A Single-Dish Data-Handling Environment Within *AIPS++*

HARVEY S. LISZT
Rational Radio Astronomy Observatory,
520 Edgemont Road, Charlottesville VA USA 22903-2475 *

January 14, 1992

1. Introduction

1.1. What This Document Is

I was asked to produce (quickly) some words which might serve as a jumping-off point for further discussion of the treatment of singledish (SD) data handling under the *AIPS++* umbrella. Those words exist somewhere between the larger memo by Hjellming *et al.* and the recent committee report of Maddalena *et al.* discussing online data handling at the GBT. That is, I have had in mind the various needs of the GBT (which nonetheless is usually not mentioned explicitly or afforded exclusive rights) but have happily adopted the tenets of *AIPS++*. I write in the belief that *AIPS++* will afford any ability for which SD might use AIPS even *now*, and that the requirements cited by Hjellming *et al.* will be part of the *AIPS++* design. This being the case, I need mostly only provide elaboration and detail work here. The reader is assumed to have some familiarity with the memo by Hjellming *et al.*.

1.2. Why incorporate singledish data-handling within *AIPS++*?

There are many reasons for this. NRAO needs *AIPS++* as a vehicle for coordinating and unifying software development; SD astronomers need better and more widely-distributed software; smooth integration of SD data-handling will improve *AIPS++* generally and substantially; there will be increasingly tight integration and sophisticated use of SD data with synthesis projects to capture short-spacing data.

1.3. About the rest of this memo

My words are arranged as follows. Section 2 deals with the various kinds of information that flow during observing and are joined to make a database; this is relevant to the online use of *AIPS++*. Section 3 speaks to the ways that the data should be viewed, filtered, and recreated on disk in order to move from the raw state to being calibrated and reduced. Section 4 details

* The Rational Radio Astronomy Observatory is operated by the Rational Science Foundation

some capabilities of the shell and command line interpreter, and samples some of the 1-dimensional operations which reduce data and extract information from it. Section 5 touches briefly on some display capabilities.

2. Making Data

2.1. INFORMATION FLOW

Several streams of information flow back to the astronomer during observing, tagged with the time (or some other means of achieving synchrony)

- slowly-varying configuration information–site latitude and telescope or baseline positions, operator's initials and program id.
- ambient conditions– things which are independent of the act of observing, and sensed but not influenced, such as ground temperature and humidity.
- explicitly commanded conditions–everything from the desired filter bank setup or receiver center frequency to telescope position and desired length of integration–as these are understood by the telescope's operating system (TOS).
- implicitly commanded conditions–the hardware is configured in response to the explicit commands, for example, to set center frequency with allowance for Doppler shifts, to track and point, to move surface panels, to start observing at a certain UT, to tune the receiver and so on.
- resultant astronomical data.

The possible set of implicitly commanded conditions will typically be much larger than those commanded explicitly, as even a simple order to the TOS to integrate for five more minutes requires the use of many cooperating pieces of hardware. Note that the explicitly commanded conditions must be regarded as apochryphal until they are checked against the actual observing record. Any telescope will have some safeguards–it shouldn't integrate if incorrectly pointed or configured–but none will be entirely failsafe.

In current TOS, these datastreams are handled in curious ways. Some pieces of information may be stored only on pen-recorders, while others appear but momentarily on CRT's. Those judged most important are written on log sheets or to disk. In most singledish TOS, the permanent disk record is a conflation of all the datastreams in a binary form determined by whatever is the native analysis system. This is written to tape, taken home by the observer, and erased from disk soon thereafter. The detailed, permanent records of what happened on-site are dispersed world-wide in observers' datatapes or simply lost.

2.2. When does information become data?

The non-astronomical datastreams may be as needy and deserving of analysis as the astronomical data itself; they are the first line of defense against corrupted or invalid data. Moreover, this flow of not-purely-astronomical information is the *chief* concern of those who build or maintain a telescope and its OS.

There has been a substantial discussion of the degree to which the GBT's monitor and control group must cooperate with the data analysis effort. While it might be argued that any clean interface between M&C and analysis will suffice, and that the analysis side could probably recover even if M&C were simply to dump an undifferentiated bitstream on disk, the mode of operation of a SD telescope has historically been the single biggest determinant of subsequent analysis procedures and software. Our analysis procedures and data structures have been more nearly attuned to the operation of the telescope than to underlying astronomical practices.

