A PROGRAM FOR DEVELOPMENT OF HIGH QUALITY MATHEMATICAL SOFTWARE

By

Wayne R. Cowell
Computer Scientist
Argonne National Laboratory
Argonne, Illinois 60439

and

Lloyd D. Fosdick
Professor of Computer Science
University of Colorado
Boulder, Colorado 80302

Report #CU-CS-079-75 September 1975

This work was supported by the National Science Foundation under grant number DCR74-21785 to Argonne National Laboratory and DCR74-24547 to the University of Colorado.

We have been heavily influenced by conversations with many experts in the mathematical software area, most especially with the following consultants:

Dr. Edward Battiste
International Mathematical & Statistical Libraries, Inc.

Professor Garrett Birkhoff
Harvard University

Mr. W. J. Cody
Argonne National Laboratory

Professor C. William Gear
University of Illinois - Urbana

Dr. Nicholas Metropolis
Los Alamos Scientific Laboratory

Professor Cleve Moler
University of New Mexico

Dr. Hans J. Oser
National Bureau of Standards

Professor Joseph F. Traub
Carnegie-Mellon University

2. Background

"Mathematical software" denotes computer programs which carry out the basic computations of science and engineering. Most mathematical software, at the present time, is numerical in nature, often performing a very large number of elementary arithmetic operations on finite-precision approximations to real and complex numbers. Typical examples include programs to solve systems of equations, to evaluate functions, and to calculate statistical estimates. On the other hand, some mathematical software makes extensive use of integer arithmetic, typically programs to solve combinatorial problems (sorting and searching, permutations, enumeration, etc.). Programs to perform symbolic mathematical operations represent an emerging new area of mathematical software. However, computer programs which perform accounting functions, though they deal primarily with numerical data, are not generally considered mathematical software, nor are assemblers and operating systems, though they perform many symbolic and combinatorial functions.

A PROGRAM FOR DEVELOPMENT OF HIGH QUALITY MATHEMATICAL SOFTWARE

By

Wayne R. Cowell
Computer Scientist
Argonne National Laboratory
Argonne, Illinois 60439

and

Lloyd D. Fosdick
Professor of Computer Science
University of Colorado
Boulder, Colorado 80302

Report #CU-CS-079-75 September 1975

This work was supported by the National Science Foundation under grant number DCR74-21785 to Argonne National Laboratory and DCR74-24547 to the University of Colorado.

ABSTRACT

This is a report of a study entitled "Planning a Mathematical Software Alliance" carried out with support from the National Science Foundation for the purpose of arriving at recommendations for the structure of a national effort to develop high quality mathematical software. The report consists of a transmittal letter containing the recommendations of the study and a position paper, "A Program for Development of High Quality Mathematical Software".

ARGONNE NATIONAL LABORATORY

September 4, 1975

Dr. J. Richard Phillips
Software Quality Research Program
Division of Computer Research
National Science Foundation
1800 G. Street N.W.
Washington, D. C. 20550

Dear Dr. Phillips:

Grants DCR74-21785 to Argonne National Laboratory and DCR74-24547 to the University of Colorado provided support for a project entitled "Planning a Mathematical Software Alliance". This planning effort was a sequel to earlier studies[1] which explored the needs for high-quality mathematical software and the problems associated with its production. The current study has dealt with the issues of organizational structure and costs necessary to mount a national effort to improve the quality of mathematical software available to scientists and engineers. Our report is presented in the attached position paper "A Program for Development of High Quality Mathematical Software".

We were fortunate in having the counsel of eight consultants, listed on page 2 of the paper, who were thorough in their analysis and candid in their responses. We acknowledge their strong influence and their invaluable help but, at the same time, we recognize that each of them speaks for himself and we assume full responsibility for the conclusions we have reached.

Several ongoing activities have provided insight into the organization of mathematical software projects and their associated costs, notably the NATS Project, International Mathematical and Statistical Libraries, Inc., and the

[1] In particular, those funded by NSF under Grant GJ31681 to the University of Colorado and Grants AG325 and AG435 to Argonne National Laboratory.

