非数値行列処理言語 MPL-1 について*

三　島　健　稔**

Abstract

The idea of applying an digital computer to symbol manipulations has existed for some years. In symbol manipulations by an digital computer, but, arise some difficulties which have not made the idea put to practical performance.

The objective of this paper is to illustrate a simple but effective method decreasing the difficulties mentioned above and to show the details of MPL-1, the language proposed in this paper. In other words, by means of showing the facilities and simplicity of MPL-1, to point out the availabilty of the single purpose language particularly in symbol manipulations.

The prominent features of MPL-1 are:

(1) to be a single purpose language processing non-numeric matrix operations as well as numeric ones,

(2) the notations for writing of the opertions are the similar form of standard mathematical ones,

(3) MPL-1 requires non of special knowledges and techniques in use and the programs written in this language are readable with a little explanations, and

(4) the implementation of MPL-1 interpreter is easy.

1. まえがき

超高速大容量化したハードウェアと，充実したソフトウェアとを利用し，従来相当困難であった数値計算も比較的容易に処理できるようになってきた．

他方，電子計算機を非数値計算に適用することが試みられている．

非数値計算は

(1) 処理の結果得た式が，一般性をもっている．

(2) 誤差が生じない．

(3) 結果として得た式を利用して，数値計算を効果的に行なえる．

などの特長をもっている．しかし，非数値計算には，本質的に困難な問題があり，その実用化は，簡単な数式に関するものも容易でないとされている．

筆者は，非数値計算に関する基本的問題が，行列の演算に典型的な形で現われることに着目し，行列を対象にとり，非数値計算の研究を進めてきた．そして，最も単純ではあるが，最も効果的と思える原理に基づき，特殊目的言語を作成することが，非数値計算を可能にすると考えるに至った．本稿では，その原理を，行列処理言語に適用し，簡単な，非数値行列処理言語 MPL-1 を設計し，そのインタープリタを作成する．MPL-1 では，数値および非数値要素をもつ行列の演算を，丁度，算法言語を用いて，簡単な四則演算を記述するような形式で表現し，処理することを意図した．

2. 非数値計算の問題点と簡単な対処法

2.1 問題点とその性質

非数値計算を困難にしている原因の一つは，そのデータの性質にある．数値計算においては，周知のように，1回の処理で得た結果は，一般に，1個の数値で表わされる．しかし，非数値計算においては，このようなことはまず期待できない．逆に，処理するたびにデータは急速に増大し，かつ，複雑なものになる．このことが，非数値計算を困難にする大きな問題である．これは，次のような形で現われる．まず，結果として得た式を記憶するため，莫大な記憶容量が必要である．また，結果の数式が，そのままでは非常に長く，見に

* On a Non-Numeric Matrix Processing Language MPL-1, by Taketoshi MISHIMA (Faculty of Engineering, Meiji University)

** 明治大学工学部

くい．このため，数式をできる限り単純にしなければならない．しかし，これは人間の高度の知力に関係する問題であって，現在，機械で処理することは非常に困難である．式の等値性の判別も同種の問題であって，いまのところ，定式化された解決方法はないようである．

さらに，以上のことから明らかなように，一般に，処理された中間結果が簡単にならないため，超高速度大型計算機を用いて処理できたとしても長時間を必要とし，かなりの費用を要する．

これらの問題は，現在，電子計算機の認識力が，本質的に一次元的認識手段に依存していることに起因すると考えられ，人工知能研究の他の分野の困難な問題と同種の性質をもつものと考えられる．すなわち，非数値計算に現われる基本的問題の根本的解決は，人間特有の知力を生ぜしめる機構が解明され，あるいは，それと等価なモデルが開発され，かつ，それが物理的に実現されるまで与えられないと思われる．

2.2 簡単な対処法

2.1 に示したように，非数値計算の問題は，電子計算機に，現在，効率よい知力をもたせることができないことから生ずる．したがって，人間の知力と電子計算機の能力を融合する対話型システムは，一般にこのような問題に対する有効な解決手段となる．

しかし，たとえば行列の演算のように，データの膨張速度が早く，かつ，処理回数の多い問題に対しては疑問である．なんとなれば，計算機の能力を活用しようとして，逆に人間の側で，最初より大量の複雑なデータを処理しなければならないことになるからである．大量のデータに単純な操作を繰り返すことは，人間の苦手とするところであり，それだけに間違いも生じやすい．筆者は，非数値計算を現在の計算機で行なうには，結局，処理の対象を同じような性質をもつものに固定し，簡単な表現形式をもつ単目的言語の作成する方法が最も簡単で，かつ，最も有効と考える．

