{r_{xy}^2}+\frac{(y-y_0)^2}{r_{xy}^2}+\frac{(z-z_0)^2}{r_z^2}\leq 1 \tag{2}$$

where $(x_0, y_0, z_0)$ is the centre pixel the neighbours of which are to be checked, and $r_{xy}$ and $r_z$ are the user-selected merging radii. All pixels $(x, y, z)$ that fulfil the inequality above will be considered as neighbours of pixel $(x_0, y_0, z_0)$ that need to be merged.

During the process of linking, SoFiA 2 will immediately reject linked sources that fall below (or above) user-specified minimum (or maximum) size criteria, thereby providing a basic method of removing potentially false detections caused by noise peaks or artefacts. The minimum size requirement for sources can be specified with the `linker.minSizeXY` and `linker.minSizeZ` options for the spatial and spectral dimensions, respectively. Likewise, the corresponding `linker.maxSizeXY` and `linker.maxSizeZ` options can be used to specify the maximum size requirement. Likewise, sources can be discarded based on the total number of spatial and spectral pixels they contain (`linker.minPixels`, `linker.maxPixels`) or their filling factor withing their rectangular bounding box (`linker.minFill`, `linker.maxFill`) which is defined as the fraction of pixels within the bounding box that are part of the source.

Lastly, the linker will set a quality flag to indicate if a source is located near the edge of the cube or near blanked pixels. The flag is a simple integer number that will take values of 0 (no issues), 1 (near spatial edge of cube), 2 (near spectral edge of cube) and 4 (near blanked pixels). The flags are set whenever the respective structure is located within the merging radius of the source, even if the mask does not directly touch the edge or any blanked pixels. Flag values are additive, e.g. a value of 5 means that the source is at the spatial edge of the cube and near blanked pixels at the same time.

Note that the linker can be disabled by setting `linker.enable = false`. In this case, the pipeline will terminate after source finding, and no catalogue or source data products will be created. Disabling the linker will therefore only be useful if the user is merely interested in the raw mask produced by the source finder (setting `output.writeRawMask = true`).

# Reliability calculation

Reliability calculation provides a way of automatically determining the reliability of detections and, by setting a simple reliability threshold, discarding all sources that are deemed unreliable. This allows very low source finding thresholds in the range of 3 to 4 times the RMS noise level to be applied without the risk of having to deal with a large number of false detections in the resulting source catalogue.

Reliability calculation in SoFiA 2 is based on the method described by Serra et al. (2012). For this to work, we must make the fundamental assumption that a data cube contains stochastic noise with a normal distribution centred on zero plus the astronomical signal that we are interested in. We further assume that all astronomical signal has positive flux. If we then apply a threshold, $S_{\mathrm{thresh}}$, to the data cube to detect all signals with $|S_i| > S_{\mathrm{thresh}}$, we can conclude that all signals with negative flux must be noise peaks, whereas signals with positive flux can either be noise peaks or genuine astronomical signal.

## Algorithm

SoFiA 2's reliability calculation works by comparing the density of positive and negative detections in a specific parameter space, be default $(s_{\mathrm{max}}, s_{\mathrm{sum}}, s_{\mathrm{mean}})$, where $s_{\mathrm{max}} = S_{\mathrm{max}}/\sigma_{\mathrm{rms}}$ is the peak flux density, $s_{\mathrm{sum}} = S_{\mathrm{sum}}/\sigma_{\mathrm{rms}}$ is the summed flux density and $s_{\mathrm{mean}} = s_{\mathrm{sum}}/n_{\mathrm{pix}}$ is the mean flux density across the source, which is equal to the sum divided by the number of pixels, $n_{\mathrm{pix}}$. All three parameters are normalised by the RMS noise level, $\sigma_{\mathrm{rms}}$, of the data cube.[6]

Genuine detections are generally expected to populate a different region of parameter space than false detections caused by noise peaks. By comparing the density of positive and negative detections in different regions of parameter space, we can therefore estimate the probability for any of the positive detections in that region to be genuine,

$$R = \begin{cases} \frac{n_{\mathrm{pos}} - n_{\mathrm{neg}}}{n_{\mathrm{pos}}} & \text{for } n_{\mathrm{pos}} \geq n_{\mathrm{neg}}, \\ 0 & \text{otherwise,} \end{cases} \tag{3}$$

where $n_{\mathrm{pos}}$ and $n_{\mathrm{neg}}$ are the number of positive and negative detections in that region, respectively. If we only have positive detections, then $R = 1$, and all sources are deemed fully reliable. If, on the other hand, we have an equal number of positive and negative detections, then $R = 0$, and we have no reason to assume that any of the positive detections in that region are genuine. Note that, as $R < 0$ for $n_{\mathrm{neg}} > n_{\mathrm{pos}}$, we simply set $R = 0$ in this case.

The density of positive and negative detections is measured by performing a Gaussian kernel density estimation (KDE) on the individual data points in parameter space. The optimal size of the the three-dimensional Gaussian kernel is determined from the covariance matrix of the negative detections, which are assumed to have a three-dimensional Gaussian distribution, as can be expected from false detections caused by stochastic noise. The relative size of the Gaussian kernel used in the KDE can be controlled by applying a constant scale factor using the `reliability.scaleKernal` option. The densities of the positive and negative detections resulting from this analysis are then fed into Eq. 3 to determine the reliability of all positive detections.

An example of the reliability analysis is shown in Fig. 2, where the dense cluster of positive and negative detections marks the region in parameter space that is occupied by false detections due to noise. The black sources in the upper-right corner are positive sources with more than 90% reliability that are well separated from the cluster of noise peaks. SoFiA 2 produces such plots for diagnostic purposes if requested by the user (option `reliability.plot`).

## Parameter space

By default, SoFiA 2 uses a three-dimensional parameter space made up of peak flux density, summed flux density and mean flux density (all of which are divided by the global RMS noise level and then logarith-

[6] If the noise varies across the cube, prior noise normalisation is required to ensure that the noise level is constant.

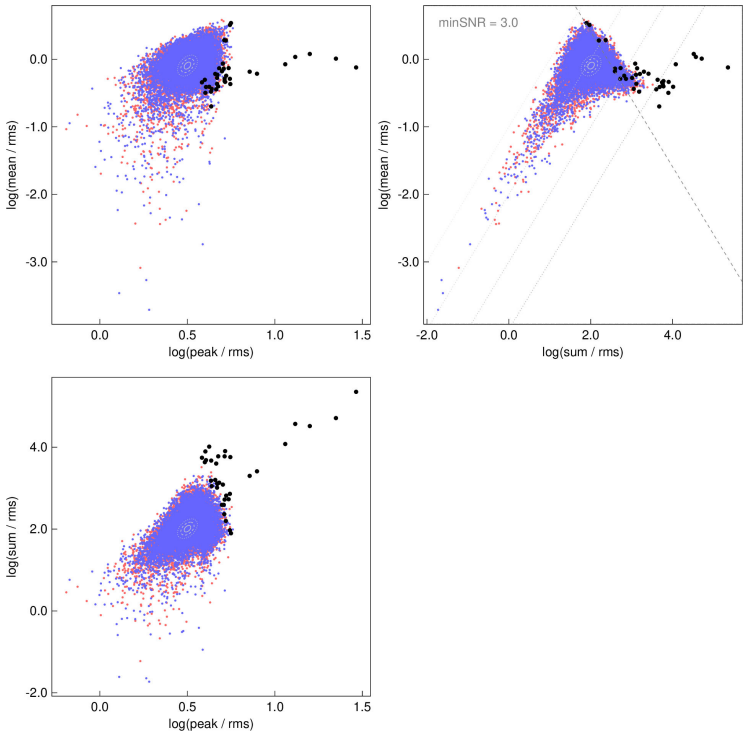


**Figure 2:** 2D projection of the distribution of positive (red) and negative (blue) detections in the 3D parameter space used by SoFiA 2 for determining the reliability of sources. The black dots mark positive sources that have been deemed more than 70% reliable. The light-grey ellipses are $1\sigma$, $2\sigma$ and $3\sigma$ contours of the 3D Gaussian function used in the kernel density estimator, while the dark-grey, dashed line marks the signal-to-noise limit below which sources are generally discarded irrespective of their reliability.

mised) to determine the density of positive and negative detections in parameters space for the purpose of reliability calculation. However, the parameters to be used and the dimensionality of the parameter space can be controlled by the user with the `reliability.parameters` option. This option expects a comma-separated list of parameter names, the total number of which defines the dimensionality of the parameter space to be used. It should be noted that this option is intended for expert users, and casual users of SoFiA 2 are strongly advised to use the default parameter space instead. Possible parameters that can be supplied to `reliability.parameters` include:

- `peak` – Logarithm of the peak flux density divided by the global RMS noise level.
- `sum` – Logarithm of the summed flux density divided by the global RMS noise level.
- `mean` – Logarithm of the mean flux density divided by the global RMS noise level.
- `pix` – Logarithm of the total number of spatial and spectral pixels.
- `chan` – Number of spectral channels.
- `fill` – Logarithm of the filling factor, i.e. the number of spatial and spectral pixels within the rectangular source bounding box that have been detected.
- `std` – Standard deviation.
- `skew` – Skewness.
- `kurt` – Kurtosis.

All parameters are derived across the three-dimensional source mask. It is generally not advisable to increase the dimensionality of the parameter space beyond 3–5, as the density of detections in a higher-dimensional space will decrease rapidly to the point where a meaningful reliability can no longer be calculated within the size of the Gaussian kernel, as there may be too few detections. Likewise, the kernel size will need to be adjusted whenever the dimensionality of the parameter space is changed (see Section 6.4).

## SNR and pixel thresholds

The dashed, grey line in the right-hand panel of Fig. 2 represents a line of constant integrated signal-to-noise ratio, $\mathrm{SNR}_{\mathrm{min}}$, and is controlled by the `reliability.minSNR` option. All detections that fall below that line, i.e. detections where

$$\frac{\sum S_i}{\sigma_{\mathrm{rms}}\sqrt{N_{\mathrm{pix}}\Omega_{\mathrm{PSF}}}} < \mathrm{SNR}_{\mathrm{min}}, \tag{4}$$

are automatically removed from the catalogue and assigned a reliability of zero by default. Here, $\sum S_i$ is the sum of the flux densities across the source mask, $\sigma_{\mathrm{rms}}$ is the global RMS noise level of the data cube (assumed to be constant), $N_{\mathrm{pix}}$ is the total number of spatial and spectral pixels forming the detection, and $\Omega_{\mathrm{PSF}}$ is the beam solid angle (in pixels).