NAG Project. Among these, the NATS Project has received the most attention in this study because we have focused on directions to be taken by publicly supported efforts in the United States.

From past experience we felt we could describe the processes that would lead to quality software and could estimate the resources required by a program to implement those processes; but that experience could be extrapolated in various ways and several organizational approaches were explored with the consultants before we settled on recommendations to the Foundation. The questions that were especially difficult to resolve related to the degree of centralization of work and of policy-making. Since various kinds of expertise must be brought to bear on mathematical software problems, it is tempting to propose centers where experts can gather and work together. Surely such interaction is absolutely necessary but it can occur in the context of work on a particular problem rather than work at a particular place. There is the danger that the stability provided by a center can become inertia, causing an activity to persist after its usefulness has declined.

Akin to the centralization issue is the question of who makes the key decisions. If the financial resources for mathematical software development were placed in the hands of a trusted few (even if that possibility were open), it is likely that an efficient program would result; but such a course is neither politically realistic nor likely, in the long run, to be innovative and adaptive to fresh ideas. On the other hand, a program involving such a diversity of interests must have strong input from a group with a comprehensive view of the problems and the possibilities.

Our recommendations are compromises between too much and too little centralization. They were hammered out through interplay with the consultants wherein we submitted proposals in working papers which they reviewed. This interaction culminated in an intensive 1-1/2 day meeting on February 21-22, 1975. The recommendations are contained in Section 9 of the position paper. We repeat them here:

1. To accomplish the task of producing high-quality mathematical software, we recommend that the National Science Foundation encourage and entertain proposals for collaborative mathematical software projects as discussed in Sections 3-7 of the position paper and be prepared to fund such projects at a level of approximately $4.5 Million over a period of six years.
2. To provide for information exchange, analysis of specific needs, and stimulation of mathematical software production, we recommend that the National Science Foundation entertain proposals for the establishment of a Mathematical Software Panel as discussed in Section 4 of the position paper.
3. Since the concern for good mathematical software is felt in research centers with various missions, we recommend that the National Science Foundation seek to develop its program (as articulated in the two recommendations above) in cooperation with the mission-oriented programs of other agencies, in particular the Energy Research and Development Administration, with a view toward encouraging the expenditure by other agencies of an additional $4.5 Million over a period of six years.

A proposal (The LINPACK Project) consistent with recommendations 1 and 3 has already been transmitted to you from Argonne and five universities with companion proposals from several university test sites.

We thank the National Science Foundation for their support of this planning project and we look forward to further action by the Foundation to stimulate the development of high quality mathematical software.

Yours sincerely,

[illegible]

Wayne R. Cowell
Computer Scientist
Argonne National Laboratory

Lloyd D. Fosdick

Lloyd D. Fosdick
Professor of Computer Science
University of Colorado

WRC:LDF:mbg

A PROGRAM FOR DEVELOPMENT OF HIGH QUALITY MATHEMATICAL SOFTWARE

A Position Paper for the Project
"Planning a Mathematical Software Alliance"

1. Introduction

This paper describes a coordinated program aimed at improving the quality of mathematical software available to scientists and engineers. The recommended program consists of a number of software development projects of three basic types with coordination provided by an advisory panel. Rationale for the program rests upon the following considerations:

1. The quality of much basic software for numerical computations is poor. The software is unreliable, non-transportable, and inadequately documented. This constitutes a weakness which pervades the whole structure of computing, wasting vital resources, and impeding scientific and technological development;
2. Mathematical software development is not well established either as an academic or industrial activity which it must be in order to meet vital national needs. A publicly funded program will encourage the involvement of creative people and stimulate the growth of private ventures;
3. Several prototypical efforts serve as models of what is possible and information derived from them (particularly from the NATS project) provides a basis for planning an expanded program.

Reason and experience point convincingly to the need for collaboration among mathematical software workers in research laboratories and universities. Any program must draw together the best talent to attack selected problems. We shall recommend a program that emphasizes the careful selection of problems, collaborative efforts toward their solution, and education to explain and encourage the use of software resulting from such efforts.

