<< TECHNICAL REPORT >>

Matrix procedures for REDUCE3.3: programming notes*

Yoriaki Fujimori

1 Preface

Since the advent of the computer system, the human inteligence has exerted great effort on the computer algebra system which carries out complex and energy consuming symbolic computations in mathematical problems. REDUCE is one of such systems which are now available in educational institutions.

Matrix algebra is one of such troublesome matters which take much of our time and energy. Indeed, many problems in economics and econometrics are formulated in matrix. Hence, economists will be liberated from tiresome computation of matrices, if they are provided with good intelligent tools to carry out computation for them.

The objective of this programming note is to make memorundum for those who are interested in coding matrix-valued functions in REDUCE.[1]

2 Lisp-mode matrix-valued procedures

Many ready-made functions are available in REDUCE, so that users can define new functions in the algebraic mode without difficulty. As for matrix manipulation, however, there is no way to define matrix-valued functions in the ordinary algebraic mode: they must be defined in the symbolic mode. Hence, we shall attempt to make a brief manual as to how to make matrix-valued functions in REDUCE3.3.

2.1 Preliminaries

REDUCE is implemented on LISP systems. The standard lisp system for REDUCE is literally the STANDARD LISP. Terminology of the LISP is helpful for understanding REDUCE, so that we shall introduce basic terminology in advance.

A basic data form of LISP is a bare element called an *atom*. Numbers like 1 and 2 are atoms, and characters like *a* and *b* are also atoms. A series of characters starting with one of 26 alphabets is called an *identifier*, or in short *id*.

A set of elements in a pair of blankets is called a *list*. A list itself can be an element of other list. The list is a fundamental data form of LISP.

Operations to work on lists are called LISP-*functions*. Major functions concern obaining elements of lists, creating lists and so forth.

Cons is a basic function to create lists. It is often denoted by dot ' . ' put between two elements,

```
A . B
```

*The contents of this note is prepared on REDUCE3.3, but the current version of REDUCE as of May 1992 is 3.4.

[1]We identify functions with procedures with respect to REDUCE, so that they are used interchangeably in the following.

This leads to the definition of the *dotted pair*, which is another fundamental data format of LISP.

If the second argument of **cons** is a list, then the first argument is inserted at the top position of that second argument.

The first element of a list L is obtained by applying function called *car*. The first element of a list is often called its *car*-part.

A list which has no element is said to be the *null* list.

An operator on list which returns the list without its car-part is *cdr*. Hence, let L be a list such that $L = (l_1\ l_2 \ldots l_n)$, and one gets

$$\mathrm{car}\, L \Longrightarrow l_1$$

and

$$\mathrm{cdr}\, L \Longrightarrow (l_2 \ldots l_n).$$

2.2 Fundamental REDUCE data

In the algebraic mode of REDUCE, one has only to put in data as one manipulates by hand writing. Inside REDUCE, however, all data are provided with their specific internal expressions.

At the outset, we shall place the following typographical assumption. That is, when we explain mathematical facts, symbols are represented by italics, whilst key-board inputs of them is expressed by small typewriter prints, say `x`. Since most of the REDUCE systems are implemented in such a way that symbols are treated as capital letters, symbols returned by REDUCE are expressed by capital letters.

2.2.1 Numbers

One of the most basic data of REDUCE is numbers, especialy integers. Rational numbers and floating point numbers are expressed by combining relevant integers.

In the first place, let us take a numeric datum $\frac{2}{3}$. Type from the key-board as:

```
share u;
u := 2/3$
lisp u;
```

and it is seen that the fundamental number datum is a list consisting of three elements:

```
(*SQ (2 . 3) T)
```

inside REDUCE.[2]

The middle element of the list is a dotted pair which is the core of the numeric object. A dotted pair of the numerator and the denominator parts of a number is called the *standard quotient*, or in short *sq*.

Since $\frac{2}{3}$ needs no further calculation. REDUCE puts the tag `T` immediately after the standard quotient. `T` indicates that the expression is the final and simplified expression. Hence, this can be called the *evaluation suffix*. REDUCE also inserts `*SQ` to the top of the standard quotient to indicate that the datum concerned is ready for parsing as a scalar.

Thus, the datum format as above

```
(*SQ <standard-quotient> T)
```

is created. This is called the *prefix form* of the numeric object.

[2] In REDUCE, integers are looked at as a rational number the denominator of which is 1. In the case of integers, however, it is not easy to look into the format of data.

2.2.2 Scalars

In the next place, let us consider a scalar expression containing non-numeric symbols and ids in detail.

