A PROPOSAL FOR EXTENSIONS TO REDUCE

G. Üçoluk*, A. Hacınlıyan**
Department of Physics, Boğaziçi University
E. Karabudak
Department of Mathematics, Boğaziçi University

Informative Abstract:

Three classes of extensions are prooosed for REDUCE: A facility for evaluating arbitrary functions of matrices; a facility for grouping, modifying or restoring the status of various flags in REDUCE; further extensions and modifications for separating terms, coefficients of expressions, concatenation, and noncommuting algebra. These proposals have been implemented on the UNIVAC 1100 REDUCE system . Inclusion of these extensions on all version of REDUCE is suggested because of their usefulness.

REDUCE is probably the most widespread symbolic processing language. The extensions proposed below are designed to increase its flexibility and to speed up many operations normally requiring several steps, particularly in an interactive environment, which is the natural habitat of REDUCE. With a single exception to be noted, all of the proposed extensions are written in the symbolic mode of REDUCE. They should therefore be portable to any. REDUCE system, although they have been implemented on the UNIVAC 1100 REDUCE system.

It is well known that REDUCE does not intrinsically possess facilities for evaluating arbitrary functions of matrices and creating unit matrices of arbitrary dimensions. The latter is already available in the symbolic level of REDUCE but is not accessible to users in the algebraic mode.

By making use of this facility, the following two functions for generating unit matrices are introduced. 1) The operator UNIT (N) where

*Part of this work to be submitted to the Department of Physics in partial fulfillment of the requirements of a MS.C Degree

**Supported in part by the Turkish Scientific and Technical Research Council (TBTAK)

N is a positive integer or identifier evaluating to a positive integer generates the unit matrix of indicated size.It can be used in matrix expressions.

2)The operator TYPEUNIT (M) where M is anything that evaluates to a square matrix generates the unit matrix of the same dimensionality as M.

Further facilities introduced in connection with matrix operations are the operators DIMR (M) and DIMC (M) where M is anything which evaluates to a matrix. DIMR gives the number or rows, DIMC gives the number of columns in the matrix M.

Introduction of the operator MATFUNC(F,M) to evaluate an arbitrary, Taylor expandable function, F of an identifier evaluating to a square matrix M is proposed. Eigenvalues of M should be available to the system by one of the following mechanisms prior to the execution of MATFUNC.

(a) The eigenvalues may be given to the system by the declaration EIGENVALUE $V_1,V_2,V_3.. V_4$. Otherwise :

If M is 2x2, MATFUNC uses the quadratic formula to evaluate the eigenvalues. If M is of higher dimensionality and no eigenvalue declaration has been made, the system automatically creates the atoms e1,e2,...,eN in which e is the atom of the matrix name. (in the case above M1, M2,.....). For example, if the second argument of MATFUNC is the 3x3 matrix RØTAT,the eigenvalues are taken to be RØTAT1, RØTAT2, and RØTAT3 in the absence of an explicit declaration.

The result of such a calculation would, in general, depend on the atoms e1,e2,..... It is wortwhile to mention that the result must not depend on the eigenvalue, although the eigenvalues are used in the intermediate steps of the evaluation. The mathematical algorithm used by MATFUNC is described in APPENDIX 1.

REDUCE has over 30 flags controlling various aspects of its operation. Each of these flags mustbe turned on or off individually. Certain combinations of these flags frequently need to be manipulated as a group, especially in the interactive mode, to adjust the appearance of output.

It is a tedious task to make these manipulations individually. The following statements are now proposed in order to group flags, their status, and to store or reset their status as a group.

The notation is as follows: Anything in capital letters is directly written. Anything in lower case letters implies that something will be substituted for it. Parentheses imply choice of only one of the several alternatives. Square brackets denote an optional feature. Curly brackets imply choice of at least one of the several alternatives. All commas are optional. l-list stands for any list of labels introduced by a LABEL statement in the id position. f-list stands for any list of flags. st-list stands for a list of identifiers introduced via a STATE or STØRE statement. Backus normal forms of all the statements are given in Appendix II. All statements must terminate either with $ or; in conformity with standard REDUCE syntax. If $ is used as terminator, printing of information concerning the execution of that statement is suppressed. Such suppressions can be made permanent by turning off a newly introduced flag, FLGMSG. If; is used as terminator, the flag status information will be given if FLGMSG is ON.

