AIPS++ Newsletter

November 2000

AIPS++ Project News

Tim Cornwell- NRAO/Socorro

The third release of the AIPS++ package will be made on November 13 at the ADASS meeting in Boston. This release has many new features aimed at scientific completeness. The most notable improvement is the ability to process VLA observations entirely in AIPS++, from VLA archive tape to final deconvolved image. Advances have been made in many different areas of synthesis processing:

- Filling from a vla archive tape can be performed using the vlafiller tool.
- Flagging can be performed either interactively with visplot or in a batch mode using flagger.
- Calibration, including both flux scale and phase determination can be performed using the calibrater tool.
- Imaging, including many variants of deconvolution, can be performed using the imager tool.

In addition, numerous aspects of the user interface have been improved. We have been aided in this by a group of dedicated (and persistent!) testers in Socorro. The testers (Debra Shepherd, Greg Taylor, and Steve Myers) have worked closely with the AIPS++ group to improve all areas of the processing of VLA data. There still remains some work to be done in making the end-to-end path for VLA data as easy to use as possible. For example, an improved chapter for Getting Results in AIPS++ is in preparation. Another examples is that we have been experimenting with vertical integration of the synthesis tools into one tool called map. Improvements in these areas will be made available as patches to the new release as they become available.

The single dish processing environment, dish, has also seen significant improvements. Bob Garwood converted the data access mechanism (the "sditerator") to use a MeasurementSet in addition to the flat table previously used. In addition to spending eight weeks in National Guard training, Joe McMullin modified the interface of dish to be fundamentally command line based. A very detailed description of the use of dish is given in the Dish User's Manual (AIPS++ Note 236).

We have changed the format of the CDROM for this release. AIPS++ can no longer be run from the CDROM but instead must be installed to a disk. This has enabled the two distributions (Linux and Solaris) to be placed on one CDROM, saving considerable copying and distribution costs.

Finally, I would like to highlight some of the recent documents in the AIPS++ Notes series. The Notes series is used to document many different things. Designs of major packages are often written as a Note. The work planned for a development cycle is documented as a Note. Some recent examples are:

- In the development cycle leading to this third release, we completed a carefully planned change to a revised version of the observational data format used in AIPS++: the MeasurementSet. This second version of the format has been designed to rectify some shortcomings in the first version, but also to better support a wider range of observations, including single dish, VLBI, and optical interferometry. In putting together this definition, we benefited from extensive discussions with a number of people throughout the various astronomical communities. The definition of the MeasurementSet is available at AIPS++ Note 229, and a summary of the key changes can be found in the MeasurementSet Version 2.0 article in this Newsletter.

- The AIPS++ Measures system is a set of C++ classes and glish tools that handle measurements with units, coordinates, and reference frames. The Measures system is documented in the C++ header files (as is our standard practice) and in the User Reference Manual. A programmers guide to using the C++ classes is found in AIPS++ Note 233.
- Throughout its history, AIPS++ has benefited from a number of software engineering practices. Those interested in the type of practices that we have found to be useful may find a detailed description in AIPS++ Note 237.
- The Display Library is one of the most important packages within AIPS++. It is designed to provide display capabilities for all types of data within AIPS++. Originally designed and partially implemented by Tom Oosterloo and John Pixton, it has been filled out and extended by David Barnes. An excellent description of the design, implementation, and use of the Display Library is found in AIPS++ Note 231.

Tim Cornwell
November 2000

Mark Holdaway

AIPS++ Newsletter, November 2000

Science

CO in Serpens

Joseph McMullin - NRAO/Charlottesville

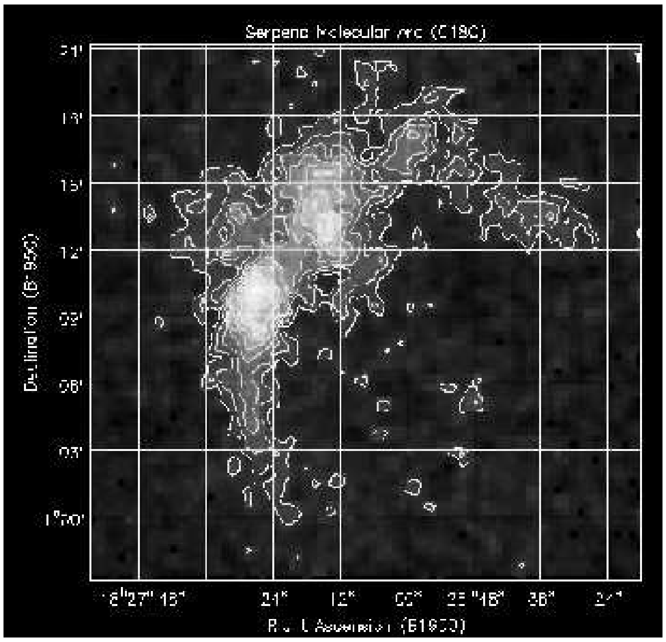

Figure S1: [Click on figure for higher resolution.]

Image of integrated $C^{18}O$ J=1-0 emission in Serpens.

The image shown is the nearby (distance=220 pc) stellar factory in Serpens as seen in its integrated $C^{18}O$ J=1-0 emission, as observed by the NRAO 12-meter telescope. This molecule traces the overall column density structure in the region. The backbone of the cloud is an arc curving through roughly 180 degrees, with a length of 30'. Two main peaks in the $C^{18}O$ emission are located near the center of the arc, separated by 4'. The Submillimeter and far infrared continuum emission from the region are dominated by sources that lie primarily *between* the two $C^{18}O$ peaks, including a cluster of 50-140 young stars. A first moment map of the $C^{18}O$ emission displays a broad velocity gradient running from a velocity centroid of 9.0 km/s at the northwest end to 7.5 km/s at the southeast end. It is unclear whether this gradient arises from rotation, two clouds at different velocities or interaction of cloud material with recent star formation.

In the paper on this source published earlier this year (McMullin et al. 2000, Ap. J., 536, 845), five different software packages were used to examine the array of data: 1) AIPS for reduction of NRAO

12-m OTF data, 2) CLASS and 3) COMB for analysis of single dish spectra from the CSO and NRAO 12-m, 4) MIRIAD for analysis of the BIMA interferometric data and 5) WIP for fashioning the final, publication-quality figures. While the calibration had been done, I threw in a sixth package, AIPS++, to explore better, more streamlined avenues for reducing a diverse set of data, which is typically the norm rather than the exception in observational astronomy.