Suppose a variable x, i.e., a scalar expression x, in the algebraic mode, then its standard quotient is expressed as

```
((((X . 1) . 1)) . 1)
```

Although this looks a very complicated datum, it is still a standard quotient.

The standard quotient is a dotted pair of the numerator and the denominator of a scalar object.

The standard quotient looked at as a dotted pair, gives the exact information of itself: the last dot split the pair into the numerator and the denominator.

That is, the numerator part of x is represented by

```
(((X . 1) . 1))
```

whereas its denominator part is exactly 1.

One may regard the above standard quotient from the angle of the list manipulation. That is, let <A> be the standard quotient of a scalar object a, and car <A> gives the numerator part while cdr <A> gives the list of the denominator part of the standard quotient.

Now, let us take the numerator part of the standard quotient of x. The first innermost dot indicates that the power of x is 1, and the second dot indicates that the coefficient of x is again 1. At the end of the numerator part, there is two duplicated closing blackets, which shows that the constant term to x is 0 (nil). Thus, x in the algebraic mode is here recognized as a rational expression, $\frac{1 \times x^1 + 0}{1}$.

X in the standard quotient of x is the ultimate core, and hence it is called the *kernel*. A dotted pair of a kernel and a positive integer, as in (X . 1), is called the *standard power*. A dotted pair of a standard power and a positive integer, as in ((X . 1) . 1), is called the *standard term*.

In short, when one types numbers or any other id to indicate a scalar expression, REDUCE recognises it in its prefix form, whilst in the LISP-mode its standard quotient matters.

In what follows, the correspondence of variable x in mathematics and its counterparts in REDUCE is illustrated by the following:

Table 1: Forms of variables

mathematics	keyboard input	prefix form	standard quotient
x	x	(*SQ <X> T)	<X>

2.2.3 Standard forms

In the third place, let us consider a scalar expression which is defined by an arithmetic operation, say $x = \frac{a}{b}$. In this case, the standard quotient of x is represented by

```
((((A . 1) . 1)) ((B . 1) . 1))
```

In appearance, one sees no dot between the numerator and the denominator, but the structure of the standard quotient is still the same from the angle of the list manipulation.

The story does not change, even if we take a more complicated scalar expression $y = \frac{ax^2 + c}{b}$, for instance. Its standard quotient is

```
((((A . 1) ((X . 2) . 1)) ((C . 1) . 1)) ((B . 1) . 1))
```

That is, the first element of this list corresponds to the numerator $ax^2 + c$, while the second corresponds to the denominator b.

The numerator part and the denominator part of the standard quotient belongs to the same category called the *standard form* or the *standard polynomial.* To put it in the opposite direction, a dotted pair of two standard forms is the standard quotient.

Take the numerator part of y, for instance. `((A . 1) ((X . 2) . 1))` is a dotted pair of a standard power `(A . 1)` and a standard form `((X . 2) . 1)`, which, corresponding to ax^2, constitutes a standard term. The dotted pair of a standard term `((A . 1)((X . 2) . 1))` and a standard form `((C . 1) . 1)` constitutes a standard form.

2.2.4 Summary of data types

The following summarises the basic data types of REDUCE.

The most fundamental data-type in REDUCE are positive integers and symbols.

0 is represented by `nil` in REDUCE. A ratio of two integers is expressed by a dotted pair of the two: $\frac{m}{n}$ is indiated by `(M . N)`. The core structure of rational numbers in REDUCE is a dotted pair.

Integers, float, `nil` and dotted pairs of integers constitute the REDUCE data type called *constants.*

A symbol which is not assigned a value is called the *kernel.*

A LISP's s-expression can be a kernel.

A dotted pair of a kernel `X` and a positive integer `N`, `(X . N)` , is called the *standard power.*

A dotted pair of a standard power and a positive integer `N`, say `((X . 1) . N)`, is called the *standard term.*

Constants and dotted pairs of standard term and standard polynomial constitute the data type called the *standard polynomial.*

Dotted pairs of two standard polynomials are called the *standard quotient.*

Table 2: Summary of data types

Class	Ingredients	
Constants	integer	`N`
	rational numbers	`(N . M)`
	floats	
	0	`nil`
Kernel	symbols	
Standard power	dotted pair of kernel and integer	
Standard term	dotted pair of standard power and standard form	
Standard form	constant	
	dotted pair of standard term and standard form	
Standard quotient	dotted pair of standard forms	

2.2.5 Data conversion and handling

In order to carry out mathematical manipulations in the LISP-mode, important points are two: the one is to convert data format, and the other is to handle standard quotients.

Data conversion in REDUCE is also carried out by functions. The following table gives the summary of major conversions. (Remark that scalar indicates the prefix form.)

