AD-A068 023 NAVAL SURFACE WEAPONS CENTER DAHLGREN LAB VA F/G 9/2
DEVELOPMENT OF MATHEMATICAL SOFTWARE AND MATHEMATICAL SOFTWARE --ETC(U)
APR 79 A H MORRIS

UNCLASSIFIED NSWC/DL-TR-79-102 NL

| OF |
AD
A068023

END
DATE
FILMED
6-79
DDC

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: NSWC TR 79-102	2. GOVT ACCESSION NO. / 3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)*: (6) DEVELOPMENT OF MATHEMATICAL SOFTWARE AND MATHEMATICAL SOFTWARE LIBRARIES	5. TYPE OF REPORT & PERIOD COVERED
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): (10) Alfred H. Morris, Jr.	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Naval Surface Weapons Center, Code K74, Dahlgren, VA 22448 — Dahlgren Lab.	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: Naval Surface Weapons Center, Code K74, Dahlgren, VA 22448	(11) 12. REPORT DATE: Apr[illegible]79 / 13. NUMBER OF PAGES: 27
14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)*: (12) 31 p.	15. SECURITY CLASS. *(of this report)*: UNCLASSIFIED / 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited.

(14) NSWC/DL-TR-79-102

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

Library Development, numerical mathematical software, subroutine reliability and portability, subroutine evaluation criteria
CR Categories: 1.3, 4.2, 4.6, 5.1

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

The purpose of this paper is to briefly examine several of the major issues concerning the development of numerical mathematical software and numerical mathematical software libraries. The paper begins with a brief summary of the evolution of general-purpose mathematical software libraries. This is followed by an introductory discussion on software reliability. It is often tacitly assumed that most of the basic numerical mathematics problems have satisfactorily been solved. This is shown not to be the case. Indeed, it is noted →

next page

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

UNCLASSIFIED

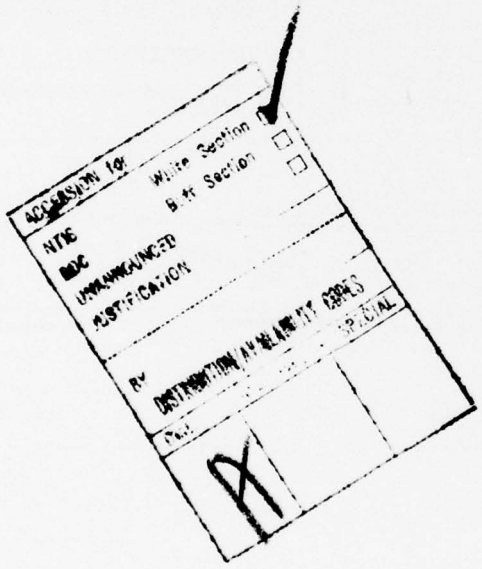
SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

391 598

that many of the problems encounter not only deep theoretical difficulties, but also numerous software engineering problems.

The next major topic is software portability. Here the emphasis is on portability difficulties that arise from design deficiencies in the programming languages. It is noted that FORTRAN permits an arithmetic expression to be altered when it is known that the modification can produce different results!

The final issues considered are those involved in forming a library. If the purpose of the library is to serve as broad an audience as possible, then it is recognized that the subroutines in the library should be as simple to use and as comprehensive as is practical. Thus formation of the library can be characterized as a packaging problem, the objective being to package mathematical formulae and theory into comprehensive, simple-to-use subroutines.



TR 79-102
April 1979

DEVELOPMENT OF MATHEMATICAL SOFTWARE AND MATHEMATICAL SOFTWARE LIBRARIES

Alfred H. Morris, Jr.

Strategic Systems Department

FOREWORD

This paper briefly examines several of the major issues concerning the development of numerical mathematical software and numerical mathematical software libraries. The paper was funded by the Computer Facilities Division. The author is grateful to H.W. Thombs, R.A. Niemann, L.R. Diesen, J.G. Perry, A.P. Gass, H.G.M. Huber, and many other individuals for their comments and suggestions during the development of this paper.