この方法には，次のような特長がある．

（1） 言語プロセッサに，人間のすぐれた知識をもたせやすくなる．
（2） 処理の冗長度をかなり削減できる．
（3） 言語プロセッサの実現が容易である．
（4） 簡単に利用できる．

その反面

（1） 処理対象が制限される．
（2） 多くの言語が必要となる．

などの欠点もある．

多くの言語を覚えるのはめんどうであるが，この場合，一言語の処理対象が狭いので，慣用的表現に統一しやすく，おのおの言語を覚えるめんどうは相当減るであろう．

3. 行列処理の試み

2.2 の方法を，非数値要素行列の演算を対象として具体化する．

3.1 言語設計の視点

行列の処理に関して，現在の算法言語は，(1) 行列を一つの単位として扱えない．(2) 演算の記述は，各要素ごとの演算を記述することで行なう．サブルーチンを利用するにしても不自然な表現になる．(3) 演算に不必要な，計算機に関する指令にまで関与しなければならない場合が多い，などの難点をもつといわれている[4]．MPL-1 では，以上の難点や 2.1 の問題を考慮し，設計にあたり以下のような目標を設けた．

（1） 通常用いられている標準的・数学的表現をできる限り採用する．
（2） 行列を一つの単位として扱えること．
（3） 行列の構造を自由に表現できること．
（4） 非数値要素の行列と数値要素の行列とを全く同じように処理できること．
（5） 単純化の問題には，なるべく関与しないで利用できること．

4. MPL-1 の構成

4.1 basic symbol, identifier, string について

4.1.1 basic symbol

⟨basic symbol⟩ : : = ⟨letter⟩|⟨digit⟩|⟨delimiter⟩|⟨sign⟩|$

basic symbol は，バッカス記法で上記のように表わせる．

basic symbol は，基準語の構成に用いられるが，固有の意味はもっていない．

4.1.2 letter

letter は，英大文字 26 字および特殊文字，＊，/，＝，＃，&，@，；，：，，，％，〈，〉，↑からなる．

letter は，identifier や string を構成するために用いられ，固有の意味はもっていない*．

4.1.3 sign

⟨sign⟩ : : = ＋|－

* 以下，便宜上，I は例外として虚数単位を表わすものとする．

sign は identifier, stringの構成要素としても用いられるが，おもに符号として用いる．

4.1.4 digit

⟨digit⟩::=0|1|2|3|4|5|6|7|8|9

digit は number, identifier, string の構成に用いる．

4.1.5 delimiter

⟨delimiter⟩::=,|.|(|)|␣

delimiter は区切りを表わす．␣ はブランクを示すものとし，以後便宜上␣で1個以上の␣を示すものとする．

4.1.6 identifier

⟨identifier⟩::=⟨letters except *I*⟩|⟨sign⟩|$|$⟨identifier part⟩|⟨letter⟩⟨identifier part⟩

⟨identifier part⟩::=⟨character⟩|⟨character⟩⟨identifier part⟩

⟨character⟩::=⟨letter⟩|⟨digit⟩|⟨sign⟩

identifier は，variable や procedure を識別するために用いる．

4.1.7 string

⟨string⟩::=⟨basic symbol⟩|⟨basic symbol⟩⟨string⟩

4.1.8 $ bracket

⟨$ bracket⟩::=$$⟨a certain basic symbol *X*⟩⟨string⟩*X*

$ bracket により任意の string を構成できる．

4.2 number

⟨octal digit⟩::=0|1|2|3|4|5|6|7

⟨unsigned octal integer⟩::=⟨octal digit⟩|⟨octal digit⟩⟨unsigned octal integer⟩

⟨unsigned integer⟩::=⟨digit⟩|⟨digit⟩⟨unsigned integer⟩

⟨non-negative integer⟩::=⟨unsigned integer⟩|+⟨unsigned integer⟩

⟨integer⟩::=⟨unsigned integer⟩|⟨sign⟩⟨unsigned integer⟩

⟨decimal fraction⟩::=.⟨unsigred integer⟩

⟨decimal exponent part⟩::=*E*⟨integer⟩

⟨octal exponent part⟩::=*Q*⟨non-negative integer⟩|*Q*

⟨decimal real⟩::=⟨integer⟩⟨decimal fraction⟩|⟨integer⟩⟨decimal fraction⟩⟨decimal exponent part⟩