In order to carry out manipulations of scalars in the LISP-mode, one has only to perform arithmetics of standard quotients. Arithmetic operators to handle standard quotients are realized as LISP functions in REDUCE. They are `addsq()`, `multsq()`, `invsq()` and `negsq()`.

Remark that in the above table all the arguments and return values are all standard quotients.

Table 3: Basic built-in conversion functions

Function	Argument	Return-value
`simp!*`	scalar	standard quotient
`prepsq`	standard quotient	scalar
`!*a2k`	scalar	kernel
`!*k2q`	kernel	standard quotient
`!*q2a`	standard quotient	scalar
`!*q2k`	standard quotient	kernel

Table 4: Basic built-in arithmetic functions of standard quotients

Syntax	Return-value
`addsq(a,b)`	`a+b` (summation)
`multsq(a,b)`	`ab` (multiplication)
`invsq (a)`	`1/a` (inversion)
`negsq (a)`	`-a` (multiplication by -1)

If one needs to operate on numerators, one has only to handle the numerators of object. We do not, however, explain the details of manipulations of the standard form.

2.3 Matrix as a structured list

So far, we have been concerned with the so-called scalar valued mathematical object. In REDUCE, they are called non-structured for its list format is not restricted. Simply, they are bare prefix forms.

Some data format of REDUCE are, however, characterised by their tag. Most of REDUCE data are regarded as list. If a list has the tag `LIST`, then it is treated as REDUCE-LIST, while if a list has `MAT` tag, it is treated as `matrix`.

Again, we have to make distinction between the algebraic and the lisp modes.

In the algebraic mode, one needs to type, say,

```
mat ((a11,a12), (a21,a22));
```

to define 2×2 matrix A. Since a_{ij} are scalars, the above input creates the list of lists of prefix forms the tag of which is `MAT`, i.e.,

```
(MAT ((A11 A12) (A21 A22))
```

For short, we call the above the *matrix form.*

Inside REDUCE, however, one needs not to handle the above matrix form. Since the arithmetic operation can be made in terms of the standard quotient, one may remove the `MAT` tag of the list, and converts the matrix form to the list of lists of standard quotients, i.e.,

```
((<A11> <A12>) (<A21> <A22>))
```

A list of lists of standard quotients is called the *matrix canonical form.*

2.4 Matrix-valued function and parsing

Let `foo` be a matrix-valued function, the body of which is adequately defined.

`Foo` will not yet function properly, when called in the manipulation, if one more step is not cleared.

If one wishes to use the function `foo` as a matrix-valued function in the algebraic mode, it must be declared in advance that the function returns a matrix value. This needs two statements:

```
put ('foo,'rtypefn,'(lambda (x) 'matrix));
flag ('(foo),'matflg);
```

The first says that `foo` is a matrix-type function. The second registers `foo` as a matrix-valued function.

If the first argument of function `foo` is a matrix variable, the tag of which is `MAT`, then the above two can be replaced by a single statement as

```
put ('foo,'rtypefn,'getrtypecar);
```

Remark that there is no defining a matrix-valued procedure which can take an arbitrary number of arguments. That is, the number of arguments of a matrix-valued procedure should be fixed when that procedure is defined.

3 Algorithms

We outline here some of algorithms, with which economists may not be familiar. We do not repeat fundamental definitions related to matrices. By default, elements of matrices are assumed to be real numbers, because real matrices are most important in economics.

For most of algebras, the equality and the difference of their elements are defined in such a way that $A = B$ is equivalent to $A - B = 0$. A set of matrices is no exception. Most of matrix equality formula are *mechanical* in the sense that they are immediately confirmed by the computer algebra system by giving fundamental definitions to the system.

3.1 Quadratic forms

A quadratic form F of variables $x_1, x_2, \ldots, x_n$ is abreviated as

$$F(x_1, \ldots, x_n) = \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}x_i x_j. \tag{3.1.1}$$

Without loss of generality, coefficients of a quadratic form can be chosen as $a_{ij} = a_{ji}$. Given a quadratic form $F(x_1, \ldots, x_n)$, and it can be expressed by ${}^t xAx$, where $A = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix}$ is a symmetric matrix and $x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$.

