# A MIRIAD ROUTINE FOR ANALYZING STRUCTURES IN SPECTRAL LINE DATA CUBES

Jonathan Williams
Harvard–Smithsonian Center for Astrophysics, Cambridge, MA 02138
and
Peter Teuben
Astronomy Department, University of Maryland, College Park, MD 20742

## ABSTRACT

We describe the use of an automated clump–finding routine for analyzing spectral line data cubes that has been recently installed in the MIRIAD package. There are three programs: the main analysis program, `clfind`, that reads in the data cube and outputs a cube of the same dimensions with the clump assignments; a statistics program, `clstats`, that calculates clump positions, sizes, linewidths and masses; and a clump plotting program, `clplot`, based on `velplot`, that allows the user to isolate and produce maps of a user input list of clumps.

## DESCRIPTION OF THE ALGORITHM

Spectral line mapping of a molecular cloud results in a "data cube" of brightness temperatures, $T_{\rm b}(x,y,v)$, where $(x,y)$ describes the map position, and $v$, the velocity. Molecular clouds are inhomogeneous: $T_{\rm b}$ is not a constant, but varies from pixel to pixel, both spatially, and in velocity. This variation is typically shown in the form of two dimensional slices of the data cube, e.g. channel maps (fixed $v$) and position-velocity cuts (fixed $x$ or fixed $y$).

By looking at a series of such slices, it is possible to follow the observed structures in the third dimension. However, systematically identifying and characterizing all the structures in the cube is a slow, pain-staking, and subjective task, particularly in the case where different features merge together at lower intensities. To both speed up and objectify the analysis, an algorithm, `clfind`, was developed (Williams, de Geus, & Blitz, 1994). The original motivation and assumption behind this was that the fluctuations in the data cube (in this case $^{13}$CO) correspond to true physical density enhancements – clumps (Blitz, 1993) – in the cloud. This is undoubtedly a simplified description of a cloud, and the validity of such a clump decomposition has been challenged (Houlahan & Scalo, 1992). `clfind` is best viewed as an analysis tool for decomposing a complex dataset into emission peaks which may be processed further however one chooses.

We briefly describe the method here. For a more detailed discussion, the reader is referred to the Williams et al. (1994) paper. `clfind` is based closely on how the eye would analyze the maps: it contours the dataset, finds peaks, and follows them down through sequentially lower contour levels. Here, however, the contours are in the full three $(x, y, v)$ dimensions of the cube.

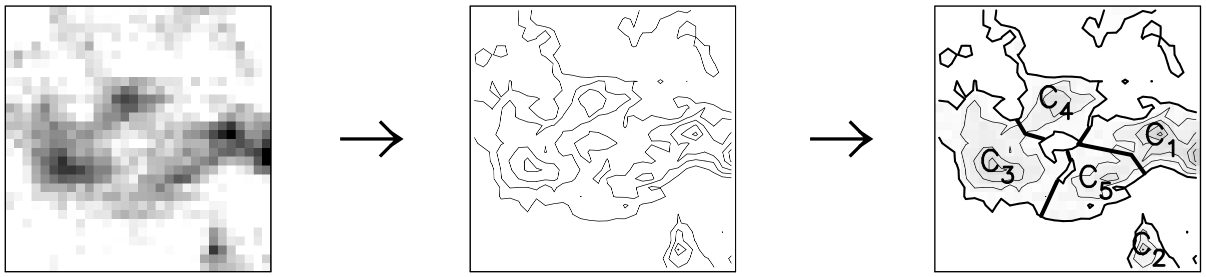

**Figure 1:** Schematic diagram of how `clfind` works on a dataset. The data is a set of continuous values which is then contoured or "quantized" into a discrete set of linear temperature increments. The program starts from the highest contour and works its way down to lower levels, finding new clumps and extending previously defined ones until it reaches the final level. Contour levels are split (heavy lines) if they encircle more than one clump. The final result is a set of clumps, $C_1$ through $C_5$.

There are two fundamental issues that must be addressed with such a scheme. First, how the data is contoured, and second, how to handle the case when two (or more) clumps merge together.

