1977

Software for Numerical Computation

John R. Rice
Purdue University, jrr@cs.purdue.edu

Report Number:
77-214

SOFTWARE FOR NUMERICAL COMPUTATION

John R. Rice

Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

CSD TR #214
January 1977

SOFTWARE FOR NUMERICAL COMPUTATION

John R. Rice
Mathematical Sciences
Purdue University

CSD-TR 214

January 12, 1977

Article to appear in the book: Research Directions in Software Technology.

SOFTWARE FOR NUMERICAL COMPUTATION

John R. Rice
Mathematical Sciences
Purdue University

I. INTRODUCTION AND MOTIVATING PROBLEMS.

The purpose of this article is to examine the research developments in software for numerical computation. Research and development of numerical methods is not intended to be discussed for two reasons. First, a reasonable survey of the research in numerical methods would require a book. The COSERS report [Rice et al, 1977] on Numerical Computation does such a survey in about 100 printed pages and even so the discussion of many important fields (never mind topics) is limited to a few paragraphs. Second, the present book is focused on software and thus it is natural to attempt to separate software research from numerical computation research. This, of course, is not easy as the two are intimately intertwined.

We want to define numerical computation rather precisely so as to distinguish it from business data processing, symbolic processing (such as compilers) and general utilities (such as file manipulation systems or job schedulers). We have the following definition: Numerical computation involves real numbers with procedures at a mathematical level of trigonometry, college algebra, linear algebra or higher. Some people use a somewhat narrower definition which restricts the term to computation in the physical sciences and a few people even think of numerical computation as research and development computation (as opposed to production) in science.

There are two principal sources of the problems in numerical computation: Mathematical models of the physical world and the optimization of models of the organizational world. The scope and range of the sources and the associated software is illustrated by the following list:

1. Simulation of the effects of multiple explosions.

   The software is a very complex program of perhaps 20,000 Fortran statements. It is specially tailored to this problem and may have taken several years to implement. The program requires all the memory and many hours of time on the largest and fastest computers.

2. Optimization of feed mixtures for a chicken farmer.

   This is standard software of modest length (500-2000 statements) even with an interface for a naive user. It might take substantial time to execute on a small computer.

3. Analysis of the structural vibration of a vehicle.

   The software is similar to that of example 1. One might also use NASTRAN (see II.G.3) with only a few months for an implementation. More computer time and memory would be used by this approach.

4. Simple linear regression on demographic data (e.g. age or income).

   This is standard software, but classical algorithms are neither reliable nor robust. Modern algorithms are short (200-400 statements) and execute quickly except for exceptionally large data sets.

5. Optimization of the design parameters of a gyroscope.

   A mathematical model of a complex physical system is required and then optimization algorithms are applied. Determination of the gyroscope performance for a single set of parameters might involve the solution of a system of partial differential equations. Considerable human interaction is probably used to avoid astronomical computer costs and yet achieve some reasonable progress toward the optimum.

6. Calculation of the capacity of the wing tank of a jet liner.

   This is a simple problem except for the complex geometry of the wing tank. Once the wing tank is broken into simple pieces (probably by a person)

then standard algorithms are reliable, short and efficient. The automatic processing of the complex shape requires much more sophisticated software of moderate size (perhaps 2000 statements), but still gives a short calculation.

The creators and users of this software have very diverse backgrounds. Some are ultra-sophisticated scientists while others are just naive users of "black boxes" and canned programs. The size of the software ranges from the short routines for Fortran functions like sine and logarithm to elaborate systems which require hundreds of man years of programming and strain the capacity of the largest computers. Much of this software is written by people outside the computer science community, by people who call themselves engineers or physicists instead of programmers. The style is a combination of engineering art, mathematical science and hope. I have observed the following fact: Many sophisticated scientists produce naive software just as many sophisticated computer programmers produce naive science. Experts from each group, of course, are reluctant to acknowledge this state of affairs. This fact must be kept in mind when considering the totality of numerical computation.

Current folklore in Computer Science has it that numerical computation is a small, probably negligible, portion of the total computer usage. I have made a reasonable effort to determine the proportion of computing expenditures from numerical computation and I can attest to the difficulty of getting reliable data on just what computers are dong. However, I have concluded that the folklore is wrong. For example, suppose one divides the Department of Defense computing into four categories: numerical, non-numerical, dedicated and non-programmable (e.g. process control) and general support (e.g. compilers and operating systems). Then view the general support as overhead to be

distributed proportionally over the applications areas. My best estimate is that numerical computation then accounts for about 75% of the Department of Defense computing expenditures. Similarly, I believe that numerical computation accounts for about 50% of the computing expenditures in the United States. I will not reproduce the data upon which I base this conclusion, but I do want to emphasize the need for more reliable information in this area.

Finally, I want to note a feature of software for numerical computation that is growing and of crucial importance. It provides a means for the rapid dissemination of knowhow through the scientific community. Traditionally, research results are disseminated in a three stage process of technical journal articles, research monographs and surveys and, finally, textbooks. The scientist studies these publications in order to absorb the knowledge which he then applies to his particular problems. Now many important kinds of knowledge (mostly the "knowhow" or techniques variety) can be (and is being) incorporated into software. Then research results can be used (and quickly too!) by scientists without needing to absorb the knowledge themselves. This may one day be the aspect of software that has the most significant impact.