The LABEL statement:

$$\text{LABEL id :}\ \left(\begin{array}{c}\left\{\begin{array}{l}\text{l-list}\\ \text{f-list}\end{array}\right\}\\ \text{ALL}\end{array}\right)\ \left[\text{EXCEPT}\ \left\{\begin{array}{l}\text{l-list}\\ \text{f-list}\end{array}\right\}\right]$$

will give id as a label to the list of flags in the parentheses excluding any which may appear optionally in the EXCEPT clause. This group of flags can be collectively referred to with id as label until another group is assigned to the id, or the label is extended or modified.

The STØRE statement:

$$\text{STØRE}\ [\text{id}\quad :]\ \left(\begin{array}{c}\left\{\begin{array}{l}\text{l-list}\\ \text{f-list}\end{array}\right\}\\ \text{ALL}\end{array}\right)\ \left[\text{EXCEPT}\ \left\{\begin{array}{l}\text{l-list}\\ \text{f-list}\end{array}\right\}\right]$$

will store the current status of all flags in its operand into the identifier id if it is given. Otherwise this information is stored into a temporary storage.

The STATE statement

$$\text{STATE id :}\ \left\{\begin{array}{l}\left(\text{ØN}\ \left\{\begin{array}{l}\text{l-list}\\ \text{f-list}\end{array}\right\}\quad\right)\\ \left(\text{ØFF}\ \left\{\begin{array}{l}\text{l-list}\\ \text{f-list}\end{array}\right\}\quad\right)\end{array}\right\}$$

will store into id the indicated states of the indicated flags without actually modifying them. The ØN clause immediately following: may be omitted.

The RESTØRE statement is

$$\text{RESTØRE}\quad \left(\begin{array}{c}\left\{\begin{array}{l} \text{l-list}\\ \text{f-list}\\ \text{st-list}\end{array}\right\}\\ \text{ALL}\end{array}\right)\quad \left[\text{EXCEPT}\quad \left\{\begin{array}{l} \text{l-list}\\ \text{f-list}\end{array}\right\}\right]$$

This statement will restore the status of the indicated flags to that which was last assigned via a STATE or STØRE statement. An error message is generated, if no value has been assigned to any of its active operands before the execution of RESTØRE. Values for active operands other than those in a st-list are obtained from the temporary storage mentioned in connection with the STØRE statement.
The IDLE statement

$$\text{IDLE}\quad \left(\begin{array}{c}\left\{\begin{array}{l} \text{l-list}\\ \text{f-list}\end{array}\right\}\\ \text{ALL}\end{array}\right)\quad \left[\text{EXCEPT}\quad \left\{\begin{array}{l} \text{l-list}\\ \text{f list}\end{array}\right\}\right]$$

will set its active operands into their system default values.

The STATUS statement

$$\text{STATUS}\quad \left(\begin{array}{c}\left\{\begin{array}{l} \text{l-list}\\ \text{f-list}\end{array}\right\}\\ \text{ALL}\end{array}\right)\quad \left[\text{EXCEPT}\quad \left\{\begin{array}{l} \text{l-list}\\ \text{f-list}\end{array}\right\}\right]$$

will print out the status of its active operands.
It is hoped that these statements will greatly enhance flag manipulation in REDUCE.

The implementation of the following extensions is also proposed.
1) GLUE, a n-ary concatenation operator of the form

GLUE (a1, a2,... an)

where

a1, a2.... an are identifiers which alone or after concatenation may evaluate to other identifiers. GLUE will concatenate the latest value the operands have evaluated themselves into. If this combination itself evaluates into another expression, that value will be returned .
2) SEPARATE, an operator to separate an expression relative to a binary operator, of the form

SEPARATE (S,op,ARR)

where S is an expression which is to be separated into terms with respect to the binary operator op. ARR is an identifier which becomes an array, its 0 element containing the number of terms to which S has been separated, its further elements containing successive terms into which S has been separated from left to right.

