*J. Symbolic Computation* (1985) **1**, 69–100

# MACSYMA from F to G†

RICHARD PAVELLE‡ AND PAUL S. WANG§

‡*Symbolics Inc.*, 11 *Cambridge Center, Cambridge, Massachusetts* 02142, *USA and*
§*Department of Mathematical Sciences, Kent State University, Kent, Ohio* 44242, *USA*

(*Received* 14 *January* 1985)

A descriptive, non-technical tutorial on MACSYMA, a well-known and widely used Computer Algebra system, is presented. Several examples of its capabilities are exhibited using actual MACSYMA input and output. A discussion of computer-based symbolic mathematical computation is motivated by pointing out inherent difficulties with familiar numeric computations. The inner workings of Computer Algebra systems are briefly discussed in addition to some on-going work on MACSYMA and future directions.

## 1. Introduction

In this first article on Computer Algebra systems, for the *Journal of Symbolic Computation*, we are faced with a dilemma. We intend to present a descriptive, non-technical tutorial on MACSYMA, a well-known and widely used Computer Algebra system. However, many readers will likely know little about our field and will be familiar only with traditional (numeric) computing techniques. Therefore our approach will be to provide a short discussion on the inherent difficulties involved with floating-point computations as motivation for further discussion on computer-based symbolic mathematical computation. We shall then touch on the inner workings of Computer Algebra systems. After these general discussions we shall concentrate on the capabilities of MACSYMA and present several examples. We will conclude with a brief discussion of current directions.

While this paper is directed towards MACSYMA, the development of Computer Algebra systems has been the result of an international effort. There are many systems, world-wide, of various sizes and designs which have been developed over the past 15 to 20 years (van Hulzen & Calmet, 1983). Research related to the development of these systems has lead to new results in mathematics and algorithms. These results in turn helped the development of MACSYMA as well as other systems.

Computer Algebra systems‡ are large software programs which compute with numbers, symbols, and formulas. These systems vary greatly in their capabilities, and there is a great deal of information available on them (Pavelle *et al.*, 1981; Yun & Stoutemeyer, 1980). They are now important research tools and a daily resource for many engineers and scientists. They are characterised by exact computation rather than numeric

† Work reported herein has been supported in part by the US National Aeronautics and Space Administration under Grant NAG 3–298 and by the US National Science Foundation under Grant MCS–82–01239 and by the Department of Energy under Grant DE–AC02–ER760275–A013.

‡ We prefer to call these "Computer Algebra" systems but they are commonly called "Algebraic Manipulation" systems, "Algebraic Computation" systems, "Symbolic Manipulation" systems and "Symbolic Mathematical" systems.

0747–7171/85/010069+32 $03.00/0

© 1985 Academic Press Inc. (London) Ltd

approximation. These systems perform the kind of computations people are trained to do using pencil and paper. Some calculations which took decades to do by hand now take hours or indeed seconds. These Computer Algebra systems can manipulate complicated formulas and return answers in terms of symbols and formulas as well as numbers. With these revolutionary tools computing will never be the same again. It is our hope that information provided here will create new users of Computer Algebra systems by showing what one might expect to gain by using them and what one will lose by not using them.

## 2. Symbolic and Numeric Computations

The two broad categories of scientific computation are numeric and symbolic. Numeric computation signifies that a computer is used as a number cruncher. Many problems can be cast in such a way that its solution calls for nothing more than repeated computations with numbers or the processing of numeric data. In numeric computation, a computer system is used to carry out calculations only with numbers. On the other hand we now have Computer Algebra systems available which perform symbolic computation and these crunch symbols. The purpose of Computer Algebra systems is to compute with symbols, equations, and operators as well as with numbers. As we shall show, modern Computer Algebra systems perform many highly sophisticated computations that will amaze people who use mathematical tools.

Computer Algebra systems emphasise exactness in the calculations while numeric systems operate on numbers that are as nearly correct as the precision of the computer system allows. Both approaches are important for the solution of problems, but for many calculations Computer Algebra systems offer enormous advantages over numeric systems.

### 2.1 ACCURACY OF NUMERIC COMPUTATIONS

In typical examples of numeric computation one computes with floating-point numbers. The number of places retained after the decimal point, the precision, is usually dependent on the computer system used. In our examples, for the purposes of discussion, we shall assume a precision of six digits.

To maintain speed of operation with floating-point numbers the precision is usually fixed. For many applications numeric approximations are accurate enough to provide reasonable answers but often they are not. Because of the limited precision, numbers such as 1/3 carry a built-in error called "roundoff error". When we add the fractions $1/3+1/3+1/3$ we obtain

$$1/3+1/3+1/3 = 1$$

In floating-point arithmetic, however, we have

$$0.333333+0.333333+0.333333 = 0.999999$$

instead of 1.0. While this example is trivial it points out that numeric systems will often give the wrong answer while all modern Computer Algebra systems can add rational numbers properly.

Consider now 2/3 which is represented as the floating-point number 0.666667. What is $0.666667-0.333333$? It is, of course, 0.333334. Therefore we see that 1/3 is equally well represented by 0.333334 as well as by 0.333333. A point one must remember is that when large numbers of operations are done with floating-point numbers, one must be very careful. The magnitude of the errors generated by roundoff can actually be greater than

the answer one seeks. The users of numeric systems understand many of these difficulties and have developed procedures for dealing with these problems (Knuth, 1980). However, they are expensive in terms of human and machine resources.

Here is another example which provides a way to manufacture "hard" problems for floating-point systems. Consider the expression

$$UVWXY-(UVW)^{1/3}(VWX)^{1/3}(WXY)^{1/3}(UXY)^{1/3}(UVY)^{1/3},$$

where $U$, $V$, $W$, $X$ and $Y$ are positive integers. This expression is identically zero. However, a numeric system uses floating-point arithmetic and will produce a number which can differ greatly from zero. For example, if the consecutive prime numbers 7919, 7927, 7933, 7937, 7949 are chosen, the floating-point result (using 16 digit precision) is 512.0 which is not even close to zero. In fact, one can always choose the five integers to be sufficiently large such that any fixed degree of precision will give a roundoff error. And the magnitude of the error can be made as large as one wishes by choosing sufficiently large integers.

Another class of problems poses greater difficulties to numeric systems. Consider sqrt(-2*log(cos(u^2)))/u^2, and suppose we are interested in values of this expression as $u$ approaches 0. Numeric systems tend to evaluate this expression to 0, whereas the answer, found taking the limit or the Taylor series about the origin, is 1. In fact, with MACSYMA, it is readily found that

$$\text{sqrt(-2*log(cos(u^2)))/u^2} = 1+u^4/12+3*u^8/160+\ldots.$$

In addition to accuracy problems, resulting from keeping a fixed number of significant digits, we shall now see that some familiar mathematical operations actually lose their inherent properties.

## 2.2 ASSOCIATIVITY PROPERTY AND FLOATING-POINT NUMBERS

The familiar mathematical operations plus, "+", and times, "*" satisfy "associativity". This means that if one wants to compute $A+B+C$, it can be done by either $(A+B)+C$ or $A+(B+C)$ and the result obtained is always the same. Similarly, for the multiplication operator, $A*(B*C)=(A*B)*C$. Most of the common mathematical operations we are accustomed to satisfy associativity. However, for floating-point numbers, this all-important property is no longer valid for either addition or multiplication.

Let us retain 6-digit accuracy and consider the computation which adds the numbers 111113.−111111.+5.51111. We can perform it as

$$(111113.-111111.)+5.51111 = 2.00000+5.51111 = 7.51111$$

and obtain one answer. Or we can perform it as

$$111113.+(-111111.+5.51111) = 111113.-111105. = 8.00000$$

to obtain a different answer. Which is correct? They both are correct and this is another difficulty with floating-point numbers.

There are many other examples of this too. Given a matrix whose entries are floating-point numbers suppose one computes the inverse. It is often the case that the matrix product of the matrix and its inverse will not be the identity matrix. Further, some matrices which are almost singular cannot be inverted at all by numerical systems. These difficulties further motivate the use of Computer Algebra systems which provide exact operations with numbers and mathematical expressions.

## 2.3 NUMERIC FEATURES OF COMPUTER ALGEBRA SYSTEMS

Modern Computer Algebra systems perform exact computations with integers, rational numbers and symbolic expressions. In a Computer Algebra system, one can deal with integers of arbitrary size. The integer 200! (two hundred factorial) can be handled just as easily as 20. Fractions as 1/3 and irrational numbers such as SQRT(2) are all kept exact rather than turned into floating-point numbers.

Expressions can be composed of operators, numbers and indeterminates. Built-in procedures are provided for manipulation of polynomials, quotients of polynomials (rational functions), and other expressions that involve trigonometric or hyperbolic functions and many other higher transcendental functions. Many high-level operations are provided as simple commands. For example differentiation, factorising, partial fraction expansion, solution of a system of linear equations, integration, series expansion of functions, matrix operations, etc., are capabilities of MACSYMA which will be discussed in detail later.

Another aspect of Computer Algebra systems is that they often provide not only exact operations, but also the ability to carry out floating-point operations under a user-defined precision. This means that a user first decides how many significant digits should be kept, and then the system carries out all floating-point calculations in accordance with the user-defined precision. Many people feel this is much more convenient than the fixed precisions provided by most numeric systems.

In Computer Algebra systems, arithmetic with numbers can be kept exact. This is done by using software-controlled structures to represent the numbers that are being computed with rather than turning them into floating-point numbers. Consider arithmetic with fractions. The fraction 1/3 can be represented by a list

$$1/3 = (\text{fraction } 1\ 3).$$

The list has three elements. The first element is the name "fraction" which identifies what the list is representing. The second and third elements are the numerator and denominator of the fraction being represented, respectively. Here are a few more examples,

$$1/4 = (\text{fraction } 1\ 4)$$
$$5/2 = (\text{fraction } 5\ 2)$$
$$-7/11 = (\text{fraction } -7\ 11).$$

The last fraction could be represented as (fraction 7 −11), but we do not allow this. It is best to represent equal quantities with a consistent representation. This allows the computer to make faster comparisons and cancellations. Thus the denominator in a fraction representation is restricted to positive integers. One might also believe we should represent both 2/4 and 1/2 as (fraction 1 2) by reducing the fraction to its lowest terms. However, this is not economical because the greatest common divisor between the numerator and the denominator is not always obvious. For example, it is not obvious that (fraction 7178 53835) and (fraction 74 555) both are equivalent to (fraction 2 15). To reduce these fractions to lowest terms, a Computer Algebra system needs to have fast, built-in, programs for the computation of the greatest common divisor between any pairs of integers. It is more efficient to allow unreduced fractions at intermediate steps in a computation.