Overall, I was pleased with the capabilities of the images and viewer modules in particular. It was gratifying to obtain the same answers as I had measured in MIRIAD (which I'm quite fond of). In addition, I found these tools better for exploring the data; interactively examining the structure on both the large and small scales is one of the more versatile tools handed to you in the AIPS++ GUI. The ability to script (or construct a command line record of your sundry GUI button clicks) was also a valuable addition for streamlining future reductions. In general it was my experience that the basic toolkit has grown to house quite an array of analysis tools, so it's more a question of finding the name of the task you need than whether you can perform an operation.

I look forward to completing my next data set as the previous battery of packages can now be replaced, from calibration->imaging->figure production, with AIPS++.

Using AIPS++ in the Research Experience for Teachers (RET) Program

Tim Spuck - Oil City Area Sr High School, Oil City, Pennsylvania

The HI luminosity-linewidth profile method is a valuable tool in radio astronomy. Among other things, it helps astronomers determine distances, rotation velocities, and hydrogen composition for distant galaxies. During my summer Research Experience for Teachers (RET), I worked with J. Richard Fisher at the National Radio Astronomy Observatory in Green Bank. This past October, Dr. Fisher used the Arecibo Radio Telescope to collect data for over 500 galaxies; my job was to reduce this data. The data helps to support research completed with the Infrared Astronomical Satellite and provide a comparison of luminosity-linewidth and supernovae Ia distance scales. In order to complete the task I had to first acquire a working knowledge of AIPS++ and Glish, the command and scripting language of AIPS++; Glish allows users easy access to all AIPS++ analysis routines and provides a means of writing customized applications by bundling together collections of commands into new functions. This past summer, Dr. Fisher and I wrote AIPS++ scripts that would implement data correction factors, display data scans, and assist in making precise measurements.

As a volunteer researcher, I plan to continue my work with this project during the coming school year. I will be installing AIPS++ on a Linux machine at my school in Oil City, PA using the latest version of AIPS++ on CD. Selected students will train and assist me in completing the data reduction process for these galaxies. In any science education program whether it be elementary school or college, it is crucial that students be provided the opportunity to experience science rather than to simply be told about it. If anything this summer, I've become even more convinced that science is active, not passive. Science is a vehicle by which man contributes to his understanding of the universe around him. Thanks to my RET experience, my students will be going on a little "drive" this year.

Multi-Scale Clean for Extended Structure

Jason Peterson - NMSU

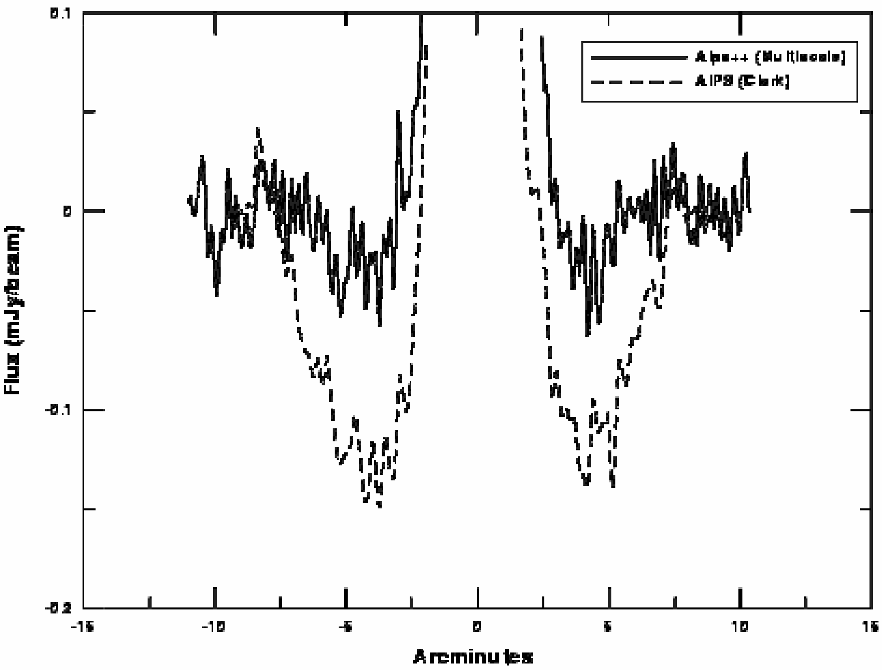

Figure S2:

A slice through NGC 4631's minor axis shows the relative performance of Multi-Scale Clean (solid line) and Clark Clean (dashed line).

We have used the Multi-Scale Clean algorithm in AIPS++ in an effort to characterize the faint emission from radio halos. The negative bowl caused by the missing short spacings problem can have a serious impact on the analysis of such emission. Figure S2 shows a minor axis profile across the egde-on galaxy NGC 4631 from VLA C-array data at 1.4 GHz. It is obvious from the plot that the Multi-Scale Clean algorithm is superior to the standard Clark Clean in compensating for the effects of the negative bowl surrounding the source. The benefits gained in the outer regions of the halo are not insignificant, amounting to an increase in scale height of the extended component by almost 25 percent over the Clark algorithm. The addition of D-array data substantially improves the situation by adding more short spacings, but Multi-Scale Clean is still a marked improvement over the Clark Clean, all but eliminating the negative bowl.

Backing into AIPS++ -- a Layman's Experience

F. J. Lockman - NRAO/Green Bank

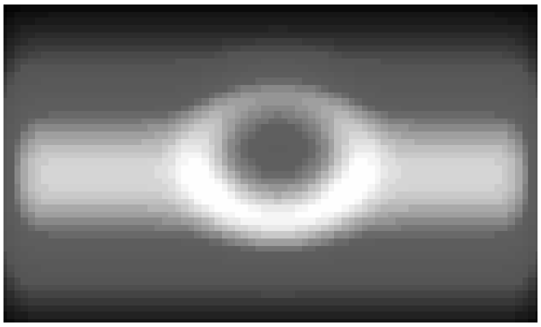

Figure S3:
Model of an HI cloud seen against background emission. The cloud has a cold core which appears in self-absorption and a hotter shell. The model was convolved to simulate observation with the 140 Foot Telescope.