⟨decimal integer⟩::=⟨integer⟩|⟨integer⟩*E*⟨non-negative integer⟩

⟨octal integer⟩::=⟨sign⟩⟨unsigned octal integer⟩⟨octal exponent part⟩|⟨unsigned octal integer⟩⟨octal exponent part⟩

⟨real number⟩::=⟨decimal real⟩|⟨decimal integer⟩|⟨octal integer⟩

⟨complex number⟩::=(⟨real number⟩␣+␣⟨real number⟩␣*␣*I*)|(⟨real number⟩␣*␣*I*)|*I*|(⟨variable⟩␣+␣⟨variable⟩␣*␣*I*)

⟨number⟩::=⟨real number⟩|⟨complex number⟩

decimal integer は，-2^{63} から $2^{63}-1$ の範囲の数，すなわち，絶対値が 10^{19} 以内で，底は 10，指数は 0 から 18 以内である．

octal integer は，絶対値が 8^{22} 以内の数で，底は 8，指数は 10 進数で 0～21 以内である．

decimal real は，整数部と小数部のけた数の和が 16 以内で，絶対値 0 および 10^{-38}～10^{38} 以内の数である．指数は，10 を底とする．

4.3 expression

計算過程は arithmetic expression で記述する．

E-arithmetic expression は，行列の要素として用いられるもので，形式は arithmetic expression と同じである．

4.3.1 variable

⟨variable⟩::=⟨identifier⟩

variable は，arithmetic expression, *E*-arithmetic expression の変数として用いる．

4.3.2 function designator

⟨procedure identifier⟩::=⟨identifier⟩

⟨actual parameter⟩::=⟨variable⟩|⟨function designator⟩|(⟨arithmetic expression⟩)

⟨argument⟩::=⟨argument part⟩|empty

⟨argument part⟩::=⟨actual parameter⟩|⟨actual parameter⟩␣⟨argument part⟩

⟨function designator⟩::=⟨procedure identifier⟩␣(⟨argument⟩)

利用者は，function designator によって，standard function や defined function を利用できる．

4.3.3 arithmetic expression

⟨term⟩::=⟨number⟩|⟨scalar constant⟩|⟨variable⟩|⟨function designator⟩|(⟨arithmetic expression⟩)

⟨scalar constant⟩::=⟨the identifier defined by scalar definition⟩

⟨arithmetic expression⟩::=⟨term⟩␣⟨operator⟩

␣⟨term⟩|⟨term⟩␣|⟨operator⟩␣⟨arithmetic expression⟩

⟨operator⟩::=＊|＋|－

⟨matrix⟩::=((⟨element list⟩)⟨matrix part⟩)

⟨element list⟩::⟨element⟩|⟨element⟩␣⟨element list⟩

⟨matrix part⟩::=␣(⟨element list⟩)|␣(⟨element list⟩)⟨matrix part⟩|empty

⟨element⟩::=⟨variable⟩|⟨number⟩|(⟨*E*-arithmetic expression⟩)

arithmetic expression は，definition statement で用いられる．また，operator の優先順序，意味は，**Table 1** に示すとおりである．

Table 1 The meaning and priority of operators

operator	優先順序	意味
＊	1	行列の積および行列とスカラーとの積
＋	2	行列の和
－	2	行列の差

4.3.4 *E*-arithmetic expression

E-arithmetic expression は，行列の要素の記述に用いられるもので，構成は arithmetic expression と同形である．ただし，operator の意味が異なり，*E*-operator と呼ぶことにする．意味，優先順序は **Table 2** に示すとおりである．

Table 2 The meaning and priority of *E*-operators

E-operator	優先順序	意味
＊	1	スカラーの乗法
／	1	スカラーの除法
＋	2	スカラーの加法
－	2	スカラーの減法

4.4 standard function

MPL-1 standard function として，**Table 3** に示す関数が登録されている．

4.5 out-put function

出力関数としては，**Table 4** に示す PRINT, FLPRINT が準備されている．

4.6 erase

erase ($X1$ $X2 \ldots XN$)

