MACSYMA USERS MANUAL

SEPTEMBER 1972

Richard A. Bogen
Hillary O. Capps
Richard J. Fateman
Michael R. Genesereth
Jeffrey P. Golden
Stavros Macrakis
William A. Martin
Joel Moses
Eric Rosen
Linda Rothschild
Steve Saunders
Richard Schroeppel
Robert Siegel
Guy L. Steele
Barry M. Trager
Paul S. Wang
Jon L. White
David Y. Y. Yun
Richard E. Zippel

Work reported herein was supported in part by Project MAC, an M. I. T. interdepartmental laboratory sponsored by the Advanced Research Projects Agency (ARPA), Department of Defense, under Office of Naval Research Contract N00014-70-A-0362-0001.

Acknowledgements

This manual was written by Richard Bogen with the assistance of Jeffrey Golden and is based on a previous manual by Richard Fateman. [3] Chapters 6,11 and 12 were written by Jeffrey Golden, Chapter 7 is by Eric Rosen, Chapter 8 is by Robert Siegel, and Chapters 9 and 10 are by Richard Fateman.

Preface

This manual is an initial draft and is constanly undergoing revision as new features are added to MACSYMA. Any comments, suggestions, criticisms, etc. are welcome and should be sent to:

Jeffrey P. Golden
Room 827
Project MAC
545 Technology Square
Cambridge, Mass. 02138

Table of Contents

1.0 Introduction 1
2.0 General Information 1
3.0 Syntax and Semantics of MACSYMA Expressions
  3.1 Numbers 2
  3.2 Symbols 2
  3.3 Quoted Strings 2
  3.4 Names and Assignment 3
  3.5 Operators 3
  3.6 Functions and Arrays
    3.6.1 Functions 4
    3.6.2 Arrays 5
    3.6.3 Subscripted Functions 6
  3.7 Lists 7
  3.8 Matrices 7
  3.9 IF Statement 9
  3.10 FOR Statement 10
  3.11 Program Blocks 11
  3.12 Evaluation 12
  3.13 Miscellaneous Hints and Facilities 13
4.0 Predefined Mathematical Constants and Functions 15
5.0 MACSYMA Commands and Switches
  5.1 General Purpose 18
  5.2 Manipulation 21
    5.2.1 The Evaluation Command 21
  5.3 Part Selection and Substitution 24
    5.3.1 The Part Commands 25
  5.4 Graphing 26
  5.5 List Handling and LISP-like Functions 27
  5.6 Rational Function Commands 28
    5.6.1 Generalized Rational Function Commands 30
  5.7 Matrix Commands 31
  5.8 Type Testing Commands 32
  5.9 Utility, Input-Output, and Display 32
  5.10 Debugging Commands 35
  5.11 Pattern Matching and Related Commands 36
6.0 The Batch Commands
  6.1 Introduction 39
  6.2 The Simple Format 39
  6.3 The More Complicated Format 39
  6.4 The BATCON Command 40
  6.5 Miscellany 41
7.0 The Secondary Storage Commands
  7.1 Introduction 42
  7.2 Automatic Storage of Expressions 42
  7.3 Explicit Storage 43
  7.4 Saving a MACSYMA Overnight 45
8.0 The Plotting Commands 47

9.0 The Rational Function Commands
  9.1 Basic Commands 51
  9.2 Contagious CRE Commands 52
  9.3 The Rational Coefficient Command 53
  9.4 Simple Extensions to Rational Simplification 55
10.0 The SOLVE Program 56
11.0 Debugging in MACSYMA 59
12.0 The MACSYMA Editor
  12.1 Introduction 61
  12.2 Entering the Editor 61
  12.3 A Description of the Commands 62
Appendix
Bibliography and References

1.0 Introduction

MACSYMA (Project MAC's SYmbol MAnipulator) is a large computer program written in the LISP programming language [6,7] currently running under the ITS time sharing system [8] on a DEC PDP-10 computer at Project MAC, M.I.T. It has capabilities for manipulating algebraic expressions involving constants, variables, and functions. One can differentiate, integrate, take limits, solve equations, factor polynomials, expand functions in power series, plot curves, etc. In addition there are facilities for manipulating lists, subscripted variables, equations, and matrices with many of the usual operations on them being available. Facilities exist which permit the user to extend MACSYMA by adding new functions and operations.

This manual is intended to be a complete index to MACSYMA as of the date shown. We do not pretend to discuss all of the issues involved in the efficient manipulation of algebraic expressions nor is this manual intended to be tutorial in nature. The novice may benefit from reading the MACSYMA Primer first.

2.0 General Information

Commands to MACSYMA are strings of characters representing mathematical expressions, equations, arrays, functions, and programs. Extra spaces and all carriage returns are ignored.

Commands are terminated by @ or $. @ causes the command to be evaluated and the result displayed. $ differs in that the display of the result is suppressed. When typing commands, "rubout" or "delete" deletes (and echoes back at the console) the previous character; ?? deletes the whole command, and causes the line number to be redisplayed.

Lines are consecutively numbered, except that the input line Ci will be followed by an output line (if one is generated) named Di. The next input-output pair will be labelled C(i+1) and D(i+1), respectively. The most recently computed expression may be referred to as "%".

If one command produces several intermediate lines of output, the labels will begin with an E, and the line number will be incremented by one for each additional line.

Any command or expression can be referred to by its line label. The use of C, D, and E as labels can be changed by the user if desired by resetting the values of certain variables (refer to section 5.9).

3.0 Syntax and Semantics of MACSYMA Expressions

This section is intended to give the user a reasonable idea of the expressions MACSYMA permits and their meanings. This section should be read in conjunction with sections 4.0 and 5.0.

3.1 Numbers

Numbers are either integers, rational numbers, or floating point numbers. Integers consist of a string of digits not containing a period, rational numbers are the quotient of two integers and are written as numerator/denominator, and floating point numbers are written as in FORTRAN, i.e. strings of digits containing a period and optionally followed by an integer exponent beginning with the letter E. Negative numbers begin with a minus sign. There is no limit on the number of digits in an integer or rational number but non-zero floating point numbers must have absolute value between .14E-38 and 1.7E38 and are limited to approximately 8 digits precision. (PDP-10 limitations).

```
-17253733574534     6.023E23  -1.6E-19

3.14159      227    -3354665557334/66724255465544
```

3.2 Names

Names designate variables, functions, and arrays. A name consists of a string of letters (including %) and digits of any length beginning with a letter.
(Lower case letters may be typed, but they are immediately converted into the corresponding upper case letters.)

```
X  %PI  LAMBDA65353773EPSILON  A%123
```

3.3 Quoted Strings

A string of characters of any length may be constructed by enclosing the string in ?'s. To include a ? or an @ , # or $ in the string it is necessary to precede it with a \. Quoted strings are useful in printing titles for output or messages such as those giving instructions for inputting data.

```
PRINT(?IS X POSITIVE\??)
```

3.4 Names and Assignment

Names used as variables are assigned values by writing the name of the variable followed by a : followed by an expression representing the value to be assigned to the variable. A name can be assigned a new value at any time. The value of a variable can be a number, a matrix, a list, a symbol, or any other MACSYMA expression. Some simple examples follow. More complicated ones will be presented later. (The comments in parentheses are only for the reader's benefit and are not actually typed to MACSYMA.)

```
line numbers   expressions        comments

    (C1)    A:16$                 (integer)
    (C2)    LAMBDA: -3/37$        (rational number)
    (C3)    X:D1@                 (X is assigned the value of D1)
    (D3)           16
(Note that every input expression Ci produces an output
expression Di whether it is displayed or not).
    (C4)    RHO:SIGMA@            (since SIGMA has no value at this time
                                   RHO is assigned the symbol SIGMA)
    (D4)           SIGMA
    (C5)    SIGMA: .005$          (floating point)
    (C6)    RHO@                  (RHO still has its old value
    (D6)           SIGMA           since it hasn't been reassigned
                                   a new one)
```

The MACSYMA variable VALUES is a list of all the variables which are bound (i.e. have been assigned values) up to the present time.

The special assignment operator :: assigns the value of the expression on its right to the value of the symbol on its left, which must evaluate to a simple variable or array element. Thus continuing with the above examples:

```
    (C7)    RHO::LAMBDA$
    (C8)    SIGMA@
    (D8)           -3/37
    (C9)    VALUES@
    (D9)           [A,LAMBDA,X,RHO,SIGMA]
```

3.5 Operators

Expressions are constructed by using the operators + , - , * , . , / , ** (or ^) , and functional composition. The usage and priorities from highest to lowest are:

Operator Name	Symbol	Usage
factorial	!	postfix
exponentiation	** ^	infix
negation	-	prefix
divide,multiply	/ * .	infix
add,subtract	+ -	infix

Period is used for non-commutative product. It must be preceded and followed by a space when any ambiguity can arise with respect to floating point numbers.

Operators of equal priority are performed left to right. Parentheses can be used to change the order of evaluation. Also functional application has the highest priority. Thus SIN(A*X**Y/Z)**2 means (SIN(A*(X**Y)/(Z)))**2

In order to reduce the number of operators to be dealt with X-Y becomes X+(-1)*Y and X/Y becomes X*Y**(-1) in MACSYMA but this usually need not concern the user.

3.6 Functions and Arrays

3.6.1 Functions

Functions of any fixed number of arguments can be defined in MACSYMA by using the operator := . The left side of a function definition consists of the name of the function followed by the list of formal parameters enclosed in parentheses. The right side consists of the function body. Note that the function body is not evaluated at function definition time, but only when the function is called. The operator " preceding a name can be used to force immediate evaluation of that name inside the body of a function definition. When a function is called, any free variables in the function body will take on the values which they have at the time of the call, the formal parameters will be bound to the actual arguments, and the function body will be evaluated. It is permissible to define functions which call themselves or which are recursive to an arbitrary depth.

The MACSYMA variable FUNCTIONS is a list of all user defined functions.

```
(C3) F(X):=X**2+Y$

(C4) F(2)@

(D4)                 Y + 4

(C5) Y:7$

(C6) F(2)@
(D6)                   11

(C7) G(Y,Z):="D4+Z**2@
                          2
(D7)        G(Y, Z) := Z   + Y + 4
```

```
(C8) G(X,2)@
(D8)               X + 8

(C9) FUNCTIONS@
(D9)           [F(X), G(Y, Z)]
```

3.6.2 Arrays

Arrays in MACSYMA can have any number of dimensions. Array elements are referred to by subscripted names. A subscripted name is a name followed by a list of subscripts enclosed in brackets. Arrays can be declared or undeclared. Declared arrays are similar to FORTRAN arrays. The user declares the number of dimensions and indicates the maximum value of each subscript. The system then allocates space for the entire array. To declare an array the user types the command ARRAY(name,dim1,dim2,...,dimk). This sets up a k-dimensional array. The subscripts for the ith dimension are the integers running from 0 to dimi-1. If the user uses a subscripted name before declaring the corresponding array, an undeclared array is set up. Undeclared arrays are in some sense more general than declared arrays. The user does not declare their maximum size, and they grow dynamically as more elements are assigned values. The subscripts of undelcared arrays need not even be numbers. However, unless an array is rather sparse, it is probably more efficient to declare it than to leave it undeclared. The ARRAY command can be used to transform an undeclare array into a declared array.

Array elements can be assigned values explicitly with the : operator or implicitly by means of an associated function, and the values assigned may be any MACSYMA expression. To understand the implicit assignment we must understand what MACSYMA does when asked to evaluate a subscripted name. MACSYMA first evaluates the subscripts left to right. Then it does an array access to see if the requested array element already has a value. If it does, the value is returned. If it does not, MACSYMA checks to see whether the array has an associated function. If not, the subscripted name (with the subscripts evaluated) is returned. (This is standard MACSYMA practice - if there is no value for a name, the name itself is returned when an evaluation is done.) If there is an associated function, the parameters of the function are bound to the given subscripts, and the function body is evaluated. The value of the function is stored in the appropriate array element and returned. Note that once an element is computed by the associated function it is stored so that next time it is needed it will not have to be recomputed. A consequence of this is that unless the user uses the REMOVE or REMVALUE commands to kill an array element, the associated function will never be called a second time on the same arguments. Thus the user should be aware that even if he redefines the associated function, those values which already exist will stay around. The only way to change the value of an array element is to use the : operator.

These associated functions are defined with the := operator. Their definition looks exactly the same as ordinary function definitions, except that the parameters in the left side of the definition are enclosed in brackets instead of parentheses.
The MACSYMA variable ARRAYS is a list of all the arrays that have been allocated, both declared and undeclared.

```
(C11) A[N]:=N*A[N-1]$

(C12) A[0]:1$

(C13) A[5]@
(D13)                  120

(C14) A[N]:=N$

(C15) A[6]@
(D15)                   6
```

{Note that the definition in C14 is being used because A[6] had no value up to this time.}

```
(C16) A[4]@
(D16)                  24
```

{Since A[4] was assigned a value as a result of A[5] being computed, the new definition is not used.}

3.6.3 Subscripted Functions (Arrays of Functions)

MACSYMA provides a very natural notation for subscripted functions. They are defined as are ordinary functions, using the := operator. The left side of the defintion however, consists of the function name followed by the subscripts, enclosed in brackets, followed by the arguments, enclosed in parentheses. The subscripts (which are not evaluted at defintion time) must be either all numeric or all symbolic. If they are symbolic, they are treated as additional arguments to the function.

```
(C18) T[1](X):=X!@
(D18)
                        T (X)  := X!
                         1

(C19) T[1](5)@
(D19)                  120

(C20) F[I,J](X,Y):=X**I+Y**J@
                               J    I
(D20)       F    (X, Y)  := Y   + X
             I, J
```

```
(C21) F[2,B](C,3)@
                         2    B
(D21)                   C  + 3
```

The user should realize that MACSYMA handles subscripted functions as a special type of array, not as functions. Thus the user should never redefine a subscripted function without KILL'ing or REMARRAY'ing it first. If he does, he may find that the old definition is still being invoked.

A subscripted function is actually an array of functions. The subscripts pick out the correct function from the array. The function is then applied to the arguments.

3.7 Lists

Lists are ordered sets of elements which can be any MACSYMA expressions including lists. They are written enclosed in brackets with elements separated by commas. There are functions for many list operations such as deleting elements, selecting an element, reversing a list, etc. These are described in section 5.5. Lists may be used whenever it is desired to evaluate a sequence of MACSYMA expressions.

```
(C1)   [X**2,Y/3,-2]$       (D1 exists even though it
                              hasn't been displayed)
(C2)   FIRST(%)*X@
                              3
(D2)                         X
(C3)   [A,D1,D2]@
                                  2  Y        3
(D3)                         [A, [X , - ,-2] ,X ]
                                     3
```

3.8 Matrices

Matrices are like 2 dimensional arrays but are treated differently in MACSYMA because of special operations defined on them. They may be constructed by using the command MATRIX whose arguments are lists representing the rows of the matrix. (The command ENTERMATRIX may also be used to construct a MATRIX. See section 5.7).

