A Tutorial Introduction to Maple

BRUCE W. CHAR, GREGORY J. FEE, KEITH O. GEDDES, GASTON H. GONNET AND MICHAEL B. MONAGAN
Symbolic Computation Group, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1

(Received 13 July 1985)

The Maple computer algebra system is described. Brief sample sessions show the user syntax and the mathematical power of the system for performing arithmetic, factoring, simplification, differentiation, integration, summation, solving algebraic equations, solving differential equations, series expansions, and matrix manipulations. Time and space statistics for each sample session show that the Maple system is very efficient in memory space utilization, as well as in time. The Maple programming language is presented by describing the most commonly used features, using some non-trivial computations to illustrate the language features.

1. Overview

Maple is an interactive system for algebraic computation. It is designed for the calculations one learns in high school algebra and university mathematics—integers, rational numbers, polynomials, equations, sets, derivatives, indefinite integrals, etc. This article explains how to use Maple interactively as an algebraic calculator, for casual users. It also explains the basic elements of Maple's programming language, giving examples of how to use it for programming extended algebraic calculations.

The Maple project was started at Waterloo in 1980. While the facilities provided by Maple are, at a superficial level, similar to the facilities of other computer algebra systems such as MACSYMA (Pavelle & Wang, 1985) and REDUCE (Fitch, 1985), several design features make Maple unique. The most distinctive feature is Maple's compactness which reduces the basic computer memory requirements per user to a few hundred kilobytes rather than the few thousand kilobytes typically required by MACSYMA or REDUCE. (Of course, Maple's data space may grow to a few thousand kilobytes when performing difficult mathematical computations, if necessary, but this is not generally the case for the calculations required by undergraduate student users nor for many research calculations.) Maple was also designed to allow portability to a variety of different operating systems. Concurrent with the above goals, Maple incorporates an extensive set of mathematical knowledge via a library of functions. The library functions are coded in the user-level Maple programming language which was designed to facilitate the expression of, and the efficient execution of, mathematical operations. A consequence of Maple's design is user extensibility since user-defined functions are equal in status to the system's library functions.

These design goals led to several novel design features. Maple's fundamental data structures are tagged objects represented internally as dynamic vectors (variable-length arrays). Specifically, each instance of a data structure is a vector in which the first component encodes the following information: the length of the structure, the type of data object (such as sum, product, set, rational number, etc.), the simplification status, and the
garbage collection status. The other components of the vector are the operands of the data object and they are typically pointers to similar data structures.

There is extensive use of hashing to minimise storage requirements (by ensuring that equivalent objects are stored only once) and to minimise execution time (by remembering the result of critical operations). Maple uses a fast interpreter to execute the library codes, allowing the compiled "kernel" of the system to include only those operations that have been identified to be efficiency bottlenecks. Another important feature designed to make Maple easy to use is that global "flags" have been avoided as far as possible, by designing hybrid algorithms that incorporate automatic algorithm selection. Further discussion of the design of the Maple system can be found in Char et al. (1983, 1984).

Maple is currently being distributed for a variety of mainframe computers (VAX VMS and Berkeley Unix, IBM VM/CMS, DEC20 TOPS-20) and microsystems (MASSCOMP, Microvax, Cadmus, and various other systems running a Unix-like operating system). A user invoking Maple on any of these systems causes the loading of a binary module (about 140 kilobytes on the VAX) referred to as the "Maple kernel", and the creation of some initial objects. The kernel contains code for doing integer, rational, polynomial and series arithmetic, the Maple programming language interpreter, input/output routines, and other essential system services. The source code for the kernel is written in terms of macros that allow retargeting to a variety of C-like programming dialects, compilers, operating systems, etc. The mathematical functionality of Maple (e.g. the code for integration, the computation of Taylor series, linear algebra, solution of algebraic equations, etc.) resides in its library directory. The sum total of the library source codes, if they were all loaded (an unlikely event), is approximately 1.5 megabytes. When the user requires particular functionality for a calculation, the appropriate files are automatically loaded into main memory for interpretation by Maple's interpreter. Thus the user and system only pay the cost of having essential code resident in main memory. The last line in each "sample Maple session" presented in this article shows the time and space statistics run on a VAX 11/780 under the UNIX BSD 4.2 operating system. It can be seen from these statistics that the largest memory space requirement was about 840 kilobytes.

2. Interactive Maple

2.1. STARTING A MAPLE SESSION

A Maple session is typically initiated by the command maple, and the Maple logo appears (Fig. 1). You can direct Maple to compute something after the logo appears. On

```plaintext
\|/\ | /\| Watnot at University of Waterloo
\ MAPLE \ Version 3.3 --- March 1985
<--- ---\ For on-line help, type help();
>
```

Fig. 1. What you see when starting up Maple.

most implementations of Maple, an interactive prompt (usually >) will appear when Maple is waiting for user input. As you work with Maple, you will see what you type at the keyboard, and the numbers and formulae that Maple will generate in response to your
keyboard commands. Additionally, you may see *words used* messages about the amount of computer memory being used. They typically look like this:

```
words used 10434
```

You may also see *garbage collection* messages. A typical message is:

```
words ret=4140, avail=14318, alloc=120286
```

Several of either kind of message may be printed during a lengthy calculation before the result appears. The user has some control over the occurrence of these messages—see the *Maple User's Guide* (Char et al., 1985). Many users will simply treat their occurrence as an indication that Maple is working on a calculation. A comment to Maple is any line beginning with a #. Maple does not treat a comment as a command to do anything; it simply echoes the comment. In the examples in this tutorial, we have suppressed the "words used" and garbage collection messages and the comment line repetitions, to make the document more compact.