N 引数関数 erase は，各引数の値を消し，記憶容量を有効に利用するために用いる．

4.7 defined function

definition statement で定義した関数．

Table 3 MPL-1 standard functions

関数名	引数の個数	内容
TRANS (X)	1	X^r を与える
INVM (X)	1	X^{-1} を与える
DETM (X)	1	$\|X\|$ を展開する
ELM (X I J)	3	X_{IJ} を与える
SN 1 (X)	1	(m, n) 行列 X の $(m-1, n-1)$ 行列を与える
MALPHA 12 (X)	1	(m, n) 行列 X の $\langle X_{1n}X_{2n}\ldots X_{mn-1})^r$ なる列ベクトルを与える
MALPHA 21 (X)	1	(m, n) 行列 X の $(X_{m1}X_{m2}\ldots X_{mn-1})^r$ なる行ベクトルを与える
SMAT ($X1$ $X2$ $X3$ $X4$)	4	部分行列 $X1$, $X2$, $X3$, $X4$ から行列を合成する
COL (X N)	2	X の第 N 列
ROW (X N)	2	X の第 N 行
SPUR (X)	1	X の spur
SIMUE (A B)	2	方程式 $AX=B$ の解
MINOR (X I J M N)	5	X_{IJ} を (1, 1) 成分とし，M, N を行，列の次数とする (NM) 小行列を与える
EIGEN (X)	1	X の固有多項式を与える
TELM (X)	1	X の対角成分のリスト
EXCOL (X I J)	3	X の I 列と J 列を入れ替える
EXROW (X I J)	3	X の I 行と J 行を入れ替える
ELMEX (X I J Y)	4	X_{IJ} を Y と入れ替える
EMAT (X N)	1	N 次単位行列を作る
INPROD (X Y)	2	ベクトル X, Y の内積
EXPROD (X Y)	2	三次元ベクトル X, Y の外積
INVERSVECTOR (X)	1	ベクトル X の逆ベクトル

Table 4 Out-put functions

関数名	引数の数	内容
PRINT (X)	1	引数の値を印字する
FLPRINT (X)	1	引数の値をトップレベルごとに印字する

4.8 statement

⟨statement⟩ ::=⟨assignment statement⟩|⟨procedure statement⟩|⟨definition statement⟩

処理される言語の単位を statement と呼ぶ．

4.8.1 assignment statement

⟨assignment statement⟩::=⟨variable⟩␣:=␣⟨expression part⟩

⟨expression part⟩::=⟨function designator⟩(⟨*E*-expression part⟩)|⟨matrix⟩

⟨*E*-expression part⟩::=⟨*E*-term⟩␣⟨*E*-operator⟩␣⟨*E*-arithmetic expression⟩

assignment statement は，行列や式を1つの記号で代用するために用いる．

4.8.2 definition statement

⟨definition statement⟩::=⟨identifier⟩␣(⟨variable list⟩)␣=␣(⟨arithmetic expression⟩)

⟨variable list⟩::=⟨variable⟩|⟨variable⟩␣⟨variable list⟩

definition statement で関数を定義し，function

designator でそれを利用する.

4.8.3 procedure statement

⟨procedure statement⟩ : : =⟨procedure identifier⟩␣(⟨argument⟩)

⟨procedure identifier⟩ : : =⟨identifier⟩

⟨argument⟩ : : =⟨argument part⟩|empty

⟨argument part⟩ : : =⟨actual parameter⟩|⟨actual parameter⟩␣⟨argument part⟩

⟨actual parameter⟩ : : =⟨variable⟩|⟨number⟩|⟨matrix⟩|⟨function designator⟩|⟨arithmetic expression⟩

4.9 declaration

MPL-1 にはスカラーの宣言と関数名の宣言とがある.

4.9.1 scalar declaration

⟨scalar declaration⟩ : : =SCALAR␣(⟨identifier list⟩)

definition statement で用いるスカラーを宣言する. 数の場合は必要ない.

4.9.2 funname declaration

⟨funname declaration⟩ : : =FUNNAME␣(⟨identifier list⟩)

definition statement で定義する関数につける名前を宣言する. これは, 関数の引数として, defined function を用いる場合に必要となるもので, そのような用法をしない場合は必要ないものである.

4.10 program

statement を S, declaration を D とすると MPL-1 プログラムは, 次のいづれかの型をとる.

(1) $S␣S␣...␣S␣$END␣

(2) $D␣S␣...␣S␣$END␣

ただし, Dは必要に応じて用いればよい 先頭にくる必要はない.

5. MPL-1 インタープリタ

5.1 流れ図

MPL-1 プログラムは, MPL-1 インタープリタによって処理される.

