Mathematics and Computers
(UC-32)

ANL-84-83

ARGONNE NATIONAL LABORATORY
9700 South Cass Avenue
Argonne, Illinois 60439

SECOND THOUGHTS ON THE MATHEMATICAL SOFTWARE EFFORT: A PERSPECTIVE

W. J. Cody

Mathematics and Computer Science Division

October 1984

Second Thoughts on the Mathematical Software Effort: A Perspective

W. J. Cody†

Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, Illinois 60439

Abstract. The mathematical software effort bridges the gap between the discovery of numerical algorithms and the consumption of numerical software. The spectrum of activities is surprisingly wide, including tasks often associated with numerical analysis, program design and testing, programming practices, language standards, documentation standards, software organization, distribution methods, and even the specification of arithmetic engines. This paper highlights the most important accomplishments in the field over the last twenty years. It also examines current problems and future challenges posed by the rapid advance of technology.

1. INTRODUCTION John Rice coined the term "mathematical software" in 1969, and focussed attention on the subject the following year with a symposium held as part of a Special Year in Numerical Analysis at Purdue University. The movement spawned by that first meeting has been fruitful. In 1969 only a few individuals worked on what we now call mathematical software, and only a few fortunate computer sites had access to decent numerical programs. Today many talented people work in the field, large collections of good numerical software are widely available, and specialized meetings are common.

A description of the mathematical software effort is difficult because it is so broad. Its domain is that nebulous region between the discovery of numerical algorithms and the consumption of numerical software. On the one hand numerical analysts devise new computational methods, and on the other hand individuals wish to apply effective methods to their immediate problems. It is the job of the mathematical software effort to bridge the gap by packaging numerical analysts' work in software appealing to the consumer. Strictly speaking, work on mathematical software is limited to tasks related to the implementation of numerical algorithms. In practice the spectrum of activities is surprisingly wide because the process of implementation is itself worthy of study. In addition to obvious concerns with program design and testing, there are major concerns with programming practices, documentation standards, software organization, and distribution methods. Other activities involve the development of programming tools to partially automate design, implementation, testing and maintenance of software, and work on the computational environment, including the design of arithmetic systems and programming languages properly supportive of good numerical software. Major contributions have been made in each of these areas by individuals who consider their primary interest to be mathematical software.

† This work was supported by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U. S. Department of Energy, under contract W-31-109-Eng-38.

The published proceedings of the Purdue meeting contain Rice's appraisal of the mathematical software effort as it stood in 1970 [Rice, 1971], including a chronological account of progress. This paper is a similar appraisal of the effort as it stands today. Instead of updating the chronological record, however, we discuss what we consider to be major milestones marking progress to this point. We then examine current problems in the field and future challenges posed by an advancing technology.

An earlier version of this work [Cody, 1984a] was inspired by a panel session on the same subject that ended the week-long International Seminar on Problems and Methodologies in Mathematical Software Production held November 1980 in Sorrento, Italy, under the sponsorship of the University of Naples and the C.N.R. [de Mao, 1982]. That version was outdated by the time it was published, however, so we have taken this opportunity to revise it to reflect the events of the past four years. Hence, the "second thoughts" in the title.

References have been modified to include the latest relevant publications. In particular, references for various "PACKS" and libraries have been changed to articles in [Cowell, 1984], which contains valuable historical and technical information on the mathematical software effort written by the participants, as well as additional references for those interested in deeper reading on a topic.

We gratefully acknowledge the contributions of our fellow panelists at the Sorrento meeting, B. Ford, T. J. Dekker, M. Gentleman, J. N. Lyness, and P. C. Messina, a responsive audience there, and an anonymous host of others who have since then unknowingly contributed to our thoughts on these matters. With the benefit of leisurely reflection we have reorganized and expanded some of their ideas and combined them with our own. We alone are responsible for the selection and expression of the opinions that follow, however.

The reader should be aware that the views presented here may be colored by personal bias, and that other views exist. The surveys and suggestions for research in Gear [1979], Huddleston [1979], Morris [1979], Rice [1979], and Rice *et al.* [1978] are especially recommended to the interested reader.

2. THE PAST Many people associate the beginning of the mathematical software effort with Rice's 1969 call for a meeting at Purdue University. The roots go back further, however. While it would be trite to trace them to the first numerical subroutine libraries, we detect an emerging concern for software quality in the early 1960's. By then individuals at the University of Toronto, The University of Chicago, Stanford University, Bell Laboratories and Argonne National Laboratory were critically examining software and advertising their findings through technical reports and discussions at computer user group meetings. The ideas and evaluation techniques were not well enough established for publication in refereed journals, however, and efforts were hampered by poor communications. Often workers at one location were completely unaware of similar work elsewhere. Yet each of these computing centers developed outstanding program libraries by contemporary standards.

In early 1966 J. F. Traub organized SICNUM, the Special Interest Committee on Numerical Mathematics. The group grew quickly; and by midyear, when the first informal *SICNUM Newsletter* appeared, it had a membership of almost 1000. Two articles in the first *Newsletter* typify SICNUM's interests. The first announced the establishment of a working group "to investigate testing and certification techniques for numerical subroutines," and the second announced a SICNUM-sponsored evening session at the 1966 National ACM Conference. The session included a panel discussion "in the area of machine implementation of numerical algorithms." By constantly emphasizing efforts to improve the quality of numerical software, SICNUM and its successor SIGNUM set the stage for Rice's 1969 call for a symposium.

In that call Rice [1969] defined mathematical software as "computer programs which implement widely applicable mathematical procedures." This contrasts with the definition he later included in the published proceedings, "the set of algorithms in the area of mathematics" [Rice, 1971]. These two definitions illustrate the fundamental confusion between algorithms and computer programs that plagued the early development of numerical software. The realization that an implementation is different from the underlying algorithm marks the emergence of mathematical software as a separate field of endeavor.