Editor's note: This article traces the path of a non-programmer astronomer learning how to program in AIPS++ at the Glish level in order to perform customized scientific calculations. It frankly lays out both the frustrations one might expect to encounter on this path and the ease and power of Glish/AIPS++ programming once you have learned how to use these tools. The bottom line is best summarized by Jay's concluding remark: "I have found Glish/AIPS++ a major advance over other software tools that I have used and recommend it to anyone looking for a general purpose package for analysis of astronomical data."

Some months ago while trying to model a large HI cloud seen in 21cm emission and self-absorption against the galactic HI background, I was faced with the need to make some simple but rather extensive calculations. At Green Bank, I am surrounded by people using glish and developing AIPS++, so it was a natural temptation to play with these new tools for what I thought would be just a few day's worth of modelling. By the time the project was finished, months later, I had written several large glish scripts, each with hundreds of lines of code, and had analyzed several large spectral-line data sets using glish/AIPS++ functions. The results will appear in a forthcoming issue of the ApJ; here I have been asked to write about the experience and what lessons might be offered to others who would use the AIPS++ system as a general computing tool. I must warn readers that I have a very unclear understanding of how the tools I used fit into the greater scheme of AIPS++ and likely much of what I think I understand is incorrect. In particular, I will refer to glish and AIPS++ without a clear distinction between them because that is how I used them. I "backed into" AIPS++ by adopting a "bottom-up" approach, that is, by using glish in a primitive way, and only incorporating AIPS++ functions where necessary. But perhaps the lesson is that this system can be quite useful even if it is not understood at all.

I began the project in total ignorance of AIPS++ and glish, but armed with Rick Fisher's excellent Note Getting Started with Glish for AIPS++ and as important, with Rick himself just down the hall, within a short time I was manipulating arrays in the effortless way that glish encourages. In fact, if there is one major impression I have of working in glish, its that most things are either effortless or impossible. But more on this later. I was quite impressed on the ease with which simple radiative transfer problems could be coded in glish, and the results displayed using the built-in pgplot functions. It was a short step to incorporating AIPS++ functions in the code, creating AIPS++ images from arrays, convolving the images to mimic observations with a radio telescope, and so on. A few months later the results were in hand, and it had been a satisfactory experience. I now find myself beginning a totally different project, again using the Glish environment. It has become my default language for data analysis and modeling.

Here are a few general impressions derived from this experience:

- Glish is compact and powerful. With its basic functionality it is possible to do an enormous amount quite quickly. I was continually impressed with how straightforward and logical it was to do more or less exactly what I wanted. For example, given an array of spectral line measurements in various directions, one might want to form a smaller array of only the data with detections. It is simple in glish to say `detections:=positions[Peak_T>3.0*sigma]` and get an array of, well, of the good data! Rick Fisher's memo conveys this power quite well. When programming in glish, the AIPS++ functions are easily available and these make processes like convolution quite simple.
- The style of glish/AIPS++ is quite different from anything I had experienced before, and perhaps because I lack any understanding of the intellectual basis behind it, in places it seems somewhat weird. But it is, for the most part, consistent, and I was gratified to discover that once I learned a pattern it could be applied in many different circumstances. This is to say that glish/AIPS++ has a reasonably consistent set of rules.
- The online AIPS++ documentation is quite extensive though I found it almost uniformly obscure. Luckily, it usually contains enough examples to allow me to deduce the rules governing whatever function I need while sparing me the necessity of learning any comp-sci terminology.
- That glish is an interpreting language (rather than one which is compiled) has both advantages and disadvantages. On the plus side, I quickly developed the habit of trying out swatches of code in the interactive environment, testing to make sure that each variable had the correct value, then embedding the code into a function which operated in a sort-of batch mode during actual use. For most of what I did the loss of speed due to the continued interpretation of the glish scripts was not significant -- it takes so much time simply to get glish started on my workstation that I typically logged on to the fastest machine available and worked entirely there. However, if my programs get any more complex than the current ones, the speed of glish will probably become a factor, and I may need to optimize the run-time of certain small blocks of code.
- Errors: Whatever you do, don't make a mistake in Glish. It's error-reporting is crude and often less than helpful. For example, if you forget to supply all the parameters in a function call, it lists the entire function as part of the error message. Also, an error which involves an array can generate a number of error messages equal to the length of the array. I am making this sound worse that it actually is, for in a relatively short time I developed a bit of sense for deducing the true error from the Glish spasm. Still, some errors can still be maddeningly difficult to track down. Particularly insidious, though, are errors which glish does not consider to be errors at all. This is the topic of the next section.

  Editor's note: Glish code checked into the system is pretty good at handling errors. Because of Glish's flexible typing, a variable which usually has an array in it might be set to a "fail" value if an error was made, the `is_fail()` function can detect the failure of the array, and the glish script can orchestrate an appropriate response. More advanced user scripts should utilize this sort of error checking as well. On the topic of Glish's habit of printing out lots of stuff, this is often used as a "feature": for example, if you can't remember the name of some function of the `Table` tool, just typing the name of your Table tool will list all of the functions defined for Table. Similarly, typing the tabletoolname.functionname() will show you the parameters that function requires.

- Gotcha! Sometimes things didn't work out the way I expected, and this was typically the largest source of wasted time in my experience with glish. An example that I ran into just today occurred while I was trying to integrate a number of spectra over some minimum to maximum velocity, with the constraint that the minimum should be at least 100 km/s . Given an array of minimum velocities I wrote `min_vel:=max(min_vel,100)`. Well, the variable min_vel goes into this statement as an array but comes out as a scalar. This legerdemain is actually discussed in the glish documentation, but unless you are the person (I am told there is one) who reads all relevant documentation before beginning to write code, you might feel, as I do, that this is a fault rather than a virtue of the language.
- I/O. This can be quite clunky. You should see what I have had to go through to read the ASCII text output of a spread sheet into useful arrays in glish. It is my impression that glish is comfortable only with string I/O, which is ok if all you pass is messages, but less useful if data needs to come and go.

