Lecture Notes in Control and Information Sciences

Edited by A.V. Balakrishnan and M.Thoma

63

Analysis and Optimization of Systems

Proceedings of the Sixth International Conference on Analysis and Optimization of Systems
Nice, June 19–22, 1984

Part 2

Edited by
A. Bensoussan and J. L. Lions

Springer-Verlag
Berlin Heidelberg New York Tokyo 1984

IMPACT

Interactive Mathematical Program for Automatic Control Theory

Magnus Rimvall
François Cellier
Institute for Automatic Control
Swiss Federal Institute of Technology (ETH)
CH-8092 Zuerich, Switzerland
Tel. 01 / 256 28 42

Abstract.

IMPACT, a new CAD-program for Control Systems which is presently under development at our institute, is presented. The program will give access to algorithms useful in control systems theory in an interactive manner. It is aimed at inexperienced students as well as skilled control scientists for the analysis, synthesis and simulation of control systems. IMPACT is coded in ADA, portability is one of the main design goals.

A first section discusses the chosen mode of interaction, and compares it with other common methods. A second section presents the data structures available in IMPACT, and discusses the operations which can be performed on these structures. The IMPACT command language is thereafter presented, in particular are the very versatile macro-facilities explained. Finally, some implementational aspects are discussed.

1. INTRODUCTION

In the last decades, digital computers have thoroughly changed the computational tools used by control engineers. However, this revolution is not yet over, its thrust has just shifted from the point of raw computing power to the question of user friendly and adaptive systems.

Let us look at the (nowadays) simple problem of calculating the eigenvectors and inverse eigenmatrix of several 8*8 matrices. Forty years ago, you would need a lot of paper and almost unlimited patience to solve this problem.

Twenty years ago, you probably had a digital computer at your disposal. However, you would most likely have to write a program yourself, which calculated the eigenvectors and inverted the eigenvector-matrix. Only if you were extremely lucky, you might have had access to one of the first libraries containing general-purpose programs for mathematical operations (e.g. SSP /SSP68/).

Ten years ago, you most certainly had access to some library containing mathematical algorithms, e.g. IMSL /IMSL82/, EISPACK /GARB77/ /SMIT74/ or LINPACK /DUNG79/. Unfortunately, you still had to write a program which read the matrices, called the algorithm-routine(s) and printed the result. You were bound to loose a lot of time until the input format corresponded to the input data, all the parameters of the library calls were correct and in the right order, and so on.

Today, for most people, things are not that much different. New and better algorithms have emerged, but you still loose lots of time writing programs accessing these algorithms. Needed is a package adapted to control theory, which has not only a software interface, but also an interactive interface for easy access.

1.1 MATLAB

One of the first persons to realize the importance of an interactive interface to packages containing complex mathematical algorithms was C. Moler /MOLE80/. In his program MATLAB, a milestone in the history of interactive programs, an easy-to-use, interactive interface is provided to the LINPACK and EISPACK matrix manipulation libraries. Using a very natural input command language, it is possible to perform matrix operations in MATLAB with the same ease as one makes scalar computations on a pocket calculator. For example can the above mentioned problem be solved in a few lines of input:

```
A = <1, 4, 5, 0, 0, -1, 3, 1
     0, 1, 0, 1, 0,  0, 0, 0
       ...
    -3, 0, 0, 0, -1, 0, 0, 2>;
<P,DUM> = EIG(A);
INVP = INV(P)
```

Although MATLAB is, as we have seen, extremely easy to use, and moreover very versatile, many problems in control theory are not (and were never intended to be) solvable using MATLAB. One reason for this is of course the lack of suitable algorithms, another, more fundamental cause is the lack of data structures.

As MATLAB is written in a very well-structured manner, it is quite simple to add new algorithms. MATRIX$_x$ is one such extended MATLAB package for control scientists /WALK82/, another extended version of MATLAB is presently under development by the group of P. van Dooren (unpublished). Although these new products are definite upgradings of

MATLAB, they only partially provide the control engineer with an adequate tool. The reason for this is the mentioned lack of data structures adapted to control problems. MATLAB uses the complex matrix (with the scalar as a special case) as the only data structure. Control engineers often work with more complex structures, like polynomial matrices, transfer-function matrices and linear as well as non-linear system descriptions. Furthermore, although the input command language of MATLAB is well suited for smaller problems, a better structured command language is needed for more complex problems. In particular, some form of macro/procedure facility accepting parameters must be available. Finally, versatile graphical output and an interface to a data base should be present.