## 2.2. SIMPLE ARITHMETIC IN MAPLE

The basic operators in Maple are the arithmetic operators: +, -, *, /, **, ~, !, the relational operators: =, <, <=, >, >=, and the boolean operators: and, or, not.

After you enter an expression, followed by a semicolon and the return key, Maple calculates the result. You may stretch an expression across several lines; Maple will not compute anything until you finally end the expression with a semicolon and return. Also, multiple commands may appear on one line, separated by semicolons. When directing Maple through a sequence of commands, one can refer to the previously computed result through the symbol "" (double-quote). If you do not wish to print the result of a computation, you can end the expression to be computed with a colon (:) instead of a semicolon. Figure 2 shows some simple arithmetic expressions computed in a Maple session.

```
> (4 + (4 * 6))/(999999-32516);
28

> 2 * 3 * 4 * 5 * 6 * 7 * 8 * 9 * 10 * 11 * 12 * 13 * 14 * 15 * 16
> * 17 * 18;
> " - 18!":
> 0

> quit
words = 4230, memory = 159456, time = .42
```

Fig. 2. Simple numerical calculations.

The command *quit* followed by the return key, ends a Maple session. *done* or *stop* also will work. Note that Maple prints out a final message summarising the computer resources used during the session, as it quits. In this summary, the first number indicates how many computer *words* of memory have been requested of the internal Maple memory manager as data space for this computation. (In a longer session, this measure is counting requests monotonically without taking into account the fact that many requests are
Table 1. Some mathematical functions known to Maple

<table>
<thead>
<tr>
<th>Function</th>
<th>Maple name</th>
</tr>
</thead>
<tbody>
<tr>
<td>exponential</td>
<td>exp</td>
</tr>
<tr>
<td>natural logarithm</td>
<td>ln or log</td>
</tr>
<tr>
<td>square root</td>
<td>sqrt</td>
</tr>
<tr>
<td>absolute value</td>
<td>abs</td>
</tr>
<tr>
<td>sine, cosine, tangent</td>
<td>sin, cos, tan</td>
</tr>
<tr>
<td>secant, cosecant, cotangent</td>
<td>sec, csc, cot</td>
</tr>
<tr>
<td>inverse trigonometric functions</td>
<td>arcsin, arccos, arctan, arccsc, arccot</td>
</tr>
<tr>
<td>hyperbolic trigonometric functions</td>
<td>sinh, csch, tanh, sech, coth</td>
</tr>
<tr>
<td>inverse hyperbolic functions</td>
<td>arsinh, arccosh, arctanh arccsech, arccsch, arccoth</td>
</tr>
<tr>
<td>$\Gamma$ (generalised factorial)</td>
<td>GAMMA</td>
</tr>
<tr>
<td>$\psi, \psi', \psi'' \ldots \left( \frac{\Gamma'(x)}{\Gamma(x)} \right)$</td>
<td>Psi, Psi1, Psi2, ...</td>
</tr>
<tr>
<td>Riemann $\zeta$ function</td>
<td>zeta</td>
</tr>
<tr>
<td>Error function $\int_0^\infty \frac{e^{-t^2} dt}{\sqrt{\pi}}$</td>
<td>erf</td>
</tr>
</tbody>
</table>

satisfied by re-using physical memory that was previously used and then released.) The second number is a measure of the memory space actually allocated during the Maple session (including space for the system code and data space) and the unit of measurement is bytes on most computers. The third number is the total CPU time for the session, measured in seconds.

2.3. BUILT-IN MATHEMATICAL KNOWLEDGE

Maple “knows” about certain mathematical objects (Tables 1 and 2), such as whole and fractional numbers, multivariate polynomials and rational functions, and expressions involving the exponential, logarithmic, and trigonometric functions. This “knowledge” consists of built-in procedures for arithmetic, differentiation, and floating-point approximation of numerical expressions such as $\sin(1)$ or $3^{1/3}$. Maple can also find limits or compute indefinite integrals of some such expressions.

2.4. INTRODUCING MAPLE’S LIBRARY OF PROGRAMS

The heart of Maple is pre-programmed knowledge of mathematical operations. In addition to mathematical functions such as $\text{abs}$, $\text{exp}$, $\ln$, $\cos$, or $\tanh$, there are operations on mathematical functions known to Maple. In this section, we introduce some of the

Table 2. Some mathematical constants known to Maple

<table>
<thead>
<tr>
<th>Constant</th>
<th>Maple name</th>
</tr>
</thead>
<tbody>
<tr>
<td>true, false</td>
<td>true, false</td>
</tr>
<tr>
<td>$\pi$</td>
<td>$\pi$</td>
</tr>
<tr>
<td>$e$ (natural log base)</td>
<td>exp(1), $E$</td>
</tr>
<tr>
<td>$\sqrt{-1}$</td>
<td>$i$</td>
</tr>
</tbody>
</table>
commonly used “simplification” functions: i factor, factor, normal, radsimp, and expand, and some functions for common mathematical operations: diff, int, sum, solve, and taylor. We will also take a look at subs, a useful function for applying identities to expressions.

2.4.1. Simplification functions

The concept of “simplification” in algebra is pervasive. Different variants of this notion are used in computer algebra, see, e.g., Buchberger & Loos (1982). The examples in Fig. 3 show some of the transformation functions that are available in Maple.

2.4.2. Differentiation

diff(expr, var1, var2, \ldots)

diff computes the derivative of expr with respect to a variable. Higher order partial derivatives can be taken by diff with a sequence of variables, such as in Fig. 4, where the partial derivative $\partial^2/\partial s \partial t$ of an expression in $s$ and $t$ is computed.
2.4.3. Indefinite and definite integration

\[ \text{int(integrand, variable of integration)} \]

\[ \text{int(integrand, variable of integration = lo..hi)} \]

Maple will try to calculate a definite or indefinite integral when the \text{int} command is given. However, unlike differentiation, where the computer coding of "how to differentiate" is straightforward, there is no guaranteed success for integration. When Maple fails to solve an integration problem, the request for the calculation is returned as the symbolic "answer" (Fig. 5).