Contours should be evenly spaced, $T = \Delta T, 2\Delta T, 3\Delta T, \ldots$, since the noise adds in linearly at each level and, as we see, it is noise considerations that dictate what the optimum contouring is. If $\Delta T$ is too small, then the contour map will appear to be full of structure, but it will be impossible to distinguish the true features apart from noise spikes. On the other hand, if $\Delta T$ is too large, the contour map will lack contrast and subtle features may be missed. Optimum contouring is a tradeoff between recovering as many real features from the map as possible, but not being misled by noise. By testing with simulated data, Williams et al. (1994) concluded that the optimum contours were spaced at $\Delta T = 2T_{\rm rms}$, where $T_{\rm rms}$ is the (constant) rms noise in the map. For these contours, the percentage of false detections is less than 2%.

For maps with non-uniform noise, e.g. interferometer maps, the contours should be set to twice the maximum $T_{\rm rms}$ in the region of interest. This is because the percentage of false detections rises steeply to $\sim 50\%$ as the contouring level decreases to the rms noise. Note that this is a general result that is worth keeping in mind when presenting contour maps.

The contouring has the effect of quantizing the data. That is, $T_{\rm b}$, which is a set of continuous numbers, is transformed to a discrete set of contour levels. `clfind` defines a clump to be a collection of pixels which, at least at its highest contour, is isolated from (not connected to) any other clump. Two pixels are defined to be connected if they are within one beam size and a velocity resolution element of each other. When `clfind` reads in the data cube, it searches the header for the beam size. If it does not find it, it assumes that the beam is the pixel separation in each of the $xy$-axes. The velocity resolution is taken to be the channel width in the $v$-axis.

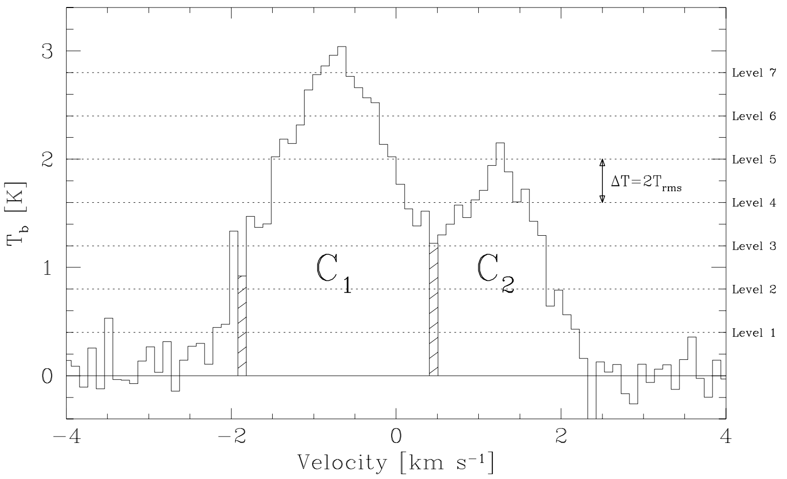

**Figure 2:** Spectrum with contour levels at $T = 2T_{\rm rms}, 4T_{\rm rms}, 6T_{\rm rms}...$ overlayed. `clfind` works from the highest contour level (7) where the first clump is defined, and works down through the levels. A new clump is defined at level 5, and the two merge at level 3.

Any two clumps, then, must be isolated at some contour level. They may, however, merge at a lower level. Figure 2 shows a one-dimensional example. The horizontal dotted lines show the contouring levels spaced at $\Delta T = 2T_{\rm rms} = 0.4$ K. Level 7 (2.8 K$\leq T <$3.2 K) is the highest level for which there is data, and `clfind` therefore defines clump C$_1$. It does so by creating a clump assignment array, $A$, which has the same dimensions as $T_{\rm b}$, where $A(\mathbf{x}) = n$ if pixel $\mathbf{x} = (x, y, v)$ belongs to clump $n$. The clump number $n$ is initially set to zero, and increments by one for each new clump that is found. This clump is then extended (pixels are assigned) to level 6, then level 5, where clump C$_2$ is defined. Both clumps are extended to level 4, and they then merge at level 3, i.e. the contour at 1.2 K surrounds both clumps. This contour level is split between the two by a "friends-of-friends" algorithm: `clfind` takes the sets $(x_1, y_1, v_1)$ and $(x_2, y_2, v_2)$ of pixels that have been assigned to clumps C$_1$ and C$_2$, and finds their "friends" (pixels that are connected to at least one member of the set). If a friend of clump C$_1$ lies in level 3, and is not a friend of clump C$_2$, then it is assigned to C$_1$. If it is a friend of both clumps, it lies on the border of the two and is flagged as being unassignable. Once all the friends have been assigned or flagged, the process is iterated to the next set of friends and so on until there are no more in that level. This process is easily generalized to more than two merged clumps. In this way the contour level is split up so each pixel is either assigned to a certain clump, or is flagged as being on the border and not assigned. The process is then continued to lower levels (where new clumps may be defined, and other clumps may merge with C$_1$ and C$_2$). The process of defining new clumps stops at level 2 ($4T_{\rm rms}$), but clumps are extended, and may merge, down to level 1.