  Editor's note: while ASCII I/O is not one of Glish's strong points, the tablefromascii() and `toascii()` *Table functions are extremely useful for dealing with ASCII data. For this path to be fruitful, the casual user will need to learn how to manipulate AIPS++ Tables, but given the power of the Table system, the good documentation, and the "cut, paste, and test" aspects of Glish, this is not a great hurdle.*

- Bug reporting: Everyone should have the experience of submitting a formal bug report to the AIPS++ community, for it gives one the impression of having pulled a cosmic fire alarm. Email messages fly back and forth, progress reports appear, and I could not shake the feeling that somewhere, deep under a mountain in Colorado, lights on a gigantic panel had turned from green to red. Within a few days the bug was reported to be fixed. This experience is one which I can recommend quite highly.

If you are considering giving glish/AIPS++ a try, here are a few suggestions that may ease your way:
If you are really stuck, give up. In my experience, if you are having a problem it is a waste of time trying to understand exactly what you are doing wrong in some particularly tricky piece of code. My advice is to stop trying to debug. Rather, find a different way to write the code, perhaps less elegantly, and see if that works. You may be up against a hitherto unrealized feature of glish that might stump the experts for days, or more likely, something discussed in part of the Glish manual that you have not yet read.
Sometimes glish just needs to rest. Are you told that "no graphics device has been selected" or that you have "too many active plots"? Do you find a persistent, inexplicable, parse error when trying to include a function you wrote? Sometimes Glish just needs time off. Exit from Glish, count to ten, start it up again, and you may find your code working like a charm.
Don't hesitate to call for help. One of the most important features of Glish and AIPS++ is that NRAO is staffed with people who really want to make this system work for astronomers. They are eager and willing to help and you should take advantage of that. It is easy to criticize, so I do not want to leave an unfavorable impression of my experiences. Despite its current problems, many of which are due no doubt to teething pains, I have found Glish/AIPS++ a major advance over other software tools that I have used and recommend it to anyone looking for a general purpose package for analysis of astronomical data.

AIPS++ Newsletter, November 2000

Tools

Polarimetric Analysis of Images

Neil Killeen - ATNF

The Image tool is the basic AIPS++ tool with which you manipulate images. Working with it is the Imagepol tool; its job is dedicated polarimetric analysis of images.

Creating Imagepol tools

The **imagepol** tool is constructed from either an AIPS++ disk image file, or from an Image tool. This image must have a Stokes coordinate or an error will occur. The idea is that you will generally provide an Image holding RA, DEC, Frequency and Stokes coordinates.

Here are some examples:

```
- include 'imagepol.g'
- p1 := imagepol('pks1333.iquv')                    # 1
- im1 := image('pks1333.iquv')                      # 2
- p2 := imagepol(im1)                               # 3
- p2.summary()                                      # 4
```

The first example makes the Imagepol tool directly from a disk file which has Stokes I, Q, U and V. The second line makes an Image tool from the same file and then we construct the Imagepol tool from that Image tool. We use the **summary** function of the Imagepol tool to summarise the image (it's just the Image tool function **summary** that is really invoked).

What can I do with it?

So far we have seen how to construct Imagepol tools, and the use of the summary function. Let us assume from now on that we constructed an Imagepol tool from an image with RA,DEC, Frequency (many channels), and Stokes (IQUV say)

```
- include 'imagepol.g'
- p := imagepol('pks1333.iquv')
```

Recovering specific Stokes

You can recover a particular Stokes Image :

```
- i := p.stokesi()           # Make Image tools
- v := p.stokesv()
- q := p.stokes('q')
-
```

```
- i.statistics()           # Run Image tool functions
- v.view()
- i2 := i.subimage('pks1333.i')  # Copy to disk file
-
- imagedones()             # Destroy all Image tools when fed up
```

You can use either the dedicated functions such as **stokesi, stokesq** or use the **stokes** function which takes an argument saying which Stokes you want.

Note that the variables **i, q** and **v** are in fact Image tools. These Image tools are actually 'virtual reference images' (see the August 2000 Newsletter); they reference bits of the image from which the Imagepol tool was originally constructed. You can use all of the Image tool functions on them as usual (a couple of examples are shown).

Generating Secondary Images

You can generate all of the usual secondary polarimetric analysis secondary images such as polarized intensity, linearly polarized intensity position angle and so on. There are dedicated functions and a generic one **pol** to which you supply an argument saying which quantity you want (useful for scripts). Here are a few examples.

```
- lpi := p.linpolint()           # linearly polarized intensity
- lpa := p.linpolposang()        # linearly polarized position angle
- tpi := p.totpolint)()          # total polarized intensity
- flp := p.fraclinpol()          # fractional linear polarization
- ftp := p.fractotpol()          # fractional total polarization
- lpi2 := p.pol('lpi')           # Specify which polarized quantity
-
- lpi.statistics()               # Run Image tool function
```

Again all of the returned variables are virtual Image tools upon which you can run all the **Image** tool functions.

Generating Error Images

You can also recover the error images assuming simplistic propagation of Gaussian errors. In AIPS++ we do not (yet ?) associate an error value with each image pixel value and propagate errors automatically (very hard problem...). The errors are stored in their own image as needed. The standard deviation of the thermal noise is either worked out for you (with a clipped-mean algorithm) from the data, or you can supply it if you know it better.

Some of the functions return a scalar (as the error is constant over the image), some return an Image tool.

```
- sig := p.sigma(clip=5)          # A scalar; best guess at thermal noise
- sigi := p.sigmastokes('i')      # A scalar; standard deviation of noise
- sigi := p.sigmastokesi();
- sigv := p.sigmastokesv();       # A scalar
-
- siglpi := p.sigmalinpolint()    # A scalar; linearly pol intensity
- siglpa := p.sigmalinpolposang() # An Image tool;
                                  # linearly polarized position angle
- sigflp := p.sigmafraclinpol()   # An Image tool; fractional linear pol
```

The philosophy behind the handling of errors is to defer it as long as possible. Many of you are probably familiar with AIPS and Miriad where we blank output images based on a variety of statistical tests. For example, you might blank the linear polarization position angle image when its error is greater than some value. Doing that means you have to keep on regenerating the output image every time you want to try a different blanking value or method.