That difference was not widely understood in 1969. Despite early admonitions from G. Forsythe [1966] and other prominent researchers, most numerical analysts still believed that their work was finished when they had defined an algorithm. Programming was a job for programmers; numerical analysts programmed only when it was necessary for their research (and pure mathematicians never programmed). A university professor seeking advancement and tenure shied away from working on numerical software. As a result most of the early work was concentrated in government and industrial laboratories, with only a few selfless university people involved. Unfortunately, the same attitudes are still common. While work on mathematical software has gained some professional stature and there are more talented people involved in the effort today than were involved in 1969, many others still do not dare to become involved if they seek promotion. This is still especially true at many universities.

Three software projects that greatly influenced the mathematical software effort began about the time of the Purdue meeting. Each project — IMSL, NAG and NATS — resulted in a widely used collection of high-quality numerical software. Certain software collections were publicly available before this. Computer users' groups had organized program repositories by the early 1960's, and the IBM Scientific Subroutine Package (SSP) was available on the IBM 7094, for example. Although these collections contained a few good programs, their general reputations were deservedly notorious. The IMSL, NAG, and NATS collections were the first to combine quality with wide distribution.

IMSL (International Mathematical and Statistical Libraries, Inc.) was founded in 1971 [Aird, 1984] by some of the people involved in the IBM SSP effort. It delivered the first purely commercial numerical subroutine library to IBM customers a year later. By mid-1973 the library had also been delivered to UNIVAC and CDC customers, and for the first time the same library of numerical programs became available on a variety of computing equipment. This enabled numerical programmers to write and distribute applications programs without worrying about the availability of a decent support library. Today IMSL supports most major computers. The success of this venture is suggested by the number of computing centers now relying on IMSL and its competitors for their core library, thus freeing local personnel to develop the specialized programs necessary for their own work.

IMSL's main competition comes from the NAG and, to a lesser extent, PORT libraries. NAG, originally Nottingham Algorithms Group but now Numerical Algorithms Group, was organized in 1970 in Great Britain as a cooperative venture between universities using ICL 1906A/S computers [Ford and Pool, 1984]. Originally supported by heavy government subsidies, NAG extended its coverage to other machines and now has become self-supporting. The PORT library is a product of Western Electric arising from the early library work at Bell Laboratories [Fox, 1984]. It is not aggressively marketed and is therefore not as widely used as the IMSL and NAG libraries.

The NATS project, National Activity to Test Software, was conceived in 1970 and funded in 1971 by the National Science Foundation and the Atomic Energy Commission to study problems in producing, certifying, distributing and maintaining high-quality numerical software [Boyle *et al.*, 1972]. This was a cooperative effort between personnel at Argonne National Laboratory, Stanford University, The University of Texas at Austin, and scattered test sites to examine software production as a research problem. Intrinsic to this effort was the production of two software packages, the EISPACK collection of matrix eigensystem programs [Dongarra and Moler, 1984] based on Algol procedures published by Wilkinson and Reinsch [1971], and the FUNPACK collection of special function programs [Cody, 1984b]. The project formally ended with the distribution of extended second releases of both packages in 1976.

By any measure, the NATS project was a spectacular success. Not only did it produce superior software, but it also pioneered in organizational and technical achievements that are still being exploited. For example, the project developed an early system for automated program transformation and maintenance [Smith, Boyle, and Cody, 1974] that led directly to current research on the TAMPR system [Boyle and Matz, 1977]. We believe that the NATS aids were developed before similar aids for program transformation were developed at the Jet Propulsion Laboratory [Krogh, 1977] and within the IMSL [Aird, 1977] and NAG [Du Croz, Hague, and Siemieniuch, 1977] projects. They were certainly the first to be successfully used in a software project. Important as such technical achievements were, however, they were overshadowed by the organizational concepts the project developed. Machura and Sweet [1980] have stated, "The most important lesson learned from the EISPACK project is that the development and distribution of quality software can be achieved by the joint efforts of several different organizations." Before the NATS success software was typically developed with the limited resources of one organization; since the NATS success cooperative ventures have become common.

None of this would have mattered if the NATS software had not been superior. Fortunately, the software produced by the project was well received and is still considered to be among the best available. EISPACK, in particular, set and met high standards for performance, transportability, and documentation. It has become a paradigm for thematic numerical software collections with the term "PACK" now implying all that is good in numerical software. Attesting to EISPACK's influence, the following PACKs in addition to FUNPACK either exist or are in advanced planning stages: ELLPACK [Rice, 1977], FISHPAK [Swarztrauber and Sweet, 1979], ITPACK [Grimes *et al.*, 1978], LINPACK [Dongarra and Stewart, 1984], MINPACK [Moré *et al.*, 1984], PDEPACK [SCCS, 1975], QUADPACK [Piessens *et al.*, 1983], ROSEPACK [Coleman *et al.*, 1980], SPARSPAK [George and Liu, 1981], TESTPACK [Buckley, 1982], and TOOLPACK [Osterweil, 1983]. While many of these are superb packages, the use of a "PACK" name does not automatically instill quality. We return to this point later.

It is disappointing that the NATS experience was not fully exploited. Attempts to establish a central organization for software production based on the NATS concept [Cowell and Fosdick, 1975 and 1977] failed for various political and technical reasons. This denied segments of the numerical software community access to experienced people and important resources. Many of the projects mentioned above had to rely on their own resources to coordinate production, certification, and distribution of their software, duplicating similar capabilities already developed in other projects.

The first Purdue symposium was followed by three other important meetings. SIGNUM sponsored a meeting in 1971 in Ljubljana, Yugoslavia, concurrent with the 1971 IFIP Congress, that ultimately led to the establishment in late

1974 of WG 2.5, the IFIP Working Group on Numerical Software. Members of the working group now represent numerical software interests in language and hardware standardization efforts, often with detailed advice from the group as a whole. In addition, the working group has organized several international workshops on software topics and has drafted and published several technical reports.

