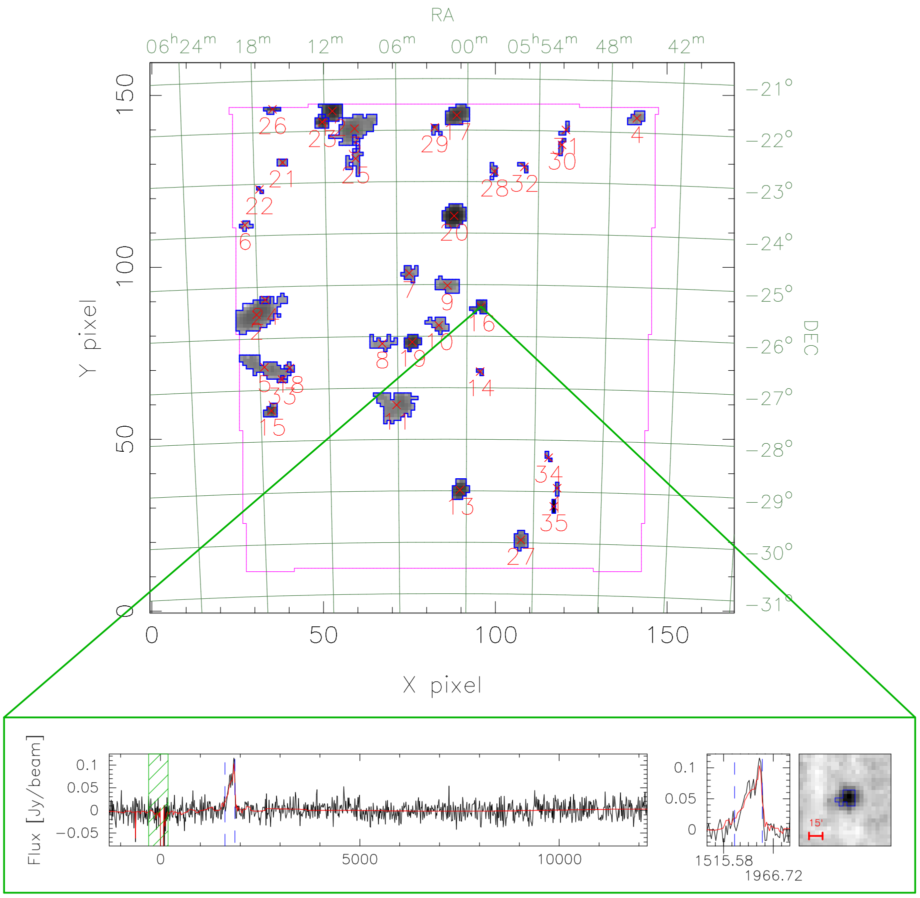
Source Detection with *Duchamp*

A User's Guide

Matthew Whiting
Australia Telescope National Facility
CSIRO

Duchamp version 1.2
August 8, 2012



Contents

1 Introduction and getting going quickly

1.1 About *Duchamp*

This document provides a user's guide to *Duchamp*, an object-finder for use on spectral-line data cubes. The basic execution of *Duchamp* is to read in a FITS data cube, find sources in the cube, and produce a text file of positions, velocities and fluxes of the detections, as well as a postscript file of the spectra of each detection.

Duchamp has been designed to search for objects in particular sorts of data: those with relatively small, isolated objects in a large amount of background or noise. Examples of such data are extragalactic HI surveys, or maser surveys. *Duchamp* searches for groups of connected voxels (or pixels) that are all above some flux threshold. No assumption is made as to the shape of detections, and the only size constraints applied are those specified by the user.

Duchamp has been written as a three-dimensional finder, but it is possible to run it on a two-dimensional image (i.e. one with no frequency or velocity information), or indeed a one-dimensional array, and many of the features of the program will work fine. The focus, however, is on object detection in three dimensions, one of which is a spectral dimension. Note, in particular, that it does not do any fitting of source profiles, a feature common (and desirable) for many two-dimensional source finders. This is beyond the current scope of *Duchamp*, whose aim is reliable detection of spectral-line objects.

Duchamp provides the ability to pre-process the data prior to searching. Spectral baselines can be removed, and either smoothing or multi-resolution wavelet reconstruction can be performed to enhance the completeness and reliability of the resulting catalogue.

1.2 Acknowledging the use of *Duchamp*

Duchamp is provided in the hope that it will be useful for your research. If you find that it is, I would ask that you acknowledge it in your publication by using the following: "This research made use of the Duchamp source finder, produced at the Australia Telescope National Facility, CSIRO, by M. Whiting."

Additionally, *Duchamp* has been described in a journal paper (Whiting 2012). This paper covers the key algorithms implemented in the software, and provides some simple completeness and reliability comparisons of different modes of operation. Users of *Duchamp* are encouraged to read the paper in conjunction with this user guide, as while some things are repeated herein, not everything is. Whiting (2012) should be cited when describing the use of *Duchamp* in your research.

1.3 What to do

So, you have a FITS cube, and you want to find the sources in it. What do you do? First, you need to get *Duchamp*: there are instructions in Appendix A for obtaining and installing it. Once you have it running, the first step is to make an input file that contains the list of parameters. Brief and detailed examples are shown in Appendix C. This file provides the input file name, the various output files, and defines various parameters that control the execution.

The standard way to run *Duchamp* is by the command

```
> Duchamp -p [parameter file]
```

replacing `[parameter file]` with the name of the file listing the parameters.

An even easier way is to use the default values for all parameters (these are given in Appendix B and in the file `InputComplete` included in the distribution directory) and use the syntax

```
> Duchamp -f [FITS file]
```

where `[FITS file]` is the file you wish to search.

The default action includes displaying a map of detected objects in a PGPLOT X-window. This can be disabled by setting the parameter `flagXOutput = false` or using the `-x` command-line option, as in

```
> Duchamp -x -p [parameter file]
```

and similarly for the `-f` case.

Once a FITS file and parameters have been set, the program will then work away and give you the list of detections and their spectra. The program execution is summarised below, and detailed in §3. Information on inputs is in §2 and Appendix B, and descriptions of the output is in §4.

1.4 Guide to terminology and conventions

First, a brief note on the use of terminology in this guide. *Duchamp* is designed to work on FITS "cubes". These are FITS[1] image arrays with (at least) three dimensions. They are assumed to have the following form: the first two dimensions (referred to as x and y) are spatial directions (that is, relating to the position on the sky – often, but not necessarily, corresponding to Equatorial or Galactic coordinates), while the third dimension, z, is the spectral direction, which can correspond to frequency, wavelength, or velocity. The three dimensional analogue of pixels are "voxels", or volume cells – a voxel is defined by a unique (x, y, z) location and has a single value of flux, intensity or brightness (or something equivalent) associated with it.

Sometimes, some pixels in a FITS file are labelled as BLANK – that is, they are given a nominal value, defined by FITS header keywords BLANK (and potentially BSCALE and BZERO), that marks them as not having a flux value. These are often used to pad a cube out so that it has a rectangular spatial shape. *Duchamp* has the ability to avoid these: see §3.3.1.

Note that it is possible for the FITS file to have more than three dimensions (for instance, there could be a fourth dimension representing a Stokes parameter). Only the two spatial dimensions and the spectral dimension are read into the array of pixel values that is searched for objects. All other dimensions are ignored[2]. Herein, we discuss the data in terms of the three basic dimensions, but you should be aware it is possible for the FITS file to have more than three. Note that the order of the dimensions in the FITS file does not matter.

[1] FITS is the Flexible Image Transport System – see Hanisch et al. (2001) or websites such as http://fits.cv.nrao.edu/FITS.html for details.

[2] This actually means that the first pixel only of that axis is used, and the array is read by the `fits_read_subsetnull` command from the CFITSIO library.

With this setup, each spatial pixel (a given (x, y) coordinate) can be said to be a single spectrum, while a slice through the cube perpendicular to the spectral direction at a given z-value is a single channel, with the 2-D image in that channel called a channel map.

Detection involves locating contiguous groups of voxels with fluxes above a certain threshold. *Duchamp* makes no assumptions as to the size or shape of the detected features, other than allowing user-selected minimum size criteria. Features that are detected are assumed to be positive. The user can choose to search for negative features by setting an input parameter – which will invert the cube prior to the search (see §3.7 for details).

1.5 A summary of the execution steps

The basic flow of the program is summarised here – all steps are discussed in more detail in the following sections.

1. The necessary parameters are recorded.

   How this is done depends on the way the program is run from the command line. If the `-p` option is used, the parameter file given on the command line is read in, and the parameters therein are read. All other parameters are given their default values (listed in Appendix B).

   If the `-f` option is used, all parameters are assigned their default values.

2. The FITS image is located and read in to memory.

   The file given is assumed to be a valid FITS file. As discussed above, it can have any number of dimensions, but *Duchamp* only reads in the two spatial and the one spectral dimensions. A subset of the FITS array can be given (see §3.1 for details).

3. If requested, a FITS file containing a previously reconstructed or smoothed array is read in.

   When a cube is either smoothed or reconstructed with the *à trous* wavelet method, the result can be saved to a FITS file, so that subsequent runs of *Duchamp* can read it in to save having to re-do the calculations.

4. If requested, BLANK pixels are trimmed from the edges, and the baseline of each spectrum is removed.

   BLANK pixels, while they are ignored by all calculations in *Duchamp*, do increase the size in memory of the array above that absolutely needed. This step trims them from the spatial edges, keeping a record of the amount trimmed so that they can be added back in later.

   A spectral baseline (or bandpass) may optionally be removed at this point as well. This may be necessary if there is a ripple or other large-scale feature present that will hinder detection of faint sources.

5. If the reconstruction method is requested, and the reconstructed array has not been read in at Step 3 above, the cube is reconstructed using the *à trous* wavelet method.

   This step uses the multi-resolution *à trous* method to determine the amount of structure present at various scales. A simple thresholding technique then removes random noise from the cube, leaving the significant signal. This process can greatly reduce the noise level in the cube, enhancing the reliability of the resulting catalogue.

6. Alternatively (and if requested), the cube is smoothed, either spectrally or spatially.

   This step presents two options. The first considers each spectrum individually, and convolves it with a Hanning filter (with width chosen by the user). The second considers each channel map separately, and smoothes it with a Gaussian kernel of size and shape chosen by the user. This step can help to reduce the amount of noise visible in the cube and enhance fainter sources, increasing the completeness and reliability of the output catalogue.

7. A threshold for the cube is then calculated, based on the pixel statistics (unless a threshold is manually specified by the user).

   The threshold can either be chosen as a simple $n\sigma$ threshold (i.e. a certain number of standard deviations above the mean), or calculated via the "False Discovery Rate" method. Alternatively, the threshold can be specified as a simple flux value, without care as to the statistical significance (e.g. "I want every source brighter than 10mJy").

   By default, the full cube is used for the statistics calculation, although the user can nominate a subsection of the cube to be used instead.

8. Searching for objects then takes place, using the requested thresholding method.

   The cube is searched either one channel-map at a time ("spatial" search) or one spectrum at a time ("spectral" search). Detections are compared to already detected objects and either combined with a neighbouring one or added to the end of the list.

9. The list of objects is condensed by merging neighbouring objects and removing those deemed unacceptable.

   While some merging has been done in the previous step, this process is a much more rigorous comparison of each object with every other one. If a pair of objects lie within requested limits, they are combined.

   After the merging is done, the list is culled (although see comment for the next step). There are certain criteria the user can specify that objects must meet: minimum numbers of spatial pixels and spectral channels, and minimum separations between neighbouring objects. Those that do not meet these criteria are deleted from the list.

10. If requested, the objects are "grown" down to a lower threshold, and then the merging step is done a second time.

    In this case, each object has pixels in its neighbourhood examined, and if they are above a secondary threshold, they are added to the object. The merging process is done a second time in case two objects have grown over the top of one another. Note that the rejection part of the previous step is not done until the end of the second merging process.

11. The baselines and trimmed pixels are replaced prior to output.

    This is just the inverse of step #4.

12. The details of the detections are written to screen and to the requested output file.

    Crucial properties of each detection are provided, showing its location, extent, and flux. These are presented in both pixel coordinates and world coordinates (e.g. sky position and velocity). Any warning flags are also printed, showing detections to be wary of. Alternative output options are available, such as a VOTable or a Karma annotation file.

13. Maps showing the spatial location of the detections are written.

    These are 2-dimensional maps, showing where each detection lies on the spatial coverage of the cube. This is provided as an aid to the user so that a quick idea of the distribution of object positions can be gained e.g. are all the detections on the edge?

    Two maps are provided: one is a 0th moment map, showing the 0th moment (i.e. a map of the integrated flux) of each detection in its appropriate position, while the second is a "detection map", showing the number of times each spatial pixel was detected in the searching routines (including those pixels rejected at step 9 and so not in any of the final detections).

    These maps are written to postscript files, and the 0th moment map can also be displayed in a PGPLOT X-window.

14. A pixel mask is written to a FITS file.

    A FITS file of the same size as the input file can be written. Here, each pixel has a value indicating whether or note it was detected and falls in one of the catalogue sources. Different objects can be traced by different non-zero pixel values.

15. The integrated or peak spectra of each detection are written to a postscript file.

    The spectral equivalent of the maps – what is the spectral profile of each detection? Also provided here are basic information for each object (a summary of the information in the results file), as well as a 0th moment map of the detection.

16. If requested, a text file containing all spectra is written.

    This file will contain the peak or integrated spectra for each source, as a function of the appropriate spectral coordinate. The file is a multi-column ascii text file, suitable for import into other software packages.

17. If requested, the reconstructed or smoothed array can be written to a new FITS file.

    If either of these procedures were done, the resulting array can be saved as a FITS file for later use (for instance, reading in as described at step #3). The FITS header will be the same as the input file, with a few additional keywords to identify the file.

1.6 Why "*Duchamp*"?

Well, it's important for a program to have a name, and the initial working title of *cubefind* was somewhat uninspiring. I wanted to avoid the classic astronomical approach of designing a cute acronym, and since it is designed to work on cubes, I looked at naming it after a cubist. *Picasso*, sadly, was already taken (Minchin 1999), so I settled on naming it after Marcel Duchamp, another cubist, but also one of the first artists to work with "found objects".

2 User Inputs

Input to the program is provided by means of a parameter file. Parameters are listed in the file, followed by the value that should be assigned to them. The syntax used is

```
parameterName value.
```

Parameter names are not case-sensitive, and lines in the input file that start with # are ignored. If a parameter is listed more than once, the latter value is used, but otherwise the order in which the parameters are listed in the input file is arbitrary. Example input files can be seen in Appendix C.

If a parameter is not listed, the default value is assumed. The defaults are chosen to provide a good result (a simple 5σ search with no pre-processing), so the user doesn't need to specify many new parameters in the input file. Note that the image file **must** be specified! The parameters that can be set are listed in Appendix B, with their default values in parentheses.

The parameters with names starting with `flag` are stored as `bool` variables, and so are either `true = 1` or `false = 0`. They can be entered in the file either in text or integer format – *Duchamp* will read them correctly in either case.

An example input file is included in the distribution tar file. It is as follows:

```
imageFile       your-file-here
logFile         logfile.txt
outFile         results.txt
spectraFile     spectra.ps
minPix          2
flagATrous      1
snrRecon        5.
snrCut          3.
minChannels     3
flagBaseline    1
```

You would, of course, replace the "`your-file-here`" with the FITS file you wanted to search. Further examples are given in Appendix C.

3 What *Duchamp* is doing

Each of the steps that *Duchamp* goes through in the course of its execution are discussed here in more detail. This should provide enough background information to fully understand what *Duchamp* is doing and what all the output information is. For those interested in the programming side of things, *Duchamp* is written in C/C++ and makes use of the CFITSIO, WCSLIB and PGPLOT libraries.

3.1 Image input

The cube is read in using basic CFITSIO commands, and stored as an array in a special C++ class. This class keeps track of the list of detected objects, as well as any reconstructed arrays that are made (see §3.4). The World Coordinate System (WCS)[3] information for the cube is also obtained from the FITS header by WCSLIB functions (Calabretta and Greisen 2002; Greisen and Calabretta 2002; Greisen et al. 2006), and this information, in the form of a `wcsprm` structure, is also stored in the same class. See Sec. 3.2 for more details.

A sub-section of a cube can be requested by defining the subsection with the `subsection` parameter and setting `flagSubsection = true` – this can be a good idea if the cube has very noisy edges, which may produce many spurious detections.

There are two ways of specifying the `subsection` string. The first is the generalised form `[x1:x2:dx,y1:y2:dy,z1:z2:dz,...]`, as used by the CFITSIO library. This has one set of colon-separated numbers for each axis in the FITS file. In this manner, the x-coordinates run from `x1` to `x2` (inclusive), with steps of `dx`. The step value can be omitted, so a subsection of the form `[2:50,2:50,10:1000]` is still valid. In fact, *Duchamp* does not make use of any step value present in the subsection string, and any that are present are removed before the file is opened.

