SU36 P30-69

A NUMERICAL LIBRARY AND ITS SUPPORT

by

Tony F. Chan, William M. Coughran, Jr.,
Eric H. Grosse and Michael T. Heath

STAN-M-78-673
NOVEMBER 1078

COMPUTER SCIENCE DEPARTMENT
School of Humanities and Sciences
STANFORD UNIVERSITY

JUNIOR UNIVERSITY
ORGANIZED 1891

Su326 P30-59

A NUMERICAL LIBRARY AND ITS SUPPORT

by

Tony F. Chan[1]

William M. Coughran, Jr.

Eric H. Grosse

Michael T. Heath[2]

Computer Science Department
Stanford University
Stanford, CA 94305

[1]Applied Mathematics Department, 101-50 Caltech, Pasadena, CA 91125.

[2]Computer Sciences Division, Oak Ridge National Laboratory, P. O. Box X, Oak Ridge, TN 37830.

This research was supported in part by Department of Energy contract No. EY-76-S-03-0326 PA#30 and in part by National Science Foundation grant No. MCS77-02082.

Abstract

Reflecting on four years of numerical consulting at the Stanford Linear Accelerator Center, we point out solved and outstanding problems in selecting and installing mathematical software, helping users, maintaining the library and monitoring its use, and managing the consulting operation.

Key words and phrases

library management and organization

mathematical software

numerical analysis

CR Categories: 4.6, 5.1

"Where no counsel is, the people fall: but in the multitude of *counsellors there is safety."*

Proverbs 11:14 (KJV)

1. Introduction

The final delivery system, a routine library with its supporting services, is an area of mathematical software that has received comparatively little attention in the literature (although the papers by Barinka [2] and Cody [5] do discuss some relevant points). By summarizing our experience in setting up and running a numerical program library, we hope to warn those embarking on such a course elsewhere of the amount of effort required to succeed and to describe to those already working in the area how we have tackled various problems.

Historically, people at Stanford wanting numerical help came to the Numerical Analysis Group within the Computer Science Department. Professor George Forsythe, the original leader of the research group, encouraged software work in general and numerical consulting in particular. Eventually, however, the load became heavy enough that the campus computing service was asked to make more formal arrangements, and soon the computing centers on campus and at the associated Stanford Linear Accelerator Center (SLAC) agreed to support Ph.D. students in a part-time consulting capacity. For the first year or so at SLAC, most effort was devoted to selection of good software and writing of high-level documentation. This led to a high-quality library, but one that was not heavily used. So a second phase began, publicizing the library and improving the user interface. With the success of that effort, we entered a third, more stable, stage in which actual contact with the users and their problems was emphasized. Expansion and revision of selected areas continues, in order to meet needs better and to generate the enthusiasm without which the library would wither.

The SLAC consulting operation may be broken down into four main activities: providing software, advising on problem formulation and library use, maintaining and monitoring, and managing the operation.

2. Providing Software

One of the most exciting opportunities presented by the growth of computers is the transfer of research results from one field to another via general purpose software. Program libraries are becoming a main channel between numerical analysis and applications.

Libraries have been in use since the earliest days of computing for a number of good reasons:

- duplication of effort is reduced
- well-tested, well-tuned routines are used
- dangers are flagged
- state-of-the-art algorithms are available
- storage and compilation costs are reduced
- implementation details are done correctly
- elapsed time to get a working program is reduced

In the computing center environment, a well-designed library is particularly important, since few users are willing to put much initial effort into an unfamiliar algorithm. Also, consultants can quickly answer common, easy questions by pointing to a routine.

Unfortunately, we found that existing libraries had serious flaws. They were slow to incorporate advances in the state of the art because of their desire for a systematic collection, their administrative organization, or their relative isolation either from users or researchers. Their documentation tended to overwhelm users with a multitude of choices, and did not provide much guidance in how to make those choices. Despite this, the documentation was so massive that only a couple of reference copies could be kept, which was inconvenient for users housed in the various scattered buildings. Some of the routines were mediocre, hurting the credibility of the entire package. Finally, and perhaps most fundamentally, the large size of the libraries prevented familiarity, even by experts, with much of the library.

With the exception of special functions, we felt that it should be possible to cover a large fraction of user needs with a core library consisting of a few dozen high-quality routines. To deal with less common problems, we collect experimental routines as well, but even this library is far from the "dumping ground" of user-supplied codes found at many installations. (Most of the remainder of this paper is concerned with the small core library rather than the experimental code collection.)