The operators + , - , * , . ,and ** (for raising to a positive or negative integer exponent) may be applied to matrices and have their usual meaning (note that A**(-1) is the inverse of the matrix A but 1/A is not). If a matrix is multiplied by a list, the list will be taken to be a vector. If a matrix is multiplied by a scaler, the scaler will be kept outside the matrix unless the command EXPAND is used (see section 5.1 and example below).

An element of a matrix may be referenced by subscripting as with arrays. Many functions are available for operating on matrices and are described in section 5.7. Some examples follow: (D1 is the list shown in the previous section)

```
(C4)    MATRIX(D1,[0,5/2,X],REVERSE(D1))@

                    * 2   Y     *
                    *X    -   -2*
                    *     3     *
                    *           *
                    *     5     *
(D4)                *0    -   X *
                    *     2     *
                    *           *
                    *     Y    2*
                    *-2   -   X *
                    *     3     *

(C5)   1+%[3,3]@
                                 2
(D5)                            X  + 1
(C6)   D4[2,1]: -1/2$

(C7)    MATRIX([1,X],[-X,2])@

                         * 1    X*
(D7)                     *       *
                         * - X  2*

(C8)    D7*MATRIX([A],[B])@

                        * B X + A *
(D8)                    *         *
                        *2 B - A X*

(C9)    TRANSPOSE(%)@

(D9)                *B X + A  2 B - A X*

(C10) %*D7@

(D11)      *   2                 2           *
           *A X  - B X + A   B X  - A X + 4 B*

(C12)    D7**(-1)@

                                 *2   - X*
                                 *       *
                                 *X    1 *
(D12)                            ---------
                                   2
                                  X  + 2


(C13)    %,X=2,EXPAND@
```

```
                   *1    1*
                   *-  - -*
                   *3    3*
(D13)              *      *
                   *1  1  *
                   *-  -  *
                   *3  6  *
```

3.9 IF Statement

The syntax is IF logical-expression THEN expression1 ELSE expression2. The result of an IF statement is expression1 if logical-expression is true and expression2 if it is false. Expression1 and expression2 are any MACSYMA expressions including lists, and logical-expression is an expression which evaluates to TRUE or FALSE and is composed of relational and logical operators which are as follows:

Name	Symbol	Type
greater than	>	relational
equal to	=	"
less than	<	"
greater than or equal to	>=	"
less than or equal to	<=	"
AND	& , AND	logical
OR	OR	"

NOT is also provided but as a function (not as a prefix operator).

The relational operators all have equal priorities which are less than the priorities of the arithmetic operators and greater than that of the logical operators. The priority of AND is greater than that of OR.

The ELSE clause may be omitted but not in cases where the IF statement is expected to yield a value as in assignment. IF statements may be part of other IF statements. Some examples are given below.

```
(C1)  FIB(N):= IF N=1 OR N=2 THEN 1
            ELSE FIB(N-1)+FIB(N-2)$
(C2)  FIB(1)+FIB(2)@
(D2)                    2
(C3)  FIB(3)@
(D3)                    2
(C4)  FIB(5)@
(D4)                    5
```

```
(C5)  ETA(MU,NU):= IF MU=NU THEN MU
             ELSE  IF  MU>NU  THEN MU-NU
             ELSE MU+NU$
(C6)  ETA(5,6)@
(D6)                   11

(C7)  ETA(ETA(7,7),ETA(1,2))@

(D7)                   4

(C8)   IF (W:FIB(7))>10 THEN [G:X*SIN(X/W),INTEGRATE(G,X)]
     ELSE [G:COS(X+W),DIFF(G,X,2)]@

                        X                X                X
(D8)            [X SIN(--),169 SIN(--) - 13 X COS(--)]
                        13               13               13

(C9)   IF 4+1>3 OR NOT(5>=2) AND 6<=5 THEN TRUE@
(D9)             TRUE
```

3.10 FOR Statement

This statement is useful in iteration and is analogous to the DO statement in FORTRAN although it is much more general. There are three forms the syntaxes of which are:

(a) FOR variable:initial-value STEP increment THRU limit DO expr

(b) FOR variable:initial-value STEP increment WHILE logical-expression DO expr

(c) FOR variable:initial-value STEP increment UNLESS logical-expression DO expr

The increment must be a positive number, but the initial-value and limit may be any arithmetic expressions. Logical-expression is as in the IF statement and. The function GO may be used to transfer control out of a FOR statement (see section 5.11). If the increment is 1 then "STEP 1" may be omitted. The iterated expression may be any single MACSYMA expression including an IF or FOR statement. If the user wants to have more than one expression he may use the BLOCK statement (see section 5.11). Rather than explain each case some examples will be given.

```
(C1)   FOR A:-3 STEP 7 THRU 26 DO DISPLAY(A)$
(E1)          A = -3
(E2)          A =  4
(E3)          A = 11
(E4)          A = 18
(E5)          A = 25
```

```
    (C6)   S:0$
    (C7)   FOR I:1 WHILE I<=10 DO S:S+I0
    (D7)          DONE
```

(DONE is the value of a FOR statement. All statements in MACSYMA have a value even if it is a trivial one.)

```
    (C8)   S@
    (D8)          55

    (C9)   SERIES:1$
    (C10)  TERM:%E**SIN(X)$
    (C11)  FOR P:1 UNLESS P>7 DO
             BLOCK(TERM:DIFF(TERM,X)/P,
             SERIES:SERIES+EV(TERM,X=0)*X**P)$
    (C12)   SERIES@
```

(D12) $$\frac{X^7}{96} - \frac{X^6}{240} - \frac{X^5}{15} - \frac{X^4}{8} + \frac{X^2}{2} + X + 1$$

The last example computes seven terms of the Taylor series for EXP(SIN(X)). Here %E represents the transcendental number e. EV is a command which evaluates its first argument subject to the conditions imposed by the rest of its arguments.

3.11 Program Blocks

BLOCKs in MACSYMA are analogous to subroutines in FORTRAN or procedures in ALGOL. BLOCKs are useful for grouping together a sequence of related calculations.
The syntax is:

```
          BLOCK([v1,...,vk], stmt1,...,stmtn)
```

where the vi are variables which are local to the BLOCK and the stmti are any MACSYMA expressions. If no variables are to be made local then the list may be omitted but then any variables used within the block will be identical to variables with the same names used outside of the block.

The value of the block is the value of the last statement or the value of the argument to the function RETURN which may be used to exit explicitly from the block. In addition the function GO may be used to transfer control to the statement of the block that is labeled with the argument to GO which is an atomic symbol. To label a statement precede it by an atomic symbol as another argument to BLOCK. For example BLOCK([X],X:1,LOOP,X:X+1,...,GO(LOOP),...) . The argument to GO may be any expression which evaluates to a label. For example GO(IF X>Y THEN PLACE1 ELSE COMPUTEPLACE(X)). Going out of a block causes the unbinding of all variables which were bound in the block. Blocks typically appear on the right side of a function definition but can be used in other places as well.

```
(C30) HESSIAN(F):=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),
          RETURN(DETERMINANT(MATRIX([DFXX,DFXY,DFXZ],[DFXY,DFYY,DFYZ],
                [DFXZ,DFYZ,DFZZ]))))$

(C31) HESSIAN(X**3+Y**3-3*A*X*Y*Z)@
                            3              2   3          2   3
(D31)                 - 54 A  X Y Z - 54 A  Y  - 54 A  X

(C32) QUOTIENT(%,-54*A**2),Z=1@
                                 3            3
(D32)                           X  + A Y X + Y
```

The above example computes the Hessian of a cubic curve (The Folium of Descartes) which turns out to be invariant under this transformation since the result is of the same form.

3.12 Evaluation

In general whenever the user types in an expression (not appearing in a function definition) it is evaluated once, i.e. all bound variables and functions in it are replaced by their values and then simplified. If it is desired to suppress the evaluation of a particular variable or function it can be preceded by a single quote, i.e. ´. The fact that any expression can be referred to by its line label implies that variables such as C2, D1, or E7 when used in an expression may have values automatically set as a result of a previous computation. Therefore care must be taken not to use these as atomic variables (those which stand for themselves) because they will be replaced by their values. When referring to a past expression it is preferable to use Di rather than Ci since Di is already the result of evaluating Ci and there is no need to do this again unless the values of some variables used in Ci have changed.

All functions are either noun-type or verb-type, most being verb-type. A verb function is a function which attempts to effect an application of the itself to its arguments and thereby remove itself from the expression. For example, INTEGRATE is a verb-function and ordinarily will attempt to perform an integration. On the other hand, SIN is a noun-function, and will not attempt to evaluate itself, although it will evaluate its arguments. The EV command (see section 5.2) can be used to evaluate an expression in a context which says that (for example) all SINs should be numerically evaluated. That is, selected noun forms can be converted to verb forms. Similarly, if a normally verb-function is desired to operate as a noun-type function, it may be so declared via the function NOUN. Thus INTEGRATE, when declared a noun via the command NOUN(INTEGRATE), would normally return an integral, even if the integration could be performed. If the function F is a verb ´F can be used as the noun form for F. If F is a noun, ´´F (2 single quotes) can be used as the verb form. If a verb-function cannot be evaluated, as for example an integral which cannot be computed, it is simply returned as though it were a noun-function. If F is already a noun, ´F is the same as F.

```
(C1)    X:2$
(C2)    Y:'X+X$
(C3)    Y+1@
(D3)                        X + 3
(C4)    X+3@
(D4)                        5
(C5)    F(X):=(X+Y)*"X/2@
(D5)                        F(X) := Y + X
(C6)    'DIFF(F(Z),Z,2)@
                               2
                              D
(D6)                          --- (Z + X + 2)
                                2
                              DZ
(C7)    %,DIFF@
(D7)                       0
```

3.13 Miscellaneous Hints and Facilities

There are several uses of ? in MACSYMA: (1) ?? is used to cancel a line typed to MACSYMA. (2) ?string? is used to quote a string of characters as in printing a message. (3) ?operator? is used to refer to the operator in an expression such as the result of PART(expression,0).

The following characters typed while holding down the control key have special functions:

G - enters top-level LISP after resetting all variables and breaking out of all functions. It is not possible to continue an interrupted calculation after a control-G, but typing (CONTINUE) will return to MACSYMA.

K - reprints the current line. This is useful when many rubouts have obscured the line.

L - clears the screen on Display consoles and does a control-K.

H - makes a "breakpoint" in MACSYMA and prints the time used in the current computation. Control-H does not reset any values. Altmode (or Escape) P followed by a space will return to MACSYMA and resume the computation.

X - aborts a computation and returns control to top-level MACSYMA.

D - causes garbage collection statistics to be printed out each time a garbage collection takes place.[6,7] If the percentages drop below 5% consistently, chances are that the computation is too large for the MACSYMA system being used.

C - stops printout of garbage collection statistics turned on by control-D.

W - stops printout at the console (the computation continues).

V - resumes printout at the console turned off by control-W.

B - seizes the lineprinter so that subsequent output at the user's console will be printed.

E - frees up the lineprinter which was seized by conrtol-B.

A - similar to control-H except that MACSYMA has control at this breakpoint. To exit type EXIT@.

Sometimes when a user gives a command the message "... being loaded" will be printed. This means that the command used is not in the initially loaded MACSYMA but is being loaded in now via the dynamic loader. Infrequently used commands are not initially loaded into MACSYMA in an effort to save space.

When in LISP typing (CONTINUE) will return to MACSYMA. Typing (SUPERVISOR) will also return to MACSYMA but will do a KILL(HISTORY) in addition. (see section 5.9).

4.0 Predefined Mathematical Constants and Functions

All of the functions mentioned below take one argument (say X) unless stated otherwise. Certain properties are indicated in braces following each function.

Properties

N: the function can be evaluated for numeric arguments by using the NUMER flag of EV. (see section 5.2)

I: the function may be integrated.

ND: the function can't be differentiated.

S: the function is automatically simplified only for certain numeric arguments. (like integers or special angles).

A: the function is automatically evaluated for any numeric argument.

Constants

%E is the base of the natural logarithm. It also serves as the exponential function, i.e. %E**X is used for EXP(X). %PI is used for pi and %I is the square root of minus one.

Simple Functions

ABS {A,ND} - absolute value.
ENTIER {A,ND} - largest integer <= X.
SIGNUM {A,ND} - if X<0 then -1 else if X>0 then 1 else 0.

Miscellaneous Functions

SQRT {N,I,S} - same as X**(1/2)
LOG {N,I,S} - the natural logarithm.
BINOMIAL(X,Y) {S,ND} - X*(X-1)*...*(X-Y+1)/Y!
GAMMA {N,S} - the gamma function. GAMMA(I)=(I-1)!
BETA(X,Y) {N,S,ND} - same as GAMMA(X)*GAMMA(Y)/GAMMA(X+Y)
FACT(X,Y) {S,ND} - X*(X-1)*...*(X-Y+1)
if Y is omitted then same as X!
EULER {ND,S} - gives the Xth EULER number for integer X.
BERN {ND,S} - gives the Xth BERNOULLI number for integer X.
The switch ZEROBERN [TRUE] if set to FALSE excludes zeroes from the BERNOULLI numbers.
PSI {ND} - derivative of LOG(GAMMA(X)).

Circular Functions

COS {N,I,S} - cosine.
SIN {N,I,S} - sine.
TAN {N,I,S} - tangent.
SEC {N,I,S} - secant.
CSC {N,S} - cosecant.
COT {N,I,S}- cotangent.

Inverse Circular Functions

ACOS , ASIN , ATAN , ASEC , ACSC , ACOT

Hyperbolic Functions

COSH {N,S} , SINH {N,S} , TANH {N,S} , SECH , CSCH , COTH

Inverse Hyperbolic Functions

ACOSH , ASINH , ATANH , ASECH , ACSCH, ACOTH

Examples

```
(C11) SIN(%PI/12)+TAN(%PI/6)@
                           %PI        1
(D11)                  SIN(---) + -------
                            12    SQRT(3)

(C12) %,NUMER@
(D12)                      0.8361693

(C15) BETA(1/2,2/5)@
                                        2
                       SQRT(%PI) GAMMA(-)
                                        5
(D15)                  ------------------
                                   9
                             GAMMA(--)
                                   10

(C16) %,NUMER@
(D16)                      3.6790924

(C19) DIFF(ATANH(SQRT(X)),X)@
                                 1
(D19)                   -----------------
                        2 SQRT(X) (1 - X)
```

Complex Expressions

MACSYMA attempts to simplify expressions involving %I although it may not do quite everything the user desires. Sometimes the required manipulation may be achieved by various operations directed by the user as the examples below illustrate.

```
(C20) (SQRT(-4)+SQRT(2.25))**2@
                                         2
(D20)                     (2 %I + 1.5)

(C21) EXPAND(%)@
(D21)                     6.0 %I - 1.75

(C23) EXPAND(SQRT(2*%I))@
(D23)                        %I + 1
```

5.0 MACSYMA Commands (Functions) and Switches (Variables)

Following is a list of all MACSYMA Commands divided into functional classes. Special variables are mentioned in the description of some commands which affect their operation. Their default value is enclosed in brackets.

5.1 General Purpose

MIN(n1,n2,...) gives the minimum of the expressions n1, n2,...

MAX(n1,n2,...) gives the maximum of the expressions n1, n2,...