Released by:

Ralph A Niemann

RALPH A. NIEMANN, Head
Strategic Systems Department

ABSTRACT

The purpose of this paper is to briefly examine several of the major issues concerning the development of numerical mathematical software and numerical mathematical software libraries. The paper begins with a brief summary of the evolution of general-purpose mathematical software libraries. This is followed by an introductory discussion on software reliability. It is often tacitly assumed that most of the basic numerical mathematics problems have satisfactorily been solved. This is shown not to be the case. Indeed, it is noted that many of the problems encounter not only deep theoretical difficulties, but also numerous software engineering problems.

The next major topic is software portability. Here the emphasis is on portability difficulties that arise from design deficiencies in the programming languages. It is noted that FORTRAN permits an arithmetic expression to be altered when it is known that the modification can produce different results!

The final issues considered are those involved in forming a library. If the purpose of the library is to serve as broad an audience as possible, then it is recognized that the subroutines in the library should be as simple to use and as comprehensive as is practical. Thus formation of the library can be characterized as a packaging problem, the objective being to package mathematical formulae and theory into comprehensive, simple-to-use subroutines.

CR Categories: 1.3, 4.2, 4.6, 5.1

TABLE OF CONTENTS

1. Introduction

Attitudes concerning the development of numerical mathematical software libraries have changed considerably in the last 20 years. In the early and middle 1960's each laboratory that had a computer found it to be expedient to have a library of routines which everyone could use. Normally the library was just a repository for commonly used software. The libraries seem to have evolved by osmosis. If a routine was found to be particularly useful, then more often than not the routine was blindly dumped into the library. Little or no quality control was maintained for the mathematics software. As a result, most libraries contained a mixture of good, mediocre, and unbelievably poor routines.

In the latter 1960's and early 1970's, because of the increasing cost in the production and maintenance of software, the increased complexity of the problems being considered, the overwhelming (and frequently needless) duplication of code, and the general unreliability of many of the existing codes, the relaxed attitudes concerning the formation of numerical mathematical software libraries began to change. By now it was clear that any laboratory which employed computers intensively for a variety of scientific applications should contain a library of accurate, efficient general-purpose mathematics subroutines. However, it was equally clear that the formation of a reliable, comprehensive library was not necessarily a simple task. Among other things, the production of software frequently required technical expertise that was not available in-house. Also there was the time involved in the development of accurate, efficient routines for carrying out basic tasks. The development

of routines to compute Bessel functions, to calculate the eigenvalues and eigenvectors of matrices, or to solve stiff systems of differential equations was found to be extremely time consuming, frequently requiring the development of new mathematical formulae and/or theory.

In 1971 the NATS (National Activity to Test Software) project began. NATS was a joint effort by the Atomic Energy Commission and the National Science Foundation to examine the problems, costs, and resources involved in the production, certification, dissemination, and maintenance of high-quality mathematical software. The project was centered at Argonne National Laboratory, and involved the collaborative effort of individuals at some two dozen university and research laboratories. The two products of NATS were EISPACK, a comprehensive package of FORTRAN subroutines for eigenvalue/eigenvector computation, and FUNPACK, a small collection of special function subroutines. EISPACK has had significant impact, having been distributed to approximately 450 sites. Exceedingly high quality control was maintained in the development of EISPACK and FUNPACK. The overall cost for EISPACK was approximately $ 900,000.