**⚠ WARNING**

For the `reliability.minSNR` threshold to be meaningful, the relevant **WCS and beam information** must be available from the FITS header (keywords `BMAJ`, `BMIN` and `CDELT1`), and the beam must not vary with position or frequency. If no beam information is available then SoFiA 2 will print a warning and implicitly assume that $\Omega_{\mathrm{PSF}} = 1$, i.e. the beam size equals the pixel size.

In the same way as the signal-to-noise threshold, the user can choose to set a minimum threshold for the total number of spatial and spectral pixels within the source mask using the `reliability.minPixels` option. Any detections with fewer pixels in their mask will be considered as unreliable by default and will have their reliability set to zero.

## Kernel size optimisation

Another useful diagnostic plot produced by the reliability module is the Skellam plot, as it can be used to asses if the kernel scale factor is optimal. If the kernel is too small then too few positive and negative detections will contribute to the reliability calculation, resulting in large statistical uncertainties and thus inaccurate reliability values. If, on the other hand, the kernel is too large then negative detections could influence the reliability of genuine sources, thus reducing the completeness of the source catalogue produced by SoFiA 2.

The Skellam plot (generated if `reliability.plot = true`) can help with choosing an optimal kernel size by showing the cumulative distribution of the Skellam parameter,

$$K = \frac{n_{\mathrm{pos}} - n_{\mathrm{neg}}}{\sqrt{n_{\mathrm{pos}} + n_{\mathrm{neg}}}}, \tag{5}$$

evaluated at the positions of all negative detections. SoFiA 2 will then normalise the resulting Skellam parameter values such that their standard deviation from the median becomes 1, allowing their distribution to be compared to a standard Gaussian. If the kernel size is optimal, then the cumulative distribution of the normalised values of $K$ should be the same as that of a Gaussian of standard deviation $\sigma = 1$. If the kernel is too small, then the median, $\mu$, of the distribution will be shifted into the negative range, while a kernel that is too large would result in a shift into the positive range. An example Skellam plot for a well-matched kernel is shown in Fig. 3. In this example, the median of $\mu = -0.053$ is very close to zero, indicating that the kernel scale factor of $0.4$ chosen by the user is optimal.

As an alternative to manual optimisation of the reliability kernel, SoFiA 2 also offers a method of automatically adjusting the kernel to an optimal size. This auto-kernel feature can be enabled by setting `reliability.autoKernel = true`. In this case, SoFiA 2 will run the reliability analysis repeatedly until the absolute value of the median of the renormalised Skellam distribution decreases below a user-defined tolerance of `reliability.tolerance`. The algorithm will start with an unreasonably small kernel scaling factor of $0.1$ and then gradually increase the scaling factor in each iteration until the median

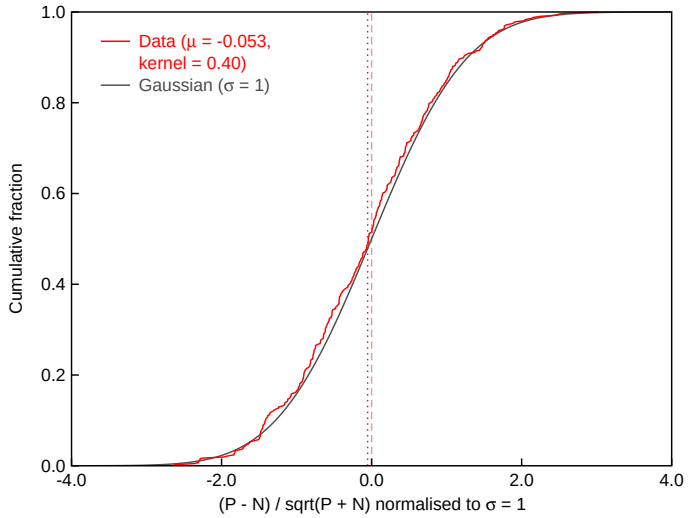


**Figure 3**: Example of a Skellam plot generated by the reliability module of SoFiA 2. The cumulative distribution of the Skellam parameter (Eq. 5) is shown as the red curve and closely follows that of a Gaussian of standard deviation $\sigma = 1$ (grey curve), indicating that the size of the kernel is optimal. The parameters listed in the legend are the location of the median of the Skellam distribution (here: $\mu = -0.053$) and the kernel scale factor set by the user (here: 0.4).

is found to have dropped below the tolerance. If the algorithm fails to converge within a certain number of iterations (defined by `reliability.iterations`), then the value of `reliability.scaleKernel` will be used instead and a warning message will be printed. Users are strongly advised to check the output from the reliability module, including the Skellam plot, to ensure that the final kernel size is indeed acceptable.

## Dealing with artefacts

One potential issue with reliability calculation is that negative artefacts have the potential to reduce the reliability of astronomical sources if they occupy the same parameters space. Such issues could arise from continuum subtraction residuals or from actual HI absorption seen against bright continuum sources. SoFiA 2 offers a solution to this problem by allowing the user to specify a catalogue of sky positions to be excluded from the reliability analysis (using the `reliability.catalog` option). Negative detections with a spatial bounding box encompassing any of the positions in the catalogue will then be excluded from the reliability analysis. This can help with reducing the impact of continuum residuals or HI absorption (which is typically collocated with bright continuum sources) on the reliability analysis and reduce the number of genuine detections that are erroneously discarded as unreliable by SoFiA 2.

The catalogue file must contain exactly two columns listing the longitude and latitude of the sky positions to be excluded in the native coordinate system and units of the input data cube (e.g. right ascension and declination in decimal degrees). Columns can be separated by spaces, tabulators or commas, and entire lines can be commented out using the `#` character. Note that while the affected negative detections will be excluded from the reliability analysis itself, they will nevertheless be retained in the list of detections and will therefore still show up in the reliability plot, although having no effect on the surrounding positive detections in this case.

## Mask dilation

The problem with any source finding algorithm based on setting a threshold is that it will exclude any emission from the source that sits below that threshold, potentially resulting in source masks that are too small and therefore missing some of the flux from the outer regions of the source. This becomes particularly problematic if a fairly high source finding threshold needs to be set, e.g. due to artefacts in the data.

One way of alleviating the problem of missing flux is to grow the source mask outwards until all source flux is included. This is the purpose of SoFiA 2's mask dilation algorithm. It works by iteratively growing the mask outwards until the relative increase in integrated flux per iteration drops below a relative threshold set by the user or until a maximum number of iterations is reached, whichever occurs first. Mask dilation will be carried out separately along the spectral axis and within the spatial plane, with spectral dilation occurring before spatial dilation.

The maximum number of spatial and spectral iterations can be set by the `dilation.iterationsXY` and `dilation.iterationsZ` parameters, which default to 10 and 5, respectively. The relative threshold is set with the `dilation.threshold` parameter, with dilation stopping as soon as the flux increase in an iteration is smaller than the relative threshold times the total flux of the source. The default threshold value is $0.001$, i.e. $0.1\%$ of the total flux. Note that a positive value of `dilation.threshold` will work correctly for sources with either positive or negative total flux, while any negative value of `dilation.threshold` will disable the flux check altogether and instead always carry out the maximum number of iterations.

SoFiA 2's mask dilation algorithm uses a circular dilation kernel in the spatial plane, the radius of which will increase by 1 pixel in each iteration. This ensures that dilation occurs as evenly as possible in all directions while roughly preserving the overall shape of the source. Dilation along the spectral axis simply extends the mask by 1 channel in each iteration.

All relevant source parameters will be updated during mask dilation. In addition, the quality flag will get updated with a flag value of 8 if the mask of a source touches that of a neighbouring source during the dilation process. Note that pixels belonging to a different source will in general be excluded from the mask dilation. Hence, checking the flag value after mask dilation is an important way of identifying possible cases of adjacent sources that may be connected and could be part of one and the same object.

Lastly, mask dilation should be used with caution, as it has the potential to introduce a systematic positive flux bias. In particular, mask dilation should not generally be required if the S+C finder is used with a fairly low detection threshold in the range of $3$–$4\sigma$.

# Source parameterisation

Once linking, reliability filtering (optional) and mask dilation (also optional) have been completed, SoFiA 2 will be able to measure a range of basic source parameters in a process referred to as source parameterisation (or source characterisation). Source parameters are measured across the source mask, i.e. including all pixels considered part of the source according to the mask. An overview of all source parameters measured by SoFiA 2 is presented in Table 1. The methodology used for measuring some of the less intuitive parameters is explained below.

An important aspect of parameterisation is that by default SoFiA 2 will measure all parameters using the native pixel and flux density values of the data cube. Hence, positions will be provided in pixels, line widths in channels, integrated fluxes as simply the sum of all flux densities across the source, etc. However, conversion to physically meaningful parameters can be enabled by setting the `parameter.wcs` and `parameter.physical` options to `true`. The former will convert pixel and channel coordinates to the appropriate world coordinate system (WCS) using WCSLIB and the relevant FITS header information, while the latter will multiply relevant parameters by the spectral channel width and divide spatially integrated parameters by the beam solid angle to correct for the correlation of spatial pixels due to the beam size.

**⚠ WARNING**

If `parameter.physical = true` then SoFiA 2 will try to automatically convert some parameters to physical units under the implicit assumptions that the **beam size** specified in the header (FITS header keywords `BMAJ` and `BMIN`) is correct and does not change with position or frequency and that the **spectral channels** are uncorrelated and the spectral resolution is equal to the channel width. If any of these assumptions is incorrect then the resulting physical parameters, such as integrated flux, may be wrong. **SoFiA 2 does in principal not correct line widths for instrumental broadening**.

## Source position

Measurement of the three-dimensional location, $(\bar{x}, \bar{y}, \bar{z})$, of a source in the data cube is one of the most fundamental parameterisation steps. The most meaningful approach – and the one used by SoFiA 2 – is to determine the *flux-weighted centroid* of the source, which is equivalent to the first moments in $x$, $y$ and $z$, hence

$$\bar{x} = \frac{\sum_i x_i S_i}{\sum_i S_i} \tag{6}$$

and likewise for $\bar{y}$ and $\bar{z}$, where the summation is over all pixels of the source, and $S_i$ is the flux density measured in pixel $i$.[7] Note that in order to prevent negative signals from affecting the centroid measurement, SoFiA 2 will only use pixels with positive (negative) flux density in its measurement of the source position and uncertainty of positive (negative) sources. Source positions will be provided in raw pixel coordinates relative to either the full data cube or, if specified, the requested subregion. Additional celestial and velocity/frequency coordinates can be created by enabling the `parameter.wcs` option.