Once the fractions are represented as above, then simple programs can be written in a Computer Algebra system to add, subtract, multiply and divide fractions by manipulating the parts in the fractional representation in a manner rather similar to what one would do

by hand. In software terminology, such structures for representing quantities or data are referred to as "data structures".

Another aspect of exact computation with numbers is handling large integers. Instead of having a limit as to how large an integer can become, modern Computer Algebra systems allow arbitrary large integers. Thus, it is possible to compute expressions such as

```
200! = 788657867364790503552363213932185062295135977687173263294742533244
       359449963403342920304284011984623904177212138919638830257642790242
       637105061926624952829931113462857270763317237396988943922445621451
       664240254033291864131227428294853277524242407573903240321257405579
       568660226031904170324062351700858796178922222789623703897374720000
       000000000000000000000000000000000000000000000000
```

extremely fast. The size of integers is only limited by the amount of available memory.

### 2.4 SYMBOLIC DIFFERENTIATION

Many new users look at Computer Algebra systems as performing symbolic computation by black magic. Others believe there is nothing more than huge lookup tables for derivatives and integrals. Therefore, as an example of the symbolic computation of expressions, let us look at how a Computer Algebra system might compute derivatives.

Suppose we have an arbitrary elementary function $g(x)$ and we wish to differentiate it with respect to $x$. The following procedure, while overly simplified, is similar to ones used in some Computer Algebra systems, and it can be programmed to compute diff($g(x)$, $x$) using the following rules:

STEP 1: if $g(x)$ = constant, then 0

STEP 2: if $g(x) = x$, then 1

STEP 3: if $g(x) = a(x)+b(x)$, then diff($a(x)$, $x$)+diff($b(x)$, $x$)

STEP 4: if $g(x) = a(x)*b(x)$, then $a(x)$*diff($b(x)$, $x$)+$b(x)$*diff($a(x)$, $x$)

STEP 5: if $g(x) = a(b(x))$, then apply chain rule.

It is remarkable that these simple rules, when applied recursively, form the heart of a differentiation procedure. This procedure obviously needs much more to be effective. If we implement these steps as given, the program will not run very well. For one we must supply a lookup table where the derivatives of the functions are stored. The resulting answer will not be simplified because like terms will not be combined. The software will not know various simplifications such as $\log(\exp(x)) = x$, and it must be programmed to do so. Therefore one has a great deal more to write to achieve a usable differentiation program. Most Computer Algebra systems have these facilities and are very efficient at symbolic differentiation.

## 3. MACSYMA

### 3.1 WHAT IS MACSYMA

Having provided a general introduction to Computer Algebra, we now focus our discussion on the MACSYMA system.

MACSYMA is a large, interactive computer system designed to assist engineers, scientists, and mathematicians in solving mathematical problems. A user supplies symbolic inputs

and MACSYMA yields symbolic (or numeric) results. The development of MACSYMA began at MIT in the late 1960s, and its history has been described elsewhere (Moses, 1974). A few facts are worth stressing. A great deal of effort and expense went into MACSYMA. There are estimates that 100 man-years of development and debugging have gone into the program. While this is a large number, let us consider the even larger number of man-years to use and test MACSYMA. At MIT between 1972 and 1982 we had about 1000 MACSYMA users. If we had 50 serious users using MACSYMA for 50% of their time, 250 casual users at 10% and 700 infrequent users at 2%, then the total is over 600 man-years. MACSYMA has been at 50 sites for 4 years and at 400 sites today. Well, we could conclude that at least 1000 man-years have been spent in using MACSYMA. MACSYMA is now very large and consists of about 3000 lisp subroutines or about 300 000 lines of compiled lisp code joined together in one giant package for performing symbolic mathematics. As a well-used system, MACSYMA has been widely acknowledged as being indispensable for solving problems in science and engineering. The number of papers acknowledging its use numbers more than 400.

### 3.2 WHY MACSYMA IS USEFUL OR NECESSARY

The most important reasons for needing MACSYMA are:

1. The answers one obtains are exact and can often be checked by independent procedures. For example, one can compute an indefinite integral and check the answer by differentiating; the differentiation algorithm is independent of the integration algorithm.

2. The user can generate FORTRAN expressions from MACSYMA expressions. The FORTRAN capability is an extremely important feature combining symbolic and numeric capabilities. The trend is clear, and in a few years we will have powerful, inexpensive desk-top or notebook computers which merge the symbolic, numeric and graphical capabilities in a scientific workstation.

3. The user can explore extremely complex problems which cannot be solved in any other manner. This capability is often imagined to be the major use of Computer Algebra systems. However, one should not lose sight of the fact that MACSYMA is more often used as an advanced calculator to perform everyday symbolic and numeric problems. It also complements conventional tools such as reference tables or numeric processors.

4. A great deal of knowledge has gone into the MACSYMA knowledge base. Therefore the user has access to mathematical techniques which are not available from any other resources, and the user can solve problems even though he may not know or understand the techniques which the system uses to arrive at an answer.

5. A user can test mathematical conjectures easily and painlessly. One frequently encounters mathematical results in the literature and questions their validity. Often MACSYMA can be used to check these using algebraic or numeric techniques or a combination of these. Similarly, one can use the system to show that some problems do not have a solution.

6. MACSYMA is easy to use. Individuals without prior computing experience can learn to solve fairly difficult problems with MACSYMA in a few hours or less. Of course, difficult problems tend to retain their complexity. While MACSYMA is written in a dialect of LISP, the user need never see this base language. MACSYMA itself is a full programming language almost mathematical in nature whose syntax resembles ALGOL.

There are two additional reasons for using MACSYMA which are more important than the others.

7. The first is that one can concentrate on the intellectual content of a problem leaving

computational details to the computer. This often results in accidental discoveries and these occur at a far greater rate than when calculations are done by hand, owing to the power of the program.

8. The most important reason is that, to quote R. W. Hamming, "The purpose of computing is insight, not numbers". However, a second quotation reputed to be by Hamming (with tongue in cheek) is probably correct as well, namely that "The purpose of computing is not yet in sight".

### 3.3 CAPABILITIES OF MACSYMA

It is not possible to list the capabilities of MACSYMA in a few lines since the reference manual (1984) itself occupies more than 500 pages. However, some of the more important capabilities include (in addition to the basic arithmetic operations) facilities to provide analytical answers for:

| | |
|---|---|
| Limits | Taylor series (several variables) |
| Derivatives | Poisson series |
| Indefinite integration | Laplace transformations |
| Definite integration | Indefinite summation |
| Ordinary differential equations | Matrix manipulation |
| Systems of equations (non-linear) | Vector manipulation |
| Simplification | Tensor manipulation |
| Factorisation | FORTRAN generation |

There are other tools for calculations in number theory, combinatorics, continued fractions, set theory and complex arithmetic. There is also a share library currently containing about 80 subroutines. Some of these perform computations such as asymptotic analysis and optimisation, while others deal with many of the higher transcendental functions. In addition one can evaluate expressions numerically at most stages of a computation.

To put the capabilities of MACSYMA in perspective we could say that MACSYMA knows a large percentage of the mathematical techniques used in engineering and the sciences. We do not mean to imply that MACSYMA can do everything. It is easy to come up with examples which MACSYMA cannot handle. Perhaps the following quotation will add the necessary balance. It is an exit message from some MIT computers which often flashes on our screens when logging out. It states: "I am a computer. I am dumber than any human and smarter than any administrator". MACSYMA is remarkable in both the questions it can and cannot answer. It will be many years before it evolves into a system which rivals the human in more than a few areas. But until then, it is the most useful tool that any engineer or scientist can have at his disposal.

### 3.4 USES OF MACSYMA

It is difficult to list the application fields of MACSYMA because users often do not state the tools which helped them perform their research. However, from MACSYMA users' conferences (1977, 1979, 1984) we do know that the program has been used in the following fields:

| | |
|---|---|
| Acoustics | Celestial mechanics |
| Algebraic geometry | Computer-aided design |
| Antenna theory | Control theory |

| | |
|---|---|
| Deformation analysis | Numerical analysis |
| Econometrics | Particle physics |
| Experimental mathematics | Plasma physics |
| Fluid dynamics | Solid-state physics |
| General relativity | Structural mechanics |
| Number theory | Thermodynamics |

And, researchers have used MACSYMA to explore problems in:

| | |
|---|---|
| Airfoil design | Maximum likelihood estimation |
| Atomic scattering cross sections | Nuclear magnetic resonance |
| Ballastic missile defence systems | Optimal control theory |
| Decision analysis in medicine | Polymer modelling |
| Electron microscope design | Propeller design |
| Emulsion chemistry | Resolving closely spaced optical targets |
| Finite element analysis | Robotics |
| Genetic studies of family resemblance | Ship hull design |
| Helicopter blade motion | Spectral analysis |
| Large-scale integrated circuit design | Underwater shock waves |

## 4. Examples of MACSYMA

We shall now present some examples of MACSYMA. In the remainder of this paper MACSYMA output is used and CPU times are often given. In some cases we have modified the output slightly to make it more presentable. The CPU times correspond to a Symbolics 3600 and to the MACSYMA Consortium machine (MIT–MC) which is a Digital Equipment KL10. These machines are about equal in speed and about twice as fast as a Digital Equipment VAX 11/780 for MACSYMA computations.

### 4.1 POLYNOMIAL EQUATIONS

An elementary example of MACSYMA demonstrates the ability to solve equations. In MACSYMA, as with most systems, one has user input lines and computer output lines. Below, in the input line (C1) we have written an expression in an ALGOL-like syntax, terminated it with a semi-colon, and in (D1) the computer echoes the expression by displaying it in a two-dimensional format in a form similar to hand notation.

```
(C1) X^3+B*X^2+A^2*X^2-9*A*X^2+A^2*B*X-2*A*B*X-
                    9*A^3*X+14*A^2*X-2*A^3*B+14*A^4=0;

     3      2    2 2        2    2                  3
(D1) X  + B X  + A  X  - 9 A X  + A  B X - 2 A B X - 9 A  X

                                    2        3        4
                              + 14 A  X - 2 A  B + 14 A  = 0
```

In (C2) we now ask MACSYMA to solve the expression (D1) for X and the three roots appear in a list in (D2).

```
(C2) SOLVE(D1,X);

                                        2
(D2)      [X = 7 A - B,   X = - A ,   X = 2 A]
```

Notice that the roots have been obtained in symbolic form. Quadratic, cubic and quartic equations as well as some higher degree equations can also be solved. Equations can not be solved analytically in closed form when methods are not known, e.g. a general fifth degree (or higher) equations.