If the entire range of a coordinate is required, one can replace the range with a single asterisk, e.g. `[2:50,2:50,*]`. Thus, the subsection string `[*,*,*]` is simply the entire cube. Note that the pixel ranges for each axis start at 1, so the full pixel range of a 100-pixel axis would be expressed as 1:100. A complete description of this section syntax can be found at the FITSIO web site[4].

Making full use of the subsection requires knowledge of the size of each of the dimensions. If one wants to, for instance, trim a certain number of pixels off the edges of the cube, without examining the cube to obtain the actual size, one can use the second form of the subsection string. This just gives a number for each axis, e.g. `[5,5,5]` (which would trim 5 pixels from the start *and* end of each axis).

All types of subsections can be combined e.g. `[5,2:98,*]`.

Typically, the units of pixel brightness are given by the FITS file's BUNIT keyword. However, this may often be unwieldy (for instance, the units are Jy/beam, but the values are around a few mJy/beam). It is therefore possible to nominate new units, to which the pixel values will be converted, by using the `newFluxUnits` input parameter. The units must be directly translatable from the existing ones – for instance, if BUNIT is Jy/beam, you cannot specify mJy,

[3] This is the information necessary for translating the pixel locations to quantities such as position on the sky, frequency, velocity, and so on.

[4] http://heasarc.gsfc.nasa.gov/docs/software/fitsio/c/c_user/node91.html

it must be mJy/beam. If an incompatible unit is given, the BUNIT value is used instead.

3.2 World Coordinate System

Duchamp uses the WCSLIB package to handle the conversions between pixel and world coordinates. This package uses the transformations described in the WCS papers (Calabretta and Greisen 2002; Greisen and Calabretta 2002; Greisen et al. 2006). The same package handles the WCS axes in the spatial plots. The conversions used are governed by the information in the FITS header – this is parsed by WCSLIB to create the appropriate transformations.

For the spectral axis, however, *Duchamp* provides the ability to change the type of transformation used, so that different spectral quantities can be calculated. By using the parameter `spectralType`, the user can change from the type given in the FITS header. This should be done in line with the conventions outlined in Greisen et al. (2006). The spectral type can be either a full 8-character string (e.g. 'VELO-F2V'), or simply the 4-character "S-type" (e.g. 'VELO'), in which case WCSLIB will handle the conversion.

The rest frequency can be provided as well. This may be necessary, if the FITS header does not specify one and you wish to transform to velocity. Alternatively, you may want to make your measurements based on a different spectral line (e.g. OH1665 instead of HI-21cm). The input parameter `restFrequency` is used, and this will override the FITS header value.

Finally, the user may also request different spectral units from those in the FITS file, or from the defaults arising from the WCSLIB transformation. The input parameter `spectralUnits` should be used, and Greisen and Calabretta (2002) should be consulted to ensure the syntax is appropriate.

3.3 Image modification

Several modifications to the cube can be made that improve the execution and efficiency of *Duchamp* (their use is optional, governed by the relevant flags in the parameter file).

3.3.1 BLANK pixel removal

If the imaged area of a cube is non-rectangular (see the example in Fig. 3, a cube from the HIPASS survey), BLANK pixels are used to pad it out to a rectangular shape. The value of these pixels is given by the FITS header keywords BLANK, BSCALE and BZERO. While these pixels make the image a nice shape, they will take up unnecessary space in memory, and so to potentially speed up the processing we can trim them from the edge. This is done when the parameter `flagTrim = true`. If the above keywords are not present, the trimming will not be done (in this case, a similar effect can be accomplished, if one knows where the "blank" pixels are, by using the subsection option).

The amount of trimming is recorded, and these pixels are added back in once the source-detection is completed (so that quoted pixel positions are applicable to the original cube). Rows and columns are trimmed one at a time until the first non-BLANK pixel is reached, so that the image remains rectangular. In practice, this means that there will be some BLANK pixels left in the trimmed image (if the non-BLANK region is non-rectangular). However, these are ignored in all further calculations done on the cube.

3.3.2 Baseline removal

Second, the user may request the removal of baselines from the spectra, via the parameter `flagBaseline`. This may be necessary if there is a strong baseline ripple present, which can result in spurious detections at the high points of the ripple. The baseline is calculated from a wavelet reconstruction procedure (see §3.4) that keeps only the two largest scales. This is done separately for each spatial pixel (i.e. for each spectrum in the cube), and the baselines are stored and added back in before any output is done. In this way the quoted fluxes and displayed spectra are as one would see from the input cube itself – even though the detection (and reconstruction if applicable) is done on the baseline-removed cube.

The presence of very strong signals (for instance, masers at several hundred Jy) could affect the determination of the baseline, and would lead to a large dip centred on the signal in the baseline-subtracted spectrum. To prevent this, the signal is trimmed prior to the reconstruction process at some standard threshold (at 8σ above the mean). The baseline determined should thus be representative of the true, signal-free baseline. Note that this trimming is only a temporary measure which does not affect the source-detection.

3.3.3 Ignoring bright Milky Way emission

Finally, a single set of contiguous channels can be ignored – these may exhibit very strong emission, such as that from the Milky Way as seen in extragalactic HI cubes (hence the references to "Milky Way" in relation to this task – apologies to Galactic astronomers!). Such dominant channels will produce many detections that are unnecessary, uninteresting (if one is interested in extragalactic HI) and large (in size and hence in memory usage), and so will slow the program down and detract from the interesting detections.

The use of this feature is controlled by the `flagMW` parameter, and the exact channels concerned are able to be set by the user (using `maxMW` and `minMW` – these give an inclusive range of channels). These channels refer to the channel numbers of **the full cube**, before any subsection is applied.

The effect is to ignore detections that lie within these channels. If a spatial search is being conducted (i.e. one channel map at a time), these channels are simply not searched. If a spectral search is being conducted, those channels will be flagged so that no detection is made within them. The spectral output (see Fig. 2) will ignore them as far as scaling the plot goes, and the channel range will be indicated by a green hatched box.

Note that these channels will be included in any smoothing or reconstruction that is done on the array, and so will be included in any saved FITS file (see §3.6).

3.4 Image reconstruction

The user can direct *Duchamp* to reconstruct the data cube using the multi-resolution *à trous* wavelet algorithm. A good description of the procedure can be found in Starck and Murtagh (2002). The reconstruction is an effective way of removing a lot of the noise in the image, allowing one to search reliably to fainter levels, and reducing the number of spurious detections. This is an optional step, but one that greatly enhances the source-detection process, at the cost of additional CPU and memory usage (see §5 for discussion).

3.4.1 Algorithm

The steps in the *à trous* reconstruction are as follows:

1. The reconstructed array is set to 0 everywhere.
2. The input array is discretely convolved with a given filter function. This is determined from the parameter file via the `filterCode` parameter – see Appendix B for details on the filters available. Edges are dealt with by assuming reflection at the boundary.
3. The wavelet coefficients are calculated by taking the difference between the convolved array and the input array.
4. If the wavelet coefficients at a given point are above the requested reconstruction threshold (given by `snrRecon` as the number of σ above the mean and adjusted to the current scale – see Appendix H), add these to the reconstructed array.
5. The separation between the filter coefficients is doubled. (Note that this step provides the name of the procedure[5], as gaps or holes are created in the filter coverage.)
6. The procedure is repeated from step 2, using the convolved array as the input array.
7. Continue until the required maximum number of scales is reached.
8. Add the final smoothed (i.e. convolved) array to the reconstructed array. This provides the "DC offset", as each of the wavelet coefficient arrays will have zero mean.

The range of scales at which the selection of wavelet coefficients is made is governed by the `scaleMin` and `scaleMax` parameters. The minimum scale used is given by `scaleMin`, where the default value is 1 (the first scale). This parameter is useful if you want to ignore the highest-frequency features (e.g. high-frequency noise that might be present). Normally the maximum scale is calculated from the size of the input array, but it can be specified by using `scaleMax`. A value ≤ 0 will result in the use of the calculated value, as will a value of `scaleMax` greater than the calculated value. Use of these two parameters can allow searching for features of a particular scale size, for instance searching for narrow absorption features.

The reconstruction has at least two iterations. The first iteration makes a first pass at the wavelet reconstruction (the process outlined in the 8 stages above), but the residual array will likely have some structure still in it, so the wavelet filtering is done on the residual, and any significant wavelet terms are

[5] *à trous* means "with holes" in French.

added to the final reconstruction. This step is repeated until the relative change in the measured standard deviation of the residual (see note below on the evaluation of this quantity) is less than some value, given by the `reconConvergence` parameter.

It is important to note that the *à trous* decomposition is an example of a "redundant" transformation. If no thresholding is performed, the sum of all the wavelet coefficient arrays and the final smoothed array is identical to the input array. The thresholding thus removes only the unwanted structure in the array.

Note that any BLANK pixels that are still in the cube will not be altered by the reconstruction – they will be left as BLANK so that the shape of the valid part of the cube is preserved.

3.4.2 Note on Statistics

The correct calculation of the reconstructed array needs good estimators of the underlying mean and standard deviation (or rms) of the background noise distribution. The methods used to estimate these quantities are detailed in §3.7.3 – the default behaviour is to use robust estimators, to avoid biasing due to bright pixels.

When thresholding the different wavelet scales, the value of the rms as measured from the wavelet array needs to be scaled to account for the increased amount of correlation between neighbouring pixels (due to the convolution). See Appendix H for details on this scaling.

3.4.3 User control of reconstruction parameters

The most important parameter for the user to select in relation to the reconstruction is the threshold for each wavelet array. This is set using the `snrRecon` parameter, and is given as a multiple of the rms (estimated by the MADFM) above the mean (which for the wavelet arrays should be approximately zero). There are several other parameters that can be altered as well that affect the outcome of the reconstruction.

By default, the cube is reconstructed in three dimensions, using a three-dimensional filter and three-dimensional convolution. This can be altered, however, using the parameter `reconDim`. If set to 1, this means the cube is reconstructed by considering each spectrum separately, whereas `reconDim=2` will mean the cube is reconstructed by doing each channel map separately. The merits of these choices are discussed in §5, but it should be noted that a 2-dimensional reconstruction can be susceptible to edge effects if the spatial shape of the pixel array is not rectangular.

The user can also select the minimum and maximum scales to be used in the reconstruction. The first scale exhibits the highest frequency variations, and so ignoring this one can sometimes be beneficial in removing excess noise. The default is to use all scales (`minscale = 1`).

The convergence of the *à trous* iterations is governed by the `reconConvergence` parameter, which is the fractional decrease in the standard deviation of the residuals from one iteration to the next. *Duchamp* will do at least two iterations, and then continue until the decrease is less than the value of this parameter.

Finally, the filter that is used for the convolution can be selected by using `filterCode` and the relevant code number – the choices are listed in Appendix B. A larger filter will give a better reconstruction, but take longer and use more memory when executing. When multi-dimensional reconstruction is selected, this filter is used to construct a 2- or 3-dimensional equivalent.

3.5 Smoothing the cube

An alternative to doing the wavelet reconstruction is to smooth the cube. This technique can be useful in reducing the noise level (at the cost of making neighbouring pixels correlated and blurring any signal present), and is particularly well suited to the case where a particular signal size (i.e. a certain channel width or spatial size) is believed to be present in the data.

There are two alternative methods that can be used: spectral smoothing, using the Hanning filter; or spatial smoothing, using a 2D Gaussian kernel. These alternatives are outlined below. To utilise the smoothing option, set the parameter `flagSmooth=true` and set `smoothType` to either `spectral` or `spatial`.

3.5.1 Spectral smoothing

When `smoothType = spectral` is selected, the cube is smoothed only in the spectral domain. Each spectrum is independently smoothed by a Hanning filter, and then put back together to form the smoothed cube, which is then used by the searching algorithm (see below). Note that in the case of both the reconstruction and the smoothing options being requested, the reconstruction will take precedence and the smoothing will *not* be done.

There is only one parameter necessary to define the degree of smoothing – the Hanning width a (given by the user parameter `hanningWidth`). The coefficients $c(x)$ of the Hanning filter are defined by

$$c(x) = \begin{cases} \frac{1}{2}\left(1+\cos(\frac{\pi x}{a})\right) & |x| < (a+1)/2 \\ 0 & |x| \geq (a+1)/2. \end{cases}, \quad a, x \in \mathbb{Z}$$

Note that the width specified must be an odd integer (if the parameter provided is even, it is incremented by one).

3.5.2 Spatial smoothing

When `smoothType = spatial` is selected, the cube is smoothed only in the spatial domain. Each channel map is independently smoothed by a two-dimensional Gaussian kernel, put back together to form the smoothed cube, and used in the searching algorithm (see below). Again, reconstruction is always done by preference if both techniques are requested.

The two-dimensional Gaussian has three parameters to define it, governed by the elliptical cross-sectional shape of the Gaussian function: the FWHM (full-width at half-maximum) of the major and minor axes, and the position angle of the major axis. These are given by the user parameters `kernMaj, kernMin` & `kernPA`. If a circular Gaussian is required, the user need only provide the `kernMaj` parameter. The `kernMin` parameter will then be set to the same value, and `kernPA` to zero. If we define these parameters as a, b, θ respectively, we can define the kernel by the function

$$k(x,y) = \exp\left[-0.5\left(\frac{X^2}{\sigma_X^2} + \frac{Y^2}{\sigma_Y^2}\right)\right]$$

where (x, y) are the offsets from the central pixel of the gaussian function, and

$$X = x\sin\theta - y\cos\theta \qquad Y = x\cos\theta + y\sin\theta$$

$$\sigma_X^2 = \frac{(a/2)^2}{2\ln 2} \qquad \sigma_Y^2 = \frac{(b/2)^2}{2\ln 2}$$

3.6 Input/Output of reconstructed/smoothed arrays

The smoothing and reconstruction stages can be relatively time-consuming, particularly for large cubes and reconstructions in 3-D (or even spatial smoothing). To get around this, *Duchamp* provides a shortcut to allow users to perform multiple searches (e.g. with different thresholds) on the same reconstruction/smoothing setup without re-doing the calculations each time.

To save the reconstructed array as a FITS file, set `flagOutputRecon = true`. The file will be saved in the same directory as the input image, so the user needs to have write permissions for that directory.

The name of the file can given by the `fileOutputRecon` parameter, but this can be ignored and *Duchamp* will present a name based on the reconstruction parameters. The filename will be derived from the input filename, with extra information detailing the reconstruction that has been done. For example, suppose `image.fits` has been reconstructed using a 3-dimensional reconstruction with filter #2, thresholded at 4σ using all scales from 1 to 5, with a convergence criterion of 0.005. The output filename will then be `image.RECON-3-2-4-1-5-0.005.fits` (i.e. it uses the six parameters relevant for the *à trous* reconstruction as listed in Appendix B). The new FITS file will also have these parameters as header keywords. If a subsection of the input image has been used (see §3.1), the format of the output filename will be `image.sub.RECON-3-2-4-1-5-0.005.fits`, and the subsection that has been used is also stored in the FITS header.

Likewise, the residual image, defined as the difference between the input and reconstructed arrays, can also be saved in the same manner by setting `flagOutputResid = true`. Its filename will be the same as above, with `RESID` replacing `RECON`.

If a reconstructed image has been saved, it can be read in and used instead of redoing the reconstruction. To do so, the user should set the parameter `flagReconExists = true`. The user can indicate the name of the reconstructed FITS file using the `reconFile` parameter, or, if this is not specified, *Duchamp* searches for the file with the name as defined above. If the file is not found, the reconstruction is performed as normal. Note that to do this, the user needs to set `flagAtrous = true` (obviously, if this is `false`, the reconstruction is not needed).

