COMPUTER SYMBOLIC MATH & EDUCATION: A RADICAL PROPOSAL

David R. Stoutemyer
Electrical Engineering Department
University of Hawaii at Manoa
Honolulu, Hawaii 96822

Abstract

None of the commonly-taught programming languages is very relevant to the typical elementary-school through college math curriculum, but computer symbolic math is relevant to most of that curriculum. Consequently, a vast opportunity for beneficial mutual reinforcement and cross-motivation between math and computer education is being squandered. This paper substantiates these claims, explains their major causes, then proposes remedies.

1. Introduction

I still recall the sharp disappointment felt when I realized that FORTRAN, my first programming language, was essentially arithmetic. I also recall my first image of a numerical analyst as someone sitting on a high stool wearing a green eyeshade and arm garters, endlessly working a desk calculator and consulting musty volumes of math tables. I ultimately specialized in numerical analysis, but these early impressions greatly delayed my appreciation of computers and of the varied ways that they can be used for scientific computation. It is undeniably true that most students are far more intrigued and motivated by the artificial intelligence and game playing applications of computers than by the accounting and numerical scientific applications which account for most computer usage.

Why not exploit this strong preferential interest to help teach mathematics and computer science? If more good math, science, and engineering students are attracted to computers, and vice versa, more will ultimately learn to use computers effectively for both numeric and non-numeric scientific computations.

I propose that computer algebra is an ideal introductory computer programming language for math, science and engineering students, that it is an ideal principal language for these students, and that the means are at hand for making it fill this role at all levels throughout our educational system. As indicated in the title, I realize this idea may seem radical. However, I believe that the

supporting arguments in the following two sections ought to make the idea seem not radical at all.

2. ||Computer Symbolic Math ∩ Math ||
>> ||Common Programming Languages ∩ Math||

Although most university curricula do not yet take account of the fact, many entering college students have already had an exposure to computer programming in high school and/or junior high school. It will probably not be long before most students receive an exposure in high school or junior high, and there are even indications of a trend toward exposure to computer programming in elementary schools. Thus, there are increasingly numerous opportunities for mutual reinforcement and cross-motivation between math and computer education.

Unfortunately, none of the commonly-taught programming languages is very relevant to the typical elementary-school through college math curriculum.

Consider:

1. Numbers and arithmetic comprise appreciably less than half of the entire math curriculum, but numbers and arithmetic are about the only mathematical capabilities built-into all of the commonly-taught languages.

2. Most of these languages do offer arithmetic evaluation of elementary functions, but none offers algebraic simplification of expressions containing such functions with non-numeric arguments, via appropriate identities.

3. Some of these languages offer Boolean "arithmetic" on the constant values TRUE and FALSE, but none offers Boolean "algebra" involving simplification of expressions containing unbound Boolean variables.

4. Some of these languages offer some built-in matrix arithmetic on matrices having numeric element values, but none offers matrix algebra for which the elements can contain unbound variables, or for which the entire matrix can be an unbound variable.

5. A few of these languages offer complex arithmetic, but none offers complex algebra involving simplification of expressions containing i and unbound variables, using the identity $i^2=-1$.

6. One of the less-frequently first-learned languages (PASCAL) offers some built-in set arithmetic on finite sets of constants, but none offers set algebra in which sets or their elements can be unbound variables, or in which sets can be infinite, such as the set of all integers or of all positive real numbers.

7. None of the commonly taught languages even supports all of the above kinds of "arithmetic".

8. Even all of the above kinds of arithmetic total significantly less than half of the entire math curriculum, as is indicated in Table 1.

9. Computer symbolic math is relevant to far more of this curriculum, as is also shown in Table 1.

Now for the most damnatory indictments of commonly-taught languages:

10. The numbers and their arithmetic do not correspond to those generally taught in schools!

11. The numbers and their arithmetic do not correspond to those used in everyday life!

The limited-precision integer arithmetic of these languages is bad enough in these regards, even without its usual overflow asymmetry induced by 2's complement arithmetic. For the floating-point arithmetic of these languages, we can add the indictment

12. Few other than the very best numerical analysts fully understand the implications!

In contrast, the arithmetic that students learn in elementary school is

1. indefinite-precision rational arithmetic,

2. rounded and exact indefinite-precision decimal-fraction arithmetic.