1.2 IMPACT

Seen through the eyes of the user, IMPACT appears to be just another extension to MATLAB, the IMPACT command language is similar to that of MATLAB. However, seen from an implementational view, IMPACT is only a conceptual superset of MATLAB. As IMPACT is implemented in ADA, not one single line of code has been taken from MATLAB. Furthermore, several new data structures are introduced.

The development of IMPACT is made with the objective of serving a very innomogenous group of users. On one hand, IMPACT is aimed at being used by students with little experience in control theory and no experience at all in CAD. Using only the most basic structures of the input command language which are simple enough to be learnt in a few hours, these students will be able to access complex algorithms in order to solve control problems with a minimum of tutorial. An on-line HELP facility contains all needed information on the command language syntax as well as on the numerical algorithms, making self-tutorial possible. On the other hand, the experienced control scientist is provided with a full-fledged structured command language with all elements found in a higher computer language including WHILE and FOR loops, IF-THEN-ELSE statements, and so forth. Furthermore, a large selection of IMPACT functions and procedures gives the user access to a wide range of algorithms. To further enhance the structurability, four different macro facilities have been introduced. Access to a data-base will be provided to store away variables, plots and macros for later reuse.

2. MODE OF INTERACTION

When designing a new interactive system, one of the first decisions must be, in which form the man-machine interaction is to take place. This decision should not be taken lightheartedly, as this mode of interaction determines the user-friendlyness, and thereby also user acceptance, of the system; although this interface is not the brain of any CAD-system, it certainly serves as both eyes and mouth.

The mode of interaction also influences the structure of the kernel controlling the package. In particular, the data-structures of the kernel are very closely knitted to the user interface. As any late changes in the central data structures are the worst of all possible nightmares for any software developer, the design of the interactive interface should be done carefully, so that no later modifications need to be done /INF079/.

Apart from some more exotic ways of communication, like speech input and natural language input, four basically different ways of interactive input exist:

- question-and-answer method
- menu-driven operation
- command-language communication
- graphical input.

Of these four, the first three work with alphanumeric information, which, at least in principle can work on any alphanumerical terminal. The graphical input requires special hardware in form of a graphical terminal for the graphic echo and some graphical input device (in form of e.g. a joystick or a lightpen). Although very interesting in the field of system documentation and modelling /ELMQ82/, where structures rather than numerical data are entered, the complexity of a program allowing such an input makes it unattainable for a CACSD system designed primarily to solve numerical problems. However, a later interface between IMPACT and a graphical input system would be an interesting and meaningful extension.

Of the three alphanumeric input modes, the menu and the question-and-answer methods let the computer be in charge of the conversation, whereas the command language method gives the user almost total control. In the last few years, several menu-driven interactive programs using a "mouse" or joystick as input device have emerged. These systems

are extremely handy and speedy for dexterous users. However, until a hardware standard has been set, such systems are not very portable. On the other hand, the speed of portable menu-driven systems can be compared with that obtained by question-and-answer methods.

If the only design goals are minimal learning time and maximum accessibility by non-specialists, the question-and-answer method is most certainly the right answer. However, this method gets very tiresome after a while, as the user always can anticipate the next question, but cannot speed up the input. As an example, compare the conversation in INTOPS /AGAT79/ (a somewhat old-fashioned interactive program for control system operations) with the equivalent commands in IMPACT. Both examples produce a BODE diagram of the transfer function