To save the smoothed array, set `flagOutputSmooth = true`. As for the reconstructed/residual arrays, the name of the file can given by the parameter `fileOutputSmooth`, but this can be ignored and *Duchamp* will present a name

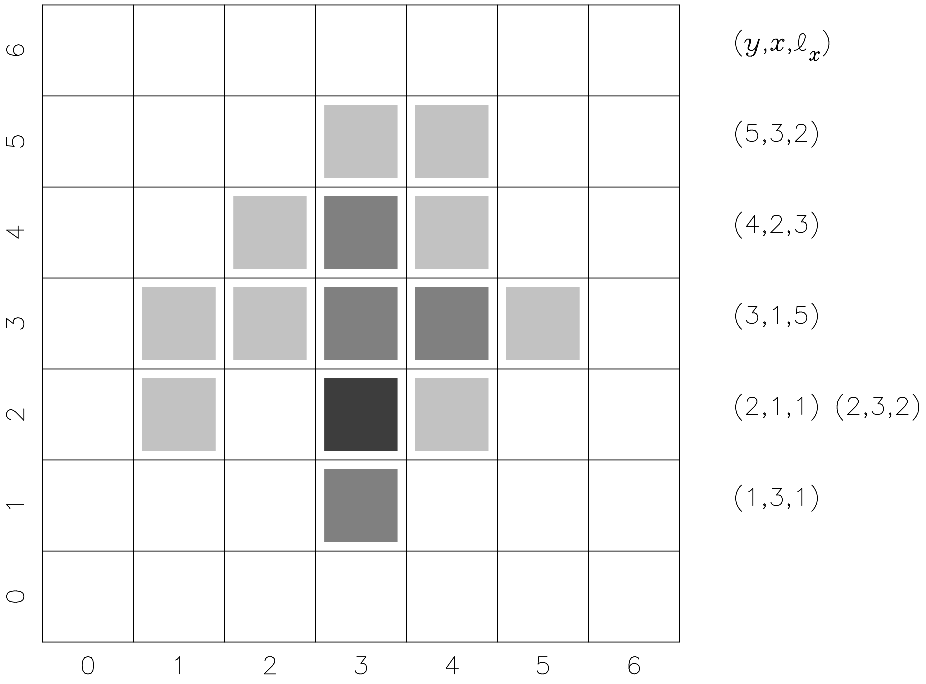


Figure 1: An example of the run-length encoding method of storing pixel information. The scans used to encode the image are listed alongside the relevant row. The pixels are colour-coded by nominal pixel values, but note that the pixel values themselves do not form part of the encoding and are not kept as part of the object class.

that indicates the both the type and the details of the smoothing method used. It will be either `image.SMOOTH-1D-a.fits`, where a is replaced by the Hanning width used, or `image.SMOOTH-2D-a-b-c.fits`, where the Gaussian kernel parameters are a,b,c. Similarly to the reconstruction case, a saved file can be read in by setting `flagSmoothExists = true` and either specifying a file to be read with the `smoothFile` parameter or relying on *Duchamp* to find the file with the name as given above.

3.7 Searching the image

3.7.1 Representation of detected objects

The principle aim of *Duchamp* is to provide a catalogue of sources located in the image. While running, *Duchamp* needs to maintain for each source several data structures that will contribute to the memory footprint: a record of which pixels contribute to the source; a set of measured parameters that will go into the catalogue; and a separate two-dimensional map showing the spatial location of detected pixels (carrying this around makes the computation of detection maps easier – see §4.3.2).

To keep track of the set of detected pixels, *Duchamp* employs specialised techniques that keep the memory usage manageable. A naive method could be to store each single pixel, but this results in a lot of redundant information being stored in memory.

To reduce the storage requirements, the run-length encoding method is used for storing the spatial information. In this fashion, an object in 2D is stored as a series of "runs", encoded by a row number (the y-value), the starting column (the minimum x-value) and the run length (ℓ_x: the number of contiguous pixels in that row connected to the starting pixel). A single set of (y, x, ℓ_x) values is called a "scan". A two-dimensional image is therefore made up of a set of scans. An example can be seen in Fig. 1. Note that the object shown has fourteen pixels, and so would require 28 integers to record the positions of all pixels. The run-length encoding uses just 18 integers to record the same information. The longer the runs are in each scan, the greater the saving of storage over the naive method.

A 3D object is stored as a set of channel maps, with a channel map being a 2D plane with constant z-value. Each channel map is itself a set of scans showing the (x, y) position of the pixels. The additional detection map is stored as a separate channel map, also made up of scans.

Note that these pixel map representations do not carry the flux information with them. They store just the pixel locations and need to be combined with an array of flux values to provide parameters such as integrated flux. The advantage of this approach is that the pixel locations can be easily applied to different flux arrays as the need permits (for instance, defining them using the reconstructed array, yet evaluating parameters on the original array).

3.7.2 Technique

The basic idea behind detection in *Duchamp* is to locate sets of contiguous voxels that lie above some threshold. No size or shape requirement is imposed upon the detections, and no fitting (for instance, fitting Gaussian profiles) is done on the sources. All *Duchamp* does is find connected groups of bright voxels and report their locations and basic parameters.

One threshold is calculated for the entire cube, enabling calculation of signal-to-noise ratios for each source (see §4 for details). The user can manually specify a value (using the parameter `threshold`) for the threshold, which will override the calculated value. Note that this option overrides any settings of `snrCut` or FDR options (see below).

The cube can be searched in one of two ways, governed by the input parameter `searchType`. If `searchType=spatial`, the cube is searched one channel map at a time, using the 2-dimensional raster-scanning algorithm of Lutz (1980) that connects groups of neighbouring pixels. Such an algorithm cannot be applied directly to a 3-dimensional case, as it requires that objects are completely nested in a row (when scanning along a row, if an object finishes and other starts, you won't get back to the first until the second is completely finished for the row). Three-dimensional data does not have this property, hence the need to treat the data on a 2-dimensional basis at most.

Alternatively, if `searchType=spectral`, the searching is done in one dimension on each individual spatial pixel's spectrum. This is a simpler search, but there are potentially many more of them.

Although there are parameters that govern the minimum number of pixels in a spatial, spectral and total senses that an object must have (`minPix`, `minChannels` and `minVoxels` respectively), these criteria are not applied at this point - see §3.8.4 for details.

Finally, the search only looks for positive features. If one is interested instead in negative features (such as absorption lines), set the parameter `flagNegative = true`. This will invert the cube (i.e. multiply all pixels by -1) prior to the search, and then re-invert the cube (and the fluxes of any detections) after searching is complete. If the reconstructed or smoothed array has been read in from disk, this will also be inverted at the same time. All outputs are done in the same manner as normal, so that fluxes of detections will be negative.

3.7.3 Calculating statistics

A crucial part of the detection process (as well as the wavelet reconstruction: §3.4) is estimating the statistics that define the detection threshold. To determine a threshold, we need to estimate from the data two parameters: the middle of the noise distribution (the "noise level"), and the width of the distribution (the "noise spread"). The noise level is estimated by either the mean or the median, and the noise spread by the rms (or the standard deviation) or the median absolute deviation from the median (MADFM). The median and MADFM are robust statistics, in that they are not biased by the presence of a few pixels much brighter than the noise.

All four statistics are calculated automatically, but the choice of parameters that will be used is governed by the input parameter `flagRobustStats`. This has the default value `true`, meaning the underlying mean of the noise distribution is estimated by the median, and the underlying standard deviation is estimated by the MADFM. In the latter case, the value is corrected, under the assumption that the underlying distribution is Normal (Gaussian), by dividing by 0.6744888 – see Appendix G for details. If `flagRobustStats=false`, the mean and rms are used instead.

The choice of pixels to be used depend on the analysis method. If the wavelet reconstruction has been done, the residuals (defined in the sense of original $-$ reconstruction) are used to estimate the noise spread of the cube, since the reconstruction should pick out all significant structure. The noise level (the middle of the distribution) is taken from the original array.

If smoothing of the cube has been done instead, all noise parameters are measured from the smoothed array, and detections are made with these parameters. When the signal-to-noise level is quoted for each detection (see §4), the noise parameters of the original array are used, since the smoothing process correlates neighbouring pixels, reducing the noise level.

If neither reconstruction nor smoothing has been done, then the statistics are calculated from the original, input array.

The parameters that are estimated should be representative of the noise in the cube. For the case of small objects embedded in many noise pixels (e.g. the case of HI surveys), using the full cube will provide good estimators. It is possible, however, to use only a subsection of the cube by setting the parameter `flagStatSec = true` and providing the desired subsection to the `StatSec` parameter. This subsection works in exactly the same way as the pixel subsection discussed in §3.1. The `StatSec` will be trimmed if necessary so that it lies wholly within the image subsection being used (i.e. that given by the `subsection` parameter - this governs what pixels are read in and so are able to be used in the calculations).

Note that `StatSec` applies only to the statistics used to determine the threshold. It does not affect the calculation of statistics in the case of the wavelet reconstruction. Note also that pixels flagged as BLANK or as part of the "Milky Way" range of channels are ignored in the statistics calculations.

3.7.4 Determining the threshold

Once the statistics have been calculated, the threshold is determined in one of two ways. The first way is a simple sigma-clipping, where a threshold is set at a fixed number n of standard deviations above the mean, and pixels above this threshold are flagged as detected. The value of n is set with the parameter `snrCut`. The "mean" and "standard deviation" here are estimated according to `flagRobustStats`, as discussed in §3.7.3. In this first case only, if the user specifies a threshold, using the `threshold` parameter, the sigma-clipped value is ignored.

The second method uses the False Discovery Rate (FDR) technique (Hopkins et al. 2002; Miller et al. 2001), whose basis we briefly detail here. The false discovery rate (given by the number of false detections divided by the total number of detections) is fixed at a certain value α (e.g. $\alpha = 0.05$ implies 5% of detections are false positives). In practice, an α value is chosen, and the ensemble average FDR (i.e. $\langle FDR \rangle$) when the method is used will be less than α. One calculates p – the probability, assuming the null hypothesis is true, of obtaining a test statistic as extreme as the pixel value (the observed test statistic) – for each pixel, and sorts them in increasing order. One then calculates d where

$$d = \max_j \left\{ j : P_j < \frac{j\alpha}{c_N N} \right\},$$

and then rejects all hypotheses whose p-values are less than or equal to P_d. (So a $P_i < P_d$ will be rejected even if $P_i \geq j\alpha / c_N N$.) Note that "reject hypothesis" here means "accept the pixel as an object pixel" (i.e. we are rejecting the null hypothesis that the pixel belongs to the background).

The c_N value here is a normalisation constant that depends on the correlated nature of the pixel values. If all the pixels are uncorrelated, then $c_N = 1$. If N pixels are correlated, then their tests will be dependent on each other, and so $c_N = \sum_{i=1}^{N} i^{-1}$. Hopkins et al. (2002) consider real radio data, where the pixels are correlated over the beam. For the calculations done in *Duchamp*, $N = B \times C$, where B is the beam area in pixels, calculated from the FITS header (if the correct keywords – BMAJ, BMIN – are not present, the size of the beam is taken from the input parameters - see discussion in §4.2.1, and if these parameters are not given, $B = 1$), and C is the number of neighbouring channels that can be considered to be correlated.

The use of the FDR method is governed by the `flagFDR` flag, which is `false` by default. To set the relevant parameters, use `alphaFDR` to set the α value, and `FDRnumCorChan` to set the C value discussed above. These have default values of 0.01 and 2 respectively.

The theory behind the FDR method implies a direct connection between the choice of α and the fraction of detections that will be false positives. These detections, however, are individual pixels, which undergo a process of merging and rejection (§3.8), and so the fraction of the final list of detected objects that are false positives will be much smaller than α. See the discussion in §5.

3.8 Merging, growing and rejecting detected objects

3.8.1 Merging

The searches described above are either 1- or 2-dimensional only. They do not know anything about the third dimension that is likely to be present. To build up 3D sources, merging of detections must be done. This is done via an algorithm that matches objects judged to be "close", according to one of two criteria.

One criterion is to define two thresholds – one spatial and one in velocity – and say that two objects should be merged if there is at least one pair of pixels that lie within these threshold distances of each other. These thresholds are specified by the parameters `threshSpatial` and `threshVelocity` (in units of pixels and channels respectively).

Alternatively, the spatial requirement can be changed to say that there must be a pair of pixels that are *adjacent* – a stricter, but perhaps more realistic requirement, particularly when the spatial pixels have a large angular size (as is the case for HI surveys). This method can be selected by setting the parameter `flagAdjacent=true` in the parameter file. The velocity thresholding is always done with the `threshVelocity` test.

3.8.2 Stages of merging

This merging can be done in two stages. The default behaviour is for each new detection to be compared with those sources already detected, and for it to be merged with the first one judged to be close. No other examination of the list is done at this point.

This step can be turned off by setting `flagTwoStageMerging=false`, so that new detections are simply added to the end of the list, leaving all merging to be done in the second stage.

The second, main stage of merging is more thorough, Once the searching is completed, the list is iterated through, looking at each pair of objects, and merging appropriately. The merged objects are then included in the examination, to see if a merged pair is suitably close to a third.

3.8.3 Growing

Once the detections have been merged, they may be "grown" (this is essentially the process known elsewhere as "floodfill"). This is a process of increasing the size of the detection by adding nearby pixels (according to the `threshSpatial` and `threshVelocity` parameters) that are above some secondary threshold and not already part of a detected object. This threshold should be lower than the one used for the initial detection, but above the noise level, so that faint pixels are only detected when they are close to a bright pixel. This threshold is specified via one of two input parameters. It can be given in terms of the noise statistics via `growthCut` (which has a default value of 3σ), or it can be directly given via `growthThreshold`. Note that if you have given the detection threshold with the `threshold` parameter, the growth threshold **must** be given with `growthThreshold`. If `growthThreshold` is not provided in this situation, the growing will not be done.

The use of the growth algorithm is controlled by the `flagGrowth` parameter – the default value of which is `false`. If the detections are grown, they are sent through the merging algorithm a second time, to pick up any detections that should be merged at the new lower threshold (i.e. they have grown into each other).

3.8.4 Rejecting

Finally, to be accepted, the detections must satisfy minimum size criteria, relating to the number of channels, spatial pixels and voxels occupied by the object. These criteria are set using the `minChannels`, `minPix` and `minVoxels` parameters respectively. The channel requirement means a source must have at least one set of `minChannels` consecutive channels to be accepted. The spatial pixels (`minPix`) requirement refers to distinct spatial pixels (which are possibly in different channels), while the voxels requirement refers to the total number of voxels detected. If the `minVoxels` parameter is not provided, it defaults to `minPix`+`minChannels`-1.

It is possible to do this rejection stage before the main merging and growing stage. This could be done to remove narrow (hopefully spurious) sources from the list before growing them, to reduce the number of false positives in the final list. This mode can be selected by setting the input parameter `flagRejectBeforeMerge=true` – caution is urged if you use this in conjunction with `flagTwoStageMerging=false`, as you can throw away parts of objects that you may otherwise wish to keep.

4 Outputs

4.1 During execution

Duchamp provides the user with feedback whilst it is running, to keep the user informed on the progress of the analysis. Most of this consists of self-explanatory messages about the particular stage the program is up to. The relevant parameters are printed to the screen at the start (once the file has been successfully read in), so the user is able to make a quick check that the setup is correct (see Appendix C for an example).

The extent of memory allocation made at the start is indicated. This will include the arrays needed for the pixel array, the reconstruction or smoothed array, and the 2D detection map, but *not* additional space needed for working within individual algorithms, nor storage needed for the detected objects.

Duchamp will report the amount of memory that is allocated when the image is read in. This includes the storage for the array as well as additional storage for the reconstructed/smoothed array and/or the baseline arrays (if these are needed).

If the cube is being trimmed (§3.3), the resulting dimensions are printed to indicate how much has been trimmed. If a reconstruction is being done, a continually updating message shows either the current iteration and scale, compared to the maximum scale (when `reconDim = 3`), or a progress bar showing the amount of the cube that has been reconstructed (for smaller values of `reconDim`).

During the searching algorithms, the progress through the search is shown. When completed, the number of objects found is reported (this is the total number found, before any merging is done).

In the merging process (where multiple detections of the same object are combined – see §3.8), two stages of output occur. The first is when each object in the list is compared with all others. The output shows two numbers: the first being how far through the list the current object is, and the second being the length of the list. As the algorithm proceeds, the first number should increase and the second should decrease (as objects are combined). When the numbers meet, the whole list has been compared. If the objects are being grown, a similar output is shown, indicating the progress through the list. In the rejection stage, in which objects not meeting the minimum pixels/channels requirements are removed, the total number of objects remaining in the list is shown, which should steadily decrease with each rejection until all have been examined. Note that these steps can be very quick for small numbers of detections.

Since this continual printing to screen has some overhead of time and CPU involved, the user can elect to not print this information by setting the parameter `verbose = false`. In this case, the user is still informed as to the steps being undertaken, but the details of the progress are not shown.

There may also be Warning or Error messages printed to screen. The Warning messages occur when something happens that is unexpected (for instance, a desired keyword is not present in the FITS header), but not detrimental to the execution. An Error message is something more serious, and indicates some part of the program was not able to complete its task. This is not necessary fatal, but it may mean the full functionality requested will not be achieved. The message will also indicate which function or subroutine generated it – this is primarily a tool for debugging, but can be useful in determining what went wrong.