2.4.4. Summation (Fig. 6)

\[ \text{sum(summand, index variable = lo..hi)} \]

\[ \text{sum(summand, index variable)} \]

Fig. 4. Ordinary and partial differentiation.
# A simple indefinite integral.
> int(x^3, x);
\[ \frac{1}{4} x^4 \]

# A simple definite integral.
> int(C*x*ln(x), x = 2..3);
\[ \frac{5}{4} - \frac{3}{2}\ln(3) - 2\ln(2) \]

# Integrate then differentiate, and normalize to recover original expression.
> int(1 / ((x+a)*(x-2+1)), x);
\[
\ln(x + a) - \frac{1}{2}\ln(x + 1) - a\arctan(x)
\]
\[
\frac{1}{2a + 1} - \frac{1}{2a + 1}
\]
> diff(" , x);
\[
\frac{x}{a + 1} - \frac{a}{x + 1}
\]
> normal(");
\[
\frac{1}{(x + a)(x + 1)}
\]

# There are certain cases where the integral can't be computed by Maple.
# The "answer" is then the symbolic integral itself.
> int( exp(x-2), x);
\[ \int \exp(x) \, dx \]
> quit

words = 161987, memory = 483328, time = 23.37

Fig. 5. Symbolic definite and indefinite integration with int.

2.4.5. Solution of equations

\[
\text{solve(set of equations, set of variables),}
\]
\[
\text{solve(equation, variable)}
\]

The solve\text{er} should be given an equation or a set of equations, and an unknown or a set of unknowns to solve for. The notation used for equations and sets in Maple is natural. For example,

\[
\text{solve}\{a=b+2, b-a=t\}, \{a,b\}
\]

is a command to solve a system of two equations for the unknowns \(a\) and \(b\).

Maple will try to solve an equation for an unknown, or a system of equations for a set of unknowns. Any sensible algebraic equation can be given to solve\text{e}, including equations where there are symbols other than the "unknowns to solve for". Maple will always succeed at solving one or more linear equations (given enough computer time and memory for computing). However, its methods for non-linear equations do not always succeed. If it cannot find any solutions, it will print out Warning: no solutions found.
# Definite sums.

```maple
> sum( 1^3, i = 1..123456789087654321 );
580764038888409391286735899637612390226595413138621249148115968669085407761
```

```maple
> sum( (2/3*1^2 - 1/2*1) * (1 -2 + i + 1), i = 0..n-1 );
```

```maple
\[ \frac{-7}{24} n + \frac{7}{36} n - \frac{7}{24} n + \frac{2}{15} n \]
```

# An "indefinite" symbolic sum is similar to an indefinite integral -- namely, it is the function F such that F(b+1)-F(a) would yield the sum with index ranging from a to b.

```maple
> sum(i*1!, i = 1..n-1);
> expand( );
```

# A rational function summation

```maple
> f := (50*n^4+140*n^3+322*n^2-304*n+81); 
```

```maple
\[ f = \frac{50 n + 140 n - 122 n + 304 n + 81}{54 + 43 + 2 + 3} \]
```

```maple
> sum( f, n );
```

```maple
1 + Psi(n + 1/2)
```

```maple
\[ 5 n + 2 n - 11 \]
```

> quit

words = 246837, memory = 843776, time = 43.55

Fig. 6. Summation with sum.

Sometimes this means that no solutions exist, while in other cases it may mean that there are solutions but Maple cannot find them. Additionally, Maple does not always find all solutions to an equation.

By assigning the result of solve to a programming variable (say, s) it is possible to refer to a particular component of a multiple solution using a selection operation or indexing notation (e.g., s[1], s[2], etc.) (Fig. 7).

solve can also solve simple first and second order ordinary differential equations (Fig. 8).

2.4.6. Expression substitution with subs

Often in a calculation of several steps, a known identity such as \( f = ma \) or \( v = ir \) has to be applied to an expression that is the result of a previous step. What is desired is replacing every \( f \) in the expression by \( ma \), or every \( v \) by \( ir \). This can be done in Maple with the subs command. The result of the command

```maple
subs(var = replacement, expression)
```
# Solution of a set of linear equations. The variables to be solved for
# are also collected together into a set. A solution set is returned.
> eqnset := {x+y = b, a*x-2/3*y = k};

\[
\text{eqnset} := \{x + y = b, ax - \frac{2}{3}y = k\}
\]

\[
\text{varset := \{x, y\};}
\]

\[
\text{SolutionSet := solve( eqnset, varset );}
\]

\[
\begin{align*}
2b + 3k & = 0 \\
3a + 2 & = 0
\end{align*}
\]

\[
\text{SolutionSet} := \{(x = \frac{2b + 3k}{3a + 2}, y = \frac{3b - 2k}{3a + 2})\}
\]

# Solution of a nonlinear equation. When just the equation and the
# solution variable are given without set brackets, just the solution
# values are given, also without set brackets.
> solve( q^2-k = a, q );

\[
\frac{1}{2} (k + a), -\frac{1}{2} (k + a)
\]

# One can assign this result to a programming variable, and then
# refer to each solution by an index.
> solutions := ";

\[
\text{solutions} := (k + a), -(k + a)
\]

\[
\text{solutions[2]};
\]

\[
\frac{1}{2} (k + a), -(k + a)
\]

# Solution of a nonlinear system with four solutions.
> solve( {x*y=2, x^2+y-2=5}, {x, y} );

\[
\{(y = 1, x = 2), (x = -2, y = -1), (x = 1, y = 2), (x = -1, y = -2)\}
\]

