Computer algebra in modern physics

J. F. Ogilvie[a)]
Department of Chemistry, National Tsing Hua University, Hsinchu 30043, Taiwan

(Received 10 May 1988; accepted 2 September 1988)

After an illustrative definition of computer algebra, the common capabilities of modern processors are outlined. The systems that have been used for applications in physics are briefly described. A summary of the uses of algebraic processors in several branches of physics is presented. The different approaches to algebraic and numerical processing are discussed and sources of further information are surveyed.

INTRODUCTION

What is computer algebra? In Fig. 1(a) we see a short list of statements in no particular language that might form part of some program. These instructions are a coding of the recurrence relation for the generation of one form of the Chebyshev orthogonal polynomials. The first two lines give the values of the initial elements, then in two successive loops the polynomials are formed and printed. The coding resembles BASIC, a popular language for numeric processing; if we insert a previous statement to set x equal to 7, then the output of the print statement would be that given in Fig. 1(b). In numeric processing, if the value of a variable is not set, then either the default value (commonly zero) is assumed by the processor or the value left in the storage location assigned to that variable in the previous program is used (a practice common to old models of IBM computers). In computer algebra, if no value is assigned to a variable, its value is itself; in this case the output might look like what appears in part in Fig. 1(c). Notice the raised exponents in the two-dimensional form of the output given (optionally) by the processor. The appearance of the output thus follows the natural form to some extent, although to my knowledge subscripts still disdain to adopt a lowered position. This is computer algebra which, for an increasing number of practitioners of physical science and engineering, is as much a component of their arsenal of methods to solve problems as mere numeric computing.

Computer algebra, or symbolic and algebraic computation, has been applied to physics almost since its emergence as a viable computing activity. The publication of a well-known review[1] can be taken to mark the end of the first era of computer algebra. About that time (1972) the first primitive versions of processors for this task were being succeeded by more refined and versatile processors. Correspondingly, the mid 1980s mark another turning point, this time to the personal algebra machine that has greater capability than the mainframe computers of a few years earlier for two reasons, namely the maturing of the processors (like languages—the distinction is explained in a subsequent section) and of course the large CPU cores and rapid execution of these small and relatively inexpensive machines. Thus it is timely both to review the progress and capabilities of the available systems and to outline some recent applications that may lead the physicist reader to foresee the ways in which computer algebra can assist in the tasks of his vocation, in both research and teaching.

Computer algebra should be intrinsic to the practice of physics in any environment, as the great range of applications to be outlined is intended to demonstrate. It should be as natural to use the digital computer for algebra as to use the four-function hand-held calculator for arithmetic, and at least one pocket calculator having explicit algebraic capabilities (though necessarily limited in scope and size of problem) has already been marketed. In a context much more general than physics, Fitch[2] stated "At present [1985] it is unfortunately still the case that much algebra is being done by hand." There is no need for this situation to persist as far as physics is concerned, for the facilities are available and are in general more easily used than the "languages" for mere numeric computing.

I. GENERAL CAPABILITIES OF COMPUTER ALGEBRA

Although there is naturally some variation among the available implementations, processors commonly incorporate the following capabilities.

A. Arbitrary precision integer and (if desired) real arithmetic

For instance, the instruction 1000! (typically followed by a semicolon or dollar sign as a terminator of input, plus a carriage return) generates a number having about 2800 digits, expressed as an integer requiring some 40 or 50 lines on the monitor or printer (depending how the line length is set). Similarly, provided that the user has chosen floating point (i.e., inexact) arithmetic, the setting of some switch called PRECISION or POINT to have the value 50 (or other value, also at the user's option) results in numerical

[a)] Present address: Institute of Atomic and Molecular Sciences, Academia Sinica, P. O. Box 23-166, Taipei 10764, Taiwan.

```
X = 7
      ......
      T(0) = 1
      T(1) = X
      FOR J = 2 TO 8
      T(J) = 2*X*T(J - 1) - T(J - 2)
      NEXT J
      FOR J = 0 TO 8
      PRINT T(J)
      NEXT J
      ......
```

(a)

```
1
7
97
1351
18817
262087
3650401
50843530
708159000
```

(b)

$T(0) = 1$
$T(1) = X$
$T(2) = 2X^2 - 1$
$T(3) = 4X^3 - 3X$
$T(4) = 8X^4 - 8X^2 + 1$
$T(5) = 16X^5 - 20X^3 + 5X$
.....
.....
.....

(c)

FIG. 1. (a) Putative segment of text to form Chebyshev orthogonal polynomials; (b) additional preliminary text statement and output through numerical evaluation; (c) part of the output through symbolic evaluation.

values containing about 50 decimal digits; with such a setting, the instruction

10/3

would result in an answer like

3.33333333333333333333333333333333333333333333333333.

If the floating point arithmetic is not deliberately chosen, then the natural form of numerical values other than integers is a rational fraction, just 10/3. Ordinarily this is printed in the natural form of two integers separated vertically by a line, but if the denominator or numerator is inconveniently small or large, then two numbers might be printed with a horizontal separation by a slash as printed here.

B. Expansion and ordering of polynomials and rational functions

In some systems, the normal representation is the compact form, such as $(x + 1)^2$, whereas in others, the expanded form $1 + 2x + x^2$, or perhaps in the reverse order $x^2 + 2x + 1$, is the default form. To change from compact to expanded form is always at the user's option, but obviously the reverse operation implies some factorization capability that is not so trivial. Similarly, the simplification of expressions, such as having a common denominator or not, is at the user's control.

C. Factorization of polynomials

During recent years much research in mathematics has been devoted to this and related topics, and consequently new algorithms have been developed that are already incorporated in various algebra processors. Further applications of factorization are required for the integration operation.

D. Substitutions and pattern matching in many and various forms

Either local substitutions or global replacements are possible, and such procedures, among the most tedious to do manually, are among the most valuable operations in computer algebra.

E. Analytic differentiation and integration

The processors incorporate the standard rules of differential calculus, specifically to find the first-order partial derivative of a specified expression with respect to a specified variable; that variable must commonly be an atom or a kernel, i.e., not have any value assigned to it (other than itself). Extended use of this operator permits differentiation with respect to more than one variable or more than once with respect to a given variable or to both, i.e., general nth-order partial differentiation. The procedures of differentiation and integration are invoked through operators, and these particular operators are but two of many supplied with a given processor.