In 1970 development of the IMSL and NAG libraries began. The IMSL library was a commercial venture by the International Mathematical and Statistical Libraries corporation. To gain acceptance, emphasis was placed on developing a quality library that was comprehensive in both mathematics and statistics, thereby providing greater capability than most laboratory libraries possessed. The NAG (Numerical Algorithms Group) project began as (and currently is) a joint effort by British universities and government research laboratories to produce a comprehensive numerical library. The effort is now established as

a non-profit organization, being subsidized by the British government. Part of NAG's income is derived from renting its library, which places it in direct competition with IMSL. The IMSL and NAG libraries are now fairly good libraries. They provide broad capability for a variety of computers at an economical price. In recent years these libraries have begun to have an impact on the activities of many laboratories. Some organizations have adopted either IMSL or NAG to be their sole library, whereas other organizations are employing IMSL and/or NAG in conjunction with an in-house ' rary. The acceptance of the IMSL and NAG libraries reflects the need that exists for good general -purpose numerical software.

In the last 10-12 years Sandia Laboratories and other organizations have also begun a slow, methodical development of numerical mathematical software libraries. The purpose for many of these libraries has not been to lease or sell software, but to provide quality software for in-house use and/or for general use by other organizations. Development of much of the software has been in-house, requiring a substantial research and development investment. In many cases, a laboratory has leased a commercial library such as IMSL while developing its own in-house software capability. When this occurs, normally the two libraries (the leased library and the in-house library) are kept totally separate from one another. They tend to provide complementary rather than duplicate capability. Currently, all software libraries are deficient to some degree. Deficiencies cannot be avoided, since there are possibly more basic numerical mathematical questions that have not yet been answered than have been answered. Thus, the development of basic high-quality mathematical

software can be expected to continue for the next 20 years.

Since the latter 1960's, as a result of the extensive work done in the development of high-quality software, many of the problems concerning the production of mathematical software and mathematical software libraries have been brought into sharp focus. The purpose of this paper is not to provide an encyclopedic treatment of all the issues and techniques involved with mathematical software and mathematical software library development. Instead, the purpose is to provide an introduction to the subject by discussing some of the issues which the author considers to be particularly important.

2. Mathematical Software Reliability

Since the middle 1960's reliability has become one of the critical issues in mathematical software research and development. To illustrate the dependence of reliability on mathematical theory, consider the EISPACK package of routines that was produced by NATS. EISPACK is possibly the highest quality and most comprehensive collection of eigenvalue/eigenvector routines currently in existence. Does EISPACK take care of all the eigenvalue/eigenvector problems that arise? Not at all! The EISPACK routines can yield high precision results for the eigenvectors of a matrix if the matrix is not too badly conditioned and the eigenvalues are distinct and well separated from one another. However, if the eigenvalues are repeated or exceedingly tightly clustered, then all accuracy can be lost and one can obtain eigenvectors that are not eigenvectors! The problem is due not to the software engineering of the routines, but to the inability of the underlying algorithms to cope with the situation. Currently no algorithms are known that satisfactorily compute the eigenvectors of exceedingly close eigenvalues. Indeed, it is debatable whether such algorithms can be constructed.

It is often tacitly assumed that most of the basic mathematics problems have satisfactorily been solved. Unfortunately, this is not true as EISPACK illustrates. Indeed, many of the problems involve surprisingly sophisticated and/or lengthy mathematical considerations.

In addition to the theoretical difficulties that are frequently encountered in solving a problem, there are also software engineering difficulties that can arise. The linear programming problem is a classic example of where both theo-

retical and developmental difficulties abound. Not only has the problem generated an impressive amount of new mathematics, also its software engineering difficulties are numerous and varied. It has been stated that most implementations of the simplex algorithm (the procedure for solving linear programming problems) are inherently numerically unstable. This is quite probably the case.