The second important meeting was the second software symposium held at Purdue in 1974. While its influence was not as great as that of the first meeting, it did lead to the establishment of the *ACM Transactions on Mathematical Software* with John Rice as editor. Since its appearance in early 1975 with papers from the Purdue meeting, *TOMS* has complemented the *SIGNUM Newsletter* by providing an outlet for refereed numerical software papers.

The second Purdue meeting was also noteworthy for the first open discussion of the BLAS, or Basic Linear Algebra Subprograms [Lawson *et al.*, 1979]. As the name implies, the BLAS are a collection of Fortran subprograms implementing low-level operations, such as the dot product, from linear algebra. The project was originally organized in 1973 as a private effort to reach consensus on names, calling sequences, and functional descriptions for such programs, but it quickly became a cooperative effort officially sanctioned by ACM-SIGNUM. Once conventions had been agreed on, it was possible for linear algebra programs to do fundamental operations in a uniform way. This was already a significant accomplishment, but the group also prepared efficient implementations of the BLAS for most popular computers. The project's most important contribution, however, was the concept of establishing popular "conventions" as opposed to official standards. Language designers are reluctant to augment standard languages to include something useful to only a small group. Even if that is done, years pass before the new feature is available in compilers. The establishment of private conventions outside language standards is a more reasonable approach, and the BLAS project demonstrates that it is also a practical one. As with NATS, this lead has not been fully exploited.

A measure of the success of this effort is the continuing appearance of proposals for new BLAS packages. These currently include suggested extensions for sparse matrices, and for fundamental matrix, as opposed to vector, operations. Just as those in the NATS project have been concerned about the use of a "PACK" name for software that may not be of the highest quality, the architects of the BLAS are concerned about the threatened proliferation of new BLAS packages. Extensive public involvement with long consideration of the implications of each suggested member of the package was important to the success of the original BLAS effort. It is not obvious that the same care will be taken by those proposing extensions. Is the inevitable price of a successful package that perhaps less-than-worthy follow-ons using a similar name will diminish the sparkle on the original? It is a problem apparently without an acceptable solution; we do not believe in trademarking as DOD has done with Ada.

The third meeting of importance was the Oakbrook Workshop on Portability of Numerical Software held in 1976. Several of the technical presentations at this meeting enspired participants in the IFIP WG 2.5 meeting immediately following to petition the Fortran Standards Committee with proposals for changes in the impending draft standard. They were too late to have any effect on the Fortran 77 Standard, but their action ultimately led to active membership on X3J3 for representatives of SIGNUM and IFIP WG 2.5. Today, there are a handful of prominent mathematical software people on X3J3 contributing to the next draft standard.

It is difficult to assess the importance of events in the immediate past, but we believe that the recently proposed IEEE standards for floating-point arithmetic will prove to be important. One of the difficulties in producing mathematical software has been the diversity and overall shabbiness of mainframe arithmetic systems. High-quality software is supposed to be fail-safe and transportable; it must work properly regardless of quirks in the host arithmetic system. Software production is seriously hampered when computer arithmetic grossly violates fundamental arithmetic axioms such as the commutative and associative laws, and the existence of a multiplicative identity. Some systems have even worse defects, often involving illogical underflow and overflow. Anomalies such as these are usually traceable to engineering economies [Cody, 1982]. Mainframe designers traditionally ignore complaints about such mathematical atrocities, and new anomalies seem to appear with each new machine.

That is changing, however. By 1977 technology had advanced to the point where small microprocessor manufacturers considered adding floating-point arithmetic to their chips. In an unprecedented move they turned to numerical analysts for advice. The result was the formation of a subcommittee of the IEEE Computer Society, IEEE Working Group P754, to draft a standard for binary floating-point arithmetic. The final draft standard [Stevenson, 1982] specifies an arithmetic system that differs radically from previous systems. Not only is it free of anomalies, but it also contains new features specifically requested and designed by numerical analysts with software experience. While the draft has not yet become an official IEEE standard, it has become a *de facto* standard in the industry. Chips (see, e.g., [Palmer and Morse, 1984]) and computers (not all of them micros) based on the proposed standard are becoming common.

This work was so successful that the IEEE established a second working group, P854, to draft a radix and format-independent floating-point standard that would be upward compatible with the P754 draft. P854 Draft 1.0 has recently been published [Cody *et al.*, 1984] for public comment. Indeed, the first implementation of the draft has already appeared in a powerful hand-held computer [Hewlett-Packard, 1983]. Although the new draft is again intended for microprocessors, its inclusion of non-binary arithmetics should interest designers of larger equipment.

These, then, are the milestones leading to where we stand today: the early work at isolated computing centers; the establishment of SIGNUM; the two Purdue symposia; the establishment of commercial numerical software libraries; the NATS project and the EISPACK package; the establishment of IFIP WG 2.5 and of TOMS; the BLAS; the Oakbrook conference and the involvement of mathematical software people in drafting language standards, and the drafting of standards for floating-point arithmetic. Each of these events added something new and important to the movement. There have also been some disappointments. We mention in particular the failure to achieve full professional recognition for software work (especially at universities), the failure to fully exploit the NATS experience, and the general lack of progress in mainframe arithmetic design.

3. THE PRESENT Today we are faced with a hardware and software revolution that is rapidly changing what we can do and how we must do it. We are still concerned about the production of high-quality transportable software, but we expect more from such software than we did in the past. In addition, it must perform on machines with bizarre new architectures. Vector and parallel "supercomputers" and high-performance micros present new challenges that must be met in new ways.