4.2 Text-based output files

4.2.1 Table of results

Finally, we get to the results – the reason for running *Duchamp* in the first place. Once the detection list is finalised, it is sorted according to the value of the `sortingParam`. This can take the value "xvalue", "yvalue", "zvalue", "ra", "dec", "vel", "w50", "iflux" (for integrated flux), or "pflux" (for peak flux), or "snr". The default value is "vel" (which means the spectral WCS value – this could be frequency or wavelength depending on the nature of the FITS file). If no good WCS exists, the mean pixel position equivalent is used ("ra" is replaced by "xvalue", "dec" by "yvalue", "vel" and "w50" by "zvalue"). The sense of the sorting will be increasing value with position in the list. To sort in the opposite sense, prepend the parameter name with a '-' (e.g. "-vel" instead of "vel"). The results are then printed to the screen and to the output file, given by the `OutFile` parameter.

The output consists of two sections. First, a list of the parameters are printed to the output file, for future reference. Next, the detection threshold that was used is given, so comparison can be made with other searches. The statistics estimating the noise parameters are given (see §3.7.3). Thirdly, the number of detections are reported.

All this information, known as the "header", can either be written to the start of the output file (denoted by the parameter `OutFile`), or written to a separate file from the list of detections. This second option is activated by the parameter `flagSeparateHeader`, and the information is written to the file given by `HeaderFile`.

The most interesting part, however, is the list of detected objects. This list, an example of which can be seen in Appendix D, contains the following columns (note that the title of the columns depending on WCS information will depend on the details of the WCS projection: they are shown below for the Equatorial and Galactic projection cases).

`Obj#:` The ID number of the detection (simply the sequential count for the list, which is ordered by increasing velocity, or channel number, if the WCS is not good enough to find the velocity).

`Name:` The "IAU"-format name of the detection (derived from the WCS position – see below for a description of the format).

`X,Y,Z:` The "centre" pixel position, determined by the input parameter `pixelCentre`.

`RA/GLON:` The Right Ascension or Galactic Longitude of the centre of the object.

`DEC/GLAT:` The Declination or Galactic Latitude of the centre of the object.

`VEL:` The mean spectral coordinate of the object [units given by the `spectralUnits` parameter]. The actual values here will depend on the WCS of the FITS file, and the name of the column will come from the CTYPE of the spectral axis (specifically, the 4-character S-type code).

`w_RA/w_GLON:` The width of the object in Right Ascension or Galactic Longitude (depending on FITS coordinates) [arcmin].

`w_DEC/w_GLAT:` The width of the object in Declination Galactic Latitude [arcmin].

`w_50:` The spectral width of the detection at 50% of the peak flux (the measured full-width at half-maximum, FWHM), in the appropriate spectral units [see note below].

`F_int:` The integrated flux over the object, in the units of flux times velocity, corrected for the beam if necessary. See below for a discussion on this correction.

`F_peak:` The peak flux over the object, in the units of flux.

`S/Nmax:` The signal-to-noise ratio at the peak pixel.

`X1, X2:` The minimum and maximum X-pixel coordinates.

`Y1, Y2:` The minimum and maximum Y-pixel coordinates.

`Z1, Z2:` The minimum and maximum Z-pixel coordinates.

`Npix:` The number of voxels (i.e. distinct (x, y, z) coordinates) in the detection.

`Flag:` Whether the detection has any warning flags (see below).

These parameters are written to the screen during execution. There are alternative ways of calculating the total flux, the position and velocity width, however, and so additional parameters are written to the output file:

`w_20:` The spectral width of the detection at 20% of the peak flux, in spectral units [see note below].

`w_VEL:` The full spectral width of the detection (max channel − min channel, in the appropriate spectral units).

`F_tot:` The sum of the flux values of all detected voxels.

`X_av, Y_av, Z_av:` The average pixel value in each axis direction i.e. X_av is the average of the x-values of all pixels in the detection.

`X_cent, Y_cent, Z_cent:`
The centroid position, being the flux-weighted average of the pixels.

`X_peak, Y_peak, Z_peak:`
The location of the pixel containing the peak flux value.

The velocity width of the detection is calculated at 50% and 20% of the peak flux, as well as the full detected width (if the detection threshold is greater than 20% or 50% of the peak, then these values will be the same as `w_VEL`. The type of position value given in the `X, Y, Z` columns in the screen output is determined by the `pixelCentre` parameter. All three alternatives are shown in the output file.

The user can specify the precision used to display the flux, velocity and S/Nmax values, by using the input parameters `precFlux`, `precVel` and `precSNR` respectively. These values apply to the tables written to the screen and to the output file, as well as for the VOTable (see below).

The `Name` is derived from the WCS position. For instance, a source that is centred on the RA,Dec position $12^h53^m45^s$, -36°24′12″ will be given the name J125345−362412, if the epoch is J2000, or the name B125345−362412 if it is B1950. The precision of the RA and Dec values is determined by the pixel size, such that sufficient precision is used to uniquely define a position. The RA value will have one figure greater precision than Dec. An alternative form is used for Galactic coordinates: a source centred on the position $(l,b) = (323.1245, 5.4567)$ will be called G323.124+05.457. If the WCS is not valid (i.e. is not present or does not have all the necessary information), the `Name, RA, DEC, VEL` and related columns are not printed, but the pixel coordinates are still provided.

The velocity units can be specified by the user, using the input parameter `spectralUnits` (enter it as a single string with no spaces). The default value comes from the FITS header. These units are also used to give the units of integrated flux.

When the cube brightness units are quoted per beam (e.g. Jy/beam), then the integrated flux `F_int` includes a correction for this. This involves dividing by the integral over the beam. This is calculated using the BMAJ, BMIN & BPA header keywords from the FITS file. BMAJ and BMIN are assumed to be the full-width at half maximum (FWHM) in the major and minor axis directions of a Gaussian beam. The integral is calculated as follows: the functional form of a 2D elliptical Gaussian can be written as $\exp(-((x^2/2\sigma_x^2) + (y^2/2\sigma_y^2)))$, and the FWHM in a given direction is then $f = 2\sqrt{2\ln 2}\sigma$. Then,

$$\int \exp\left(-\left(\frac{x^2}{2\sigma_x^2} + \frac{y^2}{2\sigma_y^2}\right)\right) = 2\pi\sigma_x\sigma_y = \frac{\pi f_x f_y}{4\ln 2}$$

provides the correction factor to go from units of Jy/beam to Jy.

If the FITS file does not have the beam information, the user can either:

1. Specify the FWHM of the beam in pixels (assuming a circular beam) via the parameter `beamFWHM`.

2. Specify the area of the beam, again in pixels, via the parameter `beamArea`[6]. This should be the value given by the equation above.

If both are given, `beamFWHM` takes precendence. If neither are given, and there is no beam information in the header, then no correction to the integrated flux is made (and so it will stay in units of Jy/beam or equivalent).

If the WCS is absent or not sufficiently specified, then all columns from `RA/GLON` to `w_VEL` will be left blank. Also, `F_int` will be replaced with the more simple `F_tot`.

The `Flag` column contains any warning flags, such as:

- **E** – The detection is next to the spatial edge of the image, meaning either the limit of the pixels, or the limit of the non-BLANK pixel region.

[6] Note that this is equivalent to the old parameter `beamSize`, which was highlighted as being ambiguous.

- **S** – The detection lies at the edge of the spectral region.
- **M** – The detection is adjacent to, or overlaps the "Milky Way" range of channels (see Sec. 3.3.3).
- **N** – The total flux, summed over all the (non-BLANK) pixels in the smallest box that completely encloses the detection, is negative. Note that this sum is likely to include non-detected pixels. It is of use in pointing out detections that lie next to strongly negative pixels, such as might arise due to interference – the detected pixels might then also be due to the interference, so caution is advised.

In the absence of any of these flags, a `-` will be recorded in this column.

4.2.2 Other results lists

Three additional results files can also be requested. One option is a VOTable-format XML file, containing just the RA, Dec, Velocity and the corresponding widths of the detections, as well as the fluxes. The user should set `flagVOT = true`, and put the desired filename in the parameter `votFile` – note that the default is for it not to be produced. This file should be compatible with all Virtual Observatory tools (such as Aladin[7] or TOPCAT[8]).

A second option is an annotation file for use with the Karma toolkit of visualisation tools (in particular, with `kvis`). There are two options on how objects are represented, governed by the `annotationType` parameter. Setting this to `borders` results in a border being drawn around the spatial pixels of the object, in a manner similar to that seen in Fig. 2. Note that Karma/`kvis` does not always do this perfectly, so the lines may not be directly lined up with pixel borders. The other option is `annotationType = circles`. This will draw a circle at the position of each detection, scaled by the spatial size of the detection, and number it according to the Obj# given above. To make use of this option, the user should set `flagKarma = true`, and put the desired filename in the parameter `karmaFile` – again, the default is for it not to be produced.

The final optional results file produced is a simple text file that contains the spectra for each detected object. The format of the file is as follows: the first column has the spectral coordinate, over the full range of values; the remaining columns represent the flux values for each object at the corresponding spectral position. The flux value used is the same as that plotted in the spectral plot detailed below, and governed by the `spectralMethod` parameter. An example of what a spectral text file might look like is given below:

[7] http://aladin.u-strasbg.fr/

[8] http://www.star.bristol.ac.uk/ mbt/topcat/

```
1405.00219727  0.01323344  0.23648241  0.04202826  -0.00506790
1405.06469727  0.01302835  0.27393046  0.04686056  -0.00471084
1405.12719727  0.01583361  0.27760920  0.04114933  -0.01168737
1405.18969727  0.01271889  0.31489247  0.03307962  -0.00300790
1405.25219727  0.01597644  0.30401203  0.05356426  -0.00551653
1405.31469727  0.00773827  0.30031312  0.04074831  -0.00570147
1405.37719727  0.00738304  0.27921870  0.05272378  -0.00504959
1405.43969727  0.01353923  0.26132512  0.03667958  -0.00151006
1405.50219727  0.01119724  0.28987029  0.03497849  -0.00645589
1405.56469727  0.00813379  0.29839963  0.04711142  0.00536576
1405.62719727  0.00774377  0.26530230  0.04620502  0.00724631
1405.68969727  0.00576067  0.23215000  0.04995513  0.00290841
1405.75219727  0.00452834  0.16484940  0.04261605  -0.00612812
1405.81469727  0.01406293  0.15989439  0.03817926  -0.00758385
1405.87719727  0.01116611  0.11890682  0.05499069  -0.00626362
1405.93969727  0.00687582  0.10620256  0.04743370  0.00055177
      ⋮             ⋮           ⋮           ⋮           ⋮
```

In addition to these three files, a log file can also be produced. As the program is running, it also (optionally) records the detections made in each individual spectrum or channel (see §3.7 for details on this process). This is recorded in the file given by the parameter `LogFile`. This file does not include the columns `Name, RA, DEC, w_RA, w_DEC, VEL, w_VEL`. This file is designed primarily for diagnostic purposes: e.g. to see if a given set of pixels is detected in, say, one channel image, but does not survive the merging process. The list of pixels (and their fluxes) in the final detection list are also printed to this file, again for diagnostic purposes. The file also records the execution time, as well as the command-line statement used to run *Duchamp*. The creation of this log file can be prevented by setting `flagLog = false` (which is the default).

4.3 Graphical output

4.3.1 Spectral plots

As well as the output data file, a postscript file (with the filename given by the `spectralFile` parameter) is created that shows the spectrum for each detection, together with a small cutout image (the 0th moment) and basic information about the detection (note that any flags are printed after the name of the detection, in the format `[E]`). If the cube was reconstructed, the spectrum from the reconstruction is shown in red, over the top of the original spectrum. The spectral extent of the detected object is indicated by two dashed blue lines, and the region covered by the "Milky Way" channels is shown by a green hashed box. An example detection can be seen in Fig. 2.

The spectrum that is plotted is governed by the `spectralMethod` parameter. It can be either `peak` (the default), where the spectrum is from the spatial pixel containing the detection's peak flux; or `sum`, where the spectrum is summed over all spatial pixels, and then corrected for the beam size. If the `peak` method is used, the detection threshold (and growth threshold, if used) are indicated by dashed (and dotted) lines. These cannot be plotted on the integrated spectrum. The spectral extent of the detection is indicated with blue lines, and a zoom is shown in a separate window.

The cutout image can optionally include a border around the spatial pixels that are in the detection (turned on and off by the `drawBorders` parameter – the default is `true`). It includes a scale bar in the bottom left corner to indicate

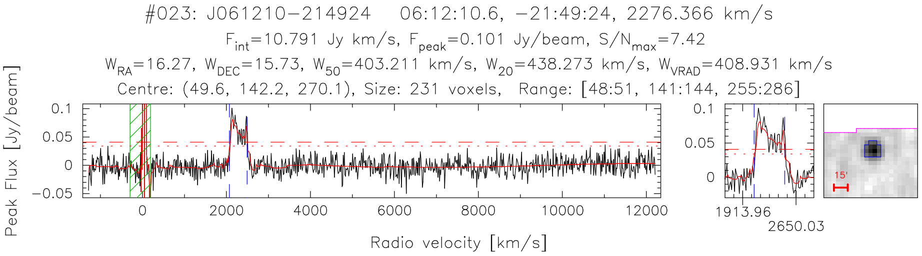


Figure 2: An example of the spectral output. Note several of the features discussed in the text: the red solid lines indicating the reconstructed spectrum; the red dashed and dotted horizontal lines indicating the detection and growth thresholds respectively; the blue dashed lines indicating the spectral extent of the detection; the green hashed area indicating the Milky Way channels that are ignored by the searching algorithm; the blue border showing its spatial extent on the 0th moment map; and the 15 arcmin-long scale bar.

size – its length is indicated next to it (the choice of length depends on the size of the image).

There may also be one or two extra lines on the image. A yellow line shows the limits of the cube's spatial region: when this is shown, the detected object will lie close to the edge, and the image box will extend outside the region covered by the data. A purple line, however, shows the dividing line between BLANK and non-BLANK pixels. The BLANK pixels will always be shown in black. The first type of line is always drawn, while the second is governed by the parameter `drawBlankEdges` (whose default is `true`), and obviously whether there are any BLANK pixel present.

Note that the creation of the spectral plots can be prevented by setting `flagPlotSpectra = false`.

When the input image is two-dimensional, with no spectral dimension, this spectral plot would not make much sense. Instead, *Duchamp* creates a similar postscript file that simply includes the text headers as well as the 0th-moment map of the detection. As for the normal spectral case, this file will be written to the filename given by the `spectralFile` parameter.

When the input image is one-dimensional, the spectral plot is identical save for the absence of the cutout image.

4.3.2 Spatial maps

Additionally, two types of spatial images are optionally produced: a combined 0th-moment map of the cube, combining just the detected channels in each object, showing the integrated flux in grey-scale; and a "detection image", a grey-scale image where the pixel values are the number of channels in which that spatial pixel is detected. These detections include pixels that are subsequently discarded (due to the minimum-size criteria). In both cases, if `drawBorders = true`, a border is drawn around the spatial extent of each detection, and if `drawBlankEdges = true`, the purple line dividing BLANK and non-BLANK pixels (as described above) is drawn. An example moment map is shown in Fig. 3. The production or otherwise of these images is governed by the `flagMaps` parameter.

The moment map is also displayed in a PGPlot XWindow (with the `/xs`

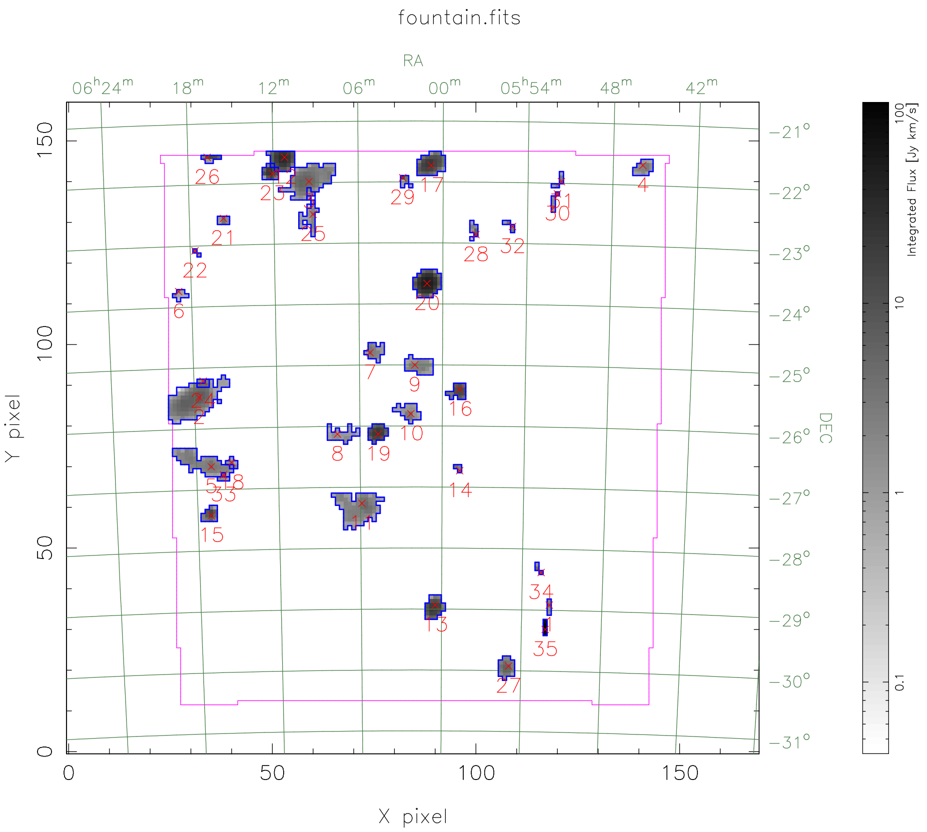


Figure 3: An example of the moment map created by *Duchamp*. The full extent of the cube is covered, and the 0th moment of each object is shown (integrated individually over all the detected channels). The purple line indicates the limit of the non-BLANK pixels.

display option). This feature can be turned off by setting `flagXOutput = false` – this might be useful if running *Duchamp* on a terminal with no window display capability, or if you have set up a script to run it in a batch mode.

