Astronomical Data Analysis Software and Systems VI
ASP Conference Series, Vol. 125, 1997
Gareth Hunt and H. E. Payne, eds.

A Home-Grown But Widely-Distributed Data Analysis System

H. S. Liszt

National Radio Astronomy Observatory, 520 Edgemont Road, Charlottesville VA, 22903-2475

Abstract. I stopped doing research during the five years 1986–1990, and crafted a series of programs for doing spectral line and image analysis on DOS-based IBM PCs. These programs, most notably drawspec and hazel, are now fairly widely distributed. With only a modicum of hard evidence, I attribute this to their ease of use, small hardware demands, lack of cost, and high level of support. Recognizing that my experience is unusual, and perhaps even of marginal relevance to the design of new programs by teams of expert professional programmers, I have attempted to distill from it some words of advice for the attendees of ADASS VI.

1. Introduction

After accepting the invitation to address the meeting, I began to wonder whether I could actually contribute to a gathering of professional astronomical programmers, my own experiences in programming being so bizarre and particular. For my talk I did not try to discuss in detail the analysis system from which the title was drawn, but rather focussed on how programming and computing look to me now that I am supposed to be able to tell others how to go about them.

At the end of 1985, NRAO turned off the IBM mainframe which, with its predecessors, had been the backbone of all Observatory computing. Although each internal computing review had based its recommendations on the continuing presence of such a machine—how else to preserve the vast library of PL/1 code we had amassed in support of single dish observing in Green Bank and Tucson?—synthesis computing needs had overtaken such concerns. The IBM was turned off, and its code libraries were abandoned without compunction or provision.

For pure numerics, I had always programmed in FORTRAN, but most of my effort was in a set of custom, PL/1, spectral line data-handling routines for VLA and single dish data, mostly to gain the use of record structures and the NRAO code libraries. Even if my library of data analysis routines was not rendered exactly obsolete—we had bought a PL/1 compiler for our VAX/780—the VAX was next to impossible to use. Bogged down by AIPS, addressed over internal serial links, with only a tiny fraction of a megabyte allotted per personal account, even editing a file was next to impossible. Still less disk space was available on our shiny new AIPS-machine, a Convex C-1.

Over Christmas vacation, just before the termination of the IBM, I read some data off a round tape on the VAX and ported it to the AT&T 6300 PC/XT-clone which the Observatory Director, Hein Hvatum, had ordered placed on my desk some months before. I had complained pointedly that each of our secretaries had a PC with more disk space (20 MB) than was alloted for the entire scientific staff, and Hein, in essence, dared me to come up with a reason why this should not continue to be the case. Having been warned away from C by one of the professional programmers in our building ("it's so ugly"), I had happily adopted Turbo Pascal as my PL/1 successor.

Having earlier figured out the 6300's non-standard 640×400 graphics scheme, and with my own nascent library of fast graphics routines, I devised a binary structure to hold some spectral line scans and proceeded to write a program to call up and draw on-screen spectra in histogram form. The speed with which this occurred, compared to all my past experience, was breathtaking and utterly seductive: in a few seconds I was made to understand that I had more computing power on my desk than was available at the NRAO to the entire community of spectral line observers, of which I was a member. The source file was drawspec.pas, a name which persists to this day. One minor change to the binary file format was made in 1986. In the early 1990s I added to drawspec the native ability to read FITS images and cubes.

Between 1986 and 1991, I all but abandoned my research career and coded a series of programs which deal with spectral line and image analysis. This was a passionate and costly exercise (I estimate that I spent over $10,000 in current dollars on two machines and sundry software) carried out largely in my basement in what otherwise would have been all my spare time. When I came to my senses in 1991 and began to rebuild my research career, a whole generation of graduate students was unfamiliar to me.

The drawspec program, comprised of 24,000 lines of code (with another few thousand lines of i86 and i87 assembler thrown in for good measure), is supplemented by scanmstr.exe, a 9,000 line utility program which imports and exports various other data formats and reproduces most of the functionality of the code libraries abandoned by NRAO in 1985. There is also 5,500-line hazel.exe (Harvey's AZ-EL program) which is used to view the sky and plan observing. Additionally, I wrote for myself a simple curve-plotting package (4,000 lines) which does the line and point graphics not available in drawspec. The first three of these programs are documented, supported, and distributed by me as a serious sideline. Anyone can access them from my home page.