High-performance micros are mostly scaled-down, but souped-up von Neuman machines that we understand. The only unusual feature appears to be the decent arithmetic; compilers are still unreliable on these machines, but we are used to that. Thus, there is no technical reason why good numerical software cannot be provided for these machines. IMSL, NAG and numerous others are busily addressing this market. The only question is how important the market will be. (We are privately guessing that it will be important.)

The "supercomputers" and new architecture machines are another matter, however. A bewildering assortment of prototype and production hardware, and of "paper machines," has descended upon us, differing in processor organization, in memory organization, and in schemes for communicating between processors. These are described using terms such as vector machines, array processors, parallel machines, data flow machines, hypercubes, pipelined machines, and MIMD machines. Floating-point operations are now as fast as memory access on many of these machines, and we can no longer ignore access problems when designing software. At first glance, writing high-performance software for a given machine is difficult, and making that software transportable across differing architectures is hopeless. Nevertheless, some pursue those goals with encouraging preliminary results.

There are two or three main themes to this work. One possibility is to first define and implement a virtual machine in software that is written in a high-level language, but can be tailored to a specific machine architecture. This is the approach now taken in TOOLPACK with TIE [Iles, 1984], and in automated reasoning with LMA [Lusk, McCune, and Overbeek, 1982]. Once software defining the virtual machine has been moved and made efficient, any software built on top of it becomes available. In a sense, this is a generalization of what the BLAS do; that is, once the BLAS have been moved and made efficient, linear algebra packages built on them can be imported.

Unfortunately, this approach may not be suitable for numerical work. New algorithms may be necessary to keep the machines busy at a respectable fraction of their potential performance. In the case of linear algebra, one approach to high-performance algorithms uses a handful of modular matrix-vector operations [Dongarra, 1984] as fundamental building blocks in much the same way that the BLAS were previously used. High-level modules of this sort can be designed to exploit parallel, pipelined, vector, or other architectures.

Additional difficulties are posed by parallel architectures where it is important to identify candidates for independent computations, then to schedule and coordinate them in such a way that available processors are used efficiently. Surprisingly few additional control structures are needed when scheduling and coordination are accomplished through monitors and macros [Lusk and Overbeek, 1984].

All of this work is new. The references cited are only indicative of a vigorous research field; they are not exhaustive, nor are the results obtained so far necessarily the best way to do things. Only time will tell. What is important is that the software lessons of the past have not been forgotten in the rush to understand these new machines. The game is still to achieve both performance and transportability without slighting either, despite dramatic differences in machine architecture. For example, it is even possible to make a pipelined MIMD machine give vector performance with proper programming [Sorensen, 1984].

Meantime, work continues on thematic numerical software packages such as those mentioned in the last section. We believe it is significant that most of the early success in any numerical software endeavor has involved linear algebra programs. EISPACK, LINPACK, the BLAS, and the matrix-vector modules are examples. It is true that linear algebra is a fundamental mathematical tool, and that good software for other problem areas is not likely to be produced until good linear algebra programs are ready. But it is also true that linear algebra long ago reached an algorithmic maturity that invited software production. The algorithms were well developed, well understood, and backed by error analysis that clearly displayed the limitations of software implementations.

Because the production of EISPACK required minimal algorithmic work, the producers could concentrate on recasting algorithms to enhance desirable software attributes. The effort thus produced a significant software package within three years of funding. In contrast, the MINPACK effort required about five years to produce its first small package. This lengthy development time reflects the difficulty of the task and is likely to be typical of future projects. As in many other fields, prominent researchers in optimization do not agree on the best algorithms; new methods frequently appear accompanied by confusing claims of superiority over existing methods and programs. The situation is common in a vigorous, dynamic research field, but it does not encourage the quick production of high-quality software. All the "easy" implementations may have been done already.

Despite these difficulties, we believe that some additional problem areas could be harvested for software now. We are frankly puzzled by the lack of an effort in ordinary differential equations, for example. Existing algorithms seem to be well-enough understood, but no group has emerged with the necessary dedication and support.

There is one other little-understood aspect of successful numerical software projects that we believe to be important. Part of the variation in quality in the numerous PACKs previously mentioned is due to an improper appreciation of a fundamental lesson from the NATS project. We stated above that linear algebra was in a good algorithmic position when the EISPACK work began. That does not mean the field was stagnant, however; new algorithms were being introduced. The project deliberately ignored new work because it felt that algorithms had to prove themselves before being included. Further, the project found that there is a one- to two-year delay between the completion of the first pass at software and its final release. This time is spent iteratively testing, revising, and documenting to ensure that the package does what it claims. Thus there must be a one- to two-year moratorium on the introduction of new material into the package. This simple discovery has far-reaching implications. Algorithmic researchers find it almost impossible to observe such a moratorium; they are intent on wide distribution of their latest discoveries. Further, they cannot effectively polish software they feel to be inferior. Therefore, control over software projects should be vested in individuals who understand and are dedicated to software production rather than in individuals who primarily produce algorithms. Algorithm producers should be involved in software packaging, but they should not control it.

This approach has another advantage. Software packages benefit from a uniformity of style that simplifies documentation and maintenance. As EISPACK demonstrated, different programs may contain large segments of code that can be rendered almost identical, e.g., by using similar variable names and identical labels. A package also benefits from a uniform philosophy for detecting and reporting errors. The necessary surgery to produce package uniformity is best done by someone with no particular attachment to the original programs.

Aside from algorithmic development, the most difficult problem facing us today is testing. There are two fundamentally different reasons for testing, hence two fundamentally different approaches. On the one hand, algorithm creators want to show that their creations are in some way superior to existing algorithms, and they approach performance testing as a contest. The tests they design specifically highlight whatever advantage the new algorithm may have;

there is usually no attempt to uncover weaknesses in the algorithm or its implementation.