DIFF(exp,var1,n1,...,vark,nk) differentiates exp with respect to each vari, ni times. If just the first derivative with respect to one variable is desired then the form DIFF(exp,var) may be used. If the noun form is required (as, for example, when writing a differential equation), 'DIFF should be used and this will display in a two dimensional format. DIFF(exp) gives the "total derivative", that is, the sum of the derivatives of exp with respect to each of its variables times the function DELTA of the variable. DERIVATIVEABREV [FALSE] if TRUE will cause derivatives to display as subscripts.

DEPENDENCIES(f1,...,fn) declares functional dependencies used by DIFF. Each fi (i=1,n) has the format f(v1,...,vm) where each vj (j=1,m) is a variable on which f depends. Thus DIFF(Y,X) is 0, initially. Executing DEPENDENCIES(Y(X)) causes future differentiations of Y with respect to X to be displayed as

$$\frac{DY}{DX}$$

Dependencies need not be declared when they are given explicitly in the expression as in DIFF(Y(X),X).
Sometimes the user may forget what functional dependency relations he created for some functions. They may be retrieved by using the command GETDEPENDS (see sect. 5.9).

GRADEF(f(x1,...,xn),g1(x1),...,gn(xn)) defines the derivatives of the function f with respect to its n arguments. That is, df/dx1 = g1(x1), etc. This is needed when, for example, the function is not known explicitly but its first derivatives are and it is desired to obtain higher order derivatives.

INTEGRATE(exp,var) integrates exp with respect to var or returns an integral expression if it cannot perform the integration. INTEGRATE(exp,var,low,high) finds the definite integral of exp with respect to var from low to high. Several methods are used, including direct substitution in the indefinite integral and contour integration. Improper integrals may use the names INF for positive infinity and MINF for negative infinity. If an integral "form" is desired for manipulation (for example, an integral which cannot be computed until some numbers are substituted for some parameters), the noun form 'INTEGRATE may be used and this will display with an integral sign.

```
(C30) INTEGRATE(SIN(X)**3,X)@
                   3
                COS (X)
(D30)           ------- - COS(X)
                   3

(C31) INTEGRATE(X**A/(X+1)**5/2,X,0,INF)@
IS  A + 1  POSITIVE, NEGATIVE, OR ZERO?

POS@
IS  4 - A  POSITIVE, NEGATIVE, OR ZERO?

POS@
                        BETA(A + 1, 4 - A)
(D31)                   ------------------
                                2
```

LIMIT(<u>exp</u>,<u>var</u>,<u>val</u>,<u>dir</u>) finds the limit of <u>exp</u> as the real variable <u>var</u> approaches the value <u>val</u> from the direction <u>dir</u>. <u>Dir</u> may have the value PLUS for a limit from above, MINUS for a limit from below, or may be omitted (implying a two-sided limit is to be computed). LIMIT uses the following special symbols: INF (positive infinity) and MINF (negative infinity). On output it may also use UND (undefined) and IND (indefinite but bounded). 'LIMIT may be used to simply create a limit noun form and this will display in a two-dimensional form.

```
(C32) LIMIT((1+X)**(1/X),X,0)@
(D32)                    %E
```

RESIDUE(<u>exp</u>,<u>var</u>,<u>val</u>,<u>order</u>) computes the <u>order</u>th residue in the complex plane of the expression <u>exp</u> when the variable <u>var</u> assumes the value <u>val</u>. This is defined to be the coefficient of (<u>var</u> - <u>val</u>)**(-<u>order</u>) in the Laurent series for <u>exp</u>.

```
(C33) RESIDUE(S/(S**2+A**2),S,A*%I,1)@
                             1
(D33)                        -
                             2
```

SOLVE(<u>exp</u>,<u>var</u>) solves the algebraic equation <u>exp</u> for the variable <u>var</u>. If <u>exp</u> is not an equation, it is assumed to

be an expression to be set equal to zero. Var may be a function (e.g. F(X)), or other non-atomic expression except a sum or product. It may be omitted if exp contains only one variable. Exp may be a rational function, and may contain trigonometric functions, exponentials, etc.
SOLVE([eq1,...,eqn],[v1,...,vn]) solves a system of linear algebraic equations. It takes two lists as arguments. The first list (eqi, i=1,...,n) represents the equations to be solved; the second list is a list of the unknowns to be determined. If the total number of variables in the equations is equal to the number of equations, the second argument-list may be omitted. If the given equations are not compatible, the message INCONSISTENT will be displayed. If no unique solution exists, SINGULAR will be displayed. If the equations are not linear in the variables of interest, a reduced system of polynomials is returned. These may be processed further to find sets of solutions by calling SOLVE or in some cases - REALROOTS (section 5.6). The solutions are exact, assuming the user has not used floating-point numbers in his input, and may involve symbolic variables. The solution set consists of a list of numbered equations and an index to the list. If GLOBALSOLVE [FALSE] is set to TRUE then variables which are SOLVEd for will be set to the solution of the equation if it is unique. The success of SOLVE may depend partly on the setting of the following switches.
SOLVEFACTORS [TRUE] - if FALSE then SOLVE will not try to factor the expression. This may be desired in some cases where factoring is not necessary.
SOLVERADCAN [FALSE]- if TRUE then SOLVE will use RADCAN (see section 5.6) which will make SOLVE slower but will allow certain exponential problems to be solved.

SUM(exp,ind,lo,hi) performs a summation of the values of exp as the index ind varies from lo to hi. If the summation cannot be performed, or if 'SUM is used, the value is a sum noun form which is a representation of the sigma notation used in mathematics. CAUCHYSUM [FALSE] when TRUE causes the Cauchy product to be used when multiplying sums together rather than the usual product. In the Cauchy product the index of the inner summation is a function of the index of the outer one rather than varying independently.

```
(C4) SUM(1/I,I,1,4)*SUM(F(I*I),I,1,N)@
                              N
                             ====
                             \        25 F(I^2)
(D4)                          >       ---------
                             /           12
                             ====
                             I = 1
```

Sums may be differentiated, added, subtracted, or multiplied with some automatic simplification being performed.

PRODUCT(exp,ind,lo,hi) performs a product of the values of exp as the index ind varies from lo to hi.

POWERSERIES(exp,var,pt) attempts to generate the general form of the power series expansion for exp in the variable var about the point pt (which may be INF for infinity).

TAYLOR(exp,var,pt,pow) expands the expression exp in a truncated Taylor series in the variable var around the point pt. The terms through (var-pt)**pow are generated.

DEFTAYLOR(function(var),exp) defines the Taylor series of function with respect to variable var to be the expression exp.

```
(C1) DEFTAYLOR(F(X),SUM(X**I/(I**2),I,1,INF))$
```

5.2 Manipulation

EXPAND(exp) will cause an expansion of the argument. The MACSYMA variables MAXNEGEX and MAXPOSEX (originally set to 6) control the maximum negative and positive exponents, respectively, which will expand. EXPAND(exp,p,n) expands exp, using p for MAXPOSEX and n for MAXNEGEX.
It is faster to use the command RATSIMP to expand, if exp is a polynomial in one variable. [see section 5.6]

```
(C24) PRODUCT((X+I*(I+1)/2),I,1,4)@
(D24)                    (X + 1) (X + 3) (X + 6) (X + 10)

(C25) %,EXPAND@
                           4        3         2
(D25)                     X  + 20 X  + 127 X  + 288 X + 180
```

MULTTHRU(expr) expr must be a product containing one sum. Each term in that sum is multiplied by the other factors in the product. MULTTHRU(exp1,exp2) multiplies each term in exp2 (which must be a sum) by exp1.

```
(C8) X/(X-Y)**2-1/(X-Y)-F(X)/(X-Y)**3@
                                       1          X           F(X)
(D8)                               - ----- + ---------- - ----------
                                     X - Y           2            3
                                              (X - Y)      (X - Y)

(C9) MULTTHRU((X-Y)**3,%)@
                                             2
(D9)                               - (X - Y)  + X (X - Y) - F(X)
```

5.2.1 The Evaluation Command

EV(exp,arg1,...,argn) causes the expression exp to be evaluated and simplified with switches set according to the values of the argi. The switches are as follows:

EVAL reevaluates the expression so that variables in it which have values will be evaluated.

SIMP causes the expression to be simplified regardless of the setting of the switch SIMP which inhibits simplification if FALSE.

EXPAND causes expansion. EXPAND(n,m) set the values of MAXPOSEX and MAXNEGEX.

DIFF causes all differentiations indicated to be performed. DIFF(var1,...,vark) causes only differentiations with respect to the indicated variables.

NUMER causes some mathematical functions (including exponentiation) with numerical arguments to be evaluated. (see section 4.0).

PRED causes predicates (expressions which evaluate to TRUE or FALSE) to be evaluated.

FLOAT causes rational numbers to be converted to floating point.
(The NUMER flag causes this to take place also).

RATSIMP causes the expression to be rationally simplified. (see section 5.6)

v=exp causes the substitution of exp for v.

In addition a list of equations may be given or a list of names of equations such as that returned by SOLVE.

Any other function names (e.g. SUM) cause evaluation of occurrences of those names as though they were verbs.

The arguments following the first (exp) may be given in any order. The switches may also be set locally in a user defined function or globally at the "top level" in MACSYMA so that they will remain in effect until being reset.

```
(C4) SIN(X)+COS(Y)+(W+1)**2+'DIFF(SIN(W),W)@

                           D                   2
(D4)  COS(Y) + SIN(X) + ——SIN(W) + (W + 1)
                          DW

(C5) EV(%,NUMER,EXPAND,DIFF,X=2,Y=1)@

                    2
(D5)       COS(W) + W  + 2 W + 1.425324
```

An alternate "top level" syntax has been provided for EV, whereby one may just type in its arguments, without the EV(). That is, one may write simply exp,arg1,...,argn. (This is not permitted as part of another expression, i.e. in Functions, Blocks, etc.)

```
(C2) 2*X-Y=3$

(C3) -3*X+2*Y=-4$

(C5) SOLVE([D2,D3])@
SOLUTION
```

```
(E5)                                          Y = 1

(E6)                                          X = 2
(D6)                                        [E5, E6]

(C7) D2,D6@
(D7)                                          3 = 3

(C15) X+1/X >GAMMA(1/2)@
                                      1
(D15)                             X + - > SQRT(%PI)
                                      X

(C16) %,NUMER,X=1/2@
(D16)                                   2.5 > 1.772454

(C17) %,PRED@
(D17)                                       TRUE
```

MACSYMA Special Variables Affecting General Simplification

%EMODE [FALSE] - when TRUE some exponentials involving complex numbers (like %E**(%PI*%I)) are simplified, otherwise (the default) they are not.

LOGSIMPOFF [FALSE] - if TRUE then simplification of forms like %E**LOG(X) is inhibited.

EXPON [0] - the lowest negative exponent which is automatically expanded (independent of calls to EXPAND).

EXPOP [0] - the highest positive exponent which is automatically expanded. Thus (X+1)**3 when typed will be automatically expanded only if EXPOP is greater than or equal to 3. If it is desired to have (X+1)**N expanded where N is greater than EXPOP then executing EXPAND((X+1)**N) will work only if N is less than or equal to MAXPOSEX.

FACTLIM [-1] the highest factorial to be evaluated automatically. If -1 all factorials are evaluated.

NONCOMEXPT [FALSE] - if TRUE permits immediately adjacent common factors of the non-commuatative product operator to be combined as in collecting exponenets.

NONCOMNONASSOC [FALSE] - if TRUE then the non-commutative product operator is assumed to be non-assoociative otherwise (the default) it is assumed to be associative.

TRIGSIGN [TRUE] - if TRUE permits simplification of negative arguments to trigonometric functions. That is, SIN(-X) will become -SIN(X) only if TRIGSIGN is TRUE.

5.3 Part Selection and Substitution

COEFF(exp,var,n) obtains the coefficient of var**n in exp. For best results, exp should be expanded. n must be an integer or a rational number and may be omitted if it is 1. Coefficients of var**n which are functions of var are ignored.

```
(C2) COEFF(Y+X*%E**X+1,X,0)@
(D2)                             Y + 1
```

HIPOW(poly,var) the highest exponent of the variable var in the polynomial poly which should be fully expanded.

LOPOW(poly,var) the lowest exponent of the variable var in the polynomial poly.

FIRSTINSUM(exp) yields the first term of exp if it is a sum otherwise yields exp.

RESTINSUM(exp) yields all of the terms of exp except the first if it is a sum otherwise yields FALSE.

SUBST(a,b,c) substitutes a for b in c. b must be an atom or a complete subexpression. For example, X+Y+Z is a complete subexpression of 2*(X+Y+Z)/W while X+Y is not. When b does not have these characteristics, one may sometimes use SUBSTPART or RATSUBST (see section 5.6). SUBST(eq1,exp) or SUBST([eq1,...,eqk],exp) are other permissible forms. The eqi are equations indicating substitutions to be made. For each equation, the right side will be substituted for the left in the expression exp . Although it is possible to achieve the same effect using the EV command, SUBST is faster since EV is more general.

```
(C1) SUBST(A,X+Y,X+(X+Y)**2+Y)@
                                   2
(D1)                      Y + X + A
```

FIRST(exp) yields the first part of exp which may result in the first element of a list, the first row of a matrix, the first term of a sum, etc.

REST(exp,n) yields exp with its first n elements removed. If n is one it may be omitted. exp may be a list, matrix, or other expression.

LAST(exp) yields the last part (term, row, element, etc.) of the exp.

DELETE(exp1,exp2) removes all occurrences of exp1 from exp2. exp1 may be any part or term which occurs in exp2.

LENGTH(exp) gives the number of elements in a list, the number of rows of a matrix, the number of terms in a sum, etc.

IMAGPART(exp) the imaginary part of the expression exp.

REALPART(exp) the real part of exp. IMAGPART and REALPART will work on expressions involving trigonometic and hyperbolic functions, as well as SQRT, LOG, and exponentiation.

LHS(eqn) the left side of the equation eqn.

RHS(eqn) the right side of the equation eqn.

NUMFACTOR(exp) the numerical factor multiplying the expression exp.

5.3.1 The Part Commands

The part commands make it possible to reference or replace any part of any MACSYMA expression. A part is referred to by a set of indices these always being non-negative integers. For example, in exponentiation the base is considered part 1 and the exponent part 2. In a quotient the numerator is part 1 and the denominator part 2. In a sum or product the ith term or factor is part i and in any expression the main operator is part 0. Note that unary minus is considered an operator.

Oftentimes an expression is reordered after being typed in so it is important to note that part selection applies to an expression as it would be displayed when typed by MACSYMA. Thus the first part of X+Y is Y not X because the expression displays as Y+X.