# A message is printed when no solution can be found.
> solve( sin(q-k) = cos(q^5)-q, q );

\[
\text{Warning: no solutions found}
\]

# Maple does not always give all solutions.
> p := x^7-x^5+4x^3-x^2-4x+1;

\[
\text{solve( p=0, x );}
\]

\[
\text{Warning: polynomial solutions lost - degree} \geq 5
\]

\[
-1, 1
\]

# To better understand this result, look at the factorization of p:
> factor(p);

\[
5 (x + 1) (x - 1) (x + 4 x - 1)
\]

> quit

words = 136281, memory = 450560, time = 22.26

Fig. 7. Solving equations, systems of equations with solve.

is an expression where every occurrence of the symbol \( \text{var} \) in \( \text{expression} \) is replaced by \( \text{replacement} \). A double substitution can be specified by

\[
\text{subs(var1 = replacement1, var2 = replacement2, expression}).
\]

The result is computed by first replacing every occurrence of \( \text{var1} \) by \( \text{replacement1} \), then replacing every occurrence of \( \text{var2} \) by \( \text{replacement2} \). Multiple substitutions can be specified by using \text{subs} in a similar way—a sequence of replacement equations, followed
A differential equation is an equation involving the derivative of an unknown function $y$ of an independent variable $x$. It is necessary to say "$\text{diff}(y(x),x)$" instead of "$\text{diff}(y,x)$" because diff needs explicit declaration of $y$'s dependency upon $x$. Here is a second order D.E. where the unknown function is called $u$.

```maple
> diff(u(x),x,x) + diff(u(x),x) = sin(x);
> solve(%, u(x), x);
```

# A differential equation solution is called for by including the equation, and the names of the dependent and independent variables.

```maple
> solve(%, u(x), x);
```

# solve cannot solve all differential equations as an algebraic formula in the independent variable. Sometimes the formula also includes integrals in the independent variable that can't be further simplified.

```maple
> solve( diff(y(x),x) + y - sin(x)*ln(x) = 0, y, x );
```

# subs is not limited to symbol replacement. It is also possible to specify simultaneous substitutions by enclosing the substitution equations in a set—the examples below will illustrate the distinction.

```maple
> evalf(expr)
> evalf(expr, digits)
```

besides exact integers and fractions, Maple also has "calculator numbers", known also as "floating-point" numbers. Floating-point arithmetic in Maple simulates a decimal
The "subs" (substitute) command can be used to replace subexpressions by new expressions. All but the last argument are substitution equations and the last argument is the expression to be substituted into.

```maple
> msolve := solve( e=m*c^2, m );
msolve := ----
    2

> subs( m=msolve, a=9.8, c=300000, f= m * a );
f = 1088888889*10^-9

> subs( c=300000, m=msolve, a=9.8, f= m * a );
f = 9.8*10^-9
```

Note that the replacements occur in order from left to right as they are specified. In the (misguided) calculation below, the substitution of "c" occurs before there is any "c" in the expression to replace!

```maple
> subs( c=300000, m=msolve, a=9.8, f= m * a );
f = 9.8*10^-9
```

Simultaneous substitution, in contrast to the above "sequential subs", is specified by enclosing substitution equations in a set.

```maple
> subs( {a=b, b=c, c=a}, a+2*b+3*c );
b+2*c+3*a
```

In contrast:

```maple
> subs(a=b, b=c, c=a, a+2*b+3*c );
s a
```

2.4.9. The linear algebra package

There is a package concept in Maple which provides for the development of different packages of functions without worrying about the names of functions clashing between packages. One of these library packages is the linear algebra package, which works with the particular kinds of arrays used in Maple to represent vectors and matrices (Fig. 12). These functions are all referenced via a table of functions called linalg. The subscripts in calculator with a user-specified number of digits of accuracy, by default 10. Numbers are printed in conventional scientific notation, as a decimal fraction times a power of ten. (A more natural fixed-point notation is used if the number is within a few orders of magnitude of 1.) Maple will convert mixed expressions involving both floating-point and rational numbers into floating-point representation.

The value of Digits controls the number of (decimal) digits used during the calculation. The value of Digits is set initially to 10, but you can change it via assignment, just as you would any other variable.

evalf(expr) ("evaluate in a floating-point context") will invoke procedures from the Maple library to numerically evaluate the expression expr. Maple users can extend Maple's built-in knowledge of floating-point evaluation. For example, defining the Maple procedure 'evalf/func' causes the procedure to be invoked when evalf is applied to an expression involving the function func (Fig. 11).
# Compute the first few terms of the Taylor series for exp(x) about x=0.
> t1 := taylor( exp(x), x=0);
\[ t_1 := 1 + x + 1/2 x^2 + 1/6 x^3 + 1/24 x^4 + 1/120 x^5 + O(x^6) \]

# An optional third parameter specifies the order to override the default.
> t2 := taylor( ln(x), x=1, 4);
\[ t_2 := x - 1 - 1/2 (x - 1) + 1/3 (x - 1)^2 + O((x - 1)^3) \]

# The order information in series results is used when doing arithmetic
# on series themselves. In the example below, the result is only O(x^6).
# since \( t_1 \) is only \( O(x^5) \).
> taylor( t1*t1, x=0, 10);
\[ 1 + 2x + 2x^2 + 4/3 x^3 + 2/3 x^4 + 4/15 x^5 + O(x^6) \]

# The series for \( \tan(\sin(x)) \) and \( \sin(\tan(x)) \) agree through several terms.
> taylor( tan(sin(x)) - sin(tan(x)), x=0, 17);
\[ 1/30 x + 1/6 x + 1/2 x + 1/2 x + 1/4 x + 1/120 x + O(x^8) \]

# Asymptotic series at \( x=\infty \) (default order \( O(x^{-6}) \)).
> asympt( exp(l/x)/(1+x^3), x);
\[ 1 - 1 + 1/2 - 1/6 - O(1/6) x \]

> quit

Fig. 10. Taylor and Asymptotic series calculations.

this table are the names of the particular matrix and vector functions. For example, 
linalg[add] is the name of the function that adds two vectors or matrices together. The 
Maple User’s Guide (Char et al., 1985) and Maple’s on-line help facility supply more 
details about which linear algebra functions exist.