II. SIGNIFICANT RESULTS AND MILESTONES

II.A. The Program Library Concept. This was an early concept [Wilkes et al, 1951], but it is still very important. This concept has been surprisingly difficult to bring to fruition in the same sense as a library of books. That is to say, widely available and good quality libraries for basic mathematical procedures did not become available until the 1970's and even now most computer users lack access to a good library of programs for numerical computation. This is inspite of expensive efforts by Share, IBM (the SSP and SL-Math Libraries) and other computer manufacturers.

II.B Higher Level Languages: Fortran and Friends. The library concept is based on the fact that many problems are of a somewhat standard nature and occur in many different contexts. This is especially true of numerical computation because scientists and engineers use the language of mathematics in their analysis. The methods one uses seem to be independent of the particular computer and thus expressable in some machine independent language. Fortran, Algol and their descendants have made it possible to attempt to develope the science, art and body of numerical computation software. These languages are not perfect as the issues of portability discussed later show, but their introduction was an essential milestone in computing.

II.C. The Critical Evaluation of Software. Professionals in numerical computation have always had their favorite methods for various kinds of problems. Occasional surveys showed that there was no consensus among the experts as to which methods were best [Krogh, 1972]. Even worse, for many years, most people did not distinguish between a somewhat vague method and a computer program implementation of the method. Now people realize that the implementation (software) is as critical as the method, as there can be (and have been) terribly poor implementations of good methods.

There are two main variables here: different implementations of the same method and different methods for the same problem. It was not at all easy to design frameworks in which meaningful comparisons could be made. However, in the late 1960's such comparisons were started for ordinary differential equation software [see, for example, Hull, 1972] and now the framework for this particular area is well defined [Shampine, 1976]. Since then there have been significant accomplishments in evaluating software for numerical integration, special functions, linear algebra and polynomial root finding. One can now state with confidence what the "state-of-the-art" is for software in these areas and back these statements with scientific and quantitative measures of performance.

There has been one rather sobering result from all of these critical evaluations: the "experts" were very poor at predicting which methods would be the best. Experts in other areas of computation would do well to take heed of this experience.

Finally, I would note the lack of the use of "program proof methods" for software for numerical computation. Some reasons for this are (i) it is difficult to incorporate the uncertainties of round-off into proofs, (ii) the software tends to be too long for current proof methods, (iii) most numerical computation software has parts whose performance cannot be specified in terms

of input-output relationships. That is to say, the question is not whether the program is right or wrong, but rather how well it performs. One can make the analogy with a program for playing chess. Even though there may exist a right move in all situations, no one knows what it is and the program's choices for moves must be judged on how well the program fares in chess competition.

One can modify the idea of program proof in a useful way in some such situations. For example, one could prove the following assertion about a subprogram:

"This code finds a value of x so that ABS(F(x)) < EPS
or it returns a value of 1 for the argument IFAIL"

Such an assertion may be critical information for establishing the reliability of a program, but it does not give any clue as to when the subprogram might produce correct results (assuming that F(x) less than EPS is required for correct results). One might attempt to find a hypothesis on F(x) so as to ensure that the case of ABS(F(x)) < EPS holds. Such hypotheses tend to involve mathematical properties of F(x) which cannot be verified in practice or which restrict the applicability of the proof to a trivial or degenerate set of input. Only verifiable hypotheses are of value in program correctness proofs.

II.D. Problem Space Definition. The critical evaluation of software brings one to the question: what problems (input data) is this program supposed to process well. The concepts of software robustness, reliability and quality are meaningless without a careful definition and description of the space of problems to which the software is to be applied. The preceding discussion of program proofs indicates the delicate nature of this question for numerical software. The classical mathematical framework (e.g. the fourth derivative is continuous, or is bounded by 7.) have proved to be of little use for most

software. This is because the hypothesis used are either not verifiable or apply to an extremely restricted problem space. For some problem areas the problem spaces being used are somewhat ad hoc, but even these can be used successfully if one realizes the situation. The problem space concept is most developed for numerical integration software [Lyness and Kaganove, 1976] where the idea of performance profile was first introduced.

As an example, consider numerical integration software which is to compute

$$\int_a^b f(x)dx$$

It is a well known mathematical fact that this is a computationally unsolvable problem. The set of functions f(x) (or space of problems) must be restricted somehow before it is possible to even consider reliable software. The standard textbook approach is to restrict attention to functions which have a certain high derivative bounded by some (unknown) constant. This is a typical example of a non-verifiable hypothesis. The current direction is define the problem space that software for this problem should accept reliably a function which, in intuitive terms,

has only a few oscillations of a gross nature

has only a few singularities

the singularities are like $\sqrt{x}$ or $\sqrt[3]{x}$ or $\sqrt[4]{x}$ or ---

has a limited amount of high frequency, regular oscillation

The crux is to make these intuitive concepts precise, then one can quantitatively parameterize the problem space (by features like number of oscillations, number of singularities, strength of singularities, frequency of regular oscillation, etc.) and measure the performance of software as a function of these parameters. Robust software would fail gracefully as the boundaries of the problem space are approached.