A quadratic form of the form $\sum_{i=1}^{n} \alpha_i x_i^2$ is called the *canonical form*. A quadratic form can be transformed into its canonical form by Lagrange's method. Equation (3.1.1) can be rearranged as

$$F(x_1, \ldots, x_n) = a_{11}x_1^2 + 2\sum_{j=2}^{n} a_{1j}x_1 x_j + g(x_2, \ldots, x_n), \tag{3.1.2}$$

so that

$$F(x_1, \ldots, x_n) = a_{11}\left(x_1 + \frac{a_{12}}{a_{11}}x_2 + \cdots + \frac{a_{1n}}{a_{11}}x_n\right)^2 + F_1(x_2, \ldots, x_n), \tag{3.1.3}$$

where

$$F_1(x_2, \ldots, x_n) = -a_{11}\left(\frac{a_{12}}{a_{11}}x_2 + \cdots + \frac{a_{1n}}{a_{11}}x_n\right)^2 + g(x_2, \ldots, x_n). \tag{3.1.4}$$

This means that via variable transformation

$$y_1 = x_1 + \frac{a_{12}}{a_{11}}x_2 + \cdots + \frac{a_{1n}}{a_{11}}x_n$$

F is rewritten as

$$a_{11}y_1^2 + F_1(x_2, \ldots, x_n).$$

By applying the same procedure to $F_1(x_2, \ldots, x_n)$, the quadratic form F will be transformed into the canonical form

$$\alpha_1 y_1^2 + \cdots + \alpha_n y_n^2,$$

where $\alpha_1 = a_{11}$.

If $a_{11} = 0$, then one has only to rearrange the order of variables, in so far as there exists a $1 \leq k \leq n$ such that $a_{kk} \neq 0$.

If for all $1 \leq k \leq n$ holds $a_{kk} = 0$, then find a pair (i, j) such that $a_{ij} \neq 0$ and transform variables as

$$x_k = \begin{cases} y_k + y_j & (\text{if } k = i), \\ y_k & (\text{if } k \neq i), \end{cases}$$

for the smallest number i. The quadratic form in terms of y_is has a quadratic term $a_{ij}y_j^2$, so that the previous procedure can be applied.

quadfmatrix (q,v) returns the coefficient matrix of quadratic form q with respect to variables v. **canonquadform(y,x,w)** is the main procedure to deal with the quadratic form.

3.2 The Kronecker product and the Hadamard product

Definition 3.1 Let $A = (a_{ij})$ and $B = (b_{ij})$ be $m \times n$ and $r \times s$ matrices respectively. The *Kronecker product* of A and B, denoted by $A \otimes B$, is defined by

$$A \otimes B = (a_{ij}B). \tag{3.2.1}$$

Similarly, let A and B be n- and m-dimensional square matrices, and the *Kronecker sum* of A and B, denoted by $A \oplus B$, is defined by

$$A \oplus B = A \otimes I_m + I_n \otimes B. \tag{3.2.2}$$

Definition 3.2 Let A and B be $m \times n$ matrices. The *Hadamard product* of $A = (a_{ij})$ and $B = (b_{ij})$, denoted by $A \odot B$, is defined by

$$A \odot B = (a_{ij}b_{ij}). \tag{3.2.3}$$

Most of formula of the Kronecker product and the Hadamard product of matrices are mechanical.

Vectorization of matrices is closely related to the Kronecker product. Let A be an $m \times n$ matrix, and let a^i be its i-th column. The mn-dimensional vector obtained by concatenating a^is vertically is called the *vectorization* of matrix A, denoted as

$$\operatorname{vec} A = \begin{pmatrix} a^1 \\ \vdots \\ a^n \end{pmatrix}. \tag{3.2.4}$$

Conversely, given adequate m and n, and a vector x can be transformed into an $m \times n$ matrix X such that $\operatorname{vec} X = x$.

kronecker (a,b) yields the Kronecker product of matrices A and B, while **krsum (a,b)** gives the Kronecker sum of A and B. Meanwhile, **hadamard (a,b)** returns the Hadamard product of A and B.

3.3 Generalised inverses

Let A be an $m \times n$ matrix.

Definition 3.3 A matrix X which satifies

$$AXA = A \tag{3.3.1}$$

is called a *generalised inverse*, or *g-inverse* of A, and it is denoted by A^-.

Theorem 3.1 *Let A be an $m \times n$ matrix, and a solution of a system of linear equations $Ax = y$, if ever it exists, can be expressed by $x = A^- y$.*

Definition 3.4 A matrix X which satifies (3.3.1) and

$$XAX = X \tag{3.3.2}$$

is called a *reflexive generalised inverse* of A, denoted by ${A_r}^-$.

Definition 3.5 A matrix X which fulfills (3.3.1) and

$${}^t(XA) = XA \tag{3.3.3}$$

is called the *minimum norm g-inverse* of A, denoted by ${A_m}^-$.

Theorem 3.2 *$x = {A_m}^- y$ gives the minimum norm solution of the system $Ax = y$.*

Definition 3.6 A matrix X which fulfills (3.3.1) and

$${}^t(AX) = A\,{}^tA \tag{3.3.4}$$

is called the *least square g-inverse* of A, denoted by ${A_l}^-$.