The command PART(exp,n1,...,nk) obtains the part of exp as specified by the indices n1,...,nk. First part n1 of exp is obtained, then part n2 of that, etc. The result is part nk of ... part n2 of part n1 of exp. Thus PART(Z+2*Y,2,1) yields 2. Part(X+Y,0) yields +, however in order to refer to the operator it must be enclosed in ?'s. For example ...IF PART(D9,0)=?+? THEN ... PART can be used to obtain an element of a list, a row of a matrix, etc.

```
(C1)  X+Y/Z**2@
                          Y
(D1)                     --- + X
                          2
                         Z

(C2) PART(D1,1,2,2)@

(D2)                      2
```

```
(C3) 'INTEGRATE(F(X),X,A,B)@
                                                   B
                                                  /
                                                  [
(D3)                                              I F(X)DX
                                                  ]
                                                  /
                                                  A


(C4) PART(%,1)@
(D4)                              F(X)
```

DPART(<u>exp</u>,<u>n1</u>,...,<u>nk</u>) selects the same subexpression as PART, but instead of just returning that subexpression as its value, it returns the whole expression with the selected subexpression displayed inside a box. PIECE holds the value of the last expression selected using PART or DPART. If PARTSTWITCH [FALSE] is set to TRUE then END is returned when a selected part of an expression (using PART, DPART, or SUBSTPART) falls of the end otherwise an error message is given.

Continuing with the above example:

```
(C2) DPART(D1,1,2,1)@
                              Y
(D2)                         ---- + X
                               2
                            *****
                            * Z *
                            *****


(C3) PIECE@
(D3)                          Z
```

SUBSTPART(<u>x</u>,<u>exp</u>,<u>n1</u>,...,<u>nk</u>) substitutes <u>x</u> for the subexpression picked out by the rest of the arguments as in PART. It returns the new value of <u>exp</u>.

5.4 Graphing

PLOT(<u>exp</u>,<u>var</u>,<u>low</u>,<u>high</u>) produces an asterisk-plot of the expression <u>exp</u> as <u>var</u> (the independent variable) ranges from <u>low</u> to <u>high</u>. An optional fifth argument of INTEGER causes PLOT to choose only integer values for <u>var</u> in the given domain. There are several other options available which are described in section 8.0.

GRAPH(<u>xlist</u>,<u>ylist</u>,<u>xlabel</u>,<u>ylabel</u>) graphs the two lists of data points, and labels the axes as indicated or omits labels if just the first two arguments are given. The variables SCOPEHEIGHT and LINEL affect the height and width of the plot. More detail is given in section 8.0.

5.5 List Handling and LISP-like functions

APPEND(<u>list1</u>,<u>list2</u>) appends <u>list1</u> and <u>list2</u> and returns a single list of the elements of <u>list1</u> followed by the elements of <u>list2</u>.

CONS(<u>exp</u>,<u>list</u>) returns a new list constructed of the element <u>exp</u> as its first element, followed by the elements of <u>list</u>.

ENDCONS(<u>exp</u>,<u>list</u>) makes <u>exp</u> the last element of <u>list</u>.

MEMBER(<u>exp</u>,<u>list</u>) returns TRUE if <u>exp</u> occurs as a member (not within a member) of <u>list</u> otherwise FALSE.

REVERSE(<u>list</u>) reverses the members of <u>list</u> (not the members themselves if they happen to be lists).

NULL(<u>list</u>) returns TRUE if <u>list</u> is empty else FALSE.

APPLY(<u>function</u>,<u>list</u>) gives the result of applying the <u>function</u> to the <u>list</u> of its arguments. This is useful when it is desired to compute the arguments to a function before applying that function. For example, if L is the list [1,5,-10.2,4,3], then APPLY(MIN,L) gives -10.2. Note that MIN(L) is unacceptable because (1) MIN does not work on lists and (2) MIN must be given at least two arguments.

MAP(<u>fn</u>,<u>list</u>) yields a list each member of which is the result of applying the function <u>fn</u> to the corresponding member of <u>list</u>. <u>fn</u> is the name of a function or is of the form LAMBDA([x],<u>defn</u>) where [x] is the dummy variable to be used in the function <u>defn</u> and which will take on the value of successive elements of <u>list</u>. For example MAP(LAMBDA([Y],Y+1),[2,10,1]) yields [3,11,2]. MAP(<u>fn</u>,<u>exp</u>) is also acceptable. In this case <u>fn</u> is applied to each part of <u>exp</u> (term of a sum, row of a matrix, etc.) For example, MAP(RATSIMP,(X**2+2*x+1)/(x+1)+(x-1)/(x**2-1)) yields 1/(X+1)+X+1.

<u>Examples</u>

```
(C34) UNION(X,Y):=(IF NULL(X) THEN Y ELSE
            IF MEMBER(T:FIRST(X),Y) THEN UNION(REST(X),Y)
            ELSE UNION(REST(X),CONS(T,Y)))$

(C35) UNION([A,B,1,1/2,X**2],[-X**2,A,Y,1/2])@
                      2              2          1
(D35)               [X , 1, B,  - X , A, Y, -]
                                                2
```

5.6 Rational Function Commands

A rational function is the quotient of two polynomials. MACSYMA provides a special internal representation (called CRE forcanonical rational expression form) for rational functions (and polynomials as special cases) which requires less storage than the general representation. The difference is generally invisible to the user except that CRE manipulations are frequently faster. Therefore it is advisable to use these whenever the problem of interest can be expressed largely in terms of polynomials or rational functions. For a more detailed description of some of the rational function commands see section 9.0.

RATVARS(var1,...,varn) provides a method for specifying the ordering of variables in CRE form. The most main variable will be varn, the least ("most constant") will be var1. The command PRINVARLIST() may be used to print the current variable ordering.

RAT(exp,v1,...) converts the exp to CRE form by combining all terms over a common denominator and cancelling out the greatest common divisor of the numerator and denominator as well as converting floating point numbers to rational numbers. The variables are ordered according to the v1,... if these are specified. RATPRINT [TRUE] if FALSE supresses the printout of the conversion message. RATEPSILON [2.0E-8] is the value of the acceptable error in converting the numbers.

RATSIMP(exp,v1,v2,...) does much the same as RAT, but converts the expression back to general form. This allows some further simplifications to take place which are not apparent in CRE form. For example, those involving nested functions, and roots of numbers or polynomials. RAT on the other hand, does not generally deal with nested functions other than + , * , / , - , and exponentiation to an integer power. RAT's answer will be a ratio of polynomials, therefore it does not work on equations, although RATSIMP does.

RADCAN(exp) simplifies the rational expression exp which may contain logs, exponentials, and radicals using a canoncial form described in [2].

RATDISREP(x) converts a CRE x to a normal prefix expression, i.e. the usual general representation.

DIVIDE(x,y,var) computes the quotient and remainder of x divided by y, as rational functions in a main polynomial variable, var. The result is a list whose first element is the quotient and whose second element is the remainder. var may be omitted in which case the first variable which occurs in y will be used.

QUOTIENT(x,y,var) computes the quotient of the two polynomials x and y with main variable var.

REMAINDER(x,y,var) computes the remainder of the polynomial x divided by the polynomial y with main variable var.

GCD(x,y) computes the greatest common divisor of x and y. GCDOFF [FALSE] if TRUE, MACSYMA takes all gcds to be 1. GCDSWITCH [TRUE] if FALSE then GCD uses the Collins reduced prs algorithm which may work on some cases where the faster modular algorithm fails (which is the default method).

FACTOR(x) factors a polynomial or rational function x (numerator and denominator). FACTORFLAG [TRUE] if FALSE supresses the factoring of integers in polynomials and rational functions, DONTFACTOR [FALSE] may be set to a list of variables with respect to which factoring is not to occur when appearing in polynomials (if FALSE then there are no such variables). FACTOR saves factors explicitly present in x if SAVEFACTORS [TRUE] is TRUE. These are then used as trial factors in a heuristic phase of FACTOR. BERLEFACT [TRUE] if FALSE will cause the Kronecker factoring algorithm to be used rather than the usually faster BERLEKAMP algorithm. [9,10]

SQFR(x) computes a square-free factorization of the expression x. A number of special checks occasionally factor polynomials even though the factors occur singly.

GFACTOR(x) factors the polynomial x over the Gaussian integers (i. e. with SQRT(-1) = %I adjoined)

MOD(x) converts the polynomial x to a modular representation. X must be in only one variable. If MODULUS [FALSE] is set to a positive integer p, then all arithmetic in the rational function routines will be done modulo p.

PARTFRAC(x,var) expands a rational function x in partial fractions with main variable var.

RATCOEF(exp,x) picks out the coefficient of x (which may be a power, product, sum, quotient, etc.) in exp.

RATSUBST(a,b,c) substitutes a for b in c. b may be a sum, product, power, etc. RADSUBSTFLAG [FALSE] if TRUE allows RADCAN to be called by RATSUBST thus allowing substitutions like A for SQRT(X) in X+1 to yield A**2+1. RATSUBST returns the answer in CRE from if and only if c is in CRE form.

RESULTANT(x,y,var) computes the resultant of the two polynomials x and y, and eliminates the variable var. The resultant is a determinant of the coefficients of var in x and y which equals zero if and only if x and y have a non-constant factor in common.

RATDIFF(exp,var) differentiates the rational expression exp with respect to var. For rational expressions this is faster than DIFF. The result is left in CRE form.

NROOTS(poly,low,high) finds the number of realroots of the univariate polynomial poly between the limits of low and high which may be MINF and INF respectively for minus infinity to plus infinity. The method of Sturm sequences is used. [11]

REALROOTS(poly,bound) finds all of the real roots of the univariate polynomial poly within a tolerance of bound which, if less than 1, causes all integral roots to be found exactly. bound may be arbitrarily small in order to achieve any desired accuracy (if less than RATEPSILON it should be rational).

5.6.1 Generalized Rational Function Commands

Generalized rational functions are developed by taking "rational functions" as coefficients of multivariate polynomials and positive rational numbers as exponents. However, there are a number of significant extentions built into the implemented version of the generalized polynomial. A variable in a generalized polynomial can fall into one of four classes. An algebraic number (a zero of a polynomial with rational real coefficients), a truncating variable (which acts like a zero of X**N), a non-truncating variable (normal rational function variables), or a coefficient variable.

PS(exp,[var1,def1,...],,[v1,v2,...]) converts exp to generalized rational function form. The second argument is a paired list where the odd numbered members, the vari, are assumed to be atomic and are not evaluated. The even numbered arguments are the definitions of the vari and are one of the following:

1. FALSE - indicating vari is non-truncating.
2. a number - indicating vari is to be truncated to degree defi.
3. a monomial in vari indicating that vari is a truncating variable.
4. a polynomial - indicating that vari is algebraic.

The second list is a variable ordering list. If there is a variable in the list which has not been defined then it is assumed to be non-truncating. This ordering should be listed with the most important variable last.

HRAT(exp,var1,var2,...) expands exp in generalized rational form. The vari are used as in the second list in PS.

SRRAT(exp) converts exp from generalized rational form to ordinary rational form. This is equivalent to RAT(RATDISREP(exp)) but works much faster.

~~5.7 The Matrix Commands~~

5.7 The Matrix Commands

FNTERMATRIX(m,n) allows one to enter a matrix element by element with the computer asking for values for each of the m by n entries.

MATRIX (row1,...,rown) defines a rectangular matrix with the indicated rows. Each row has the form of a list of expressions, e.g. [A, X**2, Y, 0] is a list of 4 elements.

GENMATRIX(array,dim1,dim2) generates a matrix of dimension dim1 by dim2 from the array.

ADDROW(M,l) appends the row given by the list l onto the matrix M.

IDENT(n) produces an n by n identity matrix.

DIAGMATRIX(n,x) returns a diagonal matrix of size n by n with the diagonal elements all x. An identity matrix is created by DIAGMATRIX(n,1), or one may use IDENT(n).

ELEMENTX(M,i,j) gives the (i,j) element of M.

SETELMX(x,i,j,M) creates a new matrix which is identical to M except that its (i,j) element is x.

AUGCOEFMATRIX([eq1,...,eqn],[var1,...,vark]) the augmented coefficient matrix for the variables var1,...,vark of the system of linear equations eq1,...,eqn.

COEFMATRIX([eq1,...],[var1,...]) the coefficient matrix for the variables var1,... of the system of linear equations eq1,...

COL(M,i) the ith column of the matrix M.

ROW(M,i) the ith row of matrix M.

SUBMATRIX(m1,...,mn, M, n1,...,nn) creates a new matrix composed of the matrix M with rows mi deleted, and columns ni deleted.

MINOR(M,i,j) computes the i,j minor of the matrix M

TRANSPOSE(M) produces the transpose of M.

ECHELON(M) produces the echelon form of M.

RANK(M) computes the rank of the matrix M.

DETERMINANT(M) computes the determinant of M.

CHARPOLY(M,var) computes the characteristic polynomial for M with respect to var. That is, DETERMINANT(M - DIAGMATRIX(var,LENGTH(M))).

5.8 Type Testing

ATOM(exp) is TRUE if exp is atomic (i.e. a number, or name) else FALSE.

CONSTANT(exp) is TRUE if exp is a constant (i.e. composed of numbers including %PI, %E, %I or any variables bound to constants) else FALSE.

INTEGER(exp) is TRUE if exp is an integer else FALSE.

LISTP(exp) is TRUE if exp is a list else FALSE.

MATRIXP(exp) is TRUE is exp is a matrix else false.

NUMBER(exp) is TRUE if exp is an integer or a floating point number else FALSE.

RATNUM(exp) is TRUE if exp is a rational number else FALSE.

FLOATNUM(exp) is true if exp is a floating point number else FALSE.

5.9 Utility, Input-Output, and Display

%%(i) is the ith previous computation. That is, if the next expression to be computed is D(j) this is D(j-i). (see section 6.5)

DISPLAY(expr1,expr2,...) displays equations whose left side is expri, and whose right side is the value of the expression.

PRINDISPLAY(expr1,expr2,..) displays the expri one per line.

PRINT(exp1,exp2,...) evaluates and prints its arguments one after the other on a line. If expi is unbound or is preceded by an apostrophe or is enclosed in ?'s then it is printed literally. For example, PRINT(?THE VALUE OF X IS ?,X).

READ(word1,...) prints its arguments literally then reads in and evaluates one expression. For example, A:READ(?ENTER THE NUMBER OF VALUES?).

DISPFUN(f) prints the definition of the function f.

GETDEPENDS(f1,f2,...) retrieves the dependency relations for the functions f1,f2,.. as previously given to the command DEPENDENCIES. The result is a list such as [f1(x1,x2,...),f2(x1,x2,...),...].

REMFUNCTION(f1,f2,...) removes the functions f1,f2,... from MACSYMA.

REMVALUE(name1,name2,...) removes variables and matrices from the system.

REMARRAY(name1,name2,...) removes arrays and frees the storage occupied.

REMOVE(arg1,prop1,arg2,prop2,...) argi is either a single name or a list of names from which the property propi is to be removed. propi may be ARRAY, FUNCTION, VALUE, DEPENDS, ALIAS, BINDTEST.

KILL(arg1,arg2,...) eliminates its arguments from the MACSYMA system. If argi is a variable, function, or array, the designated item with all of its properties is removed from core and the storage it occupies is reclaimed. If argi=VALUES, FUNCTIONS, or ARRAYS then every item of that class is KILL'ed and if argi=ALL then every function, value, and array previously defined is KILL'ed. IF argi=HISTORY then all input, intermediate, and output lines to date (but not other named items) are eliminated. If argi=a number (n), then the last n lines are deleted. KILL removes all properties from the given argument whereas the REMOVE set of functions remove a specific property. Also they print out a list of names or FALSE if the specific argument doesn't exist whereas KILL only prints out "DONE".