True, in high-school chemistry or physics students may learn scientific notation, which could be regarded as indefinite-magnitude, arbitrary-but-fixed-precision, rounded-decimal arithmetic. In contrast, the floating-point arithmetic of commonly-taught languages is finite magnitude, with only 1 to 3 alternative precisions, usually chopped nondecimal. All of these internal differences from true

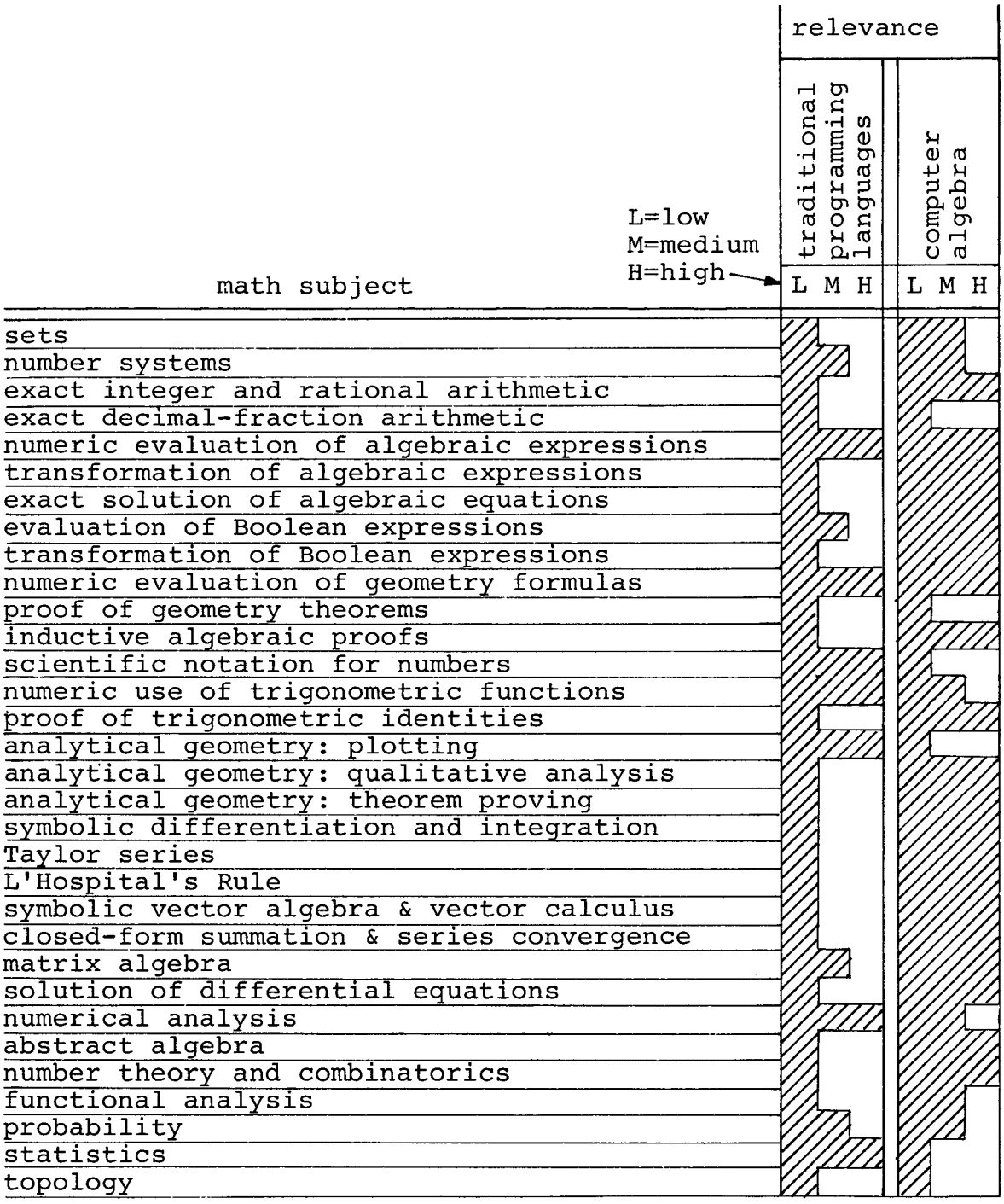

TABLE 1, RELEVANCY COMPARISON

scientific notation have external manifestations which are baffling to most people.