The online software system must be equally adept at addressing all the information which flows in. Archival storage will have to be extended to address the non-astronomical datastreams as well.

2.3. online *vs.* offline

It will be the job of *AIPS++* to make sense of the above-mentioned datastreams during testing and configuration of the telescope and TOS, and during and after observing. Historically, SD work has employed the same software at and away from the telescope, though (importantly) nearly every telescope is independently supported in software and few programs are widely distributed. Astronomically and analytically there is little difference in the abilities required of online and offline software even now. Any remaining distinctions will indeed fade under such influences as doing observing opportunistically and piecemeal to accomodate the weather, absentee and remote observing, and using more sophisticated hardware and TOS.

Of course dependence on software online carries an implicit committment that observing time will not be lost. When problems are present during observing they must be fixed expeditiously.

It will be a challenge to produce a system which satisfies the immediate needs of local testing and diagnosis as well as astronomical exploration, while maintaining portability, robustness, and immunity from low-level distortions– i.e. *hacking* in situ.

3. Creating and Maintaining a Useful SD Database

3.1. What is the atom of SD data?

The basic unit of SD data has been a 'scan', which, though universally recognized, is surprisingly ill-defined. Loosely speaking, a scan corresponds to what the telescope was doing within a brief interval while observing, ranging perhaps from 15 seconds to 5 minutes; a recognizable amount of data is tagged with a parameter group describing the conditions of observing and affording unique access (most often *via* a simple serially-ordered scan number). These descriptive data-associated parameters are grouped within the scan's so-called *header.*

In many cases, scans are a simple joining of a header block followed by a single array of data representing the output of one spectrometer or a single continuum drift scan. But scans may also be very complex. For instance, they might be subdivided in a dozen or more independent but similarly sampled subscans whose simple sum is the desired final product. The data may also be stored in long arrays which represent independent IF's (observations placed end-to-end). There is no generally-agreed limit of the amount of complexity which might be encompassed within a single scan, and it has been proposed that scans could hold maps or (with internal tables) irregular, sparsely-sampled or even redundant data.

Alternatively, in some current modes of observing necessary parts of the data are artificially *dispersed.* In doing total-power work, where on-and off-source observations are differenced, one might have a single 'off' followed by many 'on's'. Present-day NRAO software sometimes stores information in the header which identifies on *vs.* off-scans and records the on-source data without performing the necessary differencing. It uses separate commands to access switched and total power data and requires additional commands to achieve scaling to antenna temperature for the latter.

In discussions recently, the trend has been to break out independent arrays rather than conceal them within individual scans; data taken with different receivers and now stored as the A,B,C, and D quadrants of a single scan (one scan number) would become four scans or be explicitly recognizable through dotting (scan.quadrant or scan.quadrant.record).

3.2. Sometimes a cigar is just a cigar

One substantial difference between SD and SA (synthesis array) observing is that the former observes directly in the map plane; the telescope was probably pointed separately to get each line profile. SD astronomers get fewer profiles and pay more attention to each of them. They have not been able or required to deconvolve their observations; the number of FLOPS needed to reduce each pixel has been small. This will change, but it is crucial

that *AIPS++* afford the same intensity of focus on individual spectra which is the real hallmark of present SD analysis codes.

In my own code and most SD programs, a scan is kept simple and larger-scale structures, if they are ever explicitly referenced, are created only on-the-fly by the analysis program. This approach works well when data are sparse and the data-taking is haphazard, but it is hardly self-documenting. During observing, some regularity of the positional or other spacing of the instrumentation might lend itself to aggregating the data into larger structures like a cube; this kind of multiple-pointing single beam work happens now as the telescope is moved point by point over a sky grid. Increasingly however, multiple beams will be present, and these may take data at somewhat eccentric positions which are only later interpolated onto a regular sky grid.

Many interfaces remain to be defined for the GBT, including that one between the internal formats of Monitor & Control and the external world of scrutiny and analysis by astronomers. From the standpoint of analysis, what is needed is a scalable, self-documenting architecture which achieves flexibility without dependence (as now) on nuance and apochrypha, and without forcing the data into complex explicit structures when these are not needed. FITS offers a variety of possible implementations for SD databases, based either on scans as presently understood with superstructure imposed later, or on multidimensional data storage which explicitly groups related data into images, cubes, and so forth.

