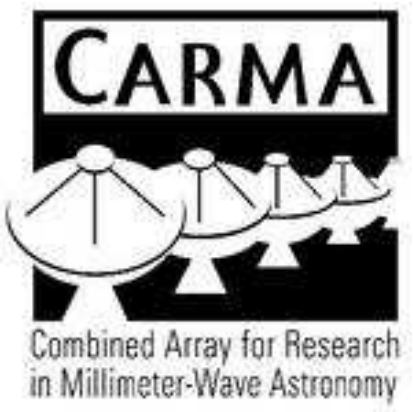

CARMA Memorandum Series #999

IMAGE CUBE I/O: Comparing access patterns

Peter Teuben
University of Maryland

April 16, 2013

ABSTRACT

A number of data cube I/O patterns are defined, and their methods are compared in performance and code complexity in a few common astronomical data analysis packages.[1] We focus on multi-dimensional access (> 3-Dim). Some relevant benchmarks are defined. For the example of clumpfinding an overview is giving of the available community codes, and a unified frontend to running and analyzing comparing their clumps is described.

This is work in progress for AStute.

[1]partially based on a talk given at STSCI on March 31, 2011

Change Record

Revision	Date	Author	Sections/Pages Affected
	Remarks		
0.1	2011-Oct-15	P. Teuben	
	Initial version		
0.2	2011-Dec-05	P. Teuben	
	Add cube examples		
0.3	2013-Apr-12	P. Teuben	
	Various updates for AStute, added clumpfinders		

1. Introduction

2. Image Cube Analysis Patterns

First of all, some example image cubes, all available upon request. The typical image "cubes" we will discuss here are 1k x 1k x 10k, but we are using several examples: [2]

1. Orion EVLA datacube is 96 x 96 x 24,012. Although spatially small, rich in spectral lines. This cube is well separated in space and line though, unlike C. Brogan's SMA data (not available yet)
2. The NGC 1333 example , about 1000 x 1000 x 150, in three molecules, two of which have 3 hyperfine lines. (CARMA 2011 data, not available in final form yet)
3. SMC ATCA (HI) cube, an example of heavily warped galactic disk. Extra bonus difficulty: this galaxy is so nearby, that perspective corrections are needed. (see also 1999MNRAS.302..417S.pdf). Cube is 578 x 610 x 78 (107MB).
4. the NGC 6503 cube, a simple galaxy rotating disk (VLA data, 2005?) See figure **??**
5. ROS13, the Rosetta Nebula, in optically thin ^{13}CO, used as the benchmark in the 1994 ClumpFind paper. This is a small 61 x 61 x 27 cube, but 39533 of the 100467 pixels (40%) are masked out.

2.1. single line

The NGC 6503 VLA datacube is fairly modest in size, and relatively straightforward to analyze. Single profiles. Some challenges are to find HVC in "forbidden" areas, as has been done in other galaxies. There is at least one suspicious blob in this cube, not clear if this has been noted in the literature. An older VLA cube is 256 x 256 x 31 (4MB) observed in 1983 (published in 1985AJ.....90.1038V). Newer VLA data is 371 x 251 x 89 (32MB) from 1996 (published in e.g. 2009AJ....137.4718G)

2.2. hyperfine transition

Our own CARMA data of three molecules for NGC 1333. HCO^+ is a single line, HCN a simple triplet, and N_2H^+ a triplet of which the low and middle component contain a triplet in themselves, usually forming a blended set of three lines each. At high enough resolution this would 7 hyperfine transition lines. This combination gives for interesting experiments in clump finding.

[2]Some of this on: ftp://ftp.astro.umd.edu/pub/carma/data

2.3. single line,complex geometry, possible warping

The ATCA HI data of the SMC is very complex. At many positions multiple components are visible.

2.4. multiple lines, with possible blending

The Orion EVLA (900MB) datacube is an example of something that will approach the size and complexity of what we are aiming at. It

The SMA data C.Brogan has been showing at meetings is overwhelmingly complex, a lot of blending, a large fraction of un-identified lines.