$$G(s) = \frac{1}{s^3 + 9s^2 + 5s + 9}$$

INTOPS question-and-answer conversation:

```
P>      OP CODE = ENTER
P>      NAME = NUME
P>      COMMENT = NUMERATOR
P>      ORDER = 0
P>      P( 0) = 1
P>      NUME        NUMERATOR
P>      P( 0) =  0.10000E+01
P>      OP CODE = ENTER
P>      NAME = DENO
P>      COMMENT = DENOMINATOR
P>      ORDER = 3
P>      P( 0) = 9
P>      P( 1) = 5
P>      P( 2) = 9
P>      P( 3) = 1
P>      DENO        DENOMINATOR
P>      P( 0) =  0.90000E+01
P>      P( 1) =  0.50000E+01
P>      P( 2) =  0.90000E+01
P>      P( 3) =  0.10000E+01
P>      OP CODE = BODE
P>      GH NUMERATOR = NUME
P>      GH DENOMINATOR = DENO
P>      NUMBER OF FREQUENCY VALUE = 100
P>      OMEMAX = 1000.
P>      OMEMIN = .1
```

(user input is underlined).

Command language input of IMPACT:

```
BODE (1/<9^5^9^1> //DOMAIN=LOGDOM(.1,1000.,100) )
```

or a little more verbose but better readable:

```
S = <^1>;
G = 1 / (S**3 + 9*S*S + 5*S + 9);
FREQ = LOGDOM(.1,1000.,100);
BODE (G //DOMAIN=FREQ )
```

With right, the advocates of menu-driven and question-and-answer interaction claim that their methods are specially advantageous for users unfamiliar with the system. In our example, the user of INTOPS needed to know only the existence of the two commands ENTER and BODE, whereas the user of IMPACT must know how to form a transfer-function, and how to put a variable as parameter of a function. However, due to the very natural notation of the IMPACT command language (which will be described in more detail later), any inexperienced user will be able to use IMPACT after only a few hours.

On the other hand, anyone familiar with both systems will save a factor 10 in time as well as in number of input lines when he uses IMPACT to construct the BODE diagram. For an easy-to-learn system like IMPACT, this means that the command language pays off heavily already after a few hours of use.

In order not to discourage the beginner during these first few hours, an on-line help will provide answers to all questions in a structured manner, also making self-tutorial possible. The help-facility can of course also be accessed by the advanced user when he needs information, e.g. about a lesser used algorithm and the corresponding function call parameters.

3. DATA STRUCTURES IN IMPACT

Whereas MATLAB supports only the double-precision complex matrix, IMPACT provides the user with several other data structures. In this chapter, most of these will be presented, together with some of the operations which can be performed on these different structures.

3.1 Matrices

As in MATLAB, all matrices in IMPACT are stored away using complex elements of high precision. Any scalar can be stored away as a one by one matrix. Matrix input is done using a very natural notation:

```
A = <1,2,3
     4,5,6
     7,8,9>;
```

constructs a 3*3 matrix A. If the column vector B has been entered as

```
B = <1.5 ; 4.3 ; 1>;
```

the equation A*x = B can be solved e.g. through

```
X = INV(A)*B
```

Many matrix operations, as different eigenvector operations, inversions and transformations, have been included in IMPACT.

3.2 Polynomial matrices

A polynomial matrix is a matrix where each element is a polynomial with (complex) coefficients.

Polynomial matrices are entered into IMPACT using a notation similar to that used for normal matrices. For example, the input line

```
Q = < 2^3^1 ; 4^4^1 >
```

will result in the polynomial column vector

```
Q(p)     =
   2. + 3.*p + 1.*p**2
   4. + 4.*p + 1.*p**2
```

which is the IMPACT form to describe the polynomial matrix

$$\begin{matrix} < & 2. + 3.*p + 1.*p^2 \\ & 4. + 4.*p + 1.*p^2 \quad > \end{matrix}$$

An alternative way of entering the polynomial matrix Q might be to first define the variable P as

```
P = <^1>;
```

Thereafter the polynomial matrix Q can be entered as

```
Q  =  <2 + 3*P + P**2; 4 + 4*P + P**2>
```

The basic matrix operations addition, subtraction and multiplication may be used on polynomial matrices (using the symbols +, - and *) if the basic dimensional rules are fulfilled. For example, the input lines

```
P     =  <^1>;
Z     =  <1+1*P , 2*P>;
WROW  =  < 1    , 2+2*P>;
WCOL  =  WROW';
XADD  =  Z + WROW , XMULT = Z * WCOL
```

will result in the output

```
XADD(p)    =
    2. + 1.*p    2. + 4.*p
XMULT(p)   =
    1. + 5.*p + 4.*p**2
```