On the other hand, the selection of software for general use requires complete performance evaluation. Usually some duplication of purpose is acceptable in building a library, for example, so the concern is more with eliminating unacceptable programs and in matching programs to problem characteristics than in determining the "best" program. Tests for this purpose should aggressively exercise a program in ways that will detect weaknesses, display strengths, explore robustness, and probe problem-solving ability. We liken this type of testing to a physical examination. Inevitably the results of such testing will be used to compare programs, but the original intent is that a program be examined in isolation to stand or fall on its own merits.

Designing and implementing test programs is an important numerical problem that has been neglected in the rush to produce software for other purposes. Software testing locates weaknesses and leads to improvements in the next software generation. Yet, except for the ELEFUNT package of transportable Fortran test programs for the elementary functions [Cody and Waite, 1980] and collections of test programs for optimization software [Buckley, 1982; Moré, Garbow, and Hillstrom, 1981], no thematic test packages exist to our knowledge. Some test materials are distributed with various PACKs mentioned earlier, but these are not intended for general use.

The trouble is that we know little about how to test most types of software. Accuracy tests, for example, are usually battery tests exercising programs on someone's haphazard collection of problems. Not only is this time-consuming, but there is little purpose behind what is done, and the mass of data gathered may be incomprehensible even to those who gathered it. We must find a better way. We must back off from the problem and critically examine what we are doing; every test should have a purpose. We must find understandable and useful ways to present test results. (Note in this regard the clever use of Chernoff faces [Chernoff, 1973] to summarize evaluations of software for solving systems of nonlinear equations [Hiebert, 1982].)

There are some leads in the literature that may prove useful. J. Lyness and J. Kaganove [1976] show that numerical software falls into two broad classes: precision-bound programs implementing methods that guarantee to produce results in a finite number of steps, and heuristic-bound programs implementing algorithms for which useful results are not guaranteed in a finite number of steps even with exact arithmetic. Programs in this second category are usually limited in accuracy by the algorithm and not by machine arithmetic.

The importance of this classification is that accuracy test results for precision-bound programs vary with the environment, while properly structured accuracy tests for heuristic-bound programs produce system-independent results when the accuracy achieved is sufficiently above machine limits. Thus certain types of accuracy tests need be done only once and only on one system.

But accuracy testing is just part of a complete test package; efficiency and robustness are also important. Where efficiency varies significantly from problem to problem, it is important to explore efficiency as a function of the problem space. In its most elegant form to date, efficiency testing has been combined with accuracy testing and parameterization of a problem space to produce "performance profiles." The prototype work on automatic quadrature programs [Lyness and Kaganove, 1977] produced curves combining probability of success and expected number of integrand evaluations as functions of requested accuracy for specific parameterized problem families. Curves for a problem family with features similar to those in a particular application should be useful in selecting a program for that application based on balancing requested accuracy and predicted cost against the probability of success. The concepts of software classification and performance profiles exemplify the abstract assault on evaluation procedures that we believe is essential to progress in this area. Except for Lyness [1979], these ideas have not been exploited beyond the work cited. We are not sure if the concepts are too complicated to be exploited, or if they have simply been overlooked.

Concerns for numerical software have spawned important work in other fields as well. For example, research on the TAMPR system [Boyle and Matz, 1977] for automated program transformation and maintenance was specifically motivated by early NATS work. TAMPR is intended to accept programs in certain standard languages, map them into abstract forms, make transformations on these abstract forms, and finally recover specific realizations of the transformed programs in standard languages again. The transformations are limited conceptually only by our ability to describe what must be done. An early version of the system was used to realize all versions of the LINPACK programs from complex single-precision prototypes, for example. This application included enforcing formatting conventions and selectively implanting either calls to BLAS or inline coding with BLAS functionality, depending on the particular target computer host. Ultimately the capabilities may include automatic translation from one programming language to another by simply specifying different source and target languages in the first and last steps.

TAMPR is only one of many useful tools now under development. The TOOLPACK project is working on an extensive collection of integrated software tools specifically designed to simplify the writing, testing, analyzing, and maintaining of numerical software in Fortran. The package is to include formatters similar to POLISH [Dorrenbacher *et al.*, 1976], static analyzers similar to DAVE [Osterweil and Fosdick, 1976] and PFORT [Ryder, 1974], dynamic analyzers similar to NEWTON [Feiber, Taylor, and Osterweil, 1980], as well as parsers, lexers, tools for instrumentation, text editors, and so on. The package is designed to be portable and internally consistent in data requirements. As mentioned earlier, portability is achieved by building the package atop a software virtual machine written in Fortran, the TIE package. When it becomes available, TOOLPACK could alter the way we work on mathematical software. Much depends on how easy the package is to transport and use in comparison to the more familiar means of software development. Release of a prototype version for evaluation and comment is tentatively set for late 1984.

We earlier mentioned the work of the IEEE on standardization of floating-point arithmetic for microprocessors. The fruits of this standardization effort are just becoming available. In the meantime we continue to write numerical software for existing computers. We can improve the portability of software by explicitly including environmental dependencies in the source code. There have been several attempts to establish a fundamental set of parameters describing arithmetic systems for this purpose. IFIP WG 2.5 published one proposal [Ford, 1978] that has proven unsatisfactory in many respects and has not been widely used. A second proposal [Brown and Feldman, 1980] related to Brown's model for floating-point arithmetic [Brown, 1981] has received important support in some areas. The entire arithmetic model is embedded in Ada [Wichman, 1981], for example, much to the consternation of some numerical analysts. We return to that in a moment. Still a third proposal is being considered by the ANSI X3J3 Fortran Standards Committee for inclusion in the next Fortran standard. The intent of this proposal is to define certain parameters and reserve their names in the same way that SIN is a reserved name. The parameter names would then be aliases for numerical values appropriate to the particular host environment.