It is important to distinguish between algorithms and software. An algorithm is a description of a computational process in terms of well defined elementary steps, and may be represented by sets of recurrence relations, flow diagrams, sequences of statements in algorithmic languages such as Algol, etc. Software is the physical realization of an algorithm. It is a computer program and must take into account such operational matters as storage allocation, error conditions, finite precision arithmetic, efficiency, portability, documentation, and the man-machine interface.

Software development and the study of algorithms (the latter is often called numerical analysis when referring to numerical algorithms) are interdependent and can enrich one another. But numerical analysts and software specialists do not always communicate well. It is convenient for the numerical analyst to invent and analyze algorithms without reference to the perversity of real computers and software designers often do not appreciate the mathematical difficulties that concern the numerical analyst. Individuals conversant in both areas are rare although there are a few notable exceptions among the most creative numerical analysts. It is healthy that numerical analysis and software design should appeal to people with different interests and orientation but both suffer when the two communities work in semi-isolation from one another. This communication failure is an important source of weakness in mathematical software.

Software designers and numerical analysts together could form a constituency influencing machine design and the future development of programming languages. One well-known instance (see [1]) shows that hardware manufacturers can be influenced by a concerted effort. With regard to programming languages, we remark that individual mathematical software specialists were far more involved in the design of Algol 60 than in setting Fortran standards and the former is a better language for mathematical software. This has long-term implications since most practical mathematical software will be written in Fortran (especially in the U.S.) for at least the next 5 years.

Recent developments in computer architecture (e.g., see [2]) providing for parallel operations synchronously or asynchronously introduce a radical new set evaluation of a large mathematical software system is a major undertaking and cannot reasonably be expected to be done by unpaid volunteers. Thus, the professional journals serve as important vehicles for communication of algorithms and software, but they are not a substitute for mechanisms to create and distribute mathematical software of very high quality.

The early installation libraries eventually contained mostly Fortran codes. There was some useful circulation of these but they had not been designed for general distribution and were never successfully distributed to a mass audience although computer users' groups, notably SHARE, made heroic efforts to do so. Computer manufacturers developed libraries of elementary functions associated with high-level language processors. These were not uniformly reliable (see [4,5]) although quality has improved in recent releases of the compilers.

The mathematical software libraries available commercially are, for the most part, collections of routines based on research and development work in universities and research laboratories. The companies which produce them are performing a valuable service to the computing community by organizing this material and offering these libraries to customers. We believe that the best of these commercial activities are accomplishing much of what the users' groups failed to do. However, it does not appear that the underlying research, nor even the exacting testing, that distinguishes the best software development projects is a commercially viable activity. That is, we expect that computer manufacturers will make their hardware more attractive to scientific customers by offering mathematical subroutines and we expect that some firms will specialize in mathematical software products but we do not expect that any of these private concerns will initiate major programs in algorithmic research or in concentrated efforts to develop extremely high quality software in a given problem area. These activities will continue to be fostered in universities and research laboratories, and supported by public funds.

The cost of bad software is difficult to determine quantitatively. Open disclosure and discussion of failure by software developers would often raise difficult questions about liability. Moreover, the complexity of the software talented individuals who saw greater opportunities for recognition in the more formal areas of computer science. We believe that more talented individuals can be attracted into the area of software development because the problems are intellectually challenging and because there is a growing awareness in the scientific community, as a result of trauma suffered from bad software, that efforts to develop good software are important. Mathematical software, in particular, is a promising area for increased attention because (a) the products are valuable basic building blocks in scientific and engineering computation; (b) the problems, while difficult, are tractable and significant progress can be made; (c) insights gained in developing quality mathematical software may lead to a better understanding of the larger problems of quality software in general. These reasons convince us that a carefully guided program of publicly-funded support for work in mathematical software and in vitally related areas would have a significant beneficial effect on software technology, its academic foundations, and its associated industry.

3. The Evolutionary Process

We recognize six principal stages in the evolution of mathematical software:

1. Mathematical analysis;
2. Constructing algorithms to perform computations suggested by the analysis;
3. Converting the algorithms into software for a particular class of computers;
4. Organizing collections of software designed for problems in a mathematical area;
5. Validating, certifying, and disseminating the collections;
6. Incorporating the collections into subroutine libraries.