The way of cutting contours shown diagrammatically in Figures 1 and 2 is a rather crude way of decomposing the data cube. Each pixel is either assigned to one clump or none at all, but the reality is that the emission at any one pixel in the dataset is the combined sum from many clumps. However, in Appendix B of Williams et al. (1994), it is argued that for a clump to be visible as an isolated peak, the separation between clumps is so large that only $\sim 10\%$ of the emission of a clump extends beyond the boundary defined using the friends-of-friends algorithm. The advantage of this technique is the simplicity and speed it affords. Moreover, by simply dividing up contours, and not interpolating into the blended regions, `clfind` makes no assumptions about clump shapes or profiles. However, `clfind` is not well suited to analyzing regions of very high blending (see last section).

## Example

As an example of its use, here is the (edited) output from running `clfind` on a $^{13}$CO dataset of the Rosette Molecular Cloud (Williams, Blitz, & Stark, 1995):

```
CLFIND version 5.0 24-Feb-93
Data file: ros13co
The beam size is: 1.00 by 1.00 pixels
Output clump assignment file: ros13co.cf
------------------------------------------
 CLFIND
 Starting level:                      1
 Temperature increment:            0.50 K
------------------------------------------
 Level 17:    1 pixels
 .
 .
 Level  3:  600 pixels
 Level  2: 1188 pixels
 Level  1: 3586 pixels

 Level 17:    1 regions
 .
 .
 Level  3:   86 regions
 Level  2:   90 regions

 Level 17:    1 new clumps
 .
 .
 Level  3:   23 new clumps
 Level  2:   28 new clumps
```

```
Extending to level 1
Testing clumps for badness
==========================================
Program terminates sucessfully
   88 clumps found (    8 clumps stopped)
==========================================
```

After first reading in the data file, the starting contour level, and the contouring interval, `clfind` first determines how many contour levels there are, and how many pixels are in each level. It then determines the number of regions (contour "islands") for each level, and finally, clumps are defined and regions split between clumps in the manner discussed above. Regions and clumps are only defined to the second level, but having completed this task, `clfind` then extends the previously defined clumps down to level 1, and then counts the number of pixels in each clump. Any clumps with less than $n_{\rm min}$ pixels are rejected (the clump assignment array is nulled). $n_{\rm min}$ has a default value of 4, but is a parameter that may be set by the user. Here, 88 clumps pass the test, and 8 fail.

The $\sim 10\%$ of clumps that fail are either defined at level 2 and even after extending to level 1 are too small to be deemed significant, or they are noise spikes that poke up above a contour level, but which, because they are next to a large clump, do not grow. An example of this can be seen in Figure 2, where – restricting ourselves to this one dimension only – there is a solitary disconnected pixel at level 3 (at $v \simeq -2$ km s$^{-1}$) that would initially be defined as a new clump. At the next lower level, however, it would immediately merge with C1, and the pixel on the border of the two would be flagged as unassignable. Although at level 1 it gains 2 more pixels, this is insufficient to pass the minimum pixel criterion ($n_{\rm min} = 4$) to be a viable clump. Note that, for visualization purposes, this example is one dimensional, and in a three dimensional dataset, it is rarer for a single pixel to be isolated in this way. Nevertheless, due in part to such failed clumps and the unassigned pixels on clump borders, and, in particular, to the large numbers of pixels at and below the first contour level that are not assigned, the integrated flux contained in all the clumps will be less than the summed flux in the data cube. For the case here of the Rosette (peak signal to noise $\simeq 30$, but more typically $\sim 10$), 89% of the emission above $2T_{\rm rms}$, and 68% of the total emission, was assigned to clumps (Figure 5 of Williams et al., 1995). Percentages will be higher for data with greater signal to noise.