The *statistical uncertainty* of the flux-weighted centroid measurement can be derived by applying the error propagation law under the fundamental assumptions that the only source of statistical error is from Gaussian noise in the image and that we operate in the linear regime required for the error propagation

[7] The geometric centroid could in principle be calculated from the same equation by setting a constant weight of $S_i = 1$ instead of using the actual flux density value.

law to be applicable. The variance of the centroid is then given as

$$\sigma_{\bar{x}}^2 = \sum_i \left( \frac{\partial \bar{x}}{\partial S_i} \right)^2 \sigma_{S_i}^2 \tag{7}$$

and likewise for $\sigma_{\bar{y}}$ and $\sigma_{\bar{z}}$. By applying the quotient rule to solve the partial derivative and assuming a constant noise level, $\sigma_{S_i} = \sigma_{\mathrm{rms}}$, across the entire image, we obtain

$$\sigma_{\bar{x}}^2 = \left( \frac{\sigma_{\mathrm{rms}}}{S_{\mathrm{tot}}} \right)^2 \sum_i (x_i - \bar{x})^2 \tag{8}$$

and likewise for $\sigma_{\bar{y}}$ and $\sigma_{\bar{z}}$, where $S_{\mathrm{tot}} = \sum_i S_i$ is the summed flux density across the source. It should be noted that Eq. 8 would only be strictly valid if no flux threshold had been applied to the data when calculating the centroid, as any threshold would result in an additional *aliasing bias* that Eq. 8 does not account for in its current form.

Another issue with Eq. 8 is that is does not account for the fact that the flux density values in adjacent pixels might be correlated due to the finite beam size. If the `parameter.physical` setting is enabled, SoFiA 2 will additionally multiply the uncertainty by the square root of the beam solid angle, $\Omega_{\mathrm{PSF}}$, to correct for the effect of correlated pixels, thus

$$\sigma_{\bar{x}}^2 = \Omega_{\mathrm{PSF}} \left( \frac{\sigma_{\mathrm{rms}}}{S_{\mathrm{tot}}} \right)^2 \sum_i (x_i - \bar{x})^2 \tag{9}$$

and likewise for $\sigma_{\bar{y}}$ and $\sigma_{\bar{z}}$. The beam solid angle is derived from the `BMAJ` and `BMIN` keywords in the header of the input data cube and measured in units of pixels. It is further assumed to be constant across the entire spatial and spectral range covered by the data cube and described by an elliptical Gaussian (see Eq. 13). It should be noted that Eq. 9 is only an approximation to the true statistical uncertainty that should be accurate to within about 30%. A full covariance analysis or mock data test would be required to obtain more accurate position uncertainties.

Lastly, SoFiA 2 will also provide the position of the pixel containing the highest flux in the integrated flux map as well as the channel containing the highest flux density in the integrated spectrum of the source. These are identified by the additional suffix '`_peak`' and can be useful in certain situations, for example for diagnostic purposes. Note that the peak position parameters are integer values corresponding to their respective pixels, and no statistical uncertainties will be provided for that reason.

## Flux density

The minimum and maximum flux density of a source is simply defined as

$$S_{\mathrm{min}} = \min\{S_i\}\,, \quad S_{\mathrm{max}} = \max\{S_i\}\,. \tag{10}$$

The statistical uncertainty of both the minimum and maximum flux density is simply given by the RMS noise level of the data, $\sigma_{\mathrm{rms}}$, assuming that the noise is constant across the extent of the source.

It should be noted that – in the presence of noise – selecting the maximum flux density value will result in a systematic *positive bias*, the significance of which will depend on the shape and extent of the source. This is due to the fact that the maximum flux density is more likely to be coincident with a positive noise peak than with a negative one. A morphology-dependent, statistical bias correction would therefore need to be applied to correct for this bias.

## Flux

SoFiA 2 derives the total flux by summing the flux density values of all spatial and spectral pixels covered by the source, thus

$$S_{\mathrm{sum}} = \sum_i S_i\,. \tag{11}$$

The resulting value will be in native flux units of the data cube, typically Jy/beam, and has not yet been corrected to account for the beam solid angle or the spectral channel width of the data. If the `parameter.physical` option is enabled and the native flux density unit (`BUNIT`) of the data cube is `Jy/beam`, then SoFiA 2 will multiply the total flux by the width, $\Delta z$, of a spectral channel and divide by the solid angle, $\Omega_{\mathrm{PSF}}$, of the beam,

$$S_{\mathrm{sum}} = \frac{\Delta z}{\Omega_{\mathrm{PSF}}} \sum_i S_i \,, \tag{12}$$

as derived from the `CDELT3` header keyword and the beam information stored in the data cube header (`BMAJ` and `BMIN` keywords). The native spectral unit of the cube as specified by the `CUNIT3` keyword will be used.[8] The beam is assumed to be described by a two-dimensional, elliptical Gaussian function, and the beam solid angle is calculated as

$$\Omega_{\mathrm{PSF}} = \frac{\pi \vartheta_a \vartheta_b}{4 \ln(2)} \,, \tag{13}$$

where $\vartheta_a$ and $\vartheta_b$ are the full width at half maximum of the major and minor axis of the Gaussian beam in units of pixels (not seconds of arc). Furthermore, the assumption is made that the beam is constant across the spatial and spectral range covered by the data cube.

The statistical uncertainty of the integrated flux can in principle be derived by applying the error propagation law to Eq. 11, hence

$$\sigma^2_{S_{\mathrm{sum}}} = \sum_i \left( \frac{\partial S_{\mathrm{sum}}}{\partial S_i} \right)^2 \sigma^2_{S_i} = \sum_i \sigma^2_{S_i} = N_{\mathrm{pix}} \, \sigma^2_{\mathrm{rms}} \,, \tag{14}$$

where $N_{\mathrm{pix}}$ is the total number of pixels across which the signal is integrated. In the last step we again make the assumption that the noise is constant across the extent of the source. As before, this has not yet been corrected for the spectral channel width or the correlation of spatial pixels due to the beam. If we multiply by the spectral channel width and in addition account for the degree of spatial correlation of the noise in adjacent pixels, we instead obtain

$$\sigma_{S_{\mathrm{sum}}} = \sqrt{\frac{N_{\mathrm{pix}}}{\Omega_{\mathrm{PSF}}}} \, \Delta z \, \sigma_{\mathrm{rms}} \tag{15}$$

for the uncertainty of the true integrated flux of a source. It should be noted that the purely statistical flux uncertainties derived by SoFiA 2 are not usually representative of the true errors of the flux measurement. In most cases, the integrated flux error is dominated by contributions from systematic errors that are significantly larger than the small statistical errors caused by stochastic noise. A realistic error analysis would therefore have to be based on numerical methods such as the injection of model sources of known flux into the data cube.

## Line widths

SoFiA 2 will measure the width of the integrated spectral profile of sources at levels of 20% and 50% of the peak flux density of the profile ($w_{20}$ and $w_{50}$, respectively). The algorithm will move from both edges of the spectrum inwards up to the point where the flux density is found to have increased to more than 20% or 50% of the peak. The separation between those two points on either side of the spectral profile then defines the width of the line. In order to improve the accuracy of the measurement, SoFiA 2 will carry out a linear interpolation across the two channels in between which the flux density exceeds the respective threshold level.

It should be noted that this method of measuring $w_{20}$ and $w_{50}$ is strongly affected by the noise level in the spectrum, and line widths may get overestimated due to the impact of individual noise peaks. This will

[8] If the `CUNIT3` keyword is missing, the default units defined by the FITS standard will be used instead.

in part be alleviated by the fact that the peak flux density will generally be overestimated as well in the presence of noise. A more accurate method of measuring line widths would be to fit an analytic function to the spectral profile, e.g. a Gaussian function for simple profiles, or a Busy Function (Westmeier et al. 2014) in the case of double-horn profiles.

## Ellipse fitting

The purpose of ellipse fitting is to obtain a measure of the angular size and orientation of a source. One of the fastest and easiest ways of fitting an ellipse to the two-dimensional image of a source is through spatial moment analysis (Banks et al. 1995). Let us define the second-order moments of the image as

$$M_{xx} = \frac{\sum_i (x_i - \bar{x})^2 S_i}{\sum_i S_i} \,, \tag{16}$$

$$M_{yy} = \frac{\sum_i (y_i - \bar{y})^2 S_i}{\sum_i S_i} \,, \tag{17}$$

$$M_{xy} = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y}) S_i}{\sum_i S_i} \,, \tag{18}$$

where the summation is over all spatial pixels, $i$, of the source, and $(\bar{x}, \bar{y})$ is the spatial centroid of the source as defined in Eq. 6. Following Banks et al. (1995), the parameters of the ellipse are then given as

$$a = \sqrt{2\left(M_{xx} + M_{yy} + \sqrt{(M_{xx} - M_{yy})^2 + 4M_{xy}^2}\right)} \,, \tag{19}$$

$$b = \sqrt{2\left(M_{xx} + M_{yy} - \sqrt{(M_{xx} - M_{yy})^2 + 4M_{xy}^2}\right)} \,, \tag{20}$$

$$\theta = \frac{1}{2} \arctan\left(\frac{2M_{xy}}{M_{xx} - M_{yy}}\right) \,, \tag{21}$$

where $a$ and $b$ are the full major and minor axis size, respectively, and $\theta$ is the position angle of the ellipse. If the source is a two-dimensional, elliptical Gaussian function, then $a/2$ and $b/2$ will be identical to the standard deviations, $\sigma_a$ and $\sigma_b$, along the major and minor axis of the Gaussian, because the second moment describes the dispersion of the data.

SoFiA 2 performs two different types of spatial ellipse fits to the moment-0 map of each source. It first fits an ellipse to all pixels of the moment map with positive flux density, weighting each pixel by its flux density. The second fit is carried out on all pixels of the moment map with a signal-to-noise ratio of $> 3$ and with equal weighting of each pixel. The former will typically place more weight on the bright, central regions of a source, while the latter will normally provide a better description of the overall shape and size of an object at the $3\sigma$ level, although this critically depends on the surface brightness of the object relative to the noise level of the data.

It should be noted that the ellipse sizes and orientation angles derived in this way will be relative to the pixel grid, not relative to the world coordinate system associated with the data cube. The major and minor axis of the ellipse will be provided in units of pixels, while the orientation angle will be given in degrees, running from $-90°$ (right) across $0°$ (top) to $+90°$ (left) in the mathematically positive sense.

## Kinematic major axis

SoFiA 2 measures the position angle of the kinematic major axis of a source by first calculating the centroid of the emission in each individual channel, taking only values above three times the local noise level into account. If the source is a rotating galactic disc, then the resulting points should form a straight line on the sky that marks the kinematic major axis of the galaxy. In order to extract the position angle, SoFiA 2 will fit a straight line using Deming regression (also known as orthogonal regression) and convert the resulting slope into a proper position angle in degrees such that $0°$ points upwards and the resulting

direction corresponds to the side of the galaxy that occupies the *upper end* of the channel range covered by the source.