Work in the first two stages is research oriented and is usually found at universities and certain research laboratories. Work in the third stage is development oriented. It involves converting the description of an algorithm into a program acceptable to a compiler for some particular computer or class of computers. If this conversion work is done to satisfy an immediate need of some individual then

The EISPACK and FUNPACK projects, which were carried out under the aegis of NATS, are the prototypical projects of Type 1.

Regarding Type 2 projects, experience has shown that high level languages and language tools greatly simplify the process of software development and they also make the results of that development more accessible to the scientific user. An example of a project in this area is discussed in [13]. We recommend, however, that any Type 2 project supported under this program be associated with a Type 1 (software package) project, no matter what merit it may have for support under other programs.

Our advocacy of Type 3 projects reflects the fact that the development of reliable software depends heavily on good test facilities. It has been reported by Boehm [6] and Brooks [11] that about 50% of the effort in software development goes into testing and debugging. There has been a good deal of research activity in this area recently, including the work of Fosdick and Osterweil [14] and others [15]. Close liaison between Type 3 and Type 1 projects will facilitate applications of this work to the development and validation of mathematical software packages.

4. A Mathematical Software Panel

A national program in mathematical software development will require, on the one hand, enough coordination and control to assure the most effective use of limited resources and, on the other hand, enough freedom to permit wide participation and infusion of ideas. Moreover, constraints will be imposed by the funding level that might be reasonably anticipated and the types of organizational structures that might be supported. Careful consideration of these factors leads us to recommend a program of projects of Types 1, 2, 3 above with cohesion and unity provided by the work of a coordinating and advisory Mathematical Software Panel.

The principal objectives of the Panel are to:

1. Critically evaluate and report the state-of-the-art in mathematical software research, development, and dissemination;

uity of ideas and policy while providing for a turnover that brings fresh viewpoints. The Panel should meet as a body about twice per year although subpanels may meet more frequently and for longer periods. There should be sufficient staff and clerical assistance so that the members can concentrate on issues. The selection of members should be made in a manner which will assure broad and expert representation of the various interests of the mathematical software community. SIGNUM and SIAM should be represented.

Based on these general recommendations, a host institution must determine the operational details and seek support for the establishment of the Panel. We have concluded that a university is the best choice for a host institution. In a university setting, the Panel will be in a particularly good position to maintain its independent position as reviewer and critic of software activity. Furthermore, a university can provide a good environment for meetings and workshops. The Panel need not always be centered at the same university, but might move after three or four years.

The Department of Computer Science of the University of Colorado would be willing to serve initially as host for the Panel. As already noted there is a strong interest at Colorado in problems related to software reliability and testing. Other members of the faculty have a strong interest in numerical analysis, and special languages for mathematical software. For these reasons, and because the University of Colorado enjoys an unusually attractive location, it is particularly well suited to serve as host for the various meetings and workshops that would be sponsored by the Panel.

5. Software Projects

The actual work of developing mathematical software, special languages, and debugging and testing tools will be carried out through projects which will usually involve collaboration among individuals at several institutions. It is important to recognize that we are not recommending the establishment of a new organizational entity (a "center" or an "institute") to be host to the projects. Centralization would apparently offer efficiency and the advantage of focusing mathematical software development in one superactive center of excellence. However, there are several reasons why we feel that the dispersal of such activity is necessary: First, the skilled people are widely dispersed.

The principal institution would often serve as a site for meetings of workers from the collaborating institutions. At certain points during the course of a project some of these workers may need to spend an extended period, perhaps up to three months, at the principal institution. We anticipate that remote use of the principal institution's computing facilities will be desired.

6. Type 1 Projects (Mathematical Software Packages)

To make more concrete the notion of a mathematical software package project we shall review certain features of the NATS project and attempt to relate this experience to a Type 1 project carried out under the aegis of the Mathematical Software Panel.