The difference between this proposal and the Ada approach is that here only the names would be specified; the numerical values provided would be implementation-dependent. While the parameters would be based on a model of an arithmetic system, the model would not be imposed by the standard. Thus the details of the model used in a particular situation could be chosen to fit the circumstances. When portability is crucial, the model could be chosen to conservatively estimate machine parameters; when local performance is important, the model could be chosen to closely approximate the local system. Such flexibility is not available in the ADA approach, where the model specified must be conservative to be universal.

The activities and concerns just outlined are typical of the mathematical software effort today. The most exciting aspect is the rapid evolution of hardware from micros to supercomputers, and the accompanying effort to understand the software issues. Large software projects and ancillary activities aimed at improving the environment for software production and use thrive. We are not making much progress in testing methodology, however.

4. THE FUTURE We expect that the quantity and quality of numerical software will continue to increase and that the activities just described will flourish in the future. Advancing technology and even the present success of the numerical software effort pose problems that must be overcome, however.

The most significant problem we face plagues every technical field: communications. As we become more specialized, we lose touch with one another and especially with potential customers.

Good communications with customers is crucial. Superb software is worthless unless software consumers are persuaded to use it. It is not enough to make users aware of software existence, though that is a difficult task in itself; consumer lethargy must be overcome at the same time. Consumers are reluctant to modify running programs unless they are convinced that the software they are currently using is inferior enough to endanger their work and that the new software will remove that danger. Fortunately, the appearance of radical new machines has lessened total dependence on existing software, and we now have an opportunity to communicate with users who may listen. Publications in the technical literature have never solved this type of communications problem, however. The consumers we must reach are applications people who seldom read numerical analysis or mathematical software literature. We must find other ways to reach them.

Almost ten years ago the Albuquerque and Livermore branches of Sandia Laboratories inserted library monitors in their operating systems [Bailey and Jones, 1975] to provide information on who was using which routines and for what. That information led to improvement of the library and to personal contact when it appeared that a program was being misused, when program bugs were found, or when better programs became available.

That approach is not a universal answer to our problem, however, because we face a revolution in the way computers are being used. The small "personal" computer is common. While it is often originally acquired for monitoring experiments and gathering data, the temptation to use it for numerical purposes is strong. This is especially true when the cost of using a central computing facility grows and the "free" personal machine would otherwise sit idle. Such usage is not necessarily bad; we have seen that smaller machines are approaching the hardware capabilities of larger machines of only a few years ago. Software is the problem. Owners of such machines are frequently unaware of the good numerical libraries that are now becoming available. They, therefore, frequently write their own software or obtain it from friends. In this respect they operate as large computing centers did twenty or more years ago. The software movement has completely lost whatever contact it may have had with these users, and that contact will be difficult to regain.

One possibility may be to contribute to journals like *Byte* that many of these people read. While some of the articles in these journals are written by highly qualified people, much of the numerical advice is amateurish, reflecting techniques that lost favor long ago. We cannot legitimately complain about this situation unless we are willing as a profession to provide the proper advice and software through these journals. We must be the ones to initiate communications with the users.

Unfortunately, we are also losing communications with those who continue to use larger machines. Often the original motivation for numerical software work was provided by users with applications that were endangered by poor computer programs. As our effort has matured, many of us have become more concerned with software production for the sake of production and less concerned about the real needs of users. We have tended to communicate among ourselves and to neglect the users. Perhaps that behavior pattern is typical of a new field. We hope that it will change in our field.

On the plus side, we are seeing more of the intensive numerical work being done at remote sites where supercomputers are available. This concentrates the work at a few locations, thus providing the opportunity for a few competent numerical software people with dedication to influence many users.

At the technical level we have mentioned challenges posed by new computer hardware. The diversity is exciting for numerical software people. IEEE-style arithmetic systems, for example, must provide square root and mod functions, among others, that are as accurate as the usual arithmetic operations. Some implementations include square root in the hardware, where it becomes no more expensive to use than an ordinary division operation. This combination of speed and accuracy in square root coupled with other features must influence our selection of algorithms. I believe we will see dramatic changes in algorithms, software, and even computer languages as these new systems become common.

Overall we view the future with confidence and expectation. We will probably never satisfactorily solve the communications problem, but we expect that the quality of numerical software will continue to improve and that software production will become easier as new tools and hardware appear.

5. REFERENCES

Aird, T. J. [1977]. "The IMSL Fortran converter: an approach to solving portability problems." *Lecture Notes in Computer Science, Vol. 57, Portability of Numerical Software*. Ed. W. Cowell. Springer-Verlag, New York, pp. 368-388.

Aird, T. J. [1984]. "The IMSL Library." *Sources and Development of Mathematical Software*. Ed. W. R. Cowell. Prentice-Hall, Englewood Cliffs, New Jersey, pp. 264-301.

Bailey, C. B., and R. E. Jones [1975]. "Usage and argument monitoring of mathematical library routines." *ACM Trans. on Math. Soft.* 1:196-209.

Boyle, J. M., W. J. Cody, W. R. Cowell, B. S. Garbow, Y. Ikebe, C. B. Moler, and B. T.

Smith [1972]. "NATS, a collaborative effort to certify and disseminate mathematical software." *Proceedings 1972 National ACM Conference, Vol. II.* Association for Computing Machinery, New York, pp. 630-635.

Boyle, J. M., and M. Matz [1977]. "Automating multiple program realizations." *Proceedings of the M.R.I. International Symposium XXIV: Computer Software Engineering*. Polytechnic Press, Brooklyn, New York.

Brown, W. S. [1981]. "A simple but realistic model of floating-point computation." *ACM Trans. on Math. Soft.* 7:445-480.

Brown, W. S., and S. I. Feldman [1980]. "Environmental parameters and basic functions for floating-point computation." *ACM Trans. on Math. Soft.* 6:510-523.