Collectively the linalg functions use the programming convention that a vector is an 
array of the form array(1..n), and a matrix is an array of the form 
array(1..m, 1..n).

2.5. OTHER FUNCTIONS

Maple has many more built-in mathematical functions, such as numer(expr) and 
denom(expr) which give the numerator and denominator, respectively, of a rational 
expression, mod(expr,p) which performs modular arithmetic to compute expr mod p, etc.

Maple also has some auxiliary operations of a less formal mathematical nature, such as 
map, table, and words. The Maple User’s Guide (Char et al., 1985) provides an 
explanation of all Maple functions. Additionally, descriptions of them are available via 
the on-line help command within a Maple session.
# Floating-point instead of exact arithmetic is used when an expression contains a floating-point and rational number.

\[ 1.0 + \frac{3}{5}; \]
\[ 1.600000000 \]

\[ 3.0 \times 10^{-4321} - 2.99 \times 10^{-4319}; \]
\[ 4322 \]

\[ 1.2970100000 \times 10^{4322} \]

# The value of Digits controls the accuracy of floating-point calculation.

\[ \text{Digits} := 40; \]
\[ 2.0^{(1/3)}; \]
\[ 1.2599210498487316476721060727822835970 \]

\[ \text{Digits} := 10; \]

# Floating-point evaluations involving functions require evalf.

\[ \sin(1/2) + \sinh(0.57); \]
\[ \sin(1/2) + \sinh(0.57) \]

\[ \text{evalf}(); \]
\[ 1.080798345 \]

# You can temporarily override the value of Digits for a particular computation by including the number of digits as a second argument.

\[ \text{evalf} ( \arcsin(0.78), 40 ); \]
\[ 0.8946658172342359204788218088834256137890 \]

# Some mathematical constants are known to evalf, such as Pi and Euler's constant.

\[ \text{evalf}( \pi * \gamma, 40 ); \]
\[ 1.8133764928892407206106566024491054563250725970 \]

\[ \text{evalf}( \text{exp}(\sqrt{183} \cdot \pi), 40 ); \]
\[ 2.62537412640787343 \]

> quit

words = 47179, memory = 283648, time = 7.55

Fig. 11. Evaluating expressions in floating-point mode.

3. Programming in Maple

Previous examples have illustrated how Maple can be used to solve many problems by making use of Maple's built-in library functions such as `solve`, `int`, `factor`, etc. In this section we describe the Maple programming language. We have chosen to describe the most commonly used language features, illustrating them with examples to give the reader a feel for the style of the language and the facilities available. Because of space limitations, however, our description is necessarily incomplete. For a more detailed description, we refer the reader to Char et al. (1985).

The choice of language constructs and features for a symbolic computation system should result in programs which possess those features which are traditionally associated with "good" programming: readability, simplicity, succinctness, maintainability, the existence and comprehensibility of error/diagnostic information, etc. However, our primary goal is for a language which makes it easy to express algorithms which manipulate mathematical objects. Since mathematical notations and standards already exist, it is important that we choose a syntax which is not too different from that of standard mathematics.
Because of the large number of mathematical objects that we would like to be able to manipulate, an additional necessity of the language is that it must be extensible. Users will want to define new mathematical constants, functions, new domains of computation, and the data structures and operations to represent and manipulate objects in those domains. Another issue that needs to be addressed is the question of efficiency, particularly space efficiency. Because of the huge amount of potential code, and the so-called "intermediate expression swell" phenomenon, symbolic algebra systems make heavy demands on memory. As was mentioned in the Overview, a major goal of the Maple system is that the system be compact and the algorithms and data structures be space efficient.

We have chosen a procedural language. Most of the language constructs are taken from Algol 68. Since, by and large, programmers are most familiar with procedural languages such as Ada, Algol, C, Fortran, Pascal and PL/I, most programmers will find it relatively straightforward to learn how to program in Maple. We have, however, also included some language constructs which support a functional programming style. Because mathematics is functional in nature, it would seem desirable to be able to express
algorithms and functions in this way whenever it seems most natural. Another major design feature of the language is that it is typeless like Snobol and APL. This helps to make the system easy to learn and use in an interactive mode.

3.1. LANGUAGE SYNTAX

As mentioned above, the syntax of the language resembles Algol 68. The control structures provided are described informally below, with the syntax presented in a modified Backus Naur Form where the square brackets [ ] denote optional clauses and the vertical bar | separates two alternatives.

A statement sequence is a sequence of statements (possibly empty) separated by semicolons which are executed sequentially.

\[
\langle\text{statseq}\rangle ::= \langle\text{emptystat}\rangle|\langle\text{statement}\rangle|\langle\text{statseq}\rangle;
\]

The simplest form of a statement is an expression, simply \(\langle\text{expr}\rangle\). An expression consists of names, numbers, lists, sets, arrays, tables, ranges, equations, and functions combined with operators. Sequences (expressions separated by commas) are also valid expressions. The result of executing an expression is that it is evaluated and the result is displayed.