So to avoid this we have tried to defer this step until you actually use the images. For example

```
- lpa := p.linpolposang()            # linearly polarized position angle
- siglpa := p.sigmalinpolposang()    # and its error image
-
- lpa.statistics(mask='$siglpa<10')  # Blank when error > 10 degrees
- lpa.view(mask='$siglpa<20')        # Blank when error > 20 degrees
```

You can see we have used the **mask** argument of the **statistics** and **view** functions to do this on-the-fly. This is very convenient. Note that we have used the $ syntax (e.g. mask='$siglpa < 20') because the error Image tools are virtual (there is no disk file associated with them); the only way to get at their values is via the tool.

You can of course always copy virtual images to disk if you wish:

```
- siglpa2 := siglpa.subimage('lpa')
- siglpa2.done()
```

Finding the Rotation Measure

Now we come to the vexatious question of the Rotation Measure.

The Imagepol tool offers you two algorithms. The first is a 'traditional' approach of fitting position angle as a function of frequency. The algorithm used is that of Leahy et al (Astronomy & Astrophysics, 156, 234).

Recall that our Imagepol tool was constructed from an image holding (probably) RA, DEC, Frequency, and Stokes coordinates.

```
- ok := p.rm(rm='rm', rmerr='rmerr', rmmax=800, maxpaerr=10)
```

For each spatial pixel, a spectrum (frequency) of position angles is computed (from Q and U stored in the image) and fit. It is very important to specify the **rmmax** for this algorithm. All RM algorithms of this nature struggle with the n-pi ambiguity and this argument allows you to constrain the process of handling it in some vaguely useful way. We have also specified here a maximum allowed error in the position angle; otherwise that frequency is rejected for that spatial pixel. We output two images, the RM image and its error image. Note that these are disk image files, not virtual images.

```
- ok := p.rm(rm='rm', rmerr='rmerr', rmmax=800, maxpaerr=10)
```

The Imagepol tool also offers a new and novel Fourier-based algorithm (Killeen, Fluke, Zhao and Ekers, 2000, submitted). This algorithm generates an output image which is the polarized intensity as a function of Rotation Measure. It does not suffer from ambiguity and it can recover multiple RM screens unresolved by the beam (as long as they are not along the same line-of-sight).

```
- p := imagepoltestimage(outfile='iquv.im', rm="1e5 5e5", nx=8,
+                        ny=8, nf=256, f0=1.4e9, bw=8e6)
- p.frm(amp='amp', pa='pa')   # Fourier transform;
- amp := image('amp')         # Look at polarized intensity
- amp.view()                  # And reorder to put RM along X-axis
```

In this example, we have used the other Imagepol constructor **imagepoltestimage** to generate a test image with the specified Rotation Measure values. The generated test image is 4-dimensional (RA/DEC; 8 by 8 pixels), Stokes (4 pixels; IQUV) and Frequency (256 pixels). The source is just a constant I (if you don't add noise all spatial pixels will be identical) and V. Q and U vary with frequency according to the given Rotation Measures.

The Fourier algorithm Fourier transforms the Complex polarization P = Q + iU image as a function of frequency to a function of Rotation Measure. In the example we save images of the polarized intensity and position angle.

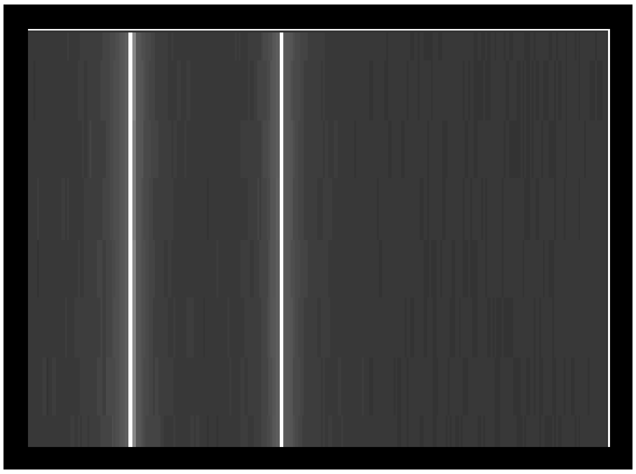

Figure T1: [Click on figure for higher resolution.]
The figure shows the result of viewing the polarized intensity where we have reordered the display to put Rotation Measure along the first axis. You can see the two RM components recovered; the one to the left is the larger one.

What's missing

Presently there is no specialized display for polarimetric quantities. For example, the traditional vector overlay of linearly polarization. This will be developed for release 1.5. One can also imagine doing interesting things with Complex representations of polarimetric quantities. The Image tool does not support Complex images yet (although of course we can create them and write them to disk with C++ code) and this path is also not yet well persued.

Functions for depolarization ratio and errors will also be in place for release 1.5

The Mosaicwizard in **AIPS++**

Mark Holdaway - NRAO/Tucson

Glish, the scripting language and command interpreter for AIPS++, allows the astronomer to have a great deal of control over a series of complex processing steps, running tools and automating decision-making based on the tools' output. But glish also permits astronomical programmers to bundle up their understanding of complex data processing into simple scripts or wizards which are very easy for non-experts to use. Mosaicwizard and imagerwizard, included in release 1.4 of AIPS++, are good examples of the simplicity which can be achieved using glish in this manner.

To start the mosaicwizard, start up AIPS++ and type at the command line:

```
- include 'mosaicwizard.g';
- mosaicwizard();
```

Two new windows will appear, one for the mosaicwizard GUI, and one for the scripter, which displays commands emitted by the mosaicwizard.

The mosaicwizard GUI guides you through the mosaicing process, asking for any required information and making reasonable defaults whenever possible. Each step of the process has a title (such as "Select an AIPS++ MeasurementSet or UVFITS file"), along with a few sentences that explain in more depth about what is required and what will be happening. Below the explanatory message there will usually be one or more widgets to accept your input. After you have given your input, the green "Next" button moves you along to the next step. The status line near the bottom of the GUI tells you brief messages about what is going on, such as problems with any input you've given the wizard, or whether an image file has been created or not.

