AIPS++ Note 242: Pixon imaging in AIPS++

Tim Cornwell, NRAO

16 March, 2001

Contents

1 Purpose

The purpose of this document is to detail the implementation of Pixon processing in AIPS++, and to explain how Pixon processing may be enabled.

2 Background

Pixon processing allows solution of various types of imaging problems, including image-plane convolution, synthesis measurements, and single dish measurements. The Pixon algorithm can be described as finding the smallest collection of blobs (Pixons) that fit the data for an image. This approach is robust, and produces excellent smoothing of noise, and featureless prediction residuals. The method was developed by Puetter and Pena in the early nineties. Fast algorithms have been developed by Amos Yahil in the last few years. The Pixon algorithm is patent pending. Pixon LLC makes IDL code implementing the algorithm available for scientific, non-commercial use. Each installation of the Pixon code must be obtained directly from Pixon LLC.

More information is available from the Pixon home page. A review of the Pixon approach is available in a recent ADASS paper by Puetter and Yahil The literature is as yet quite thin on Pixon-related papers but see *e.g.* Jenness et al. "Dual-Beam Rastering and Deconvolution Techniques for SCUBA".

3 Deployment in AIPS++

To simplify use of the Pixon algorithm in AIPS++, we have provided two main interfaces:

deconvolver deconvolver can apply the pixon algorithm to deconvolve a point spread function from an image. The noise level must be provided, either as an image or as a single number. The standard pixon text may be invoked as well.

imager imager can apply the pixon algorithm to image a region of the sky from either single dish or synthesis observations.

In addition, the standard pixon test case may be invoked from the test program in trial/implement/DataSampling/test/tPixonProcessor. This will start IDL, run the standard test, and display the images.

4 IDL

Since the algorithm is written in IDL code, IDL must be available for AIPS++ to be able to run the pixon code. The current AIPS++ code is known to work for IDL 5.3 and 5.4.

The location of the IDL pixon code and other public domain code must be specified in two environment variables:

```
IDL_PIXON_HOME=/home/pixon/oopixon/;export IDL_PIXON_HOME
IDL_PUBLIC_HOME=/home/pixon/public/;export IDL_PUBLIC_HOME
```

If you wish to evaluate the Pixon algorithm but do not have an IDL installation then you should be able to use an evaluation version of IDL quite satisfactorily. The only caveat is that no IDL session may last more than 7 minutes. This will enable you to run small test cases.

The current implementation uses IDL as an embedded application. In principle IDL can also be used as a Remote Procedure Call (RPC) server. This may be worth considering in the future for parallelization.

5 C++ design

The C++ design is quite straightforward. DataSampling classes convert specific types of data into the format expected for pixon processing, and the PixonProcessor class performs the pixon algorithm. An IDL class is responsible for starting, stopping IDL, for executing IDL commands (as strings), and for interchanging AIPS++ arrays with IDL. A PixonProcessor class makes an instance of IDL for its own use.

5.1 DataSampling classes

The base class is DataSampling. This provides standard access to the data needed by the PixonProcessor for the pixon algorithm. Derived classes fill in these arrays as appropriate on construction. Examples are ImageDataSampling (for image convolution), SDDataSampling (for single dish observations), and SynDataSampling (for synthesis observations).

The data prepared by the DataSampling class are:

Data a simple vector of the data points

Sigma a simple vector of expected sigmas for the data points

X a matrix giving the locations of the sample points in some normalized space

PRF a matrix giving the point response function

5.2 PixonProcessor class

The PixonProcessor class is responsible for:

- Initializing the embedded IDL
- Sending the data arrays from AIPS++ to IDL,

- Invoking the pixon algorithm in IDL
- Getting the resulting solution array from IDL into AIPS++.