3. Modeling

Understanding observations, in particular their modeling, will be a key component of analysis code. First some notation, in a very generic way (cf. CASA's measurement equation):

- M : model
- T : telescope model that transforms a model into telescope (observable) data
- O : observations
- R : reduction operation (e.g. dimensional, 3d to 2d or 1d)

3.1. Invert the observations

Either the observations can directly be inverted:

$$M = O^{-1}$$

or the model can be inverted after some reduction operator

$$M = (O.R)^{-1}$$

3.2. Match the transformed model to the observations

The model can be virtually observed through with a model telecope, and then compared (and minimized w.r.t. various parameters) to the observations:

$$||M.T - O||$$

3.3. Examples of reduction operators

- Taking moments along the 3rd spectral axis, one obtains the total emission, the mean velocity[3], velocity dispersion and even higher order moments.
- For given geometry of an ellipse (center, position angle and eccentricity), one can fit, in this case a circularly rotating disk

$$V_{obs}(x, y) = V_t(R) \cos\theta \sin i$$

Several reduction parameters can be used in succession. For example, after taking a first moment to obtain a mean velocity along the 3rd axis,

3.4. Segmenting and Clump Finding

Astronomers and Computer Scientists can use different words to mean the same thing. Segmenting in CS (normally done in 2D images) is the essentially the same a clump-finding in 2D and 3D cubes. All it does is identify which pixels/voxels belong to a "clump", after which various properties can be determined for these clumps, and multi-variate analysis , principle component (PC) be used to find relationships.

Idea: masks can be used to define clumps, each clump their own mask? Giving more modular code? Mask has on optional convex area definition, with a classic true/false bitmask for each pixel inside.

4. Package Overview

1. **miriad** : Uses the classic low-memory line-based I/O to fortran-style (column-major) multi-dimensional arrays. Access to (hyper) cubes is done by indexing. This is done in the `xyio` routines. This approach makes for slightly complicated application code. An alternative approach is done in the `xyzio` routines, which simplify access when higher level needs access different
2. **nemo** : uses memory based I/O, the data-array can be read in one big chunk, and simple `data[ix][iy][iz]`, as well as the miriad style line based access for very large cube, can be used.
3. **eclipse** : uses memory based fits image based I/O. Developed in 1995, their last version appears to be from 2005 and perhaps fell out of fashion. Was meant to be the driving engine for (ESO) pipeline. Now there is the Common Pipeline ... Appears

[3]mean, median, peak; this and many more used

4. **gipsy** : a novel new approach using GDS - cf. Starlink's HDS?
5. **CASA** : lattice I/O, very general slicing access methods
6. **pyfits** : using scipy **data[iz,iy,ix]** Note that you can opt to store the arrays in column major or row major order.

5. MIRIAD

MIRIAD started development in 1988, and thus (similar to Classic AIPS) adopted a low memory approach to what we then realized could be sizeable 100 MP cubes (1k x 1k x 100 channels).

5.1. xyio

The industry standard at the time was AIPS, and thus a typical access to miriad images was row based[4]

```
real sum, data(MAXDIM)

call xyopen(lun,'cube.mir',nx,ny,nz)
do k=1,nz
  call xysetpl(lun,k)
  do j=1,ny
    call xyread(lun,j,data)
    do i=1,nx
       sum = sum + data(i)
    enddo
  enddo
enddo
call xyclose(lun)
```

Two important aspects to image I/O should be added to this simplified code example: **masking** and **region** selection. A masking cube lives separate from the image data cube, and is boolean in nature, instead of floating point. Programs exists to mask out data that are deemed "bad". The programmer needs to call masking I/O routines that are very similar to the image data I/O. Region selection is normally parsed from the commandline `region=` keyword, and transformed into an `integer runs(3,MAXRUNS)` data structure. Instead of looping over the cube in the before mentioned method, acccess would now become:

[4]the examples below use a simplified argument list to illustrate the method, the actual implementation is different

```
real sum, data(MAXDIM)

call xyopen(lun,'cube.mir',nx,ny,nz)
do k=1,nz
  call xysetpl(lun,k)
  do j=1,ny
    call xyread(lun,j,data)
    do i=1,nx
       sum = sum + data(i)
    enddo
  enddo
enddo
call xyclose(lun)
```

Application code that does not need to accumulate data beyond a single dimension can easily be written. For routines that need to keep an image plane in memory, can be written lazy as follows:

```
real data(MAXDIM,MAXDIM)

do j=1,ny
  call xyread(lun,j,data(1,j))
enddo
```

for large values of `MAXDIM` it is more efficient to fill the array as a one dimensional array, and pass the work on to a subroutine that automagically knows about the two dimensions:

```
real data(MAXDATA)

idx = 1
do j=1,ny
  call xyread(lun,j,data(idx))
  idx = idx + nx
enddo
call worker(data,nx,ny)
..
subroutine worker(data, nx, ny)
integer nx, ny
real data(nx,ny)
..
end
```

5.2. xyzio

When access to datacubes is not channel based, the algorithms using the `xyio` routines can become very clumsy. They were developed (with ideas taken from the Gipsy GDS routines) to allow application programmers to write cube orientation independant code, where the `xyio` routines made this very complex.

```
xyzopen(lun, name, naxis,axlen())    -->   same as xyopen()
xyzclose(lun)
xyzsetup( tno, subcube, blc(*), trc(*), viraxlen(*), vircubesize(*) )
   subcube:   ' ' = single pixel
              'z' = i/o on profiles in the z direction
             'xy' = i/o on XY-planes
            'xyz' = i/o to get subcubes
             'zy' = plane and permute

xyzmkbuf()
xyzs2c( tno, subcubenr, coords(*) )
xyzc2s( tno, coords(*), subcubenr )

xyzread( tno, coords(*), data(*), mask(*), ndata )
xyzpixrd( tno, pixelnr, value, mask )
xyzprfrd( tno, profilenr, profile(*), mask(*), ndata )
xyzplnrd( tno, planenr, plane(*), mask(*), ndata )

xyzwrite( tno, coords(*), data(*), mask(*), ndata )
xyzpixwr( tno, pixelnr, value, mask )
xyzprfwr( tno, profilenr, profile, mask, ndata )
xyzplnwr( tno, planenr, plane, mask, ndata )
```

A surprising benchmark: A 1482 x 1482 x 150 cube (1.3GB) needed to be Hanning smoothed (miriad task: `hanning`), which took about 210 cpusec straight on this cube. Instead, using `reorder mode=312; hanning; reorder mode=231`, we measured 5.5, 6.6 and 10.4 cpusec resp, so only a sum of 22.5. Clearly, the `xyzio` access was not optimized here. An in-core smooth is much faster. NEMO, even though the file size is twice the size (double precision cubes) was able to hanning smooth this cube in 3 cpusec!!

```
casa> %time importfits('mem1res.all.mask15/fits','h0')
```

```
      CPU times: user 3.87 s, sys: 1.24 s, total: 5.11 s
      Wall time: 5.80 s

casa> ia.open('h0')
casa> ia.coordsys().names()
      ['Right Ascension', 'Declination', 'Frequency', 'Stokes']

casa> %time ia.hanning('h0s0',axis=0)
      CPU times: user 1.95 s, sys: 0.90 s, total: 2.86 s
      Wall time: 3.17 s
casa> %time ia.hanning('h0s1',axis=1)
      CPU times: user 2.56 s, sys: 0.76 s, total: 3.32 s
      Wall time: 3.57 s
casa> %time ia.hanning('h0s2',axis=2)
      CPU times: user 9.47 s, sys: 0.35 s, total: 9.82 s
      Wall time: 10.30 s

# expected, VEL axis is slowest

casa> %time imtrans('h0','hv0','2013')
      CPU times: user 4.97 s, sys: 1.43 s, total: 6.39 s
      Wall time: 6.70 s
casa> ia.open('hv0')
casa> ia.coordsys().names()
      ['Frequency', 'Right Ascension', 'Declination', 'Stokes']

casa> %time ia.hanning('hv0s0',axis=0)
      CPU times: user 8.31 s, sys: 0.56 s, total: 8.87 s
      Wall time: 9.10 s
casa> %time ia.hanning('hv0s1',axis=1)
      CPU times: user 1.95 s, sys: 0.66 s, total: 2.61 s
      Wall time: 2.85 s
casa> %time ia.hanning('hv0s2',axis=2)
      CPU times: user 3.41 s, sys: 0.62 s, total: 4.04 s
      Wall time: 4.74 s

# this was not expected, VEL axis should be fastest
```