To get a feel for the type of reasoning that can be involved in software engineering, consider the task of computing $\sin \pi x^2$ for large x, say $x \geq 100$. Now why should $\sin \pi x^2$ present difficulties? The reason, of course, is that when x^2 is computed, the leading digits of x^2 are retained and the least significant digits are discarded. However, because of the periodicity of the sine function, many of the leading digits are actually of no consequence and many of the digits that are discarded are quite important. Thus it is clear that the angle πx^2 must be reduced to a smaller angle before the sine function routine is invoked. An appropriate first reduction step might be to partition x into its integer and fractional parts n and r. Then $x = n + r$ where $0 \leq r < 1$, and we note that $\sin \pi x^2 = (-1)^n \sin \pi (2nr + r^2)$. The following FORTRAN instructions can be used for computing n and r :

```
N = X
R = X - N
```

Here it is assumed that x is positive and that x is small enough so that its integer portion n is representable in the integer format of the computer being used. Now if we are using a k-digit arithmetic and the integer n is j digits long, then it is clear that only the first k - j digits of r are of consequence. Thus the product nr will have at most $j + (k - j)$ significant digits, none of

which need be discarded. This implies that the following FORTRAN statements

```
T = N*R
M = T
A = T - M
```

should cause no loss of accuracy. Here the product nr is computed, and nr is partitioned into its integer and fractional parts m and a. Also, since $nr = m + a$ where $0 \le a < 1$, we note that $\sin \pi(2nr + r^2) = \sin \pi(2a + r^2)$ where $0 \le 2a + r^2 < 3$. Consequently, the angle πx^2 can be accurately and efficiently reduced to an angle in the interval $[0,3\pi]$. Further angle reductions are, of course, also possible.

It has been stated that software engineering can be made into a science. This is debatable since rules cannot be prescribed for how to detect and solve the variety of sensitive situations that can occur. Difficulties can arise from numerical roundoff, the lack of a fully defined algorithm for the finite precision arithmetic, or from heuristic decision making that may have to be performed. The detection and solution of sensitive situations frequently requires considerable expertise, both theoretical and developmental.

3. Portability Considerations

Since the latter 1960's, because of the increasingly high costs involved in the production and maintenance of general-purpose mathematical software, considerable emphasis has been placed upon development of software which would require the minimum amount of change when being moved from one computing environment to another. The goal has not been to produce software which would yield the same exact results on all computers. Identical results are normally not possible since different bases are used for the floating point arithmetics. Instead, the goal has been to produce software that would compute to a prescribed level of accuracy in each computing environment. Because of the dissimilarities of the existing computer arithmetics, it was recognized that complete portability could not always be achieved. Nevertheless, the problem of designing code so as to minimize change has frequently been found to be considerably more difficult than expected. Possibly the primary reason for the difficulty has been the many design deficiencies of the programming languages.

For general-purpose numerical scientific work one frequently needs at least two arithmetics, say a k-digit arithmetic and a higher precision m-digit arithmetic. The higher precision arithmetic may be needed for checking the accuracy of the k-digit software. Also, some algorithms require certain intermediate calculations to be performed in higher precision. ALGOL 60 does not recognize the need for two arithmetics. FORTRAN does.

It is recognized that a laboratory which employs the computer for a variety of scientific calculations normally needs at least 8-10 decimal digit capability. However, many computers employ arithmetics which do not have this

capability. For example, the IBM 370 computer has a 32 bit hexadecimal arithmetic which can assure one of only 6 decimal digit accuracy. Thus, in order to develop software for the IBM 370 that can meet the minimal 8-10 decimal digit requirement, it is necessary that two higher precision arithmetics be available, say a 64 bit arithmetic and a 128 bit arithmetic. The 64 bit arithmetic is required for normal scientific computation, and the 128 bit arithmetic supplies the needed higher precision capability. The most appropriate arrangement for scientific purposes is to let the 64 bit arithmetic be the standard FORTRAN arithmetic and the 128 bit arithmetic be the double precision arithmetic. In this case, FORTRAN can be extended by defining the 32 bit arithmetic to be a "half precision" arithmetic. However, this is not the setup used by the IBM 370. Instead, the 32 bit arithmetic serves as the standard FORTRAN arithmetic, the 64 bit arithmetic is the double precision arithmetic, and the 128 bit arithmetic is a "quadruple precision" arithmetic. Because of the widespread use of the IBM 370 FORTRAN, the negative impact that this arrangement has had on the portability of FORTRAN mathematical software has been overwhelming. However, the arrangement does not violate either the FORTRAN or the FORTRAN 77 standards. Indeed, according to these standards a 1-digit arithmetic would be satisfactory.