## 4.2 DIFFERENTIAL CALCULUS

MACSYMA knows about calculus. The expression (D1) is an often used example in differential calculus.

```
                                  X
                                 X
(D1)                            X
```

MACSYMA is now asked to differentiate (D1) with respect to X to obtain this classic textbook result of differentiation. Notice the speed, 3/100 CPU seconds to compute this derivative.

```
(C2) DIFF(D1,X);
Time= 30 msec.

                    X
                   X    X                        X - 1
(D2)              X   (X  LOG(X) (LOG(X) + 1) + X     )
```

Below is a more complicated function, the error function of the tangent of the arc-cosine of the natural logarithm of X. Notice that MACSYMA does not display the identical input. This is because the input in (C1) passes through the MACSYMA simplifier. MACSYMA recognises that the tangent of the arc-cosine of a function satisfies a trigonometric identity, namely TAN(ACOS(X)) = SQRT(1-X^2)/X. It takes this into account before computing and displaying (D1).

```
(C1) ERF(TAN(ACOS(LOG(X))));
                                        2
                                 SQRT(1 - LOG (X))
(D1)                      ERF(-----------------)
                                    LOG(X)
```

Now when MACSYMA is asked to differentiate (D1) with respect to X, it does so in a straightforward manner and simplifies the result using the canonical rational simplifier, RATSIMP. This command puts the expression in a numerator-over-denominator form cancelling any common divisors. In (D2) the symbols %E and %PI are MACSYMA's representations for e (the base of the natural logarithms) and pi.

```
(C2) DIFF(D1,X),RATSIMP;
Time= 1585 msec.
                                        1
                                  1 - -------
                                        2
                                     LOG (X)
                              2 %E
(D2)                  - ------------------------------------
                                 2                   2
                        SQRT(%PI) X LOG (X) SQRT(1 - LOG (X))
```

## 4.3 TRIGONOMETRY

MACSYMA knows about basic trigonometry. For example, it can manipulate expressions containing multiple angles of trigonometric functions. In (D1) below is an expression

which has no obvious value. First, the TRIGEXPAND command is used, which expands out trigonometric (hyperbolic) functions of sums of angles and of multiple angles. Then, (D2) is EXPANDed and we find it vanishes identically.

This is an example of using MACSYMA to do "experimental mathematics". If one suspected that (D1) vanished, trying to verify this by hand is time consuming and error prone with the multitude of signs. It would be interesting if someone could find a geometrical interpretation of this identity which does not appear to be known.

```
(D1) SIN(Y - X) SIN(Z - X) SIN(Z - Y) + COS(Y - X) COS(Z - X) SIN(Z - Y)

   - COS(Y - X) SIN(Z - X) COS(Z - Y) + SIN(Y - X) COS(Z - X) COS(Z - Y)

(C2) TRIGEXPAND(%);
Time= 1400.0 msec.

(D2) (COS(X) SIN(Y) - SIN(X) COS(Y)) (SIN(X) SIN(Z) + COS(X) COS(Z))

     (SIN(Y) SIN(Z) + COS(Y) COS(Z)) - (SIN(X) SIN(Y) + COS(X) COS(Y))

     (COS(X) SIN(Z) - SIN(X) COS(Z)) (SIN(Y) SIN(Z) + COS(Y) COS(Z))

     + (SIN(X) SIN(Y) + COS(X) COS(Y)) (SIN(X) SIN(Z) + COS(X) COS(Z))

     (COS(Y) SIN(Z) - SIN(Y) COS(Z)) + (COS(X) SIN(Y) - SIN(X) COS(Y))

     (COS(X) SIN(Z) - SIN(X) COS(Z)) (COS(Y) SIN(Z) - SIN(Y) COS(Z))

(C3) EXPAND(%);
Time= 1350.0 msec.

(D4)                                    0
```

MACSYMA can also go the other way, so to speak, beginning with an expression such as (D5) and expressing it in terms of multiple angles using the TRIGREDUCE command.

```
                 5          2       3          4
(D5)     2 SIN (X) - 8 COS (X) SIN (X) + 6 COS (X) SIN(X)

(C6) EXPAND(TRIGREDUCE(D5));

(D6)                        SIN(5 X) + SIN(X)
```

## 4.4 FACTORISATION

MACSYMA can factor expressions. Below is a multivariate polynomial in four variables.

```
          2 7 4 8      2 6 3 8      3 7 4 6
(D1) - 36 W  X  Y  Z  + 3 W  X  Y  Z  - 24 W  X  Y  Z

     3 6 3 6      2 8 6 5       4 7 6 5
 + 2 W  X  Y  Z  + 96 W  X  Y  Z  - 168 W  X  Y  Z

      2 7 6 5       2 10 5 5      2 7 5 5      7 5 5
 + 12 W  X  Y  Z  - 216 W  X  Y  Z  - 8 W  X  Y  Z  + 9 X  Y  Z

      4 6 5 5    2 6 5 5       2 9 4 5       7 3 5
 + 14 W  X  Y  Z  - W  X  Y  Z  + 18 W  X  Y  Z  + 87 X  Y  Z

     2 6 3 5       7 5 3        7 3 3      3 6 3 3
 - 3 W  X  Y  Z  + 6 W X  Y  Z  + 58 W X  Y  Z  - 2 W  X  Y  Z
```

```
      8  7  2       2  7  7  2      7  7  2       10  6  2
- 24 X  Y  Z  + 42 W  X  Y  Z  - 3 X  Y  Z  + 54 X   Y  Z

      8  5  2       2  7  5  2      7  5  2      4  6  5  2
- 232 X  Y  Z  + 414 W  X  Y  Z  - 29 X  Y  Z  - 14 W  X  Y  Z

   2  6  5  2       10  4  2      2  9  4  2
+ W  X  Y  Z  + 522 X   Y  Z  - 18 W  X  Y  Z
```

The function FACTOR is now called on (D1), and

```
(C2) FACTOR(D1);
Time= 111998 msec.

       6  3  2    3              2      2  2    2      3
(D2) - X  Y  Z  (3 Z  + 2 W Z - 8 X Y  + 14 W  Y  - Y  + 18 X  Y)

                                 2     3    2  3       2          2
                           (12 W  X Y Z  - W  Z  - 3 X Y  - 29 X + W )
```

MACSYMA factors this massive expression in about 2 CPU minutes. One can also extend the domain of factorisation to the Gaussian integers or other algebraic fields (Wang, 1978).

The time required to factor expressions depends on the number of variables and the degree of the exponents. It is not difficult to construct examples which look simple but take inordinate amounts of time to factor. For example, X^2026−X^1013+1 factors into two parts. One factor is X^2−X+1, but the second is an irreducible polynomial with 1351 terms. The current factorisation algorithms do not return after several CPU hours on this example. It is possible to enhance the factorisation program to run faster on this case, but one could always invent other "simple looking" examples which would push CPU times to excess.

Factoring expressions by hand is next to impossible for all but the most trivial polynomials. A Computer Algebra system can factor expressions with amazing speed. MACSYMA uses sophisticated algorithms which were developed over the past 10 years (Wang, 1976). The basic scheme is to transform the complicated problem to a simpler case, find its factors, and then build up the desired factors of the complicated case. For example, to factor the expression

$$F(X, Y) := X^4 + 10YX^3 + 35Y^2X^2 + 50Y^3X + 24Y^4$$

the univariate polynomial F(X, 1) is factored first. It turns out that

$$F(X, 1) = (X+1)(X+2)(X+3)(X+4).$$

Having obtained the univariate factors of F(X, 1), the actual factors of F(X, Y) are found with the algorithms. In this case $Y = 1$ suffices and

$$F(X, Y) = (X+Y)(X+2Y)(X+3Y)(X+4Y).$$

The factorisation of univariate polynomials over the integers uses methods of factoring expressions over finite fields (Berlekamp, 1967).

### 4.5 SIMPLIFICATION

A very important feature of MACSYMA is the ability to simplify expressions. When one of the authors studied the gravitational radiation for a new gravitation theory (Pavelle, 1978; Mansouri & Chang, 1976), a particular calculation produced an expression with

several hundred thousand terms. From geometrical arguments it was known the expression must simplify and indeed, using MACSYMA, the expression collapsed to a small number of pages of output. The following expression occurred repeatedly in the course of the calculation and caused the collapse of the larger expression during simplification.

$$(D1)\quad \frac{(\mathrm{SQRT}(R^2+A^2)+A)\,(\mathrm{SQRT}(R^2+B^2)+B)}{\dfrac{R^2}{\mathrm{SQRT}(R^2+B^2)+\mathrm{SQRT}(R^2+A^2)+B+A}-\dfrac{}{\mathrm{SQRT}(R^2+B^2)+\mathrm{SQRT}(R^2+A^2)-B-A}}$$

```
(C2) RATSIMP(D1);
Time= 138 msec.

(D2)                                0
```

When the canonical simplifier RATSIMP is called on (D1) above it vanishes identically. It is not easy to prove this by hand. Combining the expressions over a common denominator results in a numerator which contains 20 terms when fully expanded, and one must be very careful to assure cancellation.