II.E. The Importance of Human Engineering. Everyone agrees that the human engineering of software is important. It's just that so few people do anything about it. There have been instances of numerical software that was widely used because they had good human engineering even though the results computed were unreliable. These and other experiences have convinced many (but far from most) developers of numerical software that the human engineering (user convenience) aspects are critical. This is, in itself, a milestone; unfortunately, there have been few advances in how to do human engineering. It still seems to take a lot of hard, patient work.

II.F. Portability. While everyone recognizes the potential savings from distributing good software, it has been hard to achieve even when good, usable software is written. The dependency of numerical computation software on machine word length as well as the idiosyncracies of compilers and operating systems pose formidable barriers to the dissemination of quality software. It has been shown [Parlett, 1975] that portability and top efficiency cannot be achieved simultaneously in a high level language like Fortran because of compiler variations. A 100% loss in efficiency may be an acceptable price to pay for portability in some instances, but there are even more severe problems with error handling, precision changes and arithmetic unit behavior. These difficulties have been isolated and methods found to overcome them in an automated system. The solutions involve the use of subsets of standard Fortran like PFORT [Ryder, 1974] and systems which tailor programs for a

particular target environment (machine, operating system and compiler) [Aird, Battiste and Gregory, 1977], [Ford and Sayers, 1976]. It is now possible to have a collection of 500 programs and to automatically produce versions of them that are reliable and efficient for a wide variety of environments.

II.G. Examples of Significant Software for Numerical Computation. We do not try to give a "top 10" of the most important or best software. Rather each example illustrates a significant class of numerical software or a step in its developments.

1. Kuki's functions for IBM Fortran. [Kuki, 1967] This was the first really good software for the elementary functions of Fortran which was adopted by a manufacturer. It was common in the 1960's for such software to be grossly inaccurate for several years after a new computer was introduced. Much of this software is still unreliable, inefficient and/or inaccurate even though it is "well known" how to do it well.

2. Ordinary Differential Equations Programs. The critical evaluation in the late 1960's of programs for initial value problems in ordinary differential equations produced several codes which were truly superior to most of their contemporaries [Shampine, 1976]. These codes have been steadily improved since then and are now an order of magnitude better than the code that even a knowledgeable person can write with a reasonable effort. This software illustrates the tight interplay between software and methods. This improvement came both from new methods and techniques and from better implementation to achieve reliability and robustness.

3. NASTRAN, a structural engineering package. This is an example of the large specialized application package which we can expect to proliferate in the future. This system is large (about 200,000 Fortran cards); it is

complex (1200 page primer); it uses lots of memory (it can handle matrices where one row will not fit in core); it uses lots of time (runs of 5 or 10 hours on big machines are not unheard of) and it is widely used on a variety of computers. Mathematically speaking, it simply solves the biharmonic equation; its size comes from the fact that it does this on a domain which is a supertanker, an automobile or a jet fighter. Thus the mathematical difficulties are only one of several challenges for this software.

4. EISPACK - a systematized collection of programs for eigenvalue problems. [Smith et al, 1976]. This was the first project of the NATS (National Activity to Test Software) project. It started with some Algol programs of J. H. Wilkinson and produced a set of Fortran programs that were exhaustively tested and extensively documented. The result is reliable, efficient and robust software that runs on a variety of machines. This software is discussed in more detail in the next section.

5. SPSS - Statistical Package for the Social Sciences. This is one of several widely used statistical computing systems. It is designed to be used by non-programmers, even non-statisticians and provides a wide variety of statistical procedures and associated data handling facilities. The statistical computations in these packages are now implemented with quality software. Thus the naive user gets the right numbers, even if he might not know quite what they mean.

6. PDEONE - a program for partial differential equations. [Sincovec and Madsen, 1975] This is a short program (less than 100 lines of Fortran) that provides a surprisingly versatile tool for solving time dependent partial differential equations. Its approach is to build directly on the quality ordinary differential equations software (see 2. above) via the method of lines. While good

programs have been incorporated into libraries or large software packages, this is one of the first instances of a program that explicitly builds on existing, independently produced software.

7. Jenkins-Traub Polynomial Root Finder. [Jenkins and Traub, 1975], [Jenkins, 1975] Computing the roots of polynomials is one of the oldest problems in numerical computation and literally dozens of methods have been proposed. A few of these methods form the basis for quality software, but still most software for this problem available in the 1960's was unreliable or inefficient or both. This particular algorithm incorporates a new method (really a synthesis of older methods) into software where care has been taken about those factors which determine quality. This program has made literally hundreds of programs obsolete.

8. Good Numerical Libraries. Two organizations have produced large, portable, good quality and inexpensive libraries. They are IMSL (Inter. Mathematical and Statistical Libraries, Houston, Texas) and NAG(Numerical Algorithms Group, Oxford, England). Both organizations are discussed in more detail in the next section.

9. Software for Roundoff Analysis [Miller, 1975] This software provides automated support to locate numerical instabilities in various numerical processes. When these instabilities are triggered (usually by roundoff), they cause unreliable results. Thus it is essential to locate them and, unfortunately, formal analytical methods are very tedious and often impossible to carry out. Thus a software tool to help discover them is very valuable.