Routines have been collected from commercial sources (e.g., International Mathematical and Statistical Libraries (IMSL)), from public or government distribution sites (e.g., the Argonne Code Center and the National Center for Atmospheric Research), by participation in software activities (e.g., LINPACK, a linear system package developed under the auspices of the National Activity to Test Software (NATS)), from the open literature (e.g., SICIEI, a routine to compute trigonometric integrals), directly from authors (e.g., SLEIGN, a routine to solve Sturm-Liouville eigenvalue problems), and from local sources (e.g., VARPRO, a routine to solve nonlinear fitting problems). Few codes have been written directly for the library, since we view the librarian's task as one of selection, not production. (However, to bide time until LINPACK became available, a linear least squares routine was prepared.)

Numerical analysts often remark on the difficulty of comparing codes and, therefore, rarely provide the user with selection guidelines. Our aim is to provide at least a clear decision procedure for choosing a routine based on characteristics of the problem which the user understands and, if possible, to select a single routine [10]. We have based these choices on what limited published comparisons are available, personal experience, and discussions with the many visitors to the Numerical Analysis Group at Stanford.

The principal criteria have been: ease of use, machine efficiency, coding practices, and availability. These standards are taken seriously; even if it leaves noticeable gaps in the library, we keep a code in experimental status until it measures up. However, we do not require nice but inessential features like uniformity, so that in practice we find we can install new algorithms in the library soon after they become available.

SLAC, a government laboratory devoted to high-energy physics, has a fairly powerful computing facility (currently 2 IBM 370/168's, a 360/91, and numerous minicomputers, with a substantial hardware upgrade strongly being considered) which is made available to researchers with few accounting limits. For these reasons, the user community is relatively sophisticated and demanding; there is a consensus that physicist time is more important than machine time.

Consequently, ease of use is our first criterion. By this we mean a clean user interface for the subroutine, good documentation, a reliable algorithm implemented with safeguards so that (as far as possible) wrong answers are not computed without warning, and generality and flexibility so that users can develop familiarity with the routine's behavior and even adapt it to their special problems.

Besides human efficiency, machine efficiency is also considered. Since the numerical work is done on large and busy computer systems, time rather than storage tends to be the main constraint. Even with the heavily-increased use of laboratory minicomputers here, it appears that number crunching will remain on the central machines.

Some routines have been rejected for inclusion in the core library because we did not feel that their coding style was clear enough for maintenance, or because they seemed unreasonably long or overly complicated to us.

Obviously, a routine must be available for us to include it. Note, however, that only importability, not exportability, is required. Thus we even find IBM Assembler Language coding adequate (although personally distasteful except in special circumstances). In practice we have found our greatest portability problems to be political rather than technical.

When we receive codes developed as single precision versions, it is necessary to make a number of changes to generate a double precision version. (It is widely felt that for a number of computations that the single precision of IBM 360/370 equipment is not adequate and double precision should be used in all but the most stable processes.) However, it is well known that any direct modification of source leads to the introduction of new errors and we have, therefore, made increasing use of the AUTODBL precision increase feature of the local compiler [8]. Besides avoiding introducing errors, we are able to easily install new releases of routines without having to remodify them.

Any numerical library such as SLAC's tends to become known to outside agencies. This generates requests for library documentation and actual code. Providing software to a large number of outside users is an enormous task, so it has been determined that library documents and locally-produced codes can be disseminated on a very limited basis but most other requests must be politely refused.

3. Advising on Problem Formulation

Good documentation is crucial to the success of a library, and one can greatly enhance its value by keeping it online. First, everyone can get a copy, either directly at his terminal or, with only slightly more delay, from a line printer (or microfiche equivalent). Single reference copies in the consulting office or even documentation published in book form is simply not good enough for a scattered and diverse user community. A second important advantage of machine-readable documentation is ease of maintenance using text editors and formatting programs, which make continually up-to-date documents possible. (If storage of knowledge in procedural form is one of the most exciting opportunities presented by computers, text handling may be one of the most widely useful.)

The Numerical Analysis Program Library User's Guide [3], or NAPLUG as-it is aptly known, is the heart of our library. It provides the main source of advice to users on how to formulate their mathematical problem in a numerically meaningful way and then how to get a solution. It also provides an educational tool for new consultants, allowing some cumulation of expertise. Finally, it forms the focal point of the library, the key document that forces explicit, careful decisions and coherent organization [4].