### 4.6 EVALUATING NUMERIC EXPRESSIONS AND SIMPLIFICATION OF RADICALS

MACSYMA is also useful in proving that numeric expressions vanish. Below is a nested radical and evaluating it numerically in (D2) to 8 decimal places, with the NUMER command, shows only that it is small. It was known that it had to vanish but the numeric evaluation routines did not prove this. How does one prove it is zero?

```
(D1) 173 SQRT(34) SQRT(2 SQRT(34) + 35) +

                1394 SQRT(2 SQRT(34) + 35) - 1567 SQRT(34) - 7276

(C2) D1,NUMER;

(D2)                - 0.00048828125
```

It is difficult to deal with nested expressions by hand or by computer, but MACSYMA has facilities for dealing with some problems of this kind. We first call the function SQRTDENEST on (D1) which simplifies the radicals by denesting them. Comparing (D1) with (D3) we see that SQRTDENEST tells us that SQRT(2*SQRT(34)+35) = SQRT(34)+1. This is far from obvious by inspection, although it is easy to verify by hand. Now, when we expand (D3), MACSYMA returns zero.

```
(C3) SQRTDENEST(D1);

(D3) 173 SQRT(34) (SQRT(34) + 1) + 1394 (SQRT(34) + 1)

                - 1567 SQRT(34) - 7276

(C4) EXPAND(D3);

(D4)                        0
```

## 4.7 INDEFINITE INTEGRATION

One of the most useful features in MACSYMA is the ability to perform integration. This is useful, of course, in and of itself. But it is also necessary because some of the commonly used integral tables have error rates as high as 25% (Risch, 1969, 1970; Klerer & Grossman, 1971; Rosenlicht, 1972). One can wonder how many disasters have occurred because engineers have used faulty tables or made errors in hand calculations.

Below, for example, is an error which was found in one of the most popular tables (Gradshteyn & Ryzhik, 1965).

$$\int \frac{1}{X^2\,(C\,X+B\,X)^{3/2}}\,dX = \frac{2\,(-8\,C^2\,X^2+4\,B\,C\,X-B^2)}{3\,B^3\,X^{3/2}\,\mathrm{SQRT}(C\,X+B)}$$

The correct integral, found by MACSYMA, is

$$\int \frac{1}{X^2\,(C\,X+B\,X)^{3/2}}\,dX = \frac{2\,(8\,C^2\,X^2+4\,B\,C\,X-B^2)}{3\,B^3\,X^{3/2}\,\mathrm{SQRT}(C\,X+B)}$$

It is interesting that a large percentage of these errors are actual blunders rather than misprints.

MACSYMA can handle integrals involving rational functions and combinations of rational, algebraic functions, and the elementary transcendental functions. It also has knowledge about error functions and some of the higher transcendental functions.

Below is an integral which is quite difficult to do by hand. It is not found in standard tables in its given form although it may transform to a recognised case. It is especially difficult to do by hand unless one notices a trick which involves performing a partial fraction decomposition of the integrand with respect to the LOG(X). However, MACSYMA handles it readily.

```
                          /
                          [  LOG(X) - 1
(D1)                      I  ------------ dX
                          ]    2      2
                          /  LOG (X) - X

(C2) INTEGRATE(D1,X);
Time= 744 msec.

                    LOG(LOG(X) + X)   LOG(LOG(X) - X)
(D2)                ---------------   ---------------
                          2                 2
```

Below is another integral which is not found in standard tables and which MACSYMA could not handle until recently owing to a bug. We needed the answer and could not wait for the bug to be fixed. We will use this opportunity to illustrate a point. MACSYMA could not integrate (D1) directly and returned the integral in noun form with the integral sign around it in (D1). This normally means that MACSYMA cannot perform the computation. We assumed this was because the system had a difficulty with error functions. Therefore, we got rid of the error function by differentiating (D1) with respect to B resulting in (D2). This is often the same procedure one tries by hand when confronted by difficult integrals.

```
(C1) INTEGRATE(X*EXP(-Q^2*X^2)*ERF(B*X-A),X);

                    /      2 2
                    [   - Q  X
(D1)                I X %E        ERF(B X - A) dX
                    ]
                    /


(C2) DIFF(D1,B);

                    /                  2     2 2
                    [  2   - (B X - A)  - Q  X
                  2 I X  %E                          dX
                    ]
                    /
(D2)              -------------------------------
                               SQRT(%PI)
```

In this form MACSYMA did perform the integration with respect to X. We then asked MACSYMA to integrate (D2) with respect to X and then with respect to B to obtain the result we were seeking, namely

```
      /      2 2
      [   - Q  X
(D3)  I X %E        ERF(B X - A) dX =
      ]
      /
            2 2
           A  Q
       - -------
            2    2       2   2
           Q  + B      (Q + B ) X - A B
      B %E          ERF(-----------------)      2 2
                                2   2        - Q  X
                          SQRT(Q + B )     %E        ERF(B X - A)
      ------------------------------------ - ----------------------
                  2        2   2                       2
               2 Q  SQRT(Q  + B )                   2 Q
```

The point we wish to make is that MACSYMA sometimes believes it cannot solve a problem, and the user should not always take the computer program at its word!

### 4.8 DEFINITE INTEGRATION

The Risch algorithm is a powerful decision procedure for indefinite integration. No such algorithm is known for definite integration. A computer program for definite integration must implement a collection of methods applicable in different situations for many different types of integrands. The problem is more difficult than indefinite integration. One often has the added difficulty of taking limits at the endpoints of the integral.

MACSYMA has some impressive capabilities for computing definite integrals. Here is an example of a function whose definite integral does not appear to be tabulated:

```
                                    2
                           2   - U X
(D1)                       X  %E        LOG(X)
```

```
(C2) INTEGRATE(D1,X,0,INF),FACTOR;
Time= 138442 msec.

                SQRT(%PI) (LOG(U) + 2 LOG(2) + %GAMMA - 2)
(D2)        - ------------------------------------------
                                3/2
                             8 U
```

In (C2) above we have asked MACSYMA to integrate (D1) with respect to X from 0 to infinity. MACSYMA uses %GAMMA for the Euler–Mascheroni constant = 0.577215664 . . . .

We can now generate new integrals by differentiating (D2). From (D1) and (D2) we have

```
       INF
      /                2
      [     2   - U X
(D3)  I    X  %E       LOG(X) dX =
      ]
      /
       0

                    SQRT(%PI) (LOG(U) + 2 LOG(2) + %GAMMA - 2)
                  - ------------------------------------------
                                    3/2
                                 8 U
```

and let us differentiate (D3) four times with respect to U and factor to obtain

```
(C4) DIFF(D3,U,4),FACTOR;
       INF
      /                2
      [     10  - U X
(D4)  I    X   %E       LOG(X) dX =
      ]
      /
       0

                3 SQRT(%PI) (315 LOG(U) + 630 LOG(2) + 315 %GAMMA - 1126)
              - ---------------------------------------------------------
                                      11/2
                                  128 U
```

In addition to definite integration, MACSYMA can perform numeric integration using the Romberg numeric integration procedure. There are a number of other numeric techniques available. For example, one has the capability to evaluate expressions numerically to arbitrary precision.

## 4.9 LAPLACE TRANSFORMATIONS

There are a large number of special purpose routines built into MACSYMA. For example, there is a package for solving some classes of simultaneous differential equations using Laplace transformations. Below we express our equations in (D1) and (D2) where the subscripts denote differentiation.

```
(C1) EQ1;

                    F(X)    - 6 G(X)  = 6 SIN(X)
                       X X         X
(C2) EQ2;
                                  2
(D2)                6 G(X)    + A  F(X)  = 6 COS(X)
                       X X          X
```

Next we specify initial values for F(X) and G(X) and their first derivatives while inhibiting the D-lines by terminating the commands with a $.

```
(C3) ATVALUE(F(X),X = 0,0)$

(C4) ATVALUE(G(X),X = 0,1)$

(C5) ATVALUE('DIFF(F(X),X),X = 0,0)$

(C6) ATVALUE('DIFF(G(X),X),X = 0,1)$
```

Finally we use the DESOLVE command which uses Laplace transforms to find the particular integral for the system of equations.

```
(C7) DESOLVE([EQ1,EQ2],[F(X),G(X)]);

                    12 SIN(A X)   6 COS(A X)   12 SIN(X)   6
(D7)  [F(X) = - ----------- - ---------- + --------- + --,
                        2            2          2         2
                  A (A  - 1)        A         A  - 1     A

                                                  2
                      SIN(A X)   2 COS(A X)   (A  + 1) COS(X)
            G(X) = - -------- - ---------- + ---------------]
                        A           2             2
                                   A  - 1        A  - 1
```

## 4.10 FUNCTIONS OF A COMPLEX VARIABLE

MACSYMA has some capabilities of dealing with functions of a complex variable. For example, it is quite good at computing residues:

```
                                    2 Z
                                1 - %E
(D1)                            ------
                                  4
                                 Z  SIN(Z)

(C2) RESIDUE(D1,Z,0);

(D2)                              - 1
```

Above it found the residue of (D1) at $Z = 0$. Below MACSYMA finds the real and imaginary parts of (D3).

```
                                  %I Z
                               A %E     + B
(D3)                           ------------
                                  %I Z
                               C %E     + D

(C4) REALPART(D3),FACTOR;

                    (A D  + B C) COS(Z) + B D + A C
(D4)           ---------------------------------
                                      2    2
                        2 C D COS(Z) + D  + C
```

```
(C5) IMAGPART(D3),FACTOR;

                                (A D - B C) SIN(Z)
(D5)                        ----------------------
                                          2    2
                            2 C D COS(Z) + D  + C
```

This example is quite difficult to do by hand.

## 4.11 COMBINATORIAL FUNCTIONS

MACSYMA has the capability of manipulating and simplifying combinatorial expressions. For example, in (D1), we have an expression involving binomial coefficients where BINOMIAL(N, M): = N!/(M!*(N-M)!).

```
(C1) BINOMIAL(N,M)*BINOMIAL(N+3,M+2)/BINOMIAL(N+3,M+4);

                        BINOMIAL(N, M) BINOMIAL(N + 3, M + 2)
(D1)                    -------------------------------------
                              BINOMIAL(N + 3, M + 4)
```

We now call the function MAKEFACT on (D1) which converts binomial coefficients to factorials.

```
(C2) MAKEFACT(D1);

                              (M + 4)! N! (N - M - 1)!
(D2)                        --------------------------------
                            M! (M + 2)! (N - M)! (N - M + 1)!
```

Calling MAKEGAMMA on (D2) converts this expression of factorials to Gamma functions.

```
(C3) MAKEGAMMA(D2);

                        GAMMA(M + 5) GAMMA(N + 1) GAMMA(N - M)
(D3) ------------------------------------------------------------------
     GAMMA(M + 1) GAMMA(M + 3) GAMMA(N - M + 1) GAMMA(N - M + 2)
```

We can also simplify expressions involving factorials. Using the MINFACTORIAL function on (D2) results in

```
(C4) MINFACTORIAL(D2);

                                  (M + 3) (M + 4) N!
(D4)                        ---------------------------------
                                        2
                            M! (N - M)  (N - M + 1) (N - M - 1)!
```

## 4.12 TAYLOR/LAURENT SERIES

The Taylor (Laurent) series capability is very impressive. Below we ask for the first 15 terms of the series of (D1) about the point X = 0. Notice that the resulting expression is obtained in less than 1/2 CPU second.

```
                                                  2
                                3    B LOG(X  - X + 1)
(D1)                      A SIN(X ) + ----------------
                                              5
                                             X
```

```
(C2) TAYLOR(D1,X,0,15);
Time= 365 msec.

                                                          2             3
        B     B     2 B     B    B   B X   B X    (B + 8 A) X
(D2)/T/ - -- + ---- + ---- + --- - - - --- - ---- + ------------
         4     3      2    4 X  5   3     7           8
        X    2 X    3 X

        4      5      6      7      8                9
  (2 B) X    B X    B X    B X    B X    (3 B - 7 A) X
+ -------- + ---- - ---- - ---- - ---- + --------------
     9        10     11     6      13          42

        10       11       12       13       14              15
  (2 B) X     B X      B X      B X      B X     (6 B + A) X
+ --------- + ----- - ----- - ----- - ----- + ------------- + . . .
     15        16       17       9        19          120
```

The program can also compute Taylor (Laurent) series in several variables.

