Particle Accelerators, 1986, Vol. 19, pp. 237–245
0031-2460/86/1904–0237/$15.00/0

Printed in the United States of America

SYMBOLIC MATHEMATICAL COMPUTING: ORBITAL DYNAMICS AND APPLICATIONS TO ACCELERATORS†

RICHARD J. FATEMAN
Electrical Engineering and Computer Sciences Department
University of California
Berkeley, CA 94720

(Received March 7, 1985)

Computer-assisted symbolic mathematical computation has become increasingly useful in applied mathematics. In this talk we present a brief introduction to such capabilities and some examples related to orbital dynamics and accelerator physics.

I. INTRODUCTION TO THE CONCEPT OF COMPUTER-ASSISTED SYMBOLIC MATHEMATICS

Most users of computers in advanced scientific applications have at least an inkling of the prospects of the use of computers in non-numeric data-processing tasks. Recent students of computer science have usually been exposed to these concepts if they have taken at least one class in "data structures." In such courses students learn that abstractions such as "arithmetic expression" can be implemented in a linked-list form and manipulated by simple algorithms. In a number of schools, including UC Berkeley, lower-division undergraduate students are assigned the task of writing programs which implement arithmetical operations and compute symbolic derivatives of such symbolic expressions.

Clearly the tools have changed since the time thirty years ago when two people were awarded Masters' degrees from distinguished universities for "symbolic differentiation programs." Our new tools give us some hope for optimism: that "general-purpose" computer systems of the future, building upon the capabilities of current programs such as MACSYMA, REDUCE, SMP, or MAPLE, will increase the productivity of scientists.

II. SOME SIMPLE EXAMPLES

The large variety of tasks that can be performed by programs can only be touched upon here. Excellent bibliographical references can be found in the recent monograph. "Computer Algebra," edited by Buchberger *et al.*[1]

† This work was supported in part by the Army Research Office, Grant # 1-483964-25546, through the Center for Pure and Applied Mathematics, University of California, Berkeley.

2.1. *Simple Polynomials Arithmetic*

The transcript of a dialog with MACSYMA, probably the largest of the symbolic mathematics systems, is given below. The typeset responses were produced by the program in the conventional mathematical notations. The commands are fairly self-explanatory except the line "ratsimp(%)," which RATionally SIMPlifies the result of the previous computation (named "%"). This is done by multiplying through products and reducing the resulting quotient to lowest terms,

.....

(c1) Legendre Polynomials, by Rodigues' formula*/
p[n](x):=1/(2^n*n!)*diff((x^2-1)^n, x, n);

$$p_n[x] := \frac{1}{2^n n!}\frac{d^n}{dx^n}(x^2-1)^n \tag{d1}$$

(c2) p[5](y);

$$\frac{7200y(y^2-1)^2 + 3840y^5 + 19200y^3(y^2-1)}{3840} \tag{d2}$$

(c3) ratsimp(%);

$$\frac{63y^5 - 70y^3 + 15y}{8} \tag{d3}$$

2.2. *Other Examples*

In the workshop talk, other examples were given.

III. WHAT CAN WE SAY ABOUT THE USEFULNESS OF THESE PROGRAMS?

Most presentations of capabilities of computer-assisted symbolic mathematics programs strike people in one of the following ways:

(a) Where can I get this stuff?
(b) That's cute, but not what I do.
(c) I tried one of those programs but it didn't work for me.
(d) If it doesn't run on a Cray it can't be good.

Since this is a conference on orbital dynamics, it seemed incumbent upon me to find a few applications which would encourage additional scientists in this field to look at the existing programs, to see if scientists in category (b) can be encouraged to try symbolic computation systems. Since the programs have gotten better, partly by running on better hardware, perhaps category (c) persons can be convinced to try programs again. Persons in category (a) can examine the references for access to current programs.