Definition 3.7 A reflexive g-inverse X of A which fulfills

$${}^t(AX) = AX \tag{3.3.5}$$

$${}^t(XA) = XA \tag{3.3.6}$$

is called the *Moore-Penrose g-inverse* or *MP-inverse* of A, which is denoted by A^+.

Remark 3.1 ${A}^-, {A_r}^-, {A_m}^-$ and ${A_l}^-$ are not uniquely determined, but A^+ is uniquely determined.

Theorem 3.3 *Let A be an $m \times n$ matrix. If* $\operatorname{rank} A = \min(m, n)$, *then MP-inverse of A is given by*

$$A^+ = {}^tA(A\,{}^tA)^{-1}, \text{ if } \operatorname{rank} A = m \leq n \tag{3.3.7}$$

$$A^+ = ({}^tA\,A)^{-1}\,{}^tA, \text{ if } \operatorname{rank} A = n \leq m. \tag{3.3.8}$$

If A is not of full row- or column-rank, then one has only to apply the following:

Theorem 3.4

$$\lim_{\varepsilon \to 0}({}^tA\,A + \varepsilon I)^{-1} \cdot {}^tA = A^+. \tag{3.3.9}$$

In view of the above, one has only to evaluate:

$$({}^tA\,A + \varepsilon I)^{-1} \cdot {}^tA$$

with $\varepsilon \to 0$,

Theorem 3.5 *Let A be an $m \times n$ matrix, and let Z be a g-inverse of A. Then, for an arbitrary chosen $n \times m$ matrix Y,*

$$X = Z + Y - ZAYAZ \tag{3.3.10}$$

gives an arbitrary g-inverse of A.

Since one may take $Z = A^+$, (3.3.10) can be rewritten as

$$X = A^+ + Y - A^+AYAA^+. \tag{3.3.11}$$

mpinv (a), which is based on (3.3.9), returns the MP-inverse of A, whilst **ginv (a)**, based on (3.3.11), yields a generalised inverse of A.

3.4 Minimum polynomials

Let f be a polynomial in x as

$$f(x) = a_0 + a_1 x + \cdots + a_n x^n$$

and we write for square matrix A a matrix polynomial[3]

$$f(A) = a_0 I + a_1 A + \cdots + a_n A^n.$$

Definition 3.8 A polynomial which takes null value for A is called the *minimum polynomial* of A, if there is no other polynomial with less degree.

Regardeng $I, A, A^2, \ldots, A^k$ as n^2-dimensional vectors, we can find the coefficients of the minimum polynomial of matrix A in a finite number of manipulations. That is, find the smallest k such that $I, A, A^2, \ldots, A^k$ are linealy dependent: there exists a set of scalars α_is

$$\alpha_0 I + \alpha_1 A + \cdots + \alpha_k A^k = O. \tag{3.4.1}$$

Since k is the smallest, α_k can be assumed to be 1. If $\alpha_k = 0$, then $I, A, A^2, \ldots, A^{k-1}$ are linearly dependent, which contradicts that k is the smallest. Since $\alpha_k = 1$, (3.4.1) will be reduced to

$$\alpha_0 I + \alpha_1 A + \cdots + \alpha_{k-1} A^{k-1} = -A^k, \tag{3.4.2}$$

that is,

$$\mathbf{A}\alpha = -\operatorname{vec} A^k, \tag{3.4.3}$$

where

$$\mathbf{A} = \left(\operatorname{vec} I, \operatorname{vec} A, \ldots, \operatorname{vec} A^{k-1}\right), \alpha = \begin{pmatrix} \alpha_0 \\ \vdots \\ \alpha_{k-1} \end{pmatrix}.$$

Since (3.4.3) is an inhomogeneous equation with a full column rank, the following facts can apply.

Definition 3.9 Let M be an $m \times n$ matrix and b be an $m \times 1$ vector. An inhomogeneous equation

$$Mx = b \tag{3.4.4}$$

is *consistent*, if there exists a vector x that satisfies the equation.

Theorem 3.6 *Equation* (3.4.4) *is consistent, if and only if M has full row- or column-rank. In this case, the solution of* (3.4.4) *can be expressed by $x = M^+ b$.*

It follows that the solution of (3.4.3) is given by

$$\alpha = -\mathbf{A}^+ \operatorname{vec} A^k. \tag{3.4.5}$$

By evaluating (3.4.5), the minimum polynomial of A can be determined as

$$f(t) = t^k + \cdots + \alpha_1 t + \alpha_0. \tag{3.4.6}$$

[3]Traditionally, polynomials are denoted by

$$a_0 x^n + \cdots + a_{n-1} x + a_n$$

but, from the standpoint of computer algebra, it is convenient to write polynomials in such a way that the coefficient's suffix corresponds to the degree of variables.

minipolycoeff (m) yields the list of coefficients of the minimum polynomial of matrix M in the ascending order in the light of (3.4.5), while **minipoly (m,x)** gives the minimum polynomial of M in x.