Besides an introduction providing a general orientation to the library and a description of how various parts of the system may be used, the NAPLUG consists of a series of chapters on numerical topics, each containing:

- observations on the state of the art
- characterization of important problem classes
- pitfalls in computation
- library routine recommendations
- suggested reading

The latest version, to be released during the summer or autumn of 1978, includes: linear system, eigensystem, and special function chapters based largely on LINPACK, EISPACK, and FUNPACK; an optimization chapter based mainly on the National Physical Laboratory library; an approximation and data fitting chapter dealing with a variety of approximating forms and featuring the nonlinear fitting routine VARPRO; fast Fourier transform and integration chapters; an ordinary differential equation chapter based on fine codes from the Sandia and Lawrence Livermore Laboratories; and finally, a partial differential equation chapter which, because of the primitive state of the art and relative lack of partial differential equation problems locally, currently includes only the fast Poisson solvers from the National Center for Atmospheric Research.

Each routine has a short individual description in a public WRITEUPS library. Any online user can immediately obtain a particular routine's documentation. The descriptions include the calling sequence of each routine (describing the type and meaning of each parameter), possible error returns, and a short description of the method used.

Many of the more complicated routines have short EXAMPLES programs associated with them, which often help people in understanding how to use routines. Since examples can frequently be modified into a form which will solve the user's problem, the effort required to write and debug a program is reduced. Such fill-in-the-blanks programming has become popular here.

The NAPLUG, as it discusses each problem area, refers to books and papers that one might read for help in solving a difficult problem. Also mentioned are user's handbooks for individual codes, which can vary from non-existant (the routine may only be documented by self-contained comments) to major-sections of an entire book (as in the case of Shampine and Gordon's ODE code [11]). The packages produced by the NATS project, EISPACK and FUNPACK, are particularly noteworthy because they supply a machine-readable handbook describing each routine in some detail. Whenever feasible, copies of reference materials are added to the general SLAC library and to a special collection kept in the consulting office.

As an example of how a user might take advantage of this documentation structure, let us consider the problem of solving some ordinary differential equations. The user may have a first-order system with appropriate initial conditions. The user's guide would allow him to conclude that he has an initial value problem, and upon further reading he might decide his problem is not stiff. Combining this information with the amount of accuracy he demands he would be able to select a particular code, say Shampine and Gordon's ODE. The user could then obtain a description of the code and its calling sequence from the public WRITEUPS library; further information could be gotten by running an example of

ODE from the public EXAMPLES library. If at a later time the user wanted to understand ODE in more detail he could come to the consulting office and examine a copy of Shampine and Gordon's book [11].

While we strongly believe in the written word as the primary source of advice, we realize that direct personal contact is also important [12]. Because of the core library philosophy, we must supplement the NAPLUG by personally referring users to experimental codes for nonstandard problems. Questions on such problems and on the core library provide feedback to us on what is considered inconvenient or unclear, and in what directions expansion of the library should proceed. Summer visitors and other transients, who are rather common at a national laboratory like SLAC, tend to be particularly heavy users of personal consulting. Others who come in may have read our user's guide, but want to be reassured that their own problem really is covered by our general routines.

This personal contact is facilitated by holding regular consulting office hours, by being available by phone at other times (within reason), by occasional seminars, and by visits. The seminars seemed to be most effective at the time the library was introduced, but can be usefully rerun only every few years. They emphasized the practical use of routines and what to watch out for, rather than the theory behind the algorithms, which would have been more appropriate for a mathematical audience. The visits to user groups include both "big game hunting," in which system accounting information is used to identify heavy computing projects where one hopes to have the greatest leverage, and "general safari" trips, stimulated by interesting problems brought into the consulting office or heard about on the grapevine.

Assisting in use of the library also extends to implementation details. To save the user as much effort as possible, and at the same time eliminate pointless recompilation, the library routines are stored online in object form and are automatically linked into programs, just like the SQRT function is. In order to discourage divergent variants of the library routines, source text is kept offline and in some cases involving copyright protection made inaccessible.