If the input image is one-dimensional, such a spatial map is not possible. Instead, the detection map becomes a detection spectrum. This shows the full spectral range, indicating (as for the spectral plots above) the detection and growth thresholds, as well as the 'Milky Way' range and every detection that appears in the final catalogue. It also indicates all pixels that were detected, including those subsequently discarded, by thick black lines above the spectrum. An example can be see in Fig. 4. Again, this plot is also displayed in a PGPlot XWindow.

The purpose of these images is to provide a visual guide to where the detections have been made, and, particularly in the case of the moment map, to provide an indication of the strength of the source. In both cases, the detections are numbered (in the same sense as the output list and as the spectral plots), and the spatial borders are marked out as for the cutout images in the spectra file. Both these images are saved as postscript files (given by the parameters `momentMap` and `detectionMap` respectively), with the latter also displayed in a

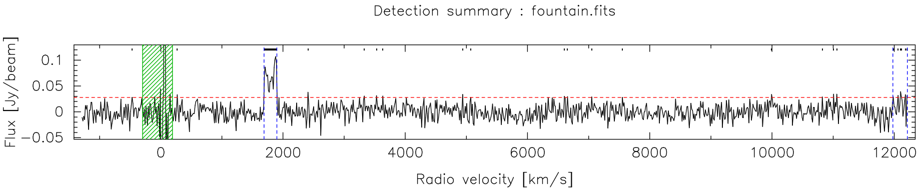


Figure 4: An example of the one-dimensional detection spectrum plot, indicating detected sources and detected pixels, including those subsquently discarded due to the minimum-size criteria. The detection threshold is low to show the effect of detecting lots of single-pixel channels, which are then discarded, leaving just the two detections delimited by the blue lines.

PGPLOT window (regardless of the state of `flagMaps`).

4.4 FITS output

4.4.1 Moment map

The moment map described above can also be written to a FITS file, so that it can be examined more closely, and have annotation files overlaid. This works in the same way as for the mask image. To create the FITS file, set the input parameter `flagOutputMomentMap=true`. The file will be named according to the `fileOutputMomentMap` parameter, or, if this is not given, `image.MOM0.fits` (where the input image is called `image.fits`).

4.4.2 Mask image

It is also possible to write the mask array to a FITS file, for use in other forms of post-processing. This array is designed to indicate the location of detected objects. The value of the detected pixels is determined by the input parameter `flagMaskWithObjectNum`: if `true`, the value of the pixels is given by the corresponding object ID number; if `false`, they take the value 1 for all objects. Pixels not in a detected object have the value 0. To create this FITS file, set the input parameter `flagOutputMask=true`. The file will be named according to the `fileOutputMask` parameter, or, if this is not given, `image.MASK.fits` (where the input image is called `image.fits`).

4.4.3 Smoothed or Reconstructed image

As discussed in §3.6, the reconstructed array, its residual, or the smoothed array can be saved to a FITS file. This allows examination of them offline, as well as their re-use by *Duchamp* to save the expense of re-calculating. This behaviour is controlled by `flagOutputRecon`, `flagOutputResid` and `flagOutputSmooth`. Consult §3.6 for further details.

4.5 Re-using previous detections

It may be the case that, once you have run *Duchamp* with a set of parameters, you are unsatisfied with the output spectra – perhaps you would have preferred integrated rather than peak flux to be plotted. However, the searching might have taken a while to run, and the thought of doing it again just for different plots may be a bit off-putting.

Well, provided you have made a log file when running *Duchamp* (with the `flagLog=true` setting), it is possible to do this easily without having to go through the process of detecting your sources a second time. By using the same input file, with the additional parameter `usePrevious=true`, the log file that was created with a previous *Duchamp* run can be read to extract each of the individual detections. The output stage is then run again, with the parameters (in particular `pixelCentre` and `spectralMethod`) as given in the input file.

Perhaps you would also like to extract a single source's spectral plot (e.g. for use in a journal paper). The use-previous method allows you to specify particular sources to re-plot. Only these sources will be plotted in the `SpectraFile` file, and individual files will be created for each of the listed sources. Their filenames will follow the format of `SpectraFile`: if, `SpectraFile=file.ps`, source #3 will appear in `file-03.ps`. To give a list of sources, use the `objectList` parameter, and provide a string with individual object numbers or object ranges: e.g. 1,2,4-7,8,11.

5 Notes and hints on the use of *Duchamp*

In using *Duchamp*, the user has to make a number of decisions about the way the program runs. This section is designed to give the user some idea about what to choose.

5.1 Memory usage

A lot of attention has been paid to the memory usage in *Duchamp*, recognising that data cubes are going to be increasing in size with new generation correlators and wider fields of view. However, users with large cubes should be aware of the likely usage for different modes of operation and plan their *Duchamp* execution carefully.

At the start of the program, memory is allocated sufficient for:

- The entire pixel array (as requested, subject to any subsection).
- The spatial extent, which holds the map of detected pixels (for output into the detection map).
- If smoothing or reconstruction has been selected, another array of the same size as the pixel array. This will hold the smoothed/reconstructed array (the original needs to be kept to do the correct parameterisation of detected sources).
- If baseline-subtraction has been selected, a further array of the same size as the pixel array. This holds the baseline values, which need to be added back in prior to parameterisation.

All of these will be float type, except for the detection map, which is short.

There will, of course, be additional allocation during the course of the program. The detection list will progressively grow, with each detection having a memory footprint as described in §3.7.1. But perhaps more important and with a larger impact will be the temporary space allocated for various algorithms.

The largest of these will be the wavelet reconstruction. This will require an additional allocation of twice the size of the array being reconstructed, one for the coefficients and one for the wavelets - each scale will overwrite the previous one. So, for the 1D case, this means an additional allocation of twice the spectral dimension (since we only reconstruct one spectrum at a time), but the 3D case will require an additional allocation of twice the cube size (this means there needs to be available at least four times the size of the input cube for 3D reconstruction, plus the additional overheads of detections and so forth).

The smoothing has less of an impact, since it only operates on the lower dimensions, but it will make an additional allocation of twice the relevant size (spectral dimension for spectral smoothing, or spatial image size for the spatial Gaussian smoothing).

The other large allocation of temporary space will be for calculating robust statistics. The median-based calculations require at least partial sorting of the data, and so cannot be done on the original image cube. This is done for the entire cube and so the temporary memory increase can be large.

5.2 Timing considerations

Another intersting question from a user's perspective is how long you can expect *Duchamp* to take. This is a difficult question to answer in general, as different users will have different sized data sets, as well as machines with different capabilities (in terms of the CPU speed and I/O & memory bandwidths). Additionally, the time required will depend slightly on the number of sources found and their size (very large sources can take a while to fully parameterise).

Having said that, in Whiting (2012) a brief analysis was done looking at different modes of execution applied to a single HIPASS cube (#201) using a MacBook Pro (2.66GHz, 8MB RAM). Two sets of thresholds were used, either 10^8 Jy beam^{-1} (no sources will be found, so that the time taken is dominated by preprocessing), or 35 mJy beam^{-1} (or $\sim 2.58\sigma$, chosen so that the time taken will include that required to process sources). The basic searches, with no pre-processing done, took less than a second for the high-threshold search, but between 1 and 3 min for the low-threshold case – the numbers of sources detected ranged from 3000 (rejecting sources with less than 3 channels and 2 spatial pixels) to 42000 (keeping all sources).

When smoothing, the raw time for the spectral smoothing was only a few seconds, with a small dependence on the width of the smoothing filter. And because the number of spurious sources is markedly decreased (the final catalogues ranged from 17 to 174 sources, depending on the width of the smoothing), searching with the low threshold did not add much more than a second to the time. The spatial smoothing was more computationally intensive, taking about 4 minutes to complete the high-threshold search.

The wavelet reconstruction time primarily depended on the dimensionality of the reconstruction, with the 1D taking 20 s, the 2D taking 30 - 40 s and the 3D taking 2 - 4 min. The spread in times for a given dimensionality was caused by different reconstruction thresholds, with lower thresholds taking longer (since more pixels are above the threshold and so need to be added to the final spectrum). In all cases the reconstruction time dominated the total time for the low-threshold search, since the number of sources found was again smaller than the basic searches.

5.3 Why do preprocessing?

The preprocessing options provided by *Duchamp*, particularly the ability to smooth or reconstruct via multi-resolution wavelet decomposition, provide an opportunity to beat the effects of the random noise that will be present in the data. This noise will ultimately limit ones ability to detect objects and form a complete and reliable catalogue. Two effects are important here. First, the noise reduces the completeness of the final catalogue by suppressing the flux of real sources such that they fall below the detection threshold. Secondly, the noise provides false positive detections through noise peaks that fall above the threshold, thereby reducing the reliability of the catalogue.

Whiting (2012) examined the effect on completeness and reliability for the reconstruction and smoothing (1D cases only) when applied to a simple simulated dataset. Both had the effect of reducing the number of spurious sources, which means the searches can be done to fainter thresholds. This led to completeness levels of about one flux unit (equal to one standard-deviation of the noise) fainter than searches without pre-processing, with $> 95\%$ reliability. The smoothing did slightly better, with the completeness level nearly half a flux unit fainter than the reconstruction, although this was helped by the sources in the simulation all having the same spectral size.

5.4 Reconstruction considerations

The *à trous* wavelet reconstruction approach is designed to remove a large amount of random noise while preserving as much structure as possible on the full range of spatial and/or spectral scales present in the data. While it is relatively more expensive in terms of memory and CPU usage (see previous sections), its effect on, in particular, the reliability of the final catalogue makes it worth investigating.

There are, however, a number of subtleties to it that need to be considered by potential users. Whiting (2012) shows a set of examples of reconstruction applied to simulated and real data. The real data, in this case a HIPASS cube, shows differences in the quality of the reconstructed spectrum depending on the dimensionality of the reconstruction. The two-dimensional reconstruction (where the cube is reconstructed one channel map at a time) shows much larger channel-to-channel noise, with a number of narrow peaks surviving the reconstruction process. The problem here is that there are spatial correlations between pixels due to the beam, which allow beam-sized noise fluctuations to rise above the threshold more frequently in one-dimension. The other effect is that when compared to a spectrum from the 1D reconstruction, each channel is independently reconstructed, and does not depend on its neighbouring channels. This is also why the 3D reconstruction (which also suffers from the beam effects) has improved noise in the output spectrum, since the information on neighbouring channels is taken into account.

Caution is also advised when looking at subsections of a cube. Due to the multi-scale nature of the algorithm, the wavelet coefficients at a given pixel are influenced by pixels at very large separations, particularly given that edges are dealt with by assuming reflection (so the whole array is visible to all pixels). Also, if one decreases the dimensions of the array being reconstructed, there may be fewer scales used in the reconstruction. These points mean that the reconstruction of a subsection of a cube will differ from the same subsection of the reconstructed cube. The difference may be small (depending on the relative size difference and the amount of structure at large scales), but there will be differences at some level.

Note also that BLANK pixels have no effect on the reconstruction: they remain as BLANK in the output, and do not contribute to the discrete convolution when they otherwise would. The use of the Milky Way channel range, however, has no effect on the reconstruction – these are applied after the preprocessing, either in the searching or the rejection stage.

5.5 Smoothing considerations

The smoothing approach differs from the wavelet reconstruction in that it has a single scale associated with it. The user has two choices to make - which dimension to smooth in (spatially or spectrally), and what size kernel to smooth with. Whiting (2012) show examples of how different smoothing widths (in one-dimension in this case) can highlight sources of different sizes. If one has some *a priori* idea of the typical size scale of objects one wishes to detect, then choosing a single smoothing scale can be quite beneficial.

Note also that beam effects can be important here too, when smoothing spatial data on scales close to that of the beam. This can enhance beam-sized noise fluctuations and potentially introduce spurious sources. As always, examining the smoothed array (after saving via `flagOutputSmooth`) is a good idea.

5.6 Threshold method

When it comes to searching, the FDR method produces more reliable results than simple sigma-clipping, particularly in the absence of reconstruction. However, it does not work in exactly the way one would expect for a given value of `alpha`. For instance, setting fairly liberal values of `alpha` (say, 0.1) will often lead to a much smaller fraction of false detections (i.e. much less than 10%). This is the effect of the merging algorithms, that combine the sources after the detection stage, and reject detections not meeting the minimum pixel or channel requirements. It is thus better to aim for larger `alpha` values than those derived from a straight conversion of the desired false detection rate.

If the FDR method is not used, caution is required when choosing the S/N cutoff. Typical cubes have very large numbers of pixels, so even an apparently large cutoff will still result in a not-insignificant number of detections simply due to random fluctuations of the noise background. For instance, a 4σ threshold on a cube of Gaussian noise of size $100 \times 100 \times 1024$ will result in ~ 340 single-pixel detections. This is where the minimum channel and pixel requirements are important in rejecting spurious detections.

6 Future developments

Here are lists of planned improvements and a wish-list of features that would be nice to include (but are not planned in the immediate future). Let me know if there are items not on these lists, or items on the list you would like prioritised.

Planned developments:

- Parallelisation of the code, to improve speed particularly on multi-core machines.
- Better determination of the noise characteristics of spectral-line cubes, including understanding how the noise is generated and developing a model for it.
- Include more source analysis. Examples could be: shape information; measurements of HI mass; more variety of measurements of velocity width and profile.
- Improved ability to reject interference, possibly on the spectral shape of features.
- Ability to separate (de-blend) distinct sources that have been merged.

Wish-list:

- Incorporation of Swinburne's S2PLOT [9] code for improved visualisation.
- Link to lists of possible counterparts (e.g. via NED/SIMBAD/other VO tools?).
- On-line web service interface, so a user can upload a cube and get back a source-list.
- Embed *Duchamp* in a GUI, to move away from the text-based interaction.

[9] http://astronomy.swin.edu.au/s2plot/

7 Acknowledgements

Thanks are due to the many people who have provided assistance and advice during the development and testing of *Duchamp*, particularly Ivy Wong, Kathrin Wolfinger, Tobias Westmeier, Mary Putman, Cormac Purcell, Attila Popping, Tara Murphy, Enno Middelberg, Korinne McDonnell, Malte Marquarding, Philip Lah, Russell Jurek, Simon Guest, Jose Francisco Gomez, BiQing For, Luca Cortese, Mark Calabretta, David Barnes and Robbie Auld. Additionally, Emil Lenc and Anita Richards both provided valuable comments on the journal paper that have helped the descriptions therein. I'd like to thank the members of the ASKAP Working Group 2 (Source Finding) for their interest and feedback - *Duchamp* has been considered as part of the development of ASKAP Survey Science Projects, and this has driven further development of the core *Duchamp* software.

The graphics in *Duchamp* are created using the PGPLOT[10] library, FITS file access is controlled through the CFITSIO[11] library (Pence 1999), while the world-coordinate transformations are performed using the WCSLIB[12] library (described in Calabretta and Greisen (2002)).

The bulk of this work was conducted as part of a CSIRO Emerging Science Postdoctoral Fellowship, and *Duchamp* continues to be maintained both as a standalone package and as part of the software development effort for the ASKAP telescope. This work was supported by the NCI National Facility at the ANU.

[10] http://www.astro.caltech.edu/~tjp/pgplot/
[11] http://heasarc.gsfc.nasa.gov/docs/software/fitsio/fitsio.html
[12] http://www.atnf.csiro.au/people/mcalabre/WCS/index.html

References

M.R. Calabretta and E.W. Greisen. "Representations of celestial coordinates in FITS". *A&A*, 395:1077–1122, December 2002. .