The eigensystem package EISPACK and the special function package FUNPACK were based on state-of-the-art algorithms and earlier software that had been carefully selected. EISPACK was constructed from the algorithms expressed in Algol in [16] and FUNPACK from Fortran routines developed at Argonne after considerable study by W. J. Cody. In general, a review of the state-of-the-art would be an explicit prelude to a project and would be an appropriate activity to be carried out under the auspices of the Panel. One approach would be to form a study group, possibly with joint sponsorship by other institutions interested in a particular software area. The study group of three to five experts would agree to:

1. Review numerical techniques in the problem area;
2. Review available software;
3. Write a state-of-the-art report.

The work of the study group should begin with a one month workshop to exchange ideas, parcel out work assignments, agree to a timetable, and begin the reviews. At the end of the month the study group members would return to their home institutions to work on the reviews on a part-time basis. The study group would have a chairperson who, in the ideal case, would devote a major portion of his time to this effort.

After a suitable period (say nine months to be compatible with the academic calendar) the study group would reconvene to complete their reviews and write the state-of-the-art report. Early in the nine month period a conference in-

Release 1 of EISPACK, six for Release 2 of EISPACK, one for the EISPAC control program, and three for FUNPACK. The approach to field testing of EISPACK is discussed in [7] and [17] and of FUNPACK in [7] but more completely in [18].

The test site meetings shown in Table 1 were one-day report and planning sessions. The Eigensystem Workshop was a five-day intensive meeting of about thirty international experts in numerical linear algebra who could be expected to influence the use of EISPACK. Five one-hour lectures (four by J. Wilkinson, one by C. Moler) were videotaped at the workshop and have been widely distributed through interlibrary loans and sale at duplication cost. Such a workshop would be a highly appropriate activity of the Mathematical Software Panel.

The meaning of "certification" is discussed in [7] and may be illuminated by the following certification statement which appears on codes in Release 2 of EISPACK.

> Under the auspices of the NATS Project, the subroutines constituting Release 2 of EISPACK, including (name of routine), have been thoroughly tested on the following computer systems: (List of machines, operating systems, compilers, and locations).
>
> The performance of this software on these systems was satisfactory and Release 2 of EISPACK is hereby certified for these systems. The NATS Project fully supports this certified software in the sense that reports of poor or incorrect performance on at least the machines and operating systems listed above will be examined and any necessary corrections made. This assurance of support applies only when the software is obtained directly from the Argonne Code Center and has not been modified.
>
> Questions and comments should be directed to:
>
> (Name, address, and phone of contact)
>
> The developers of EISPACK intend to support the package throughout its useful life or until, in the estimation of the developers, it is superseded or incorporated into other supported, widely-available program libraries. Information about any change in the status of EISPACK support will be supplied to recipients of the package or may be obtained from the above-named individual.

Table 1 - EISPACK

MILESTONE DATES (APPROXIMATE)	Release 1	MONTHS	CUM. $ IN THOUSANDS	Release 2	MONTHS	CUM. $ IN THOUSANDS
11/1/70	1 SPS man-year prior to 11/1/70 Major effort committed 4.2 SPS .5 CS	0	46			
9/1/71	Package to test sites Test Site Meeting 4.0 SPS .5 CS 2 PS equiv. for testing	10	218			
5/1/72	Package Certified, Distribution and Support Begin 1.3 SPS Test Site Meeting .3 CS .1 Grad.	18	407	Effort for Release 2 committed 1.4 SPS .3 CS .1 Grad.	0	0
7/1/73 9/1/73	Eigensystem Workshop ($25T) Users Guide Manuscript to Publisher (see ref. [19])	34	528			
				Increase effort level 1.5 SPS .75 PS .5 CS	20	122
9/1/74				Package to test sites 1.5 SPS .75 PS .5 CS 2 PS equiv. for testing	28	197
5/1/75				Certification, Distribution, Support 1.5 SPS .75 PS .5 CS	36	324
9/1/75				Second edition Users Guide to Publisher	41	371
				Continuing Support (.1 SPS)		

Table 3

EISPACK Size

(For each applicable machine)

	Total Cards	Comment Cards	
Release 1 (34 Routines)	5719	2794	Control Program not included
Release 2 (70 Routines)	11432	5556	Control Program not included but 12 certified drivers included
EISPAC Control 2			
Fortran	3089	1117	
Assembler	688	151	
Release 2 (70 Routines)			
Machine Readable Documentation	12373	----	