3) ARG, to return individual arguments of an prefix operator, of the form ARG(pop,n). It returns the n^{th} argument of the prefix operator pop.
4) Another operator NAME (argument) where the argument must evaluate to a prefix operator returns the name of this operator.
5) In REDUCE, a built in function CØEFF is available for extracting the coefficients of a polynomial. CØEFF takes three arguments (e, v, name). e is the polynomial, v is a kernel. Coefficients of its powers in e are to be separated by CØEFF. name is an identifier. If it is an array, its i^{th} element contains the coefficient of v^i in e. If it is a non-array identifier, new atoms name1 , name2 , namei are created namei contains the coefficient of v^i.

   In many problems, the highest power of v in e may not be known in advance and one may still wish to place the coefficients in an array. The operator KØEFF (e, v, name) is proposed exactly for this purpose. The identifier name is automatically generated as an array of the proper dimensionality.
6) NØNCØMMUTE and CØMMUTE declarations are proposed to implement noncommuting algebra.

   The former

   $$\text{NØNCØMMUTE} \quad V_1, V_2, \cdots, V_n$$

   where $V_1,V_2....V_n$ are variables, declares these variables to be noncommuting. This declaration is cancelled by the declaration

   $$\text{CØMMUTE} \quad V_1, V_2, \cdots, V_k$$

   Variables that are declared to be noncommuting are placed in a special set. Any variable in this set is treated as noncommuting with all other variables in the set. For example, if one declares NØNCØMMUTE A,B , A and B do not commute with each other. A further declaration NØNCØMMUTE C,D will now cause D to be noncommutative with A,B and C, A to be noncommutative with B,C and D.
7) RØØT, to calculate the numerical approximations to roots of a polynomial. It has the form RØØT (P, V1, ARR) where P is a polynomial of the variable V1 with complex coefficients which must evaluate to complex numerical expressions at the time of the call to RØØT. ARR will become an array containing the roots of P.

Since LISP is not suitable for numerical work, interfacing to a FØRTRAN, or PL/1 routine is needed. Such an interfacing will necessarily involve Operating System facilities such as checkpoint/ restart,interrupt, priviledged instruction, supervisor call dynamic runstream modification and return. All modern Operating systems contain such facilities. However, the precise form of these facilities and their availability to LISP and FØRTRAN (or PL/1) will differ from computer to computer. Hence, the details of implementation will be different for every system. In spite of this difficulty, the availability of this facility, or a more general facility enabling interfacing between REDUCE and a FØRTRAN program, is extremely useful, since it will enable blending of symbolic and numerical compuatation.

The implementation of the operator for the UNIVAC 1100 system is based on the checkpoint/restart feature and the CSF command enabling execution of certain EXEC-8 Job Control Commands in UNIVAC REDUCE. The coefficients of the polynomial are written in FØRTRAN compatible form into a file, REDUCE exits to the supervisor which then calls a FØRTRAN routine to evaluate the roots by a CERN library routine numerically. Control is then returned to REDUCE with the array ARR containing the numerical values of the roots.

Since similar features also exist in other REDUCE systems, a similiar implementation should be feasible in other systems.

An illustrative example is given in Appendix III. Further illustrative examples and source codes can be obtained from the authors upon written request.

The authors would like to express their gratitude to Prof. J.A. Campbell for his constant encouragement and valuable guidance during the progress of this work. Partial support from the Turkish Scientific and Technical Research Council is also acknowledged.