6. NEMO

The NEMO package started development in 1986, just a little ahead of MIRIAD, but is mostly written in C and also assumes images to be present in memory. A compiler option determined if images are stored in column-major ("fortran") or row-major ("C") order. The `tsf` program can show how images are stored, in a human readable form:

```
% ccdmath out=- fie="%z+10*%y+100*%x" size=2,3,4 | tsf -
set Image
  set Parameters
    int Nx 2
    int Ny 3
    int Nz 4
    ...
    char Storage[5] "CDef"
  tes
  set Map
    double MapValues[2][3][4] 0.00000 1.00000 2.00000 3.00000 10.0000 11.0000
      12.0000 13.0000 20.0000 21.0000 22.0000 23.0000 100.000 101.000 102.000
      103.000 110.000 111.000 112.000 113.000 120.000 121.000 122.000 123.000
  tes
tes
```

showing that in this mode data are stored in C order but with the (at least odd in C) convention of writing `data[ix][iy][iz]`

```
imageptr iptr;
real sum = 0.0;

read_image('cube.nemo',&iptr);
for (i=0; i<Nx(iptr); i++)
  for (j=0; j<Ny(iptr); j++)
    for (k=0; k<Nz(iptr); k++)
       sum += CubeValue(iptr,i,j,k);
```

7. KARMA

Intelligent arrays, as defined by Karma, use a different indexing concept for arrays:

```
data[x[i]+y[j]+z[k]]
x[i] = i
y[j] = nx*j
z[k] = nx*ny*k
```

This scheme also has the advantage that non-contiguous arrays may be implemented by simply changing the index arrays. Tiled and toroidal arrays fall in this category.

Note that recent experiments on 2011 type architecture showed that classic `data[ix][iy][iz]` access is about 10% faster than the intelligent access method . Perhaps during its development on and with different architectures and compiler there was a significant advantage. However, non-contiguous array access can be an interesting advantage in a distributed environment.

8. GIPSY

9. ECLIPSE

10. CASA

Lattice I/O. now isolated within `casacore`.

11. PYTHON

The `scipy` and `numpy` module in python has quite a capable datastructure

```
import numpy as np

a = np.arange(6.0)
a.resize(2,3)
print a[1][2]
print a[1,2]
```

12. idl/gdl

In a few cases IDL (the one you need a license for)[5] and GDL (GNU Data Language)[6]

13. matlab

14. R

The `R` language has a solid strong following in statistics, and a small group of loyal users in astronomy as well.

15. Coordinate Systems

Within Astronomy the WCS conventions (mostly implemented within FITS) are defined in a series of papers (Calabretta & Greisen). Astronomers/FITS uses a right handed system where the lower left pixel (center) is (1,1), and counts to (NAXIS1,NAXIS2). Hence the first axis really runs from 0.5 to NAXIS1+0.5.