The first step of the mosaicwizard is to select a MeasurementSet or UVFITS file to process. If you don't have one, just leave the input field , and by default AIPS++ will create a MeasurementSet of a seven-pointing VLA D array observation of Cas A at 8 GHz. If you start from a FITS file, as in the case of the Cas A default, handling MeasurementSet arcana will add some extra processing time to a few of the initial steps, so you probably want to keep the MeasurementSet and use it directly in subsequent work with the mosaicwizard. If you already have MeasurementSets in the current directory, click on the wrench icon to the right of the input field, and a file catalog will show you all available MeasurementSets. You can then select a MeasurementSet and click on "send & dismiss" to send the selected MeasurementSet to the mosaicwizard GUI.

The next steps include reading in an optional image that can be used as an initial starting model, selecting the spectral window to image, and selecting the fields you wish to image. after completing this input, you are ready to start thinking about how we want to make our mosaic image.

Because mosaics often have several pointings, cover a large angle on the sky and have many pixels in their final images, and they often reproduce extended structure which takes a long time to deconvolve, mosaics are famous for being slow. However, if you make a small mosaic at low resolution, they aren't slow at all. By default, the mosaicwizard will first make a low resolution image with pixels 4 times larger than full resolution, using only the inner 0.25 of the (u,v) plane. The low resolution mosaic is followed by an intermediate resolution image with pixels 2 times larger than full resolution, using only the inner 0.5 of the (u,v) plane. Finally, a full resolution image is made using all of the (u,v) data. In most cases, the low or intermediate resolution image can be used as a starting model for the next higher resolution image, hence saving on deconvolution time: the high resolution image doesn’t need to deconvolve all the flux from scratch, it just needs to incrementally fill in the fine details that were missing from the low resolution model. The first input on the deconvolution page, "(u,v) scaling parameter", controls the maximum baseline used, as a fraction of the maximum baseline present in the MeasurementSet.

In addition to the (u,v) scaling, you can specify the visibility weighting (by the way, when the (u,v) scaling is less than 1.0, a Gaussian taper is also applied in the (u,v) plane so that there is no sharp cutoff in the Fourier plane coverage), the number of Clean or Mem iterations, the deconvolution algorithm (choose between the mfclark, mfhogbom, and mfmultiscale Clean algorithms, or the mfemptiness or mfentropy MEM algorithms), and the presence of a progress display. The multiscale and MEM algorithms require additional input on a supplemental page.

Note that nowhere does the user input the image cell size or the image size. The cell size is set by the reciprocal of the maximum baseline length used for each resolutions’ image (divided by 3 for super Nyquist sampling). Initially, the image size is set so that it covers all selected pointings plus half a primary beam’s worth of slack on all sides. In later stages, or at higher resolution, the user is invited to generate a mask region which will determine the image size.

While the low resolution deconvolution is progressing, you may want to focus your attention on the mosaicwizard scripter window showing the most important mosaicing commands, the AIPS++ Log Message window, and the PGPlotter window (only if display progress was invoked) which shows the peak image residual and deconvolved flux as a function of iteration number. Because AIPS++ deconvolves the entire mosaic image with a single approximate PSF, the multifield (ie, mf) algorithms are implemented with a major/minor cycle framework: deconvolution can only proceed so deeply until errors will be made due to the differences between the approximate PSF and the actual PSF for each field. At the end of each cycle, a User Choice window will ask you if you want to proceed with the deconvolution. (If you don’t choose after 30 seconds, the default is to continue.) At the start of the next major cycle, the residuals will likely be a bit higher due to errors made from cleaning with an inexact PSF, but these errors are corrected in just a few iterations.

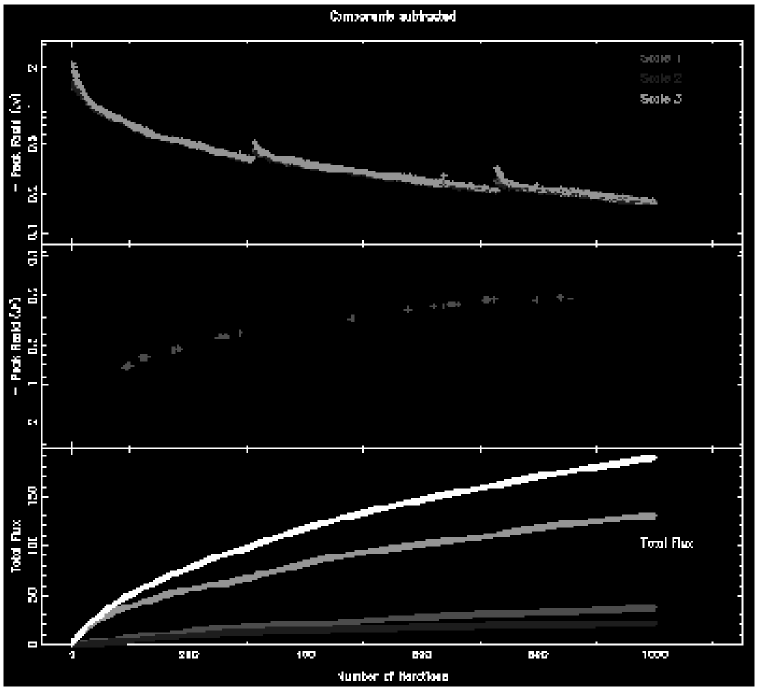

Figure T2: [Click on figure for higher resolution.]
The progress display in imager shows algorithm-specific indicators of the deconvolution progress. For Multi-Scale Clean, the three panels are the peak positive residual, the peak negative residual, and the total flux as a function of iteration number. The different scales' data are color coded. Each major cycle can clearly be seen by its early correction of errors made in the previous cycle.

When deconvolution ends (by manually stopping at a major cycle, or by achieving the requested number of iterations), the image is displayed with the viewer, along with instructions for how to create an optional rectangular or polygonal region. This mask region will determine the angular size of the next higher resolution image and will also restrict modeled emission to reside within that region.