APPENDIX I

The algorithm used in MATFUNC is the following. Let F be any Taylor expandable function of a nxn matrix M, of the form

$$F(M) = a_0 I + a_1 M + a_2 M^2 + \cdots + a_k M^k + \cdots \tag{1}$$

M satisfies its own characteristic equation, so that p^n (M) = 0 where p^n is a n^{th} degree polynomial. Hence we can write

$$M^n = c_0 + c_1 M + c_2 M^2 + \cdots + c_{n-1} M^{n-1} \tag{2}$$

and using this relation repeatedly, all powers of M^k with $k > n$ can be expressed in terms of $M^k, k < n$. Hence, without loss of generality, we may truncate (1) as

$$F(M) = b_0 I + b_1 M + b_2 M^2 + \cdots + b_{k-1} M^{k-1} \tag{3}$$

Eq(3) is satisfied by the eigenvalues of the matrix, $\lambda_1, \lambda_2, \cdots, \lambda_n$ Hence, if there is no degeneracy, we have the following system of n equations to determine the n coefficient b_o, $b_1 \ldots b_{n-1}$.

$$F(\lambda_k) = b_0 + b_1 \lambda_k + b_2 \lambda_k^2 + \cdots + b_{n-1} \lambda_k^{n-1} \qquad k = 1, 2, \cdots, n \tag{4}$$

This gives for the unknown coefficients the following formula

$$\begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ \lambda_1 & \lambda_2 & \cdots & \lambda_n \\ \lambda_1^2 & \lambda_2^2 & \cdots & \lambda_n^2 \\ \vdots & \vdots & & \vdots \\ \lambda_1^{n-1} & \lambda_2^{n-1} & \cdots & \lambda_n^{n-1} \end{bmatrix}^{-1} \cdot \begin{bmatrix} F(\lambda_1) \\ F(\lambda_2) \\ \vdots \\ F(\lambda_n) \end{bmatrix} \tag{5}$$

If M is degenerate and hence has only $k < n$ independent eigenvalues, the above procedure must be slightly modified. It can be shown that M now satisfies a polynomial equation of degree k, and hence all powers of M greater than or equal to k may be reduced. Thus F(M) may be written as

$$F(M) = b_0 I + b_1 M + b_2 M^2 + \cdots + b_{n-1} M^{n-1} \tag{6}$$

and the k independent eigenvalues furnish k equations for determining b_o, $b_1 \ldots b_{k-1}$ and hence F(M).

APPENDIX II

BACKUS NORMAL FORM DEFINITIONS OF EXTE,NSIONS TO REDUCE FOR FLAG HANDLINGS

```
<sdec list> ::= <lf list> | <lf list> <on/off list> | <on/off list>
<on/off list> ::= ON <sdec list> | OFF <sdec list>
<lf list> ::= <l-id> <sep> <lf list> | <f-id> <sep> <lf list> | <l-id> | <f-id>
<l-id> ::=<id>
<f-id> ::= <flag>
<s-id> ::=<id>
<st-id> ::=<id>
<sep> ::=    | ,
<tm> ::= ; | $
<alf list> ::= ALL  <lf list>
<alfst list> ::= <s-id> | <s-id> <sep> <alfst list> | <alf list> | <st-id> |
                 <st-list> <sep> <alfst list>
<state statement> ::= STATE <s-id> : <sdec list> <tm>
<common statement tail> ::= <alf list> <tm> | <alf list> EXCEPT <lf list>  <tm>
<label statement> ::= LABEL <l-id> : <common statement tail>
<idle statement> ::= IDLE <common statement tail>
<store statement> ::= STORE <st-id> : <common statement tail> |
                      STORE <common statement tail>
<status statement> ::= STATUS <common statement tail>
<restore statement> ::= RESTORE <alfst list> <tm> |
                        RESTORE <alfst list> EXCEPT <lf list> <tm>
```