Following the discovery by Risch[3] of an algorithm for symbolic integration of certain types of integrands, Norman and Moore[4] succeeded in implementing this for machine execution. Not only the resulting package has been incorporated into several processors, but also pattern matching is used for this purpose. The indefinite integral is commonly formed directly, but a substitution into the result of the upper and lower limits followed by the appropriate subtraction naturally generates the definite integral. In some cases, pattern matching can be applied to specific definite integrals for which no indefinite integral is known in explicit analytic form, e.g., functions containing e^{-x^2}, and even positive and negative infinities can be handled as limits in such cases. A digression about printed tables of integrals may be worthwhile at this point; the older tables contain many cases of actual errors or inadequately specified results (i.e., the given result is correct when some parameter a has a positive value but not when it is negative, but possibly no such cautionary advice appears). In some well-known tables, between 7% and 20% of the listed integrals are so affected. The integration functions or operators in the computer algebra processors are generally more reliable, producing a null result if the integration cannot be performed and querying the user if the possible answers depend on the sign of some parameter; also, to check the result by an immediate differentiation is readily effected and is indeed a recommended procedure for uncommon types of integrands. (It is notable that the latest edition of an extensive table[5] of integrals and series

contains far less than the indicated proportion of errors, primarily because many answers were tested by means of computer algebra.)

F. Facilities to define additional functions and to extend the syntax

Typically the functions defined within the processor include the common trigonometric functions (sine, cosine, tangent, and cotangent), their inverses (arcsine, arccosine, arctangent, and arccotangent), the hyperbolic (sinh, cosh, and tanh) and their inverse (asinh, acosh, and atanh) trigonometric functions, the natural logarithmic (log or ln) functions, and exponential (exp) functions. Their properties (derivatives and integrals) are also known to the processor. If the user wishes to define a further function of particular interest, such as secant, then the appropriate definition and the rules for differentiation and integration may be added in a global context; thereafter reference to any such function no matter what the operand invokes a correct algebraic manipulation.

G. Calculation with symbolic matrices

Beside the standard operations, addition, subtraction, and multiplication, of confirming matrices, the processors commonly include transposition, determinant, and inversion operations.

Other than the above categories of capabilities, most available processors designed for general purposes include further specific capabilities that depend on the experience or objectives of the designers. In particular, some processors designed by scientists with a strong background in physics incorporate further functions or operators with specific applications in certain branches of physics and cognate disciplines. Allusion to some such capabilities is made in Sec. II.

II. COMMONLY USED PROCESSORS

The endeavor in the following paragraphs is to provide a critical summary and appraisal of each processor or language for symbolic computation that has been, or is likely to be, applied for the purposes of research and education in the physical sciences. Naturally, for some processors, different versions exist, either for different types of machines or from different sources; here such distinctions between versions are ignored.

Among the earliest processors for computer algebra was FORMAC, developed by IBM Corporation to be a superset of the PL-1 language for numerical processing that was itself designed to contain the best features of structured programming, a broad range of mathematical functions for scientific and technological applications and the handling of files and characters. IBM terminated the development of FORMAC before 1968, but some further development embodied in FORMAC-73 occurred in Europe.[6] In particular, FORMAC lacks factorization and integration facilities, but is nevertheless useful for many applications involving expansion and substitution. Distributed in the form of macros that restrict its implementation to IBM (or compatible) computers having an architecture of the 360 type, currently this processor is still in use mainly in countries lacking access to the more modern processors.

The list processor LISP was designed not only as a language for programming computers but also as a formal mathematical language[7] primarily for symbolic data. As such, it has seen little application directly for computer algebra, partly because of lack of many capabilities listed above but more because typical expressions are characterized by a proliferation of parentheses that prove an impediment to use by casual users, although it remains the most popular language for research in artificial or machine intelligence. However, some calculations[8] of interest in physics have been done directly in LISP in the early years of computer algebra. Furthermore, many processors with higher levels of capabilities for computer algebra are themselves programmed in LISP, so that its importance in this field far transcends its visible applications.

Another early processor developed (like LISP) at the Massachusetts Institute of Technology and based on LISP is MATHLAB.[9] Beside most standard features listed above, it included direct and inverse Laplace transforms and the solution of linear differential equations and of simultaneous linear equations. Now quite obsolescent, MATHLAB has been succeeded by MACSYMA, which, beside all the specified capabilities (general and those of MATHLAB), currently includes, among other capabilities, manipulation of vectors and tensors, plotting of functions in two or three dimensions, the standard numerical capabilities, solution of partial differential equations, and the generation of FORTRAN output of MACSYMA expressions. For many years users had access to MACSYMA at only a few computing sites, but now this well-developed and tested processor, already having many applications in physics, is commercially marketed for a few models of computers and operating systems. MACSYMA is probably the largest and most comprehensive processor in current use, but has run on relatively few types of machine both because of its size and because it was not originally designed to be portable.

A highly developed processor especially designed for engineering applications at Bell Laboratories is ALTRAN, denoting algebra translator. This processor has the capability to perform rational operations on rational expressions in one or more indeterminates (algebraic quantities or variables), with integer coefficients, and is designed to handle relatively large problems involving such objects with considerable efficiency. The syntax of ALTRAN is similar to that of FORTRAN; in fact ALTRAN is based on FORTRAN (rather than LISP), so that in principle it can be run on any machine having a FORTRAN compiler and sufficient core memory. Although ALTRAN contains neither factorization nor integration facilities, reflecting its maturity achieved about 1973, for certain types of problems it nevertheless remains an efficient and reliable processor. However, the execution of ALTRAN must be in batch mode, not interactively, because of the compiled nature of the FORTRAN preprocessor.