IV. MORE APPROPRIATE EXAMPLES

4.1. The Kepler Equation

In 1972, Barton and Fitch published a survey article on applications of symbolic manipulation programs to physics.[2] They asserted that solution of Kepler's equation was of some interest, or at least a good example of the kinds of calculations that appear in the field of celestial mechanics. Although I have yet to encounter a real physicist stuck on exactly this problem, the techniques illustrated are more than the parlor tricks used to show off MACSYMA and similar programs. There is also a moral to be learned from various computational approaches.

The Kepler equation is

$$Y = u + e \sin Y,$$

where e is to be regarded as a small quantity, typically the eccentricity of an elliptic orbit. A formal solution is possible in terms of Bessel functions, namely

$$Y = u + 2 \sum_{n=1}^{\infty} \frac{J_n(ne) \sin (nu)}{n}$$

We propose to obtain an explicit approximation for the function Y as a Fourier-type series with coefficients in e up to order n. From the formal solution this would be fairly painful, requiring the careful generation and series expansion of Bessel functions and sines. A direct approach by iterated approximation is possible, and it is this which we illustrate. Since $Y = u$ is the solution to order 0, let us assume the solution is of the form

$$Y = u + A_k$$

correct to order k in e. Then by substitution we get the simple formula

$$A_k = e \sin (u + A_{k-1}), \qquad k > 0,$$

and

$$A_0 = 0.$$

In order to express A_k to the desired kth order in e and in the conventional form of a linear expression in the trigonometric functions $\sin (nu)$ with coefficients which are power series in e, it is necessary to expand the $\sin (u + A_{k-1})$ as $\sin u \cos A_{k-1} + \cos u \sin A_{k-1}$, and then approximate $\sin (A_{k-1})$ and $\cos (A_{k-1})$ as truncated series. Thus the iteration for A_k is more accurately expressed by

$$A_k = \left[e \sin u \left(1 - \frac{A_{k-1}^2}{2!} + \cdots\right) + e \cos u \left(A_{k-1} - \frac{A_{k-1}^3}{3!} + \cdots\right)\right]_k,$$

where the outer brackets indicate that terms of degree greater than k can be ignored. The sin/cos series must also be extended to an order sufficient to assure that all terms up to e^k are accounted for.

All these details in the calculation must be specified in some formal programming notation and their semantics adequately (efficiently) provided. In the Barton–Fitch survey it is pointed out that some systems (e.g., REDUCE) provide the means whereby the necessary rules for multiplying "Poisson Series" can be introduced, and the truncation on powers of e can be stated; other systems (notably CAMAL, a system written by Barton and Fitch) have these facilities built in by design. We present an abbreviated dialog with REDUCE and then MACSYMA (which includes CAMAL-like facilities) below.

Interaction with REDUCE

```
REDUCE 3.0, 15-Apr-83 . . .

for all u,v let cos(u)*cos(v) = (cos(u + v) + cos(u - v))/2;
for all u, v let cos (u)*sin(v) = (sin(u + v) + sin(v - u))/2;
for all u,v let sin(u)*sin(v) = (cos(u - v) - cos(u + v))/2;
for all u let sin(u)**2 = (1 - cos(2*u))/2;
for all u let cos(u)**2 = (1 + cos(2 + u))/2;
a:= 0;
for k:= 0:4 do begin
let e**(k + 2) = 0;
write a:= e*sin(u)*(1 - a**2/2 + a**4/24) + e*cos(u)*(a - a**3/6 + a**5/120);
clear e**(k + 2)
end;

A:= SIN(U)*E

A:= (E(SIN(2 + U)*E + 2*SIN(U)))/2

A:= (E*(3*SIN(3*U) + E^2 + 4*SIN(2*U)*E - SIN(U)*E^2 + 8*SIN(U)))/8

A:= (E*(8*SIN(4*U)*E^3 + 9*SIN(3*U)*E^2 - 4*SIN(2*U)*E^3 + 12*SIN(2*U)*E
    -3*SIN(U)*E^2 + 24*SIN(U)))/24

A:= (E*(125*SIN(5*U)*E^4 + 128*SIN(4*U)*E^3 - 81*SIN(3*U)*E^4 + 144*SIN
      (3*U)*E^2 - 64*SIN(2*U)*E^3 + 192*SIN(2*U)*E + 2*SIN(U)*E^4 - 48*
      SIN(U)*E^2 + 384*SIN(U)))/384
```