Once all the assignments have been made, the clump assignment array, $A$, is written out as a MIRIAD file with the name of the input data file appended with ".cf". Each pixel, $\mathbf{x} = (x, y, v)$, has a temperature, $T_{\rm b}(\mathbf{x})$, stored in the input data file, and a clump number, $n_c = A(\mathbf{x})$ stored in the output file. Each pixel either belongs to one clump ($n_c \geq 1$), or none at all ($n_c = 0$). These two files can then be read in by other MIRIAD routines to determine clump properties and statistics, or simply for plotting. Two such programs, `clstats`, and `clplot` are described in this memo, but it is relatively straightforward to copy and edit the code to analyze the results in other ways. By using the routine FITS to translate to fits format, other programs, such as IDL[1], which is particularly well suited to handling arrays, can be used.

**Tree Structure**

After sucessful completion of the clump-finding algorithm, `clfind` concludes by listing some basic statistics about each clump that was found. For each level, it lists any new clumps that were found, their peak temperature, and their pixel count. As an example, here is a section of the output from the tail of the Rosette log:

```
New clump  91  Tpeak = 1.1  Npix =   0
New clump  92  Tpeak = 1.0  Npix =  13
New clump  93  Tpeak = 1.1  Npix =   0
New clump  94  Tpeak = 1.3  Npix =  13
New clump  95  Tpeak = 1.2  Npix =   6
New clump  96  Tpeak = 1.1  Npix =   0
Merged clumps: 46  47   0   0   0   0
Merged clumps: 34  48   0   0   0   0
Merged clumps:  2   3   4   6  11  12
               15  16  19  20  23  27
               28  32  35  43  50  54
```

As can be seen, it also lists which clumps merged together at this level. Here, for example, clumps 46 and 47 blended, as did clumps 34 and 48. Then there is a string of clumps that are blended together. With this information, it is possible to reconstruct a "structure tree" (Houlahan & Scalo, 1992), which shows the interrelation between the clumps in the cloud. Such a diagram, or a mathematical description of it, may be used as a basis for a cloud-cloud or cloud-simulation comparison (e.g. Adams & Wiseman, 1994).

## HOW TO RUN THE PROGRAMS

CLFIND

Like other MIRIAD programs, `clfind` (and `clstats` and `clplot`) can be run from either within the MIRIAD shell by typing at the Unix prompt, for example,

```
  > miriad
```

and then within the shell,

```
  Miriad% inp clfind
```

[1]`clfind`, `clstats`, and `clplot` have been coded in IDL and are available from http://cfa–www.harvard.edu/~jpw/clfind.html

which will respond with the four keywords,

```
Task:   clfind
in       =
dt       =
start    =
nmin     =
```

`in` is the name of the input data file, `dt` is the contouring interval, `start` is the starting contour level, and `nmin` is the minimum number of pixels required for it to be considered a clump. `start` and `nmin` have default values of 1 and 4 respectively, `dt` has no default. The output file name is derived automatically from the input by appending ".cf" to the end. For each program, there is a help file which lists and describes each keyword.

Alternatively, all 3 programs may be run directly from the command line. For example, to run `clfind` on the Rosette molecular cloud,

```
clfind in=ros13co dt=0.5 > clfind.log
```

The output here is piped to a file for safekeeping. Otherwise it will scroll to the screen.

## CLSTATS

Once the clumps have been identified, the statistics program, `clstats`, calculates clump positions, sizes, linewidths and other basic properties. It reads in both the data and clump assignment cubes and cycles through each clump in turn. For each clump number, $n$, it takes the set of pixels, $\mathbf{x_n} = (x_n, y_n, v_n)$ where $A(\mathbf{x_n}) = n$, and calculates the peak position, clump area, velocity dispersion, and integrated intensity. (The user may create their own routines for calculating their favourite clump statistics here).