For general-purpose numerical scientific work, it is agreed that the basic mathematical and relational operations should be as accurate as possible for the particular floating point arithmetic being used. Also it is agreed that all floating point numbers should be normalized, and that all real numbers which are small integers should be represented exactly in the floating point format. None of these conditions are currently required by FORTRAN and the other major

programming languages. However, normally these deficiencies are not too much of a problem since most computer manufacturers do attempt to satisfy these standards. More serious are the difficulties arising from overflow and underflow. For example, if the division operation x/y results in a value that is too small to be represented as a floating point number, then should x/y be assigned the value 0 or a special value "infinitesimal", or should it be regarded as a fatal error? Clearly, an exceedingly awkward situation exists when all these options are permitted. Nevertheless, none of the major programming languages currently specify how the underflow should be treated.

Other design deficiencies can also be found in the current programming languages. For example, it is particularly irksome that both the FORTRAN and FORTRAN 77 standards permit

```
X = A/5.0
IF (X .GE. 1.0) N = N+1
```

to be interpreted by the FORTRAN compiler as

```
X = A*0.2
IF (X .GE. 1.0) N = N+1
```

when it is known that the second pair of statements will compute the wrong value for N on many computers when A = 5. It should be emphasized that this is no accident. The FORTRAN 77 standards state quite clearly that the compiler is permitted to alter an arithmetic expression when it is known that the modification can produce different results!

Another class of deficiencies that can cause considerable suffering is the lack of a complete set of operations for processing arithmetic data. For

example, neither the FORTRAN nor the FORTRAN 77 standards provide for double precision complex number capability. This deficiency alone eliminates the transportability of many single precision codes to double precision form. A second less obvious example is the lack of an intrinsic function for deciding when two pieces of data are stored in the same location. Why is this needed? The answer is clear. If one wishes to construct a FORTRAN subroutine

```
SUBROUTINE MPROD(A,B,C,...)
```

for computing the product of two matrices A and B and storing the results in C, and if one further wishes to permit the storage area C to begin in the same location as A or B (but not both), then it is required that different algorithms be used for the cases

(1) C begins in the same location as A,

(2) C begins in the same location as B, and

(3) C does not overlap with the storage areas for A and B

if the amount of work space involved is to be kept to a minimum. A typical situation when case (1) would occur is when a statement such as

```
CALL MPROD(R,S,R,...)
```

is invoked. Then it would be required in the MPROD routine that some procedure be available for checking the locations of A(1) and C(1). A suitable procedure would be to use a FORTRAN function such as :

```
FUNCTION LOC(X,Y)
XOLD = X
YOLD = Y
Y = 0.0
IF (YOLD .EQ. 0.0) Y = 1.0
```

```
      IF (X .NE. XOLD) GO TO 10
C     X AND Y ARE IN DIFFERENT LOCATIONS
      Y = YOLD
      LOC = 0
      RETURN
C     X AND Y ARE IN THE SAME LOCATION
   10 Y = YOLD
      LOC = 1
      RETURN
      END
```

The intent, of course, is to define a function LOC that computes

$$LOC(x,y) = \begin{cases} 0 & \text{if x and y are in different locations} \\ 1 & \text{if x and y are in the same location} \end{cases}$$

for any real data x and y. However, does this code actually work? According to the FORTRAN and FORTRAN 77 standards it should compute properly. However, on the CDC 6700 it malfunctions. The reason is not that the statements

```
      Y = 0.0
      IF (YOLD .EQ. 0.0) Y = 1.0
```

do not properly perform their task. They do indeed modify the value of X in memory when X and Y refer to the same location. However, the value used for X in the statement

```
      IF (X .NE. XOLD) GO TO 10
```

is the old value for X still residing in a register! This insidious type of entrapment is classic. It illustrates the exceedingly subtle problems that

can arise when one tries to develop portable software, and standards are lacking or are not adhered to.