Until now, all polynomial matrices have been entered in a non-factorized manner, specified through all non-zero coefficients of the polynomial elements. To further enhance the flexibility of IMPACT, polynomials can also be given in factorized form. Example:

```
QF  = FACTOR (Q)
```

will transform the matrix Q to a factorized form, resulting in

```
QF(p)   =
   (p + 1.)*(p + 2.)
   (p + 2.)*(p + 2.)
```

It is of course possible to enter factorized polynomial matrices directly:

```
QF = <-1¦-2
      -2¦-2>
```

Due to an ill-conditioned re-factorization, operations on factored polynomial matrices, where at least part of the factors need to be de-factorized before the operation (like addition), can be extremely badly conditioned. In IMPACT, the user is responsible for the testing on the accuracy of the result. However, IMPACT provides the user with a few tools: you can simulate a smaller computer word-length in that you specify the accuracy to be used in each arithmetic operation. This enables you to test the error-propagation of the used algorithm. Furthermore, IMPACT will warn you each time an ill-conditioned operation is performed. Moreover, as neither of the two "classical" polynomial representations is optimal with respect to numerical behaviour, IMPACT offers yet another representation which shall be discussed in Section 3.5 of this paper.

Furthermore, IMPACT supports several complex polynomial operations. For instance have algorithms calculating the least common left/right denominators, the Smith-form and eigenvalues /KAIL80/ been included.

3.3 Transfer-function matrices

Only in special cases is the inverse of a polynomial matrix another polynomial matrix. However, the inverse of a polynomial matrix can (as long as the matrix is non-singular) always be defined as matrix with rational function elements, a so called transfer-function matrix.

Transfer-function matrices are entered in a manner similar to that used by polynomial matrix entry. The input sequences

```
S = <^1>;
G = < 1/S      , 1/(S+1)
      1/(S+1) , 1/(S*(S+1)) >;
G = FACTOR(G)
```

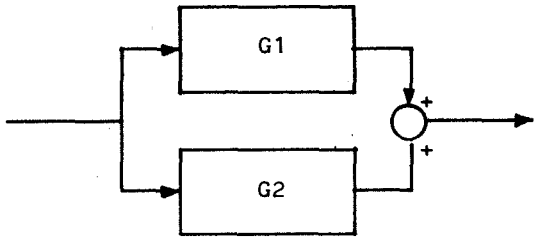
and

```
G = ONES(2) ./ < ¦0, ¦-1; ¦-1, 0¦-1 >
```

(where ONES(2) returns a 2*2 matrix filled with ones and ./ denotes an element-by-element division) both result in the factored 2*2 transfer-function matrix

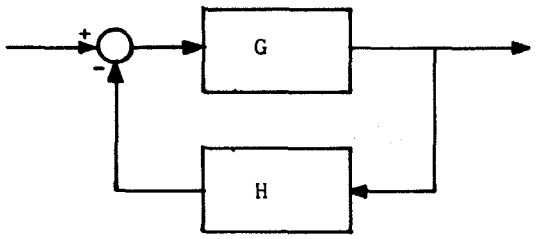
```
G(p)   =
          1.              1.
       -------         ----------
          p             (p + 1.)

          1.              1.
       --------        ----------
       (p + 1.)        p*(p + 1.)
```

In control theory, transfer-function matrices are used to describe systems in the frequency domain. Interesting enough, many mathematical operations on transfer-functions have a physical meaning. For example, the addition of two systems corresponds to the parallel connection:




```
GTOT = G1 + G2
```

A cascading of two systems is mathematically described through the multiplication in reverse order of the two system components. A feedback can either be described directly:




```
GTOT = G / (1 + G*H)
```

or through the use of the special feedback operator \\ (which does not correspond to any trivial mathematical operation):

```
GTOT = G \\ (-H)
```

3.4 System descriptions

In the time domain, a linear system is normally described by four different matrices:

$$\dot{x} = A*x + B*u$$
$$y = C*x + D*u$$

As this is a very common representation, IMPACT provides the user with a special data-structure, the linear system description. Given three matrices A, B, C of right dimensions, the function LCSYS will form a continuous linear system description out of these matrices,

```
CSYS1 = LCSYS(A,B,C)
```

whereas LDSYS will form a discrete linear system description with a sampling rate of DT:

```
DSYS1 = LDSYS(F,G,H,DT)
```

The D matrix was here assumed to be a null matrix of correct dimensions. However, if the user wants to define a D-matrix, this can be entered through the use of default redefinition:

```
CSYS2 = LCSYS(A,B,C //D=DD)
```

will include the matrix DD as the direct-path matrix.