Interaction with MACSYMA

```
(c4) /* solution to Kepler equation Y = u + e sin (Y) by approximation*/
/* illustrating the use of special representations */

(ratweight(e, 1) /* e is a small variable */,
  poisvars:[u], /* the only variable is poisson series is u */
  load("rat/pois2"),
  ratvars(e) /* the only variable is coefficients is e */,
```

```
load("rat/ratpoi") /* see rational coefficients */);

(c5) /* solution is of the form Y = u+a[k] as k→inf*/

a[0]:0;

(c6) a[k]:= block([ratwtlvl:k],
        a[k−2]:0, /* delete unused old values */
        /* substitute u+a[k−1] for u in e*sin(u), and expand
        the resulting series to k terms */
        poissubst(u,u,e*sin(u),a[k−1],k));

(c7) a[1];
```

$$e\sin(u) \tag{d7}$$

```
(c8) a[2];
```

$$\frac{e^2\sin(2u)}{2} + e\sin(u) \tag{d8}$$

```
(c9) a[3];
```

$$\frac{3e^3\sin(3u)}{8} + \frac{e^2\sin(2u)}{2} - \frac{(e^3 - 8e)\sin(u)}{8} \tag{d9}$$

```
(c10) a6:a[6];
```

$$\frac{e^6\sin(6u)}{3840} + \frac{e^5\sin(5u)}{384} - \frac{(e^6 - 20e^4)\sin(4u)}{960} - \frac{(5e^5 - 48e^3)\sin(3u)}{384} + \frac{(e^6 - 32e^4 + 384e^2)\sin(2u)}{768} + \frac{(5e^5 - 72e^3 + 192e)\sin(u)}{192} \tag{d10}$$

```
(c11) fortranize(%);
Totaltime = 7650 msec. GCtime = 3550 msec.
        retvar =
     c  0.0002604166666666667*e**6*sin(6*u)+0.002604166666666667*e**5*sin(
     1     5*u)−0.0010416666666666667*(e**6−20*e**4)*sin(4*u)−0.00260416666
     2     6666667*(5*e**5−48*e**3)*sin(3*u)+0.001302083333333333*(e**6−32
     3     *e**4*384*e**2)*sin(2*u)+0.005208333333333333*(5*e**5−72*e**3+1
     4     92*e)*sin(u)
```

4.2. Comments on the Kepler Equation Problem

Although it is not evident from the abbreviated print-outs above, the time to complete this computation is highly dependent upon the approach. With the special Poisson series calculation data format (used in MACSYMA), the computation is much simpler than in the "general" representation used by REDUCE. The resulting speed difference grows dramatically as additional terms are computed. For example, the term A_3 is computed in 1.19 seconds in the general representation, but in 0.5 seconds with the Poisson series form. The term A_8 shows a more dramatic difference: 43.5 seconds vs 9.4 seconds. Of course, a

skilled programmer using LISP, the implementation language of REDUCE, could implement the appropriate special data types in that system too, so the difference is not in the system, but in the effort taken by the system programmer to solve problems of this nature and in the awareness of the user that these facilities are appropriate. (It is also quite possible to use MACSYMA's general representation and be far less efficient compared to REDUCE on this problem; it is also possible to recode these manipulations in assembler language for a particular computer: this approach is costly in programmer time.)

In our experience, with symbolic computation systems it is even more difficult to obtain measures of performance than is the case with FORTRAN benchmarks. Just becoming acquainted with the available tools may be a barrier; once the tools are understood, substantial progress can be made.

It may also be worthwhile to note that the MACSYMA program, by using typeset quality output and using the more conventional display of terms, makes a qualitative difference in the understandability of the results. Line (c11) illustrates the possibility of converting computed expressions into FORTRAN code. This feature is also present in REDUCE and almost all other symbolic mathematics systems.