The processor probably most widely used for computer algebra around the world is REDUCE. Based on LISP, this language was developed (later) as a result of a felt need (about 1960) to execute by machine some calculations in high-energy physics by A. C. Hearn (then in the Cavendish laboratory at Cambridge University), so the inclusion of a set of commands for such applications should not astonish one. Some such commands are a dot operator to denote the scalar product of two Lorentz four-vectors, gamma matrices associated with fermion lines in a Feynman diagram, the EPS completely antisymmetric tensor of order four, the use of vector and Dirac expressions, trace

calculations, and the extension from the normal four dimensions of calculation in quantum electrodynamics to an arbitrary number of dimensions. Although REDUCE-3 already constitutes a flexible and mature processor with broad and extensive capabilities, REDUCE is under continual development at several centers. However, since it is already widely used on many different machines and for applications of a great range and variety, REDUCE is a well-tested and proven system. Furthermore, a useful set of extensions[10] for matrix and other operations (including more than 100 additional operators) has been produced for REDUCE to enhance its capabilities for certain applications. REDUCE is available not only for most types of mainframe computers, including the Cray 1 and X-MP supercomputers, but also for some minicomputers and workstations and even (for smaller problems) on common types of microcomputers (Apple Macintosh, and IBM-PC/XT and AT or compatible using a DOS operating system). The processor is designed to run interactively if the operating system permits it, or in batch mode otherwise. The speed of execution varies considerably, depending both on the efficiency of the underlying LISP interpreter and on the intrinsic CPU speed, but, especially in environments such as workstations or microcomputers, this property is not of great concern. Recent versions contain both a machine-readable manual and tutorial aids to the understanding of the system. Beside the features listed above of interest in connection with high-energy physics, noncommutative algebra and the possible definition of symmetric and antisymmetric operators are of direct application in physical problems.

A more recently initiated system, also developed by a physicist (S. Wolfram while at California Institute of Technology), is SMP, denoting symbolic manipulation program. Beside some features of interest to physicists, this processor writen in C contains the now standard mathematic capabilities in algebra, calculus, and matrices, as well as having extended capabilities in graphics and permitting generation of code in FORTRAN and C languages for numeric processing. SMP is regarded as being controversial in that inexact arithmetic is performed by default in order to achieve computational efficiency. Hence this processor is claimed to execute particularly rapidly, but this claim and another of reliability has been challenged by Fateman[11] who states "It appears that the versions of SMP examined will be less satisfactory than almost any other system in fulfilling any of these expectations;" the expectations were specified to be exact and correct results, that if advanced systems cannot solve a problem then it must be quite difficult, and that symbolic systems should work regardless of singularities or near singularities as computation is done exactly. It is possible that versions later than those examined by Fateman may have been improved.

A processor, the development of which commenced in 1965 although it was released for general use only about 1985, is SCRATCHPAD II (work on version I ceased in 1975 and this version seems to have never been released). This processor, which may be considered the successor to FORMAC but of a quite different design, has been written in LISP for IBM computers, although at present work is in progress to rewrite all the code in a special language SPAD. SCRATCHPAD II is viewed as a general-purpose programming language rather than as merely a language for computer algebra, and is designed to be used both by a naive user as a sophisticated calculator and by an expert to perform sophisticated mathematical computations. As an interactive system for manipulating formulas, it has an extremely broad range of intrinsic capabilities including calculus, modern algebra, number theory, and the solution of equations. As a compiled language for the formal description of algorithms, it provides a strongly typed programming language for the description of algorithms in their most general context. Possibly SCRATCHPAD II will have its greatest applications in pure mathematics, but in principle any physical problems of a strongly mathematical kind could take better advantage of this processor than of other processors listed here. Of the initial fifty users of this processor outside IBM, most were associated with mathematics and computer science departments in universities; the list included only two groups associated with engineering and none directly with physics. At present SCRATCHPAD II runs only on mainframe IBM computers under the VM/CMS operating system, but release for execution on IBM RT PC workstations having at least 8MB of real memory is expected; it is viewed not as a commercial product (and therefore receives no official marketing support) but as a research project.

Differing in purpose from the above processors, MAPLE is an interactive system for algebraic computation designed initially for educational applications. It has been developed at the University of Waterloo, Canada, well known for its WATFOR and WATFIV processors for testing FORTRAN programs, by the symbolic computation group. The capabilities of MAPLE include, beside all the standard features, powerful procedures for simplification, summation, and limits; solution of equations (one or more linear equations, some systems of nonlinear equations, and even first- and second-order ordinary differential equations); Taylor or asymptotic series; linear algebra (intrinsically providing many kinds of orthogonal polynomials); a "remember" option to avoid unnecessary recomputation; and an online "help" facility. MAPLE is distinctively designed to be compact and modular; thus literally dozens of students can simultaneously use MAPLE from terminals of the same central processor for applications of greater or (probably mostly) lesser complexity without excessive degradation of system response, in marked contrast with the situation that but one user of MACSYMA or REDUCE for a significant calculation on a shared system may suffice to produce degraded response. MAPLE is an interpreted language written in C, but most library functions are written in the MAPLE programming language designed to facilitate the expression of, and the efficient execution of, mathematical operations; thus the procedures that the user programs for specific applications are equal in status to these library functions. MAPLE requires no type declarations, but the available data structures include sequences, lists, sets, tables, arrays, matrices, and tensors. Generation of code in C or FORTRAN is possible, but there seem to be no facilities especially applicable to physical problems. MAPLE runs on several types of mini- and mainframe computers and workstations but apparently not yet on microcomputers.

On the other hand, designed especially for microcomputers is MUMATH. The first version was developed for 8 bit processors, and an even smaller version, PICOMATH, had primitive facilities for such machines with only 8K of main memory. All these versions are of essentially no interest for applications in physics, but the version MUMATH-83 for 16 bit microcomputers with 8086 or 8088 processor chips running a DOS operating system

has some limited applications. At present the main limitation to serious algebraic computations is the fact that MUMATH-83, although it can run in as little as 128K, can take advantage of at most 320K of core memory with at most 256K of user workspace. Within this restriction, the processor has several attractive features such as solution of linear simultaneous equations, limits of functions, closed-form summation and products, vector algebra and calculus, and solution of some ordinary differential equations. The integration facility suffers from the limited memory space to rather simpler types of indefinite integrals than the above processors, but some definite integrals are evaluated even at infinity limits. For a similar reason, the extent of factorization provided is almost negligible. Noncommutative algebra does not seem possible. However, because the underlying language MUSIMP is more natural than LISP, programming of additional procedures and functions is not difficult. Development of MUMATH appears to have been terminated, apart from the correction of minor flaws. To succeed it, a new processor for computer algebra on microcomputers that includes use of two-dimensional display, access to greater core memory, automatic graphic presentation, provision of factorization, and more flexible simplification rules is in course of production.