3.5 Matrix equations

In many fields of matrix application, we encounter various types of important equations which involves unknown matrices. Regarding $m \times n$ matrices as mn-dimensional vectors, the equation concerned can be seemingly reduced to an system of ordinary linear equations. Nevertheless, the problem of the null factor remains.

By means of the Kronecker product, solutions of the matrix equation can be well represented. Apart from the efficiency and the speed of computation, the Kronecker product representation is easy to handle for applications.

3.5.1 Lyapunov

Let x be an n-dimensional vector, the i-th component of which is a function of t. Let A be an $n \times n$ constant matrix, and consider a simultaneous system of linear differential equations:

$$\frac{dx}{dt} = Ax \ . \tag{3.5.1}$$

With regards to the asymptotic stability of its equilibrium, the following theorem is known:

Theorem 3.7 *The equilibrium of* (3.5.1) *is asymptotically stable, if and only if there exists a positive definite symmetric matrix X such that*

$$A^*X + XA = C \tag{3.5.2}$$

for an arbitrary chosen positive definite symmetric matrix C.

Equation (3.5.2) is called the *Lyapunov equation*. The solution of the Ljaponov equation is represented in terms of the Kronecker product.

Theorem 3.8 *Let A and B be m- and n- dimensional square matrices respectively. An $n \times m$ matrix X satisfying*

$$AX + XB = C \tag{3.5.3}$$

is given by

$$\mathbf{vec}\, X = \left({}^tB \oplus A\right)^{-1} \mathrm{vec}\, C. \tag{3.5.4}$$

Remark 3.2 If A and $-B$ have common eigenvalues, then (3.5.3) may not have a unique solution.

3.5.2 $AXB = C$ type

The above can be generalised to the matrix equation with more than two terms in the left-hand side of the equality sign.

Theorem 3.9 *The solution of a matrix equation $AXB = C$ can be expressed as*

$$\mathrm{vec}\, X = \left({}^tB \otimes A\right)^{-1} \mathrm{vec}\, C. \tag{3.5.5}$$

The above can be extended to the matrix equation

$$A_1XB_1 + \cdots + A_rXB_r = C. \tag{3.5.6}$$

Theorem 3.10 *The solution X of* (3.5.6) *can be expressed by*

$$\mathrm{vec}\, X = \left(\sum_{i=1}^{r} {}^tB_i \otimes A_i\right)^{-1} \mathrm{vec}\, C. \tag{3.5.7}$$

3.6 The Hurwitz test

The Hurwitz test is another useful test for examining the stability of the system which is described by the linear differential equation of higher degrees.

Consider a linear differential equation of order n:

$$a_n \frac{d^n y}{dx^n} + \cdots + a_1 \frac{dy}{dx} + a_0 = 0, \tag{3.6.1}$$

and the characteristic polynomial of this system is given by

$$a_n t^n + \cdots + a_1 t + a_0 = 0. \tag{3.6.2}$$

If all roots of (3.6.2) possess negative real parts, then the system is said to be *asymptotically stable.* Whether the system is asymptotically stable or not is examined by constructing the Hurwitz matrix:

$$H = \begin{pmatrix} a_{n-1} & a_{n-3} & a_{n-5} & \cdots & a_{2n-1} \\ a_n & a_{n-2} & a_{n-4} & \cdots & a_{2n-2} \\ 0 & a_{n-1} & a_{n-3} & \cdots & a_{2n-3} \\ 0 & a_n & a_{n-2} & \cdots & a_{2n-4} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & a_0 \end{pmatrix},$$

where $a_r = 0, r < 0$.

For example, in case $n = 6$, the Hurwitz matrix is expressed by

$$H = \begin{pmatrix} a_5 & a_3 & a_1 & 0 & 0 & 0 \\ a_6 & a_4 & a_2 & a_0 & 0 & 0 \\ 0 & a_5 & a_3 & a_1 & 0 & 0 \\ 0 & a_6 & a_4 & a_2 & a_0 & 0 \\ 0 & 0 & a_5 & a_3 & a_1 & 0 \\ 0 & 0 & a_6 & a_4 & a_2 & a_0 \end{pmatrix}.$$

The Hurwitz test is as follows:

Theorem 3.11 (Hurwitz) *If all the principal diagonals of H are positive, then the system described by* (3.6.1) *is asymptotically stable.*

4 Lisp-mode utility procedures

The following table, Table 5, lists up some of utility procedures which are originally defined in MATR.RED and widely employed in MATRLIB.[4]

5 Syntax of matrix procedures—Matrlib.red

In this section, the names of procedures are classified by their return-values. In the Function column of the following tables, the sequence of arguments is also indicated. Except a few cases, the number of arguments called by the procedures is determined as specified.