It is also true that some of the less-frequently taught languages, such as PL/I, do support decimal-fraction arithmetic. However,

1. This arithmetic is finite precision.

2. This arithmetic does not alter the overall conclusion suggested by Table 1.

3. Most elementary PL/I texts assiduously avoid decimal-fraction arithmetic in favor of floating-point arithmetic.

Admittedly, extended sequences of indefinite-precision arithmetic operations on experimental data suffers an unjustifiable growth in digits, but to that one can respond:

1. Render onto floating-point arithmetic that which one must.

2. Render onto more rational arithmetic all that one can.

Perhaps if floating-point computation becomes a choice rather than an imposition, users will regard floating-point with more of the caution it deserves.

Perhaps indefinite-precision rational and decimal-fraction arithmetic are inevitably less efficient than their respective finite-precision floating-point and integer counterparts. However:

1. The difference in efficiency would greatly decrease if the indefinite-precision arithmetics were microcoded or hardwired as are their finite-precision counterparts in most computers.

2. Matula and Kornerup [1979] have been exploring promising finite-precision rational arithmetic schemes called fixed-slash and floating-slash arithmetic, which combine many of the advantages of floating-point and rational arithmetic.

3. Even if the indefinite-precision arithmetic is substantially slower, computing has become so inexpensive that for the computational needs of most people, the cost of indefinite-precision computation is negligible compared to the labor of

assessing results done in an unnatural arithmetic.

How negligible do computing costs have to become before software and hardware designers abandon this historical obsession with efficiency? If a certain computation costs 10 times as much in rational arithmetic as in floating-point, and the latter method was deemed worthwhile a few years ago when computer costs were more than 10 times as much, is it not worthwhile now to do the computation in a more humane arithmetic?

In the early days of computers, scientific computation was usually done using binary fixed-point fractions having magnitudes restricted to lie between 0 and 1. The widespread acceptance of floating-point brought substantially greater convenience for a large loss in efficiency. For most work, computer costs have now decreased enough to justify another such step in favor of human understanding. Those who cling to efficiency-worship should defend fixed-point fractions rather than floating-point.

It is significant that calculator manufacturers, who must sell directly to the individual end users, are generally more sensitive than are computer manufacturers in this regard. This is true even between the calculator and computer divisions of companies which manufacture both: Most calculators at least use rounded decimal floating-point arithmetic having 8 or more digits, whereas computer manufacturers increasingly use chopped non-decimal floating-point in a dangerously meagre 32-bits.

3. Obstacles to Teaching Computer Symbolic Math

As detailed in the previous section, computer symbolic math is far more relevant to math education than are the commonly-taught programming languages. Consequently, it behooves us to be teaching computer symbolic math -- preferably as a first programming language, beginning as early as possible, perhaps distributed throughout the curriculum rather than concentrated in one course.

There is no doubt in my mind that mathematically-inclined students are enormously stimulated by the experience. Wherever I have demonstrated interactive computer symbolic math to students, witnessing their compelling excitement has increased mine to such an extent that I am now determined to help make this educational tool be used to its full potential throughout our educational system. Accordingly, during the past several years I have been teaching the subject at various levels, in various ways, in

order to informally identify obstacles and promising approaches. This teaching experience so far consists of

1. A one-semester graduate course on theory, practice, and applications of computer symbolic math. (Taught twice).

2. An undergraduate introductory one-semester scientific-programming course of which about 20% is devoted to using a computer-algebra system. (Taught 4 times).

3. A one-week intensive enrichment program for gifted high school students. (Taught once, over spring vacation).

3.1 Tutorial Obstacles

There is an almost total lack of published tutorial material about computer symbolic math.