Two aspects of this example are really of general relevance:

(a) This problem, as is the case for most others, cannot be solved without some preliminary mathematical analysis. Unless the day-to-day work of an applied mathematician is quite routine and over well-trod paths, the computer system will only be an aid, not a replacement for mathematical work.

(b) While general tools may do the job, as illustrated by REDUCE, such a first approach may have to be abandoned and the computation reformulated for a more ambitious study. Naive approaches to symbolic mathematics are more likely to have ballooning costs as the problem size rises than numerical calculations. Polynomial growth in cost functions (e.g., n^3) is typical in numerical scaleups. Exponential growth (e.g., 2^n) is more typical in algebraic scaleups.

It is our hope that once a mathematical formulation, an appropriate algorithm, and the data structures are carefully designed to meet the needs of the computation, they can be reused. That is, success on a given problem, can also lead to the solution of a whole class of similar problems. Libraries of symbolic solution methods should be collected and used to strengthen the scientific computation facilities of the future, much as numerical subroutine libraries are used today.

4.3. Hamiltonian Mechanics

An interesting application of symbolic computation is presented in work by Char and McNamara.[3] They implement a Lie-transform technique for studying the perturbation theory of Hamiltonian system:

> A single scalar-generating function is calculated to give a canonical transformation of the Hamiltonian, the coordinates, and any function of the

coordinates from old to new or new to old coordinates. When the perturbations are periodic along the unperturbed orbits, the Hamiltonian can be averaged, an adiabatic invariant may exist, and the averaged equations of motion may be considerably simplified. The invariants of the motion have played a key role in many plasma devices and continue to be of interest for the containment of plasma, the structure of magnetic surfaces in toroidal devices, and the propagation of large-amplitude waves for plasma heating.

The algebra of the methods depends upon three operators: Poisson bracket, average along an orbit, and integral along an orbit. A convenient form of the algorithm is described by Nayfeh,[4] and a convenient notation used by McNamara[5] was adopted.

The program as produced in 1978 was able to reproduce some previous results and find some published errors. It successfully computed results which were several pages long.

As in some other application areas, there were really two needs which were initially mixed together, because humans tend to mix them: the need to do explicit evaluations and the need to simplify expressions.

(a) The need to do explicit computations: What we mean by this is the evaluation of expressions so that the definitions below, of Poisson (or Lie) bracket ($pb(f, g)$, also written $[f, g]$), the integral operator, and the averaging operator could be applied to any given explicit expressions f, g, P, Q:

$$pb(f, g)(P, Q) := [f, g](P, Q) := \sum_{i=1}^{n} \frac{\partial f}{\partial Q_i}\frac{\partial g}{\partial P_i} - \frac{\partial f}{\partial P_i}\frac{\partial g}{\partial Q_i}$$

$$ii(f) := \frac{1}{y}\int_0^y f(p, q)\, dq_1,$$

where y is the period of f in q_1:

$$av(f) := \int (f - ii(f))\, dq_1.$$

(b) The need to simplify expressions (implicit computation): By this we mean the application of those observations that make the computation practical for humans (and for finite computers). A number of identities are quite important in carrying out the computations, and they can be applied regardless of the context of particular values of f, g, etc. If such identities are ignored, the computations being posed may require more resources than are practical. A few of the useful identities are listed below:

$$ii[f, av(g)] + ii[av(f), g] = 0;$$

$$[f, g] = -[g, f];$$

$$[f[g, h]] + [h, [f, g]] + [g, [h, f]] = 0.$$

In the past, a similar separation of such components was recognized in tensor manipulation: MACSYMA has two separate packages of programs for computing

with tensors. We suspect that recognition of these two components may be an important step in thinking about the automation of other mathematical manipulations.

Although the Lie-transform programs were by some measures successful, they were not heavily used by the authors and were probably not used by anyone else. Why was this?