Note that it is the responsibility of the user to rotate the position angle by $180°$ if necessary to comply with the desired definition with respect to the approaching or receding side of the galaxy. This will also depend on whether the data cube is provided in units of frequency or velocity. It should also be noted that the position angle will be relative to the pixel grid of the data cube, not the underlying celestial coordinate system.

**Table 1**: List of source parameters and associated Unified Content Descriptors (UCD; Derriere et al. 2005) as supplied by SoFiA 2. Depending on the actual data cube and user settings, not all of the parameters will be present in a particular source catalogue.

| Parameter | UCD | Description |
|---|---|---|
| `name` | meta.id | Name of source. |
| `id` | meta.id | Unique integer number identifying source. |
| `x`<br>`y`<br>`z` | pos.cartesian.x<br>pos.cartesian.y<br>pos.cartesian.z | Flux-weighted centroid position of source in pixels. |
| `x_peak`<br>`y_peak` | pos.cartesian.x<br>pos.cartesian.y | Pixel with highest flux in integrated flux map. |
| `z_peak` | pos.cartesian.z | Channel with highest flux density in integrated spectrum. |
| `x_min`<br>`x_max`<br>`y_min`<br>`y_max`<br>`z_min`<br>`z_max` | pos.cartesian.x;stat.min<br>pos.cartesian.x;stat.max<br>pos.cartesian.y;stat.min<br>pos.cartesian.y;stat.max<br>pos.cartesian.z;stat.min<br>pos.cartesian.z;stat.max | Bounding box of source in $x$, $y$ and $z$ in pixels, inclusive of the limit itself. |
| `n_pix` | meta.number;instr.pixel | Total number of spatial and spectral pixels covered by source. |
| `f_min`<br>`f_max` | phot.flux.density;stat.min<br>phot.flux.density;stat.max | Minimum and maximum flux density in native units of data cube. |
| `f_sum` | phot.flux | Sum of flux densities across source in native units of data cube. |
| `rel` | stat.probability | Reliability in range of 0 (unreliable) to 1 (reliable). |
| `flag` | meta.code.qual | Flag indicating if source is located near spatial (1) or spectral (2) edge of cube, near blanked pixels (4) or adjacent to other source (8). Flag values are additive. |
| `fill` | stat.filling | Filling factor of mask within its bounding box. |
| `mean`<br>`std` | phot.flux.density;stat.mean<br>phot.flux.density;stat.stdev | Mean and standard deviation of flux densities across mask. |
| `skew`<br>`kurt` | stat.param<br>stat.param | Skewness and kurtosis of flux densities across mask. |
| `rms` | instr.det.noise | RMS of local noise within bounding box of source in native units of data cube. |
| `w20`<br>`w50` | spect.line.width<br>spect.line.width | Width of spectral line at 20% and 50% of peak flux density. |
| `wm50` | spect.line.width | Line width at 50% of mean flux density. |

| | | |
|---|---|---|
| ell_maj<br>ell_min | phys.angSize<br>phys.angSize | Major and minor axis size of ellipse fitted to integrated flux density map in units of pixels. Pixels are weighted by flux density. |
| ell3s_maj<br>ell3s_min | phys.angSize<br>phys.angSize | Same as ell_maj and ell_min, but including only pixels above 3 times the local RMS and without weighting. |
| ell_pa | pos.posAng | Position angle for ell_maj in range of $-90°$ to $+90°$, where $0°$ points up. |
| ell3s_pa | pos.posAng | Same as ell_pa, but for ell3s_maj. |
| kin_pa | pos.posAng | Position angle of kinematic major axis in range of $0°$ to $360°$, where $0°$ points up. Refers to side of object occupying upper end of spectral channel range covered by source. |
| ra<br>dec<br>l<br>b | pos.eq.ra<br>pos.eq.dec<br>pos.galactic.lon<br>pos.galactic.lat | Right ascension and declination, or longitude and latitude, of source centroid position, (x, y), in native world coordinates of data cube; usually in degrees. The same parameters with the additional suffix '_peak' correspond to (x_peak, y_peak) instead. |
| freq<br>v_rad<br>v_opt<br>v_app | em.freq<br>spect.dopplerVeloc.radio<br>spect.dopplerVeloc.opt<br>spect.dopplerVeloc | Frequency or velocity of source centroid position, z, in native world coordinates of data cube; usually in Hz or m/s. The same parameters with additional suffix '_peak' correspond to z_peak instead. |
| err_x<br>err_y<br>err_z | stat.error;pos.cartesian.x<br>stat.error;pos.cartesian.y<br>stat.error;pos.cartesian.z | Statistical uncertainty of centroid position, (x, y, z). |
| err_f_sum | stat.error;phot.flux | Statistical uncertainty of integrated flux, f_sum. |

# Output products

SoFiA 2 offers a wide range of different output products to be generated from the source finding and parameterisation results that are not restricted to just source catalogues. All output products will be written to the same directory as the input data file unless a different output directory has been specified by the user (`output.directory` option). Likewise, the name of the input data file will be used as the base name for all output products unless a different base name has been specified (`output.filename` option). Output file names consist of the base name followed by an underscore (_) and a product identifier (e.g. `cat` for catalogues). As an example, if the input file name was `ngc_300_mosaic.fits`, then the output catalogue would be named `ngc_300_mosaic_cat.txt` and so forth.

The `output.overwrite` option allows the user to decide whether existing files should automatically be overwritten or not. If set to `false`, SoFiA 2 will explicitly check for the presence of any output files at the beginning of the pipeline and terminate with an error message if a file already exists.

## Source catalogue

The most important output file provided by SoFiA 2 is the source catalogue resulting from the pipeline run. Source catalogues contain a list of all detected sources and their basic parameters such as position, line width, flux, etc. Catalogues are offered in four different possible formats:

- **ASCII** (`_cat.txt`) — Plain text file containing the source catalogue in human-readable ASCII format; not intended for quantitative analysis.
- **XML** (`_cat.xml`) — XML file containing the source catalogue in VO Table format for processing with VO-compliant tools such as TOPCAT.
- **SQL** (`_cat.sql`) — File containing commands for creating an SQL database table and inserting the source parameters into that table.
- **Karma** (`_cat.ann`) — Not a catalogue as such, but a Karma annotation file that can be used to display source IDs on images viewed in Karma packages such as kvis.

The ASCII catalogue is meant to enable users to quickly inspect the output catalogue by eye, but may not be ideal for quantitative analyses, as the precision of some of the columns may be limited. The XML and SQL catalogues are much more suitable for a numerical analysis of source parameters, in particular in combination with VO-compliant analysis and visualisation tools, as all parameters are stored at the full available precision.

The XML catalogue also has Unified Content Descriptors (UCDs) defined to allow VO-compliant software tools to automatically determine the relevant columns of the catalogue, e.g. for extracting sky positions. The SQL catalogue can be imported into any SQL database. It will attempt to create a new data table named `SoFiA-Catalogue` and insert the measured parameters of all sources into that table. The catalogue file would have to be edited to change the default table name.

## Image products

In addition to catalogues, SoFiA 2 can produce a range of image products to assist with the assessment and interpretation of the source finding results. These include:

- **Noise cube** (`_noise.fits`) — 3D cube containing the RMS noise level as measured by the local noise scaling algorithm.
- **Noise spectrum** (`_noise.txt`) — 1D spectrum containing the RMS noise level as measured by the spectral noise scaling algorithm.
- **Filtered cube** (`_filtered.fits`) — Copy of the 3D data cube after preconditioning such as flagging or noise scaling.

- **Mask cube** (`_mask.fits`) — Final 3D source detection mask established by SoFiA 2. All detected pixels in the mask cube are set to their respective source ID as listed in the catalogue, while undetected pixels have a value of 0.
- **Mask image** (`_mask-2d.fits`) — Spectrally projected 2D image of the 3D mask cube to show which spatial pixels contain source emission. Note that in the 2D mask image sources may be hidden behind other sources, thus making their source IDs invisible.
- **Raw mask cube** (`_mask-raw.fits`) — Raw, binary source detection mask created by the source finding algorithm prior to linking and filtering. The raw mask will contain all pixels originally picked up by the source finder (including those with negative flux) and is therefore mainly intended for debugging purposes. Detected pixels will have a value of 1, all other pixels will be 0.
- **Moment maps** (`_mom0.fits`, `_mom1.fits`, `_mom2.fits`) — 2D images of the $0^{\text{th}}$, $1^{\text{st}}$ and $2^{\text{nd}}$ spectral moments of the data cube across the source mask. All data will be used for the $0^{\text{th}}$ moment, while the $1^{\text{st}}$ and $2^{\text{nd}}$ moment will only include channels with positive flux density.
- **Channel map** (`_chan.fits`) — 2D image showing the number of channels that contributed to the $0^{\text{th}}$ spectral moment in each spatial pixel. This can be useful when estimating statistical uncertainties associated with the moment-0 map.

Note that creation of each of these data products will need to be explicitly enabled in the parameter file by setting the respective options in the `output` module.

All output images created by SoFiA 2 will by default be in native pixel and flux density units. By enabling the `parameter.wcs` option, the user can choose to create all moment maps in proper physical units by converting spectral channels to world coordinates in frequency or velocity units. It is crucial for this purpose that the WCS information in the header is accurate, as otherwise the relevant map values might be incorrect.

## Cubelets

In addition to the global image products introduced in Section 9.2, SoFiA 2 is also capable of producing cutouts of each detected source (so-called *cubelets*) and related, source-specific data products. These can be enabled by setting `output.writeCubelets = true`, and `output.writePV = true` if position–velocity diagrams are also desired, and are stored in a directory the name of which is the base name followed by `_cubelets`. The name of each of the source-specific files follows the following scheme:

```
basename_id_prod.type
```

where `basename` is the file base name, `id` is the source ID from the catalogue, `prod` is the product identifier, and `type` is the file type suffix. As an example, the cubelet of source number 3 from the catalogue obtained from the input data cube `ngc_300_mosaic.fits` would be named `ngc_300_mosaic_3_cube.fits` and stored in the subdirectory `ngc_300_mosaic_cubelets`. The following list gives an overview of the individual source-specific data products provided by SoFiA 2:

- **Cubelet** (`_cube.fits`) — Small 3D data cube containing a cutout of the source from the original data cube.
- **Mask** (`_mask.fits`) — Small 3D cube containing a cutout of the source mask. Pixels belonging to the source will have a value of 1, while all other pixels are set to 0, including those belonging to other sources.
- **Moment maps** (`_mom0.fits`, `_mom1.fits`, `_mom2.fits`) — 2D images of the $0^{\text{th}}$, $1^{\text{st}}$ and $2^{\text{nd}}$ spectral moments of the source cubelet across the source mask. Note that all data will be used in the $0^{\text{th}}$ moment, while the $1^{\text{st}}$ and $2^{\text{nd}}$ moment will include only those channels with a flux density greater than `output.thresholdMom12` times the local RMS noise level.