4. Maintaining the Library

In any computing environment when one makes a piece of software (numerical or not) generally available, problems arise with any attempt to upgrade. Some users, because of investment or personality considerations, resist modifications to the behavior of any software component. For numerical routine libraries this implies that some sort of guarantee of static conditions must be advanced.

For the SLAC numerical library, upward compatibility is a grave difficulty since it conflicts with the design goal of maintaining a state-of-the-art library. In general, the solution has been to upgrade routines rather quickly if the calling sequence and meaning of parameters for a routine have not changed (the routine has changed in a manner transparent to the user). In the case of a routine being substantially modified so that its user interface has changed, a determination is made as to how heavily it is used. If the routine is used infrequently then those persons who do use it are contacted and the routine upgraded. On the other hand if usage is heavy, a new name is usually introduced.

A routine that has become outdated, but not dangerous, also poses a problem. The solution used at SLAC is to drop the routine from the

NAPLUG, later drop its short documentation from the WRITEUPS library, and finally remove papers describing the routine from the consulting office and expunge the source from the system libraries. The load module for the routine (i.e., the compiled machine code for the routine) is left in place (if it causes no name conflicts); the source for the routine can be obtained from a backup tape volume.

Three versions of a routine can be kept on the system by having three independent system load module libraries designated as new, production, and old [7]. The basic idea is to migrate routines through these libraries. For example, suppose a routine in the production library breaks down in a particular case. The numerical analyst in charge examines the problem and refers it to the proper specialist who (hopefully) fixes the code. The updated routine is compiled and the resulting load module put into the new library; the users who discovered the problem are advised to access the new version. After some time, the erroneous production version of the routine is moved into the old library displacing any previous version; subsequently, the patched version in the new library will be moved into the production library. (If a disastrous problem is discovered in a routine, a corrected load module may be immediately injected into the production library without any "seasoning" time.)

Another technical problem arises from the fact that, since several separate subroutines may be involved in one "routine" and FORTRAN only allows identifiers to be six characters long, names (including COMMON block identifiers) may be duplicated somewhere in the library. Further, since FORTRAN is not block-structured, all routine names are essentially global unless great care is taken. In the case of a unified library, where each piece of software is specifically designed for inclusion in the package, routine names can be made unique (being derived from the problem solved by a particular routine). However, in the case of our core library, real name conflicts can exist and duplication of code does exist. Duplication of code means that when stand-alone (research) codes are imported for use in the library they usually include all necessary routines and, since some tasks (like forming a LU or Cholesky decomposition of a matrix) are quite common, several different versions of a subroutine to do the same task may exist.

The library ignores the code duplication problem since the local system has adequate disk capacity to handle the superfluous code. The problems of name conflicts that can arise with stand-alone codes are resolved with the CHANGE facility of the IBM Linkage Editor [9] which allows one to modify the name of a routine after it has been compiled. This has proven an effective solution since unique names can be supplied for routines that users need not be aware of.

Library routines are monitored in a manner similar to that described by Bailey and Jones [1]. The collection procedure is based upon the use of the run-time LINK/LOAD facility and the System Management Facility of IBM's OS [7]. This monitoring serves two major purposes: determining the pattern of routine usage and protecting users from old or obsolete versions of routines.

. Enough statistical information is collected to study how heavily different parts of the library are used. For example, it appears that data fitting routines are the most important component of the library to most SLAC users while routines to solve differential equations are infrequently used. (It is interesting that at other Department of Energy laboratories this situation is reversed; this implies that libraries and their supporting services should be adapted to the local environment.) Such information guides allocation of effort and is valuable in justifying the library project as a whole.

Monitoring helps users in avoiding old versions of routines. This is important when a user obtains a personal copy of a routine because later it may be discovered that it is in error. The monitor normally informs the user if he is using such a routine by writing a message into the offending job's system message log. For efficiency on subsequent calls of the library routine within the same job, the monitor call is overwritten with a NO-OP instruction.

Each version of each routine is assigned a unique "maintenance number," which it passes to the monitor. The monitor looks up the number in a status table to be sure that no recent changes have made the routine obsolete, then writes out a system accounting record of which user made the call, and optionally what the parameters were. Note that the status table must be dynamically loaded at run-time, so that its data is current even if the user has his own (old) copy of the monitor routine.

Other monitoring schemes are also used. When a user retrieves a source copy of a routine, his identification and the name of the routine he requested are recorded. This enables the systems staff to determine if someone is trying to obtain copies of routines belonging to packages that contractually cannot be removed. Moreover, the monitoring of source retrieval insures that it is possible to track down users with particular problems.