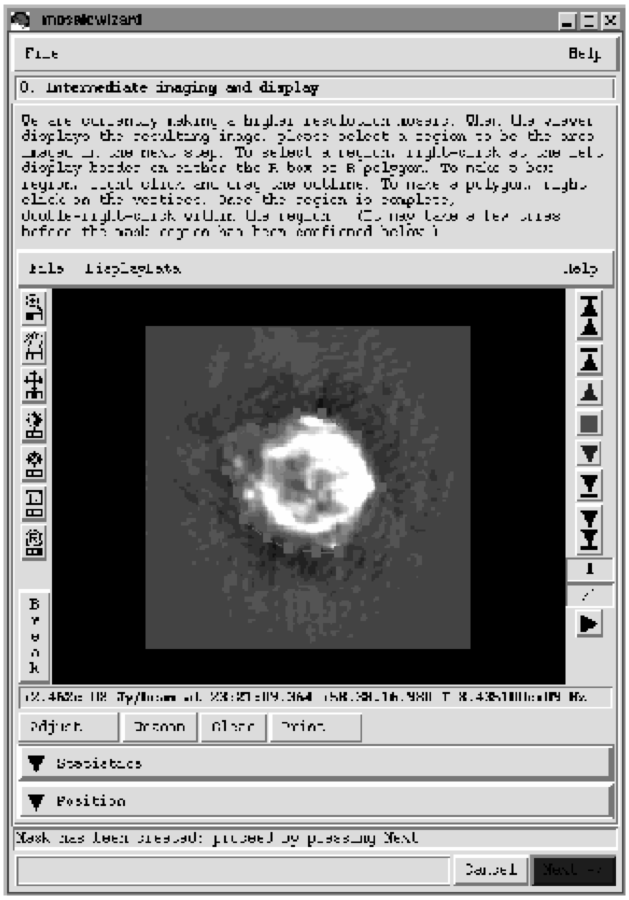

Figure T3: [Click on figure for higher resolution.]
When deconvolution is complete, the restored low resolution image is displayed by the viewer. You have the option of defining a rectangular or polygonal mask for the next stage of deconvolution at high resolution.

When you click on "Next" to leave the viewer, the mosaicwizard leads you back to set the deconvolution parameters once again. The (u,v) scaling has automatically increased by a factor of 2. After you adjust the deconvolution parameters to your liking, you are off making the intermediate resolution image, then inspecting that image with the viewer, perhaps prescribing a new mask. And then on to the full resolution mosaic, and you are finished.

There is nothing special about increasing the resolution in steps of 2. You can jump the (u,v) scaling however you want. You could, of course, make your initial image at full resolution, but I prefer to make my mistakes at low resolution where they aren't quite so glaringly obvious and they don't take so much time.

We are planning a number of enhancements to the mosaicwizard, such as the addition of total power data, dealing with problematic point sources, and handling spectral line data. Look for a more capable version of the mosaicwizard in the release 1.5 of AIPS++.

For more detailed information about the mosaicwizard, see the reference manual under Synthesis:imager:mosaicwizard. For more detailed information about multifield imaging in AIPS++, see the multifield chapter of getting results.

Synthesis III: Visibility plane calibration components

A. J. Kemball - NRAO/Socorro

The previous article in this series provided a description of the Measurement Equation (ME), which is the calibration formalism used in AIPS++. As described in that article, the ME allows antenna-based uv- and image-plane calibration terms, as well as additive and multiplicative baseline-based corrections. This article considers the uv-plane calibration components in more detail for the case of synthesis reduction, and their identification with well-known physical effects.

The antenna-based uv-plane calibration components are represented in the ME by [2,2] Jones matrices, J^{vis}_i, in a selected polarization representation, using either a circular or linear polarization decomposition. The previous article described how these calibration components are applied in the ME formalism. Further information regarding this formalism can be found in Cornwell (AIPS++ Note 183) and Noordam (AIPS++ Note 182). We enumerate the different types of J^{vis}_i components here:

- G_i: This diagonal term represents the composite complex electronic gain for all components located after the feed in the signal path.
- B_i: This term denotes the bandpass response, and allows a dependence on frequency in the complex gain terms.
- D_i: This term represents the instrumental polarization response of the feed.
- T_i: This term denotes atmospheric corrections.

The terms listed here can be solved for; there are also two terms of this class in the ME which are either known a priori or pre-computed, namely: C_i, the polarization configuration matrix, which performs polarization conversion operations, and the parallactic angle correction P_i. Note that a subset of the solvable, and non-solvable terms may also may a dependence on angular position, and have image-plane counterparts in the ME to reflect this. The image-plane effects will be the subject of a future article in this series.

The letter designation [G, B, D, T] is used to identify and select calibration component types in the user interface to the imaging and calibration software in AIPS++. The two AIPS++ tools most directly concerned with calibration and imaging are `calibrater` and `imager`. The use of these tools is fully described in the AIPS++ cookbook, particularly in the chapters on synthesis calibration and imaging. These chapters detail all the steps required to calibrate and image uv-data, and the associated `calibrater` and `imager` tool functions required.

An important point to note is that the tools `calibrater` and `imager` represent the uv- and image-plane sides of the Measurement Equation respectively. Therefore `imager` offers functions concerned with imaging, image-plane calibration components (such as the primary beam correction) and deconvolution, while `calibrater` is concerned with the application and solution for uv-plane calibration components, as enumerated above. The tools communicate via the **MODEL_DATA** and **CORRECTED_DATA** columns in the Measurement Set. The `imager` tool fills the **MODEL_DATA** column by integrating across the ME from right to left, starting from the source model or image. For different ways in which to do this see `imager.setjy()` and `imager.ft()`. The `calibrater` tool fills the **CORRECTED_DATA** column by integrating across the ME from left to right, and applying the known calibration components to the observed data in this process. The associated tool functions for the latter process are `calibrater.setapply()` and `calibrater.correct()`.

The solvers can form chi squared at any point in the ME at which a calibration component is being solved for; the solvers will be the subject of the next article in this series.

GBT's Observer Interface Written in Glish

Rick Fisher - NRAO/Green Bank

Several years ago I threw together a prototype graphical interface for the GBT specifically tailored to observers' control of the telescope. This prototype was written in glish/Tk plus a bit of C++, mainly because I was familiar with glish from early aips++ experience. Glish does not benefit from the enormous user community of a language and toolbox like Tcl/Tk, but this does not appear to have been a big disadvantage. Much of the GUI design involved detailed creation of widget combinations at a fairly low level to make optimum use of screen space and quite a bit of widget interconnection to make the GUI respond to interdependent parameters in a way that might be expected by an observer.