APPENDIX III

```
ΔBOGAZICI*OURREDUCE.REDUCE,F
ΔLISP$PF.LISP        BOGAZICI*REDUCEXTEND.
EXTENDED REDUCE (FEB-2-82) ...

% MATFUNC EXAMPLE ;

MATRIX M ;
M := MAT((1,0,0) , (0,COS X,SIN X) , (0,-SIN X,COS X)) $
FOR ALL X,Y LET
          LOG( X**Y ) = Y*LOG X ,
          LOG(1/X**Y) = -Y*LOG X ,
          SIN X = (E**(I*X) - E**(-I*X))/2/I ,
          COS X = (E**(I*X) + E**(-I*X))/2 ;
% HERE WE INTRODUCE THE EIGENVALUES : ;
EIGENVALUE M,   E**(I*X),E**(-I*X),1 ;
%MATFUNC TAKES THE LOG OF M : ;
MATFUNC(LOG,M) ;

MAT(1,1) := 0

MAT(1,2) := 0

MAT(1,3) := 0

MAT(2,1) := 0

MAT(2,2) := 0

MAT(2,3) := X

MAT(3,1) := 0

MAT(3,2) :=  - X

MAT(3,3) := 0


%EXAMPLES OF DIMR,DIMC,UNIT,TYPEUNIT ;
DIMR M  ;

3
DIMC M ;

3
UNIT 2 ;

MAT(1,1) := 1

MAT(1,2) := 0

MAT(2,1) := 0

MAT(2,2) := 1
%A SHORT STATEMENT WHICH CALCULATES THE
          CHARACTERISTIC POLYNOMIAL OF MATRIX M :
CHRPOL := DET(  M - LAMDA*TYPEUNIT(M)   ) ;

              (2*X*I)       2    (2*X*I)             (X*I)       3
CHRPOL := (E         *LAMDA  - E         *LAMDA _ E      *LAMDA  +

             (X*I)       2    (X*I)
            E     *LAMDA  - E      *LAMDA +

             (X*I)         2              (X*I)
            E      + LAMDA  - LAMDA)/E
```

```
% LET US TAKE ANY ARBITRARY TAYLOR
  EXPANDABLE FUNCTION :F OF A GENERAL
  2 X 2 MATRIX WITH ENTRIES A(I,J) ;
  MATRIX M(2,2) ;
EIGENVALUE M ,E1,E2 ;
OPERATOR A ,F ;
FOR I:=1:2 DO FOR J:=1:2 DO M(I,J):=A(I,J) ;
MATFUNC(F,M) ;

MAT(1,1) := (A(1,1)*F(E1) - A(1,1)*F(E2) - F(E1)*E2 + F(E2)*E1)/(E1 - E2)

MAT(1,2) := (A(1,2)*(F(E1) - F(E2)))/(E1 - E2)

MAT(2,1) := (A(2,1)*(F(E1) - F(E2)))/(E1 - E2)

MAT(2,2) := (A(2,2)*F(E1) - A(2,2)*F(E2) - F(E1)*E2 + F(E2)*E1)/(E1 - E2)


%LET US TAKE ASSUME THAT F IS AN TAYLOR
EXPENDABLE  FUNCTION : WE DECLARE IT
AS AN OPERATOR, AS USUAL, BUT GIVE
NO SPESIFIC DEFINITION FOR IT ;
OPERATOR F ;
%LET US DEFINE A MATRIX AS ;
M := MAT(( X , 3 - X )  ,  ( 2 + X , 1 - X )) $

*** M DECLARED MATRIX
%LET US TAKE F OF M ;
MATFUNC (F,M) ;

MAT(1,1) := (F(3)*X + 2*F(3) - F((-2))*X + 3*F((-2)))/5

MAT(1,2) := ( - F(3)*X + 3*F(3) + F((-2))*X - 3*F((-2)))/5

MAT(2,1) := (F(3)*X + 2*F(3) - F((-2))*X - 2*F((-2)))/5

MAT(2,2) := ( - F(3)*X + 3*F(3) + F((-2))*X + 2*F((-2)))/5


%EXAMPLES FOR SEPARATE,GLUE,ARGUMENT,NAM  ;

OPERATOR ANY ;