II.G. Numerical Computation Research Achievements. The emphasis in this paper is not on numerical methods, but they are intimately associated with software. A panel [Rice et al, 1977] has chosen the most significant accomplishments in numerical computation research and it seems appropriate to list their choices

here without elaboration.

1. Simplex Method in Linear Programming
2. Analysis of Iterative Methods for Partial Differential Equations
3. Fast Fourier Transform
4. Splines and Piecewise Polynomial Methods for Curves and Surfaces
5. Finite Element Method for Partial Differential Equations
6. Stability Analysis for Time Dependent Partial Differential Equations
7. Backward Error Analysis
8. QR-Algorithm for Eigenvalue Problems
9. Variable Metric and Quasi-Newton Methods in Optimization
10. Adaptive Numerical Integration
11. Techniques and Software for Ordinary Differential Equations
12. Sparse Matrix Techniques

Note that only one of these involves software directly.

III. FOUR NUMERICAL SOFTWARE PROJECTS.

In this section we discuss four long term, substantial projects to develop numerical software. They are, in alphabetical order, the IMSL library of International Mathematical and Statistical Libraries, Inc., the NAG library of the Numerical Algorithms Group, NATS (National Activity to Test Software) and the PORT library of the Bell Telephone Laboratories. For each of these we describe the objectives, make some comments on the history and size of the organization and describe the current status of the software. Further, we discuss the approaches used to achieve portability and quality and, possibly, roughly assess the costs of the software.

III.A The IMSL Library. IMSL was organized in 1970 to produce a high quality library of mathematical and statistical subroutines. The aim was to have a low cost and resultant high volume in order to achieve a commercial success.

IMSL has a few experienced senior people supported by some programmers and other staff for a total size of 20-25 people. They also have an advisory board of 12-15 experts who give them both general and specific advice.

The initial IMSL library had about 200 subroutines and was available in IBM, CDC and UNIVAC versions. The library now has over 400 subroutines and is available for seven computers: Burroughs, CDC, DEC, Honeywell, IBM, UNIVAC and Xerox. The library is leased at a cost of about $100/month and is updated every 12-18 months. The library subscription includes consulting service for any problems that may arise. Note that the annual cost of this rather complete library is substantially less than the cost of developing one typical program for such a library.

IMSL uses a variety of sources for programs (including writing their own from scratch) and all programs are rewritten with a uniform style. Quality control is exercised by (a) choosing good sources (the advisors assis in this regard) (b) using knowledgeable programmers with good supervision (some of the senior IMSL people work regularly on the library programs) (c) testing (reasonably exhaustive for new programs, check point testing for maintenance or new machine versions) and (d) continual upgrading. There is, of course, no way to produce 200 programs with a few people in a year or two without some inefficiencies, poor design, etc. Even so, the IMSL library was initially a substantial improvement over libraries available from manufacturers or those existing in a typical good computing installation. The current library is both substantially larger and better than the initial one and has about 480 installations in over 20 countries.

IMSL originally maintained separate, but closely related, versions for each program in the different machine versions of the library. They are now moving to a "Fortran converter" system where a master deck contains all the information needed for each machine version [Aird et al, 1977].

Much of the standard information is not explicitly in the deck. A converter program then automatically produces the program for a particular target machine. The master deck is itself a Fortran program that runs on one of the machines.

IMSL spent about $375,000 to develop their first library and accumulated a total deficit of about $1 million before reaching profitability in 1976. They estimate that it costs about $15 per Fortran statement to develop, test and document a new library routine. It costs them $2-$3 per Fortran statement to convert (by hand) a program from one machine to another. The Fortran converter is expected to increase slightly the cost of preparing the first (master) version of a new library routine and substantially reduce the cost of other versions.

III.2 The NAG Library. The NAG (Numerical Algorithms Group, formerly Nottingham Algorithms Group) project was initiated in 1970 by a group of English universities to produce a high quality numerical algorithms library for general university use on the ICL 1906A. The library was to have equivalent Fortran and Algol 60 versions. The NAG project is now run from a central office at Oxford University with associated individuals in five other universities comprising a full-time staff of 22. The project is a collaborative one between the English universities and government research laboratories (notably NPL and Harwell) and involves 120 people in part-time and voluntary capacities. The original aim has been broadened and emphasis given to creating a transportable numerical algorithms library. NAG is now a special kind of non-profit corporation with the goal of becoming financially self-supporting by renting the library to users in industry or outside England.

The late 1971 NAG library had just under 100 subroutines (in each of Fortran and Algol 60) and the current version (Mark 5) has over 300 subroutines.

Implementation of an Algol 68 version of the library is underway. Machine implementations include Burroughs, CDC 6000/7600, IBM 360/370, ICL 1900/1906/4100/System 4/KDF9, Honeywell, Siemens, Univac, Telefunken, Prime, Varian and DEC. All English universities now support the NAG library involving about 25,000 users and just prior to the January 1977 announcement of world wide availability, there were 40 installations on four continents using a preliminary library service.

Programs are obtained from a contributor (usually an expert from one of the cooperating universities or research establishments) who chooses the method and then writes, tests and documents the program. The program is then given to a validator who is also an expert in the relevant area. He is to critically examine the merit of the algorithm and test the usuability of the program and its documentation. Once a program is validated for general merit, it is then validated by the NAG central office as regards formatting, language standards, etc. Various software aids are used for this second stage of validation. See [Ford and Sayers, 1976].