Buckley, A. [1982]. "A portable package for testing minimization algorithms." *Lecture Notes in Economics and Mathematical Systems, Vol. 199, Evaluating Mathematical Programming Techniques*, Ed. J. M. Mulvey. Springer-Verlag, Berlin, pp. 226-235.

Chernoff, H. [1973]. "The use of faces to represent points in k-dimensional space graphically." *J. Amer. Stat. Ass.* 68:361-368.

Cody, W. J. [1982]. "Basic Concepts for Computational Software." *Lecture Notes in Computer Science, Vol. 142, Problems and Methodologies in Mathematical Software Production.* Ed. P. C. Messina and A. Murli. Springer-Verlag, Berlin, pp. 1-23.

Cody, W. J. [1984a]. "Observations on the mathematical software effort." *Sources and Development of Mathematical Software.* Ed. W. R. Cowell. Prentice-Hall, Englewood Cliffs, New Jersey, pp. 1-19.

Cody, W. J. [1984b]. "FUNPACK - a package of special function routines." *Sources and Development of Mathematical Software.* Ed. W. R. Cowell. Prentice-Hall, Englewood Cliffs, New Jersey, pp. 49-67.

Cody, W. J., J. T. Coonen, D. M. Gay, K. Hanson, D. Hough, W. Kahan, R. Karpinski, J. Palmer, F. N. Ris, and D. Stevenson [1984]. "A proposed radix- and wordlength-independent standard for floating-point arithmetic." *IEEE Micro.* 4(4):86-100.

Cody, W. J., and W. Waite [1980]. *Software Manual for the Elementary Functions.* Prentice-Hall, Englewood Cliffs, New Jersey.

Coleman, D., P. Holland, N. Kadeen, V. Klema, and S. C. Peters [1980]. "A system of subroutines for iteratively reweighted least squares computations." *ACM Trans. on Math. Soft.* 6:327-336.

Cowell, W. R., ed. [1984]. *Sources and Development of Mathematical Software.* Prentice-Hall, Englewood Cliffs, New Jersey.

Cowell, W. R., and L. D. Fosdick [1975]. *A Program for Development of High Quality Mathematical Software.* University of Colorado Department of Computer Science Report CU-CS-070-75.

Cowell, W. R., and L. D. Fosdick [1977]. "Mathematical software production." *Mathematical Software III*. Ed. J. R. Rice. Academic Press, New York, pp. 195-224.

de Mao, I. P. [1982]. "Panel session on the challenges for developers of mathematical software." *Lecture Notes in Computer Science, Vol. 142, Problems and Methodologies in Mathematical Software Production.* Ed. P. C. Messina and A. Murli. Springer-Verlag, Berlin, pp. 254-271.

Dongarra, J. J. [1984]. "Increasing the performance of mathematical software through high-level modularity." To appear in *Proceedings of the Sixth International Symposium on Computing Methods in Engineering and Applied Sciences, held in Versailles, France*. North-Holland.

Dongarra, J. J., and C. B. Moler [1984]. "EISPACK - a package for solving matrix eigenvalue problems." *Sources and Development of Mathematical Software*. Ed. W. R. Cowell. Prentice-Hall, Englewood Cliffs, New Jersey, pp. 68-87.

Dongarra, J.J., and G. W. Stewart [1984]. "LINPACK - a package for solving linear systems." *Sources and Development of Mathematical Software.* Ed. W. R. Cowell. Prentice-Hall, Englewood Cliffs, New Jersey, pp. 20-48.

Dorrenbacher, J., D. Paddock, D. Wisneski, and L. D. Fosdick [1976]. *POLISH, a Program to Edit Fortran Programs.* University of Colorado Department of Computer Science Report CU-CS-050-76 (Rev.).

Du Croz, J. J., S. J. Hague, and J. L. Siemieniuch [1977]. "Aids to portability within the NAG project." *Lecture Notes in Computer Science, Vol. 57, Portability of Numerical Software.* Ed. W. Cowell. Springer-Verlag, New York, pp. 390-404.

Feiber, J., R. N. Taylor and L. J. Osterweil [1980]. *NEWTON - A Dynamic Testing System for Fortran 77 Programs; Preliminary Report.* University of Colorado Department of Computer Science Technical Note.

Ford, B. [1978]. "Parameterization of the environment for transportable numerical software." *ACM Trans. on Math. Soft.* 4:100-103.

Ford, B., and J. C. T. Pool [1984]. "The evolving NAG library service." *Sources and Development of Mathematical Software*. Ed. W. R. Cowell. Prentice-Hall, Englewood Cliffs, New Jersey, pp. 375-397.

Forsythe, G. [1966]. "Algorithms for scientific computation." *Comm. ACM* 9:255-256.

Fox, P. [1984]. "The PORT mathematical subroutine library." *Sources and Development of Mathematical Software*. Ed. W. R. Cowell. Prentice-Hall, Englewood Cliffs, New Jersey, pp. 346-374.

Gear, C. W. [1979]. *Numerical Software: Science or Alchemy?* University of Illinois Department of Computer Science Report UIUCDCS-R-79-969.

George, A., and J. W. Liu [1981]. *Computer Solution of Large Sparse Positive Definite Systems.* Prentice-Hall, Englewood Cliffs, New Jersey.

Grimes, R. G., D. R. Kincaid, W. I. MacGregor, and D. M. Young [1978]. *ITPACK Report: Adaptive Iterative Algorithms Using Symmetric Sparse Storage.* Center for Numerical Analysis, University of Texas at Austin Report CNA-139.

Hewlett-Packard [1983]. *HP-71 Owner's Manual.* Hewlett-Packard, Corvalis, Oregon.

Hiebert, K. L. [1982]. "An outline for comparison testing of mathematical software — illustrated by comparison testings of software which solves systems of nonlinear equations." *Lecture Notes in Economics and Mathematical Systems, Vol. 199, Evaluating Mathematical Programming Techniques.* Ed. J. M. Mulvey. Springer-Verlag, Berlin, pp. 214-225.