The help file describes the many keywords necessary for operation of this program. Output is again scrolled to the screen, but may be piped to a file. As an example, here is a possible command line for the Rosette molecular cloud,

```
clstats in=ros13co dist=1600 X=4.4 xy=abs > ros13co.stats
```

## CLPLOT

Having cataloged the clumps in your data cube, it is essential to look at each one in turn and verify that its identification is reasonable, and that it is not, for example, an unusual noise spike, large blend of many low level features, or that it lies on the edge of the map and is not well defined. `clplot` provides a rudimentary means of doing this.

`clplot` is a direct clone of `velplot`, edited so as to read in both data and clump assignment data cubes, and with an option to read in a list of clump numbers to plot. It is a menu driven program that offers the possibility of plotting spectra, velocity integrated maps, and position velocity slices.

For completeness, here is how to plot the results from the Rosette analysis,

```
clplot in=ros13co device=/xw
```

Plots of individual clumps and their relation to each other and the cloud as a whole are, of course, also very useful scientifically. `clplot` does not have the flexibility to produce publication quality plots, but like `velplot`, can write out MIRIAD images for further processing by, e.g. WIP. Alternately, the identical format of the data and clump assignment cubes makes it relatively straightforward to create your own custom designed plots.

## LIMITATIONS OF THE ALGORITHM

The algorithm can be very slow for large datasets. Although it took only $\sim 30$ seconds to reduce the moderately sized $61 \times 61 \times 27$ Rosette $^{13}$CO dataset on a Sparc 20 computer, there are many nested loops, and the computation time increases roughly as $N_{\rm pix}^3$, where $N_{\rm pix} = N_x N_y N_v$ is the number of pixels in the datacube. Thus a cube of size $100 \times 100 \times 50$ (and with $15-20$ contour levels) would take about an hour to analyze. To improve efficiency it is highly recommended that only a subimage of the datacube be analyzed. The velocity axis, for example, should be truncated to the range where there is signal. For example,

```
imsub in=ros13co.all out=ros13co.sub region="images(12,38)"
```

`imsub` also has the advantage that it can take an odd stride through the cube, useful in the case where the data is oversampled (as may occur, for example, during processing with AIPS).

In addition to the large number of CPU cycles, `clfind` also requires a large amount of memory to handle the large arrays that it creates. The maximum array dimensions are defined in the include file, `clfind.h`; current settings require 112 Mbyte. This file can be edited (`clfind` must then be recompiled) to tailor the program to your particular dataset.

`clfind` works well for analyzing large scale maps of GMCs, for which it was designed. The emission in these cases is fairly well separated and has the appearance of being in mostly distinct clumps. It is, therefore, a good approximation to assign all the emission at any one pixel to a single clump. Higher resolution maps centered on a region of emission have a qualitatively different appearance of many peaks close together. In this case, the degree of blending of the different clumps is high, and it becomes more difficult to individually allocate the emission to any one peak. When many clumps merge together, a significant fraction of the emission at each pixel may originate from more than one clump, and the approximation of assigning all the emission to a single clump is no longer valid. `clfind` will still find the significant emission peaks and show their relation to each other, but it will not calculate clump sizes, linewidths or masses very accurately. In this case, other methods that interpolate into the merged region, e.g. the tri-axial gaussian technique described by Stutzki & Güsten (1990), are more appropriate. Such an interpolation, however, necessarily involves an assumption about clump shapes and profiles.

## References

Adams, F.C., & Wiseman, J. 1994, Ap. J., 426, 629

Blitz, L. 1993, in *Protostars and Planets III*, Levy and Lunine, eds., University of Arizona Press: Tucson, p.125

Houlahan, P., & Scalo, J., 1992, Ap. J., 393, 172

Stutzki, J., & Güsten, R., 1990, Ap. J., 356, 513.

Williams, J.P, de Geus, E.J., & Blitz, L. 1994, Ap. J., 428, 693

Williams, J.P, Blitz, L., & Stark, A.A. 1995, Ap. J., 451, 252