Many other processors or languages for computer algebra have been devised, but generally they are either limited to specific types of computers or designed for particular types of problems. Some of these are SCHOONSCHIP (developed at C. E. R. N. originally for CDC computers and based on COMPASS, but later adapted[12] to IBM computers), which was originally designed primarily for calculations in quantum electrodynamics and high-energy physics; ASHMEDAI (based on FORTRAN), CAMAL (developed at Cambridge University for the Titan computer and later adapted to an IBM system, but running only as a batch processor), and LAM (also ALAM and CLAM) for general relativity; CAMAL again and TRIGMAN (based on FORTRAN) for celestial mechanics; and SAC, for symbolic and algebraic calculations, mostly of a formal mathematical kind, which operates by calling a series of FORTRAN subroutines. SAC has now been succeeded by SAC-2 which is commonly used in association with the programming language ALDES, denoting algorithm description, which translates the user statements to FORTRAN code for compilation and execution. As a language for artificial intelligence applications such as automated theorem proving, PROLOG, denoting programming in logic, is planned to serve, instead of LISP, as a basis of a new symbolic formula manipulation system for "fifth-generation" machines. CAYLEY has been developed at the University of Sydney for group theory. Even APL has been adapted for computational algebra.[13] Of course other processors have been brought into existence for either various special purposes or particular machines, and some of these, as well as some mentioned above, have also passed from practicality with their host computers. However, the preceding list provides at least a summary of typical approaches to processor design and capabilities; it also demonstrates the extremely broad range of interest and diversity of applications in pure and applied mathematics, science, and engineering.

Which processor should a physicist use? Of course the selection must depend on the machine available on which a particular processor can run, and on the nature of the applications. REDUCE has the advantage of being a well-established but still evolving and technically well-supported system that is available, at modest cost, for many types of machines from micro- to supercomputers. MACSYMA has required much larger resources on computers of only a few types and is more expensive to acquire, but probably provides the most extensive capabilities. Having excellent documentation and "help" facilities, MAPLE has been developed later than the other two systems just named, is made available for many types of machines at modest subscription costs (like WATFIV), and operates relatively efficiently. MUMATH-83 operates on microcomputers of the IBM type at a purchase price not much less than that of REDUCE. These four processors represent the systems likely at present to be worth initial consideration for either general purposes or most applications in the physical sciences.

III. APPLICATIONS IN MODERN PHYSICS

In their extensive review,[1] Barton and Fitch described in detail the application of algebraic processors to problems in celestial mechanics, general relativity, and quantum electrodynamics, later reviewed again by Brown and Hearn.[14] In general relativity, the field equations express the relationship between the geometrical structure of four-dimensional space-time and the distribution of mass and energy within it, using tensor notation and exponential and circular functions. In celestial mechanics, according to the analytic perturbation theory the coefficients of the perturbation series are first obtained as symbolic functions of the parameters, giving rise to thousands of terms. Thus it is natural that computer algebra should find early application in these fields because the field equations play a role in general relativity similar to that of the Hamiltonian equations in celestial mechanics. Barton and Fitch illustrated[1] their discussion with sample programs in the then available languages applied to these problems. Fitch also discussed[2] in detail the problem of Keplerian motion solved in terms of the f and g series with REDUCE; this problem has frequently been used as a demonstration of algebra processors. In another demonstration of the power of computer algebra, the calculations on the lunar motion that required 20 years of Delaunay's manual labor in the mid-nineteenth century were duplicated[1] about 1970 on a computer in about 9 h of processor time, and might take appreciably less time on a currently available microcomputer. Just as the motion of the moon can be computed by an analytic approach, so the motion of a particle inside an accelerator can be solved[15] through the use of REDUCE. The Compton scattering cross section of an electron interacting with a photon is an example in quantum electrodynamics also illustrated[2] by Fitch by means of a program in REDUCE. In the past, SCHOONSCHIP, REDUCE, and, to a lesser extent, ASHMEDAI were applied[16] to problems in quantum electrodynamics in general and Feynman diagrams[17-19] in particular; FORMAC, CAMAL, and LAM to problems in general relativity; and TRIGMAN, CAMAL, and REDUCE for celestial mechanics, but now all such problems can probably be tackled successfully through the use of MACSYMA or MAPLE in addition to REDUCE or other general-purpose processors. Further in relation to high-energy physics and general relativity, the use of SCHOONSCHIP for multiquark calculations,[20] FORMAC for computation of the Einstein tensor,[21] and REDUCE for problems[22] of supersymmetry and

supergravity, the geometrical characteristics of compactified multidimensional Riemannian space,[23] Einstein metrics,[24] and the computation of the group-theoretic weight of Feynman diagrams in nonabelian gauge theory[25] are also worthy of reference for the approaches to programming.

The application of computer algebra to problems in atomic and molecular physics is less well documented than to problems in the above fields. There are, however, many such examples, some of which are mentioned here before a more extensive description of one particular molecular problem. The exact integration of Slater integrals in calculations of atomic energies and properties provides two examples before the development of automatic integration packages: ALTRAN was used in one case[26] and FORMAC in the other,[27] whereas now the same results could probably be obtained with much less explicit programming by means of the integration packages within MACSYMA, REDUCE, or even MUMATH, for instance. A recursive procedure[28] to yield analytic expressions of Dirac–Coulomb r^l integrals is particularly readily adapted to symbolic computation such as by REDUCE or MACSYMA. Exchange integrals in the impact-parameter formulation of atomic charge-transfer collisions have been evaluated[29] by means of REDUCE. Explicit formulas for Clebsch–Gordan coefficients have been derived[30] by means of FORMAC. A general package[31] for exact Coulombic interaction matrix elements has been developed in REDUCE, and even SCHOONSCHIP has been used[32] for algebraic computations in molecular physics and quantum chemistry. When the hypervirial perturbative method cannot be used to derive recurrence relations because of nonseparable problems in multiple dimensions,[33] then moment perturbation theory may be applicable; for instance, for the problem of a hydrogen atom in a magnetic field, the first five energy perturbation corrections were determined[34] by means of REDUCE.