I would say that the essential elements of building this family of programs began on the system side with a good development environment. Turbo Pascal was developing rapidly but always usefully, and extraordinary insights (and much dross) were available on the BBSs one could prowl. On my side, I think the most important elements were good I/O routines; a good test environment (the programs themselves!) consisting of a GUI and a consistent pattern of interaction, with the user (usually myself) relying on menus to take the place of explicit language; and good on-screen and PostScript graphics (post-1990). Once a useful programming idiom had been mastered, exhaustion, or perhaps stupefaction, became the ultimate bound on my productivity. My efforts would not have survived difficulties as great as those which seem to have been encountered by AIPS++ in the C++ environments they have used.

It is not my intention to proselytize for drawspec or to discuss its capabilities, because 1996 is hardly the time to advertise a DOS-based program written in an archaic language. I have kept the program current (relative to its competitors) and it is used all around the world by a community which, I must admit, is often largely unknown to me. When Sandy Sandquist noticed the abstract of this talk on the Web, he sent me a copy of Sandquist & Hagström (1995) which points out that drawspec has been adopted as part of the standard undergraduate astronomy curriculum in Sweden. My programs are also used in Mongolia, Brazil, Texas, and the Netherlands.

2. Imagery and Dialogue

Recently I had occasion to look for a rental car in a small lot of such cars. The tag on my keys gave the make, model and license number, as well as the color, written as "Rust." This seemed an unlikely and singularly infelicitous choice; are we to imagine ourselves actually selecting "rust" from a palette of car colors? Or asking a store clerk for rust-colored touchup paint at some time in the future? This same color on my kitchen refrigerator is called "Harvest Gold."

The computer industry is rife with its own barely-noticed versions of this phenomenon. Compaq recently unveiled its line of "Armada" laptops, ignoring that the Spanish Armada had been a fleet of singularly heavy, unwieldy vehicles which (according to English-language text books) was roundly defeated by a more mobile assortment of much smaller and therefore more useful craft. The logo for Java, today's language *du jour*, is a cup of coffee which exists only because of the evanescent arrangement of its own vapors. Vapors? Would I, as a computer manager, not be instantly forgiven for believing I was being mocked by anyone who suggested basing a new project on a technology whose very symbol was "vapor?"

Not paying attention to the use of inappropriate ordinary language isn't terribly costly in the examples I've given, merely absurd. But there is a misuse of language which has had a cost. I believe it is brought on by too much raw contact with objects (in the sense of OOP programming) and I call it speaking in tongues, reminiscent as it is of the snake-handling in some local religious ceremonies that leads to unusual verbal outbursts (which are misconstrued by other true believers as being divinely inspired).

Simply put, there is a lamentable tendency among OO-programmers to attempt to grant false dignity to their activities by renaming everything. In the AIPS++ project, I objected to the desktop being called the "arena" (since arenas are used locally only for cock and dog fights), and to the command log being called a "transparency tool" (I found this usage opaque). There seem to be no such things as "data" or "files," just measurementSets, Tables, Iterators, *usw*. Simple questions which astronomers know how to ask, like "What will I have to do to read a scan from the file containing last night's data" are practically guaranteed to elicit answers which contain not a single intelligible noun or verb phrase.

The inevitable result of this is to inspire terror in the hearts of users who have struggled to gain whatever meager purchase they might have on their computer literacy. The terms "object" and "object-oriented" inspire genuine, instinctive, all-consuming, visceral fear and loathing in almost any astronomer not fully engaged in programming. The problems of incapacitated companies like Borland seem to have been assimilated into the collective unconscious. This firm, as you may remember, went into the drink when it tried to use OO techniques in its own product development, instead of trying to make money by foisting OOP tools on slow-witted consumers (this is a common failing of dope dealers, too, of course). When there was left not a single Borland employee who could speak intelligibly to a banker or a spouse, the firm withered like a community of Shakers.