(a) The mathematical algorithms were written for more generality than was needed for the examples immediately at hand and thus were wasteful of resources. Techniques comparable to the efficient Poisson series code should have been developed for the evaluation component and for the simplification component. This would have required the development of effective canonical forms for the computations, which would have, in turn, focused efforts more clearly.

(b) The generality of the simplification methods was simultaneously too delicate and dependent on special tricks, and also sufficiently restricted by the implementation so that other examples which required more power could not be done with the facilities available. The simplification was done in an essentially rule-driven fashion and was difficult to control.

(c) MACSYMA, the system which was used, lacked a clear notation for the operator calculus needed. This meant that some of the necessary simplifications could not be compactly coded. No current algebra system is really much better, although some claims are made for various systems.

(d) Sufficiently large computers, in particular with respect to address space, could not (in 1978) run MACSYMA. (This has been substantially remedied now). Far more programmer effort was spent trying to conserve storage than any on other part of the development.

(e) The programs (MACSYMA and the Lie–transforms applications) are not easily available to potential users. The distribution of symbolic programs has been much more difficult than that of FORTRAN subroutine libraries partly for the technical reasons of lack of standardization. Unfortunately a contributing factor in the difficulty in obtaining MACSYMA (and some similar programs) has been ownership disputes between universities and the government.

V. FRONT-END WORKSTATIONS TO SCIENTIFIC COMPUTING ENVIRONMENTS

While this is not a major theme for this conference, we would like briefly to bring up an associated topic: workstation environments and their impact. It is probably the case that the future use of symbolic-mathematics systems will continue to be largely through interactive computing. Our experience at UC Berkeley has been that interactive timesharing on midsized to large computers is an adequate way of using symbolic systems, but that scientific workstations which can provide immediate graphical output and can accept commands from a "mouse" pointing device, are going to provide far superior interfaces to humans. They will also assist in the development and organization of programs and the use of (for

example) menu-driven access to libraries. These will make it easier for people to use the results of other investigations.

Because MACSYMA is written in LISP, a language which supports both a compiler (for fast execution) and an interpreter (for immediate evaluation of commands), it has certain advantages over FORTRAN-based systems. Since easy linkage to FORTRAN subroutine libraries is available in at least one LISP system, we have seen that access to libraries can be arranged. A package linking to the Numerical Algorithms Group library has been demonstrated by Prof. Kevin Broughan of the University of Waikato, New Zealand. We believe that symbolic computing in the workstation can assist not only in the direct solution of some problems symbolically but also in the preparation of programs for eventual numerical computation.

It may be that the organizing principles of algebra which are being used for the latest developments in symbolic-mathematics systems can be extended to incorporate physics notations and laws. These extensions may provide additional guidance for applications packages and solution environments for orbital dynamics problems.

VI. SUMMARY AND CONCLUSIONS

Symbolic computation is a powerful technique for problem solving, but not a panacea. It should be one of the tools available in our environment, and as we produce algorithms and data structures for tasks in any area of applications, we should try to share the solution methods. We must be aware of the tradeoffs in design between generality and efficiency.

One might envision a situation for the future where we would see a form of publication of algorithms and results for symbolic computation as machine-readable procedures and tables of formulas. This may be harder to achieve than the already existing approaches to publication of numerical procedures, but the results may be even more valuable, as we can with such techniques develop an effective computational model of what now may be buried in journals.

REFERENCES

1. *Computer Algebra: Symbolic and Algebraic Computation,* edited by B. Buchberger, G. E. Collins, and R. Loos (Springer-Verlag, N.Y, Second Edition, 1983).
2. D. Banton and J. Fitch, *Rep. Prog. Phys.*, **35,** (pp. 235–314) (1972).
3. B. Chan and B. McNamara, computer code LIEPROC; computer code LCPT (MACSYMA Users' Conference, M.I.T., Cambridge, Mass., 1979).
4. A. H. Nayfeh, *Preturbation Methods,* Wiley, New York, 1973.
5. B. McNamara, *J. Math. Phys.*, **19,** 2154 (1978).