An analytic theory of the vibration–rotational spectroscopy of diatomic molecules is obviously a potential application of computer algebra. In this case, the powerful substitution or replacement capabilities of computer algebra, which seem to have been less widely appreciated than other capabilities, have been of immense value. In order to illustrate this capability, we examine cursorily this application, including a brief summary of the requisite physical theory.

In 1932 Dunham developed an analytic theory,[35] based on his function for the internuclear potential energy $V(x)$ in terms of a reduced displacement coordinate $x \equiv (R - R_e)/R_e$, involving the equilibrium R_e and instantaneous R internuclear distances. The form of the potential-energy function is a power series, starting at the quadratic term

$$V(x) = a_0 x^2\left(1 + \sum_{j=1} a_j x^j\right);$$

the leading coefficient is related to the harmonic force constant through the harmonic vibrational parameter ω_e and to R_e through the rotational parameter B_e, $a_0 = \omega_e^2/(4B_e) = B_e/\gamma^2$. The terms in all the important expressions resulting from the Dunham theory have as a coefficient $\gamma = 2B_e/\omega_e$ to various powers; since for known molecules $0.026 \gtrsim \gamma \gtrsim 10^{-4}$, a rapid convergence of these expressions is ordinarily achieved. Then the energies of the vibration–rotational states were expressed[35] as a double sum involving the vibrational v and angular momentum J quantum numbers to various powers:

$$E(v,J) = \sum_{k=0} \sum_{l=0} Y_{kl}[v + 1/2]^k[J(J+1)]^l;$$

the energy coefficients Y_{kl} become functions of the potential-energy coefficients a_j multiplied by B_e and γ to various powers. The set[35] of explicit analytic results Y_{kl} that Dunham obtained manually essentially by substitution methods contains no errors, and the extended collection[36] due to Sandeman contains only one obvious misprint. To compute a much larger collection[37] of these Y_{kl} expressions required[38] only 20 min of processor time with REDUCE. (A more recent manual recalculation,[39] according to a more complicated procedure, of expressions equivalent mostly to Dunham's Y_{kl}, however, produced several mistakes,[40] whereas later calculations of related quantities by the same workers[41] using REDUCE seem accurate.) These Y_{kl} quantities take into account only the mechanical effects (related to the vibration and rotation of the nuclei with respect to the center of molecular mass). Other effects arise from two possible sources: for all molecules there are adiabatic effects (that depend on the finite mass of the nuclei) and nonadiabatic effects (attributed to the interaction with other electronic states) that produce deviations from the mass-scaling ratios for isotopic molecules; for those molecules with net orbital or spin angular momenta due to either the nuclei or the electrons, extra branches of lines (beyond the simplest case of only two branches P and R), or, equivalently, lines split into multiplets, are found in the spectra. To take into account each such effect, we add to each Y_{kl} a further coefficient[42] Z_{kl} of the functionals of the quantum numbers v and J to various powers. Then the problem arises to express these additional coefficients in terms of the potential-energy parameters a_j and the parameters of some other radial function[39]

$$K(x) = \sum_{j=0} k_j x^j.$$

In the Dunham procedure,[35] the effect of the centrifugal term $B_e J(J+1)/(1+x)^2$ is taken into account in a formal way through the binomial expansion of the denominator

$$(1+x)^n = 1 + \sum_{j=1}^{m} x^j \prod_{k=0}^{j-1} (n-k)/k!$$

by means of, for instance, the following simple procedure in REDUCE:

```
PROCEDURE BN(x,n,m)$ 1 +  FOR J: = 1:m SUM x^J*(FOR K: = 0: (J - 1)
PRODUCT (n - K)) / ( FOR K: = 1:J PRODUCT K ) $
```

After the linear term is eliminated by a coordinate translation, the resulting coefficients of the J-dependent variable x_j in the potential-energy function in the Dunham form are also J-dependent; for instance,

$$a_{1J} = a_1 + \gamma^2 J(J+1)(-3a_1^2 - 3a_1 + 4a_2 - 4) + \gamma^4 [J(J+1)]^2(\cdots) + \cdots \quad .$$

Then the energy coefficients Y_{kl}, to take account of the vibration–rotational effects, may be generated from those $Y_{k,0}$, for purely vibrational effects and previously derived through the JBKW procedure, by the substitution into the $Y_{k,0}$ expressions of $a_{1,J}$ for a_1, $a_{2,J}$ for a_2, etc. Analogously, the additional function $K(\chi)$ may be taken into account by incorporating the coefficients k_j into the potential-energy coefficients that accordingly become dependent on both $J(J+1)$ and the coefficients k_j in $K(\chi)$; for instance,

$$a_{1JK} = a_1 + \gamma^2 J(J+1)(-3a_1^2 - 3a_1 + 4a_2 - 4) + 1/2\gamma^2(3a_1^2 k_1 - 2a_1 k_2 - 4a_2 k_1 + 2k_3) + \gamma^4 J(J+1)(a_j, k_j \cdots) + \cdots \quad .$$

Then, after the substitution into $Y_{k,0}$ of a_{1JK} for a_1, and so on, the terms that appear as coefficient of a given $(v+1/2)^k [J(J+1)]^l$ additional to those in Y_{kl} become the quantities Z_{kl}. Thus the expressions for the Y_{kl} and Z_{kl} coefficients are readily generated by a sequence of operations involving substitution and evaluation, with simplification wherever practicable, and are most physically meaningful in the form exhibiting factor $J(J+1)$ and γ to different powers. However, the use of the resulting expressions is hindered by a defect of the Dunham theory, namely the property of the reduced displacement variable that as $R \to \infty$, so $\chi \to \infty$. An alternative variable $z \equiv 2(R - R_e)/(R + R_e)$ has equivalent finite limits (specifically ± 2) at the limits corresponding to the united ($R = 0$) and separate ($R \to \infty$) atoms. The potential-energy function[43] incorporating this variable has precisely the Dunham form,