Regarding theory, portions of the books by Borodin and Munro [1975], Aho, Hopcroft and Ullman [1975], and Knuth [1968,1969] are appropriate, but none of these is devoted entirely to the area or to the entire area. It is possible but difficult for a researcher in the area to teach a thorough theoretical computer-algebra course to self-reliant graduate students from these books together with scattered dissertations, conference proceedings, and journal articles. Moreover, the fine course notes by Fateman [1978] greatly ease the burden by collecting together a lot of this material. Unfortunately, those outside the field will not know of these notes or of the many relevant scattered references. Consequently, there is a great need for a widely-advertised book of this nature to be published by an aggressive major textbook publisher. Otherwise, few new professors outside the field will learn of the field and easily assimilate enough to introduce such a course at their university. David Yun and I are working to complete such a text, but it is a substantial impediment not to have one already available.

Regarding applications, there is also a wealth of scattered references and a dearth of material conspicuously and conveniently collected under one cover. An anthology of reprints would be a quick and easy remedy, which could attract many engineers, scientists, and applied mathematicians into the field. In fact, companion volumes of reprints on theory and applications would make a nice combination.

Regarding specific computer-algebra systems, MACSYMA has a great deal of excellent on-line and off-line tutorial material, as described by Lewis [1977]. Regretably, there is almost no tutorial material to support any of the other systems, and it is these other systems which most of the math students in this country will have to use. If we really wish to attract math teachers to enrich their courses with supplementary instruction about one of these systems, then we must write and publicize a variety of tutorial aids. Arithmetic, algebra, trigonometry, and calculus teachers each need different material having numerous exercises and full detail relevant to their particular subject, preceded by brief coverage of lower levels. These primers and guides must be written to work well with the most popular corresponding math texts. Similarly, if we really wish to attract computer programming teachers to use a computer-algebra system as a first language, then we must write and publicize appropriate primers and guides for them too, in the style of the currently popular programming texts. This means also including some nonalgebraic applications, such as approximate numerical computations, data processing, string processing, games, and list processing, together with discussion of structured programming, etc.

3.2 Economic Obstacles

Batch processing is fine for certain kinds of computation, but for typical student exercises batch processing is liable to give students the lasting impression that the computing community is making a mountain out of a molehill. Moreover, most computer-algebra results are so unpredictable in form that in practice their computation is almost always exploratory rather than via an *a priori* discernable sequence of steps. Consequently, interaction is particularly desirable for computer-algebra students.

You can imagine my disappointment when, infused with missionary zeal to bring interactive computer algebra to the masses, I discovered that I could not afford to bring even a batch computer algebra to even one full-size undergraduate class! Having guest accounts at ARPA-net sites had kept me blissfully ignorant of the costs involved. On our campus:

1. Minimal WATFIV, WATBOL, or PL/C batch jobs on our IBM 370/158 cost about 15¢, which includes cards, time, and paper.

2. Any other batch job costs a minimum of about 50¢.

3. If the input expressions are carefully constrained to require just enough computation to yield impressive results, a typical REDUCE batch job for a trivial program costs about $2.00. When a program of more than a page is involved, the cost can easily be several times this large even for trivial input expressions. I have no experience running other systems on our machine, but I am more struck by the consistency than the disparity of the various SIGSAM Bulletin timing comparisons. Moreover, algebraic efficiency is secondary compared to other overhead costs for small student jobs. For such exercises I would be surprised if any other suitable existing batch system incurred notably different charges on our computer.

4. TSO (or APL) time sharing costs $1.00 per hour of connect time plus computing charges which average about $5.00 per hour for students developing REDUCE programs.

5. Time-shared BASIC on our HP-2000 costs only $1.00 per hour of connect time for students developing BASIC programs.

6. For classes of more than about 15 students, university computation allowances make WATFIV, WATBOL, PL/C, or HP BASIC the only affordable programming languages at these two principal computation resources.

I know that some schools have computer systems or charge rates or computation allowances which permit interactive computing as the norm rather than the exception, using languages which do not instill bad programming habits. However, I also know that some schools have systems, rates, and allowances which are even more inhibitory than ours.

The economy of WATFIV, WATBOL, PL/C, and HP BASIC for students jobs is largely attributable to compactness and sharing. Most of the latter system is permanently-resident reentrant code, so that only the trivially small student BASIC programs have to be swapped. For each of the other three languages, our operating system collects together a number of small student jobs and runs them as one job step, using a compact resident translator. Thus, here are two possible ways to make computer symbolic math economically feasible for the masses at such schools:

1. Encourage Cornell University and the University of Waterloo to develop compact resident algebra

systems for batch student use. (Waterloo could call theirs WATALG!