E.W. Greisen and M.R. Calabretta. "Representations of world coordinates in FITS". *A&A*, 395:1061–1075, December 2002.

E.W. Greisen, M.R. Calabretta, F.G. Valdes, and S.L. Allen. Representations of spectral coordinates in FITS. *A&A*, 446:747–771, 2006.

R.J. Hanisch, A. Farris, E.W. Greisen, W.D. Pence, B.M. Schlesinger, P.J. Teuben, R.W. Thompson, and A. Warnock. "Definition of the Flexible Image Transport System (FITS)". *A&A*, 376:359–380, September 2001.

A.M. Hopkins, C.J. Miller, A.J. Connolly, C. Genovese, R.C. Nichol, and L. Wasserman. "A New Source Detection Algorithm Using the False-Discovery Rate". *AJ*, 123:1086–1094, February 2002.

R.K. Lutz. "An algorithm for the real time analysis of digitised images". *The Computer Journal*, 23:262–269, 1980.

M.J. Meyer et al. "The HIPASS catalogue - I. Data presentation". *MNRAS*, 350:1195–1209, June 2004.

C.J. Miller, C. Genovese, R.C. Nichol, L. Wasserman, A. Connolly, D. Reichart, A. Hopkins, J. Schneider, and A. Moore. "Controlling the False-Discovery Rate in Astrophysical Data Analysis". *AJ*, 122:3492–3505, December 2001.

R.F. Minchin. "Finding the Bivariate Brightness Distribution of Galaxies from an HI Selected Sample". *PASA*, 16:12–17, 1999.

W. Pence. CFITSIO, v2.0: A New Full-Featured Data Interface. In D. M. Mehringer, R. L. Plante, & D. A. Roberts, editor, *Astronomical Data Analysis Software and Systems VIII*, volume 172 of *Astronomical Society of the Pacific Conference Series*, page 487, 1999.

J.-L. Starck and F. Murtagh. *"Astronomical Image and Data Analysis"*. Springer, 2002.

M. T. Whiting. DUCHAMP: a 3D source finder for spectral-line data. *MNRAS*, 421:3242–3256, 2012.

A Obtaining and installing *Duchamp*

A.1 Installing

The *Duchamp* web page can be found at the following location:
http://www.atnf.csiro.au/people/Matthew.Whiting/Duchamp
Here you can find a gzipped tar archive of the source code that can be downloaded and extracted, as well as this User's Guide in postscript and hyperlinked PDF formats.

To build *Duchamp*, you will need three main external libraries: PGPLOT, CFITSIO (this needs to be version 2.5 or greater – version 3+ is better) and WCSLIB. If these are not present on your system, you can download them from the following locations:

- PGPLOT: http://www.astro.caltech.edu/ tjp/pgplot/
- CFITSIO: http://heasarc.gsfc.nasa.gov/docs/software/fitsio/fitsio.html
- WCSLIB: http://www.atnf.csiro.au/people/Mark.Calabretta/WCS/index.html

A.1.1 Basic installation

Duchamp can be built on Unix/Linux systems by typing (assuming that the prompt your terminal provides is a `>` – don't type this character!):

```
> ./configure
> make
> make lib (optional -- to create a library for development purposes)
> make clean (optional -- to remove the object files)
> make install
```

This default setup will search in standard locations for the necessary libraries, and install the executable (`Duchamp-1.2`) in `/usr/local/bin`, along with a `Duchamp` symbolic link (a copy will also be in the current directory). The library (if you've made it) will be installed in `/usr/local/lib`, and the full set of header files will be installed in `/usr/local/include/duchamp` and subdirectories thereof. If you want these to go somewhere else, e.g. if you don't have write-access to that directory, or you need to tweak the libraries, see the next section. Otherwise, jump to the testing section.

A.1.2 Tweaking the installation process

The `configure` script allows the user to tailor the installation according to the particular requirements of their system.

To install *Duchamp* in a directory other than `/usr/local/bin`, use the `--prefix` option with configure, specifying the directory above the `bin/` directory e.g.

```
> ./configure --prefix=/home/mduchamp
```

and then run `make`, (`make lib` if you like), and `make install` as stated above. This will put the binary in the directory `/home/mduchamp/bin`. The library, if made, will be put in `/home/mduchamp/lib` and the header files will be put in `/home/mduchamp/include/duchamp` and subdirectories.

If the above-mentioned libraries have been installed in non-standard locations, or you have more than one version installed on your system, you can specify specific locations by using the `configure` flags `--with-cfitsio=<dir>`, `--with-wcslib=<dir>` or `--with-pgplot=<dir>`. For example:

```
> ./configure --with-wcslib=/home/mduchamp/wcslib-4.2
```

Duchamp can be compiled without PGPLOT if it is not installed on your system – the searching and text-based output remains the same, but you will not have any graphical output. To manually specify this option, you can either give `--without-pgplot` or `--with-pgplot=no` as arguments to `configure`:

```
> ./configure --without-pgplot
```

(Note that CFITSIO and WCSLIB are essential, however, so flags such as `--without-wcslib` or `--without-cfitsio` will not work.). Even if you do not give the `--without-pgplot` option, and the PGPLOT library is not found, Duchamp will still compile (albeit without graphical capabilities).

An additional option that is useful is the ability to specify which compiler to use. This is very important for the Fortran compiler (used for linking due to the use of PGPLOT), particularly on Mac OS X, where `gfortran` is often used instead of `gcc`. To specify a particular Fortran compiler, use the `F77` flag:

```
> ./configure F77=gfortran
```

Of course, all desired flags should be combined in one `configure` call. For a full list of the options with `configure`, run:

```
> ./configure --help
```

Once `configure` has run correctly, simply run `make` and `make install` to build *Duchamp* and put it in the correct place (either `/usr/local/bin` or the location given by the `--prefix` option discussed above).

A.1.3 Making sure it all works

Running make will create the executable `Duchamp-1.2`. You can verify that it is running correctly by running the verification shell script:

```
> ./VerifyDuchamp.sh
```

This will use a dummy FITS image in the `verification/` directory – this image has some Gaussian random noise, with five Gaussian sources present, plus a dummy WCS. The script runs Duchamp on this image with four different sets of inputs, and compares to known results, looking for differences and reporting any. There should be none reported if everything is working correctly. You can then install *Duchamp* on your system via:

```
> make install
```

A.2 Running *Duchamp*

You can then run *Duchamp* on your own data. This can be done in one of two ways. The first is:

```
> Duchamp -f [FITS file]
```

where [FITS file] is the file you wish to search. This method simply uses the default values of all parameters.

The second method allows some determination of the parameter values by the user. Type:

```
> Duchamp -p [parameter file]
```

where [parameterFile] is a file with the input parameters, including the name of the cube you want to search. There are two example input files included with the distribution. The smaller one, InputExample, shows the typical parameters one might want to set. The large one, InputComplete, lists all possible parameters that can be entered, and a brief description of them. To get going quickly, just replace the "your-file-here" in the InputExample file with your image name, and type

```
> Duchamp -p InputExample
```

To disable the use of X-window plotting (in displaying the map of detections), one can either set the parameter flagXOutput = false or use the -x command-line option:

```
> Duchamp -x -p [parameter file] , or
> Duchamp -x -f [FITS file]
```

The following appendices provide details on the individual parameters, and show examples of the output files that *Duchamp* produces.

A.3 Feedback

It may happen that you discover bugs or problems with *Duchamp*, or you have suggestions for improvements or additional features to be included in future releases. You can submit a “ticket” (a trackable bug report) at the *Duchamp* Trac wiki at the following location:
http://svn.atnf.csiro.au/trac/duchamp/newticket
(there is a link to this page from the Duchamp website).

There is also an email exploder, duchamp-user[**at**]atnf.csiro.au, that users can subscribe to keep up to date with changes, updates, and other news about *Duchamp*. To subscribe, send an email (from the account you wish to subscribe to the list) to duchamp-user-request[**at**]atnf.csiro.au with the single word “subscribe” in the body of the message. To be removed from this list, send a message with “unsubscribe” in its body to the same address.

A.4 Beta Versions

On the *Duchamp* website there may be a beta version listed in the downloads section. As *Duchamp* is still under development, there will be times when there has been new functionality added to the code, but the time has not yet come to release a new minor (or indeed major) version.

Sometimes I will post the updated version of the code on the website as a “beta” version, particularly if I’m interested in people testing it. It will not have been tested as rigorously as the proper releases, but it will certainly work in the basic cases that I use to test it during development. So feel free to give it a try – the `CHANGES` file will usually detail what is different to the last numbered release.

B Available parameters

The full list of parameters that can be listed in the input file are given here. If not listed, they take the default value given in parentheses. Since the order of the parameters in the input file does not matter, they are grouped here in logical sections.

Input related

`ImageFile [no default]`:
The filename of the data cube to be analysed.

`flagSubsection [false]`:
A flag to indicate whether one wants a subsection of the requested image.

`Subsection [ [*,*,*] ]`:
The requested subsection – see §3.1 for details on the subsection format. If the full range of a dimension is required, use a `*` (thus the default is the full cube).

`flagReconExists [false]`:
A flag to indicate whether the reconstructed array has been saved by a previous run of *Duchamp*. If set true, the reconstructed array will be read from the file given by `reconFile`, rather than calculated directly.

`reconFile [no default]`:
The FITS file that contains the reconstructed array. If `flagReconExists` is true and this parameter is not defined, the default file that is looked for will be determined by the *à trous* parameters (see §3.4).

`flagSmoothExists [false]`:
A flag to indicate whether the Hanning-smoothed array has been saved by a previous run of *Duchamp*. If set true, the smoothed array will be read from the file given by `smoothFile`, rather than calculated directly.

`smoothFile [no default]`:
The FITS file that has a previously smoothed array. If `flagSmoothExists` is true and this parameter is not defined, the default file that is looked for will be determined by the smoothing parameters (see §3.5).

`usePrevious [false]`:
A flag to indicate that *Duchamp* should read the list of objects from a previously-created log file, rather than doing the searching itself. The set of outputs will be created according to the flags in the following section.

`objectList [no default]`:
When `usePrevious=true`, this list is used to output individual spectral plots, as well as a postscript file for all spectral plots as given by `SpectraFile`. The filenames of the plots will be the same as `SpectraFile`, but with

-XX at the end, where XX is the object number (e.g. `duchamp-Spectra-07.ps`). The format of the parameter value should be a string listing individual objects or object ranges: e.g. 1,2,4-7,8,14.

Output related

`OutFile [duchamp-Results.txt]`: The file containing the final list of detections. This also records the list of input parameters.

`flagSeparateHeader [false]`: A flag to indicate that the header information that would normally be printed at the start of the results file (containing information on the parameters, image statistics and number of detections) should instead be written to a separate file.

`HeaderFile [duchamp-Results.hdr]`: The file to which the header information should be written when `flagSeparateHeader=true`.

`flagPlotSpectra [true]`: Whether to produce a postscript file containing spectra of all detected objects. If PGPlot has not been enabled, this parameter defaults to `false`.

`SpectraFile [duchamp-Spectra.ps]`: The postscript file that contains the resulting integrated spectra and images of the detections.

`flagTextSpectra [false]`: A flag to say whether the spectra should be saved in text form in a single file. See below for a description.

`spectraTextFile [duchamp-Spectra.txt]`: The file containing the spectra of each object in ascii format. This file will have a column showing the spectral coordinates, and one column for each of the detected objects, showing the flux values as plotted in the graphical output of `spectraFile`.

`flagLog [false]`: A flag to indicate whether the details of intermediate detections should be logged.

`LogFile [duchamp-Logfile.txt]`: The file in which intermediate detections and the pixel content of the final list of detections are logged. These are detections that have not been merged. This is primarily for use in debugging and diagnostic purposes: normal use of the program will probably not require it.

`flagOutputMomentMap [false]`: A flag to say whether or not to save a FITS file containing the moment-0 map.

`fileOutputMomentMap [see text]`:
The file to which the moment-0 array is written. If left blank (the default), the naming scheme detailed in §4.4.2 is used.

`flagOutputMask [false]`:
A flag to say whether or not to save a FITS file containing a mask array, with values 1 where there is a detected object and 0 elsewhere.

`fileOutputMask [see text]`:
The file to which the mask array is written. If left blank (the default), the naming scheme detailed in §4.4.2 is used.

`flagMaskWithObjectNum [false]`:
If this flag is true, the detected pixels in the mask image have the corresponding object ID as their value. If false, they have the value 1. All non-detected pixels have the value 0.

`flagOutputRecon [false]`:
A flag to say whether or not to save the reconstructed cube as a FITS file.

`fileOutputRecon [see text]`:
The file to which the reconstructed array is written. If left blank (the default), the naming scheme detailed in §3.6 is used.

`flagOutputResid [false]`:
As for `flagOutputRecon`, but for the residual array – the difference between the original cube and the reconstructed cube.

`fileOutputResid [see text]`:
The file to which the residual array is written. If left blank (the default), the naming scheme detailed in §3.6 is used.

`flagOutputSmooth [false]`:
A flag to say whether or not to save the smoothed cube as a FITS file.

`fileOutputSmooth [see text]`:
The file to which the smoothed array is written. If left blank (the default), the naming scheme detailed in §3.6 is used.

`flagVOT [false]`: A flag to say whether to create a VOTable file with the detection information. This will be an XML file in the Virtual Observatory VOTable format.

`votFile [duchamp-Results.xml]`:
The VOTable file with the list of final detections. Some input parameters are also recorded.

`flagKarma [false]`:
A flag to say whether to create a Karma annotation file corresponding to the information in `outfile`. This can be used as an overlay in Karma programs such as `kvis`.

`karmaFile [duchamp-Results.ann]:` The Karma annotation file showing the list of final detections.

`annotationType [borders]:` Which type of annotation plot to use. Specifying “borders” gives an outline around the detected spatial pixels, while “circles” gives a circle centred on the centre of the object with radius large enough to encompass all spatial pixels.

`flagMaps [true]:` A flag to say whether to save postscript files showing the 0th moment map of the whole cube (`momentMap`) and the detection image (`detectionMap`). If PGPlot has not been enabled, this parameter defaults to `false`.

`momentMap [duchamp-MomentMap.ps]:` A postscript file containing a map of the 0th moment of the detected sources, as well as pixel and WCS coordinates.

`detectionMap [duchamp-DetectionMap.ps]:` A postscript file with a map showing each of the detected objects, coloured in greyscale by the number of detected channels in each spatial pixel. Also shows pixel and WCS coordinates.

`flagXOutput [true]:` A flag to say whether to display a 0th moment map in a PGPlot X-window. This will be in addition to any that are saved to a file. This parameter can be overridden by the use of the `-x` command-line option, which disables the X-windows output. If PGPlot has not been enabled, this parameter defaults to `false`.

`newFluxUnits [no default]:` Flux units that the pixel values should be converted into. These should be directly compatible with the units in the FITS header, given by the BUNIT keyword.

`precFlux [3]:` The desired precision (i.e. number of decimal places) for flux values given in the output files and tables.

`precVel [3]:` The desired precision (i.e. number of decimal places) for velocity/frequency values given in the output files and tables.

`precSNR [2]:` The desired precision (i.e. number of decimal places) for the peak SNR value given in the output files and tables.

Modifying the cube

`flagTrim [false]:` A flag to say whether to trim BLANK pixels from the edges of the cube – these are typically pixels set to some particular value because they fall outside the imaged area, and trimming them can help speed up the execution.

`flagMW [false]:` A flag to say whether to ignore channels contaminated by

Milky Way (or other) emission – the searching algorithms will not look at these channels.

`maxMW [112]:` The maximum channel number that contains “Milky Way” emission. This is the channel number in the original cube, before any subsection is applied.

`minMW [75]:` The minimum channel number that contains “Milky Way” emission. This is the channel number in the original cube, before any subsection is applied. Note that the range specified by `maxMW` and `minMW` is inclusive.

`flagBaseline [false]:`
A flag to say whether to remove the baseline from each spectrum in the cube for the purposes of reconstruction and detection.

Detection related

General detection

`searchType [spatial]:`
How the searches are done. Only “spatial” and “spectral” are accepted. A value of “spatial” means each 2D channel map is searched, whereas “spectral” means each 1D spectrum is searched.

`flagStatSec [false]:`
A flag indicating whether the statistics should be calculated on a subsection of the cube, rather than the full cube. Note that this only applies to the statistics used to determine the threshold, and not for other statistical calculations (such as those in the reconstruction phase).

`StatSec [ [*,*,*] ]:`
The subsection of the cube used for calculating statistics – see §3.1 for details on the subsection format. Only used if `flagStatSec=true`.

`flagRobustStats [true]:`
A flag indicating whether to use the robust statistics (median and MADFM) to estimate the noise parameters of the cube, rather than the mean and rms. See §3.7.3 for details.