Assignment statements take the form

\[
\langle\text{assignstat}\rangle ::= \langle\text{name}\rangle := \langle\text{expr}\rangle
\]

where \(\langle\text{name}\rangle\) is a variable name (a letter followed by zero or more letters, digits or underscores). Subscripted names, for example \(A[1]\) and \(t[sin(x)]\), are valid names. Names can also be constructed with the concatenation operator ".".

The selection statement

\[
\langle\text{ifstat}\rangle ::= \text{if } \langle\text{condition}\rangle \text{ then } \langle\text{statseq}\rangle \text{ else } \langle\text{elsestat}\rangle \text{ fi}
\]

\[
\langle\text{elsestat}\rangle ::= \langle\text{emptystat}\rangle ! \text{ else } \langle\text{statseq}\rangle
\]

\[
\text{elif } \langle\text{condition}\rangle \text{ then } \langle\text{statseq}\rangle \langle\text{elsestat}\rangle
\]

is used to execute statements conditionally.

Repetition is encoded by using a for statement which has the general format

\[
\langle\text{forstat}\rangle ::= \text{[for } \langle\text{name}\rangle \text{] [from } \langle\text{expr}\rangle \text{] [by } \langle\text{expr}\rangle \text{] [to } \langle\text{expr}\rangle \text{] [while } \langle\text{condition}\rangle \text{] do } \langle\text{statseq}\rangle \text{ od}
\]

Note that this includes both the for and while statements found in Pascal. As an example, the following for statement creates an \(n\) by \(n\) Hilbert matrix. Since a Hilbert matrix is symmetric, we make use of Maple's symmetric indexing function so as to only compute and store half the matrix elements (Maple then knows that \(H[i, j] = H[j, i]\)).

\[
H := \text{array(symmetric, 1..n, 1..n)};
\]
\[
\text{for } i \text{ to } n \text{ do}
\]
\[
\text{for } j \text{ from } 1 \text{ to } n \text{ do}
\]
\[
H[i, j] := 1/(i+j-1)
\]
\[
\text{od;}
\]
\[
\text{od;}
\]

Note in the above that omitting the from and by clauses means that by default, the for loop begins at 1 and the index is incremented by 1. Maple provides two forms of controlled "goto" within a for loop. The break statement causes control to exit the surrounding loop, and the next statement causes control to proceed to the next iteration.
of the loop. The following example illustrates the use of the if and break statements. Maple's `isprime` function is used to determine the first prime that is larger than $10^{12}$.

```plaintext
for k from $10^{12} + 1$ by 2 do
  if isprime(k) then print(k); break fi
od;
```

1000000000039

3.2. PROCEDURES

A subroutine is defined in Maple by a procedure definition of the form

```plaintext
<procedure>::= proc([<nameseq>]) [local <nameseq>];
  [option <nameseq>;] <statseq> end
<nameseq>::= <name> <name>, <nameseq>
```

The value returned by a procedure is the expression evaluated immediately before falling through to the end of the procedure. The local clause defines any variables which are local to the scope of the procedure. An example of the use of the option clause is given later. Other control structures provided for use inside procedures are the `RETURN` statement for immediate return of a result from inside the procedure and the `ERROR` statement for handling errors. (A form of exception handling is provided via the `ERROR` statement and the `traperror` function.)

The semantics of parameter passing are as follows. When a procedure is invoked, the actual arguments are evaluated from left to right. The formal parameters are then substituted textually for the actual parameters. This form of parameter passing is similar to call-by-name, except that the arguments are evaluated only once for efficiency reasons. A consequence of this is that formal parameters cannot be used like local variables, on both the left-hand and right-hand sides of assignment statements. We have termed it "call by evaluated name".

Names mentioned inside a procedure that are neither defined as formal parameters, nor as local variables, are by default global to the entire Maple session. That is, there are no scope rules in Maple. As an example of a Maple procedure, the following procedure forms an array of the coefficients of a polynomial with respect to a specified indeterminate $x$ and assigns the array to the third parameter (which must be a name). The degree of the polynomial is returned as the value of the procedure.

```plaintext
degreecoeffs := proc(p, x, c) local k, d;
d := degree(p, x);
c := array(0..d);
for k from 0 to d do c[k] := coeff(p, x, k) od;
dend;
```

Maple provides a mechanism for allowing a variable number of parameters to be passed to a procedure. Inside the procedure, the actual parameters passed are available as the sequence `args`. The $i$th argument is given by `args[i]` and the number of arguments is given by the variable `nargs`. The following procedure illustrates this feature in computing the minimum of an arbitrary sequence of numbers.

```plaintext
minimum := proc() local i, m;
m := nargs;
for i from 2 to nargs do
  if args[i] < m then m := args[i] fi
od;
m;
end;
```
3.3. ADDITIONAL LANGUAGE FEATURES

In this section we describe some other useful language features. Two useful functions provided by Maple are seq and map. The seq function, seq(<expr>, <name> = <expr>..<expr>) is used to create a sequence. The map function, map(<name>, <expr>, [[<sequence>]]) is used to create a new object of the same type as <expr> having applied the function <name> to the operands of the object <expr>. Any additional arguments to map are passed as additional arguments to the function <name>. For example, in the following, the seq function is used to construct a sequence of sin values, then map is used to apply the evalf function to evaluate to floating-point.

seq(sin(z/4), z = 0..4);
0, sin(1/4), sin(1/2), sin(3/4), sin(1)
map(evalf, ["3"]);
[0., .2474039593, .4794255386, .6818387600, .8414709848]

Other data structures available in Maple include lists, sets, tables and arrays. Lists and sets are sequences enclosed in square brackets [], and braces {}, respectively. The op function is used to extract elements of a list or set. For example, if s is a set, then op(i, s) extracts the ith element of s. In general, op(i, <expr>) extracts the ith operand of any object <expr>. Tables are implemented using Maple’s internal hash tables. The components of a table are accessed using a subscript syntax, for example t[sin] and A[t1,x]. Arrays are implemented as a special case of tables where the indices must evaluate to integers and must lie within specified bounds. Matrices are implemented as two dimensional arrays in Maple. Although Maple does not provide a structured datatype such as Pascal’s record, a list or table can be used to implement structured datatypes.