5. Managing the Operation

A major, often unrecognized problem with the kind of thorough consulting operation described here is staffing. Most computer people tend to ignore numerical analysis. Numerical analysts, with a few prominent exceptions, have tended to avoid the tar pits of software libraries. Users lack balance in the knowledge of systems and numerical analysis and are, therefore, unable to do a really professional job of mathematical software organization. We feel that our solution has worked well, although we doubt that it can be very widely applied.

Two numerical analysis Ph.D. students are supported as research assistants with overlapping terms, so that the new consultant can learn the ropes from the older one. The Ph.D. students benefit by the exposure to a variety of practical problems and gain experience in dealing with software collections. The computing community benefits from the students' enthusiasm and, partly through the students' link with the campus research group, numerical acumen and state-of-the-art knowledge. We have observed that with increasing experience comes rapidly decreasing enthusiasm; we see good people go in, burn out in a year, and return to research. One reason seems to be the hard, frustrating choices (about personal scheduling priorities, user problems, system questions, etc.) that must be made*; another is the clerical nature of much of the work. We conclude that no one student should be asked to work full-time on the library for an extended period; anyone happy to do so may not be active enough in research to keep up effectively.

Assuming a turnover in consultants, internal documentation of the library becomes vitally important. At the very minimum, an annotated index of system files, experimental programs, and monitoring logs should be kept, and the typical life-cycle of a routine from initial installation through final removal should be described. As much of this information as possible should be embedded in programs, so that consultants can spend their time on higher level functions and have mechanical operations performed mechanically.

Other management issues also enter into the consulting activity: requests for fiscal resources, decisions about allocation of manpower, and justification of computer resources used. It is sometimes possible for SLAC to procure improved hardware and commercial program products such as sorting packages, compilers, and operating system enhancements, but it is quite difficult to get monies set aside for obtaining numerical codes. In fact, the entire project is run on a relatively small budget since there are few code costs and research assistant salaries are quite low when compared with the cost of a full-time member of the professional staff. However, the number of Ph.D. student hours available is limited, and some staff time is required, so manpower allocation questions must be answered by management.

The automatic monitoring of routine usage provides some estimate of the immediate impact of the library on SLAC computing. Visits by consultants to groups which make heavy use of CPU resources can assist in qualitatively determining the influence of the system on the SLAC community. Contact with users seeking consulting or attending user seminars provides further feedback on the effectiveness of the library. These measurements and observations provide the basis upon which management must judge the relative success and merits of the library project.

6. Conclusion

The SLAC library project seems to have reached an equilibrium. Although new codes are being added and old ones removed, the basic organization of the library has stabilized. The hard choices of which available routines are best suited for inclusion in the core library are now made more routinely since the criteria by which routines are judged have been formalized to some extent. The design of the documentation hierarchy, the writing of the NAPLUG, the implementation of a system to insure that all routines are automatically available without conflict, and the design and implementation of a monitoring system for routine usage have now been done. Experience with the costs of maintenance and providing user consulting has been gleaned. Some idea of how users are affected by our library has been formed.

The NAPLUG has been well received at SLAC and enables many users to make proper code choices when faced with numerical problems. The simple descriptions of problem formulations, of the characteristics of problems relevant to routine selection, and the short discussions of inherent difficulties allow users to deal confidently with a variety of situations. The online WRITEUPS and EXAMPLES libraries assist in the transition from selecting a particular routine to actually using it. The unified load module libraries insure that all core subroutines are automatically callable, without the need for special job control information. The works referenced by the user's guide help interested persons in becoming more sophisticated from a numerical standpoint and give them further insights into the methods employed. The availability of numerical analysts to function as consultants insures that the concerned user can find assistance for almost any computational task.

The decision to select codes from a variety of high-quality sources enables the library to be more responsive to new developments than commercially available packages. However, the code duplication, naming conflict, and library stability issues that arise from this policy complicate maintenance. Quality control considerations are also of some concern since code authors have a variety of opinions about what constitutes mathematical software and adequate testing.