- **Channel map** (`_chan.fits`) — 2D image showing the number of channels that contributed to each pixel of the $0^{\text{th}}$ spectral moment map.
- **SNR map** (`_snr.fits`) — 2D image showing the signal-to-noise ratio in each pixel of the $0^{\text{th}}$ spectral moment map, calculated as $\text{SNR} = \sum S_i / (\sqrt{N} \times \sigma_{\text{rms}})$, where $\sum S_i$ is the sum of all flux densities along the spectral axis, $N$ is the number of spectral channels summed over, and $\sigma_{\text{rms}}$ is the local RMS noise level measured at the position of the source.
- **PV diagrams** (`_pv.fits`, `_pv_min.fits`) — FITS files containing 2D position–velocity diagrams along the kinematic major (`kin_pa`) and minor (`kin_pa` + 90°) axes through the flux-weighted centroid (`x`, `y`, `z`) of each source. These are for diagnostic purposes only, as they may not be accurate enough for scientific analysis. Note that PV diagrams need to be specifically enabled with `output.writePV = true`.
- **PV masks** (`_pv_mask.fits`, `_pv_min_mask.fits`) — FITS files containing 2D source masks in position–velocity space to be used with the position–velocity diagrams. Note that PV diagrams need to be specifically enabled with `output.writePV = true`.
- **Spectrum** (`_spec.txt`) — Text file containing the integrated spectrum of the source across the source mask. The number of spatial pixels contributing to each spectral channel is also recorded, allowing uncertainties to be derived.

As in the case of global output products, the `parameter.wcs` and `parameter.physical` options can be enabled to convert moment maps and spectra to world coordinates and divide all spectra by the beam solid angle, respectively. In addition, the amount of padding around the source in each of these data products can be controlled by the user by setting the value of `output.marginCubelets`. The default value is 10 pixels, while a value of 0 can be used to ensure that images and spectra are tightly cut without extra padding around the source, thus minimising file sizes and disk storage requirements.

## Diagnostic output

SoFiA 2 can produce a few additional output files for diagnostic purposes that may be helpful in assessing the quality and success of the source finding run. These include:

- **Reliability plot** (`_rel.eps`) — Plot summarising the outcome of the reliability filter in EPS format. This is useful for assessing the quality of the reliability calculation carried out by SoFiA 2 (see Section 6).
- **Skellam plot** (`_skellam.eps`) — Plot showing the cumulative distribution of the Skellam parameter as defined in Eq. 5. This can be used to check if the size of the kernel used in the reliability calculation is optimal (see Section 6.4).
- **Auto-flagging log** (`_flags.log`) — Log file listing the spectral channels and spatial pixels flagged by the auto-flagger (see Section 3.1).

Note that these diagnostic files will only be generated if explicitly requested by the user.

# Tips and tricks

## Example parameter file

A basic example parameter file for deep source finding on an extragalactic HI data cube is presented here. It is assumed that the cube is clean with a constant noise level across the entire cube and without any noticeable artefacts.

```
scfind.kernelsXY         =  0, 5, 10
scfind.kernelsZ          =  0, 3, 7, 15, 31
scfind.threshold         =  3.8
scfind.replacement       =  2.0

linker.radiusXY          =  2
linker.radiusZ           =  3
linker.minSizeXY         =  5
linker.minSizeZ          =  5

reliability.enable       =  true
reliability.threshold    =  0.8
reliability.scaleKernel  =  0.4
reliability.minSNR       =  3.0
reliability.plot         =  true

output.writeCatASCII     =  true
output.writeCatXML       =  true
output.writeMoments      =  true
output.writeCubelets     =  true
```

The first block of commands sets up the S+C finder (module `scfind`). We choose spatial smoothing kernel sizes of 0, 5 and 10 pixels and spectral smoothing kernel sizes of 0, 3, 7, 15 and 31 channels. These numbers may need to be slightly adjusted depending on the actual number of pixels across the beam and the actual spectral resolution of the cube, as we ideally want to smooth over a range of scales up to the largest expected spatial and spectral scale of our sources.

The second block of commands sets up the linker (module `linker`) that allows us to merge detected pixels into coherent sources. We choose to link pixels across a spatial radius of 2 and a spectral radius of 3 here. Lastly, we require a source to extend across at least 5 spatial pixels and 5 spectral channels to be retained in the catalogue. Any smaller detections will be discarded.

With the third block of commands we set up the reliability filter (module `reliability`) which will calculate the statistical reliability of each detection and discard everything below a given threshold. We set the threshold to 0.8 here (i.e. keeping only sources with a reliability in excess of 80%). In addition, we set a minimum signal-to-noise threshold of 3, thereby discarding all sources that are too faint. In addition, we set the scale factor of the kernel used in measuring the reliability to a value of 0.4 and enable the generation of diagnostic plots.

In the last block of commands we define the output products and settings (module `output`). This includes writing the source catalogue in plain-text and VO-compatible XML format, writing out moment maps and creating images and spectra for each individual detection.

Assuming that these settings are stored in a file named `sofia.par`, we can then launch SoFiA 2 by calling

```
sofia sofia.par input.data=datacube.fits
```

where `datacube.fits` needs to be replaced with the actual name of the input data cube. This will run SoFiA 2 and produce the selected output catalogues and images.

## 2D images

As mentioned before, SoFiA 2 is capable of handling 2D images such as radio continuum maps. Such images will internally be treated as 3D cubes with an axis size of 1 in the frequency dimension. For SoFiA 2 to work correctly on 2D images, a few special settings are required. Most importantly, the settings of the S + C finder will need to be adjusted to disable smoothing along the frequency axis by setting `scfind.kernelsZ = 0`. Hence, only spatial smoothing kernels can be applied using the `scfind.kernelsXY` option.

Likewise, the linker settings must be adjusted to account for the 2D nature of the input. In particular, the maximum merging radius along the frequency axis must be set to `linker.radiusZ = 1` to ensure the correct application of the spatial merging radius in the remaining two dimensions. In addition, the minimum source size along the frequency axis must be set to `linker.minSizeZ = 1`, as the spectral extent of 2D sources cannot be greater than one channel.

The source catalogue produced from 2D images will also contain parameters that are only relevant to 3D cubes, such as the frequency of the source or its spectral line width. Such parameters should simply be ignored and in many cases may have been set to a default value of zero. SoFiA 2 will in principle not produce certain output data products if the input image is 2D. Omitted products include the integrated spectrum as well as the $1^{st}$ and $2^{nd}$ spectral moment maps, although the $0^{th}$ moment will be generated, effectively containing a masked copy of the input image.

To summarise, the most important point to remember when processing 2D images is to set frequency-related control parameters to the following default values:

```
scfind.kernelsZ = 0
linker.minSizeZ = 1
linker.radiusZ  = 1
```

Similar restrictions apply with respect to the frequency-related parameters of optional methods such as the local noise scaling algorithm.

## Absorption lines

While intended for detecting HI emission lines, SoFiA 2 can in principle be used to search for spectral absorption signals in data cubes. For this to work, the data cube will need to be inverted by enabling the `input.invert` option in SoFiA 2. It should be noted that the extraction of absorption features might fail in cases where both HI absorption and emission is present in a source, as SoFiA 2 will by default merge significant positive and negative signal into the same detection.

## Extracting a sub-cube

While originally a source finding pipeline, SoFiA 2 can also be employed to simply extract a smaller subregion out of a large FITS data cube. This can be achieved by explicitly disabling all modules and output products except for the following settings:

```
input.data           = <data_cube>
input.region         = <requested_region>
output.writeFiltered = true
```

where `<data_cube>` is the input FITS data cube, and `<requested_region>` is the subregion to be extracted. Assuming all other settings are disabled by setting them to `false` where required, SoFiA 2 will then simply read the requested subregion from the input cube and write it straight into an output file with the additional suffix `_filtered`. The pipeline will terminate thereafter, as nothing else would be left to do.

Note that extracting a sub-cube will only require as much memory as is needed for storing the subregion. Sufficiently small subregions can therefore be extracted from FITS cubes that are far larger than the amount of memory available on a machine.

## Source catalogue in Python

The Astropy package contains a `Table` module for handling VOTable files, making it straightforward to load the SoFiA 2 source catalogue in XML format into Python. This can be done with the following lines of code (remember to replace `catalogue.xml` with your actual catalogue file name):

```
from astropy.table import Table
table = Table.read("catalogue.xml")
```

This will create an Astropy table object that contains the full catalogue. The individual columns of the table can then be accessed by their parameter name. For example,

```
table["f_sum"]
```

will extract the entire column of integrated flux measurements into a single 1D array. The fluxes of individual sources can then be extracted by index as `table["f_sum"][0]` and so forth.

Note that Astropy's `Table` module is also capable of reading the plain-text catalogue from SoFiA 2 using `Table.read("catalogue.txt", format="ascii")`. However, due to the limited precision and difficulty extracting the parameter names and units in this case we strongly advise users to work with the XML catalogue only. Please see the Astropy documentation for more information on reading and handling tables.

As is typical for Python, there are countless other options for reading VOTable or plain-text catalogues from SoFiA 2. Examples include the `astropy.io.votable` module which provides a method called `parse_single_table()` for reading VOTables, the `ascii()` method from `astropy.io` which can read the plain-text catalogue from SoFiA 2, and the `loadtxt()` method from NumPy which can be used to read the numerical data columns from the plain-text catalogue into a floating-point array. Each of these methods comes with its own advantages and drawbacks, and we refer the reader to the respective online documentation for further details.

# Control parameters

This appendix provides a systematic description of the control parameters accepted by SoFiA 2, grouped by topic.