## 4.13 SIMULTANEOUS LINEAR EQUATIONS

Five linear equations M1 through M5 are given as shown.

```
(C1) M1;
(D1)          3 A + 5 B + 7 C + 11 D + 13 E = 17 R

(C2) M2;
(D2)         19 A + 23 B + 29 C + 31 D + 37 E = 41 S

(C3) M3;
(D3)         43 A + 47 B + 53 C + 59 D + 61 E = 67 T

(C4) M4;
(D4)         71 A + 73 B + 79 C + 83 D + 89 E = 97 X

(C5) M5;
(D5)       101 A + 103 B + 107 C + 109 D + 113 E = 127 Y
```

The SOLVE command is used to obtain the solution in exact form. This command is also an example of many MACSYMA commands where the form is very close to common mathematical notation.

```
(C6) SOLVE([M1,M2,M3,M4,M5],[A,B,C,D,E]);
Time= 1355 msec.

                                10922 Y - 14259 X + 2706 S + 1207 R
(D6) [  A = - ------------------------------------
                                      284

                                64008 Y - 64505 X - 14271 T + 7667 S + 10863 R
             B = ---------------------------------------------
                                      852

                                8128 Y - 4365 X - 4757 T - 1845 S + 3621 R
             C = - ---------------------------------------
                                      284
```

```
      3048 Y - 776 X - 4757 T + 2583 S
D = - --------------------------------
                    284

    3810 Y + 5917 X - 14271 T + 2501 S + 3621 R
E = ------------------------------------------- ]
                      852
```

MACSYMA can also deal with systems of non-linear algebraic equations.

## 4.14 MATRIX ANALYSIS

MACSYMA has powerful capabilities for manipulating matrices and performs most of the common operations. Here, the steps leading to the eigenvalues of a matrix are presented. First, the matrix is entered as shown in (C1).

```
(C1) MATRIX([0,6,-10,8],[6,0,8,10],[-10,8,15*A,6],[8,10,6,15*A]);

                    [  0    6   - 10  - 8  ]
                    [                      ]
                    [  6    0    8     10  ]
(D1)                [                      ]
                    [ - 10  8   15 A   6   ]
                    [                      ]
                    [ - 8   10   6    15 A ]
```

MACSYMA is then asked to compute the characteristic polynomial of (D1) for the parameter "L" and then solve resulting the polynomial for L. The four eigenvalues are shown below.

```
(C2) SOLVE(CHARPOLY(D1,L),L);
                                        2
                  15 A - 3 SQRT(5) SQRT(5 A  + 8 A + 32)
[L = ---------------------------------------------------,
                                2

                                    2
                  3 SQRT(5) SQRT(5 A  + 8 A + 32) + 15 A
L = ---------------------------------------------------,
                                2

                                       2
                  15 A - SQRT(5) SQRT(45 A  - 72 A + 32)
L = ---------------------------------------------------,
                                2

                                    2
                  SQRT(5) SQRT(45 A  - 72 A + 32) + 15 A
L = ---------------------------------------------------]
                                2
```

## 4.15 ORDINARY DIFFERENTIAL EQUATIONS

Another powerful feature is the ordinary differential equation solver, ODE.† In (C1), we declare that Y is a function of X. This assures that the derivative (2nd) of Y with respect to X will not vanish when (C2) is evaluated.

† ODE is a collection of algorithms for solving ordinary differential equations on MACSYMA. It was built over several years by E. L. Lafferty, J. P. Golden, R. A. Bogen and B. Kuipers.

```
(C1) DEPENDS(Y,X)$

(C2) (1+X^2)*DIFF(Y,X,2)-2*Y=0;

                          2
(D2)                   (X  + 1) Y    - 2 Y = 0
                                X X
```

Now the system is asked to solve (D2) for Y as a function of X using the ODE command. The general solution with the two integration constants, %K1 and %K2 is given in (D3) in 2 CPU seconds. The program can also find power series solutions for some differential equations when it can solve the recurrence relation. It does this in (D4). MACSYMA can be used to check the answer (D3). In (C5), the system is told to substitute (D3) into (D2), differentiate the result, and simplify.

```
(C3) ODE(D2,Y,X);
Time= 2068 msec.

                    2       ATAN(X)      X              2
(D3)      Y = %K2 (X  + 1) (------- + --------) + %K1 (X  + 1)
                                2          2
                                       2 X  + 2

(C4) ODE(D2,Y,X,SERIES);
Time= 8766 msec.

                                       INF
                                       ====          I  2 I
                          2            \       (- 1)  X
(D4)          Y = %K1 (X  + 1) - %K2 X  >     ---------------
                                       /          1       1
                                       ====  (I - -) (I + -)
                                       I = 0      2       2

(C5) D2,D3,DIFF,RATSIMP;
Time= 2051 msec.

(D5)                          0 = 0
```

The ODE package in MACSYMA typifies a technique in Computer Algebra which is to have the computer look for patterns in a given problem and when the pattern matches generate the answer by a procedure (algorithm). Many good mathematicians do, in fact, use this technique. However, the computer is often better due to its thoroughness and the fact that it does not make the normal human errors.

ODE now has capabilities for solving a large class of differential equations analytically. While it is difficult to quantify capability of this kind, one can state that it is as capable as a particular set of books, but not as capable as the authors of those books. For example, it has all the elementary methods in the textbook of Rainville (1964) and the tables of Kamke (1943). Perhaps a better way to measure its abilities is to say that virtually all the known methods for the solution of first and second order ordinary differential equations are embodied in the program.

### 4.16 FORTRAN GENERATION

Here is an example of the FORTRAN capability in MACSYMA. In (C1) we ask MACSYMA to convert the expanded form of (X+3*Y+7*Z)^8 into legal fortran with 6 spaces at the beginning of the line, continuation lines and ** for exponentiation.

```
(C1) FORTRAN(EXPAND((X+3*Y+7*Z)^8));

      5764801*Z**8+19765032*Y*Z**7+6588344*X*Z**7+29647548*Y**2*Z**
     1   6+19765032*X*Y*Z**6+3294172*X**2*Z**6+25412184*Y**3*Z**5+2
     2   5412184*X*Y**2*Z**5+8470728*X**2*Y*Z**5+941192*X**3*Z**5+1
     3   3613670*Y**4*Z**4+18151560*X*Y**3*Z**4+9075780*X**2*Y**2*Z
     4   **4+2016840*X**3*Y*Z**4+168070*X**4*Z**4+4667544*Y**5*Z**3
     5   +7779240*X*Y**4*Z**3+5186160*X**2*Y**3*Z**3+1728720*X**3*Y
     6   **2*Z**3+288120*X**4*Y*Z**3+19208*X**5*Z**3+1000188*Y**6*Z
     7   **2+2000376*X*Y**5*Z**2+1666980*X**2*Y**4*Z**2+740880*X**3
     8   *Y**3*Z**2+185220*X**4*Y**2*Z**2+24696*X**5*Y*Z**2+1372*X*
     9   *6*Z**2+122472*Y**7*Z+285768*X*Y**6*Z+285768*X**2*Y**5*Z+1
     :   58760*X**3*Y**4*Z+52920*X**4*Y**3*Z+10584*X**5*Y**2*Z+1176
     ;   *X**6*Y*Z+56*X**7*Z+6561*Y**8+17496*X*Y**7+20412*X**2*Y**6
     <   +13608*X**3*Y**5+5670*X**4*Y**4+1512*X**5*Y**3+252*X**6*Y*
     =   *2+24*X**7*Y+X**8

(D1)                                DONE
```

Some sites have actually generated thousands of lines of FORTRAN in this way, written the output to a file and taken it over to their large numeric systems for processing. More powerful FORTRAN functions in MACSYMA will create optimized FORTRAN and allow one to generate code specific to particular computer systems.

### 4.17 INDEFINITE SUMMATION

Often, algorithms themselves are discovered by using Computer Algebra systems with the procedures of experimental mathematics. This means that one gains empirical experience with a heuristic method that works often enough to let one see what answers to a problem look like. An example of this technique is the algorithm for indefinite (finite) summation found by Gosper (1978) with MACSYMA. The restrictions for the procedure are that both the summand and the answer must be expressible as products of powers, factorials, binomials, and/or rational functions. Even with these restrictions, it is possible to handle a large class of the indefinite sums one encounters. The history of the more general problem of indefinite summation contains the names of Newton, Euler, Bernoulli and Boole and despite the many years of work of these distinguished people, the results are surprisingly sparse. One reason for believing in the existence of an algorithm is the success achieved in solving the indefinite integral problem and the close parallels between the two conceptually. Gosper's approach involved testing many cases with MACSYMA until a pattern emerged which worked for all cases and led to the discovery of the algorithm. Recently, using abstract algebra techniques similar to those Risch used for integration, Karr (1981) has extended Gosper's work. Some parts of Gosper's algorithm have been implemented in MACSYMA and here is an example of its power:

```
                                M
                               ====
                               \      N  4
(D1)                            >    A  N
                               /
                               ====
                               N = 0
```

In (D1) is the indefinite sum we wish to evaluate. The command, NUSUM, which performs this computation, is given in (C2).

```
(C2) NUSUM(A^N*N^4,N,0,M);
Time= 24801 msec.

      M + 1  4  4      3  4      2  4        4    4      3  3
(D2) A     (A  M  - 4 A  M  + 6 A  M  - 4 A M  + M  - 4 A  M

      2  3         3      3      3  2      2  2         2      2
 + 12 A  M  - 12 A M  + 4 M  + 6 A  M  - 6 A  M  - 6 A M  + 6 M

     3         2                   3
 - 4 A  M - 12 A  M + 12 A M + 4 M + A
                                          2
     2                    5    A (A + 1) (A  + 10 A + 1)
 + 11 A  + 11 A + 1)/(A - 1)  - ------------------------
                                            5
```

It takes about 25 CPU seconds to find the result, and we can check the answer by taking the first backward difference of (D2) using the UNSUM command. This is analogous to checking an integral by differentiating.

```
(C3) UNSUM(%,M);
Time= 6581 msec.

                                  M  4
(D3)                             A  M
```

## 4.18 MINIMISING A FUNCTION USING LAGRANGE MULTIPLIERS

Here we combine a number of MACSYMA commands in an elementary interactive session to find the extremum of a function with constraints using the Lagrange multiplier technique.