4. Development of Mathematical Software Libraries

A discussion of the development of mathematical software libraries is facilitated by considering a specific example, say the NSWC/DL library. In early 1976 formation of the current NSWC/DL (Naval Surface Weapons Center / Dahlgren Laboratory) library began. The task was to form a high-quality library of general-purpose mathematics subroutines that would provide a basic computational capability in a variety of mathematical activities. Emphasis was to be placed on reliability and generality, and redundant abilities were to be kept to a minimum. (It was clear that some redundancy was unavoidable since efficiency might at times be needed at the expense of accuracy and/or generality.) The library subroutines were to be written in FORTRAN. Even though the routines were intended for use on the CDC 6700 computer, every attempt was to be made to ensure their transportability.

There are two ways to proceed in forming a library: (1) start from scratch or (2) begin with an existing library. At NSWC/DL it was necessary to begin with a library which contained 40 routines. The subroutines found deficient could be removed only when suitable replacements could be found or if their removal did not impact the laboratory. Currently (31 January 1979) the library contains 105 subroutines, 7 of which are all that remain of the original 40 routines. Included in the library are routines for computing Bessel functions, finding least squares solutions for systems of linear equations, solving systems of ordinary differential equations, etc. Many of the routines are of exceptional quality, but there are still a few routines that are deficient. All subroutines in the library are periodically reviewed for possible improvement. When better routines are found

then the older routines are eliminated.

Since 1976, when formation of the current library began, all new routines have been subject to evaluation and possible modification before being accepted for the library. Primary considerations include the reliability and portability of the routine, its efficiency and ease of use, and the generality of the routine. In regard to reliability, the major concerns are accuracy, the mathematical stability of the algorithm being employed, and the routine's robustness; i.e., the routine's ability to perform satisfactorily near the limits of applicability of the underlying algorithm.

In regard to portability, it is clear that the use of machine dependent constants and precision dependent algorithms cannot be avoided. However, machine dependent code such as masking operations on floating point numbers has almost always been unacceptable for the NSWC/DL library. It is assumed that the computer arithmetic satisfies criteria such as :

(1) Additive symmetry; i.e., -x is representable as a floating point number if x is a floating point number.

(2) An addition, multiplication, and division operation which underflows is assigned the value 0.

Also it is assumed that the FORTRAN compiler does not alter arithmetic expressions, and that all mathematical and relational operations are performed as accurately as possible for the floating point arithmetic being used. To date, no portability criteria have been formulated for avoiding the difficulties that can occur when these or similar conditions are violated. Generally, the policy is to accept code if it is transportable; i.e., if it requires only changes

which are capable of being implemented by a preprocessor. Sufficient documentation must, of course, be supplied to clarify all conversion ambiguities. For example, if a routine employs a machine dependent constant such as the smallest positive floating point number u for which $1+u > u$, then it is required that u be defined in the documentation when u is not computed by portable code in the routine.

The ease of use criterion for the NSWC/DL library software is of considerable importance. The main purpose of the library is to provide a service to as broad an audience as possible. Thus it is important that duplicate abilities be kept to a minimum, and that the routines be as simple to use and as comprehensive in scope as is practical. To meet these specifications, the EISPACK routines have been incorporated into the NSWC/DL library at a subordinate level. For example, two routines (EIGV and EIGV1) in the NSWC/DL library are available for computing the eigenvalues and eigenvectors of an arbitrary real matrix. EIGV employs elementary similarity transformations to reduce the matrix to upper Hessenberg form, and EIGV1 employs orthogonal similarity transformations. Both routines permit balancing of the matrix if it is desired. The relative merits of each routine and the value of balancing are briefly discussed in the documentation. Now, in actuality EIGV and EIGV1 are driver routines for 7 separate EISPACK subroutines. EIGV and EIGV1 call the appropriate EISPACK subroutines, and then unpack the eigenvectors. Currently, the 7 supportive subroutines are not documented in the NSWC/DL library manual. However, for the specialist the appropriate references are given in the documentation. This treatment of the EISPACK subroutines illustrates the general NSWC/DL library policy for the