5.3 makedefs additions

In AIPS++, the IDL-specific code is ignored unless the variable HAVE_IDL_LIB is defined in the makedefs. Hence to compile the code, you must add the following to your makedefs under Linux.

```
# IDL 5.* (needed for Pixons): note that the true Motif (iso LessTif)
# is needed.
IDLROOT  := /usr/local/rsi/idl
IDLDEFS  := -DHAVE_IDL_LIB
IDLINCD  := $(IDLROOT)/external
IDLLIBD  := $(IDLROOT)/bin/bin.linux.x86
IDLLIB   := -L$(IDLLIBD) -lidl -lMesaGLU -lMesaGL -ltermcap
```

For Solaris:

```
# IDL 5.* (needed for Pixons): note that the true Motif (iso LessTif)
# is needed.
IDLROOT  := /usr/local/rsi/idl
IDLDEFS  := -DHAVE_IDL_LIB
IDLINCD  := $(IDLROOT)/external
IDLLIBD  := $(IDLROOT)/bin/bin.sparc.solaris
IDLLIB   := -L$(IDLLIBD) -lidl -lMesaGLU -lMesaGL -ltermcap
```

Note that LessTif is not sufficient to allow IDL to be linked in. Instead a truly compatible Motif 2.0 library must be used. Under Linux, the best solution is to use OpenMotif which has an acceptable license. OpenMotif may be found at
http://www.opengroup.org/openmotif.

6 An example

In this example, we deconvolve an image that was convolved with a gaussian, and then Gaussian noise added. The glish script is:

```
include 'deconvolver.g';
dowait:=T;
```

```
d:=deconvolver('m31.dirty', 'm31.gpsf');
d.pixon(model='m31.pixon', sigma='0.007Jy', async=F);
im:=image('m31.pixon');im.statistics();im.view();
```

The AIPS++ message log contains:

```
Starting server deconvolver
Server started: /usr/home/tcornwel/aips++/linux_gnu/bin/deconvolver
(AIPS++ version: 1.5 (build #264))
Fitting PSF
Fitting to psf
Making Lattice convolver
Making Lattice cleaner
Peak of PSF = 1 at [129, 129]
Starting deconvolver::pixon
Deconvolving image using pixon algorithm
Calculating data sampling, etc.
Making pixon processor
Started IDL successfully
IDL :
!path=expand_path('+/usr/home/tcornwel/pixon/oopixon//')+':'+expand_path('+/usr/home/
IDL : @app_startup
Finding pixon solution
Defined IDL float array ADX [2, 65536]
Defined IDL float array APRF [256, 256]
Defined IDL float array ASigma [256, 256]
Defined IDL float array AData [256, 256]
Executing pixon estimation under IDL
IDL : @app_image
% Compiled module: IMAGE_PIXON.
AIPS++ image plane processing using Pixon LLC IDL library

Your Pixon license expires on January 1, 2002.
data have been plotted from low=-0.0305738 to high=0.730769

Fit/Pixel/Parm/DOF          0       65536      65536.0      0.00000

 Iter          Chi^2    Convergence             Q      Chi^2/DOF
    1   2.505800e+04   7.039018e+06           Inf            Inf
    2   2.505800e+04   7.216326e-04           Inf            Inf
```

```
Pixon count      2021.19

Fit/Pixel/Parm/DOF           0       65536      2021.18      63514.8

 Iter          Chi^2    Convergence              Q      Chi^2/DOF
    1   6.094254e+04   1.139337e+04  -7.217153e+00   9.595011e-01
% Program caused arithmetic error: Floating divide by 0
% Program caused arithmetic error: Floating underflow
data have been plotted from low=0.00000 to high=0.733239
% SAVE: Feature disabled for demo mode.
% Execution halted at:  $MAIN$
Copied IDL float array AModel
Pixon solution succeeded
Finished deconvolver::pixon
      20.99 real       18.18 user        1.27 system
Starting server app_image
Server started: /usr/home/aips++/version1p5/linux_gnu/bin/app_image
(AIPS++ version: 1.5 (build #266))
Starting image::statistics
Selected bounding box [1, 1] to [256, 256]
Creating new statistics storage lattice of shape [9]

Number points =   6.553600e+04       Sum      =   1.542824e+03
Mean          =   2.354162e-02
Variance      =   4.690163e-03       Sigma    =   6.848477e-02
Rms           =   7.241753e-02

Minimum value   0.000000e+00 at [180, 4] (23:59:55.770, +34.57.55.995)
Maximum value   7.332394e-01 at [122, 153] (00:00:00.488,
+35.00.25.000)

Finished image::statistics
       0.19 real        0.02 user        0.02 system
```

The model, corrupted image, and reconstruction are shown in figures 1, 2, and 3. The degrees of resolution improvement and noise suppression seen in this example are typical of those acheived using the pixon algorithm when the data properties are well-understood.