2. Encourage Hewlett-Packard and its competitors to develop a dedicated time-shared algebra system intended for student use.

Actually, it was unclear whether or not a sufficiently compact system was achievable, so to explore this possibility, I first developed a truncated power series program and a toy symbolic differentiation program for the HP-67 programmable pocket calculator, as described by Stoutemyer [1979]. Although those programs are much less than what is desirable in a minimal educational system, it was encouraging that some symbolic math could be done in a memory capacity of only 224 instructions plus 26 numbers. This suggested exploring a third way to make computer symbolic math economically feasible for the masses.

Microcomputers based on the INTEL-8080 and competitive chips are becoming increasingly prevalent in schools, because they are so incredibly inexpensive. Including a terminal and a means of saving programs externally, prices range from about $600 for a stripped-down model with 4K bytes of memory and one cassette-tape drive, to about $4,000 for one with 64K bytes of memory and dual 8-inch floppy-disk drives. (There are, of course, expensive imitations which cost several times as much.) These prices are so low that it is not worth setting up an accounting system for their use. These prices are so low that increasing numbers of our students and faculty have purchased one for their own education and recreation. At $4,000 each, our campus could purchase 20 of these for the amount that our HP 2000 users are billed per year, or we could purchase 400 of these for the amount that our IBM 370/158 users are billed per year.

Despite the mere 64K bytes of address space, powerful versions of APL, COBOL, FORTH, FORTRAN, LISP, PASCAL, and PL/M have been implemented for microcomputers. The structured implementation language muSIMP-77^{tm} developed by Albert Rich is particularly suitable for implementing computer-algebra systems. Consequently, bootstrapping from that system, we developed the symbolic math system briefly summarized in the appendix. The system is successful beyond our fondest expectations, so now we have an economically feasible way to provide computer algebra to large numbers of students. Moreover, the compactness of the system provides encouraging evidence that the other two alternative solutions suggested above are also worth pursuing.

The system is, of course, nowhere near as powerful as the largest systems. However it could greatly help popularize computer algebra, leading many more people to become aware of the

more powerful systems such as MACSYMA, which could help them with their research.

3.3 Attitude Obstacles

How do math teachers and computer-programming teachers feel about the idea of teaching computer algebra?

Most of them do not know about computer algebra, so a tremendous amount of encouragement, support, and teacher education is necessary from those who know about computer algebra and about usage of specific systems.

How then do math teachers and computer-programming teachers who have been exposed to computer algebra feel about teaching it?

Probably, until enough of their peers are involved so that they begin to feel left out, most will express courteous admiration, but decline to get involved. It is more a question of human nature than perceived merit. Perhaps the reasons include

1. Many university math professors are relatively indifferent to calculus and to constructive mathematics in general. These subjects are not at the prestigious forefront of pure math research, so there is a strong incentive to devote most effort elsewhere.

2. Those who have never learned about computers may feel that it is too late to take the plunge because they anticipate a humiliating period of publicly-revealed ignorance. I have heard that personal computers available for checking out overnight and over the weekend are helping overcome this cause of reluctance.

3. Many people have a strong brand loyalty to a particular programming language, which the proposed computer-algebra language does not syntactically resemble. Probably they know only one programming language, and their expertise was too painfully acquired to contemplate enduring another learning period assumed to be of comparable length. This is actually a variant of the abovementioned phobia about learning a first programming language, but it is less well founded because the first language is generally by far the hardest to learn. Nevertheless, this obstacle is most easily overcome by

providing a large selection of surface languages which resemble those which are currently most popular at the elementary school through college levels. It is notable that none of the major computer-algebra languages, including the system described in the appendix, resembles either of the two languages most widely taught in our schools: BASIC and FORTRAN. Admittedly, to couch computer symbolic math in either syntax is making a sow's ear out of a silk purse, and it is onerous to help perpetuate archaic programming style. However, perhaps by giving the customers what they want, their perspectives will become sufficiently broadened to permit weaning to more modern syntax. It is wishful thinking to imagine that this group of potential beneficiaries is a minority. For example, given a choice between APL and BASIC on their IBM 5100 personal computers, a vast majority of the customers choose the latter.