NAG uses a master library file system which contains all versions of each program along with its complete history. Each target machine for the NAG library has a coordinator who is responsible for implementing contributed programs on their particular machine. This implementation process could be non-trivial, especially for Algol 60 programs. This problem has been eliminated in the latest release by new choices of language subsets (of Fortran, Algol 60 and Algol 68) and machine parameterization. New machine implementations are now essentially automatic. When an implementation is accepted, the programs are returned to the NAG central office for inclusion in the master library. There are stringent test programs for each library routine to assure equivalent performance of the NAG library versions. [See Prentice, 1974]. The history information and test programs in the master file have

been found useful in developing a more portable library [Hague and Ford, 1976].

The manpower used in the major aspects of the NAG project from its inception until 31st May 1976 are shown in Table 1.

TABLE 1: MANPOWER USED IN THE NAG PROJECT (in man - years)

Time Period / Aspect	1/6/70 to 31/5/73	1/6/73 to 31/5/74	1/6/74 to 31/5/75	1/6/75 to 31/5/76	Totals
Library Contribution	23	7.5	16	13.5	60
Library Implementation	7	15	10	8	40
Central Office	4	9	11	17	41
Algol 68	-	-	4	7	11
Total	34	31.5	41	45.5	152

The cost of the central office during this period was £238,000 (about $500,000). The estimated full economic cost of the project including realistic overheads and commercial rates for computing time was £1,025,000 (about $2,000,000). Dividing the costs of the central office between library contents, implementation and administration, NAG estimates that it cost £3.5 (about $6-7) per Fortran statement to develop, test and document a library routine during the first four releases of the NAG library and £4 per statement for the Mark 5 release. Implementation of the library for a new machine range during the first four releases cost approximately £0.50 to £0.60 ($0.85 - $1) per statement. Experience to date with the Mark 5 library suggests that it will cost approximately 10 cents per statement for a new implementation. When comparing the above figures with those given elsewhere in the paper it must be born in mind that all costs in the United Kingdom are significantly lower than the comparable ones in the United States.

III.3 The NATS Activity. The NATS (National Activity to Test Software) activity was initiated in 1970 with the objective of taking the [Wilkinson, Reinsh, 1971] Algol procedures for eigensystem calculations and preparing software for widespread use in the Fortran computing community. This involved a translation into Fortran plus various steps to assure the utmost in reliability, robustness and efficiency. The activity was supported by the National Science Foundation and the Atomic Energy Commission with coordination by Argonne and other principal investigators at universities. The result of this first objective was EISPACK [Smith et al, 1976] and the activity later enlarged to take on other projects using the same organizational structure and some of the same people. The NATS projects are designed to produce extremely high quality software (systematized collections) in a particular problem area.

EISPACK has had its second release and is currently in use at over 450 installations (it is available from the Argonne Code Center at the Argonne National Laboratories with the payment of a nominal handling charge). Machine versions include Burroughs 6700; CDC 6000/7000, DEC PDP-10, Honeywell 6070, IBM 360/370 and UNIVAC 1110. FUNPACK is a collection of highly machine dependent, special function subroutines [Cody, 1975]. Thirty-six routines for exponential integrals, complete elliptic integrals, Dawson's integral and Bessel functions are available for IBM, CDC and UNIVAC computers. A third project, LINPACK, is now underway to produce similar programs for solving linear equations and related problems. Plans have been made for MINPACK (non-linear optimization) and other projects are in the preliminary discussion stage.

High quality is achieved in the NATS projects by having leading experts produce the code and then exhaustively testing and validating the results. For example, the EISPACK programs were certified only after extensive use

at 15 different computing centers (test sites). This was in addition to elaborate tests for validation by the NATS principals and the earlier efforts of Wilkinson and Reinsch to assure the correctness of their Algol procedures. Similar steps were taken with the FUNPACK programs and are planned for LINPACK and MINPACK.

The NATS organization has produced a variety of software to support the creation, validation and dissemination of the numerical software. They distribute programs tailored for a particular machine. Initially, these were hand tailored, but now NATS uses a single source program approach. Their approach is to have a "generalizer" and a "selector" which translate between specific Fortran dialects and a somewhat higher level, more abstract language.

The total cost of the EISPACK project is estimated to be about $900,000 for about 12,000 lines of Fortran code (not counting the 6 machine versions). This is equivalent to about $40-50 per original Fortran statement if one assumes a new machine version costs about 15% of the original program cost. The FUNPACK collection is somewhat smaller and cost less per line of code as well as in total. Costs for EISPACK were probably unusually high for several reasons: (a) The testing, validation and documentation were unusually elaborate, (b) There were substantial "capital investments" in auxiliary software to support the creation, validation and dissemination of EISPACK, (c) There were non-trivial costs for travel and conferences because of the dispersion of the investigators and test sites.

III.D The PORT Library. In 1968 Bell Laboratories initiated a project in numerical mathematics software [Traub, 1971]. The objective then was to produce a selected set of high quality portable programs. As Traub paraphrased Santayana: "The man who doesn't write portable software is condemned

to rewrite it". This software project produced a number of quality programs over the years and in 1974 a complete library effort was initiated based on the experience gained from the earlier work. [See Fox, Hall and Schryer, 1976].