In (C1) we assign to F the expression shown in (D1). We shall find the extremum of F subject to two constraints with respective assignments K1 and K2 shown in (D2) and (D3). In this example [X, Y, Z, T] are variables and [A, B] are constants.

```
(C1) F:X^2+2*Y^2+Z^2+T^2;

                           2     2    2    2
(D1)                      Z  + 2 Y  + X  + T

(C2) K1:A+X+3*Y-Z+T;
(D2)                - Z + 3 Y + X + T + A

(C3) K2:B+2*X-Y+Z+2*T;
(D3)                Z - Y + 2 X + 2 T + B
```

A new function G is defined containing the Lagrange multipliers L1 and L2.

```
(C4) G:F+L1*K1+L2*K2;

          2     2    2    2
(D4) G = Z  + 2 Y  + X  + T

     + L2 (Z - Y + 2 X + 2 T + B) + L1 (- Z + 3 Y + X + T + A)
```

The next step is to form all partial derivatives of G as shown in (C5).

```
(C5) [DIFF(G,X),DIFF(G,Y),DIFF(G,Z),DIFF(G,T),DIFF(G,L1),DIFF(G,L2)];
```

```
(D5) [2 X + 2 L2 + L1, 4 Y - L2 + 3 L1, 2 Z + L2 - L1,

    2 T + 2 L2 + L1, - Z + 3 Y + X + T + A, Z - Y + 2 X + 2 T + B]
```

We then solve the system (D5) for each of the six parameters.

```
(C6) SOLVE(D5,[X,Y,Z,T,L1,L2]);

                27 B + 13 A       5 A - 2 B      11 A - 9 B
(D6)  [[X = - -----------, Y = - ---------, Z = ----------,
                   138              23             69

              27 B + 13 A      19 A - 3 B       A - 5 B
       T = - -----------, L1 = ----------, L2 = - -------]]
                   138              69             23
```

For the final step we substitute (D6) into F as defined in (D1). The expression (D7) is the extremum of (D1) subject to the constraints.

```
(C7) D1,D6,RATSIMP;
                                2              2
                           15 B  - 6 A B + 19 A
(D7)                       ---------------------
                                   138
```

## 5. Extending the Capabilities of MACSYMA

### 5.1 BLOCK PROGRAMS AND HESSIANS

Since MACSYMA provides a user-level programming language as well as allowing full interactive use, one can easily extend the capabilities of MACSYMA by writing one's own programs. For example, MACSYMA does not have a built-in function for computing Hessians which are determinants of symmetric matrices whose elements are the second partial derivatives of a function (Goursat, 1904).

A MACSYMA program can be written for this purpose. A block in MACSYMA is similar to the LISP "prog" construct, a subroutine in FORTRAN, or a procedure in ALGOL or PL/1. A block contains local variables which will not conflict with variables having the same names outside or global to the block. For example, in (C1) the atomic variables DFXX ... DFZZ are locally assigned the values of the various mixed second partial derivatives. We then compute the determinant and simplify it through the RATSIMP and FACTOR commands within the block statement. In fact, any MACSYMA command can be included within the block.

```
(C1) HESSIAN(F,X,Y,Z):=BLOCK([DFXX,DFXY,DFXZ,DFYY,DFYZ,DFZZ],
       DFXX:DIFF(F,X,2),
          DFXY:DIFF(F,X,1,Y,1),
             DFXZ:DIFF(F,X,1,Z,1),
                DFYY:DIFF(F,Y,2),
                   DFYZ:DIFF(F,Y,1,Z,1),
                      DFZZ:DIFF(F,Z,2),
                         FACTOR(RATSIMP(DETERMINANT(MATRIX
    ([DFXX,DFXY,DFXZ],[DFXY,DFYY,DFYZ],[DFXZ,DFYZ,DFZZ])))))$
```

This functional definition is only appropriate for the three variable cases, but it can easily be extended for any number of dimensions. Below, in (D2), we choose a function of three variables, and the routine computes the Hessian in (D3).

```
(C2) COS(X/(Y+Z))*EXP(X-2*Z);

                          X - 2 Z        X
(D2)                     %E       COS(-----)
                                      Z + Y

(C3) HESSIAN(D2,X,Y,Z);
Time= 19127 msec.

                     3 X - 6 Z       X       2   X
                4 %E           COS(-----) SIN (-----)
                                     Z + Y       Z + Y
(D3)          - ------------------------------------
                                    4
                             (Z + Y)
```

### 5.2 RECURSIVE DEFINITION FOR CHEBYSHEV POLYNOMIALS

While adding features to the knowledge base is fairly easy, careful programming is essential. Suppose MACSYMA did not know about Chebyshev polynomials. A recursive definition of these is given in (C1) and (C2).

```
(C1) T[N]:=2*X*T[N-1]-T[N-2];

(D1)          T  := 2 X T      - T
               N         N - 1    N - 2

(C2) (T[0]:1, T[1]:X)$
```

To compute the 20th polynomial, one simply calls T[20] as below. Of course, it must be simplified, and we expand it with the command in (C4).

```
(C3) T[20]$
Time= 2880.0 msecs.

(C4) RATEXPAND(D3);
  Time= 27083 msec.

            20            18            16            14
(D4) 524288 X   - 2621440 X   + 5570560 X   - 6553600 X

            12            10           8          6         4
 + 4659200 X   - 2050048 X   + 549120 X  - 84480 X  + 6600 X

       2
 - 200 X  + 1
```

This method is slow because we were not clever in our approach. We generated a huge expression in (D3) (10946 terms), as seen in (D5), which we later expanded to simplify it.

```
(C5) NTERMS(D3);

(D5)                      10946
```

We now repeat the same problem, adding a RATEXPAND command to the recursive definition. We change the name of the array to S because we have already used the array name T. If we had redefined T, but had not removed array elements already computed, then the old values would have been used. Alternately we could remove the definition of T from memory.

```
(C6) S[N]:=RATEXPAND(2*X*S[N-1])-S[N-2];
```

```
(D6)          S  := 2 X S      - S
                N          N - 1    N - 2

(C7) (S[0]:1, S[1]:X)$

(C8) S[20];
Time= 1700.0 msecs.

                20             18             16             14
(D8) 524288 X   - 2621440 X    + 5570560 X    - 6553600 X

             12             10           8          6         4
 + 4659200 X   - 2050048 X   + 549120 X  - 84480 X  + 6600 X

       2
 - 200 X  + 1
```

We now find the same (correct) expression for the 20th polynomial in 1/22 the CPU time. One can generate many examples where careful programming can save orders of magnitude of processing time.

## 6. Special Purpose Packages

There are special purpose packages in MACSYMA for use in General Relativity, high energy physics and several other popular fields of study. These auxiliary packages often offer features which are useful to non-specialists.

### 6.1 COMPONENT TENSOR MANIPULATION

The Component Tensor Manipulation Program, CTENSR, provides the user with tools for computing many of the geometrical objects encountered in General Relativity and Differential Geometry (MACSYMA Reference Manual, 1984). For a given metric tensor, CTENSR can compute the Christoffel symbols, the curvature tensor, the Weyl tensor, the Einstein and Ricci tensors, the geodesic equations of motion and other useful geometrical objects.

In the example below, CTENSR is used to reproduce Dingle's formulas (1933) which, for years, were the cornerstone of hand computations in relativity. It required a team effort of several months in 1933 to compute these formulas by hand. Below, MACSYMA does the identical calculation in less than one CPU minute.

CTENSR is an out of core file which is loaded with the command (C1). In addition, some flags must be preset which control the simplification processes, but we do not show them here. In (C2) we specify the dimension of the manifold to be 4 and then assign to OMEGA the list of coordinate labels.

```
(C1) LOAD("CTENSR")$

(C2) DIM:4$

(C3) OMEGA:[X,Y,Z,T]$
```

The metric tensor, LG, is now specified as a matrix and we declare the functional dependence of the elements of the metric in (C4).

```
(C3) LG:MATRIX([A,0,0,0],[0,B,0,0],[0,0,C,0],[0,0,0,-D]);
```

```
                    [ A  0  0   0  ]
                    [              ]
                    [ 0  B  0   0  ]
(D3)                [              ]
                    [ 0  0  C   0  ]
                    [              ]
                    [ 0  0  0  - D ]
```

```
(C4) DEPENDS([A,B,C,D],[X,Y,Z,T])$
```

The program now runs itself. We ask it to compute and simplify the 10 unique components of the mixed Einstein tensor with the command (C5), and it takes about 40 CPU seconds to accomplish this. The argument "FALSE" inhibits the display of the components.

```
(C5) EINSTEIN(FALSE)$
Time= 40200.0 msecs.
```

We now ask it to print G[1, 4] which is the factored XT component of the mixed Einstein tensor. The subscripts denotes partial differentiation.

```
(C6) G[1,4];

                2                    2          2
(D6)  - (A B  C C  D  + A B B  C  D  + A B  C  C  D
          T  X         T     X          T  X

      2              2                    2
 + A  B  C C  D - 2 A B  C C    D + A B  B  C  D
    T       X               T X          T  X

           2                  2          2  2  2
 + A  B B  C  D - 2 A B B    C  D)/(4 A B  C  D )
    T     X               T X
```

## 6.2 INDICIAL TENSOR MANIPULATION

The tensorial capabilities in MACSYMA include a program called ITENSR (MACSYMA Reference Manual, 1984; Pavelle & Bogen, 1978) which allows the user to manipulate indexed objects. One of the fundamental identities in General Relativity, and differential geometry is Bianchi's identity, which involves the covariant derivative of the Riemann tensor cycled with two of the non-skew-symmetric tensor indices. One normally proves this identity by choosing a coordinate system in which the connection coefficients vanish. There is nothing wrong with that method, but we wish to illustrate the power of the program by carrying out the computation without choosing geodesic coordinates.

```
 U          U          U
C       + C       + C
 RST;W      RTW;S      RWS;T
```

One normally writes the Bianchi identity as above where the 4-indexed object C is the curvature tensor, and the semi-colon is covariant differentiation.