handling of routines which are intended only for supportive purposes. The policy of referencing supportive code, thereby inhibiting its use except by the specialist, makes it possible to replace the code with minimal impact to the laboratory. Also, it significantly simplifies the situation for the novice (and most users at NSWC/DL), thereby promoting greater and better use of the software than could otherwise be expected.

Development of software that satisfies the ease of use criterion can be characterized as a packaging problem, the objective being to package mathematical theory and formulae into comprehensive, simple-to-use subroutines. It would appear that the importance of this objective would be self-evident, but this is not always the case. It occasionally occurs that a researcher will design a beautiful algorithm. He will exercise great care in the development of subroutines which compute separate portions of the algorithm, and then he will link the subroutines together by a poorly conceived driver routine that is difficult or almost impossible to use! When this occurs, then the evaluator of the software is frequently forced to either reject all the software, or to possibly completely repackage all the software.

In the packaging of software, extreme caution should be taken not to unnecessarily restrict the scope and versatility of the code. Currently, the only restriction for the NSWC/DL library software that has a direct bearing upon this issue involves the use of I/O. No print statments are permitted for reporting errors. If error detection is performed in a routine, then it is required that the call line of the routine contain a parameter which can be used for reporting the error. The use of such a parameter permits the user to control the subsequent

sequence of events that will occur.

The examination of software for the NSWC/DL library consists of the following steps: (1) examination of the algorithm and portions of the code, (2) testing the software, and (3) an assessment of the utility and overall performance of the software. The testing serves many purposes, including determination of the accuracy and efficiency of the software, checking for defects in the code, searching for regions of numerical instability, etc. Because of the theoretical complexity and the diversity of application of many of the mathematical activities being computerized, only infrequently can the testing be complete. Normally the testing must be highly selective, being used to locate and examine weaknesses in the algorithm and the code.

It is generally recognized that the evaluation and certification of software can be an extremely time consuming task. To ease this burden, as well as minimize maintenance problems, many organizations have begun to impose stringent requirements on the development and documentation of code. However, it is debatable whether such requirements should be imposed. It is recognized that researchers can be fussy, fanatical, sensitive, etc. To try to impose too rigid a format and/or technique frequently makes it far more difficult to coax the researcher into providing software that only he can provide. In any case, the issue is academic at NSWC/DL since such requirements cannot possibly be imposed on the NSWC/DL library. The routines in the library are selected from a variety of sources. Thus, emphasis must always be placed on quality, and not on style.

5. Conclusion

In the last decade it has become clear that any laboratory which employs computers intensively for a variety of scientific applications should contain a library of accurate, efficient general-purpose numerical mathematical subroutines. If the purpose of the library is to serve a broad user community, then the subroutines in the library should be simple to use and comprehensive. Also, emphasis must be placed on reliability and portability, and redundant abilities should be kept to a minimum.

Currently all mathematical software libraries are deficient to some degree. This cannot be avoided, since there are possibly more numerical mathematical problems that have not satisfactorily been solved than have been solved. The evaluation/certification of subroutines for a high-quality library is an expensive and time consuming task. One reason for this is that the general evaluation techniques are only partially defined. Techniques which are appropriate for the examination of subroutines for a particular mathematical activity normally have to be specialized for the activity. Thus, it must be recognized that the development of high-quality software and software libraries requires a substantial investment.

One area in which subroutine and library development can be considerably simplified is that of software portability. Currently, the major higher-level programming languages contain deficiencies that seriously impact development. The deficiencies include a lack of standards for floating point arithmetics, and the lack of a complete set of operations for processing arithmatic data. The accumulative damage that these deficiencies have inflicted is incalculable. It is recommended that the deficiencies be eliminated.