## General

| Parameter | Type | Values | Default | Description |
|---|---|---|---|---|
| pipeline.pedantic | bool | true, false | true | If set to true, the pipeline will terminate with an error message if an unknown parameter name is encountered in the input parameter file. If set to false, unknown parameters will instead be ignored. |
| pipeline.threads | int | $\geq 0$ | 0 | Sets the maximum number of parallel threads that multi-threaded algorithms within SoFiA 2 are allowed to use. If set to 0 (default value), then the OMP_NUM_THREADS environment variable is used to control the number of threads. If the value equals (or exceeds) the number of available threads, then all CPU cores will be utilised, which minimises the runtime of the pipeline at the cost of maximal CPU load. |
| pipeline.verbose | bool | true, false | false | Determines the level of output messages produced by the pipeline. Additional warning messages can be enabled by setting the value to true. |

## Input

| Parameter | Type | Values | Default | Description |
|---|---|---|---|---|
| input.data | string | | | Name of the input data cube on which to run the source finder. The absolute path to the data file must be provided. If only the file name is specified, the pipeline will assume the file to be located in the current working directory. Currently, only the FITS format is supported. |
| input.gain | string | | | Name of an optional data cube containing the gain across the image. If specified, the input data cube will be divided by the gain cube prior to source parameterisation to ensure that the correct flux values are extracted. The gain cube must have the same dimensions as the input data cube. The absolute path to the gain file must be provided. If only the file name is specified, the pipeline will assume the file to be located in the current working directory. Currently, only the FITS format is supported. |

| | | | | |
|---|---|---|---|---|
| input.invert | bool | true, false | false | If set to true, invert the data cube prior to processing. This is useful when searching for negative rather than positive signals such as absorption lines. Note that all flux-related parameters and maps will be inverted, too, in this case and hence be positive rather than negative. |
| input.mask | string | | | File name of an input mask cube. Any additional pixels detected by the source finder will be added to the input mask. This can be useful if the results from two different source finding runs should be combined into a single mask. The mask cube must have the same dimensions as the input data cube. The absolute path to the mask file must be provided. If only the file name is specified, the pipeline will assume the file to be located in the current working directory. Currently, only the FITS format is supported. |
| input.noise | string | | | Name of an optional data cube containing the noise levels across the image. If specified, the input data cube will be divided by the noise cube prior to source finding to ensure that a constant source finding threshold can be applied. The noise cube must have the same dimensions as the input data cube. The absolute path to the noise file must be provided. If only the file name is specified, the pipeline will assume the file to be located in the current working directory. Currently, only the FITS format is supported. Note that either a noise cube or a weights cube can be applied, but not both. |
| input.region | list | | | Region of the input data cube to be searched. Only the specified region will be loaded into memory and processed. A region must contain six comma-separated integer values of the following format: x_min, x_max, y_min, y_max, z_min, z_max (all in units of pixels and 0-based). If no region is specified, then the entire data cube will be loaded. |
| input.weights | string | | | Name of an optional data cube containing the weights across the image. If specified, the input data cube will be multiplied by the square root of the weights cube prior to source finding to ensure that a constant source finding threshold can be applied. The weights cube must have the same dimensions as the input data cube. The absolute path to the weights file must be provided. If only the file name is specified, the pipeline will assume the file to be located in the current working directory. Currently, only the FITS format is supported. Note that either a noise cube or a weights cube can be applied, but not both. |

## Preconditioning

| Parameter | Type | Values | Default | Description |
|---|---|---|---|---|
| contsub.enable | bool | true, false | false | If enabled, SoFiA 2 will try to subtract any residual continuum emission from the data cube prior to source finding by fitting and subtracting a polynomial of order 0 (offset) or 1 (offset + slope). The order of the polynomial is defined by contsub.order. |
| contsub.order | int | 0 . . . 1 | 0 | Order of the polynomial to be used in continuum subtraction if contsub.enable is set to true. Can either be 0 for a simple offset or 1 for an offset + slope. Higher orders are not currently supported. |
| contsub.padding | int | $\geq 0$ | 3 | The amount of additional padding (in channels) applied to either side of channels excluded from the fit. |
| contsub.shift | int | $\geq 1$ | 4 | The number of channels by which the spectrum will be shifted (symmetrically in both directions) before self-subtraction. |
| contsub.threshold | float | $\geq 0.0$ | 2.0 | Relative clipping threshold. All channels with a flux density > contsub.threshold times the noise will be clipped and excluded from the polynomial fit. |
| flag.auto | string | true, false, channels, pixels | false | If set to true, SoFiA 2 will attempt to automatically flag spectral channels and spatial pixels affected by interference or artefacts based on their RMS noise level. If set to channels, only spectral channels will be flagged. If set to pixels, only spatial pixels will be flagged. If set to false, auto-flagging will be disabled. Please see the user manual for details. |
| flag.catalog | string | | | Path to a catalogue file containing two columns that specify the longitude and latitude coordinates of sky positions to be flagged in the native coordinate system and units of the input data cube. The two columns can be separated by spaces, tabulators or commas. Also see flag.radius. |
| flag.log | bool | true, false | false | If set to true, write a list of the channels and pixels flagged by the auto-flagger to a log file. Note that if no channels or pixels were found to be in need of flagging, then the log file will not be written irrespective of the value of flag.log. |
| flag.radius | int | $\geq 0$ | 5 | Radius around the sky positions listed in the catalogue provided by flag.catalog that should be flagged. If 0, then only the nearest pixel to the position will be flagged. Otherwise, pixels within the specified radius around the nearest pixel will be flagged. |

| | | | | |
|---|---|---|---|---|
| flag.region | list | | | Region(s) to be flagged in the input data cube prior to processing. The flagging region must contain a multiple of six comma-separated integer values of the following format: x_min, x_max, y_min, y_max, z_min, z_max, ... (all in units of pixels and 0-based). Pixels within those regions will be set to blank in the input cube. If unset, no flagging will occur. |
| flag.threshold | float | | 5.0 | Relative threshold in multiples of the standard deviation to be applied by the automatic flagging algorithm. Only relevant if flag.auto is enabled. Please see the documentation for details. |
| rippleFilter.enable | bool | true, false | false | If set to true, then the ripple filter will be applied to the data cube prior to source finding. The filter works by measuring and subtracting either the mean or median across a running window. This can be useful if a DC offset or spatial/spectral ripple is present in the data. |
| rippleFilter.gridXY | int | $\geq 0$ | 0 | Spatial grid separation in pixels for the running window used in the ripple filter. The value must be an odd integer value and specifies the spatial step by which the window is moved. Alternatively, it can be set to 0, in which case it will default to half the spatial window size (see rippleFilter.windowXY). |
| rippleFilter.gridZ | int | $\geq 0$ | 0 | Spectral grid separation in channels for the running window used in the ripple filter. The value must be an odd integer value and specifies the spectral step by which the window is moved. Alternatively, it can be set to 0, in which case it will default to half the spectral window size (see rippleFilter.windowZ). |
| rippleFilter.interpolate | bool | true, false | false | If set to true, then the mean or median values measured across the running window in the ripple filter will be linearly interpolated in between the grid points. If set to false, the mean or median will be subtracted from the entire grid cell without interpolation. |
| rippleFilter.statistic | string | mean, median | median | Controls whether the mean or median should be measured and subtracted in the running window of the ripple filter. The median is strongly recommended, as it is more robust. |
| rippleFilter.windowXY | int | $\geq 1$ | 31 | Spatial size in pixels of the running window used in the ripple filter. The size must be an odd integer number. |
| rippleFilter.windowZ | int | $\geq 1$ | 15 | Spectral size in channels of the running window used in the ripple filter. The size must be an odd integer number. |

| | | | | |
|---|---|---|---|---|
| scaleNoise.enable | bool | true, false | false | If set to true, noise scaling will be enabled. The purpose of the noise scaling modules is to measure the noise level in the input cube and then divide the input cube by the noise. This can be used to correct for spatial or spectral noise variations across the input cube prior to running the source finder. |
| scaleNoise.fluxRange | string | positive, negative, full | negative | Flux range to be used in the noise measurement. If set to negative or positive, only pixels with negative or positive flux will be used, respectively. This can be useful to prevent real emission or artefacts from affecting the noise measurement. If set to full, all pixels will be used in the noise measurement irrespective of their flux. |
| scaleNoise.gridXY | int | $\geq 0$ | 0 | Size of the spatial grid across which noise measurement window will be moved across the data cube. It must be an odd integer value. If set to 0 instead, the spatial grid size will default to half the spatial window size. |
| scaleNoise.gridZ | int | $\geq 0$ | 0 | Size of the spectral grid across which noise measurement window will be moved across the data cube. It must be an odd integer value. If set to 0 instead, the spectral grid size will default to half the spectral window size. |
| scaleNoise.interpolate | bool | true, false | false | If set to true, linear interpolation will be used to interpolate the measured local noise values in between grid points. If set to false, the entire grid cell will instead be filled with the measured noise value. |
| scaleNoise.mode | string | spectral, local | spectral | Noise scaling mode. If set to spectral, the noise level will be determined for each spectral channel by measuring the noise within each image plane. This is useful for data cubes where the noise varies with frequency. If set to local, the noise level will be measured locally in window running across the entire cube in all three dimensions. This is useful for data cubes with more complex noise variations, such as interferometric images with primary-beam correction applied. |
| scaleNoise.scfind | bool | true, false | false | If true and global or local noise scaling is enabled, then noise scaling will additionally be applied after each smoothing operation in the S+C finder. This might be useful in certain situations where large-scale artefacts are present in interferometric data. However, this feature should be used with great caution, as it has the potential to do more harm than good. |

| | | | | |
|---|---|---|---|---|
| scaleNoise.statistic | string | std, mad, gauss | mad | Statistic to be used in the noise measurement process. Possible values are std, mad and gauss for standard deviation, median absolute deviation and Gaussian fitting to the flux histogram, respectively. Standard deviation is by far the fastest algorithm, but it is also the least robust one with respect to emission and artefacts in the data. Median absolute deviation and Gaussian fitting are far more robust in the presence of strong, extended emission or artefacts, but will usually take longer. |
| scaleNoise.windowXY | int | $\geq 0$ | 25 | Spatial size of the window used in determining the local noise level. It must be an odd integer value. If set to 0, the pipeline will use the default value instead. |
| scaleNoise.windowZ | int | $\geq 0$ | 15 | Spectral size of the window used in determining the local noise level. It must be an odd integer value. If set to 0, the pipeline will use the default value instead. |