[5] `http://www.exelisvis.com/ProductsServices/IDL.aspx`

[6] `http://gnudatalanguage.sourceforge.net/`

In geo spatial (and image) data (e.g. gdal.org) we find the typical convention that the origin is the upper left pixel, and the coordinate system runs from 0 to N instead of 0.5 to N+0.5.

16. BENCHMARKS

An 880MB EVLA dataset (`orionall_hannclean_hotcore_cube.fits`, 96 x 96 x 24012), was used to benchmark the various access modes our analysis code uses. We will also need a more typical ALMA dataset, 2k x 2k x 4k, but these are not available yet and 5 times bigger.

```
time fits in=orionall_hannclean_hotcore_cube.fits out=map1 op=xyin
     3.533u 1.147s 0:10.04 46.5%     0+0k 1735952+1784664io 12pf+0w
time histo in=map1
     3.253u 0.350s 0:03.62 99.4%     0+0k 64+8io 1pf+0w

%time importfits('orionall_hannclean_hotcore_cube.fits','evla1')
     CPU times: user 6.68 s, sys: 1.30 s, total: 7.98 s    Wall time: 8.61 s

%time imstat('evla1')
     CPU times: user 4.79 s, sys: 0.15 s, total: 4.94 s    Wall time: 4.98 s
```

16.1. Access Patterns

Some of the access patterns may need extra care if masking and region selection is active, particularly the latter.

- 1D processing. This can loop over the data in any fashion, in fact, a base benchmark would be like a gigantic 1D array.
- 2D processing. This requires a whole image to be in memory, e.g. fitting a galaxy image to a model, subtracting a background. Perhaps we should assume a 2D image should be the minimum assumption the software should be capable of holding in memory.
- 2D from N-D processing. This would imply transposing a higher dimension into 2D and working on this 2D, and perhaps looping over the other dimensions reducing that dimension. Examples below.
- 3D neighborhood.
- 4D neighborhood

- 4D cube transposing. This is where many molecules (`m`) are present in velocity. `A[m][v][y][x]` needs to become `B[m][y][x][v]`, fitting needs to be done to reduce to `C[m][y][x]`, where `C` designates total intensity, mean velocity and dispersion.

- 5D cube transposing. Same as previous, but where polarization is present. `A[m][p][v][y][x]` however will have a small number of discrete values of `p`, at most 4 (I,Q,U,V) and it is unclear if a dimension should be sacrificed or separate images be used.

17. Recommendations

- There should be no assumptions on what dimension (spatial, spectral) is smaller or bigger. OmegaCAM works on 16k x 16k images, but has small number of images to process. EVLA may have small 100 x 100 images, but can have 24k spectral channels. Both are about the same size (1GB).

- A simple I/O alorithm, allowing all data to fit in memory, should be available as well

- Tiling in different dimensions should be possible (xyzio? CASA::lattice?) to optimize certain algorithms on big data.

- Just thinking 3-dimensional datacubes is not correct, we need at least 4-dimensional, and if polarization is present, perhaps 5.

- scripting access (e.g. python) should be available.

18. Questions

- how well does CASA integrate their "ipython" based environment with other now common "ipython" based environments, or is the user forced to have several windows open. How then to transfer science objects accross? Do we need to pickle this?

19. Conclusion

- miriad twice more efficient than nemo i/o, looking at the system time to process an image. this is despite the fact that miriad is line based, nemo is cube based!

- casa faired rather bad, at least twice as slow as miriad.

acknowledgements: We thank bla bla. Maybe Rob Olling for his IDL prostelization.

REFERENCES

MIRIAD: http://carma.astro.umd.edu/miriad

NEMO: http://www.astro.umd.edu/nemo

ECLIPSE:

CASA:

GIPSY:

KARMA: Programmers Guide (200+ pages, 2006, by Richard Gooch)

Sault, R. J., Teuben, P. J., and Wright, M. C. H. 1995. *A Retrospective View of MIRIAD* - 1995ASPC...77..433S

Appendix A: Array Storage Nomenclature

Let us list the dimensions of an D-dimensional data structure as $N_1, N_2, N_3, ...N_D$.