MPL-1 インタープリタは, **Fig. 1** のような簡単な流れ図によって示される (次ページ参照).

5.2 消去関数 erase について

MPL-1 は, 非数値処理言語の1つであって, 記憶容量の節約は, 大きな問題である.

MPL-1 インタープリタは, 数式の処理を行ないながら自動的に, その結果に単純化の処理を行なう.

一方, erase は利用者が簡単に記憶容量の節約を行なえるように準備された関数で, 次のような消去処理を行なうものである.

(1) HELP[1)2)] の定数として登録されたアトムのインディケータとその値を取り去る.

(2) 行列や行列の要素を assign したアトムのインディケータとその値を取り去る.

(3) association list fm 926 にある関数名とその値を取り去る.

(4) HELP の association list にあるドット対の右半分を NIL にする.

これらの処理によって, 不要になった莫大な記憶容量の再生が可能になる.

(1), (2) は property list に, (3), (4) は association list に作用する.

5.3 association list fm 926 について

HELP インタープリタ evalquote は, apply および eval により, association list, property list を利用して, データの評価を行なう.

したがって, 評価に必要なものは, 前もって association list か property list に準備されなければならない. これらは通常 HELP インタープリタが行なうが, 一部 MPL-1 インタープリタで操作する必要が生じた. しかし, association list は, 全く瞬間的なものであり, 外部から必要な情報を直接付加しても, インタープリタ evalquote によって取り去られる. また, いたずらに操作すると, インタープリタの機能をそこなう心配もある.

property list の操作によるとすると, 使用する記号に細心の注意が必要になったり, HELP インタープリタそのものを作りかえなければならなくなりめんどうである. association list fm 926 はこのために作られたもので, 次のような操作を行なうことで, HELP association list と同様に利用できる.

(1) fm 926 のターミネイタを取り去り, 代わりに, current association list をさすポインタを入れる.

(2) aset (fm 926 に必要なものを登録する関数) を処理するためのassociation list を作る.

(3) 必要に応じて fm 926 を参照して, aset に必要な評価を行なう.

(4) aset により fm 926 に値を登録する.

(5) (1) の逆操作を行なう.