The PORT library is now in the early phase of development. It has enough programs to be called a library, but it is heavy on utility routines. A unique feature is the inclusion of dynamic storage allocation and automatic error handling. It is well short of the 400 or so programs that seem to be required for a reasonably complete library of mathematical and statistical programs. A second, larger edition is due in 1977. The library is implemented on the IBM 360/370, UNIVAC 1100, Honeywell 6000,Data General Nova and PDP-11 computers. It is in current use throughout the Bell Telephone research laboratories and is available to universities for a service charge. Others may obtain the library for a one-time license fee.

Quality control in the original Bell Labs project was achieved through a contributor-referee system similar to the contributor-validator method independently adopted later by NAG. The early work involved some of the first efforts to systematically compare and evaluate existing software. This approach has been continued with the PORT library.

The PORT approach to portability is to use exactly the same Fortran code for all machines. The library is written in a portable subset of Fortran called PFORT, [Ryder, 1974] , which is known to operate in a uniform manner on all major machines and is thought to do so on all reasonably designed systems. The only machine dependent routines are some basic utilities that supply machine constants for other programs. Dynamic storage allocation and automatic error-handling are even implemented in portable routines. The other projects have also found it beneficial to restrict the programs to language subsets.

Costs for the PORT library are not known and it is no doubt impossible to extract the exact amounts from the long history of this project. In the early stages Bell found that the testing of existing software mutated into the writing of new software and that an average of six man months was required for each program contributed. This did not include the referee's effort or any central administration of the project.

IV CURRENT RESEARCH DIRECTIONS

A considerable portion of the research progress in software for numerical computation has taken place in conjunction with some specific problem or method and this situation will probably continue for some time. We divide the current research activities into four broad categories: Software Development (for particular applications), Critical Evaluation of Software, System and Machine Effects, and Software Dissemination.

IV.A Software Development. Most of the research effort in numerical computation is in the discovery, analysis, development and evaluation of methods. This effort inevitably requires that the method be implemented and thus much of the software for numerical computation comes out of the general research effort. There are, of course, some specific projects to develop software such as described in the previous section. Many workers in this area are not well versed in software methodology and much otherwise good work becomes embedded in poor software. On the other hand, some of the best software people are in this area and one can continue to expect some primarily software ideas and results to come out of the larger activity in numerical computation research.

IV.B Critical Evaluation of Software. This activity is evolving toward determining general principles and methodologies for selecting good software [Rice, 1976]. Even though a general framework and methodology is essential to guide one in the evaluation of software in a particular area, most of the

difficulty is specific to a particular problem area. Thus the bulk of the activity is problem oriented. Problem areas where considerable progress has been made are ordinary differential equations, numerical integration, polynomial root finding and some parts of linear algebra. Problem areas where a significant activity has started are statistical computations, some more parts of linear algebra, optimization and some kinds of partial differential equations.

The two most difficult aspects of this activity are determining the proper problem space (allowable input to the software) and the criteria of performance. The more obvious choices here tend to have the following undesirable consequences: the software is evaluated for problems of little or no interest, all the software is unreliable because the problem space contains large numbers of impossible problems, the performance criteria are so narrow that ridiculous algorithms turn out to be "best" (for example, there is a trade-off between time and storage), the performance measures are so general that they have no intuitive or direct significance. Last, but by no means least, reasonable sets of problem space, software collections and performance criteria can lead to an impossibly large effort to actually carry out the evaluation.

Note that we emphasize performance evaluation and not program correctness. In performance evaluation one assumes that the author has made his best effort to assure that his program is correct - and is otherwise of high quality with respect to style, documentation, etc. Thus steps to assure the quality of individual programs are somewhat independent of performance evaluation efforts. Needless to say, poor quality programs have little chance of showing up well in a scientific critical evaluation. There are instances of software being eliminated from consideration without evaluation because the style, coding, etc. is inadequate.

IV.C System and Machine Effects. This activity breaks down into five somewhat separate pieces. The first is the computer's arithmetic unit and what to do about it. The roundoff problem is, of course, inherent in numerical computation, but that is not the main focus here. Rather the focus is on CPU's (and systems) that have erratic round-off behavior, inadequate facilities to prevent over/under flow problems, inadequate double precision hardware, etc. Top quality software demands that these idiosyncracies be identified and circumvented in some way. The result is sometimes the antithesis of good programming "style". Strange constructions are sometimes required to preserve accuracy and efficiency; these constructions must be clearly isolated and documented.

The design of computer arithmetic units has, on the average, gone steadily down hill over the past 15 years. It is now known how to design them so as to reduce the above difficulties to an absolute minimum. These designs cost about the same as current ones, but they are very rarely implemented.

Closely related to hardware difficulties are difficulties in the compiler and operating system software. Optimizing compilers can ruin the efficiency and/or accuracy of a computation. Different paging algorithms can completely change the performance of some programs. One goal in higher level languages (like Algol and Fortran) is to insulate the programmer from these considerations. Unfortunately, such questions cannot be ignored in some areas of numerical computation, especially those where the bulk of the computation occurs in a few statements.