For 2-dimensional arrays, or matrices, we perhaps all remember that "Fortran is column-major" and "C is row-major". For multi-dimensional arrays this naming convention is less clear

When using row-major order, the difference between addresses of array cells in increasing rows is larger than addresses of cells in increasing columns.

```
Fortran:
  real fdata(3,2)
  data fdata/1,2,3, 4,5,6/

C:
  float cdata[2][3] = { {1,2,3}, {4,5,6} };

Python/SciPY/pyfits:
  hdu  = pyfits.open('cube.fits')
  data = hdu[0].data
  data[0,0] ... data[1,2]
```

Examples of **column major** programming languages are Fortran, Matlab, R and Octave, where columns are stored behind each other. data(r,c) - major last index.

Examples of **row major** programming languages are C/C++, scipy/pyfits?. Rows are stored behind each other. data[r][c] - major first index.

19.1. NEMO

An Image/Cube in NEMO is 2D or 3D and has dimensions (nx,ny,nz). Here X is the first dimension, and Z the last dimension in the cube. The data is either stored in `CDEF` or `FORDEF` mode.

```
// CDEF:
// row major: data[iy][ix]   data[r][c]   offset = row*NUMCOLS + col
// In row-major order, the last dimension is contiguous,
// ??row-major wise, x varies fastest, so you 'd locate the (x,y,z)th item at z*ny*nx+y*nx+x
#define MapValue(iptr,ix,iy) (*( (iptr)->frame + iy + Ny(iptr)*(ix)))
#define CubeValue(iptr,ix,iy,iz) (*( (iptr)->frame + iz + Nz(iptr)*(iy + Ny(iptr)*(ix))))

// FORDEF:
// column major: data(ix,iy)  data(r,c)   offset = col*NUMROWS + row
// In column-major order, the first dimension is contiguous,
// ??? offset = row + column*NUMROWS
// ?? Column-major-wise z varies fastest and x varies slowest, so (x,y,z) would be at the z+y*nz+x*ny*nz

#define MapValue(iptr,ix,iy)  (*( (iptr)->frame + ix + Nx(iptr)*(iy)) )
#define CubeValue(iptr,ix,iy,iz) (*( (iptr)->frame + ix + Nx(iptr)*(iy+Ny(iptr)*(iz))))
```

20. Arrays and Matrices

Two concepts when dealing with arrays and matrices. First are they 0 or 1 based?

- 0-based arrays: C/C++, Python, IDL
- 1-based arrays: Fortran, matlab?

Secondly, are they row-major or column-major.

- 0-based arrays: C/C++, Python, IDL
- 1-based arrays: Fortran, R, matlab

21. Source Finding

Separating signal from noise is one of the most important first steps in the analysis of an image or a data cube. Many algorithms have been developed and published over the last 30 years. They basically can be classified in two main categories: thresholding and pattern matching.