$$V(z) = c_0 z^2 \left(1 + \sum_{j=1} c_j z^j\right),$$

and the function corresponding to $K(\chi)$ is

$$K(z) = \sum_{j=0} h_j z^j \quad .$$

Then the problem arises to transform the Y_{kl} and Z_{kl} from explicit dependences on the a_j and k_j parameters to the desired c_j and h_j sets. When substituted into these energy coefficients Y_{kl} and Z_{kl}, the relations $a_1 = c_1 - 1$, $a_2 = c_2 - 3c_1/2 - 3/4, \ldots, k_1 = h_1, k_2 = h_2 - h_1/2$, etc., effect the desired transformation. Although the transformation of early members in a given sequence, such as a_1 or k_1, appears simple, the complexity increases almost exponentially with j, rapidly testing the patience of the human worker. This use of substitution operations is intended to illustrate how computers can effect operations of an algebraic type that are tedious to perform manually. In fact a complete and quantitative description of the vibration–rotational spectra of diatomic molecules has been formulated[44] by means of computer algebra as analytic expressions for energy coefficients, expectation values, and matrix elements of displacement variables to various powers, vibration–rotational interaction parameters, etc. Accordingly, the analysis of not only wavenumber or frequency data but also intensity data (from the absorption, emission, or Raman scattering experiments) may be affected in a quantitative manner, and the expressions show their physical meaning just as well as the ultimate numeric results for any particular molecular species. Bouanich has used REDUCE extensively in order to generate vibration–rotational matrix elements.[45,46] In the Soviet Union, extensive analytic calculations in molecular spectroscopy have also been made by means of special processors.[47] The extension of these methods to polyatomic molecules remains a future challenge in molecular physics.

There are many applications in other branches of physics research, some examples of which may be cited here. In connection with optical aberrations, the phase–space transformation due to the refracting interface between two media has been obtained[48] by means of REDUCE. In fluid dynamics, matrix elements of linearized cross-collision operators for gaseous mixtures have been computed[49] in FORMAC. Many applications that require vector calculus in orthogonal curvilinear coordinates and vector integration, differentiation, and series expansion and that occur in plasma physics, hydrodynamics, and electromagnetics can benefit from the collection[50] Orthovec of REDUCE procedures for the manipulation of scalars and vectors. A program[51] to handle calculus with vectors and dyadics for applications such as plasma and fluid dynamics has been implemented in MACSYMA. In a comparison[52] of the three processors CAMAL, MAPLE, and MUMATH-83 for the solution of a problem in hydrodynamic lubrication, MUMATH was found to be ineffective for the solution of such problems; CAMAL proved effective, but MAPLE more so because of its greater generality and flexibility. Perturbation calculations[53] for the spin-up problem in thermodynamics and statistical physics have been made using REDUCE. Both the determinations of Lie symmetries of ordinary and partial differential equations[54] and of point and contact Lie symmetries[55] have been achieved by means of REDUCE, whereas FORMAC was used[56,57] to determine the Lie–Backlund symmetries of differential equations. A REDUCE program[58] for evaluating a Lax pair form has been devised, applicable to the inverse scattering problem in fluid dynamics. The use of computer algebra methods such as MAPLE and MACSYMA for the treatment of hereditary operators of higher-order solitons has been discussed[59]; in this case, of programs for similar purposes in Pascal, MACSYMA, and MAPLE, the latter provided the most rapid execution. The method of power sums has been used[60] to derive a complete set of explicit algebraic Galois resolvents, applicable to the theory of crystal elasticity; this calculation is notable for requiring only 11 days for completion on a personal computer by means of MUMATH, but without extensive modifications (as well as the provision to store intermediate results on disk) 11 years of machine time were estimated to have been required.

Computer algebra systems should be an integral part of the education of a modern physicist. First of all, the physics student might have encountered MUMATH or some similar processor for microcomputers during the years of secondary school. Then, in university education, the mathematical laboratory should equip the student with the ability to compose algorithms in various languages or processors for numeric, algebraic, or logical applications. Descriptions[61] have been reported how some universities in Europe and North America are installing such mathematical laboratories, having such processors as CAYLEY, MAPLE, MACSYMA, and REDUCE to serve students (and professors) of not only mathematics but also the physical sciences and engineering. There is a concomitant need for physics courses to be modified to take account of the availability of algebraic computing so that

the student may be successfully prepared to tackle realistic problems in a practicable manner. Some merits of using computer algebra in teaching physics have been discussed[62] recently, including some applications of REDUCE. There is also the need for physicists outside academic environments to recognize the value of the different approaches to the solution of real problems offered by processors for computer algebra.

IV. GENERAL CONSIDERATIONS

In the preceding sections the term processor has been frequently employed to specify a particular system for computer algebra. The reason is that such a system not only is a language but also in many cases produces almost instant response to instructions; i.e., the system responds directly to simple commands of the form of an algebraic or numeric operation as well as to sets of statements in the traditional form of a program. This interactive mode of operation is particularly important for computer algebra because a set of instructions therein is typically run successfully once to produce exact symbolic expressions probably checked easily (to some extent) by inspection, although the resulting expressions might later be incorporated into numeric programs for multiple production runs with different sets of input data, as required. Thus, in the algebraic mode, the user engages in transformations or substitutions on the basis of intermediate results in order to produce the final output in the most readable, or physically meaningful, form. For this reason interactive processing is greatly preferable to the batch mode. Although some early processors, such as FORMAC and CAMAL, as well as ALTRAN and any others based on a FORTRAN compiler, operated only in a batch mode, and thus in the form of a traditional programming language, practically all more recent processors can operate interactively, but not without certain disadvantages.[52]

Another distinction between numeric and algebraic computing arises in the size of the core memory, or the partition thereof, required for a particular task. For a typical numeric program, at the time of compilation and linking, the amount of core memory required to run the program is exactly known, equal to the sum of the areas, precisely specified in a memory map, for the program (and libraries called) and for the variables, arrays, etc. Characteristically in computer algebra, one starts with a set of simple relations and hopes ultimately to generate a relatively simple result, but the intermediate expressions may be voluminous and, moreover, of a size difficult to predict. During the processing of large intermediate expressions within a finite amount of core memory, space vacated by discarded data structures must be reclaimed in a procedure called (inelegantly) "garbage collection." When the system is required to devote an inordinate proportion of execution time to such consolidation and reallocation of limited space, a phenomenon known as "thrashing" occurs, resulting in a relatively large amount of processor time to produce relatively little result. For this reason, computer algebra requires relatively large initial allocations of core memory, and the processor time for a given calculation may depend sensitively on the amount of available core memory. For large problems, as much virtual memory as 53MB has been used in a calculation with MACSYMA and 60MB for REDUCE, although for smaller problems of course much less memory may be required; one may estimate that about 1MB of data space, in addition to the intrinsic size of core required by the processor, would be a reasonable minimum size for a calculation of only moderate complexity. It should thus be obvious why the applicability of MUMATH to typical physical problems is likely to prove limited. In MAPLE, although the compactly designed basic system requires only a few hundred kilobytes, the data space may grow to megabytes if required and available. In some computers with operating systems that have fixed partitions, such as DEC System 10 or 20 and CDC Cyber under NOS and NOS/BE, even REDUCE suffers from cramped conditions. Therefore operation of algebra processors on computers with virtual memory capabilities is definitely advantageous.