(2) で作られた association list は, HELP インタ

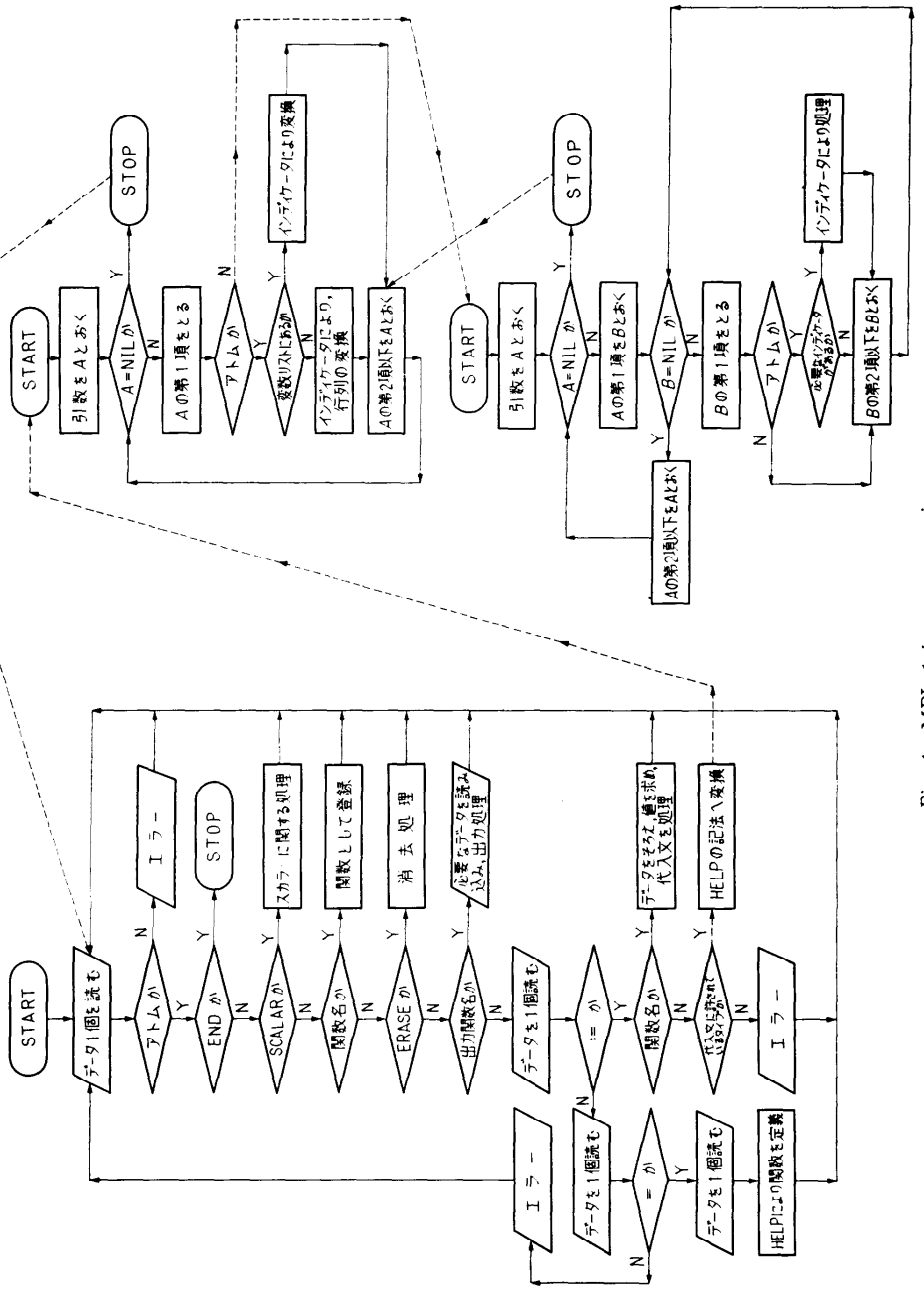

Fig. 1 MPL-1 interpreter main

ープリタによって破壊されているので，MPL-1 インタープリタで操作する心要はない．

fm 926 は，assignment statement で必要になったものであるが，assignment statement をすべて fm 926 で扱っているわけではなく property list の操作のみで処理されているものもある．

5.4 単純化について[3),5),6)]

非数値処理を行なう際，数式の単純化が大きな意味をもつことは周知のことであるが，いまのところ定式化された有力な方法はないようである．

MPL-1 においては，文献 3) に報告してある程度の単純化がなされている．単純化は，必要に応じて，処理の都度 MPL-1 によって自動的に行なわれ，利用者は全く関与する必要のないことが望ましいが，利用者が積極的に関与することで，多くの記憶容量の再生が簡単に行なえる場合がある．このような場合のために，erase が用意されている．erase が扱っている再生は，外部からなんらかの形で指示しなければ不可能な性質のものであり，このような部分に関しては，利用者が簡単に関与できる手段がぜひ必要である．

6. 使 用 例

問 1. 次の $\boldsymbol{A}$, $\boldsymbol{B}$ に対し，$\boldsymbol{F}$ を求めよ．

$$\boldsymbol{A}=\begin{pmatrix} a_{11}+a_{12}\times a_{13} & a_{14} \\ a_{21} & a_{11}+a_{12}\times a_{13} \end{pmatrix}$$

$$\boldsymbol{B}=\begin{pmatrix} b_{11} & b_{21}-b_{13}\div b_{14} \\ b_{21}-b_{13}\div b_{14} & b_{24} \end{pmatrix}$$

$$\boldsymbol{F}=\boldsymbol{A}(\boldsymbol{A}+(\boldsymbol{B}-\boldsymbol{A})\boldsymbol{B})$$

プログラム例

```
(1)*  F(A B)=(A*(A+(B-A)*B))
(2)   VA:=(A11+A12*A13)
(3)   VB:=(B21-B13/B14)
(4)   A:=((VA A14) (A21 VA))
(5)   B:=((B11 VB) (VB B24))
(6)   F1:=F(A B)
(7)   PRINT ($$$THE VALUE OF F$)
(8)   FLPRINT (F1)
(9)   END
```

解 説

(1) において，関数 F を定義している．

(2)，(3) では，行列の要素として重複して現われる算術式をそれぞれ VA, VB に assign している．要素は本例のように算術式であってもよい．また，(1) における＊，＋，－は，ここでは行列の積，和，差を表わすが，(2) においては，スカラーに関する四則演算を示している．

(4)，(5) はそれぞれ二行二列の行列に A, B なる名前を付けている．以後，A, B はおのおのの行列と解釈される．また，関数 $F(A\ B)$ の A, B とこの A, B とは無関係である．