The GBT observer interface prototype seemed to fit the requirements of observer's well enough to warrant writing the real interface, now called "GBT Observe," in glish/Tk after a redesign of the code structure. The observer interface now consists of a simulator for each GBT hardware device, written entirely in glish/Tk, a parameter interface to the GBT's monitor and control software, a GUI for interactive observing, and a basic telescope control scripting syntax for programmed observing. The interactive and programmed modes of observing are integrated so that one can use one or the other as appropriate at any given moment. The observing script syntax parser has been written by Darrell Schiebel in C++, and a few utilities and an interface to the JPL solar system ephemeris have also been written in C++. Nearly all of the other code is in glish simply because code development is easier in the interpreted language, and computation efficiency has seldom been an issue.

Telescope observing procedures, from simple tracking and on-offs to raster maps, are written in glish and built into the observer interface. These should suffice for the great majority of GBT observing, but an observer is free to write his or her own procedures using the templates provided. The scripting syntax for programmed observing intentionally excludes arithmetic operators and most flow control to keep the syntax simple. The more powerful language features needed for procedure writing are already available in glish, which most observers will know from their use of aips++.

GBT Observe is still under construction, mostly adding interfaces to the many hardware modules in the system. The core interface to the antenna and a couple of back-ends has been in use for about a year on the GBT mockup in the Jansky lab. A few key interfaces need to be completed in time for GBT commissioning, and detailed control of some of the hardware not normally set up directly by observers will be added as time permits.

A few web documents describing the basic layout from the observer's point of view and the scripting language syntax may be found at

- GBT Observer Interface Prototype
- A Very Brief Introduction to GBT Observing
- GBT Observing Tables
- Writing GBT Procedures in Glish
- Advanced GBT Procedure Writing in Glish

More complete observer documentation is being adapted for the main GBT web pages.

A bit of programmer's documentation on the layout of the code directories and the details of adding a hardware device interface to GBT Observe is posted at

- Adding a device to GBT Observe (programmer's reference)

Since a GBT simulator is part of the observer's interface it should be possible to export the entire interface to a user's home computer for testing observing scripts and user-written procedures before using them on the telescope. We still need to work out the details of exporting the associated C++ glish clients. One way might be to make the interface code part of the aips++ distribution.

Mark Holdaway

AIPS++ Newsletter, November 2000

Infrastructure

The MeasurementSet, Version 2.0

Athol Kemball - NRAO/Socorro

In AIPS++, observed data from both interferometers and single-dish telescopes are stored in a common data format, referred to as a Measurement Set (MS). The data format is designed to allow data to be recorded in a generic format that is not instrument-specific, but extensible to include local data required for individual instruments. The MS format is an important counterpart to the generic calibration formalism, known as the Measurement Equation (Hamaker, Bregman and Sault 1996; A&AS 137, 117), which underpins calibration and imaging in AIPS++, and is similarly designed to be instrument-independent.

Measurement Sets in AIPS++ are implemented as AIPS++ Tables, and are fully visible to scientific users of the package using the table and tablebrowser tools. Additional tools such as ms and visplot allow higher-level access to the data stored in an MS.

For the past several years, AIPS++ has used MS version 1, as defined in AIPS++ Note 191. This format has served the project well, but over time needed to be extended to provide additional scientific features. The discussion of possible revisions to the MS v1 format took place within the AIPS++ project as a whole, and included significant contributions from throughout the project. The resulting MS v2 format was formally adopted by the project at the end of this process, and is defined in AIPS++ Note 229. This is the MS data format expected by AIPS++ applications from release v1.5 (November 2000) onwards. A conversion tool, **ms1toms2** has been provided to allow conversion of existing MSv1 datasets. These can also be filled from FITS or other data formats into MSv2 directly.

The highlights of the new data format include:

- Closer integration of single-dish and interferometer data, including support for generalized back-end devices, loads and reference signals, and scheduling heuristics.
- Optional support for both lag-spectra and triple-product data in the main table.
- Expanded VLBI support, including correlator models and frames, pulsar gating, frequency groups and space VLBI.
- A more generalized flagging model, allowing both command- and row-based flags.
- Infrastructure to allow diverse observations and sub-arrays to be combined flexibly.
- Separation of antenna pointing, source position and field center, and polynomial representation of these positions for near-field or moving objects.
- Expanded representation of history data.
- Generalized representation of Doppler tracking, covering single-dish and interferometry Doppler tracking modes.
- Additions to the system calibration and weather tables.
- Expanded representation of polarization and frequency information.

The **AIPS++** Data Repository

Darrell Schiebel - NRAO/Charlottesville

Release 1.4 will be the first release to contain the new AIPS++ data repository. This repository was developed to move the data elements out of the source tree. By moving the data which change slowly out of the source tree, the download time for each AIPS++ build is lessened, and larger data sets can be used and distributed.

CVSup was chosen for use with the data repository for several reasons:

1. it does rsync updates of large binary files, which allows only the modified tail-end of binary files to be shipped and then patched onto the existing older version of the files.
2. it has optimizations for dealing with RCS and CVS repositories which permit it to minimize traffic.
3. it is used by FreeBSD so it has a large body of users who maintain it and ensure that any security problems are handled.
4. once the specification file is created, it is easy for users to keep the data repository up to date and to monitor the updates as they happen (we plan to include CVSup binaries with the release).

As part or the installation process for release 1.4, the repository specification file will be set up to allow the user to update the repository. The way the update happens is that new data files are generated in Socorro based on updates from the data source, e.g. US Naval Observatory for earth orientation information. These updates are then made available for AIPS++ users. The user can then choose when to synchronize with the Socorro data repository.

The repository is divided into sections based on content:

- ephemerides
- geodetic
- catalogs
  - sources
  - lines
- demo
- *individual institutions*, e.g. NRAO, BIMA, etc.

The sections of the repository which are likely to be of most interest to end users are the source and spectral line **catalogs** (these are generally not in place yet) and the **demo** section which will contain test data and data used as part of AIPS++ demos and examples.

Our goal is to continue to expand the repository and in time to allow users to choose which pieces they wish to keep locally and synchronized, and which pieces should be accessed on-demand. One of the next things we plan to add shortly is the Poynter and Pickett Submillimeter, Millimeter, and Microwave Spectral Line Catalogue. This and other catalogs should make the repository useful not only for internal AIPS++ applications, but also for direct use by end users.

Mark Holdaway