A characteristic of many of these processors for computer algebra is that they are relatively inefficient in execution time for standard numeric calculations, although the preparation of the program may consume much less time. One reason is that, in the interactive systems, the instructions are generally interpreted rather than compiled, although procedures in REDUCE are compiled and functions in MACSYMA may be translated into LISP and subsequently compiled. In order to execute most efficiently the expressions generated by computer algebra for production runs of numeric data input, one is commonly advised to convert the expressions into code that can be compiled efficiently in FORTRAN, C, or a similar language. MACSYMA, MAPLE, and REDUCE, but not MUMATH, have such facilities for output in standard numeric languages. Furthermore, the combination of algebraic and numeric processors can be a powerful method of attacking physical problems. Brown and Hearn have described[14] a problem in magnetohydrodynamics in which integrations of three-dimensional Galerkin functions were done in two dimensions analytically by REDUCE and the third (intractable analytically) numerically. They also discussed the application[14] of symbolic computation to numerical analysis, such as reduction of a system of relations in order to analyze inherent error and roundoff error of a given expression, error analysis of a known numeric algorithm, and discretization and roundoff errors of various numerical methods in order both to eliminate inaccurate or unstable methods prior to coding and testing and to develop methods in which the lowest-order errors cancel each other. Perhaps one such application[63] is that of REDUCE to calibration problems of picture-processing devices for track chambers.

Of course computer algebra is not without its limitations. Despite the delusion by novices to this type of computation, algebraic operations that are not feasible in principle by manual methods are equally infeasible by computer. A more basic problem is the technical limitation that, in general, to determine whether two expressions are equal or even equivalent is difficult. Although Gonnet has provided some examples[64] of prototypical mathematical problems solved by means of MAPLE in which this equivalence is proved, in general, expressions may tend to be become excessively large and unnecessarily complicated because such equivalences are not exploited for the purpose of simplification. Either the setting of switches or human interaction may lead to simplification, but care and experiments are generally necessary to ensure that the presentation of a given algebraic result is in its most meaningful form.

V. SOURCES OF INFORMATION ABOUT COMPUTER ALGEBRA

There now exist two pertinent scholarly journals, *The Journal of Symbolic Computation*, published by Academic Press (London), and *LISP and Symbolic Computation*, published by Kluwer. Both journals seem mostly directed to computer science aspects rather than to applications in physics, but the *Journal of Symbolic Computation* also contains systems descriptions and applications letters. For instance, the systems MACSYMA,[65] REDUCE,[2] and MAPLE[66] have already been described therein, and an applications letter[67] on an algebraic treatment of quantum vibrations in REDUCE demonstrates how noncommutative algebra may be performed with that processor. The *SIGSAM Bulletin* of the special interest group on symbolic and algebraic manipulation of the (USA) Association for Computing Machinery is a quarterly magazine that contains both archival material as well as news and comments about new processors or versions. The journal *Computer Physics Computations*, although certainly devoted to physics but not especially to computer algebra, occasionally contains papers discussing such applications in physics and presenting short programs that demonstrate programming techniques. Of course proceedings of users' conferences provide descriptions of applications of particular processors, and occasionally comparisons of different processors, and the distributors or marketers of these processors may supply a bibliography or newsletters of their applications.

Generally the printed manuals of most processors are written succinctly and lack extensive examples of program design, although tutorial or demonstration programs accompany some processors. However, some monographs have been published that present explicit programs that one may use not only directly for the particular application but also indirectly to demonstrate the general techniques of algorithm development. Perhaps the first book was that by Howard,[68] who presented programs for aeronautical applications, particle dynamics, fluid mechanics, cosmological models, and trajectory calculations, mostly in FORMAC and MACSYMA. A recent book by Rayna provides[69] examples of REDUCE programs, whilst for MUMATH, two books have been recently published, one[70] in English and the other[71] in Japanese.

ACKNOWLEDGMENT

I thank the National Science Council of the Republic of China for the support of a visiting research professorship at National Tsing Hua University.