The plan of procedures is twofold. The one is the group of procedures which carry out mathematical manipulations, while the other are utility functions to perform matrix data manipulation for programming.

Most of procedures are coded in the following design principle: procedures which are called by users in the algebraic mode are just interfaces between the algebraic mode and the LISP mode. Data conversion is done in these procedures. Auxiliary procedures embody algorithm to carry out

[4]Recall that the procedures listed in the following assume REDUCE3.3, and not REDUCE3.4.

Table 5: Built-in matrix functions

Function	Arguments	Return-value
`matsm (U)`	`U`: matrix data	matrix canonical form of U
`addm (A1,A2)`	`A1,A2`: matrix canonical form	matrix canonical form of $A_1 + A_2$
`multm (A1,A2)`	`A1,A2`: matrix canonical form	matrix canonical form of $A_1 \times A_2$
`multsm (s,A)`	`s`: sq. `A`: matrix canonical form	matrix canonical form of sA
`scalprod (L1,L2)`	`L1,L2`: list of sq.	sq. of $\langle L_1, L_2 \rangle$
`tp1 (U)`	`U`: matrix canonical form	matrix canonical form of ${}^tU$
`detq (U)`	`U`: matrix canonical form	scalar, $\det U$
`matinv (U)`	`U`: matrix canonical form	matrix canonical form of U^{-1}
`lnrsolve (U,V)`	`U,V`: matrix canonical form	matrix canonical form of $U^{-1}V$
`generateident (N)`	`N`: integer	matrix canonical form of identity matrix I_N
`augment(U,V)`	`U,V`: matrix canonical form	matrix canonical form of $(U \quad V)$
`nzero (N)`	`N`: integer	list of N zeros

desired mathematical manipulation, and that they are performed in the LISP-mode: this means that fundamental data dealt with by those auxiliary procedures are the standard quotient.

Procedures are coded from the angle of mathematical clearness, so that some of them may not be efficient. Mathematical clearness is also helpful for maintenance.

The following lists up those functions that are called in the algebraic mode. Auxiliary functions are not listed at all. As for auxiliary procedures, refer to the source.

5.1 Scalar-valued procedures

Table 6: Scalar-valued functions of matrices

Function	Arguments	Return-value
`rows (M)`	`M`: matrix	the number of rows of M
`columns (M)`	`M`: matrix	the number of columns of M
`elemat (M,i,j)`	`M`: matrix `i,j`: integer	(i,j) element of M
`minor (M,i,j)`	`M`: matrix `i,j`: integer	(i,j) minor of M
`cofactor (M,i,j)`	`M`: matrix `i,j`: integer	(i,j) cofactor of M
`matrank (M)`	`M`: matrix	rank of matrix M
`charpoly (M, x)`	`M`: matrix `x`: id	eigen equation of M in terms of x
`sqnorm (X)`	`X`: matrix	matrix norm, $(\mathrm{tr}\,{}^tXX)^{1/2}$
`minipoly (X,r)`	`X`: matrix `r`: kernel	minimum polynomial of X with respect to r

5.2 List-valued procedures

Table 7: List-valued functions of matrices

Function	Arguments	Return-value
`klistize (M)`	`M`: matrix	list which corresponds to M

`canonquadform (Q,V,W)`	`Q`: scalar
	`V,W`: lists of kernels
	$V = \{v_1, \ldots, v_n\}$,
	$W = \{w_1, \ldots, w_n\}$
		list of lists,
		`{{b1,{w1=...}},`
		`{b2,{w2=...}},`
		`... }`
		of scalars b_i and equation
		$w_i = \sum c_{ij} v_j$.
		Q is transformed into $\sum b_i w_i^2$

`minipolycoeff (M)`	`M`: matrix	coefficients of the minimum polynomial of M in the ascending order

5.3 Matrix-valued procedures

We have four tables here: Table 8, Table 9, Table 10 and Table 11. Some of procedures are utilities to hack matrices.

The source of MATRLIB.RED can be obtained on request from

fujimori@tansei.cc.u-tokyo.ac.jp

or

fujimori@cfi.waseda.ac.jp

by means of electric mail via internet.

References

[1] Barnett, S. (1990). *Matrices: Methods and Applications*, Oxford University Press.