ALIAS(newname1,oldname1,...) provides an alternative name for a function (user or system), variable, array, etc. Any even number of arguments may be used.

REMALIAS(name1,...) removes alternate names created by ALIAS.

WRITEFILE(device,username) opens up a file for writing. Usually device is DSK.

CLOSEFILE(filename1,filename2) closes a file opened by WRITEFILE and gives it the name filename1 filename2.

LOADFILE(fn1,fn2,device,username) loads a file as described by its arguments. If device and username are omitted then the last device and username seen (initially DSK and user's system name) will be used. fn1 fn2 must be a file of LISP functions.

BATCH(file specification) reads in and evaluates MACSYMA expressions from a file. (see section 6.0).

DEMONSTRATE(file specification) same as BATCH but pauses after each command and continues when a space is typed. (see section 6.0).

BATCON(argument) continues BATCHing in a file which was interrupted. (see section 6.0).

PLAYBACK() "plays back" all the input and output lines since (C1). PLAYBACK(n) plays back the last n expressions (Ci, Di, and Ei count as 1 each). PLAYBACK(SLOW) places PLAYBACK in a slow-mode similar to DEMONSTRATE's (as opposed to the "fast" BATCH). This is useful when creating a secondary-storage file in order to pick out useful expressions. PLAYBACK(STRING) strings-out (without $'s or @'s) all input lines when playing back rather than displaying them.

SAVE(args) saves quantities described by its arguments on disk and keeps them in core also. (see section 7.0).

STORE(args) same as SAVE but doesn't retain quantities in core. (see section 7.0).

RESTORE(file specification) reinitializes all quantities filed away by the SAVE or STORE commands. (see section 7.0).

REMFILE() removes files created by the secondary storage scheme (see section 7.0).

UNSTORE(name1,...) brings the named expressions into core. (see section 7.0).

STRING(expr) puts expr into the buffer for editing (it is usually Ci) (see section 2.0).

STRINGOUT(file description,A1,A2,..) outputs to a file given by file description ([filename1,filename2,device,username]) the values given by A1,A2,.. in a MACSYMA readable format. The file description may be omitted and default values will be used. The Ai are usually C expressions or may be ALL meaning all C expressions.

TIME(Di) gives the time in milliseconds taken to compute Di.

NOUN(name1,..) makes the named functions NOUNs.

LOGOUT() logs the user out of the ITS time sharing system [8]. This is useful when it is desired to BATCH in a file and have the terminal logged out automatically when the computations are finished.

MACSYMA Special Variables for I/O, Status, and Display

ARRAYS - a list of arrays defined thus far.

FUNCTIONS - a list of user functions defined so far.

VALUES - a list of variables (including matrices) which have values.

EXPTDISPFLAG [TRUE] - if TRUE MACSYMA displays expressions with negative exponents using quotients.

GENINDEX [I] -the alphabetic prefix of the index of summation for generated sums.

NOSTAR [TRUE] - if TRUE causes multiplication to be displayed as a space rather than an *.

SQRTFLAG [TRUE] - if FALSE MACSYMA displays SQRT as exponent 1/2.

INCHAR [C] - the alphabetic prefix of the names of expressions typed by the user.

LINECHAR [E] - the alphabetic prefix of the names of the values of intermediate displayed expressions.

OUTCHAR [D] - the alphabetic prefix of the names of outputted expressions.

IBASE [10] - the base for inputting numbers.

BASE [10] - the base for display of numbers.

LINENUM - the line number of the last expression.

LASTTIME - the time to compute the last expression in milliseconds.

TIME [FALSE] - if TRUE causes MACSYMA to print the time used after each computation.

LINEL - the length of the printed line on the terminal. Also used for plotting (see section 8.0).

SCOPEHEIGHT - the height of the area used for plotting. (see section 8.0).

5.10 Debugging Commands (see also section 11.0)

TRACE(name1,name2,...) gives a trace printout whenever the functions mentioned are referenced. (see section 11.0).

UNTRACE(name1,...) removes tracing incurred by the TRACE command. (see section 11.0). UNTRACE() removes tracing from all functions.

REMTRACE() removes the tracing facilities from MACSYMA thus freeing up some storage. They will be reloaded when TRACE is used again. (see section 11.0).

BINDTEST(var1,var2,...) causes MACSYMA to give an error message whenever any of the vari occur unbound in a computation.

DEBUGMODE(switch) causes MACSYMA to enter a special debugging mode if switch is TRUE and to exit if switch is FALSE.

BACKTRACE has as value a list of all functions currently entered.

EXIT resumes a computation interrupted by control-A or by an error break caused when DEBUGMODE(TRUE) has been executed.

Special Variables

DEBUG [FALSE] if TRUE causes a message to be printed each time a bound variable is used for the first time in a computation.

PREDERROR [FALSE] - if TRUE causes a message to be printed whenever the predicate of an IF statement fails to evaluate to either TRUE or FALSE.

5.11 Pattern Matching and Related Commands

FREEOF(x,expr) yields TRUE if x does not occur in expr and FALSE otherwise. x must be an atomic variable.

DECLARE(patternvar,predicate) associates a predicate with a pattern variable so that the variable will only match expressions for which the predicate is not FALSE. For pattern matching, predicates refer to functions which are either FALSE or not FALSE (any non FALSE value acts like TRUE). For example after DECLARE(Q,FREEOF(X)) is executed, Q will match any expression not containing X. If the match succeeds then the variable is set to the matched expression. The predicate (in this case FREEOF - a function of two arguments) is written without the last argument which should be the one for which patternvar is substituted.
DECLARE(var,TRUE) will permit var to match any expression.

DEFMATCH(progname,pattern,patvar1,...,patvarn) creates a function of n+1 arguments with the name progname which tests an expression to see if it can match a particular pattern. The pattern is some expression containing pattern variables patvar1,...,patvarn either explicitly, or implicitly in a previous DECLARE command. The first argument to the created function progname, is an expression to be matched against the "pattern" and the other n arguments are the actual variables occurring in the expression which are to take the place of dummy variables occurring in the "pattern". Thus the patvars in the DEFMATCH are like the dummy arguments to the SUBROUTINE statement in FORTRAN. When the function is "called" the actual arguments are substituted. For example:

```
(C1)  NONZERO(T):= NOT(T=0)$
(C2)  NONZEROANDFREEOF(X,E):= NONZERO(E) AND FREEOF(X,E)$
(C3)  DECLARE(A,NONZEROANDFREEOF(X))$
(C4)  DECLARE(B,FREEOF(X))$
(C5)  DEFMATCH(LINEAR,A*X+B,X)$
```

This has caused the function LINEAR(exp,var1) to be defined. It tests exp to see if it is of the form A*X+B where A and B do not contain X and A is not zero. DEFMATCHed functions return (if the match is successful) a list of equations whose left sides are the pattern variables and whose right sides are the expressions which the pattern variables matched. The pattern variables are also set to the matched expressions. If the match fails, the function returns FALSE. Thus LINEAR(3*Z+(Y+1)*Z+Y**2,Z) would return [B=Y**2 , A=Y+4 , X=Z]. Any variables not declared as pattern variables in DECLARE or in DEFMATCH which occur in "pattern" will match only themselves so that if the third argument to the DEFMATCH in (C5) had been omitted, then LINEAR would only match expressions linear in X not in any other variable.

SELECTOR(predicate) defines predicate as one which will also extract a particular part of the expression it is applied to.
Thus after executing DECLARE(A,SIGNUM) the pattern variable A would match and be set to any expression since SIGNUM is always non FALSE. However if prior to this SIGNUM had been made a selector, then A would match anything but would be set to the sign (1,0, or -1) of the matched expression.

DEFRULE(rulename,pattern,replacement) defines and names a replacement rule for the given pattern. If the rule named rulename is applied to an expression (by one of the APPLY programs below), every subexpression matching the pattern will be replaced by the replacement. All variables in the replacement which have been assigned values by the pattern match are assigned those values in the replacement which is then simplified. The rules themselves can be treated as functions which will transform an expression by one operation of the pattern match and replacement. If the pattern fails, the value of the rule is FALSE.

APPLY1(exp,rule1,...,rulen) applies the first rule to the expression until it fails, then recursively applies the same rule to the subexpressions of that expression, left-to-right, until the first rule has failed on all subexpressions. Then the second rule is applied in the same fashion. When the final rule fails on the final subexpression, the application is finished.

APPLY2(exp,rule1,...,rulen) differs from APPLY1 in that if the first rule fails on a given subexpression, then the second is applied, etc. Only if they all fail on a given subexpression is the whole set of rules applied to the next subexpression. If one of the rules succeeds, then the same subexpression is reprocessed, starting with the first rule.

TELLSIMPAFTER(pattern,replacement) defines a replacement for pattern which the MACSYMA simplifier uses after it applies the builtin simplification rules. For example:

```
(C44) SEC(%PI)@
```

```
(D44)                              SEC(%PI)
(C45) DECLARE(EXPR,TRUE)$
(C46) TELLSIMPAFTER(SEC(EXPR),1/COS(EXPR))$

(C47) SEC(%PI)@
(D47)                                - 1

(C48) SEC(X)@
                                      1
(D48)                              ------
                                   COS(X)
```

TELLSIMP(pattern,replacement) is similar to TELLSIMPAFTER but places new information before old so that it is applied before the built-in simplification rules. The pattern may not be a sum, product, single variable, or number. (This restriction does not apply to TELLSIMPAFTER).

6.0 Batch Commands

6.1 Introduction

The Batch set of commands in MACSYMA, namely BATCH, DEMONSTRATE or DEMO, and BATCON (mnemonic for BATchCONtinue), provide a facility for executing commands stored on a disk file rather than in the usual on-line mode. This facility has several uses, namely to provide a reservoir for working commands for giving error-free demonstrations or to help in organizing one's thinking in complex problem-solving situations where modifications may be done via the PDP6/10 TECO file editor.

A batch file consists of a set of MACSYMA commands, each with its terminating @ or $, which may be further separated by spaces, carriage-returns, form-feeds, and the like. The BATCH and DEMO(NSTRATE) commands have both a simple and more complicated format, which are described below.

6.2 The Simple Format

BATCH(filename1, filename2, device, username)

(The same command format holds for DEMO(NSTRATE) as well.) The arguments to BATCH (or DEMO) in this format specify the file which is to be batched, in standard ITS format. Here, each file is specified by two filenames of at most six characters each the device the file is on, usually DSK and the user file directory. E.g. DEMO(SOLVE, TEST, DSK, MACSYM) calls for "demonstrating" (see below) the file SOLVE TEST on the MACSYM disk directory. Latter arguments to the BATCH or DEMO commands may always be omitted if they are known from previous file-manipulating commands.

The BATCH command calls for reading in the commands from the file one at a time, echoing them on the user console, and executing them in turn. Control is returned to the user console only when the end of the file is met. Of course, the user may quit out of the file-processing by typing <control>G at any point. DEMONSTRATE differs from BATCH only in that it pauses after the execution of each command, waiting for the user to type a space which tells it to go on. If the user types any other character, file-processing will then terminate, giving control over to the user console. (The user may actually continue processing from the file at any time - see the BATCON command below.)

6.3 The More Complicated Format

BATCH([fn1, fn2, dev, uname], delay-switch, index-specification)

The arguments to BATCH or DEMO in this mode are as follows:

The first argument is the file specification (as above), enclosed in brackets.

The second argument, the delay-switch, may be answered by ON or OFF (the default). This switch has to do with the temporary inability of LISP, the system underlying MACSYMA, to have more than one input file open at a time. If in the course of batching in a file of commands, execution of a command forces a second file to be input, this would ordinarily cause an error. However, setting the delay-switch to ON causes the entire batch file to be read in before execution of it begins, thus aborting the error. The default for the delay-switch is OFF, as the circumstance described above is not frequent, it takes some time to read in a batch file, and one may always continue batching via the BATCON command. As soon as the inability of LISP is removed, this switch will no longer be needed.

The index-specification is given by one or two arguments, the possibilities being: (In the following, m and n are positive integers.)

(i) m. This indicates that processing is to begin with the mth command in the file. Thus, the default for the index-specification is 1.

(ii) m, n. This indicates that only the mth command through the nth command are to be processed.

(iii) a variable (say FOO). FOO must be non-numeric and neither TRUE nor FALSE. This causes file-processing to begin at FOO and continue until the end of the file. This makes it unnecessary to count commands as required by (i) above.

(iv) variable (say FOO), continue-flag. The continue-flag is either ON (the default, and unnecessary) or OFF. If OFF, this enables one to separate a batch file into subfiles by prefixing a command in the file with FOO . By using FOO as the index-specification, one may execute only that subfile which begins with FOO and ends with some other variable , or the end of file. If the continue-flag is ON, this causes mode (iv) to operate as (iii) above.

One can see that BATCH(SOLVE, TEST, DSK, MACSYM) and BATCH([SOLVE, TEST, DSK, MACSYM], OFF, 1) are equivalent.

6.4 The BATCON command

The BATCON command is used to continue or change the last BATCH or DEMO command, without it being necessary to mention again BATCH or DEMO, the file specification, or the setting of the delay-switch. Of course, if one wishes to change any of these, a new call to BATCH or DEMO is required.

The possible argument(s) to BATCON is (are) as follows:

(i) a number
(ii) number1, number2
(iii) a variable
(iv) variable, continue-flag

are all as above. The numeric arguments may involve the variable BATCOUNT which is set to the number in the file of the last expression BATCH'ed in. Thus BATCON(BATCOUNT-1) will resume BATCH'ing from the expression before the last BATCH'ed in from before.

One other mode is possible:

(v) skip-flag. The skip-flag is useful if an error has occurred while batching, or if the user wishes to interject commands from the console while in DEMO-mode and then to continue processing from the file. The skip-flag may be either TRUE or FALSE. If FALSE, this indicates that processing is to continue with the last command attempted (supposedly editted, in case of error); if TRUE, this indicates that processing is to continue with the next (untried) command in the file.

6.5 Miscellany

(1) Comments may be added to batch files at any point, and will, of course, be treated as such when batching in the file. A comment is any string beginning with /* and ending with */ as in PL/I.

(2) Any command in a batch file may begin with variable;; . If not in a subfile mode, this prefix will be treated as a comment.

(3) When using the batch commands, it is inconvenient to keep track of which Di label MACSYMA will assign to a computation; yet later commands often need to refer to an earlier computation. One way to get around this, of course, is for the user to explicitly label some of his commands. A function %% is also provided, such that %%(-i), where i is positive, refers to the result of the ith previous command. E.g., %%(-1) and the variable % both refer to the same computation.

7.0 Secondary Storage Commands

7.1 Introduction

There are two different reasons for wanting to use secondary storage while running a MACSYMA. Sometimes the user's intermediate expressions take up a lot of core, and it is impossible to complete the job if all the intermediate expressions are kept in core. In this case the user would like to have his intermediate expressions written automatically to disk, in order to free up core storage. On the other hand, some users would like to save some expressions on disk so that they can be read back into a future MACSYMA at a later time. In this case the user would like to specify certain expressions to be stored away and to name the disk file where they are to be stored. MACSYMA now offers the user two secondary storage schemes. The user may ask to have his expressions automatically filed away on disk. Or he may, by means of the SAVE and STORE commands, exercise explicit control over the storage of expressions. These latter commands give the user more power and flexibility at the expense of a greater effort. It is expected that the user whose only concern is to run a big job which would not run without using secondary storage will use the automatic storage scheme, while the user who wishes to save expressions for use in later MACSYMAs will use the SAVE and STORE commands.