## Source Finding

| Parameter | Type | Values | Default | Description |
|---|---|---|---|---|
| scfind.enable | bool | true, false | true | If set to true, the Smooth + Clip (S+C) finder will be enabled. The S+C finder operates by iteratively smoothing the data cube with a user-defined set of smoothing kernels, measuring the noise level on each smoothing scale, and adding all pixels with an absolute flux above a user-defined relative threshold to the source detection mask. |
| scfind.fluxRange | string | positive, negative, full | negative | Flux range to be used in the noise measurement. If set to negative or positive, only pixels with negative or positive flux will be used, respectively. This can be useful to prevent real emission or artefacts from affecting the noise measurement. If set to full, all pixels will be used in the noise measurement irrespective of their flux. |
| scfind.kernelsXY | list | $\geq 0$ | 0, 3, 6 | Comma-separated list of spatial Gaussian kernel sizes to apply. The individual kernel sizes must be floating-point values and denote the full width at half maximum (FWHM) of the Gaussian used to smooth the data in the spatial domain. A value of 0 means that no spatial smoothing will be applied. |
| scfind.kernelsZ | list | $\geq 0$ | 0, 3, 7, 15 | Comma-separated list of spectral Boxcar kernel sizes to apply. The individual kernel sizes must be odd integer values of 3 or greater and denote the full width of the Boxcar filter used to smooth the data in the spectral domain. A value of 0 means that no spectral smoothing will be applied. |

| | | | | |
|---|---|---|---|---|
| scfind.replacement | float | | 2.0 | Before smoothing the data cube during an S+C iteration, every pixel in the data cube that was already detected in a previous iteration will be replaced by this value multiplied by the original noise level in the non-smoothed data cube, while keeping the original sign of the data value. This feature can be disabled altogether by specifying a value of < 0. |
| scfind.statistic | string | std, mad, gauss | mad | Statistic to be used in the noise measurement process. Possible values are std, mad and gauss for standard deviation, median absolute deviation and Gaussian fitting to the flux histogram, respectively. Standard deviation is by far the fastest algorithm, but it is also the least robust one with respect to emission and artefacts in the data. Median absolute deviation and Gaussian fitting are far more robust in the presence of strong, extended emission or artefacts, but will usually take longer. |
| scfind.threshold | float | $\geq 0.0$ | 5.0 | Flux threshold to be used by the S+C finder relative to the measured noise level in each smoothing iteration. In practice, values in the range of about 3 to 5 have proven to be useful in most situations, with lower values in that range requiring use of the reliability filter to reduce the number of false detections. |
| threshold.enable | bool | true, false | false | If set to true, the threshold finder will be enabled. The threshold finder is a very basic source finder that simply applies a fixed threshold (either absolute or relative to the noise) to the original data cube. It can be useful if a simple flux threshold is to be applied to a pre-processed or filtered data cube. |
| threshold.fluxRange | string | positive, negative, full | negative | Flux range to be used in the noise measurement. If set to negative or positive, only pixels with negative or positive flux will be used, respectively. This can be useful to prevent real emission or artefacts from affecting the noise measurement. If set to full, all pixels will be used in the noise measurement irrespective of their flux. |
| threshold.mode | string | absolute, relative | relative | If set to absolute, the flux threshold of the threshold finder will be interpreted as an absolute flux threshold in the native flux unit of the data cube. If set to relative, the threshold will be interpreted in units of the noise level across the data cube. |

| | | | | |
|---|---|---|---|---|
| threshold.statistic | string | std, mad, gauss | mad | Statistic to be used in the noise measurement process if threshold.mode is set to relative. Possible values are std, mad and gauss for standard deviation, median absolute deviation and Gaussian fitting to the flux histogram, respectively. Standard deviation is by far the fastest algorithm, but it is also the least robust one with respect to emission and artefacts in the data. Median absolute deviation and Gaussian fitting are far more robust in the presence of strong, extended emission or artefacts, but will usually take longer. |
| threshold.threshold | float | $\geq 0.0$ | 5.0 | Flux threshold to be applied by the threshold finder. Depending on the threshold.mode parameter, this can either be absolute (in native flux units of the data cube) or relative to the noise level of the cube. |

## Linking

| Parameter | Type | Values | Default | Description |
|---|---|---|---|---|
| linker.enable | bool | true, false | true | If true, then the linker will be run to merge the pixels detected by the source finder into coherent detections that can then be parameterised and catalogued. If false, the pipeline will be terminated after source finding, and no catalogue or source products will be created. Disabling the linker can be useful if only the raw mask from the source finder is needed. |
| linker.keepNegative | bool | true, false | false | If set to true, then the linker will not discard detections with negative flux. Reliability filtering must be disabled for negative sources to be retained. Also note that negative sources will not appear in moment 1 and 2 maps. This option should only ever be used for testing or debugging purposes, but never in production mode. |
| linker.maxFill | float | $\geq 0.0$ | 0.0 | Maximum allowed filling factor of a source within its rectangular bounding box, defined as the number of spatial and spectral pixels that make up the source divided by the number of pixels in the bounding box. The default value of 0.0 disables maximum filling factor filtering. |
| linker.maxPixels | int | $\geq 0$ | 0 | Maximum allowed number of spatial and spectral pixels that a source must not exceed. The default value of 0 disables maximum size filtering. |
| linker.maxSizeXY | int | $\geq 0$ | 0 | Maximum size of sources in the spatial dimension in pixels. Sources that exceed this limit will be discarded by the linker. If the value is set to 0, maximum size filtering will be disabled. |

| | | | | |
|---|---|---|---|---|
| linker.maxSizeZ | int | $\geq 0$ | 0 | Maximum size of sources in the spectral dimension in pixels. Sources that exceed this limit will be discarded by the linker. If the value is set to 0, maximum size filtering will be disabled. |
| linker.minFill | float | $\geq 0.0$ | 0.0 | Minimum allowed filling factor of a source within its rectangular bounding box, defined as the number of spatial and spectral pixels that make up the source divided by the number of pixels in the bounding box. The default value of 0.0 disables minimum filling factor filtering. |
| linker.minPixels | int | $\geq 0$ | 0 | Minimum allowed number of spatial and spectral pixels that a source must have. The default value of 0 disables minimum size filtering. |
| linker.minSizeXY | int | $\geq 1$ | 5 | Minimum size of sources in the spatial dimension in pixels. Sources that fall below this limit will be discarded by the linker. |
| linker.minSizeZ | int | $\geq 1$ | 5 | Minimum size of sources in the spectral dimension in pixels. Sources that fall below this limit will be discarded by the linker. |
| linker.positivity | bool | true, false | false | If set to true, then the linker will only merge positive pixels and discard all negative pixels by removing them from the mask. This option should be used with extreme caution and will render the reliability filter useless. It can be useful, though, if there are significant negative artefacts such as residual sidelobes in the data. |
| linker.radiusXY | int | $\geq 1$ | 1 | Maximum merging length in the spatial dimension. Pixels with a separation of up to this value will be merged into the same source. |
| linker.radiusZ | int | $\geq 1$ | 1 | Maximum merging length in the spectral dimension. Pixels with a separation of up to this value will be merged into the same source. |

## Reliability

| Parameter | Type | Values | Default | Description |
|---|---|---|---|---|
| reliability.autoKernel | bool | true, false | false | If true, SoFiA 2 will try to automatically determine the optimal reliability kernel scale factor by iteratively increasing the kernel size until the absolute value of the median of the Skellam distribution decreases below reliability.tolerance. If the algorithm is not able to converge after reliability.iterations steps, then the default value of reliability.scaleKernel will be used instead. |

| | | | | |
|---|---|---|---|---|
| `reliability.catalog` | string | | | Path to a file containing positions on the sky to be excluded from the reliability analysis. The file must contain two columns separated by a space, tabulator or comma that specify the longitude and latitude of the position to be excluded in the native WCS coordinates and units of the input FITS file. Negative detections that contain any of those positions within their bounding box will be excluded from the reliability analysis, although they will still show up in the reliability plot. |
| `reliability.debug` | bool | true, false | false | If set to `true` and the reliability module is enabled, then two catalogue files containing relevant reliability parameters of negative and positive detections are created for debugging purposes. The catalogues will be written in VOTable format. |
| `reliability.enable` | bool | true, false | false | If set to `true`, reliability calculation and filtering will be enabled. This will determine the reliability of each detection with positive total flux by comparing the density of positive and negative detections in a three-dimensional parameter space. Sources below the specified reliability threshold will then be discarded. Note that this will require a sufficient number of negative detections, which can usually be achieved by setting the source finding threshold to somewhere around 3 to 4 times the noise level. |
| `reliability.iterations` | int | $\geq 1$ | 30 | Maximum number of iterations allowed for the reliability kernel auto-scaling algorithm to converge. If convergence cannot be achieved, then `reliability.scaleKernel` will instead be applied. |
| `reliability.minPixels` | int | $\geq 0$ | 0 | Minimum total number of spatial and spectral pixels within the source mask for detections to be considered reliable. The reliability of any detection with fewer pixels will be set to zero by default. |
| `reliability.minSNR` | float | $\geq 0.0$ | 3.0 | Lower signal-to-noise limit for reliable sources. Detections that fall below this threshold will be deemed unreliable and assigned a reliability of 0. The value denotes the integrated signal-to-noise ratio, SNR $= F_{\mathrm{sum}}/(\mathrm{RMS}\sqrt{N\Omega})$, of the source, where $\Omega$ is the solid angle (in pixels) of the point spread function of the data, $N$ is the number of spatial and spectral pixels of the source, $F_{\mathrm{sum}}$ is the summed flux density and RMS is the local RMS noise level (assumed to be constant). Note that the spectral resolution is assumed to be equal to the channel width. |

| | | | | |
|---|---|---|---|---|
| reliability.parameters | list | peak, sum, mean, chan, pix, fill, std, skew, kurt | peak, sum, mean | Parameter space to be used in deriving the reliability of detections. This must be a list of parameters the number of which defines the dimensionality of the parameter space. Possible values are peak for the peak flux density, sum for the summed flux density, mean for mean flux density, chan for the number of spectral channels, pix for the total number of spatial and spectral pixels, fill for the filling factor, std for the standard deviation, skew for the skewness and kurt for the kurtosis across the source mask. Flux densities will be divided by the global RMS noise level. peak, sum, mean, pix and fill will be logarithmic, all other parameters linear. |
| reliability.plot | bool | true, false | true | If set to true, diagnostic plots (in EPS format) will be created to allow the quality of the reliability estimation to be assessed. It is advisable to generate and inspect these plots to ensure that the outcome of the reliability filtering procedure is satisfactory. |
| reliability.scaleKernel | float | | 0.4 | When estimating the density of positive and negative detections in parameter space, the size of the Gaussian kernel used in this process is determined from the covariance of the distribution of negative detections in parameter space. This parameter setting can be used to scale that kernel by a constant factor. |
| reliability.threshold | float | 0.0 . . . 1.0 | 0.9 | Reliability threshold in the range of 0 to 1. Sources with a reliability below this threshold will be discarded. |
| reliability.tolerance | float | | 0.05 | Convergence tolerance for the reliability kernel auto-scaling algorithm. Convergence is achieved when the absolute value of the median of the Skellam distribution drops below this tolerance. |