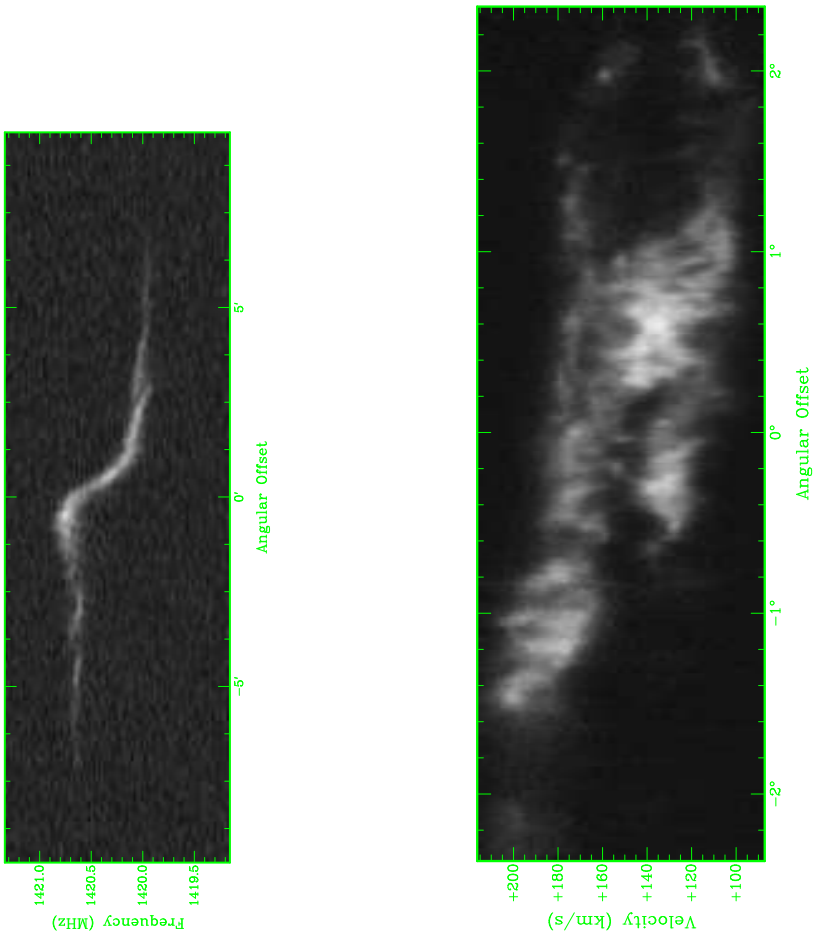

Fig. 1.— Position-Velocity diagram for NGC 6503 (left) and SMC (right)

Some recent reviews are available: Popping et al. 2012, Hassan (2012) Thesis Section 5.

Here we review some methods in common use with published papers. We are ignoring a very similar cottage industry in the N-body (points) world. A good recent review and code comparison by Knebe et al. (2013) can be found in `http://arxiv.org/abs/1304.0585`.

ASCL references listed below can be found off `http://ascl.net`, as well as in ADS `http://adsabs.harvard.edu/`. This is probably the easiest reference point to find everything related to the listed software.

21.1. clumpfind

The Williams et al. 1994 ClumpFind [7] algorithm has been implemented as `clfind` in miriad, but is also available in an IDL version. **ascl:1107.014** Now superseded by `cprops` (CloudProps, Rosolowski and Leroy 2006) **ascl:1102.012** [8].

21.2. duchamp / selavy

Whiting (2011,2012) for 2D and 3D. Used in ASKAP pipeline. `selavy` is the ASKAP 21-cm version. **ascl:1201.011** [9]

21.3. blobcat

Hales et al. 2012. 2D. **ascl:1208.009**

21.4. getsources

Menshchikov et al. 2012. Herschell based. 2D. [10]

[7] http://adsabs.harvard.edu/abs/1994ApJ...428..693W

[8] http://arxiv.org/abs/astro-ph/0601706/

[9] http://adsabs.harvard.edu/abs/2012MNRAS.421.3242W

[10] http://adsabs.harvard.edu/abs/2012A&A...542A..81M

21.5. gaussclumps

The original stutzki code is also available in GILDAS, as well as in STARLINK. It iteratively decomposes a 3D cube into a series of Gaussian shaped clumps. It starts by fitting a 3-dimensional clump locally to the maximum of the input lbv cube. It then subtracts this clump from the cube, creating a residual map, and then continues with the maximum of this residual map. The procedure is repeated until a stop criterion is met, for instance when the maximum of the residual maps drops below the 3 sigma level.

`gaussclumps` was developed by Stutzki & Guesten (1990, ApJ, 356, 513) who applied it to their M17SW data sets. It was later described and analyzed in more detail by Kramer et al. (1998, A&A, 329, 249) who also applied it to data cubes of various Galactic clouds. The papers of e.g. Heithausen et al. (1998) and Simon et al. (2000) give more examples of the application of gaussclumps to large scale CO maps.