7.2 Automatic Storage of Expressions

A- How to use it

To activate the automatic storage scheme the user merely sets the MACSYMA variable DSKUSE to TRUE. FROM THIS POINT ON labelled expressions will be written out periodically on disk. (A labelled expression is one which is referred to by a linelabel, e.g. D4, C7, E12.) Once an expression is written on disk it will no longer reside in core and most of the core storage taken up by it will be released. When the user attempts to reference an expression which has been stored on disk, MACSYMA will retrieve the correct value from the disk file.

B- Cleaning up the disk

The automatic storage scheme will in general cause several disk files to be created, which are of no further use after the user has finished running his current MACSYMA. There is a function of no arguments, REMFILE, which will delete all the files created by the automatic storage scheme. Thus if the user does not want these files to stay around, he should execute REMFILE()@ before leaving MACSYMA.

C- Options

The user may specify how often files are written, how large they are, and what they will be named. Or he may accept the default values for all these. The following MACSYMA variables are relevant.

FILENAME:	The value of this variable is the first name of the files which are generated by the automatic disk storage scheme. The default value is the first three characters of the user's login name concatenated with a three-digit random number (i.e. ECR684)
FILENUM:	The value of this variable, a number, is the second name of the last file written. Each time a file is written, this value is first increased by 1, so it must always be numeric. It is initially set to 0.
FILESIZE:	The value of this variable is the number of expressions written into each file.
RETAINNUM:	When the number of expressions in core reaches FILESIZE+RETAINNUM a file is written. ?.UNDENT 20 DEV: The value of this variable is the default device. It is initialized to DSK.
UNAME:	The value of this variable is the default sname. It is initialized to the user's login name, if he has a disk directory, and to MACSYM otherwise. UNAME determines to what directory disk files will be written.

7.3 Explicit storage of expressions — the SAVE and STORE commands

A - Use of the commands

The SAVE and STORE commands allow the user to explicitly state that certain expressions should be written onto disk. These commands also allow him to specify the file into which these expressions should be written. They allow the user to store away arrays, function definitions, and any other kind of value. The main purpose of these commands is to allow the user to save expressions on disk so that they can be read into future MACSYMAs.

SAVE and STORE are identical in all respects but one. When an expression is STORE'd it is both written on disk and removed from core. (When the expression is referenced, of course, the correct value is retrieved from disk.) When an expression is SAVE'd, it is written on disk but not removed from core. The only difference between these two commands is their effect on core storage.

SAVE and STORE take any number of arguments. If the first argument is a list it is assumed to be the file specification (i.e. [FN1, FN2, DSK, USER]). In accordance with the standard options for file specifications, the latter arguments may be omitted from the list and the default device and username will be assumed. If the first argument is not a list, the expressions will be written into a file with the default filename. The value of the MACSYMA variable FILENAME is the default first filename, and the value of the MACSYMA variable STORENUM is the default second filename. The value of STORENUM is decreased first by 1 each time a file is written, so its value must always be numeric. STORENUM is initially 0. The value of DEV is the default device, and the value of UNAME is the default username.

All arguments to SAVE or STORE, except possibly the first, must be one of the following:

1- VALUES When this atom is an argument, every user variable which has been assigned a value (i.e. with :) will be written to disk. This will not cause variables whose purpose is to communicate with the system (e.g. LINENUM, NONCOMMONASSOC, FILENAME) to be stored.

2- FUNCTIONS When this atom is an argument, every user function definition is written to disk.

3- ARRAYS When this atom is an argument, every array is written to disk. Writing an array to disk means writing out all of its elements as well as any function definition which may be associated with it.

4- LABELS When this atom is an argument, every line (i.e. every expression which is referred to by a linelabel) is written to disk.

5- When any other atom is an argument, it must be either an array, a function, or have a value. It gets written to disk.

6- A=B The effect is similar to the case where the argument is just B, i.e. B gets written to disk. The only difference shows up if the file is read into some future MACSYMA. In that case, the expression which is referred to as "B" in the present MACSYMA will be referred to as "A" in the future MACSYMA. For example, suppose I wish to save some expression, say D7, for use in a future MACSYMA. I can execute STORE([FN1, FN2, DSK, ECR], YESTERDAYSD7 = D7)@. D7 is now stored on disk. When I come back the following day and load in a fresh MACSYMA I merely execute LOADFILE(FN1, FN2, DSK, ECR)@ and the variable YESTERDAYSD7 will take on the value which D7 had yesterday. This renaming however has no effect on the present MACSYMA, where D7 must still be referred to as "D7".

A command like STORE(LABELS)@ could cause difficulty if the stored expressions are read into a future MACSYMA, because expressions referred to by linelabels will be lost when that linelabel appears again in the future MACSYMA. Therefore when reading such expressions into a fresh MACSYMA, the user should either reset LINENUM, or INCHAR and OUTCHAR. He can assure that this gets done automatically by executing a sequence such as

```
X:'F@
Y:'G@
STORE(LABELS,INCHAR=X,OUTCHAR=Y)@
```

After the resultant disk file is read into a fresh MACSYMA, INCHAR and OUTCHAR will automatically be changed into 'F and 'G respectively.

The user should note that each use of the SAVE or STORE command will cause exactly one file to be written, regardless of the number of arguments the command is given.

MACSYMA keeps lists of all the values, arrays, and functions defined by the user. These lists are the values of the MACSYMA variables VALUES, ARRAYS, and FUNCTIONS, respectively. When one of the words "VALUES", "FUNCTIONS", or "ARRAYS" is given as an argument to SAVE or STORE, every member of the corresponding list gets STORE'd.

Certain MACSYMA variables (i.e. LINENUM, FILESIZE, NONCOMMONMASSOC, etc.) are used to communicate to the MACSYMA system that certain options are in effect, or to tell the system to use certain values. These variables should not be STORE'd (though they may be SAVE'd), since the system programs will not be able to correctly retrieve their values from disk. In general, one should not attempt to STORE variables whose purpose is to provide information to the system.

B- Retrieval of expressions which have been written to disk

1- In the present MACSYMA

Expressions which are written on disk using the SAVE command also reside in core, so the notion of retrieving them from disk in the present MACSYMA is not applicable. Expressions written to disk using STORE, however, no longer reside in core. When such expressions are referenced the system will always retrieve the correct value from disk. When a STORE'd array is referenced, the array will be brought back to core. Functions and values will be read from disk correctly, but will not be returned to core. If the user wants to bring an expression back to core he may use the command UNSTORE. This command takes any number of arguments. Each argument must be an atom, and if this atom refers to an expression which is stored on disk, the expression is returned to core. Of course, when an expression is UNSTORE'd, either by the user or by the system (as happens when STORE'd arrays are accessed), a copy of the expression still remains on disk in the assigned file.

2- In future MACSYMAs

Files created by SAVE and STORE can be loaded into future MACSYMAs using the LOADFILE command. This will set up in core all those expressions which were written into the file. Some of the expressions will have different names than they had in the MACSYMA where they were created, if the renaming option (i.e. arguments of the form A=B) of the STORE or SAVE command was used.

7.4 Saving a MACSYMA Overnight

Often a user in the middle of his work would like to save everything on disk so he can go home and resume work tomorrow. There is as yet no simple way to save the complete state of a MACSYMA. It is hoped though that a thoughtful user, by following the instructions in this section, can recreate the state of a system without too much difficulty. When the user decides to go home and to save the state of MACSYMA, he should

execute
SAVE([WENT,HOME,DSK,USER],LABELS,VALUES,FUNCTIONS,ARRAYS,LINENUM)@
This will write all his arrays, functions, values, and lines into a single disk file. If the user has been using the automatic storage scheme he should now execute REMFILE() to delete useless files from disk. When the user comes back the next day he should load a fresh MACSYMA and execute one of the following two commands:

```
        LOADFILE(WENT,HOME,DSK,USER)@
        RESTORE(WENT, HOME, DSK, USER)@
```

The former command will cause all the expressions from yesterday to be loaded into today's MACSYMA. The latter command has the same effect, but executing a RESTORE command causes the expressions not to reside in core. (Of course, if RESTORE is used and the expressions therefore remain solely on disk, MACSYMA will retrieve the correct value from disk when the expressions are referenced.) Please note that whereas LOADFILE is a general command which can be used to load many differenct kinds of files, RESTORE may only be used on files which were created by the SAVE or STORE commands (also on files which were created by the automatic storage scheme). Its sole purpose is to avoid a drain on core storage when a user wants to restore the state of a MACSYMA.

This sequence is not complete because it will not save the properties of the user's variables (i.e. GRAD,DEPENDS), nor will it save any ALIAS information. Thus if the user wants to save the complete state of his MACSYMA he will have to prepare a file himself in order to reinitialize these properties in a fresh MACSYMA. The user can prepare a file of the appropriate MACSYMA commands which he can read into a fresh MACSYMA using the BATCH command.

8.0 Plotting Commands

The MACSYMA commands PLOT and GRAPH produce character plots of specified functions and sets of data points. They can also be used to produce output files for plotting on the A.I. Calcomp plotter. The format of these commands and the variables used by the corresponding routines are described below:

VARIABLES:

LINEL - width of graphing area in terms of the number of characters

default values:	68 for DATAPOINT
	88 for IMLAC
	79 for hard copy devices

SCOPEHEIGHT - height of graph in terms of number of characters

default values:	24 for DATAPOINT and hard copy devices
	38 for IMLAC

CALCOMP - if set to TRUE will cause the output of a file for use on the Calcomp; file is output on DEV;UNAME:FILENAME FILENUM+1, the variables of which can be set by the user.
(see section 7.2 for explanation of FILENUM).

AXES - if set to TRUE will cause the X=0 and Y=0 axes to be displayed.

FORMATS FOR PLOT:

PLOT (F(x),x,low,high)
Plots the expression F(x) over the domain low < x <high.

PLOT (F(x),x,low,high,INTEGER)
As above, but plots F(x) only for integer values of x.

PLOT (F(x),x,[x1,x2,x3,...,xn])
Plots the function F(x) for the values x1,x2,x3,...,xn.

PLOT (F(x1,x2,x3,...,xn))
The user is asked to define the independent variable and to set the other variables to constants. He is also asked to provide the domain for the independent variable. A plot is produced as for the other formats.

PLOT ([y1,y2,y3,...,yn])
The user is asked to provide a matching list of values for the independent variable and a graph is made of the two sets of data points.

FORMATS FOR GRAPH:

GRAPH ([x1,x2,x3,...,xn],[y1,y2,y3,...,yn],xlabel,ylabel)
Graphs the two sets of data points with the specified labels. The labels may be omitted.

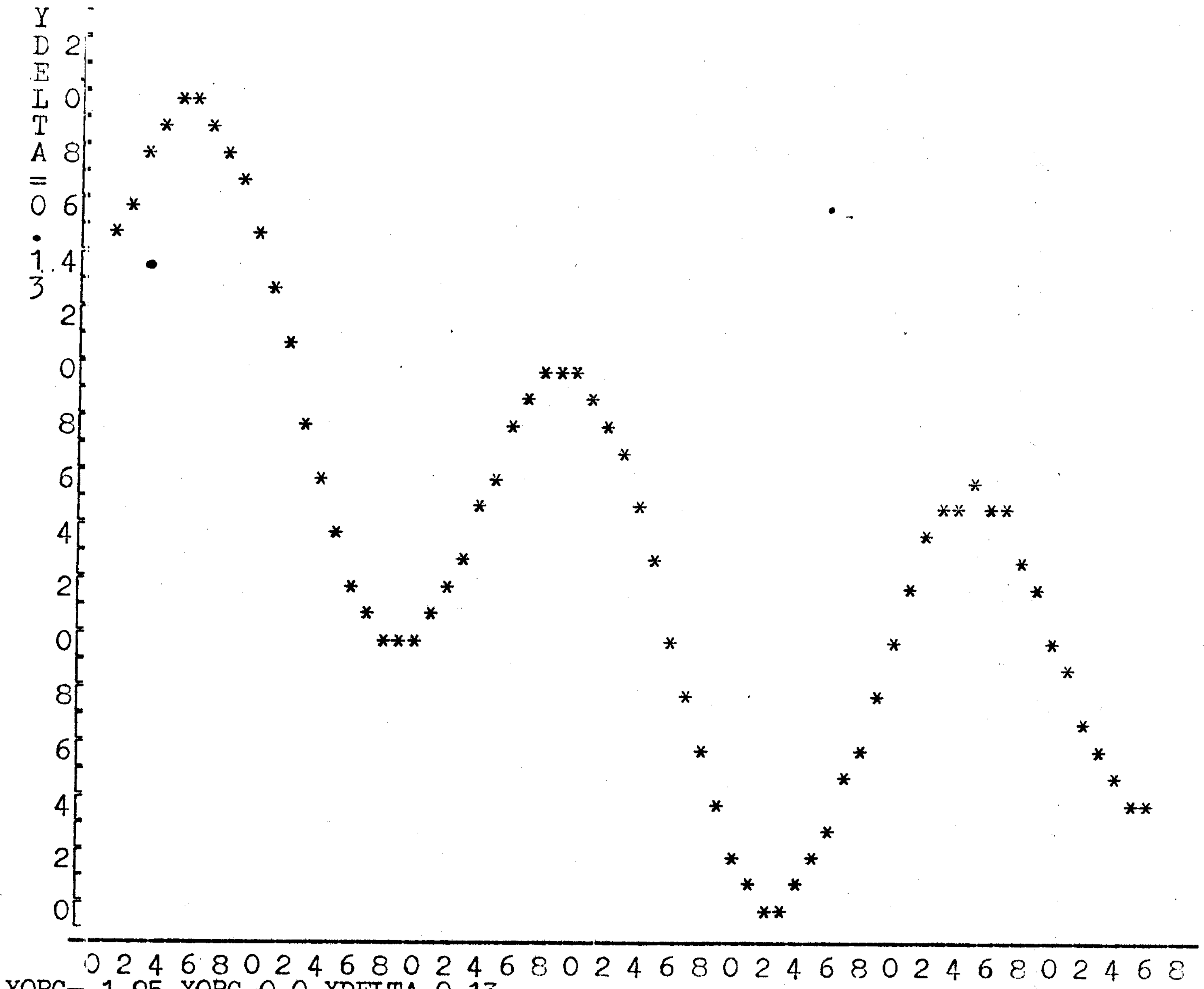
GRAPH ([[x1,y1],[x2,y2],[x3,y3],...,[xn,yn]],xlabel,ylabel)
Graphs the points specified by the list of coordinate pairs. Again, the labels may be omitted.