(6) において，(4)，(5) で assign した A, B を関数 F の引数として F の値を求めている．値は $F1$ に assign される．

(7) で見出しを印字する．実施例では $ が ¥ になっているが，これは計算機の都合によるもので特に意味はない．(7) は $ bracket の使用例である．

(8) で $F1$ の値を行ごとに印字させる．

(9) END はプログラムの終了を示す．

出力の解説

第1行はコメント．プログラム例には示さなかったがカードの第1列にCと書けばその行はコメントとみなされる．第1行から第8行まではカードイメージ．THE VALUE OF F は，第8行の PRINT の実行結果．このようにカードイメージが出るので見出しはコメントによる方がよい．第10行はカードイメージ．11行以下 END の前までが，10行 FLPRINT の値．途中，(が飛び出しているところが第二行の初めを示している．最後に END を印字し終了．

問 2. Q, R はスカラー

$$\boldsymbol{H}=(10),\quad \boldsymbol{A}=\begin{pmatrix} A & B \\ 0 & 0 \end{pmatrix},\quad \boldsymbol{B}=\begin{pmatrix} C \\ 0 \end{pmatrix},$$

$$\boldsymbol{P}=\begin{pmatrix} P11 & P12 \\ P21 & P22 \end{pmatrix}$$

として，N, S を求めよ．

$$\boldsymbol{N}=\boldsymbol{A}\boldsymbol{P}\boldsymbol{A}^{T}+\boldsymbol{B}\boldsymbol{B}^{T}Q$$

$$\boldsymbol{S}=\boldsymbol{N}\boldsymbol{H}^{T}(\boldsymbol{H}\boldsymbol{N}\boldsymbol{H}^{T}+R)^{-1}\boldsymbol{H}\boldsymbol{N}$$

解 説

Q, R がスカラーであることを

SCALAR (Q R)

によって宣言する．定義文中で用いるスカラー以外は宣言する必要はない．

＋，－，＊の operator は，スカラー，行列いずれの演算の記述にも用いられるので，組込み関数，TRANS, INVM を用いて関数 S, N を定義できる．以上のことを知って問1にならって書いたプログラムの実施例は，次のとおりである．

問1の実施例の見方にならってみると EXAMPLE 2も容易に理解できる．

ここに，たとえば N の値のように通常

$$\begin{pmatrix} A\cdot(A\cdot P11+B\cdot P21) & 0 \\ +B\cdot(A\cdot P12+B\cdot P22)+C\cdot C\cdot Q & 0 \\ 0 & 0 \end{pmatrix}$$

と表わされる式の出力が Polish prefix notation になっているが，infix notation への変換は簡単である．

またこの程度の問題は，コアアクセスタイム 2μ秒の計算機を用いて（free storage 18K 語，full-word space 2K 語として）数分のオーダで処理できる．

---

* この (1)～(9) はプログラムの説明上付したものでプログラムの一部ではない．

```
C EXAMPLE 1
 F (A B) = (A * (A + (B - A) * B))
 VA := (A11 + A12 * A13)
 VB := (B21 - B13 / B14)
 A  := ((VA A14) (A21 VA))
 B  := ((B11 VB) (VB B24))
 F1 := F (A B)
 PRINT (¥¥¥THE VALUE OF F¥)
THE VALUE OF F
 FLPRINT (F1)
((+ (* (+ A11 (* A12 A13)) (+ A11 (* A12 A13)
 (* B11 (+ B11 (* -1 (+ A11 (* A12 A13)))))
 (* (+ B21 (/ (* -1 B13) B14)) (+ B21 (/ (* -1 B13) B14) (* -1 A14)))))
  (* A14 (+ A21 (* B11 (+ B21 (/ (* -1 B13) B14) (* -1 A21))))
 (* (+ B21 (/ (* -1 B13) B14)) (+ B24 (* -1 (+ A11 (* A12 A13))))))))
 (+ (* (+ A11 (* A12 A13)) (+ A14 (* (+ B21 (/ (* -1 B13) B14))
 (+ B11 (* -1 (+ A11 (* A12 A13))))) (* B24
 (+ B21 (/ (* -1 B13) B14) (* -1 A14)))))
 (* (+ A11 (* A12 A13) (* (+ B21 (/ (* -1 B13) B14))
 (+ B21 (/ (* -1 B13) B14) (* -1 A21))) (* B24
 (+ B24 (* -1 (+ A11 (* A12 A13)))))) A14)))
((+ (* A21 (+ A11 (* A12 A13) (* B11 (+ B11 (* -1 (+ A11 (* A12 A13)))))
  (* (+ B21 (/ (* -1 B13) B14)) (+ B21 (/ (* -1 B13) B14) (* -1 A14))))
 ) (* (+ A11 (* A12 A13)) (+ A21 (* B11 (+ B21 (/ (* -1 B13) B14)
 (* -1 A21))) (* (+ B21 (/ (* -1 B13) B14))
 (+ B24 (* -1 (+ A11 (* A12 A13)))))))) (+
 (* A21 (+ A14 (* (+ B21 (/ (* -1 B13) B14))
 (+ B11 (* -1 (+ A11 (* A12 A13))))) (* B24
 (+ B21 (/ (* -1 B13) B14) (* -1 A14)))))
 (* (+ A11 (* A12 A13) (* (+ B21 (/ (* -1 B13) B14))
 (+ B21 (/ (* -1 B13) B14) (* -1 A21))) (* B24
 (+ B24 (* -1 (+ A11 (* A12 A13)))))) (+ A11 (* A12 A13)))))
 END
```