S := ANY(A1,A2,A3,A4) ;

S := ANY(A1,A2,A3,A4)
A2 := ANOTHERTHING ;

A2 := ANOTHERTHING
NAM(S) ;

ANY
ARGUMENT(S,2) ;

ANOTHERTHING
ARGUMENT(S,3) ;

A3
% LET US SEPARATE THE ABOVE CALCULATED
            CHARACTERISTIC  POLYNOMIAL, CHRPOL ;
N := SEPARATE(CHRPOL,+,HORSE) ;

N := 8
%  N ABOVE CONTAINS THE NUMBER OF TERMS ;
FOR I:=1:N DO WRITE HORSE(I) := HORSE(I) ;

               (X*I)       2
HORSE(1) := E       *LAMDA

               (X*I)
HORSE(2) :=  - E     *LAMDA

                     3
HORSE(3) :=  - LAMDA

                 2
HORSE(4) := LAMDA

HORSE(5) :=  - LAMDA

HORSE(6) := 1

                 2  (X*I)
HORSE(7) := LAMDA /E

                        (X*I)
HORSE(8) := ( - LAMDA)/E
%REMEMBER A2 FROM ABOVE ;
%LET US SET DONALD TO A ;
DONALD := A ;

DONALD := A
GLUE (DONALD,2) ;

ANOTHERTHING
```

```
% AN EXAMPLE FOR NONCOMMUTE ;
NONCOMMUTE X1,X2,X3 ;
(X1 + X2)*(X2 - X3) + (X3 - X1)**2 ;

X3*X3 - X2*X3 - X1*X3*2 + X2*X2 + X1*X2 - X3*X1 + X1*X1
%THIS RESULT IS DIFFERENT IN THE CASE OF
          COMMUTING X1,X2,X3  AS FOLLOWS ;
COMMUTE X1,X2,X3 ;
(X1 + X2)*(X2 - X3) + (X3 - X1)**2 ;

   2                            2                2
X1  + X1*X2 - 3*X1*X3 + X2  - X2*X3 + X3


% NOW EXAMPLES FOR FLAG FACILITY STATEMENTS .
STATUS GCD,MCD,NAT,DIV ;

GCD IS OFF
MCD IS ON
NAT IS ON

ON GCD,DIV ; OFF NAT ;
IDLE GCD,NAT ;
STATUS GCD,NAT,DIV ;
DIV IS OFF
GCD IS OFF
NAT IS ON

%WE MAY REFERE TO A GROUP OF FLAGS BY ;
LABEL DESY : GCD,DIV,FLOAT ;
%ANOTHER ONE       ;
LABEL TRICK : ALL EXCEPT DESY ,FORT :
STATUS DESY ;
DIV IS ON
GCD IS OFF
DIV IS ON
%NOW ANOTHER STATEMENT : STATE ;
STATE HIGH : ON DESY OFF MCD ;
%SOME ONE CAN SAVE THE CURRENT STATUS
    OF ANY GROUP OF FLAGS BY ;
STORE LIST ,DESY ;
FLOAT IS OFF
LIST STORED AS OFF
GCD STORED AS OFF
DIV STORED AS ON
%IN THE ABOVE STATEMENT THE TEMPORARY
     STORAGE WAS ASSSUMED ,
LET US HAVE A PERMANENT STORAGE :
STORE DUCK : DESY EXCEPT GCD ;
FLOAT STORED AS OFF
DUCK DECLARED STATE, HOLDING CURRENCIES :
DIV AS ON
FLOAT AS OFF
IDLE ALL ;
%WE MAY RESTORE ALL THE ABOVE STORED
    STATUSES BY ;
RESTORE DUCK ;

DIV RESTORED AS ON
FLOAT RESTORED AS OFF
RESTORE HIGH EXCEPT DIV ;

MCD RESTORED AS OFF
GCD RESTORED AS ON
FLOAT RESTORED AS ON
%THE INFORMATIVE PRINTING CAN BE SUPRESSED ;
RESTORE DUCK $


QUIT:

END OF LISP
```