The graphs produced by the above functions is a character plot on a coordinate system defined by axes along the minimum x and y values of the plot. The x and y coordinates are independently scaled to optimally use the specified graphing area. The origin of the graph (left-hand corner) is given by the values of xorg and yorg; the computed increments (= one character) are given by the values of xdelta and ydelta. The axes are labeled with the number sequence 0,2,4,6,8,0,2,4,... as an aid in counting the number of increments from the origin.

When a graph is completed, the user must type a single character, such as space or carriage return, to return control to MACSYMA. On a display-type console this causes the screen to be cleared for further MACSYMA commands.

Examples

```
(C3) PLOT(SIN(2*X)+COS(.5*X),X,0,10)@
```

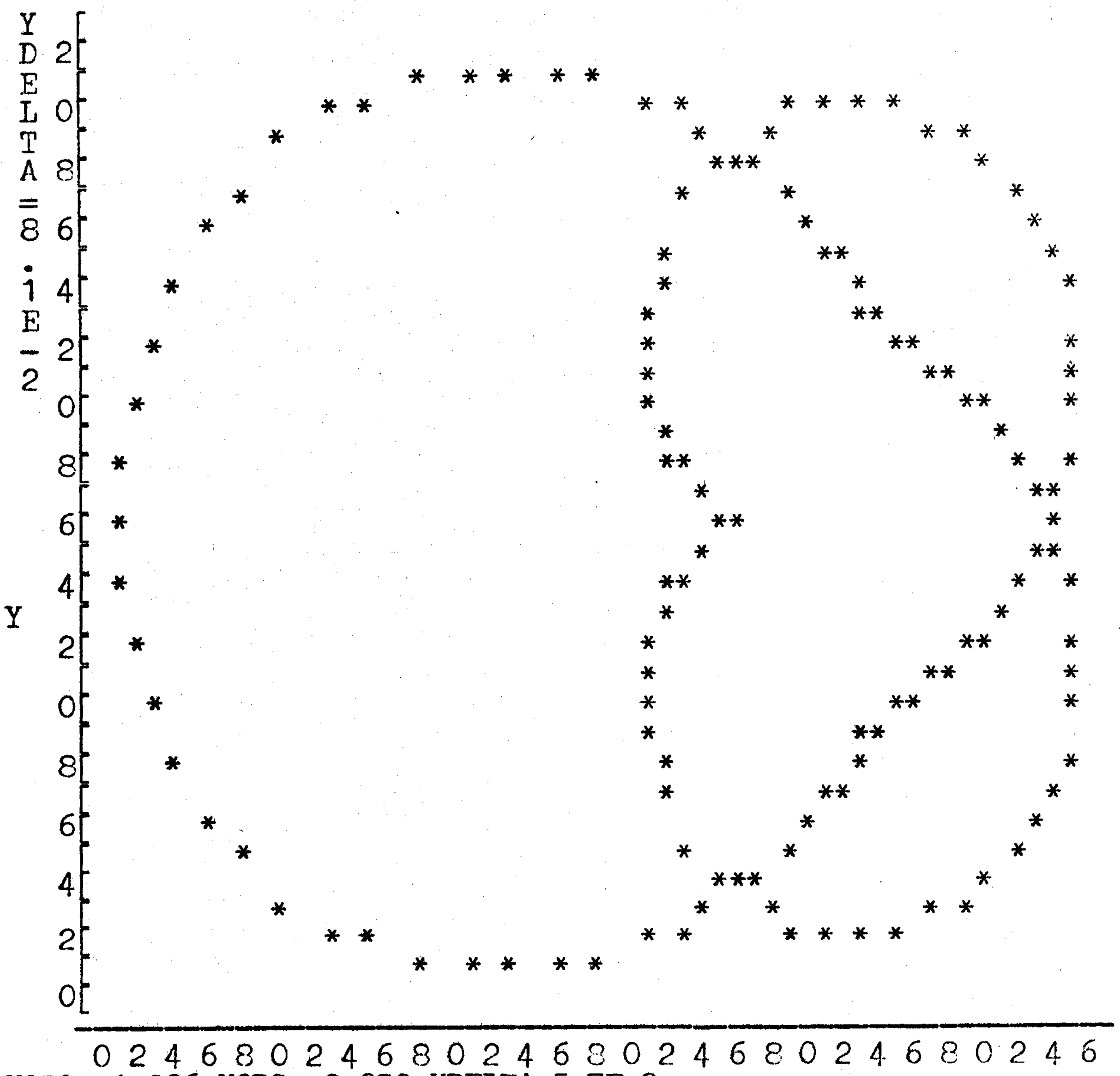


```
YORG=-1.95 XORG=0.0 XDELTA=0.13

(C4) POLARPLOT(RHO, NUMBREV) := BLOCK([THETA, P, LIMIT], NUMER : TRUE,
     RATPRINT : FALSE, THETA : 0.0, X : [], Y : [], LIMIT : 72*NUMBREV,
     FOR I : 1 STEP 1  THRU LIMIT DO [P : RHO(THETA), X:CONS(P*COS(THETA),X),
     Y : CONS(P*SIN(THETA), Y), THETA : THETA + %PI/36.0], GRAPH(X, Y, X, Y))
(C5) F(T):=1+SIN(.5*T)*COS(T)$
(C6) SCOPEHEIGHT@
(D6)
```

```
(C7) LINEL@
(D7)                                   88
(C8) LINEL:63$

(C9) POLARPLOT(F,2)@
```



```
YORG=-1.296 XORG=-2.052 XDELTA=5.7E-2
                    X

(D12)                         TRUE
```

9.0 Rational Functions

9.1 Basic Commands

In order to clarify the discussion, it is necessary to distinguish between the two major internal forms for expressions in MACSYMA. Ordinary MACSYMA form is a delimiter prefix form which is typical of many list-processing implementations of algebraic manipulation systems. For example, 3x**2 would be represented (glossing over inessential details) as (times 3 (expt x 2)), and x+y as (plus x y). By contrast, the canonical rational expression (CRE) form in MACSYMA is an internal form especially suitable for rapid manipulation of sparse polynomials and rational functions. In CRE form, 3x**2 is represented, (again, glossing over details) as (x 2 3). The first element of the list is the variable, the second is its highest exponent, and the third, the coefficient of the just preceeding exponent. Thus 6x**2 + 4 is represented as (x 2 6 0 4), and, allowing coefficients themselves to be polynomials, y*x**2 + 7*x*z is (x 2 (y 1 1) 1 (z 1 7)). Since (y 1 (x 2 2) 0 (x 1 (z 1 7))) is an equivalent CRE representation, it should be clear that the ordering of variables must be specified to insure that equivalent CRE's are identical, that is, they are in canonical form.

CRE's in general represent rational expressions, that is, ratios of polynomials, where the numerator and denominator have no common factors, and the denominator is positive. Thus a CRE has three essential parts: a variable list (VARLIST), specifying the ordering of the variables, and two polynomial parts.

With these preliminaries, we can describe the actions of the rational function commands.

RATVARS(a,b,...) orders the variables listed in its argument list on a global variable list (VARLIST) so that the rightmost element of the list a,b,... will be the main variable of future rational expressions in which it occurs, and the other variables will follow in sequence. If a variable is missing from the RATVARS list, it will be given lower priority than the leftmost element. The arguments to RATVARS can be either variables or non-rational functions (e.g. SIN(X)).

RATSIMP(EXP) rationally simplifies the expression EXP. That is, EXP is converted into a single fraction, whose numerator and denominator are polynomials over the integers, with no common factors. EXP is written in a recursive form: a polynomial in the main variable whose coefficients are polynomials in the next-higher-order variable, ..., whose coefficients are integers. This is accomplished by converting EXP into CRE, and then converting back to ordinary MACSYMA form for display.

For example:

```
(C1) (X**2-Y**2)*(Z**2+2*Z)/((X+Y)*W)@

                 2    2     2
             (X  - Y ) (Z  + 2 Z)
(D1)         --------------------
                  W (Y + X)

(C2) RATSIMP(D1)@
                     2
            (X - Y) Z  + (2 X - 2 Y) Z
(D2)        --------------------------
                        W


(C3) RATVARS(X)$

(C4) RATSIMP(D1)@
               2               2
            X (Z  + 2 Z) - Y Z  - 2 Y Z
(D4)        ---------------------------
                         W
```

FACTOR(EXP) factors the expression EXP into factors irreducible over the integers. If EXP is a rational expression (with a denominator not 1) both numerator and denominator are factored. If FACTORFLAG is set to TRUE, the integer multiplier, if any, is factored also. The algorithm can be used to factor polynomials in any number of variables; however, factorization with respect to some of the variables can be avoided by setting the global variable DONTFACTOR to a list of such variables.

For example,

```
(C5) FACTOR(X**6+1)@




                  2        4    2
(D5)          (X  + 1) (X  - X  + 1)
```

SQFR(EXP) is similar to FACTOR except that the polynomial factors are "square-free" that is, have no multiple roots. This algorithm, which is also used by the first stage of FACTOR, utilizes the fact that a polynomial has in common with its nth derivative all its factors of degree > n. Thus by taking derivatives with respect to each variable in the polynomial, all factors of degree > 1 can be found.

PARTFRAC(EXP,VAR) expands the expression EXP in partial fractions with respect to the main variable, VAR. The algorithm employed is based on the fact that the denominators of the partial fraction expansion (the factors of the original denominator) are relatively prime. The numerators can be written as linear combinations of denominators, and the expansion falls out.

```
(C6) PARTFRAC(X/(X**2-1),X)@ .
```

(D6) $$\frac{1}{2\,X-2}+\frac{1}{2\,X+2}$$

9.2. Contagious CRE Commands

The commands in this and the following sections represent significant departures from the usual use of rational function routines.

RAT(EXP) is indistinguishable on command level from RATSIMP; however, RAT leaves its internal result in rational function (CRE) form, so that operations used by the rational function commands described here can be more rapidly performed on it. Furthermore, any time the user adds to or multiplies by a CRE, the result is a CRE. That is, the CRE form is "contagious." This enables a user to easily force his entire calculation to be done in CRE form by converting one of his inputs into CRE by simply multiplying by RAT(1). Some problems require excessive amounts of storage and/or time if intermediate results are converted back into prefix form at each step of the calculation. The RAT facility, by being integrated into the simplifier, permits a user to compose a program and try it out (without any changes) on ordinary prefix form arguments or on CRE arguments. In this manner it is simple to compare the timing of "general" versus CRE methods on the same task. This very often demonstrates that CRE methods, when appropriate, are much faster.

RATDISREP(EXP), which appears to do nothing on the command level, changes its argument from rational function form (CRE) to ordinary MACSYMA form. This is sometimes necessary in order to use some of the other MACSYMA commands. If RATDISREP is not given a CRE for an argument, it does nothing.

9.3. The Rational Coefficent Program

RATCOEF(EXP,PART) returns the coefficient, C, of the expression PART in the expression EXP. C will be free (except possibly in a non-rational sense) of the variables in PART. If no coefficient of this type exists, zero will be returned. RATCOEF will give reasonable answers to reasonable requests, and will often produce reasonable answers to poorly stated requests. Generally, when PART includes a "+" or a "/", results may seem odd. (see lines D7, D8, D10, and D11 in the examples to follow). Since EXP is rationally simplified before it is examined, coefficients may not appear quite the way they were envisioned. The effect of RATCOEF should be clarified by the following examples.

```
(C1) S:A*B*X**2+B*X+2*X+5@
```

```
                         2
(D1)          A B X  + B X + 2 X + 5

(C2) RATCOEF(S,X)

(D2)                B + 2

(C3) RATCOEF(S,A*B)@
                          2
(D3)                     X

(C4) RATCOEF(S,B)@
                         2
(D4)                A X  + X

(C5) RATCOEF(S,2*X)@
                        B + 2
(D5)                    -----
                          2


(C8) RATCOEF(3*A+2*B,A+B)@
(D8)                      2

(C9) RATCOEF(S,-A)@
                           2
(D9)                   - B X

(C11) RATCOEF(3*A/D+A/D**2, A/D**2)@
(D11)                      0

(C12) RATCOEF(Z**2*X**2+(Y+Z)*X+A,(Y+Z)*X)@
(D12)                      1

(C13) RATCOEF(X**2*Z**2+X*Z+X*Y+A,X*Z+X*Y)@
(D13)                      0
```

The last two examples illustrate both the ability of the user to ask for coefficients of sums, and the ability of RATCOEF to sometimes answer correctly. We could have defined RATCOEF only for products, but it seems more in keeping with the spirit of an interactive system to avoid such restrictions on the user. Note that if the user were disappointed with the answer 0 to the above request, first executing RATVARS(X) would correct the situation.

In summary, RATCOEF will find the coefficient of PART when PART is a factor of the expression, or of some part of the expression such that the other factor has none of the same variables. The returned value is in CRE form.

An alternative to RATCOEF is available in situations where its flexibility is not needed. The COEFF command can operate on CRE forms or on ordinary MACSYMA forms which have been expanded. COEFF(EXP,VAR,POWER) will extract the coefficient of VAR**POWER (where POWER may be 0) from EXP. COEFF returns a CRE form if and only if it is given a CRE form.

9.4 Simple Extensions to Rational Simplification

RADCAN(EXP) converts the expression EXP into a form which is canonical over a large class of expressions and a given ordering of variables; that is, all functionally equivalent forms are mapped into a unique form. For a somewhat larger class of expressions, RADCAN produces a normal form; that is, all forms equivalent to zero are mapped into zero. For purely rational expressions, RADCAN is no more time-consuming than RATSIMP however, for more general expressions including radicals, logs, and non-integer exponents, RADCAN can be quite expensive. This is the cost of exploring certain relationships among the components of the expression for simplifications based on factoring and partial-fraction expansions of exponents.

The following examples should give a rough feel for the capabilities of RADCAN. (% always refers to the just-previously displayed expression, %E is the base of the natural logarithms):

```
(C3) (SQRT(X**2-1))/(SQRT(X-1))@
                          2
                   SQRT(X  - 1)
(D3)               ------------
                   SQRT(X - 1)

(C4) RADCAN(%)@
(D4)                SQRT(X + 1)

(C5) (LOG(A**(2*X)+2*A**X+1))/(LOG(A**X+1))@

                  2 X       X
             LOG(A    + 2 A  + 1)
(D5)         --------------------
                       X
                 LOG(A  + 1)

(C6) RADCAN(%)@
(D6)                  2

(C7) (%E**X-1)/(%E**(X/2)+1)@
                      X
                   %E  - 1
(D7)               --------
                    X/2
                  %E    + 1

(C8) RADCAN(%)@


                    X/2
(D8)              %E    - 1
```

10.0 The SOLVE Program

The SOLVE command in MACSYMA uses several techniques for solving for a given variable in an equation. Each of these techniques is open to extension in a straightforward manner. The roots and their multiplicities are available to other programs, and are used as building blocks for more complicated facilities, such as contour integration.

The format of the SOLVE command is:

SOLVE(equation, variable)@

where the equation may also be an expression (which is assumed to be set equal to zero), or a set of polynomial equations linear in some set of variables. This last case is a straightforward problem in Gaussian elimination, and will not be discussed further here.