References

1. American Standard Fortran (X3.9-1966), American National Standards Institute, New York, 1966.
2. American National Standards Institute, "Clarification of Fortran Standards - Initial Progress", Comm. ACM 12, 5 (May 1969), pp. 289-294.
3. American National Standards Institute, "Clarification of Fortran Standards - Second Report", Comm. ACM 14, 10 (October 1971), pp. 628-642.
4. American National Standard Programming Language Fortran (Fortran 77, ANSI X3.9-1978), American National Standards Institute, New York, 1978.
5. Cowell, Wayne, ed., Portability of Numerical Software, Workshop, Oak Brook, Illinois, June 21-23, 1976, Sringer-Verlag, 1977.
6. Cowell, W.R. and Fosdick, L.D., "Mathematical Software Production". In Mathematical Software III, John Rice, ed., Academic Press, 1977, pp. 195-224.
7. IMSL Library (Sixth Edition), International Mathematical and Statistical Libraries, Inc., Houston, Texas, 1977.
8. Jacobs, D., ed., Numerical Software - Needs and Availability, Academic Press, 1978.
9. Morris, A.H., NSWC/DL Library of Mathematics Subroutines, Technical Report TR - 3843, Naval Surface Weapons Center, Dahlgren, Virginia, 1978.
10. NAG Reference Manual (Mark 6), NAG Central Office, Oxford, England, 1978.
11. Rice, John, ed., Mathematical Software, Academic Press, 1971.
12. Smith, B.T., Boyle, J.M., et al., Matrix Eigensystem Routines - EISPACK Guide (Second Edition), Springer-Verlag, 1976.

DISTRIBUTION

Dr. Donald Amos
Division 5122
Sandia Laboratories
Albuquerque, New Mexico 87115

Dr. John R. Berg
Satellite Programs Branch
System Development Corporation
2500 Colorado Avenue
Santa Monica, California 90406

Alan B. Bligh
Code 7810
Naval Research Laboratory
Washington, D. C. 20375

C. M. Callahan
Code 760
Naval Coastal Systems Laboratory
Panama City, Florida 32401

Dr. Elizabeth Cuthill
Code 1805
Naval Ship Research and Development Center
Bethesda, Maryland 20084

Elaine Denton 41-41
System Development Corporation
2500 Colorado Avenue
Santa Monica, California 90406

Tom Galib
Code PA-4
Naval Underwater Systems Center
New London, Connecticut 06320

Gene H. Gleissner
Code 18
Naval Ship Research and Development Center
Bethesda, Maryland 20084

Gilbert Gray
Code 189
Naval Ship Research and Development Center
Bethesda, Maryland 20084

Lee E. Lakin
Code 303
Naval Weapons Center
China Lake, California 93555

Bernd Luftner
Rechenzentrum der Universitat Regensburg
Universitatsstrasse 31
8400 Regensburg
West Germany

Petro Matula
Code 1844
Naval Ship Research and Development Center
Bethesda, Maryland 20084

Lloyd Maudlin
Code 453
Naval Ocean System Center
San Diego, California 92152

Dr. Richard Nance
Department of Computer Science
562 McBryde Hall
Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24061 (2)

Charles Noble
EG and G Idaho, Inc.
P. O. Box 1625
Idaho Falls, Idaho 83401

Alvin Owens
Code 7906
Naval Research Laboratory
Washington, D. C. 20375

Harold Tremblay
Code 85
Naval Air Development Center
Warminster, Pennsylvania 18974

Defense Documentation Center
Cameron Station
Alexandria, Virginia 22314 (12)

Library of Congress
Washington, D. C. 20540
ATTN: Gift and Exchange Division (4)

Local:

E41
K
K10
K50
K60
K70
K74 (50)
N20
X21 (2)