## Mask Dilation

| Parameter | Type | Values | Default | Description |
|---|---|---|---|---|
| dilation.enable | bool | true, false | false | Set to true to enable source mask dilation whereby the mask of each source will be grown outwards until the resulting increase in integrated flux drops below a given threshold or the maximum number of iterations is reached. |
| dilation.iterationsXY | int | $\geq 1$ | 10 | Sets the maximum number of spatial iterations for the mask dilation algorithm. Once this number of iterations has been reached, mask dilation in the spatial plane will stop even if the flux increase still exceeds the threshold set by dilation.threshold. |

| | | | | |
|---|---|---|---|---|
| dilation.iterationsZ | int | $\geq 1$ | 5 | Sets the maximum number of spectral iterations for the mask dilation algorithm. Once this number of iterations has been reached, mask dilation along the spectral axis will stop even if the flux increase still exceeds the threshold set by dilation.threshold. |
| dilation.threshold | float | | 0.001 | If a positive value is provided, mask dilation will end when the increment in the integrated flux during a single iteration drops below this value times the total integrated flux (from the previous iteration), or when the maximum number of iterations has been reached. Specifying a negative threshold will disable flux checking altogether and always carry out the maximum number of iterations. |

## Parameterisation

| Parameter | Type | Values | Default | Description |
|---|---|---|---|---|
| parameter.enable | bool | true, false | true | If set to true, the parametrisation module will be enabled to measure the basic parameters of each detected source. |
| parameter.offset | bool | true, false | false | If set to false and a region of the data cube is read in using the input.region parameter, then the position parameters x, y, z, x_min, x_max, y_min, y_max, z_min and z_max in the source catalogue will be specified relative to the region. If set to true, the position parameters will instead be relative to the full cube. Note that the auto-flagging log file will also adhere to this setting. |
| parameter.physical | bool | true, false | false | If set to true, SoFiA 2 will attempt to convert relevant parameters to physical units. This involves conversion of channel widths to frequency/velocity units and division of flux-based parameters by the solid angle of the beam. For this to work, the relevant header parameters, including CTYPE3, CDELT3, BMAJ and BMIN, must have been correctly set. It is further assumed that the beam does not vary with frequency or position. |
| parameter.prefix | string | | SoFiA | Prefix to be used in source names. The default prefix is SoFiA, and the resulting default source name is SoFiA Jhhmmss.ss-ddmmss.s for J2000 equatorial coordinates (and likewise for other coordinate types). |
| parameter.wcs | bool | true, false | true | If set to true, SoFiA will attempt to convert the source centroid position (x, y, z) to world coordinates using the WCS information stored in the header. In addition, spectra and moment map units will be converted from channels to WCS units as well. |

## Output

| Parameter | Type | Values | Default | Description |
|---|---|---|---|---|
| output.directory | string | | | Full path to the directory to which all output files will be written. If unset, the directory of the input data cube will be used by default. |
| output.filename | string | | | File name that will be used as the template for all output files. For example, if output.filename = my_data, then the output files will be named my_data_cat.xml, my_data_mom0.fits, etc. If unset, the name of the input data cube will be used as the file name template by default. |
| output.marginCubelets | int | $\geq 0$ | 10 | Margin (in pixels) around detections to be added when creating cubelets, moment maps and spectra of individual sources. The same margin will be applied to all axes of the cube. A value of 0 will create tight cutouts without any extra margin, thus minimising file sizes. The default is 10 pixels. |
| output.overwrite | bool | true, false | true | If true, existing output files will be overwritten without warning. If false, SoFiA 2 will refuse to run if any of the output files and directories to be created already exists. |
| output.thresholdMom12 | float | | 0.0 | If output.cubelets is enabled, then the moment 1 and 2 maps for each individual detection will be created using only those spectral channels where the flux density exceeds this value times the local RMS noise level. For example, setting output.thresholdMom12 to a value of 3.0 would set a 3-sigma flux density threshold for moments 1 and 2. Note that this setting has no effect on moment 0 maps or global moment 1 and 2 maps. |
| output.writeCatASCII | bool | true, false | true | If set to true, an output source catalogue will be produced in human-readable ASCII format. The catalogue file will have the suffix _cat.txt. |
| output.writeCatSQL | bool | true, false | false | If set to true, an output source catalogue will be produced in SQL format. The catalogue file will have the suffix _cat.sql. The SQL catalogue can be imported into any SQL-compatible database. A new data table containing the source parameters, named SoFiA-Catalogue by default, will be generated. |
| output.writeCatXML | bool | true, false | true | If set to true, an output source catalogue will be produced in VO-compatible XML format. The catalogue file will have the suffix _cat.xml. |

| | | | | |
|---|---|---|---|---|
| output.writeCubelets | bool | true, false | false | If set to true, then individual source products for each detected source will be created, including subcubes, masks, moment maps and integrated spectra. The source products will be written to a subdirectory with the suffix _cubelets. Each source product will be labelled with the source ID number for identification. |
| output.writeFiltered | bool | true, false | false | If set to true and any input filtering algorithm is enabled, then a data cube containing the filtered data will be written in FITS format. The filtered cube will have the suffix _filtered.fits. |
| output.writeKarma | bool | true, false | false | If set to true then a Karma annotation file will be created that contains the source IDs of all detections in the catalogue. This can be used to display source IDs on output images in Karma packages such as kvis. The annotation file will have the suffix _cat.ann. |
| output.writeMask | bool | true, false | false | If set to true, then a data cube containing the final source mask produced by the source finder will be written in FITS format. The pixel values in the source mask will correspond to the respective source ID numbers in the catalogue. The mask cube will have the suffix _mask.fits. |
| output.writeMask2d | bool | true, false | false | If set to true, then an image containing a two-dimensional projection of the 3D mask cube will be written in FITS format. The 2D mask image will have the suffix _mask-2d.fits. Note that some sources may be hidden behind others in this 2D projection. |
| output.writeMoments | bool | true, false | false | If set to true, then images of the spectral moments 0, 1 and 2 and the number of channels in each pixel of the moment 0 map will be written in FITS format. The maps will have the suffix _mom0.fits, _mom1.fits, _mom2.fits and _chan.fits. Note that moments 1 and 2 and the number of channels will not be produced if the input data cube is only two-dimensional. |
| output.writeNoise | bool | true, false | false | If set to true and local noise scaling is enabled, then a data cube containing the measured local noise values will be written in FITS format. The noise cube will have the suffix _noise.fits. If spectral noise scaling is enabled, then the measured noise in each channel (in native data cube flux units) will be written to a plain text file with the suffix _noise.txt. |
| output.writePV | bool | true, false | false | If set to true then position-velocity diagrams along the kinematic major and minor axis of the data cube and mask cube of each detection will be created. Note that output.writeCubelets must also be set to true. |

| | | | | |
|---|---|---|---|---|
| output.writeRawMask | bool | true, false | false | If set to true, then a data cube containing the raw, binary source mask produced by the source finder prior to linking will be written in FITS format. The raw mask cube will have the suffix _mask-raw.fits. |

## File and directory structure

The entire SoFiA 2 pipeline is written in C, and the source code is highly modular thanks to an overall object-oriented approach. The source code of the main pipeline is stored in the file `sofia.c` in the base directory, while the rest of the source code can be found in the `src/` directory. Files defining classes start with a capital letter, e.g. `DataCube.c`, while procedural source code is found in files starting with a small letter, e.g. `statistics_flt.c`. All public declarations are stored in header files of the same name, e.g. `DataCube.h`.

The basic file and directory structure of SoFiA 2 is outlined in the following table. The file and directory names are hyperlinks to the corresponding target in the SoFiA 2 GitLab repository.

| File name | Description |
|---|---|
| `sofia.c` | Actual SoFiA 2 pipeline. |
| `template_par_file.par` | Template parameter file. |
| `compile.sh` | Shell script for compiling SoFiA 2. |
| `README.md` | Basic information and installation instructions. |
| `LICENSE` | Licence information. |
| `src/` | Directory containing source code. |
| `common.c` | Functions commonly used by all source files. |
| `statistics_dbl.c` | Basic statistical functions for type `double`. |
| `statistics_flt.c` | Basic statistical functions for type `float`. |
| `Array_dbl.c` | Class for data arrays of type `double`. |
| `Array_siz.c` | Class for data arrays of type `size_t`. |
| `Catalog.c` | Class for storing source catalogues. |
| `DataCube.c` | Class for handling FITS data cubes. |
| `Header.c` | Class for storing FITS header information. |
| `LinkerPar.c` | Class for collecting object properties during linking. |
| `Map.c` | Class for storing key–value pairs of type `size_t`. |
| `Matrix.c` | Class for handling matrices. |
| `Parameter.c` | Class for storing SoFiA 2 parameter settings. |
| `Path.c` | Class for handling file names and paths. |
| `Source.c` | Class for storing the properties of a detected source. |
| `Stack.c` | Class for implementing a basic stack of type `size_t`. |
| `String.c` | Class for storing and handling character strings. |
| `Table.c` | Class for reading tabulated data from file. |
| `WCS.c` | Class for handling WCS conversions. |
| `templates/` | Directory containing function and class templates. |
| `statistics.c` | Template for basic statistical functions. |
| `Array.c` | Template for data array class. |
| `Array_maketemplate.sh` | Shell script for instantiating data array templates. |
| `statistics_maketemplate.sh` | Shell script for instantiating statistics templates. |

## Return codes

Upon termination SoFiA 2 will provide a return code that can be used to analyse the cause of any error that occurred during execution of the pipeline. A list of return codes currently supported by SoFiA 2 can be found in the following table.

| Code | Symbol | Description |
|---|---|---|
| 0 | ERR_SUCCESS | Successful run. |
| 1 | ERR_FAILURE | An unspecified error occurred. |
| 2 | ERR_NULL_PTR | A NULL pointer was encountered. |
| 3 | ERR_MEM_ALLOC | A memory allocation error occurred. This could indicate that the data cube is too large for the amount of memory available on the machine. |
| 4 | ERR_INDEX_RANGE | An array index was found to be out of range. |
| 5 | ERR_FILE_ACCESS | An error occurred while trying to read or write a file or check if a directory or file is accessible. |
| 6 | ERR_INT_OVERFLOW | An integer overflow occurred. |
| 7 | ERR_USER_INPUT | The pipeline was aborted due to invalid user input. This could be due to an invalid parameter setting or the wrong input file being provided. |
| 8 | ERR_NO_SRC_FOUND | No specific error occurred, but no sources were detected either. |

## References

Banks T., Dodd R. J., Sullivan D. J., 1995, MNRAS, 272, 821

Derriere S., et al., 2005, An IVOA Standard for Unified Content Descriptors Version 1.10, IVOA Recommendation 19 August 2005 (arXiv:1110.0525)

Pence W. D., Chiappetti L., Page C. G., Shaw R. A., Stobie E., 2010, A&A, 524, A42

Serra P., Jurek R., Flöer L., 2012, PASA, 29, 296

Serra P., et al., 2015, MNRAS, 448, 1922

Westmeier T., Jurek R., Obreschkow D., Koribalski B. S., Staveley-Smith L., 2014, MNRAS, 438, 1176

Westmeier T., et al., 2021, MNRAS, 506, 3962

# Index

## T



## U



## W