3.3. MANAGING A DATABASE OF SCANS &/OR OTHER SD DATA

Online we create some means of joining selected parts of the various dataflows into permanent structures which can be accessed and referenced as constituents of our users' database–scans, cubes, whatever. Analytically, and later, we need a means of associating and reordering the telescope's datastreams so that the time-ordered and record-based procedures of data-taking are replaced as required by (larger) groupings which make sense astronomically.

The views needed are sometimes more like a collapsible outline than placement on a regular lattice. For instance, one could ask for a list of all the source names, or positions, or frequencies observed. Once shown, the list could be traversed from top to bottom and in depth. That is, an entry at the highest level would indicate the presence of n subitems; it would then be possible to select out those subitems to receive the focus of the program, whence one group at a time could be processed until all are done. The same organization could be symbolized graphically by segregation into folders which are very conventional icons. Opening a folder could present the user with a summary of whatever pertinent information the system was

aware of; lists of scans, maps, tables of information, whatever.

In reading some of the other specification documents, I sense a conflict between the desire for such high-level organization and allowance for access to the full range of file-handling abilities of the computer operating system.

3.4. MOTION FROM RAW TO CALIBRATED AND REDUCED DATA

During this process, a variety of operations must be easily applied to entire datasets or to subsets of data selected flexibly and perhaps even somewhat fuzzily or using wildcards. This brings up the question of how a (SD) database is to be modified. When the data are very raw, it is simple to imagine that every member would be processed and a new file or version of the data would be created *in toto*. Later on in the processing, when it is apparent that only a (small) portion of the data needs further treatment, this is less obviously desirable. Rick Fisher has described handling data in chunks and it is quite often the case that a rather monolithic database is treated as, and reassembled from, portions which are described well by the idea of a chunk of data.

From a pedagogical point of view, it is better to recreate the entire database for each change than to overwrite portions of the data without recording sufficient information to be able to backtrack. From the point of view of system resources, the most efficient way to update a database might be to create a change log which is applied to the input data only after the fact.

This is a design decision; the intention here is to alert the designers to the possibility that many small modifications are typically made in SD work and would have been made more commonly in AIPS as well if more precise tools had been available.

4. Shell and Aspects of 1-D Processing

4.1. THE SHELL, IN GRAPHICAL AND COMMAND LINE MANIFESTATIONS

I assume that *AIPS++* will be programmable, accepting and executing user-defined scripts, macros (recorded as text or graphically), and procedures, and that it will be easy to link in object modules compiled from HLL. It must be easy to develop and test procedures interactively which are proved on (perhaps) small portions of the data and/or easily applied automatically to flexibly-selected chunks of data. The facilities of *AIPS++* should be available graphically and in command-line form, with both useable interactively and the latter present when *AIPS++* is used as an engine for less-interactive or unattended operation.

At the broadest level, the shell needs to have and permit easy access to data and data associated parameters, indexed where necessary: in access-

ing data, it may be necessary to use constructions like scan.quadrant.record as suggested above, whose elements would be scan.quadrant.record.channel. The shell should have a full complement of physical and other constants (pi, c, e, h, *etc.*). It must be able to calculate with 'header' variables or channel values and to use the results of such calculations for whatever purpose (examples: scaling all the values in a vector to its peak value; comparing the rms noise in a vector to the theoretical value corresponding to the radiometer equation). Wild cards–scan.4.*.12 = (scan.4.*.11+scan.4.*.ch13)/2– would also be recognized.

We also need the facility to create vectors in various ways, see just below, and the shell should deal with multi-dimensional arrays intelligently and intuitively. It should afford access to them in part or whole, without explicit recognition of indices where that can be suppressed (r=s*t where s and t are scalar or vector, depending on the type of r, perhaps). Otherwise, it will be too tedious to use the shell for sophisticated purposes and explicit procedures would be preferred.