Table 4

FUNPACK Size

	Total Cards	Comment Cards
IBM FUNPACK	1823	1223
CDC FUNPACK	1374	913
Univac FUNPACK	1377	860
IBM Demonstrator	980	278
CDC Demonstrator	632	173
Univac Demonstrator	652	174
Machine Readable Documentation (For each machine)	2125	----

"Static analysis" generally denotes automatic scanning (but not execution) of a program in a search for errors of suspicious constructions. Such scanning and error detection is done by compilers; however recent work [14] has shown that a much deeper analysis than compilers perform is possible and practical.

Other work in this area includes automatic generation of test data, identification of "impossible paths", and symbolic execution of programs (see [22]). Here it seems that further work is needed before useful software can be developed, but it would be appropriate for the Panel to review and encourage these efforts.

The system DAVE which was developed at the University of Colorado gives us a model on which to base estimates of costs and manpower required for projects in the software testing and validation area. This system was designed and constructed over a two-year period and it is operational but not fully tested. It contains approximately 20,000 source statements. The costs to bring the system to its present state were approximately $170,000. The effort was approximately 2.5 man years at the "SPS" level, 2 man years at the "PS" level, 3.5 man years at the "Grad" level, and 0.75 man years at the "CS" level. We emphasize that DAVE is not quality software; rather it is an experimental prototype whose cost per source statement is far lower than would be the case for high quality software intended for wide distribution.

8. Costs

We estimate that the cost of Type 1 or Type 3 projects would average roughly $250 Thousand per project per year and that the cost of the activities carried out directly by the Mathematical Software Panel would be $100 Thousand per year. Further, we believe that fundamental changes in available mathematical software would be effected by a program of approximately 12 major projects extending over a period of 6 years, roughly according to the following schedule (each project is assumed to last 3 years). We are convinced that enough expertise and interest exists to mount a worthy program of this size:

Panel as discussed in Section 4.

Since the use of mathematical software pervades all of science and engineering the concern for good mathematical software is felt in research centers with various missions. In particular, the Energy Research and Development Administration has played a vital role in ongoing activities in this area. We recommend that the National Science Foundation seek to develop its program (as articulated in the two recommendations above) in cooperation with the mission-oriented programs of other agencies, with a view toward encouraging the expenditure by other agencies of an additional \$4.5 Million over a period of six years.

References (Continued)

15. Workshop 4 - Approaches for Programmers to Application Software Validation. Proceedings of Computer Science and Statistics: 8th Annual Symposium on the Interface. Feb. 13, 14 (1975). University of California, Los Angeles, CA.

16. Wilkinson, J. H. and Reinsch, C., Handbook for Automatic Computation, Volume II, Linear Algebra, Part 2, Springer-Verlag (1971).

17. Cowell, Wayne, The Validation of Mathematical Software, to be presented at IFIP-INFOPOL-76, International Conference on Data Processing, Warsaw, Poland, March 22-27, 1976.

18. Cody, W. J., The FUNPACK Package of Special Function Subroutines, ACM Transactions on Mathematical Software, Vol. 1, No. 1 (March, 1975), pp. 13-25.

19. Smith, B. T., Boyle, J. M., Garbow, B. S., Ikebe, Y., Klema, V. C., Moler, C. B., Matrix Eigensystem Routines - EISPACK Guide, Lecture Notes in Computer Science, Vol. 6, Springer-Verlag (1974).

20. Elspas, B., Levitt, K. N., Waldinger, R. O., and Waksman, A., An Assessment of Techniques for Proving Program Correctness, ACM Comp. Surv., Vol. 4, No. 2 (June, 1972), pp. 97-147.

21. Proceedings International Conference on Relaible Software. April 21-23 (1975), Los Angeles, CA.

22. Clarke, L., A System to Generate Test Data and Symbolically Execute Programs. Technical Report CU-CS-060-75, Department of Computer Science, University of Colorado, Boulder, CO (Feb., 1975).