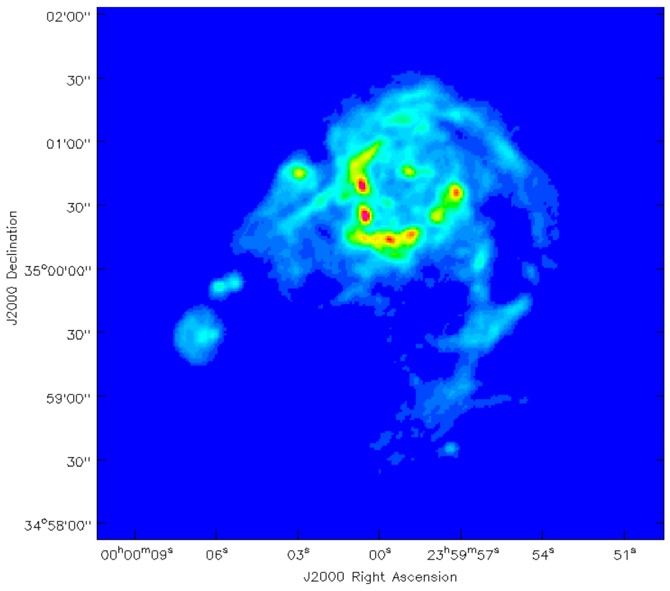

Figure 1: Model image

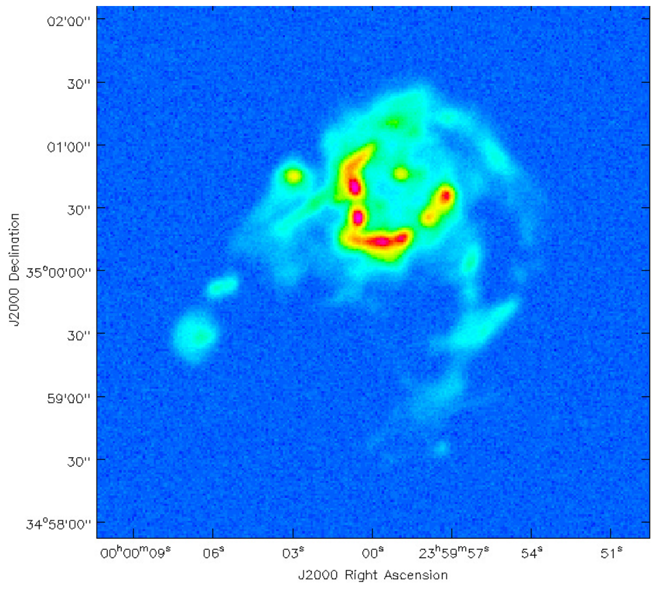

Figure 2: Dirty image: smoothed model, plus noise

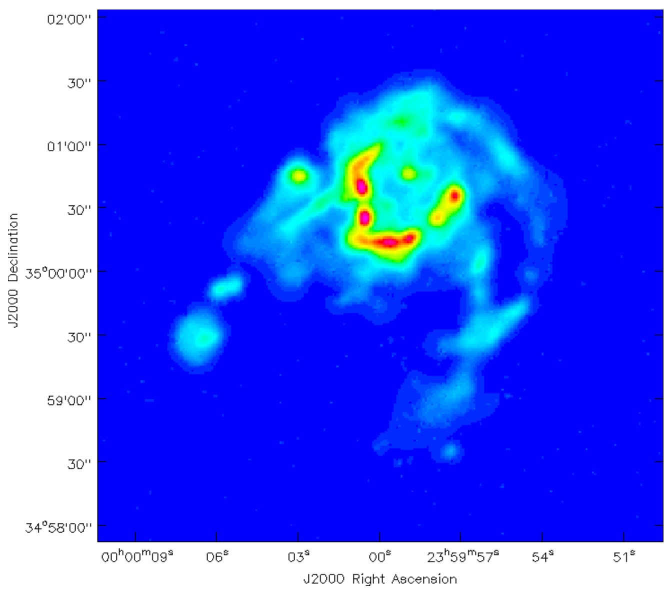

Figure 3: Pixon image

Acknowledgements

I thank Amos Yahil for many conversations, and Pixon LLC for making the IDL code available.