[2] Ben-Israel, A., Greville, T. N. E. (1974). *Generalized Inverses: Theory and Applications*, John Wiley and & Sons.

[3] Graham, A.(1981). *Kronecker Products and Matrix Calculus with Applications*, Ellis Horwood.

[4] Magnus, J. R., Neudecker, H.(1988). *Matrix Differential Calculus with Applications in Statistics and Econometrics*, John Wiley & Sons.

[5] Ochiai, T., Nagatomo, K. (1989). *Linear Algebra with REDUCE* (in Japanese), Kindai-Kagaku Pub.

[6] Rao, C. R., Mitra, S. K. (1971). *Generalized Inverse of Matrices and its Applications*, John Wiley & Sons.

[7] Sasaki, T., Motoyoshi, F., S. Watanabe. (1985). *Computer Algebra* (in Japanese), Shoukou Pub.

Table 8: Functions of submatrices

Function	Arguments	Return-value
`row (M,i)`	`M`: matrix `i`: integer	the ith row of M
`col (M,i)`	`M`: matrix `i`: integer	the ith column of M
`uppersubmat (M,n)`	`M`: matrix `n`: integer	first n rows of M
`lowersubmat (M,n)`	`M`: matrix `n`: integer	last n rows of M
`removerow (M,i)`	`M`: matrix `i`: integer	M without ith row
`removecolumn (M,i)`	`M`: matrix `i`: integer	M without ith column
`swapcolumn (M,i,j)`	`M`: matrix `i,j`: integer	interchange of ith and jth columns of M
`swaprow (M,i,j)`	`M`: matrix `i,j`: integer	interchange of ith and jth row of M
`submatrix (M,L)`	`M`: matrix `L`: list (of lists) of integers	if `L` is a list of lists, {L1,L2}, then submatrix of M, made of rows L1 and columns L2; if `L` is a list, then submatrix M, made of rows and columns in L.
`primaldiagonal (M,n)`	`M`: matrix `n`: integer	nth primal diagonal matrix of M

Table 9: Functions to create matrices

Function	Arguments	Return-value
`setelmx (x,i,j,M)`	`x`: scalar `i,j`: integer `M`: matrix	replace (i,j) element of M by x
`zeromatrix (m,n)`	`m,n`: integer	$O_{m\times n}$
`ematrix (m,n,x,i,j)`	`m,n,i,j`: integer `x`: scalar	$O_{m\times n}$ with (i,j) element x; ≡ `setelmx (x,i,j,zeromatrix(m,n))`
`zerovec (n)`	`n`:integer	$1 \times n$ zero matrix
`evec (n,i)`	`n,i`: integer	ith unit vector of $1 \times n$
`diagmatrix (n,x)`	`n`: integer `x`: scalar	xI_n
`ident (n)`	`n`: integer	I_n
`setmatrix (a,m,n)`	`a`: id `m,n`: integer	$m \times n$ matrix (a_{ij})
`matrify (L)`	`L`: list (of lists)	matrix corresponding to `L`

Table 10: Miscellaneous functions to hack matrices

Function	Arguments	Return-value
`addrow (A,B)`	`A, B`: matrices	$(A \quad B)$
`addcol (A,B)`	`A, B`: matrices	$\begin{pmatrix} A \\ B \end{pmatrix}$
`addrows (M1,M2,..)`	list of matrices	$(M_1 \quad M_2 \quad \ldots)$
`addcols (M1,M2,..)`	list of matrices	$\begin{pmatrix} M_1 \\ M_2 \\ \vdots \end{pmatrix}$
`matsub (H, M)`	`H`: (list of) equations `M`: matrix	carry out substitutions H.

addrows and **addcols** are not genuine matrix-valued procedures. Their values should be assigned to some variables before being given to other matrix-handling procedures.

Table 11: Functions of matrix algebra

Function	Arguments	Return-value
`matpoly (M, L)`	`M`: matrix `L`: list,$\{\ell_1, \ldots, \ell_n\}$	$\ell_1 I + \ell_2 M + \cdots + \ell_n M^{n-1}$
`coefmatrix (U,V)`	`U`: list of equations or scalars `V`: list of kernels	coefficient matrix of U with respect to V
`quadfmatrix (q,v)`	`q`: scalar `v`: list of kernels	matrix for quadratic form Q with respect to V
`vec (M)`	`M`: matrix	row-wise vectorization of M
`hadamard (A,B)`	`A, B`: matrices	Hadamard product of A and B
`kronecker (A,B)`	`A, B`: matrices	Kronecker product of A and B
`mpinv (M)`	`M`: matrix	Moore-Penrose inverse of M
`ginv (M)`	`M`: matrix	generalised inverse of M