As an example of a non-trivial Maple procedure, we present an algorithm taken from Knuth (1981, page 631). The algorithm, due to David Yun (1976), computes the square-free factorization of a primitive polynomial over the integers.

```
sqrfree := proc(a, x)
local d, v, w, i, t;
if gcd(a, diff(a,x),'v', 'w') = 1 then RETURN(a) fi;
t := 1;
d := w - diff(v,x);
for i while d <> 0 do
  g := gcd(v, d, 'v', 'w');
  d := w - diff(v,x);
  t := t * g^i
od;
t * v^i
end;
```

This example illustrates the use of Maple’s gcd function for computing greatest common divisors. The syntax gcd(f, g, 'cf', 'cg') returns the greatest common divisor of the polynomials f and g as the function value, and also the cofactors f/gcd(f,g) and g/gcd(f,g), through the arguments cf and cg respectively. Note that the use of single quotes here is necessary as we want to pass the names cf, cg, not their values. By “quoting” an argument, the argument is passed to the procedure unevaluated.
As stated earlier, Maple is a typeless language. By this we mean that no "type declarations" need be made. Maple allows variables to be assigned, and procedures to be passed values of any type. Because of this, some programs will need to ensure that an object is of the right type before proceeding. For example, the above code for the \textit{sqrfree} procedure should check first that the argument is in fact a polynomial over the integers. Other programs will need to know what an object's type is in order to execute different code conditionally. Maple's \texttt{type} function (knows about the built-in types such as sums, products, lists, sets, polynomials, rational functions, etc.) is used for this purpose. New types can be made known to \texttt{type} by the user. As an example of the use of the \texttt{type} function, the procedure \texttt{D} below defines the rules for differentiating constants, sums, products, powers, and one-argument functions.

\begin{verbatim}
D := proc(f, x) local a, b ;
  if f = x then 1
  elif not has(f, x) then 0
  elif type(f, '+' ) then map(D, f, x)
  elif type(f, '*' ) then
    a := op(1,f); b := subsop(1 = 1, f);
    a * D(b,x) + b * D(a,x)
  elif type(f, '^' ) then
    a := op(1,f); b := op(2,f);
    a^b * log(a) * D(b,x) + b * a^(b-1) * D(a,x)
  elif type(f, function) and assigned(DT[op(0,f)]) then
    D(op(1,f),x) * subs(Z = op(1,f), DT[op(0,f)])
  else 'D(f,x)' fi
end;
\end{verbatim}

The table \texttt{DT} allows the user to define the derivative of one-argument functions. For example, to define the derivative of the \texttt{sin} and \texttt{arcsin} functions, we would write

\begin{verbatim}
DT[sin] := cos(Z);
DT[arcsin] := 1/sqrt(1-Z^2);
\end{verbatim}

The procedure \texttt{D} implements the chain rule by substituting for the global variable \texttt{Z} into the \texttt{DT} table component.

3.4. SUBSTITUTION

Maple's \texttt{subs} function provides both sequential and simultaneous substitution. \texttt{subs} performs what is termed "syntactical" substitution. The subexpressions that can be substituted for must have a one to one correspondence with Maple's internal representation of the expression. Although the user cannot get at the internal representation from Maple, he or she can determine what the subexpressions are by application of the \texttt{op} function. That is, the operands of an expression correspond to the subexpressions which \texttt{subs} can match. In the following example, we illustrate the use of substitution and also the use of sets, where the set operators $+$, $*$, and $-$ stand for set
union, intersection, and difference, respectively. The procedure \texttt{linsolve} solves non-square systems of linear equations recursively.

```maple
linsolve := proc(equations, unknowns)
local eqns, eqn, ears, var, sole, sol;
eqns := map(numer, equations) - \{0\};
if eqns = \{\} then RETURN( map( proc(x) x = x end, unknowns ) ) fi;
eqn := op(l,egns);
ears := indets(eqn) * unknowns;
if ears = \{\} then ERROR('System has no solutions') fi;
var := op(1,vars);
# Solve the equation eqn for the unknown var
sol := - coeff(eqn,var,0) / coeff(eqn,var,1);
# Eliminate var from eqns by substitution
sols := subs(var = sol, eqns - \{eqn\});
# Solve recursively the new system for the remaining unknowns
sols := linsolve(sols, unknowns - \{var\});
# Back substitute and simplify to obtain the solution for var
sols + \{var = normal( subs(sols, sol) )\}
end;
```

For example:

```maple
eqns := \{3*z-a-a*y+2*y, 2*y+2*a*x-1/a\};
eqns := \{2 y + 2 a x - 1/a, 3 z - a - a y + 2 y\}
linsolve( eqns, \{x,y,z\} );
```

The final substitution occurring in the procedure substitutes the solutions (represented as a set of equations) into the expression sol, thus obtaining the solution for the unknown var. Substitutions that are performed using a set, are performed simultaneously. We note that Maple’s general solver employs the basic idea used here, a natural way to solve sparse systems of equations. It is, however, more general in that it handles some non-linear equations and attempts to find the best equation eqn and unknown var to solve for at each step.