EXAMPLE 1

```
C   EXAMPLE 2
 SCALAR (Q R)
 S (N1 H TH) = (N1 - TH * INVM ((H * N1 * TH + R)) * H * N1)
 N (A P B) = (A * P * TRANS (A) + B * TRANS (B) * Q)
 H := ((1 0))
 A := ((A B) (0 0))
 B := ((C) (0))
 P := ((P11 P12) (P21 P22))
 (H := TRANS (H)
 N1 := N (A P B)
 S1 := S (N1 H TH)
 PRINT (¥¥¥THE VALUE OF N¥)
THE VALUE OF N
 FLPRINT (N1)
((+ (* A (+ (* A P11) (* B P21))) (* B (+ (* A P12) (* B P22)))
 (* C C Q)) +0)
(+0 +0)
 PRINT (¥¥¥THE VALUE OF S¥)
THE VALUE OF S
 FLPRINT (S1)
(((/ (+ (* -1 (+ (* A (+ (* A P11) (* B P21)))
 (* B (+ (* A P12) (* B P22))) (* C C Q)))
 (* (+ H (* A (+ (* A P11) (* B P21))) (* B (+ (* A P12) (* B P22)))
 (* C C Q)) (+ (* A (+ (* A P11) (* B P21)))
 (* B (+ (* A P12) (* B P22))) (* C C Q))))
 (+ H (* A (+ (* A P11) (* B P21))) (* B (+ (* A P12) (* B P22)))
 (* C C Q))) +0)
(+0 +0)
 END
```

EXAMPLE 2

7. むすび

以上により非数値計算の実用化の方向を示し，MPL-1 による実験でその有効性を示したつもりである．

MPL-1 に見るように，非数値計算を行なうにあたって単目的言語を作成する方法は，implementation が容易であり，かつ，非数値計算の困難な問題を相当削減してくれる．非数値計算においては，統一した記法をもつおのおのの問題向き言語を準備し，適宜それを利用することが最善の実用化法であろう．これは，一般的記述言語の利用は，特別の興味をもっている者でないと容易でないことや，単純化の問題が表面に出てきて，その処理が困難なことなどによる．

8. 謝 辞

常日ごろ懇切なるご指導を賜わっている明治大学後藤以紀教授に深く感謝の意を表します．また，本研究にご協力をいただいた日立中研の吉村一馬 104 室長，二村良彦氏，橋本三枝氏に感謝する次第である．

参 考 文 献

1) J. McCarthy and others: "LISP 1.5 Programmers' Manual" MIT Press, August 1962.
2) "HELP Manual" Hitachi Ltd., 1968 年 7 月.
3) 雨宮，他："逆行列および行列方程式の非数値解法" 第10回プログラミングシンポジウム報告集，情報処理学会，1969年 1 月.
4) 戸川隼人："マトリックス計算のためのプログラム言語について" 第 7 回プログラミングシンポジウム報告集，情報処理学会，1966年 1 月.
5) Dean Wooldlidge Jr.: "An Algebraic Simplify Program in LISP" Stanford Artificial Intelligence Project Memo No. 11, December 1963.
6) Daniel G. Bobrow: "Symbol Manipulation Languages and Techniques" North Holland, 1968.

(昭和 45 年 10 月 2 日受付)