The third category of these effects is memory hierarchies. Some numerical computations require a huge amount of storage and techniques to exploit the available memory configuration are under steady investigation. There are substantial trade-offs between memory use and computation time. Thus a sparse array can be placed in much smaller memory space at the expense of much more

computation to retrieve an array element. Considerations along these lines sometimes lead to completely new methods as well as reorganizations of old ones.

The fourth category is the proliferation of small machines such as hand calculators and mini/micro-computers. These machines are so numerous that it is not possible to do a careful job on the numerical software (which may be permanently implemented as micro-code). The manufacturers of such machines are frequently unaware of quality software principles for numerical computation. The result has been numerous instances of inadequate algorithms - both in the hardware and in manufacturer supplied libraries and systems (e.g. the Fortran built-in functions).

The final category is novel architecture such as pipeline processors, parallel processors and associative memories. These architectures are primarily motivated by numerical computation problems and naturally a great deal of research has gone into techniques to exploit them. These studies involve both new methods and new software approaches. It has been more difficult than anticipated to exploit these devices, but the ideas behind them have a natural suitability for many important applications.

IV.D Software Dissemination. Perhaps more than other areas in computing, numerical computation workers are keenly aware that their problems should be independent of the computers used. Nevertheless there has been a tremendous duplication of effort even for the most basic problems. The establishment of large, good and portable libraries took more time and effort than anyone would have suspected. Even so , there is a real human engineering problem with library subroutines. The instructions for use are usually puzzling at first and many people end up spending a week writing their own routine instead of spending a half-day learning how to use the library routine. Furthermore, the library routine is likely to be better than one written by a typical

programmer, even though he can tailor it to his particular specifications. The key question is: How do you make it irresistible for the typical user to use existing good software rather than write his own, perhaps inferior, software?

V. PROJECTIONS AND NEEDS FOR THE FUTURE

We all know that many things do not occur even though they should. I will avoid the task of trying to distinguish between what should be done and what probably will be done by taking the optimistic position that those things needed will actually arrive. Some of the projections indicated require more than just advances in software, some of them also require new methods in numerical computation, others require the application of methods now known but not yet put into practice.

V.A Software Development Tools and Techniques. Numerical computation software benefits from the use of better tools just as much as other software. These include things like better languages and compilers, programming standards checkers, program manipulation software, debugging and verification software, structured and modularized programming, etc. A particularly useful tool for numerical software is one that gives a detailed breakdown of the timing of a program. Some more specialized tools should be useful also, such as the software for roundoff analysis mentioned earlier. Numerical computation occurs at all "levels" between very machine dependent and very high level. One tool of real utility is a language and system that is at a significantly higher mathematical level than current languages (this tool is discussed next as a separate topic).

V.B A Mathematical System. Fortran, Algol and their descendants are all at the same mathematical level: algebra and trigonometry. A large part of the numerical software should operate at a significantly higher mathematical level.

The mathematical language should have arrays, functions, formulas and equations as honest data types and with some subtypes. The operations of integration, differentiation, infinite summation should be part of the language. There should be both numerical and symbolic facilities in the language. Such a language is very susceptible to becomming so complex and large that it cannot be implemented or used. It is true that it requires substantially more resources than a typical Fortran or PL/1 system. However, a careful design (i.e. reigning in one's greed) can be implemented with resources that are reasonable with today's computers. See [Rice, 1973] for an analysis of why efforts in this direction failed in the 1960's.

V.C Problem and Discipline Oriented Systems (and Hardware). We have already seen the start of large systems tailored for a particular class of users or problems. NASTRAN is an example of a structural engineering system and such systems will proliferate in the future. The vehicles of their implementation will vary; some will use specialized hardware, some will use dedicated mini-computers and others will be part of a large computer system. The driving motivation for these systems is human engineering; by restricting their scope of use, they can communicate with users much better. Of course, software will be only one facit of the task of developing these systems; new methods, substantial financing, professional support and careful attention to human engineering are all required for a really successful system.

V.D Software to Automate Model Construction. Recall that the source of numerical computation is physical and organizational models. As these models become more complex, the task of constructing them becomes a large, even dominant, part of the effort in problem solving. Some of the problem oriented systems can be viewed as highly parameterized models of certain phenomena. Little software currently exists for model construction and thus

its nature is not yet clear. However, it is clear that this is one of the key problems to be faced. One might view it as part of the human engineering, but it goes deeper than that. After all, we might view the elimination of arithmetic by computers as "only" human engineering, but we know that this actually changes the nature of problem solving.

V.E Software Parts Technology. The problem and discipline oriented systems mentioned above are in fact another approach to disseminating software: one packages good programs in a reasonable language system and users find it irresistible. Whether or not users of such systems are still programming is open to discussion, but the real objective of getting them to use other people's programs has been met. Actually, the significance of these systems goes much deeper than that; they are knowledge transfer mechanisms that will eventually entirely change the way the scientific world operates (see Rice [1976a]).