In ITENSR we express the combination of covariant derivatives with the command (C1) where COVDIFF is a function which performs covariant differentiation. In (C2) we expand (C1) and then call a function RENAME which relabels dummy indices in a consistent way so that each term in the sum has at most one pair of dummy indices with a particular label.

```
(C1)  COVDIFF(CURVATURE([R,S,T],[U]),W)+
      COVDIFF(CURVATURE([R,T,W],[U]),S)+
      COVDIFF(CURVATURE([R,W,S],[U]),T)$

(C2) RENAME(EXPAND(D1))$
```

The SHOW command in (C3) displays (D2) with covariant and contravariant indices where CHR2 is the Christoffel symbol of the second kind and %1, %2, ... are dummy indices.

```
(C3) SHOW(D2);
```

$$
\begin{aligned}
(D3)\ & 2\,\mathrm{CHR2}^{U}_{\%1\,S}\,\mathrm{CHR2}^{\%1}_{\%2\,T}\,\mathrm{CHR2}^{\%2}_{R\,W} - 2\,\mathrm{CHR2}^{U}_{\%1\,T}\,\mathrm{CHR2}^{\%1}_{\%2\,S}\,\mathrm{CHR2}^{\%2}_{R\,W} \\
& + 2\,\mathrm{CHR2}^{\%2}_{\%1\,S}\,\mathrm{CHR2}^{U}_{\%2\,T}\,\mathrm{CHR2}^{\%1}_{R\,W} - 2\,\mathrm{CHR2}^{\%2}_{\%1\,T}\,\mathrm{CHR2}^{U}_{\%2\,S}\,\mathrm{CHR2}^{\%1}_{R\,W} \\
& - 2\,\mathrm{CHR2}^{U}_{\%1\,S}\,\mathrm{CHR2}^{\%1}_{\%2\,W}\,\mathrm{CHR2}^{\%2}_{R\,T} + 2\,\mathrm{CHR2}^{U}_{\%1\,W}\,\mathrm{CHR2}^{\%1}_{\%2\,S}\,\mathrm{CHR2}^{\%2}_{R\,T} \\
& - 2\,\mathrm{CHR2}^{\%2}_{\%1\,S}\,\mathrm{CHR2}^{U}_{\%2\,W}\,\mathrm{CHR2}^{\%1}_{R\,T} + 2\,\mathrm{CHR2}^{\%2}_{\%1\,W}\,\mathrm{CHR2}^{U}_{\%2\,S}\,\mathrm{CHR2}^{\%1}_{R\,T} \\
& + 2\,\mathrm{CHR2}^{U}_{\%1\,T}\,\mathrm{CHR2}^{\%1}_{\%2\,W}\,\mathrm{CHR2}^{\%2}_{R\,S} - 2\,\mathrm{CHR2}^{U}_{\%1\,W}\,\mathrm{CHR2}^{\%1}_{\%2\,T}\,\mathrm{CHR2}^{\%2}_{R\,S} \\
& + 2\,\mathrm{CHR2}^{\%2}_{\%1\,T}\,\mathrm{CHR2}^{U}_{\%2\,W}\,\mathrm{CHR2}^{\%1}_{R\,S} - 2\,\mathrm{CHR2}^{\%2}_{\%1\,W}\,\mathrm{CHR2}^{U}_{\%2\,T}\,\mathrm{CHR2}^{\%1}_{R\,S}
\end{aligned}
$$

In (C4) the CANFORM command simplifies (D3). It takes into account the symmetry of the CHR2s in the covariant indices and also changes the labels of dummy indices. It finds that the expression vanishes, as we had anticipated.

```
(C4) CANFORM(D3);

(D4)                              0
```

## 7. Limits of Computational Power

Many people are interested in the size of problems which can be solved now and in speculations about the size of solvable problems in the future. Here are a few examples which may give the reader some insight into this.

### 7.1 GEOMETRICAL SERIES

The first example is based on the elementary geometrical series which is defined in (D1) for an arbitrary integer M. To generate a more difficult problem, $1+1/X$ is substituted for X in (D1).

```
                    M
                   ====                 M
                   \       N          X (X  - 1)
(D1)                >    X    -    ---------- = 0
                   /                   X - 1
                   ====
                   N = 1
```

```
(C2) D1, X = 1+1/X;
```

```
                    M
                   ====
                   \      1     N     1     M       1
(D2)                >   (- + 1)  - ((- + 1)  - 1) (- + 1) X = 0
                   /     X           X             X
                   ====
                   N = 1
```

In (C3), MACSYMA is asked to sum (D2) to 1000 and the resulting expression (D3) (which has been truncated) has a large number of terms. The MACSYMA function NTERMS gives one a handle on the size an expression would be if it were fully expanded and no simplification of like terms occurred and for this case, there are 503 504. From the binomial theorem, it is not hard to show that this number is precisely (M^2+7*M+8)/2 for M = 1000. Finally, MACSYMA is asked to simplify (D3), and after almost 2 CPU hours, the expected answer is returned.

```
(C3) D2,M = 1000,SUM;

       1     1000    1     999           1     2     1
(D3)  (- + 1)     + (- + 1)    + ... + (- + 1)   + (- + 1) + 1
       X           X                   X           X

            1     1000       1
         - ((- + 1)     - 1) (- + 1) X  = 0
            X               X

(C4) NTERMS(LHS(D3));
(D4)                    503504

(C5) RATSIMP(D3);
Time= 7226000 msec. = (2 hrs)

(D5)                    0  =  0
```

## 7.2 TAYLOR SERIES

In this example MACSYMA is asked to compute the first 500 terms of the Taylor series of (D1) about X = 0. The expression is huge, completely useless, and is not displayed here. However, MACSYMA does find it in about 10 CPU minutes.

```
                                      2
                            3    B LOG(X  - X + 1)
(D1)                 A SIN(X ) + ----------------
                                        5
                                       X

(C2) TAYLOR(D1,X,0,500)$
Time= 590000.0 msec.
```

### 7.3 EINSTEIN'S FIELD EQUATIONS IN 4 DIMENSIONS

Here is a problem with more physical motivation which may better illustrate the question of limits on computational power today. For 2, 3 and 4 dimensions we have computed and completely simplified the components of the Ricci tensor (the object which is the source of the vacuum field equations of Einstein's General Theory of Relativity) using the most general metric tensor. In 4 dimensions the components of the metric tensor can be expressed by the matrix

```
[ A(X,Y,Z,T)  B(X,Y,Z,T)  C(X,Y,Z,T)  D(X,Y,Z,T) ]
[                                               ]
[ B(X,Y,Z,T)  E(X,Y,Z,T)  F(X,Y,Z,T)  G(X,Y,Z,T) ]
[                                               ]
[ C(X,Y,Z,T)  F(X,Y,Z,T)  H(X,Y,Z,T)  I(X,Y,Z,T) ]
[                                               ]
[ D(X,Y,Z,T)  G(X,Y,Z,T)  I(X,Y,Z,T)  J(X,Y,Z,T) ]
```

For this case the covariant Ricci tensor consists of 10 second-order coupled partial differential equations in the four space-time variables. The diagonal components have 9990 terms each and the off-diagonal components have 13 280 terms each. Some researchers have speculated that these equations will eventually be solved in this most general case. Pondering the complexity of the expressions, one wonders if this could really come to pass. Each component, when printed, is the size of a 200-page book.

The object of the exercise is to compute and simplify the components of the covariant Ricci tensor. Here is a table of the result of the computation:

| Dimension | Diagonal terms | Off diagonal terms | CPU MACSYMA | CPU CAMAL |
|---|---|---|---|---|
| 2 | 17 | 17 | 5 | 1 |
| 3 | 416 | 519 | 30 | 3 |
| 4 | 9990 | 13 280 | 350 | 45 |

Several Computer Algebra systems attempted to generate the table above. Not all were successful. The CPU times to complete the 3 and 4 dimensional calculations (on the systems which succeeded) ranged from an average of 30 and 350 CPU seconds/component respectively by L. S. Hornfeldt and the authors using MACSYMA to an average of 3 and 45 seconds/component respectively by R. G. McLenaghan and G. J. Fee using CAMAL (Fitch, 1975). MACSYMA was run on the MIT Multics operating system on a Honeywell 68/DPS and CAMAL on an IBM 4341 at University of Waterloo, Canada. The time difference is mainly due to the difference in machine speed. Special algorithms were constructed which reduced the computation times.