4. Some educators are concerned that students will not master algebraic operations if computers perform them for the students. This is a variant of the concern that numerical pocket calculators will destroy children's ability to do arithmetic. There are many arguments against this concern:

   (a) Similar concerns were undoubtedly expressed about Arabic numerals, multiplication tables, logarithms, and Laplace transforms; but we have survived their convenience.

   (b) Automatic computations free humans for higher pursuits.

   (c) A demonstration that an operation can be done automatically by computer can encourage average and poor students that the flashes of inspiration given only to brilliant students are unnecessary for that operation. There is revealed hope for the methodical but non-brilliant students.

   (d) Provided they are written in the surface programming language, inspection of the underlying algorithms can help students learn the methods for accomplishing the operations.

   (e) Programming extensions to the built-in operations can reinforce understanding of both the built-in and new operations.

(f) Computer algebra enables students to experimentally investigate larger examples than is otherwise practical. Patterns thus revealed may suggest useful theorems to the students. Conjectured patterns thus broken provide counterexamples against false hypotheses.

(g) A symbolic math system can be used by a computer-aided-instruction system in order to provide far more flexible math, drill, practice, and question answering than is otherwise possible.

(h) A built-in trace facility can allow students to see each step of a computation, rather than merely the final result.

The above attitude obstacles are formidable, despite the mentioned remedies and arguments. Fortunately, there are more than enough enthusiastic and receptive educators to precipitate widespread computer-algebra usage, so it is unnecessary to waste time and good will exhorting their reluctant colleagues. Almost every math department, no matter how pure, has at least one closet computer enthusiast. Almost every engineering and science department has several proclaimed computer enthusiasts. Almost every computer science department has at least one faculty member who is bored with or dissatisfied with the language taught in the introductory course. Almost every high school and junior high school has at least one math, science, or programming teacher who is enthusiastic and anxious to try new ideas. These people are not hard to find. A few phone calls to likely departments will usually lead to them.

4. The Proposal

For reasons outlined in section 2, it is highly desirable for math, science and engineering students to have computer algebra as a principal and a first exposure to programming. As explained in section 3, past obstacles to accomplishing this objective now have been or can be overcome. Consequently, the time is ripe to launch a well-organized national or international effort to develop, test, and disseminate educational materials which are necessary to take advantage of this educational opportunity.

There is much that each of us can do informally to help accomplish these objectives:

1. Introduce computer-algebra courses at our own research, development, or educational institutions, perhaps as special continuing-education courses outside the ordinary curriculum and time schedule.

2. Introduce computer-algebra exposure in an enrichment or supporting role within appropriate existing courses.

3. Locate adventuresome colleagues at the same or nearby institutions, and help them introduce computer algebra in their courses.

4. Volunteer to give lectures and demonstrations at neighboring departments, colleges, high schools, and local or national math, engineering, science, and educational professional meetings.

5. Take the initiative on acquiring, establishing access to, and publicizing some of the general-purpose systems which run on machines available at our institutions and at neighboring ones. (Because of the widely varying computational facilities, needs, and personal tastes present at various institutions, it behooves us to adopt an ecumenical attitude and become proficient with more than one system.)

6. Become known to computer-center consulting staffs and likely departments as an expert willing to help others use the locally available systems correctly and effectively.

7. Expose new audiences to computer algebra by publishing survey, tutorial, and application articles in journals and popular magazines where a computer-algebra article has never before appeared.

8. Help and encourage newcomers to publish their computer-algebra research in their professional journals.

9. Alert newcomers to relevant professional meetings, users' groups, and professional organizations.

10. Write users' guides, supplements, or other tutorial material, and share it with our colleagues in return for their suggested improvements. Such material can be publicized or distributed via announcements in the SIGSAM Bulletin, via specific user-group newsletters, and ultimately via published textbooks.

Besides these informal means, I propose that those who are interested join the SIGSAM education committee for the purpose of cooperative development, testing, and dissemination of computer-algebra educational material.