3. Programs, Not Toolkits

In my own work, it first seemed reasonable to distribute not programs but code or toolkits; I would empower those like-minded individuals who would enjoy, every bit as much as I had, the numerous overwhelmingly tedious and exacting tasks which go into handling real data or writing laser printer drivers. I would be the venerated leader of a band of *sympatico* individuals who admired the elegance and practicality of my handiwork. My particular binary data format, every odd byte of it, would become second nature for a generation of Turbo Pascal-programming, data-reducing astronomers.

This loony fantasy evaporated when I realized that ensuring correct, consistent, and productive use of my code could best be effected by providing a working program. Explaining the proper use of the code in toolkit form was not itself an easy chore, and the best example of how to use it would in fact be—what else?—a program. Mastering complexities and resolving ambiguities seemed preferable to exposing them. And the vast majority of any market for my offline data analysis programs were to be the very souls whom I would forever have disenfranchised by asking each of them to roll their very own.

4. Algorithm and Interface

Aperture synthesis projects, in which large quantities of data are merged to produce a single product like a map or cube, are compute-intensive, so that the ratio of time spent in the interface to that spent waiting for CPU and I/O to finish is typically small. Synthesis work, embodied by our AIPS package, is all algorithm and no interface. This is not to belittle the AIPS user interface, which for all its seeming clunkiness has proved very sturdy and highly serviceable. The 15-year span over which it has been used to reduce so much data is equal to the interval between Bill Monroe forming the Bluegrass Boys and writing "Blue Moon of Kentucky" and Elvis Presley doing his first commercial recording, of just that tune.

For the sort of chores which my programs tackle, the reverse is often true; large numbers of small, highly interactive steps with quick turnaround time are employed; this single dish processing is all interface and no algorithm. Of course these unequivocal statements are limits; one of my programs' failings is that they often deal with datasets as collections, not entities, and are sometimes insufficiently macroscopic. Other programs, like AIPS, are perhaps insufficiently granular.

5. Don't Write Coy Programs

Even in the most supposedly highly interactive programs, there are often elements of coyness and a certain Alphonse/Gaston quality about whose turn it actually is to go first. You bring up some program which is supposed to run interactively and it stares back blankly at you waiting for input. If the program or one of its subroutines has an easily encapsulated goal, should they not examine their own condition and suggest to you what should happen next to achieve it? If this point seems obscure, imagine trying to hold a dialog or complete a project with people who behaved the way most interactive programs do.

There is another aspect to the coy behaviour of many programs; it is when they know something the user doesn't, and so oblige the user to make a choice between begging and proceeding in ignorance. Making a request of a program has two elements of learned behaviour (how to ask and what to ask for), each of which is likely to require a separate traversal of the manual. If there are pieces of information or derived quantities which are important to the conduct of a session with the program, a way should be found to make them available without demanding that the user beg for them.

Yet another aspect of coyness concerns data formats. As the author of an offline data analysis program employing a binary data file format which no telescope produces, and so lacking captive datasets and/or clientele, my mantra is "data is where you find it." Making FITS-reading second nature for drawspec enlarged its scope greatly. Importing data shouldn't be like arranging a marriage between two people of differing, equally stubbornly-held beliefs ("I'll only marry you if you convert to 1-byte values"). If a dataset has coordinates, they should be honored. If it has values, their precision should not be degraded. If large quantities of interesting data exist in a common format (like FITS), that format should be read with a minimum of fuss.

6. Don't be Afraid to Distribute Crap

Any useful analysis program soon begins to grow horizontally and vertically as the result of a wholly natural and predictable process. Once users have made the investment of getting data into an application, they will require that it do more and more for them. Why? Because the only alternative would be to make the *additional* effort to export from that package and import into yet another, at which point the whole futile cycle would only have begun again.

As an example, consider that AIPS started out as a package to image calibrated synthesis data, and eventually subsumed all the calibration chores and the task of producing publication-quality maps. The low esthetic quality of many of these maps certainly diminishes the quality of the publications which agree to print them unadorned, but AIPS' maps have faithful axes which users really appreciate. The continuing refusal of optical astronomers to demand coordinates on their images is a mysterious phenomenon to radio astronomers.