### 7.4 COMPUTATIONAL POWER IN 2085?

What about the power of Computer Algebra systems 100 years from now? Below is a $14 \times 14$ matrix:

MATR =

$$\begin{vmatrix} M_{aa} & M_{ab} & M_{ac} & M_{ad} & M_{ae} & M_{af} & M_{ag} & M_{ah} & M_{ai} & M_{aj} & M_{ak} & M_{al} & M_{am} & M_{an} \\ M_{ba} & M_{bb} & M_{bc} & M_{bd} & M_{be} & M_{bf} & M_{bg} & M_{bh} & M_{bi} & M_{bj} & M_{bk} & M_{bl} & M_{bm} & M_{bn} \\ M_{ca} & M_{cb} & M_{cc} & M_{cd} & M_{ce} & M_{cf} & M_{cg} & M_{ch} & M_{ci} & M_{cj} & M_{ck} & M_{cl} & M_{cm} & M_{cn} \\ M_{da} & M_{db} & M_{dc} & M_{dd} & M_{de} & M_{df} & M_{dg} & M_{dh} & M_{di} & M_{dj} & M_{dk} & M_{dl} & M_{dm} & M_{dn} \\ M_{ea} & M_{eb} & M_{ec} & M_{ed} & M_{ee} & M_{ef} & M_{eg} & M_{eh} & M_{ei} & M_{ej} & M_{ek} & M_{el} & M_{em} & M_{en} \\ M_{fa} & M_{fb} & M_{fc} & M_{fd} & M_{fe} & M_{ff} & M_{fg} & M_{fh} & M_{fi} & M_{fj} & M_{fk} & M_{jfl} & M_{fm} & M_{fn} \\ M_{ga} & M_{gb} & M_{gc} & M_{gd} & M_{ge} & M_{gf} & M_{gg} & M_{gh} & M_{gi} & M_{gj} & M_{gk} & M_{gl} & M_{gm} & M_{gn} \\ M_{ha} & M_{hb} & M_{hc} & M_{hd} & M_{he} & M_{hf} & M_{hg} & M_{hh} & M_{hi} & M_{hj} & M_{hk} & M_{hl} & M_{hm} & M_{hn} \\ M_{ia} & M_{jb} & M_{ic} & M_{id} & M_{ie} & M_{if} & M_{ig} & M_{ih} & M_{ii} & M_{ij} & M_{ik} & M_{il} & M_{im} & M_{in} \\ M_{ja} & M_{jb} & M_{jc} & M_{jd} & M_{je} & M_{jf} & M_{jg} & M_{jh} & M_{ji} & M_{jj} & M_{jk} & M_{jl} & M_{jm} & M_{jn} \\ M_{ka} & M_{kb} & M_{kc} & M_{kd} & M_{ke} & M_{kf} & M_{kg} & M_{kh} & M_{ki} & M_{kj} & M_{kk} & M_{kl} & M_{km} & M_{kn} \\ M_{la} & M_{lb} & M_{lc} & M_{ld} & M_{le} & M_{lf} & M_{lg} & M_{lh} & M_{li} & M_{lj} & M_{lk} & M_{ll} & M_{lm} & M_{ln} \\ M_{ma} & M_{mb} & M_{mc} & M_{md} & M_{me} & M_{mf} & M_{mg} & M_{mh} & M_{mi} & M_{mj} & M_{mk} & M_{ml} & M_{mm} & M_{mn} \\ M_{na} & M_{nb} & M_{nc} & M_{nd} & M_{ne} & M_{nf} & M_{ng} & M_{nh} & M_{ni} & M_{nj} & M_{nk} & M_{nl} & M_{nm} & M_{nn} \end{vmatrix}$$

Let us call this matrix MATR for the purposes of illustration below. MACSYMA computes the determinant in about 4 CPU hours. The number of terms in the determinant of a general N x N matrix is N! and no simplification is possible. Indeed, simply attempting to print one term exhausts the memory of the machine. A reason for this is that the algorithm MACSYMA uses for computing determinants involves an expansion by co-factors so that the determinant is a deeply nested product. Of course, as we see, 14! is also a very large number. Perhaps 100 years from now (or perhaps in only 10 years), with advances in hardware and software, one will be able to compute and manipulate such expressions on a hand held device.

```
(C1) DETERMINANT(MATR)$
Time= 13672000 msec.

(C2) 14!;

(D2)                    87178291200
```

## 8. Current Developments

### 8.1 HARDWARE

As supermicros and LISP machines become less expensive and more powerful, large Computer Algebra systems such as MACSYMA will become more readily available. One important trend which we see today is the scientific workstation which integrates symbolic computing, numeric computing and graphics in one powerful system. These systems have a high-resolution bit-mapped display. Under software control, these displays exhibit multiple windows that are used for various functions in an advanced user interface. In this computing environment numeric code can be derived and generated by symbolic computation while graphics can be used to display results obtained on an interactive basis.

Today it costs a minimum of about $20 000 to buy a computer which can run MACSYMA. This impacts the distribution greatly. When the price drops to the equivalent of today's $3000 we can expect to see a large distribution of the software to students. There is already a trend in the application of Computer Algebra in mathematics education at the high school and college levels. Computer Algebra systems running on inexpensive hardware can make a significant impact by providing a "mathematical laboratory" for

pre-college students. Such a laboratory encourages experimentation, increases interest, and provides a degree of concreteness to a subject which is very abstract in nature. It is believed that Computer Algebra tools are necessary for most people who wish to enter into or compete within the high technology areas.

## 8.2 SOFTWARE

Along with the faster, more powerful computers one must have a Computer Algebra system which can run on many different machines. Some Computer Algebra systems are being designed from the ground up with portability in mind. In the case of MACSYMA, various efforts are under way to modify the code and develop interfaces which address the portability issue.

In very general terms, there are a number of areas which are receiving attention now. The degree of user friendliness is being substantially improved through software development. Software that facilitates the maintenance and testing of large Computer Algebra systems is receiving more attention. The interface of symbolic, numeric and graphic facilities is playing an increasingly important role. The use of Computer Algebra systems for generating numeric programs is now an important resource at many sites. Extending the capability of current systems and improving the speed of computations is continuing.

A Computer Algebra system such as MACSYMA will continue to expand and provide more capabilities and features. One useful subsystem under development at Kent State University provides the ability to derive and generate FORTRAN code automatically. Current focus of this research is the generation of code for finite element analysis (Wang & Gates, 1984). MACSYMA is used to derive the strain-displacement matrices and the stiffness coefficients needed. These are then generated into FORTRAN using a code translation package (Wang *et al.*, 1984) which is written in LISP. Techniques used to generate code include the use of automatically generated intermediate labels, interleaving code derivation and code generation, generating functions and subroutines and the use of symmetry to simplify the code generated. Mechanical aids for verifying the correctness of the generated code, as well as the employment of a "code optimiser" are under investigation. Another important area we are considering is the manipulation of expressions in the complex plane.

We previously gave an example of the capabilities of the ordinary differential equation solver ODE described in section 4.15. While this is impressive, solving differential equations is really a kind of scientific art. Given a differential equation whose form or structure is not immediately recognisable, one looks for transformations which will convert the given problem into one which is known. ODE already has some modest heuristic capability in this area. It is in this direction that we can expect future developments in MACSYMA. Over the next several years, we expect that clever expression recognisers will be added which will simplify and transform problems in very much the same way mathematicians who specialise in the field do. The rules are fairly well known; the problem is to build them within the resources of MACSYMA and provide enough user control to allow reasonably rapid convergence to either a solution or to a decision that it cannot solve the problem (i.e. the problem cannot be solved in terms of known functions).

For a large system such as MACSYMA one would like to incorporate as much useful mathematics as possible into the system. This can be done by adding capabilities to basic functions such as INTEGRATE or LIMIT or by writing new code which is added to the

existing SHARE libraries. These enhancements require a large effort. The way it will happen with MACSYMA will be to coordinate users at various sites and encourage them to submit their code to a technical advisory group. The code will then be tested and, if useful, will be prepared for redistribution.

## References


Berlekamp, E. R. (1967). *Bell System Tech. J.* **46,** 1853.

Dingle, H. (1933). *Proc. Nat. Acad. Sci., U.S.A.* **19,** 559.

Fitch, J. P. (1975). *CAMAL Users Manual.* University of Cambridge Computation Laboratory.

Gosper, R. W. (1978). *Proc. Nat. Acad. Sci., U.S.A.* **75,** 40.

Goursat, E. (1904). *A Course in Mathematical Analysis 1* (Section 30). New York: Dover Publications.

Gradshteyn, I. S., Ryzhik, I. M. (1965). *Table of Integrals, Series and Products* (4th ed.), p. 85. (The error is corrected in the 4th edition (1980).) New York: Academic Press.

Karr, M. (1981). *J. Assoc. Comp. Mach.* **28,** 305.

Kamke, E. (1943). *Differentialgleichungen: Lösungsmethoden und Lösungen.* Leipzig: Akademische Verlagsgellschaft.

Klerer, M., Grossman, F. (1971). *A New Table of Indefinite Integrals (Computer Processed).* New York: Dover Publications.

Knuth, D. E. (1980). *The Art of Computer Programming 2* (2nd edn), ch. 4. Reading, Massachusetts: Addison Wesley.

*THE MACSYMA Reference Manual (Version 10)* (1984). Vol. 2, Sect. 3. Massachusetts Institute of Technology and Symbolics.

Mansouri, F., Chang, L. N. (1976). *Phys. Rev.* **D13,** 3192.

Moses, J. (1974). MACSYMA—the fifth year. *Proceedings Eurosam 74 Conference,* Stockholm.

Rosenlicht, M. (1972). *Amer. Math. Monthly,* November.

*Proceedings of the 1977 MACSYMA Users' Conference* (1977). University of California, Berkeley, Ca.

*Proceedings of the 1979 MACSYMA Users' Conference* (1979). Washington, D.C.

*Proceedings of the 1984 MACSYMA Users' Conference* (1984). General Electric Corporate Research and Development, Schenectady, NY.

Pavelle, R., Rothstein, M., Fitch, J. P. (1981). Computer Algebra. *Sci. Am.* **245,** December.

Pavelle, R. (1978). *Phys. Rev. Lett.* **40,** 267.

Pavelle, R., Bogen, R. A. (1977). *Lett. Math. Physics* **2,** 55 (erratum **2** (1978), 255).

Rainville, E. D. (1964). *Elementary Differential Equations.* New York: Macmillan.

Risch, R. (1969). *Trans. Amer. Math. Soc.* **139,** 167.

Risch, R. (1970). *Bull. Amer. Math. Soc.* **76,** 605.

Rosenlicht, M. (1972). *Amer. Math. Monthly,* November, 963.

van Hulzen, J. A., Calmet, J. (1983). Computer algebra systems. In: (Buchberger, B., Collins, G. E. & Loos, R., eds). *Computer Algebra, Symbolic and Algebraic Manipulation,* p. 220. Vienna: Springer-Verlag.

Wang, P. S. (1976). *Math. Comp.* **30,** 324.

Wang, P. S. (1978). *Math. Comp.* **32,** 1215.

Wang, P. S., Gates, B. (1984). *Proceedings of the 1984 MACSYMA Users' Conference.* General Electric Corporate Research and Development, Schenectady, NY. July 23–25, 319.

Wang, P. S., Chang, T. Y. P., van Hulzen, J. A. *Proceedings, EUROSAM '84,* p. 237. London, England.

Yun, D. Y. Y., Stoutemyer, D. (1980). *Symbolic Mathematical Computation.* In: (Belzer, J., Holzman, A. G., eds) *Encyclopedia of Computer Science and Technology,* p. 235. New York: Marcel Dekker.