SOLVE(E,X) puts its first argument E, in radical canonical form, and attempts to factor it with respect to the variable X, and all non-rational functions in E containing X. Each factor is examined for being linear, quadratic, cubic, or biquadratic with respect to X and the non-rational functions containing it. If the factor is of degree five or more, then it is considered unsolvable unless it is of the form a(F(X))**n - b in which case the n nth roots of b/a are generated, and the n equations F(x)-(b/a)**(1/n) = 0 are solved. Any remaining unsolved factors and their multiplicities are put on a list which is returned along with the roots.

Linear terms of the form F(X)-C are examined to see if C, the constant term, is actually free of elements containing X; if so, USOLVE is called. Otherwise the term is added to the list of unsolved factors. USOLVE knows the inverses of SIN, COS, ARCSIN, ARCCOS, TAN, ARCTAN, LOG, and powers of e. It could be extended to other functions. Once the inverse has been applied, a new equation results. It may be of the form X = FINVERSE(C), in which case the term has been solved, or it may be of the form G(X) = FINVERSE(C), in which case SOLVE is called again. This recursive algorithm allows for solution of, for example, SIN(COS(X)) = 0 for X.

The quadratic (cubic, biquadratic) formula is applied to quadratic (etc.) factors, and the same sort of recursive treatment as described above is used if the equation is, for example, quadratic in SIN(X) instead of X.

The simplification done by the quadratic (etc.) routines is of some interest, in that the roots in the formulae are simplified by a special program (SIMPNRT) which takes out perfect n*k powers of a kth root. (i.e. even powers in a square root, multiples-of-three powers in a cube root, etc.) Thus SQRT(8) is simplified to 2*SQRT(2). SIMPNRT calculates a square-free factorization of the radicand, and takes appropriate multiple factors, if any, outside the radical.

The following examples illustrate the capabilities of SOLVE:

```
(C1) SOLVE(Y**(2*X)-3*Y**X+2=0,X)@
SOLUTION

(E1)                        X = 0

                                LOG(2)
(E2)                        X = ------
                                LOG(Y)

(D2)                         (E1,E2)

(C3) A:X**2-12*X+3@
                           2
(D3)                      X  - 12 X + 3

(C4) SOLVE(SIN(A)**2-5*SIN(A)+3,X)@

SOLUTION


                                      5   SQRT(13)
(E4)    X = 6 - SQRT(ARCSIN(- - --------) + 33)
                                      2      2

                                 5   SQRT(13)
(E5)    X = SQRT(ARCSIN(- - --------) + 33) + 6
                                 2      2

                               SQRT(13)   5
(E6)    X = 6 - SQRT(ARCSIN(-------- + -) + 33)
                                  2       2

                          SQRT(13)   5
(E7)    X = SQRT(ARCSIN(-------- + -) + 33) + 6
                             2       2


(D7)             (E4,E5,E6,E7)
```

```
(C8) SOLVE(ARCSIN(COS(3*X))*(F(X)-1),X)@

SOLUTION

                                ARCCOS(0)
(E8)                        X = ---------
                                    3

THE ROOTS OF

(E9)                          F(X) = 1

(D9)                           (E8,E9)

(C10) SOLVE(5**X=125,X)@

(D10)                           X=3
```

Note that SOLVE has taken advantage of radical approaches but is still able to step back and treat fairly general expressions In order to use the "radical" polynomial factoring program, it uses RADCAN to expand unlikely-looking expressions into polynomials. Thus the expression Y**(2*X)-3Y**X+2 in C1 is expanded into a polynomial in Z, where Z=Y**X (actually Z=e**(X*LOG(Y))), which is then factored into (Z-1)*(Z-2). By setting each of these factors equal to zero, the following sequence of steps is followed:
e**(X*LOG(Y)) - 1 = 0 is converted by USOLVE to
X log(Y) = log(1) which the simplifier changes to
X log(Y) = 0.

SOLVE is called recursively, and factors this; SOLVE throws out the log(Y) factor since it does not depend on X, and the factor "X" is recognized as a linear expression of the form aX+b where a=1 and b=0, which has solution X=-b/a, or in this case, X=0. The other root is handled in an analogous fashion.

11.0 Debugging in MACSYMA

When the user's commands, especially functions and BLOCK programs, do not do what he expected or generate errors, MACSYMA offers him several debugging alternatives:

(1) He may trace any of his function calls by typing TRACE(fun1,fun2,...), where the funi are either MACSYMA or user-defined functions. Upon typing this, MACSYMA returns a list of the same length, with FALSE instead of a function name if a function does not exist. This will cause a printout of the function name and its arguments each time it is entered; and of the function name and the value it returns each time it is exited. A count which is the level of recursion is also printed. Usually, this is all the tracing power the user will need, although MACSYMA offers him the full capabilities of the LISP tracing package including conditional and breakpoint tracing. This will not be described here - for information see the A.I. Lab LISP Interim Report (A.I. Memo 190), Appendix D. MACSYMA uses trace-syntax very similar to that of LISP.
To check which functions are currently under trace, the user may type TRACE(). To remove tracing of functions use UNTRACE(fun1,fun2,...). To untrace all previous traced functions type UNTRACE(). Since the TRACE package takes up some of the user's workspace in core, when he is finished with it he should type REMTRACE(). He may always reload it at a later time.

(2) By setting the variable DEBUG to TRUE, the user will be informed when each of his variables which has a value comes up for evaluation for the first time during the course of a computation. This has a dual purpose. The user will be informed of evaluations he may not have been aware of which are the result of assignments he made long ago. It also gives him a sort of chronological trace of his computations which may be helpful in finding out where an error has occurred.

(3) By setting the variable PREDERROR to TRUE, the user will be informed of predicates of IF-THEN-ELSE statements which failed to evaluate to either TRUE or FALSE. This happens automatically in the midst of BLOCK programs and FOR statements.

(4) The user may have variables which he intends not to use purely symbolically, i.e. they are to have values all the time. By typing BINDTEST(var1,var2,...) MACSYMA will give the user an error whenever any of the vari appear in a computation unbound. To remove a BINDTEST declaration, the user may use the function REMOVE. (see section 5.8)

(5) When an error occurs in the course of a computation, MACSYMA prints out an error message and terminates the computation. At times the user may find it helpful to

investigate the environment at the place of the error. To do so he may type DEBUGMODE(TRUE) and repeat his computation. This enters a special debugging mode which will "break" or pause when an error occurs. This mode may be terminated by typing DEBUGMODE(FALSE). When an error occurs in debugging mode, (ERROR-BREAK) is printed out. MACYSMA is then waiting for the user to type something. He may type any command just as if he were at "top level". The commands will be evaluated in the environment of the error. If the user types BACKTRACE@, MACSYMA will print out a backtrace, which is a list of the function calls the user is currently in together with the arguments they were called with, ordered from most recent to earliest i.e., when reversed, this list shows a trace beginning from the initial command and ending at the last call entered including only those function calls from which the user still has not exited. (For those users who know LISP, one may type control-H and enter a similar LISP break-loop. When altmode-P space is typed, thus exiting from the LISP break-loop, the MACSYMA break-loop is re-entered.) To exit from the MACSYMA error-break and return to "top-level", type EXIT@.

The user may also enter the error-break at any point, by typing control-A. This will simply cause his computation to pause, while he investigates at will. Upon typing EXIT@, his computation will resume.

12.0 The MACSYMA Editor

12.1 Introduction

The major features of the MACSYMA editor are single (alphabetic) character commands, a varied assortment of commands (14 of them), concatenation of commands as in TECO (the PDP6/10 file editor), mnemonics for command names (once you know them, as R means "move in the Reverse direction" and not "move Right"; B means "move to the Bottom" and not "move Backwards"), and compatibility with TECO as to command names (in the case of C, D, G, I, J, K, L, R, and S).

12.2 Entering the Editor

At any time while the user is inputting a command to MACSYMA, he may enter the input-stream editor by typing #. The editor is given the string of characters typed so far in the current input command. In the case of a detected syntax error, upon typing # the entire previous command string will be given to the editor. Before typing in the next command string, one may always elect instead to edit the previous command string in this manner. One may also request the editor to edit or modify a previously accepted input command by using the STRING command in MACSYMA. Typing STRING(Ci) to MACSYMA will restore the expression labeled as Ci as the current input string. This enables the user to modify it by typing #.

All the commands to the editor reference a cursor (displayed as an underscore or back-arrow, depending on the console) which is displayed within or at either end of the string of characters currently under edit (called the "input string" from now on). The editor accepts a command string which must be terminated by ##. (<alt-mode> has been made equivalent to # for in order to maintain the similarity to TECO.) A command string is any concatenation of one or more legal commands which will be processed in left-to-right order. Display of the input string occurs at the end of the processing of each command string. # is used to enter the editor, to exit from the editor (as ##), and to terminate insert or search substrings. Otherwise, spurious #'s are ignored. Rubouts (the rubout or delete key on the console) may be used at any point prior to command termination to delete the last character typed in (which is echoed at the console.) ?? deletes the entire command. At any point prior to command termination, the user may type a <control>K, and the editor will reprint the characters of the command typed so far. This is useful in case excessive rubouts have obscured the sequence of characters in the command string.

12.3 A Description of the Commands

Some commands may be prefixed by an integer (represented below by "n") which usually may be positive or negative; although it may be 0 as well in the case of K, L, and W; and it must be non-negative in case of W. The default value of n is +1. Except in the case of R, if n is positive the commands operate toward the right of the cursor, if n is negative they operate toward the left. Only I and S may be suffixed. An error message will be printed if an illegal command substring is encountered or if any command substring fails. In case of such error, the processing of the current command string will be terminated at that point, with the offending command substring indicated.

Command	Mnemonic	Action
	(Commands which move the cursor)	
nC	Character	moves the cursor n characters.
nR	Reverse	moves the cursor n characters in the reverse direction (nR = -nC).
J or T	Jump to top Top	moves the cursor to the head of the input string.
B	Bottom	moves the cursor to the end of the input string.
nL	Line	moves the cursor to the right of the nth carriage return (OL moves left); e.g., L moves to the next line.
nSstring#	Search	moves the cursor to the right (left if n is negative) of the nth occurrence of "string" in the input string.
	(Commands which delete characters)	
nD	Delete	deletes n characters, and saves them in the "save-register" (see the G command below).
nK	Kill	deletes all the characters through the nth carriage return (OK kills left), and saves them in the "save-register"; e.g., K deletes the remainder of this line.

(Commands which insert characters)

Istring#	Insert	inserts the characters "string" at the current cursor position. The cursor is positioned at the right of the inserted text.
G	Get	inserts at the current cursor position the characters deleted by the last use of D or K. Thus G may be used in combination with D or K to move characters from one place to another in the input string; or to recover from an accidental use of D or K. There is only one "save-register".

(Commands which control display of results)

P	Print	simply reprints the input string. This is useful in case of console problems.
nW	Window	controls the window size of the display, which is the maximum number of characters displayed on each side of the cursor. This is useful in case of slow consoles and large input strings. OW will cause only the cursor to be displayed.
V	View	restores the display to full view, which is the normal mode (affected only by W).

will exit from the editor and is also the command string terminator. Two examples of legal command strings are 4C3DIFOO## and -2SBAR#3R##. The first moves right over four characters, deletes the next three characters, and inserts FOO. The second searches from the current pointer position to the beginning of the text for the second occurrence of BAR then moves left over three characters.

Appendix

Following is a condensed version of the grammar for MACSYMA along with an informal explanation of some of the rules. Note that an expression, through syntactically correct, may not be meaningful as in F(X):1. This will give the message ILLEGAL VALUE ASSIGNMENT if typed to MACSYMA.

The following notation is used. Braces with a subscript of 0 mean that one or none of the quantities enclosed are to be chosen. Braces without a subscript mean that exactly one of the enclosed quantities is to be chosen.

A line typed to MACSYMA is called a Sentence. This is the starting symbol for the grammar and is defined below in terms of successively smaller parts until finally reaching a terminal quantity. Terminal symbols are written entirely in capital letters while only the first letter of non-terminal symbols is capitalized.

```
- - - - - - - - - - - - - - - - - - - - - - - -

1.  Sentence ->
                 (a)  Statement
                 (b)  Statement, Statement, ...  , Statement

- - - - - - - - - - - - - - - - - - - - - - - -

2.  Statement ->
                  (a)  Expression
                  (b)  IF Expression THEN Statement { ELSE Statement }0
                  (c)  Qfunction :  Expression
                  (d)  Qfunction :: Expression
                  (e)  Qfunction := Expression
                  (f)  FOR Quantity : Expression { STEP Expression }0
                          ( THRU Statement    )
                         {  WHILE Expression   }
                          ( UNLESS Expression )

- - - - - - - - - - - - - - - - - - - - - - - -

3.  Expression ->
                   (a)  []
                   (b)  Grelation
                   (c)  Grelation OR Expression

- - - - - - - - - - - - - - - - - - - - - - - -

4.  Grelation ->
                  (a)  Relation
                  (b)  Grelation AND Relation
```

5. Relation ->

(a) Sum

(b) Sum $\left\{\begin{matrix} < \\ > \\ = \\ <= \\ >= \end{matrix}\right\}$ Sum

6. Sum ->

(a) $\left\{\begin{matrix} - \\ + \end{matrix}\right\}_0$ Term

(b) Sum $\left\{\begin{matrix} + \\ - \end{matrix}\right\}$ Term

7. Term ->

(a) Factor

(b) Factor . Term

(c) Term $\left\{\begin{matrix} * \\ / \end{matrix}\right\}$ Factor

8. Factor ->

(a) Base

(b) Base ** Factor

9. Base ->

(a) ( )

(b) Base !

(c) Qfunction

(d) ´ ´ Qfuntion

(e) $\left\{\begin{matrix} ´ \\ '' \end{matrix}\right\}$ Quantity

(f) (Arglist)

10. Qfunction ->

(a) Quantity

(b) Quantity ( )

(c) Quantity (Arglist)

- -

11. Quantity ->

```
(a) Atom
(b) [Arglist]
(c) Atom [Arglist]
```

- -

12. Arglist ->

```
(a) Statement
(b) Statement, Statement, ... , Statement
```

- -

13. Atom ->

```
(a) Integer
(b) Fpnumber
(c) Symbol
```

Bibliography and References

1. A.C.M. Proceedings of the Second Symposium on Symbolic and Algebraic Manipulation, Los Angeles, Calif., March, 1971.

2. Fateman, Richard J. "Essays in Algebraic Simplification" - Ph.D. Thesis - M.I.T. MAC TR-95

3. ——, "MACSYM User's Manual"

4. ——. Martin, W.A., Moses, J., and Wang, P.S. "The MACSYMA Papers" 1970

5. Moses, Joel M. "MACSYMA Primer"

6. McCarthy, John, et. al. "LISP 1.5 Programmers Manual" M.I.T. Press

7. White, John L. "An Interim LISP User's Guide" A.I. Memo 190 - March 1970 - M.I.T.

8. Eastlake, Donald E. "ITS Status Report" A.I. Memo 238 - April 1972 - M.I.T.

9. Wang, Paul S. "Factoring Multivariate Polynomials Over the Integers" - to appear.

10. Van der Waerden, B. L. "Modern Algebra" Volume 1.

11. Heindel, Lee E. "Integer Arithmetic Algorithms for Polynomial Real Zero Determination" in [1].