Major difficulties seem to arise in staffing and management. The students involved in the project never act as consultants for a long period of time. Repetitive tasks, argumentative users, and detailed problems of selecting routines and system organization eventually reduce interest. Personnel turnovers make it difficult to insure that a seasoned consultant will be available since, in addition to numerical knowledge, a consultant must be familiar with a number of system details. Management has limited measurement tools to determine how to allocate the available manpower resources. Finally, management must consider the consequences of the fiscal constraints which limit the procurement of commercial codes.

Developing widespread user confidence in any software product is a relatively slow and somewhat arduous task. The means for reaching the user community is limited to system messages, the monthly computer center bulletin, seminars, and consulting contacts; hence, many of the physicists are not aware of day-to-day developments in the computer center. Personal contact, either in the consulting office or by visitations, seems to be extremely useful in converting reticent users from their "home-brewed" codes to the library. (In some sense, users must be "sold" on the value of a properly organized mathematical software library.)

The library project has required a substantial investment of intellectual effort and machine resources over the past four years. User response has been favorable; the overall library organization has proven its worth. As a resource for the researchers at SLAC, our library forms an important foundation of high-quality, state-of-the-art mathematical software which assists them in spending more of their time studying physics and not numerical analysis. Moreover, the symbiosis between the Numerical Analysis Group and SLAC has aided in the practical training of numerical specialists.

Acknowledgments

Some of the ideas contained in this paper are abstracted from a paper by Coughran [6] which was an attempt to explain the library organization to the interested SLAC user. The present paper attempts to reach a wider audience and is less technically specific.

A major project like the one described here inevitably involves many individuals. John Ehrman and Richard Lagerstrom must be singled out for their numerous suggestions, guidance, technical support, and for their belief in the project's efficacy. We wish to thank Robert Beebe, John Bolstad, June Genis, Gene Golub, John Lewis, Franklin Luk, Joseph Oliger, Michael Overton, Lennox Sweeney, and Margaret Wright for their contributions, and Marsha Berger, Daniel Boley, Stephen Nash, and Lloyd Trefethen for their consulting efforts. Numerous code authors have been generous in making their routines available for use in the library. Countless SLAC users have made suggestions that have improved the library. SLAC, SLAC Computing Services of the Stanford Center for Information Processing (SCS-SCIP), and the Department of Energy also deserve thanks for their support.

References

1. Bailey, C. B. and Jones, R. E. Usage and Argument Monitoring of Mathematical Library Routines. ACM Trans. Math. Software 1, 3 (September 1975), 196-209.

2. Barinka, L. L. Some Experience with Constructing, Testing, and Certifying a Standard Mathematical Subroutine Library. ACM Trans. Math. Software 1, 2 (June 1975), 165-177.

3. Bolstad, J. H., Chan, T. F., Coughran, W. M., Jr., Grosse, E. H., Heath, M. T., and Luk, F. T. Numerical Analysis Program Library User's Guide. SCS-SCIP User Note 82 (August 1977).

4. Brooks, F. P., Jr. The Mythical Man-Month. Addison-Wesley, Reading, Massachusetts, 1975.

5. Cody, W. J. The Construction of Numerical Subroutine Libraries. SIAM Rev. 16, 1 (January 1974), 36-46.

6. Coughran, W. M., Jr. The Construction of a Numerical Analysis Program Library. SCS-SCIP Technical Memorandum 107 (September 1977).

7. Ehrman, J. R. Program Library Maintenance and Monitoring. SCS-SCIP Technical Memorandum 103 (January 1977).

8. International Business Machines, Corp. IBM OS FORTRAN IV (H Extended) Compiler Programmer's Guide, SC28-6852-2, November 1974.

9. International Business Machines, Corp. IBM OS Linkage Editor and Loader, GC28-6538-9, January 1972.

10. Newbery, A. C. R. The Boeing Library and Handbook of Mathematical Routines. In Mathematical Software, Rice, John, Ed., Academic Press, New York, 1971, 153-169.

11. Shampine, L. F. and Gordon, M. K. Computer Solution of Ordinary Differential Equations: The Initial Value Problem, Freeman, San Francisco, 1975.

12. Smith, L. B. Elements of Success for User Services. Proc. ACM SIGUCC User Services Conf. IV, November 1976, Tucson, Arizona.

Requests for more detailed information on the current content and structure of the SLAC numerical library or for copies of SCS-SCIP documents should be directed to:

Numerical Analysis Consultant
SLAC Computing Services, Bin 97
Stanford Linear Accelerator Center
2575 Sand Hill Road
Menlo Park, CA 94025