I have discovered that there is a natural cradle-to-grave cycle which must be supported when dealing with data:

Calibration	or the answers will be wrong,
Reduction	or there will be no answers,
Analysis	or there will be no insight, and
Presentation	or there will be no readers.

This is easy to remember *via* its acronym. I am proud to support and ship programs which are so full of it.

7. Why Your Product Should Really Have a Manual, and Why the Manual Must Be Kept Secret

Astronomers hate to read manuals under all circumstances. The idea that one would sit down with a manual prior to using even the most complex device is, at best, anachronistic. Users take umbrage at the mere suggestion that they would be proffered a product which needed any level of understanding or insight whatsoever.

Indubitably, the same product, distributed with a thinner or a thicker manual, will seem easier to use if provided only with the former. In the limit, a product with no manual is deemed entirely intuitive, and one with a thick, detailed tome appears so daunting as to appear intractable. Since documentation is expensive to produce and distribute, the choice is a seeming no-brainer: dump the idea of a user's manual. Borland even carried this idea into the realm of computer languages, all but refusing to supply reference manuals for the last versions of their Object Pascal dialect.

Why then do I counsel the provision of a user's manual? Why, to explain to the programmer what he was thinking! There is nothing more embarrassing than forgetting how one's own code is supposed to work, perhaps assuming it is broken when the debugger shows that it performs as designed: nothing is more baffling than trying to figure out why now it works one way when it should be another.

Of course distributing the manual is another matter. Once the manual exists, it exposes not only the flaws and limitations of your product, but also the inability of its author to write a manual.

8. Lying

Somewhere in every program there are code fragments which are destined to return an incorrect or inappropriate response to the user; often they are well-known and seemingly inviolate. For a period of perhaps 5 years in the early 1980s there was a bug in the plotting of spectra from the standard NRAO single dish program such that the x-axis labeling was in error by one-half of one channel; any x-value measured from our hardcopies was wrong by this amount. Although the error was known to the staff and was evident in the hardcopy, there was no mechanism by which a bug fix could be mandated. The staff knew about it, most users did not, and there is no way of knowing how many slightly incorrect values are entombed in the literature on this account.

I have been mortified when, in the past, I discovered (and I have personally found more bugs in my code than my users have communicated to me) that my code was producing visibly wrong results of varying severity and consequence. This is equivalent to lying and lying is quite simply antithetical to the conduct of research. This is the reason why a bug fix always takes priority over any other work I might want to do. But do I spend my days roaming my code looking for errors? No more than I might spend them checking my own old papers in the ApJ for typos. I simply fix bugs as they become apparent. Never complain, never explain, just slipstream.

9. Programming Hazardous to Your Health

I was standing in line at a receptie (that's Nederlands) after a promotie (Ph.D. orals) at Leiden University, when the person behind me, a well-respected non-Dutch astronomer whom I had never met, introduced himself and told me that many times he had wanted to strangle me! For various reasons he had had to read the native drawspec binary format on 32-bit and 64-bit UNIX workstations, and my native format is guaranteed to have an odd number of bytes per scan.

Of course I would not make this mistake now. But in 1985, when I programmed the way astronomers usually do (without really knowing what they are doing), with files having been as exotic as they were in the FORTRAN/mainframe era when there was almost NO permanent disk storage for data, it never occurred to me that the 1-byte granularity of DOS, a 2-bit operating system, was unusual.

10. The Future

I am always asked whether I have ported my programs to the MS-Windows environment (this was the only question I was asked after my talk). The answer is no, partly because I don't own a computer which can run the 32-bit versions of Windows or Turbo Pascal and partly because I am still resting up from producing the DOS version. It's now the turn of AIPS++ to produce a single dish package for the GBT, whose Project Scientist I've been for the last year. I can't wait to see it.

Acknowledgments. I thank the ADASS VI organizers, who either didn't know what they were getting into when they invited me to speak at their meeting, or didn't care. Neologisms like Compaq, Borland, Turbo Pascal, and MS-Windows are the property of their respective owners. The National Radio Astronomy Observatory is a facility of the National Science Foundation, operated by Associated Universities, Inc. under a cooperative agreement.

References

Sandquist, Aa., & Hagström, M. 1995, Highlights in Astronomy, 10, 172