### 3.5. OPTION REMEMBER

The following procedure computes the Chebyshev polynomials of the first kind from their recurrence relation. The Chebyshev polynomials arise in several branches of mathematics. They have the property of being orthogonal on the interval \([-1,1]\) with respect to the weight function \(1/(\sqrt{1-x^2})\).

```maple
T := proc(n,x) option remember;
if n = 0 then 1
elif n = 1 then x
else expand( 2*x*T(n-1,x) - T(n-2,x) ) fi
end;
```
Note that without specifying option remember, the time taken to compute the \( n \)th Chebyshev polynomial would be exponential in \( n \) and thus horribly inefficient. A user, aware of the problem, would know to program the solution in a loop or to use a table to store the polynomials so that they are computed only once. However, in doing so, such a program would obscure the simple recursive definition of the Chebyshev polynomials that is clear from the code above. By specifying the remember option, Maple automatically remembers the values of the function \( T \) for the actual arguments, so that the results are not recomputed. The advantage of this is that the casual user can express the function in the most natural manner without potential loss of efficiency.

The remember option is implemented by the system as a Maple table associated with each function for which option remember is specified. The table is indexed by the arguments to the function, and the table component stores the function value. Note that the cost of using option remember is very small. The cost of searching the table is negligible compared with executing Maple procedures.

A basic design feature of the Maple system that allows for the efficient implementation of the remember option, is the way in which equivalent objects are handled by the internal simplifier. The first step of the simplifier is to recursively simplify the components of an object. Next it applies the basic simplification rules, simplifying, for example, \( x + x \) to \( 2x \), \( x^4/x/x \) to \( x^2 \), \( \sin(0) \) to 0, the set \( \{x, y, y\} \) to \( \{x, y\} \), \( 1/2 + 1/3 \) to \( 5/6 \), etc. Having done this the simplifier searches the simplification table (a large hash table), to see whether the simplified object already exists. If it does, the new object is discarded, and the old object (the one in the table) is used. If it was not found in the table, the new object is inserted in the table. In this way, equivalent objects are stored only once. This is one important way in which Maple achieves a compact representation for the objects in the system. When comparing two objects \( a \) and \( b \), they are equivalent at this basic level of simplification if and only if \( a \) and \( b \) point to the same object. This allows for a very efficient implementation of set operations and substitution.

The next example takes advantage of this and the remember option in performing what is known as "common subexpression optimization". The procedure optimize below, when passed an expression, returns a sequence of equations corresponding to an optimized evaluation sequence. Because Maple's simplifier has already extracted all of the common subexpressions, this simple algorithm runs in expected linear time and space in the size of the optimized expression.

```maple
optimize := proc(f) local k;
  traverse := proc(f) option remember;
    if type(f, name) or type(f, numeric) then f
    else nextassignment( map( traverse, f ) ) end;
  end;

  nextassignment := proc(f) option remember;
    n := n + 1;
    A[n] := t, n = f;
    t, n
  end;

  n := 0;
  traverse(f);
  seq(A[k], k = 1..n)
end;
```
As an example, consider

$$f := \exp(x^2) \cdot \sin(x) / \cos(x):$$

$$d := \text{diff}(f, x);$$

$$d := \frac{2x \exp(x) \sin(x)}{\cos(x)} + \frac{\exp(x) \sin(x)}{\cos(x)}$$

$$\text{optimize}(d);$$

$$t_1 = x, \quad t_2 = \exp(t_1), \quad t_3 = \sin(x), \quad t_4 = \cos(x), \quad t_5 = 1/t_4, \quad t_6 = 2x t_2 t_3 t_5.$$

$$t_7 = t_3, \quad t_8 = \frac{1}{t_2}, \quad t_9 = t_2 t_7 t_8, \quad t_{10} = t_6 + t_2 + t_9$$

We note that a more complete implementation of a common subexpression optimizer might also seek to minimize the number of temporary variables used, namely, the t.k's.

4. Future Development

The Maple project is an on-going project in which we will continue to develop more facilities and capabilities. We mention here some of the projects that are under development and some of the anticipated future developments.

Language facilities. We are presently developing a graphics package for use in plotting mathematical functions. Additionally, what currently makes up the input/output package in the kernel is being split off from the rest of the Maple system. The goal is to provide an independent system which handles the user interface. We thus hope to greatly improve the user interface, especially for terminals which support bitmap graphics.

Work is under way on the automatic translation of Maple code into Fortran code for use with existing Fortran libraries. We are defining an interface from Maple to Fortran where it will be possible to call a Fortran library function just as one would call any other Maple function. We are looking into common subexpression optimization, especially for generating optimized Fortran code.

We also have plans for an operator algebra facility and user-specified simplification rules.

Algorithmic improvements. Some of the existing mathematical packages are being improved. We have a preliminary version of the Risch integration procedure implemented but this needs to be completed to handle the trigonometric and algebraic cases. The limit package is being improved to handle one-sided limits, infinite limits, and algebraic limits. Improvements to the differential equations routines are underway. The Maple kernel is undergoing a major overhaul in an effort to improve the system's overall efficiency.

Applications packages. There are an unlimited number of applications that could be developed in the future. At present, work has begun on a tensor manipulation package. An investigation of group theory facilities for Maple has been initiated. Work is also under way on a formal power series package.

Undergraduate teaching. Maple is now being used by first and second year students in the undergraduate mathematics curriculum. A pilot project using a VAX 11/785 running
Unix and a VAX 11/780 running VMS serving approximately 400 students per term has been underway since January 1985. By 1986, we expect that upwards of 1000 students per term, including upper-year students, will have access to Maple via this facility. To further increase the capacity of the pilot project, we expect to move to a network of microprocessors connected to a file-server VAX, with the bulk of the computation being done on the microprocessors.

We are grateful for the assistance provided by our co-workers on the Maple project: Len Parent, Stan Devitt, Howard Johnson, Benton Leong, and Bonnie Kent. The Maple project is supported, in part, by grants from the Academic Development Fund of the University of Waterloo, Digital Equipment Corporation, and the Natural Sciences and Engineering Research Council of Canada.

References