`flagNegative [false]:`
A flag indicating that the features of interest are negative. The cube is inverted prior to searching.

`snrCut [5.]:` The threshold, in multiples of σ above the mean.

`threshold [no default]:`
The actual value of the threshold. Normally the threshold is calculated from the cube’s statistics, but the user can manually specify a value to be used that overrides the calculated value. If this is not specified, the calculated value is used, but this value will take precedence over other

means of calculating the threshold (i.e. via `snrCut` or the FDR method).

`flagGrowth [false]`:
A flag indicating whether or not to grow the detected objects to a smaller threshold.

`growthCut [3.]`: The smaller threshold using in growing detections. In units of σ above the mean.

`growthThreshold [no default]`:
Alternatively, the threshold to which detections are grown can be specified in flux units (in the same manner as the `threshold` parameter). When the `threshold` parameter is given, this option **must** be used instead of `growthCut`.

`beamFWHM [0.]`: The full-width at half maximum of the beam, in pixels. If the header keywords BMAJ and BMIN are present, then these will be used to calculate the beam area, and this parameter will be ignored. This will take precedence over `beamArea` (but is ignored if not specified).

`beamArea [0.]`: The **area** of the beam in pixels (i.e. how many pixel does the beam cover?). As above, if the header keywords BMAJ and BMIN are present, then these will be used to calculate the beam area, and this parameter will be ignored.

À trous reconstruction

`flagATrous [false]`:
A flag indicating whether or not to reconstruct the cube using the *à trous* wavelet reconstruction. See §3.4 for details.

`reconDim [1]`: The number of dimensions to use in the reconstruction. 1 means reconstruct each spectrum separately, 2 means each channel map is done separately, and 3 means do the whole cube in one go.

`scaleMin [1]`: The minimum wavelet scale to be used in the reconstruction. A value of 1 means "use all scales".

`scaleMax [0]`: The maximum wavelet scale to be used in the reconstruction. If the value is ≤ 0 then the maximum scale is calculated from the size of the input array. Similarly, if the value given is larger than this calculated value, the calculated value is used instead.

`snrRecon [4]`: The thresholding cutoff used in the reconstruction – only wavelet coefficients at least this many σ above the mean are included in the reconstruction.

`reconConvergence [0.005]`:
The convergence criterion used in the reconstruction. The *à trous* algorithm iterates until the relative change in the standard deviation of the residuals is less than this amount.

`filterCode [1]`: The code number of the filter to use in the reconstruction. The options are:

- **•1:** B_3-spline filter: coefficients = $(\frac{1}{16}, \frac{1}{4}, \frac{3}{8}, \frac{1}{4}, \frac{1}{16})$
- **•2:** Triangle filter: coefficients = $(\frac{1}{4}, \frac{1}{2}, \frac{1}{4})$
- **•3:** Haar wavelet: coefficients = $(0, \frac{1}{2}, \frac{1}{2})$

Smoothing the cube

`flagSmooth [false]`: A flag indicating whether to smooth the cube. See §3.5 for details.

`smoothType [spectral]`: The smoothing method used: either "spectral" (with a 1D Hanning filter) or "spatial" (with a 2D Gaussian filter).

`hanningWidth [5]`: The width of the Hanning smoothing kernel.

`kernMaj [3]`: The full-width-half-maximum (FWHM) of the 2D Gaussian smoothing kernel's major axis.

`kernMin [3]`: The FWHM of the 2D Gaussian smoothing kernel's minor axis.

`kernPA [0]`: The position angle, in degrees, anticlockwise from vertical (i.e. usually East of North).

FDR method

`flagFDR [false]`: A flag indicating whether or not to use the False Discovery Rate method in thresholding the pixels.

`alphaFDR [0.01]`: The α parameter used in the FDR analysis. The average number of false detections, as a fraction of the total number, will be less than α (see §3.7).

`FDRnumCorChan [2]`: The number of neighbouring spectral channels that are assumed to be correlated. This is needed by the FDR algorithm to calculate the normalisation constant c_N (see §3.7).

Merging detections

`minPix [2]`: The minimum number of spatial pixels for a single detection to be counted.

`minChannels [3]`: At least one contiguous set of this many channels must be present in the detection for it to be accepted.

`minVoxels [minPix + minChannels - 1]`: The minimum size of the object, in terms of the total number of voxels, for it to be accepted. This will be *at least* minPix+minChannels−1, but can be set higher.

`flagRejectBeforeMerge [false]:`
A flag indicating whether to reject sources that fail to meet the `minPix` or `minChannels` criteria **before** the merging stage. Default behaviour is to do the rejection last.

`flagTwoStageMerging [true]:`
A flag indicating whether to do an initial merge of newly-detected sources into the source list as they are found. If `false`, new sources are simply added to the end of the list for later merging.

`flagAdjacent [true]:`
A flag indicating whether to use the "adjacent pixel" criterion to decide whether to merge objects. If not, the next two parameters are used to determine whether objects are within the necessary thresholds.

`threshSpatial [3.]:`
The maximum allowed minimum spatial separation (in pixels) between two detections for them to be merged into one. Only used if `flagAdjacent = false`.

`threshVelocity [7.]:`
The maximum allowed minimum channel separation between two detections for them to be merged into one.

WCS parameters

`spectralType []:` The user can specify an alternative WCS spectral type that the spectral axis can be expressed in. This specification should conform to the standards described in Greisen et al. (2006), although it is possible to provide just the first four letters (the "S-type", e.g. 'VELO').

`restFrequency [-1]:`
If provided, this will be used in preference to the rest frequency given in the FITS header to calculate velocities and related quantities. A negative value (such as the default) will mean this is not used and the FITS header value, if present, is used instead.

`spectralUnits []:` The user can specify the units of the spectral axis, overriding those given in the FITS header. If the spectral type is being changed, these units should be appropriate for that quantity. If not provided, the FITS header information is used.

Other parameters

`spectralMethod [peak]:`
This indicates which method is used to plot the output spectra: `peak` means plot the spectrum containing the detection's peak pixel; `sum` means sum the spectra of each detected spatial pixel, and correct for the beam size. Any other choice defaults to `peak`.

`pixelCentre [centroid]`:
Which of the three ways of expressing the "centre" of a detection (see §4.2.1 for a description of the options) to use for the X, Y, & Z columns in the output list. Alternatives are: `centroid, peak, average`.

`sortingParam [vel]`:
The parameter on which to sort the output list of detected objects. Options are: xvalue, yvalue, zvalue, ra, dec, vel, w50, iflux, pflux (integrated and peak flux respectively), or snr. If the parameter begins with a '-' (e.g. '-vel'), the order of the sort is reversed.

`drawBorders [true]`:
A flag indicating whether to draw borders around the detected objects in the moment maps included in the output (see for example Fig. 2).

`drawBlankEdges [true]`:
A flag indicating whether to draw the dividing line between BLANK and non-BLANK pixels on the 2D images (see for example Fig. 3).

`verbose [true]`: A flag indicating whether to print the progress of any computationally intensive algorithms (e.g. reconstruction, searching or merging algorithms) to the screen.

C Example parameter files

This is what a typical parameter file would look like.

```
imageFile       /home/mduchamp/fountain.fits
logFile         logfile.txt
outFile         results.txt
spectraFile     spectra.ps
flagSubsection  false
flagOutputRecon false
flagOutputResid 0
flagTrim        1
flagMW          1
minMW           75
maxMW           112
flagGrowth      1
growthCut       1.5
flagATrous      1
reconDim        1
scaleMin        1
snrRecon        4
flagFDR         1
alphaFDR        0.1
snrCut          3
threshSpatial   3
threshVelocity  7
```

Note that, as in this example, the flag parameters can be entered as strings (`true`/`false`) or integers (`1`/`0`). Also, note that it is not necessary to include all these parameters in the file, only those that need to be changed from the defaults (as listed in Appendix B), which in this case would be very few. A minimal parameter file might look like:

```
imageFile       /home/mduchamp/fountain.fits
flagLog         false
flagATrous      1
snrRecon        3
snrCut          2.5
minChannels     4
```

This will reconstruct the cube with a lower SNR value than the default, select objects at a lower threshold, with a looser minimum channel requirement, and not keep a log of the intermediate detections.

The following page demonstrates how the parameters are presented to the user, both on the screen at execution time, and in the output and log files. On each line, there is a description on the parameter, the relevant parameter name that is used in the input file (if there is one that the user can enter), and the value of the parameter being used.

```
---- Parameters ----
Image to be analysed.........................[imageFile]  =  fountain.fits
Intermediate Logfile...........................[logFile]  =  duchamp-Logfile.txt
Final Results file.............................[outFile]  =  duchamp-Results.txt
Header for results file.....................[headerFile]  =  duchamp-Results.hdr
Spectrum file..............................[spectraFile]  =  duchamp-Spectra.ps
Text file with ascii spectral data.....[spectraTextFile]  =  duchamp-Spectra.txt
VOTable file...................................[votFile]  =  duchamp-Results.xml
Karma annotation file........................[karmaFile]  =  duchamp-Results.ann
0th Moment Map...............................[momentMap]  =  duchamp-MomentMap.ps
Detection Map.............................[detectionMap]  =  duchamp-DetectionMap.ps
Display a map in a pgplot xwindow?.........[flagXOutput]  =  true
Saving reconstructed cube?.............[flagOutputRecon]  =  false
Saving residuals from reconstruction?..[flagOutputResid]  =  false
Saving mask cube?.......................[flagOutputMask]  =  false
Saving 0th moment to FITS file?.........[flagOutputMask]  =  false
------
Type of searching performed.................[searchType]  =  spectral
Blank Pixel Value.......................................  =  -8.00061
Trimming Blank Pixels?........................[flagTrim]  =  false
Searching for Negative features?..........[flagNegative]  =  false
Removing Milky Way channels?....................[flagMW]  =  true
Milky Way Channels.......................[minMW - maxMW]  =  75-113
Area of Beam (pixels)...................................  =  14.6848   (beam: 3.6 x 3.6 pixels)
Removing baselines before search?.........[flagBaseline]  =  false
Smoothing data prior to searching?..........[flagSmooth]  =  false
Using A Trous reconstruction?...............[flagATrous]  =  true
Number of dimensions in reconstruction........[reconDim]  =  1
Minimum scale in reconstruction...............[scaleMin]  =  1
SNR Threshold within reconstruction...........[snrRecon]  =  3
Filter being used for reconstruction........[filterCode]  =  1 (B3 spline function)
Using Robust statistics?...............[flagRobustStats]  =  true
Using FDR analysis?............................[flagFDR]  =  false
SNR Threshold (in sigma)........................[snrCut]  =  3
Minimum # Pixels in a detection.................[minPix]  =  5
Minimum # Channels in a detection..........[minChannels]  =  3
Minimum # Voxels in a detection..............[minVoxels]  =  7
Growing objects after detection?............[flagGrowth]  =  false
Using Adjacent-pixel criterion?...........[flagAdjacent]  =  true
Max. velocity separation for merging....[threshVelocity]  =  7
Reject objects before merging?...[flagRejectBeforeMerge]  =  false
Merge objects in two stages?.......[flagTwoStageMerging]  =  false
Method of spectral plotting.............[spectralMethod]  =  sum
Type of object centre used in results......[pixelCentre]  =  centroid
--------------------
```

D Example results file

This the typical content of an output file, after running *Duchamp* with the parameters illustrated on the previous page. The table is wide, and has been split into four sections to make it easier to read.

```
-------------------------------------------------------------------
Obj#           Name     X     Y     Z         RA       DEC     VELO
                                                                [km/s]
-------------------------------------------------------------------
   1 J060946-215932  58.0 139.9 114.6 06:09:46.7 -21:59:32  225.640
   2 J060810-215122  63.5 142.0 114.9 06:08:10.6 -21:51:22  229.926
   3 J054547-214320 141.4 143.2 115.0 05:45:47.6 -21:43:20  230.210
   4 J060223-254806  83.7  83.0 118.0 06:02:23.4 -25:48:06  270.850
   5 J060539-271839  72.8  60.3 121.2 06:05:39.8 -27:18:39  312.265
   6 J061118-213644  52.6 145.5 162.5 06:11:18.6 -21:36:44  859.117
   7 J060034-285900  89.7  35.3 202.2 06:00:34.7 -28:59:00 1383.549
   8 J061707-272422  34.7  58.2 228.5 06:17:07.5 -27:24:22 1733.137
   9 J055847-252513  95.8  88.7 232.3 05:58:47.8 -25:25:13 1783.651
  10 J060054-214225  88.8 144.3 232.9 06:00:54.2 -21:42:25 1790.703
  11 J060444-260640  75.8  78.3 233.2 06:04:44.0 -26:06:40 1794.974
  12 J060107-233958  88.0 115.0 235.5 06:01:07.5 -23:39:58 1826.107
  13 J061210-214904  49.6 142.3 269.9 06:12:10.0 -21:49:04 2282.101
  14 J055508-295702 107.3  20.6 367.6 05:55:08.0 -29:57:02 3583.671
```

```
-----------------------------------------------------------------------
    w_RA    w_DEC    w_50    w_20  w_VELO     F_int     F_tot    F_peak
[arcmin] [arcmin]  [km/s]  [km/s]  [km/s] [Jy km/s] [Jy/beam] [Jy/beam]
-----------------------------------------------------------------------
   20.32    23.76  25.442  41.038  26.403     4.621     5.140     0.213
    8.12    11.94  31.285  43.053  26.403     1.102     1.225     0.095
    7.70    12.28  33.757  47.820  26.403     0.882     0.981     0.090
   12.01    11.99  29.615  41.470  26.406     1.115     1.241     0.118
   20.15    19.87  27.930  44.830  39.616     3.097     3.445     0.150
   24.32    19.63  88.130 111.537 119.061    28.601    31.748     0.410
   19.92    24.09 176.654 194.569 198.784    15.042    16.668     0.173
   16.51    15.53 307.884 327.819 305.122     4.656     5.154     0.093
   23.91    16.14 230.961 277.579 238.828     7.299     8.077     0.115
   27.95    24.13  97.257 227.378 212.301    19.527    21.610     0.166
   20.11    19.90 213.520 230.422 225.584    15.001    16.601     0.155
   27.96    28.07 201.038 251.618 278.700    54.209    59.982     0.297
   16.27    15.73 406.006 440.079 412.062     9.820    10.849     0.101
    7.76    16.14  31.769  49.223  40.044     1.770     1.947     0.161
```

```
-----------------------------------------------------------
S/Nmax  X1  X2  Y1  Y2  Z1  Z2  Npix Flag  X_av  Y_av  Z_av
                               [pix]
-----------------------------------------------------------
 16.41  56  60 137 142 114 116    48    M  57.9 139.8 115.0
  7.31  63  64 141 143 114 116    18    M  63.5 142.0 115.0
  6.93 141 142 142 144 114 116    15    M 141.4 143.2 115.0
  9.10  83  85  82  84 117 119    18    -  83.7  83.0 118.0
 11.56  71  75  58  62 120 123    42    -  72.8  60.3 121.1
 31.70  50  55 143 147 158 167   219    E  52.5 145.3 162.5
 13.35  88  92  33  38 195 210   224    -  89.7  35.3 202.2
  7.17  33  36  57  60 215 238    92    -  34.7  58.2 228.6
  8.91  93  98  87  90 221 239   136    -  95.8  88.6 231.9
 12.78  86  92 142 147 223 239   290    E  88.8 144.3 232.3
 11.93  74  78  76  80 224 241   244    -  75.8  78.3 233.1
 22.92  85  91 112 118 225 246   551    -  88.0 115.0 235.8
  7.78  48  51 141 144 256 287   195    -  49.6 142.3 270.3
 12.40 107 108  19  22 366 369    22    - 107.3  20.7 367.6
```

```
------------------------------------------
 X_cent Y_cent Z_cent X_peak Y_peak Z_peak

------------------------------------------
   58.0  139.9  114.6     59    140    114
   63.5  142.0  114.9     63    141    115
  141.4  143.2  115.0    141    144    115
   83.7   83.0  118.0     84     83    118
   72.8   60.3  121.2     72     61    121
   52.6  145.5  162.5     53    146    164
   89.7   35.3  202.2     90     36    197
   34.7   58.2  228.5     35     58    236
   95.8   88.7  232.3     96     89    237
   88.8  144.3  232.9     89    144    233
   75.8   78.3  233.2     76     78    240
   88.0  115.0  235.5     88    115    231
   49.6  142.3  269.9     50    142    259
  107.3   20.6  367.6    107     21    367
```

A good trick for those using UNIX/Linux is to make use of the `a2ps` command. The following works well:

```
a2ps -1 -r -f5 -o duchamp-Results.ps duchamp-Results.txt
```

and produces a postscript file `duchamp-Results.ps`.