Huddleston, R. E., ed. [1979]. *Program Directions for Computational Mathematics.* Unnumbered report, Dept. of Energy, Washington, D.C.

Iles, R. M. J. [1984]. *What is TIE?* Numerical Algorithms Group Report NAG/T12-W1T.

Krogh. F. T. [1977]. "Features for Fortran portability." *Lecture Notes in Computer Science, Vol. 57, Portability of Numerical Software.* Ed. W. Cowell. Springer-Verlag, New York, pp. 361-367.

Lawson, C. L., R. J. Hanson, D. R. Kincaid and F. T. Krogh [1979]. "Basic linear algebra subprograms for Fortran usage." *ACM Trans. on Math. Soft.* 5:308-323.

Lusk, E., W. McCune, and R. Overbeek [1982]. "Logic machine architecture: inference mechanisms." *Lecture Notes in Computer Science, Vol. 138, Proceedings of the Sixth Conference on Automated Deduction.* Ed. D. W. Loveland. Springer-Verlag, New York, pp. 85-105.

Lusk, E., and R. Overbeek [1984]. *Use of Monitors in FORTRAN: A Tutorial on the Barrier, Self-scheduling DO-Loop, and Askfor Monitors.* Argonne National Laboratory Report ANL-84-51.

Lyness, J. N. [1979]. "A bench mark experiment for minimization algorithms." *Math. Comp.* 33:249-264.

Lyness, J. N., and J. J. Kaganove [1976]. "Comments on the nature of automatic quadrature routines." *ACM Trans. on Math. Soft.* 2:65-81.

Lyness, J. N., and J. J. Kaganove [1977]. "A technique for comparing automatic quadrature routines." *Computer J.* 20:170-177.

Machura, M., and R. A. Sweet [1980]. "A survey of software for partial differential equations." *ACM Trans. on Math. Soft.* 6:461-488.

Moré, J. J., B. S. Garbow, and K. E. Hillstrom [1981]. "Testing unconstrained optimization software." *ACM Trans. on Math. Soft.* 7:17-41.

Moré, J. J., D. Sorensen, B. S. Garbow, and K. E. Hillstrom [1984]. "The MINPACK project." *Sources and Development of Mathematical Software.* Ed. W. R. Cowell. Prentice-Hall, Englewood Cliffs, New Jersey, pp. 88-111.

Morris, A. H., Jr. [1979]. *Development of Mathematical Software and Mathematical Software Libraries*. Naval Surface Weapons Center Report NSWC TR 79-102.

Osterweil, L. J. [1983]. "TOOLPACK - an experimental software development environment research project." *IEEE Trans. on Software Engineering*, 9:673-685.

Osterweil, L. J., and L. D. Fosdick [1976]. "DAVE —a validation, error detection and documentation system for Fortran programs." *Software Practice and Experience* 6:473-486.

Palmer, J. F., and S. P. Morse [1984]. *The 8087 Primer*. John Wiley and Sons, New York.

Piessens, R., E. De Doncker, C. Uberhuber, and D. K. Kahaner [1983]. *Springer Series in Computational Mathematics, Vol. 1, QUADPACK - A Subroutine Package for Automatic Integration*. Springer-Verlag, New York.

Rice, J. R. [1969]. "Announcement and call for papers, mathematical software." *SIGNUM Newsletter* 4 (3):7.

Rice, J. R., ed. [1971]. *Mathematical Software*. Academic Press, New York.

Rice, J. R. [1977]. "ELLPACK: a research tool for elliptic partial differential equations software." *Mathematical Software III*. Ed. J. R. Rice. Academic Press, New York, pp. 319-341.

Rice, J. R. [1979]. "Software for numerical computation." *Research Directions in Software Technology*. Ed. P. Wegner. MIT Press, Cambridge, Massachusetts, pp. 688-708.

Rice, J. R., C. W. Gear, J. M. Ortega, B. N. Parlett, M. Schultz, L. F. Shampine, and P. Wolfe [1978]. *Numerical Computation, Panel Report for the COSERS Project*. Special issue *SIGNUM Newsletter*.

Ryder, B. G. [1974]. "The PFORT verifier." *Software Practice and Experience* 4:359-377.

SCCS [1975]. *PDEPACK: Partial Differential Equations Package User's Guide*. Scientific Computing Consulting Services, Manhattan, Kansas.

Smith, B. T., J. M. Boyle, and W. J. Cody [1974]. "The NATS approach to quality software." *Software for Numerical Mathematics*. Ed. D. J. Evans. Academic Press, New York, pp. 393-405.

Sorensen, D. [1984]. "Buffering for vector performance on a pipelined MIMD machine." To appear in *Parallel Computing*.

Stevenson, D., Chairman IEEE P754 [1982]. "A proposed standard for binary floating-point arithmetic, draft 10.0," *IEEE Floating Point Subcommittee Working Document P754/82-8.6*. A copy of this draft is available from R. Karpinski, U-76, University of California, San Francisco, CA 94143. Draft 10.0 supercedes Draft 8.0 published as "A proposed standard for binary floating-point arithmetic," *Computer* 14 (3):51-62.

Swarztrauber, P., and R. Sweet [1979]. "Efficient FORTRAN subroutines for the solution of separable elliptic equations. Algorithm 541." *ACM Trans. on Math. Soft.* 5:352-364

Wichman, B. A. [1981]. *Tutorial Material on the Real Data-Types in ADA.* Final Technical Report, U.S. Army European Research Office, London.

Wilkinson, J. H., and C. Reinsch, eds. [1971]. *Handbook for Automatic Computation, Vol. II, Linear Algebra.* Springer-Verlag, New York.