Hopefully, such a committee would include

1. members knowledgeable about computer-aided instruction, math curriculum, and computer-science curriculum spanning all levels;

2. authors willing to draft written material collectively covering all suitable computer-algebra systems;

3. teachers at all educational levels who are willing and able to test the material;

4. representatives from industry or government research labs interested in developing material for self-study or in-house courses;

After an initial exchange of ideas, the committee could draft and undertake a plan of action. For example, the committee could submit joint funding proposals to appropriate agencies.

After years of relative anonymity, computer algebra is ready to emerge as a widely known and widely used beneficial tool for education and research. We can do much to assist and hasten this emergence.

5. Appendix: Summary of an Educational Computer Symbolic Math System Implemented on the Intel 8080

This algebra system has

1. A user-oriented high-level programming language in which all of the underlying math algorithms are written.

2. Interactive console I/O and batch I/O for sequential files on a storage medium such as floppy disks.

3. Bignums and exact rational arithmetic, with user control over the I/O radix and display format.

4. Automatic algebraic simplification including identity operations and collection of similar terms or factors.

5. Optional algebraic simplifications including multinomial expansion, expansions of products of sums, common denominators, and content factorization.

6. Optional simplifications for elementary functions, including expansion of logarithms of powers and/or products, trigonometric multiple-angle and/or angle-sum expansions, and the opposite logarithmic or trigonometric transformations.

7. Symbolic differentiation.

8. Symbolic integration, using derivatives-divides rules together with linearity of the integration operator.

9. Symbolic summation.

10. Matrix algebra.

11. Exact solution of a nonlinear algebraic equation.

12. An extendable Pratt parser-deparser.

13. A primitive pattern matcher.

The system is modular so that space can be saved by loading only the packages which are needed for a particular application.

In one minute on an 8080 running at 2 megahertz with 48 kilobytes, the system can expand 290!, $(1+x)^{20}$, $\sin(16x)$, $(x_1+x_2+\ldots+x_9)^2$, or $\sin(x_1+x_2+\ldots+x_5)$. Thus, the speed and capacity are clearly sufficient for typical textbook problems. In fact, we suspect that, as with hand-held calculators, the system will prove useful for much research, despite the existence of significantly more powerful but less accessible or less personal systems.

The algebra system was developed by Albert Rich and me, with support from NSF.* The source listing of the algebra system, written in muSIMP-77, is public domain, and it is being submitted for publication. That listing may be freely copied, modified, or adapted to other implementation languages.

The Soft Warehouse maintains a version of that algebra system, under the name muMATH-78tm, which it distributes together with its muSIMP-77 implementation software. The

*Grant MCS7802234

distribution charges are low, in keeping with the educational and personal-computing objectives of the software. The address of The Soft Warehouse is P.O. Box 11174, Honolulu, Hawaii 96828.

6. Bibliography

Aho, A.V., Hopcroft, J.E., and Ullman, J.D., [1975]: *The Design and Analysis of Computer Algorithms*, Addison-Wesley Pub. Co., Reading, Mass.

Borodin, A.B. and Munro, I. [1975]: *The Computational Complexity of Algebraic and Numeric Problems*, American Elsevier, N.Y.

Fateman, R.J., [1978]: unpublished notes on computer algebra, Computer Science Dept., Berkeley, Calif.

Knuth, D.E., [1968]: *The Art of Computer Programming, Vol. 1, Fundamental Algorithms*, Addison-Wesley Pub. Co., Reading, Mass.

Knuth, D.E., [1969]: *The Art of Computer Programming, Vol. 2, Seminumerical Algorithms*, Addison-Wesley Pub. Co., Reading, Mass.

Lewis, V.E., [1977]: "User Aids for MACSYMA," Proceedings of the 1977 MACSYMA Users' Conference, NASA CP-2012, pp. 277-290.

Matula, D.W. and Kornerup, P., [1979]: "An approximate rational arithmetic system with intrinsic recovery of simple fractions during expression evaluation," *Proceedings of the 1979 European Symposium on Symbolic and Algebraic Manipulation*, Springer-Verlag.

Stoutemyer D.R., [1979]: "Symbolic math on a programmable hand-held calculator," submitted.