4.2. What do I really mean by 1-D?

AIPS was particularly strong on coordinates; that should be extended. Sometimes an array is taken on a regular grid but is viewed transformed into an abcissa which is not evenly spaced. Viewed by velocity, examples are data taken with channels equally spaced in frequency at velocities not small compared to c, or data taken with equally spaced bins in wavelength. Alternatively, data are sometimes taken by AOS or other devices which employ non-constant spacing. *AIPS++* needs to be able to deal with such data by interpolating onto a regular grid if commanded, or (within limits) on the data's own, admittedly eccentric, terms if need be. This is implicit in the need to treat configuration and other not-purely-astronomical information, as discussed earlier.

4.3. An example one-dimensional data-handling toolkit

There will be (of course) be retrieval and archival and selection operators, (related to project and database management, of course, which is too big an area to be recapitulated here in its entirety)

- define portions of an array or chunks of data explicitly (alphamerically by setting ranges with words or expressions, or graphically by outlining portions of a TVWindow rectangle or polygonal outline)
- define portions of an array or chunks of data implicitly by setting up criteria for windowing and other machine-based pattern-recognition activities.
- focus the program on an atom or chunk of data

- store, restore, or replace an atom or chunk of data in the workspace or on 'disk'

There must be creation and destruction operators:

- create a new vector or workspace (which can be filled in arbitrarily)
- deallocate a vector or workspace

Some operations just take place channel by channel with only a little bit of (user or header) input, without reference to other vectors, and without explicit extraction of information:

- scale up or down by a multiplicative or additive constant
- invert left-right
- fold to undo effects of frequency switching in passband
- interpolate at same spacing as if LO/center velocity had been different (a fractional-channel barrel roll or left-right shift)
- regrid as if channel spacing had been (slightly) different originally
- replace by numerical derivative (used in Zeeman work)
- clip between limits along ordinate or abcissa
- pad out to greater length
- interpolate bad or missing data by channel or block of channels
- detect and filter spikes
- fft, filter the fft, and retransformation
- smooth by hanning; boxcar; Gaussian-convolution; user-defined filter
- replace channel values with any given alphamerically expressed function of the values &/or channel index, with explicit reference permitted to header or data values. This can be anything from a simple scaling up or down to conversion to/from opacity in emission/absorption or scaling to the highest channel or system temperature.

Some operations use an array to create another array by transformation or abstraction (fitting components) and do extract information in various forms:

- fitting for baselines: sine-wave, polynomial, etc.
- fitting for analysis: gaussians, sine-waves, parabolic shapes

Some operations only extract information:

- calculate various statistical properties such as mean, rms, and moments (or any user-defined quantity or function) over predefined regions
- window, based on threshold criteria, followed by the above.

Some operations merge/combine arrays:

- channel-by-channel arithmetic operations with two spectra (examples: apply a vector of corrections element by element multiplicatively or additively or subtract a baseline or gaussian fit) over all or part of a vector.

- join spectra end-end or insert one within another
- add two or more spectra according to very flexible criteria, with specified weighting

Some will communicate with the outside world:

- export a vector in spreadsheet, FITS, ASCII format.

Some operations will undo previous processing:

- undo the previous action (perhaps multiple)
- restore the original state at last disk/memory access

5. Display And Annotation Facilities

SD will profit from tighter coupling with *AIPS++*' ability to manipulate images in multiple dimensions. However, AIPS was weak in handling and display of 1-d vectors.

AIPS++ needs to be able to show vectors:

- individually, with error bars, as lines, histograms, bars, *etc.*
- superposed
- stacked vertically
- tiled across the page

In many cases, the displays used in SD work are sufficiently simple–spectra, overlays of two or more spectra or spectra and component fits–that it would be criminal not to create the display elegantly. This applies to diagrams like contours and when multiple plots are tiled on a single page as well (such things are easily encapsulated!). For much of the use to which SD and SA data-handling software is now put, the final product is easily displayed in such a way that further retouching by a draftsman should be unnecessary.

In a departure from current practice, the screen should not be the exclusive province of the program; rather, the user should be able to scribble on it in various ways, and have those scribblings appear on-screen or in hard copy when desired. An example is making a mark in one frame of a series of moment or channel maps so as to compare results in different frames. Another example would be simple annotation of a spectrum to point out this or that behaviour with text, boxes, lines, and arrows.

6. References

Hjellming, R. M. *et al.* 1991, *AIPS++ User Specifications: An Initial NRAO-Oriented Version*

Maddalena, R. M. *et al.* 1991, *Requirements for Data Analysis Software for the Green Bank Telescope*, GBT memo 72.