21.6. imsad

`imsad` = Image Search and Destroy. Implemented in MIRIAD, based on `imfit`, esssentially fits a gaussian to a peak that sticks out of some plateau.

21.7. SExtractor

A widely source extractor (Bertin & Arnouts 1996), but not so good for blended objects. **ascl:1010.064**

21.8. SFind

Hopkins et al. 2002. Implemented in MIRIAD.

21.9. MultiFind

perl script using a series of miriad commands. Kilborn 2001. Written for HIPASS.

21.10. tophat

Meyer et al. 2004. For HIPASS. Uses a pattern matching of shapes and cross correlation to find signals.

21.11. lovedata

Saintonge 2007. For ALFALFA. Written in IDL. Pattern matching.

21.12. GammaFinder

Boyce 2003. Masters thesis Cardiff. In java. Use gamma function to estimate noise and reject non-signal

21.13. 1d-1d wavelet

C++ by FLoer and Winkel 2012.

21.14. smooth plus clip

An old method from the WSRT Bosma time, but more formally introduced by Serra et al. 2011. ATLAS3D.

21.15. CNHI

Jurek 2012. for WALLABY project design study. CNHI = Characterised Noise HI Source Finder. Pattern matching. Also mentions the Lutz (1980) one path algorithm, which is used to group regions from the same source into a single source.

21.16. DisPerSE

DisPerSE is open source software for the identification of persistent topological features such as peaks, voids, walls and in particular filamentary structures within noisy sampled distributions in 2D, 3D. Sousbie (2013) **ascl:1302.015** [11]

Using DisPerSE, structure identification can be achieved through the computation of the discrete Morse-Smale complex. The software can deal directly with noisy datasets via the concept of persistence (a measure of the robustness of topological features). Although developed for the study of the properties of filamentary structures in the cosmic web of galaxy distribution over large scales in the Universe, the present version is quite versatile and should be useful for any application where

[11] http://www2.iap.fr/users/sousbie/web/html/indexd41d.html

a robust structure identification is required, such as for segmentation or for studying the topology of sampled functions (for example, computing persistent Betti numbers). Currently, it can be applied can work indifferently on many kinds of cell complex (such as structured and unstructured grids, 2D manifolds embedded within a 3D space, discrete point samples using delaunay tesselation, and Healpix tesselations of the sphere). The only constraint is that the distribution must be defined over a manifold, possibly with boundaries.

21.17. FIVe

Hacar et al. (2013) discuss a new Friends of Friends (FOF) algorithm they dubbed `FIVe` (Friends in Velocity), and implemented in the R scripting language. [12]

21.18. Kemper

A multi-scale approach to automated feature extraction from HI data cubes. So far only published as a research note, Kempker (2005) from RUG.

21.19. PyFind

Serra (2012) at WSRT/ASTRON has developed this not-yet-named code. PyFind is his current placeholder name.

It is a simple Python code working on FITS files. It smooths the cube using a set of specified kernels and finds emission at each resolution by applying a specified threshold. The program applies also a size filter at each resolution and makes a final mask as the union of all masks. Two papers in 2012, one on MNRAS and one on PASA, discuss it.

21.20. GRID_core

Gong and Ostriker (2013), coming from a more theoretical angle and instead of working with densities, developed `GRID_core` that works with the gravitational potential. In essense this is simply a more smooth version of the density. It uses a watershed model (much like ClumpFind) to define clumpy structures. If the potential is not directly available, a poisson solver can be be used to define densities and with an assumed conversion factor, regions of bound structures can be defined as well. **ascl:1302.007**

[12] http://arxiv.org/abs/1303.2118

21.21. TODO

Here's some catchphrases that need to be tracked down if they have relevant code.

GAIA, MATADOR, AstroMed, SPLAT-VO, Herschel DP, S2Plot, AstroMD.