REFERENCES

1. D. Barton and J. P. Fitch, Rep. Prog. Phys. **35**, 235 (1972).
2. J. Fitch, J. Symb. Comput. **1**, 211 (1985).
3. R. H. Risch, Trans. Am. Math. Soc. **139**, 167 (1969).
4. A. C. Norman and P. M. A. Moore, in *Fourth International Colloquium on Advanced Computing Methods in Theoretical Physics*, Marseilles (1977).
5. T. S. Gradshteyn and I. M. Ryzhik, *Table of Integrals, Series and Products* (Academic, Orlando, FL, 1980), 4th ed. (corrected and enlarged).
6. K. Bahr, SIGSAM Bull. ACM **9**(1), 21 (1975).
7. E. C. Berkeley and D. G. Dobrow, *The Programming Language LISP* (M.I.T. Press, Cambridge, MA, 1966).
8. J. A. Campbell, Comp. Phys. Commun. **1**, 251 (1970).
9. C. Engleman, in *Information Processing 68*, edited by A. J. H. Morrell (North-Holland, Amsterdam, 1969), pp. 462–467.
10. H. Caprasse, SIGSAM Bull. ACM **20**(4), 7 (1986).
11. R. J. Fateman, SIGSAM Bull. ACM **19**(3), 5 (1985).
12. H. Strubbe, Comp. Phys. Commun. **18**, 1 (1979).
13. A. Hohti, SIGSAM Bull. ACM **22**(1), 12 (1988).
14. W. S. Brown and A. C. Hearn, Comp. Phys. Commun. **17**, 207 (1979).
15. B. Autin and J. Bengtsson, Comp. Phys. Commun. **48**, 181 (1988).
16. V. P. Gerdt, Comp. Phys. Commun. **20**, 85 (1980).
17. J. Calmet, Comp. Phys. Commun. **4**, 199 (1972).
18. J. Paldus and H. C. Wong, Comp. Phys. Commun. **6**, 1 (1973).
19. R. Gastmans, A. van Proeyen, and P. Verbaeten, Comp. Phys. Commun. **18**, 201 (1979).
20. J. Wroldsen, Comp. Phys. Commun. **27**, 39 (1982).
21. A. D. Payne, Comp. Phys. Commun. **4**, 100 (1972); **12**, 145 (1976).
22. R. Grimm and H. Kuhnelt, Comp. Phys. Commun. **20**, 77 (1980).
23. A. P. Demichev and A. Y. Rodiovov, Comp. Phys. Commun. **38**, 441 (1986).
24. B. Nielsen and H. Pedersen, SIGSAM Bull. ACM **22**(1), 7 (1988).
25. A. P. Kryukov and A. Y. Rodiovov, Comp. Phys. Commun. **58**, 327 (1988).
26. C. F. Froese and D. W. B. Prentice, Comp. Phys. Commun. **6**, 157 (1973).
27. L. B. Golden, Comp. Phys. Commun. **14**, 255 (1978).
28. N. Bessis, G. Bessis, and D. Roux, Phys. Rev. A **32**, 2044 (1985).
29. C. J. Noble, Comp. Phys. Commun. **19**, 327 (1980).
30. G. Rudnicki-Bujnowski, Comp. Phys. Commun. **10**, 245 (1975).
31. N. Bogdanova and H. Hogreve, Comp. Phys. Commun. **48**, 319 (1988).
32. P. O. Nerbrandt, Comp. Phys. Commun. **14**, 315 (1978).
33. F. M. Fernandez and E. A. Castro, *Hypervirial Theorems* (Springer, Berlin, 1987).
34. F. M. Fernandez, J. F. Ogilvie, and R. H. Tipping, J. Phys. A **20**, 3777 (1987).
35. J. L. Dunham, Phys. Rev. **41**, 721 (1932).
36. I. Sandeman, Proc. R. Soc. Edinburgh **60**, 210 (1940).
37. J. F. Ogilvie, Comp. Phys. Commun. **30**, 101 (1983).
38. J. F. Ogilvie and R. H. Tipping, J. Symb. Comput. **3**, 277 (1987).
39. N. Bessis, G. Hadinger, and Y. S. Tergiman, J. Mol. Spectrosc. **107**, 343 (1984).
40. J. F. Ogilvie, Spectrosc. Lett. **20**, 725 (1987).
41. G. Hadinger and Y. S. Tergiman, J. Chem. Phys. **87**, 2143 (1987).
42. R. M. Herman and S. Short, J. Chem. Phys. **48**, 1266 (1968); **50**, 572 (1969).
43. J. F. Ogilvie, Proc. R. Soc. London Ser. A **378**, 287 (1981); **381**, 479 (1982).
44. J. F. Ogilvie, Int. Rev. Phys. Chem. **5**, 197 (1986).
45. J. P. Bouanich, J. Quant. Spectrosc. Radiat. Transfer, **37**, 17 (1987); **38**, 89 (1987).
46. J. P. Bouanich, Comp. Phys. Commun. **47**, 259 (1987).
47. V. N. Biryukhanov, V. Y. Galin, V. E. Zuev, Y. S. Makushkin, and V. G. Tyuterev, Sov. Phys. Dokl. **25**, 824 (1980).
48. M. Navarro-Saad and K. B. Wolf, J. Symb. Comput. **1**, 235 (1985).
49. L. M. Toff, Comp. Phys. Commun. **9**, 271 (1985).
50. J. W. Eastwood, Comp. Phys. Commun. **47**, 139 (1987).
51. M. C. Wirth, SIAM J. Comput. **8**, 306 (1979).
52. R. M. Corless and D. J. Jeffrey, SIGSAM Bull. ACM **22**(2), 50 (1988).
53. I. Cohen and F. Bark, Comput. Phys. Commun. **14**, 319 (1978).
54. F. Schwarz, Comput. Phys. Commun. **27**, 179 (1985).
55. V. P. Eliseev, B. N. Federova, and V. V. Kornyak, Comp. Phys. Commun. **36**, 383 (1985).
56. R. N. Federova and V. V. Rorsyak, Comp. Phys. Commun. **39**, 93 (1986).
57. V. P. Gerdt, A. B. Shvachka, and A. Y. Zharkov, J. Symbolic Comput. **1**, 101 (1985).
58. M. Ito, Comp. Phys. Commun. **34**, 325 (1985).
59. B. Fuchssteiner, W. Oevel, and W. Wiwianka, Comp. Phys. Commun. **44**, 47 (1987).
60. A. C. Hurley and A. K. Head, Int. J. Quantum Chem. **31**, 345 (1987).
61. Almost the entire issue No. 81 of the SIGSAM Bull. **21**, (3) (1987) is devoted to description of mathematical laboratories and the software therein provided.
62. C. Wooff and D. Hodgkinson, Eur. J. Phys. **9**, 145 (1988).
63. I. Bajla and G. A. Ososkov, Comp. Phys. Commun. **20**, 81 (1980).
64. G. H. Gonnet, SIGSAM Bull. ACM **22**(2), 8 (1988).
65. R. Pavelle and P. S. Wang, J. Symb. Comput. **1**, 69 (1985).
66. B. W. Char, G. J. Fee, K. O. Geddes, G. H. Gonnet, and M. B. Monagan, J. Symb. Comput. **2**, 179 (1986).
67. M. L. Sage, J. Symb. Comput. **5**, 377 (1988).
68. J. C. Howard, *Practical Applications of Symbolic Computation (Mathematical Modeling of Diverse Phenomena)* (IPC Science and Technology, Guilford, UK, 1980).
69. G. Rayna, *REDUCE Software for Algebraic Computation* (Springer, New York, 1987).
70. C. Wooff and D. Hodgkinson, *MUMATH: A Microcomputer Algebra System* (Academic, Orlando, FL, 1987).
71. A. Furukawa, *MUMATH Primer* (Gendai Suugaku Sha, Tokyo, 1987).