The table is preceded by information on the parameters used (see the previous section), the statistics and threshold values, and the number of detections. If `flagSeparateHeader` is set to `true`, this information is put in a separate file. This information looks like the following:

```
Results of the Duchamp source finder: Thu May  3 12:15:44 2007

---- Parameters ----
      (... omitted for clarity -- see previous page for examples...)
--------------------

--------------------
Summary of statistics:
Detection threshold = 0.0389443 Jy/beam
Noise level = 0.000122074, Noise spread = 0.0129408
Full stats:
Mean   = 0.000421491    Std.Dev. = 0.0129756
Median = 0.000122074    MADFM    = 0.0087284 (= 0.0129408 as std.dev.)
--------------------
Total number of detections = 14
--------------------
```

E Example VOTable output

This is part of the VOTable, in XML format, corresponding to a search with wavelet reconstruction.

```
<?xml version="1.0"?>
<VOTABLE version="1.1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="http://www.ivoa.net/xml/VOTable/VOTable/v1.1">
  <COOSYS ID="J2000" equinox="J2000" system="eq_FK5"/>
  <RESOURCE name="Duchamp Output">
    <TABLE name="Detections">
      <DESCRIPTION>Detected sources and parameters from running the Duchamp source finder.</DESCRIPTION>
      <PARAM name="imageFile" ucd="meta.file;meta.fits" datatype="char" arraysize="63" value="/home/mduchamp/fountain.fits"/>
      <PARAM name="flagSubsection" ucd="meta.code" datatype="boolean" value="0"/>
      <PARAM name="flagStatSec" ucd="meta.code" datatype="boolean" value="0"/>
      <PARAM name="searchType" ucd="meta.note" datatype="char" arraysize="8" value="spectral"/>
      <PARAM name="flagNegative" ucd="meta.code" datatype="boolean" value="0"/>
      <PARAM name="flagBaseline" ucd="meta.code" datatype="boolean" value="0"/>
      <PARAM name="flagRobustStats" ucd="meta.code" datatype="boolean" value="1"/>
      <PARAM name="flagFDR" ucd="meta.code" datatype="boolean" value="0"/>
      <PARAM name="snrCut" ucd="stat.snr;phot;stat.min" datatype="float" value="3"/>
      <PARAM name="flagGrowth" ucd="meta.code" datatype="boolean" value="0"/>
      <PARAM name="minVoxels" ucd="" datatype="int" value="7"/>
      <PARAM name="minPix" ucd="" datatype="int" value="5"/>
      <PARAM name="minChannels" ucd="" datatype="int" value="3"/>
      <PARAM name="flagAdjacent" ucd="meta.code" datatype="boolean" value="1"/>
      <PARAM name="threshVelocity" ucd="" datatype="float" value="3"/>
      <PARAM name="flagRejectBeforeMerge" ucd="" datatype="boolean" value="0"/>
      <PARAM name="flagTwoStageMerging" ucd="" datatype="boolean" value="0"/>
      <PARAM name="pixelCentre" ucd="" datatype="char" arraysize="8" value="centroid"/>
      <PARAM name="flagSmooth" ucd="meta.code" datatype="boolean" value="0"/>
      <PARAM name="flagATrous" ucd="meta.code" datatype="boolean" value="1"/>
      <PARAM name="reconDim" ucd="" datatype="int" value="1"/>
      <PARAM name="scaleMin" ucd="" datatype="int" value="1"/>
      <PARAM name="snrRecon" ucd="" datatype="float" value="3"/>
      <PARAM name="filterCode" ucd="" datatype="int" value="1"/>
      <FIELD name="Obj#" ID="col01" ucd="meta.id" datatype="int" units="" width="5"/>
      <FIELD name="Name" ID="col02" ucd="meta.id;meta.main" datatype="char" units="" arraysize="15"/>
      <FIELD name="RA" ID="col03" ucd="pos.eq.ra;meta.main" ref="J2000" datatype="float" units="deg" width="11" precision="6"/>
      <FIELD name="DEC" ID="col04" ucd="phys.angSize;pos.eq.ra" ref="J2000" datatype="float" units="deg" width="11" precision="6"/>
      <FIELD name="VELO" ID="col05" ucd="phys.veloc;src.dopplerVeloc" datatype="float" units="km/s" width="9" precision="3"/>
      <FIELD name="w_RA" ID="col06" ucd="pos.eq.dec;meta.main" ref="J2000" datatype="float" units="arcmin" width="9" precision="2"/>
      <FIELD name="w_DEC" ID="col07" ucd="phys.angSize;pos.eq.dec" ref="J2000" datatype="float" units="arcmin" width="9" precision="2"/>
      <FIELD name="w_50" ID="col08" ucd="phys.veloc;src.dopplerVeloc;spect.line.width" datatype="float" units="km/s" width="8" precision="3"/>
      <FIELD name="w_20" ID="col09" ucd="phys.veloc;src.dopplerVeloc;spect.line.width" datatype="float" units="km/s" width="8" precision="3"/>
      <FIELD name="w_VELO" ID="col10" ucd="phys.veloc;src.dopplerVeloc;spect.line.width" datatype="float" units="km/s" width="8" precision="3"/>
      <FIELD name="Integrated_Flux" ID="col11" ucd="phot.flux;spect.line.intensity" datatype="float" units="Jy.km/s" width="10" precision="3"/>
      <FIELD name="Peak_Flux" ID="col12" ucd="phot.flux;spect.line.intensity" datatype="float" units="Jy/beam" width="10" precision="3"/>
      <FIELD name="S/Nmax" ID="col13" ucd="phot.flux;stat.snr" datatype="float" units="" width="7" precision="2"/>
      <FIELD name="Flag" ID="col14" ucd="meta.code.qual" datatype="char" units="" arraysize="5"/>
      <FIELD name="X_av" ID="col15" ucd="pos.cartesian.x" datatype="float" units="" width="6" precision="1"/>
      <FIELD name="Y_av" ID="col16" ucd="pos.cartesian.y" datatype="float" units="" width="6" precision="1"/>
      <FIELD name="Z_av" ID="col17" ucd="pos.cartesian.z" datatype="float" units="" width="6" precision="1"/>
      <FIELD name="X_Centroid" ID="col18" ucd="pos.cartesian.x" datatype="float" units="" width="7" precision="1"/>
      <FIELD name="Y_Centroid" ID="col19" ucd="pos.cartesian.y" datatype="float" units="" width="7" precision="1"/>
      <FIELD name="Z_Centroid" ID="col20" ucd="pos.cartesian.z" datatype="float" units="" width="7" precision="1"/>
      <FIELD name="X_peak" ID="col21" ucd="pos.cartesian.x" datatype="int" units="" width="7"/>
      <FIELD name="Y_peak" ID="col22" ucd="pos.cartesian.y" datatype="int" units="" width="7"/>
      <FIELD name="Z_peak" ID="col23" ucd="pos.cartesian.z" datatype="int" units="" width="7"/>
      <DATA>
        <TABLEDATA>
        <TR>
          <TD>    1</TD><TD> J060946-215932</TD><TD>  92.444503</TD><TD> -21.992234</TD><TD>  225.640</TD><TD>    20.32</TD>
<TD>    23.76</TD><TD>  25.442</TD><TD>  41.038</TD><TD>  26.403</TD><TD>     4.621</TD><TD>     0.213</TD><TD>  16.41</TD>
<TD>    M</TD><TD>  57.9</TD><TD> 139.8</TD><TD> 115.0</TD><TD>   58.0</TD><TD>  139.9</TD><TD>  114.6</TD><TD>     59</TD>
<TD>    140</TD><TD>    114</TD>
        </TR>
... truncated ...
        </TABLEDATA>
      </DATA>
    </TABLE>
  </RESOURCE>
</VOTABLE>
```

F Example Karma Annotation file output

This is the format of the Karma Annotation file, showing the locations of the detected objects. This can be loaded by the plotting tools of the Karma package (for instance, kvis) as an overlay on the FITS file.

```
# Duchamp Source Finder results for FITS file:
#  /home/mduchamp/fountain.fits
# Threshold = 4
# No ATrous reconstruction done.
#
COLOR RED
COORD W
CIRCLE 92.3376 -21.9475 0.403992
TEXT 92.3376 -21.9475 1
CIRCLE 91.9676 -26.0193 0.37034
TEXT 91.9676 -26.0193 2
CIRCLE 91.5621 -27.3459 0.437109
TEXT 91.5621 -27.3459 3
CIRCLE 92.8285 -21.6344 0.269914
TEXT 92.8285 -21.6344 4
CIRCLE 90.1381 -28.9838 0.234179
TEXT 90.1381 -28.9838 5
CIRCLE 89.72 -26.6513 0.132743
TEXT 89.72 -26.6513 6
CIRCLE 94.2743 -27.4003 0.195175
TEXT 94.2743 -27.4003 7
CIRCLE 92.2739 -21.6941 0.134538
TEXT 92.2739 -21.6941 8
CIRCLE 89.7133 -25.4259 0.232252
TEXT 89.7133 -25.4259 9
CIRCLE 90.2206 -21.6993 0.266247
TEXT 90.2206 -21.6993 10
CIRCLE 93.8581 -26.5766 0.163153
TEXT 93.8581 -26.5766 11
CIRCLE 91.176 -26.1064 0.234356
TEXT 91.176 -26.1064 12
```

G Robust statistics for a Normal distribution

The Normal, or Gaussian, distribution for mean μ and standard deviation σ can be written as

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}}\ e^{-(x-\mu)^2/2\sigma^2}.$$

When one has a purely Gaussian signal, it is straightforward to estimate σ by calculating the standard deviation (or rms) of the data. However, if there is a small amount of signal present on top of Gaussian noise, and one wants to estimate the σ for the noise, the presence of the large values from the signal can bias the estimator to higher values.

An alternative way is to use the median (m) and median absolute deviation from the median (s) to estimate μ and σ. The median is the middle of the distribution, defined for a continuous distribution by

$$\int_{-\infty}^{m} f(x)dx = \int_{m}^{\infty} f(x)dx.$$

From symmetry, we quickly see that for the continuous Normal distribution, $m = \mu$. We consider the case henceforth of $\mu = 0$, without loss of generality.

To find s, we find the distribution of the absolute deviation from the median, and then find the median of that distribution. This distribution is given by

$$\begin{aligned} g(x) &= \text{distribution of } |x| \\ &= f(x) + f(-x),\ x \geq 0 \\ &= \sqrt{\frac{2}{\pi\sigma^2}}\, e^{-x^2/2\sigma^2},\ x \geq 0. \end{aligned}$$

So, the median absolute deviation from the median, s, is given by

$$\int_{0}^{s} g(x)dx = \int_{s}^{\infty} g(x)dx.$$

If we use the identity

$$\int_{0}^{\infty} e^{-x^2/2\sigma^2} dx = \sqrt{\pi\sigma^2/2}$$

we find that

$$\int_{s}^{\infty} e^{-x^2/2\sigma^2} dx = \sqrt{\pi\sigma^2/2} - \int_{0}^{s} e^{-x^2/2\sigma^2} dx.$$

Hence, to find s we simply solve the following equation (setting $\sigma = 1$ for simplicity – equivalent to stating x and s in units of σ):

$$\int_{0}^{s} e^{-x^2/2} dx - \sqrt{\pi/8} = 0.$$

This is hard to solve analytically (no nice analytic solution exists for the finite integral that I'm aware of), but straightforward to solve numerically, yielding the value of $s = 0.6744888$. Thus, to estimate σ for a Normally distributed data set, one can calculate s, then divide by 0.6744888 (or multiply by 1.4826042) to obtain the correct estimator.

Note that this is different to solutions quoted elsewhere, specifically in Meyer et al. (2004), where the same robust estimator is used but with an incorrect conversion to standard deviation – they assume $\sigma = s\sqrt{\pi/2}$. This, in fact, is the conversion used to convert the *mean* absolute deviation from the mean to the standard deviation. This means that the cube noise in the HIPASS catalogue (their parameter $\mathrm{Rms}_{\mathrm{cube}}$) should be 18% larger than quoted.

H Gaussian noise and the wavelet scale

The key element in the wavelet reconstruction of an array is the thresholding of the individual wavelet coefficient arrays. This is usually done by choosing a level to be some number of standard deviations above the mean value.

However, since the wavelet arrays are produced by convolving the input array by an increasingly large filter, the pixels in the coefficient arrays become increasingly correlated as the scale of the filter increases. This results in the measured standard deviation from a given coefficient array decreasing with increasing scale. To calculate this, we need to take into account how many other pixels each pixel in the convolved array depends on.

To demonstrate, suppose we have a 1-D array with N pixel values given by $F_i,\ i = 1, ..., N$, and we convolve it with the B$_3$-spline filter, defined by the set of coefficients $\{1/16, 1/4, 3/8, 1/4, 1/16\}$. The flux of the ith pixel in the convolved array will be

$$F'_i = \frac{1}{16}F_{i-2} + \frac{1}{4}F_{i-1} + \frac{3}{8}F_i + \frac{1}{4}F_{i+1} + \frac{1}{16}F_{i+2}$$

and the flux of the corresponding pixel in the wavelet array will be

$$W'_i = F_i - F'_i = \frac{-1}{16}F_{i-2} - \frac{1}{4}F_{i-1} + \frac{5}{8}F_i - \frac{1}{4}F_{i+1} - \frac{1}{16}F_{i+2}$$

Now, assuming each pixel has the same standard deviation $\sigma_i = \sigma$, we can work out the standard deviation for the wavelet array:

$$\sigma'_i = \sigma\sqrt{\left(\frac{1}{16}\right)^2 + \left(\frac{1}{4}\right)^2 + \left(\frac{5}{8}\right)^2 + \left(\frac{1}{4}\right)^2 + \left(\frac{1}{16}\right)^2} = 0.72349\ \sigma$$

Thus, the first scale wavelet coefficient array will have a standard deviation of 72.3% of the input array. This procedure can be followed to calculate the necessary values for all scales, dimensions and filters used by *Duchamp*.

Calculating these values is clearly a critical step in performing the reconstruction. The method used by Starck and Murtagh (2002) was to simulate data sets with Gaussian noise, take the wavelet transform, and measure the value of σ for each scale. We take a different approach, by calculating the scaling factors directly from the filter coefficients by taking the wavelet transform of an array made up of a 1 in the central pixel and 0s everywhere else. The scaling value is then derived by taking the square root of the sum (in quadrature) of all the wavelet coefficient values at each scale. We give the scaling factors for the three filters available to *Duchamp* below. These values are hard-coded into *Duchamp*, so no on-the-fly calculation of them is necessary.

Memory limitations prevent us from calculating factors for large scales, particularly for the three-dimensional case (hence the smaller table). To calculate factors for higher scales than those available, we divide the previous scale's factor by either $\sqrt{2}$, 2, or $\sqrt{8}$ for 1D, 2D and 3D respectively.

	B_3 Spline	Triangle	Haar
	$\{\frac{1}{16},\frac{1}{4},\frac{3}{8},\frac{1}{4},\frac{1}{16}\}$	$\{\frac{1}{4},\frac{1}{2},\frac{1}{4}\}$	$\{0,\frac{1}{2},\frac{1}{2}\}$
1 dimension			
1	0.723489806	0.612372436	0.707106781
2	0.285450405	0.330718914	0.5
3	0.177947535	0.211947812	0.353553391
4	0.122223156	0.145740298	0.25
5	0.0858113122	0.102310944	0.176776695
6	0.0605703043	0.0722128185	0.125
7	0.0428107206	0.0510388224	0.0883883476
8	0.0302684024	0.0360857673	0.0625
9	0.0214024008	0.0255157615	0.0441941738
10	0.0151336781	0.0180422389	0.03125
11	0.0107011079	0.0127577667	0.0220970869
12	0.00756682272	0.00902109930	0.015625
13	0.00535055108	0.00637887978	0.0110485435
2 dimension			
1	0.890796310	0.800390530	0.866025404
2	0.200663851	0.272878894	0.433012702
3	0.0855075048	0.119779282	0.216506351
4	0.0412474444	0.0577664785	0.108253175
5	0.0204249666	0.0286163283	0.0541265877
6	0.0101897592	0.0142747506	0.0270632939
7	0.00509204670	0.00713319703	0.0135316469
8	0.00254566946	0.00356607618	0.00676582347
9	0.00127279050	0.00178297280	0.00338291173
10	0.000636389722	0.000891478237	0.00169145587
11	0.000318194170	0.000445738098	0.000845727933
3 dimension			
1	0.956543592	0.895954449	0.935414347
2	0.120336499	0.192033014	0.330718914
3	0.0349500154	0.0576484078	0.116926793
4	0.0118164242	0.0194912393	0.0413398642
5	0.00413233507	0.00681278387	0.0146158492
6	0.00145703714	0.00240175885	0.00516748303
7	0.000514791120	0.000848538128	0.00182698115