Even though many will be spared the trouble and expense of programming by these systems, there will still be a large amount of computing outside the scope of specialized systems. This programming is now done on a hand crafted, tailor made basis. Industry already knows that the solution to the high costs of a hand crafted technology is to introduce standardized, interchangeable parts. This seems to be the best hope for eventually controlling (even reducing) the high cost of program development. A software parts technology will not arise overnight, but it is time to give serious consideration to how it will work and how it can be facilitated.

VI. ACKNOWLEDGEMENTS

I thank E. L. Battiste, W. S. Brown, S. D. Conte, W. R. Cowell, C. W. de Boor, B. Ford, L. D. Fosdick, A. Ralston and J. F. Traub for useful comments and suggestions on an earlier version of this article.

VII. REFERENCES

T. J. Aird, E. L. Battiste and W. C. Gregory [1977], Portability of Mathematical Software Coded in Fortran, ACM Trans. Math. Software, 3, to appear.

W. J. Cody [1975], The FUNPACK Package of Special Function Subroutines, ACM Trans. Math. Software, 1, pp. 13-25.

F. Ford and D. K. Sayers [1976], Developing a Single Numerical Algorithms Library for Different Machine Ranges, ACM Trans. Math. Software, 2 pp. 115-131.

P. A. Fox, A. D. Hall and N. L. Schryer [1976], The PORT Mathematical Subroutine Library, Bell Laboratories.

S. J. Hague, and B. Ford [1976], Portability-Prediction and Correction, Software Practice and Experiences, 6, pp. 61-64.

T. E. Hull, W. H. Enright, B. M. Fellen and A. E. Sedgwick [1972], Comparing Numerical Methods for Ordinary Differential Equations, SIAM J. Numer. Anal., 9, pp. 603-637.

M. A. Jenkins [1975], Algorithm 493 - Zeros of a Real Polynomial, ACM Trans. Math. Software, 1, pp. 178-189.

M. A. Jenkins and J. F. Traub [1975], Principles for Testing Polynomial Zerofinding Programs, ACM Trans. Math. Software, 1, pp. 26-34.

F. Krogh [1972], Opinions on Matters Connected with the Evaluation of Programs and Methods for Integrating Ordinary Differential Equations, SIGNUM Newsletter, 7, pp. 27-48.

H. Kuki [1971], Mathematical Function Subprograms for Basic System Libraries - Objectives, Constraints and Trade-Off, in Mathematical Software (J. R. Rice, ed.), Academic Press, New York, pp. 187-199.

J. N. Lyness and J. J. Kaganove [1976], Comments on the Nature of Automatic Quadrature Routines, ACM Trans. Math. Software, 2, pp. 65-81.

W. Miller [1975], Software for Roundoff Analysis, ACM Trans. Math. Software, 1, pp. 108-128.

B. N. Parlett and Y. Wang [1975], The Influence of the Compiler on the Cost of Mathematical Software, ACM Trans. Math. Software, 1, pp. 35-46.

J. A. Prentice [1974], The Development and Maintenance of Multi-Machine Software in the NAG Project, in Software for Numerical Mathematics (D.J. Evans, ed.), Academic Press, London, pp. 383-392.

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

REFERENCES CONTINUED

J. R. Rice [1973], NAPSS-like Systems: Problems and Prospects, Proc. Natl. Comp. Conf., pp. 43-47.

J. R. Rice [1976], The Algorithm Selection Problem, in Advances in Computers, Vol. 15, (Yovits and Rubinoff, eds.) Academic Press, New York, pp. 65-118.

J. R. Rice [1976a], Statistical Computing: the Vanguard of the Revolution in Education, Proc. Computer Science and Statistics - Ninth Symposium on the Interface, Harvard University, pp. 1-3.

J. R. Rice, C. W. Gear, J. M. Ortega, B. N. Parlett, M. Schultz, L. F. Shampine and P. Wolfe [1977], Numerical Computation, Panel Report for the COSERS project.

J. R. Rice [1977a], Mathematical Software - Madison, 1977, Academic Press, New York.

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

L. F. Shampine, H. A. Watts and S. M. Davenport [1976], Solving Nonstiff Ordinary Differential Equations - The State of the Art, SIAM Review, 18, pp. 376-411.

R. F. Sincovec and N. K. Madsen [1975], Algorithm 494 - PDEONE, Solution of Systems of Partial Differential Equations, ACM Trans. Math. Software, 1, pp. 261-263.

B. T. Smith, J. M. Boyle and W. J. Cody [1974], The NATS Approach to Quality Software, in Software for Numerical Mathematics (D.J. Evans, ed.), Academic Press, London, pp. 393-405.

B. T. Smith, J. M. Boyle, J. J. Dongarra, B. S. Garbow, Y. Ikebe, V. C. Klema and C. B. Moler [1976], Matrix Eigensystem Routines - EISPACK Guide, 2nd Edition, Springer Verlag, New York.

J. F. Traub [1971], High Quality, Portable Numerical Mathematics Software, in Mathematical Software (J. R. Rice, ed.) Academic Press, New York, pp. 131-139.

M. V. Wilkes, D. J. Wheeler and S. Gill [1951], The Preparation of Programs for an Electronic Digital Computer, Addison Wesley, Reading, Mass.

J. H. Wilkinson and C. Reinsch [1971], Handbook for Automatic Computation, Vol. II, Linear Algebra, Part 2, Springer-Verlag, Berlin.