Mathematical operations on system descriptions have been defined such that the physical meaning is the same as if the same operation

were performed on transfer-function matrices. For example, if a system of 2nd order has been defined through the matrices

```
A = <1, 1
     0, 1>;
B = <0
     1>;
C = <1, 0>;
SIMPLE = LCSYS(A,B,C);
```

the operation

```
CASC = SIMPLE * SIMPLE
```

will result in a system of order 4 with the component matrices

```
CASC.A = <1, 1, 0, 0
          0, 1, 0, 0
          0, 0, 1, 1
          1, 0, 0, 1>
CASC.B = <0
          1
          0
          0>
CASC.C = <0, 0, 1, 0>
```

Note that the dimension of the system matrix is doubled, just as the order of the physical system.

A linear system given in the time domain by three (four) matrices can always be transformed into a frequency representation, and vice versa. The transformation from the time to the frequency domain is given through the formula

```
S = <^1>;
G = C * INV(S*EYE(A) - A) * B
```

This valid IMPACT statement is also available as a separate function

```
G = TRANS(LCSYS(A,B,C))
```

The such determined transfer function matrix is not unique, as each transfer-function component could have reducible factors. The function REDUCE will shorten any common factors of a transfer-function (using the machine tolerance, or any other given tolerance, to determine if two factors are equal or not).

As the transformation from the frequency to the time-domain is not unique, IMPACT provides the user with a range of transformations resulting in linear system descriptions in different canonical forms, Jordan form etc.

3.5 Domain and trajectory variables

A domain is a sequence of discrete, increasing values on the real axis which can be used to form the independent variable of a table.

```
TIME = LINDOM(0.,50.,0.1)
```

would thus define a sequence TIME with 501 elements, the first of which has the value 0 and the last the value 50, using an increment of 0.1. With the help of the '&'-operator, domains can be concatenated. For example would

```
PULSE_BASE = LINDOM(0.,1.,0.01) & LINDOM(1.1,10.,0.1) & 20.
```

be a non-equidistant domain with 202 points.

A trajectory is a table of function values which uses a domain as independent variable. Such a table results from a variety of operations performed on domains. E.g. would the operation

```
TRA = SIN(TIME)
```

result in a table where each entry contains an independent variable copied from the domain TIME and the sine-value thereof.

Mathematical operations are defined on trajectories using the same domain, e.g. would the operation

```
TRB = TRA + COS(TIME)
```

once again be a table with one row of values as function of the independent variable TIME, whereas

```
TRC = <TRA, COS(TIME), TRA + COS(TIME)>;
```

would be a table where each entry is a row-vector with three elements. Note that TRB = TRC(3).

All graphical functions return a trajectory as result, this trajectory can then be plotted with the command PLOT.

```
PL1 = BODE(G1)
```

will compute a bode-diagram and store this diagram as a trajectory with one complex element (containing amplitude as real and phase as imaginary part) per entry. If you want to compare the BODE-diagrams of two transfer-functions, combine the trajectories into one trajectory and

plot this (the option //BODE will insure that the axes are logarithmically scaled and correctly labelled):

```
PL12 = <BODE(G1),BODE(G2)>;
PLOT(PL12//BODE)
```

On each plot, you will now find two different-colored/shaped curves from your two systems.

Furthermore, domains and trajectories can be used to simulate system behaviour /CELL83/. If SSYS is any system representation (e.g. a transfer-function matrix or a system description),


```
TABOUT = SSYS * TRA
```

will perform a simulation and store away the values of the output signal at the discrete times of the trajectory TRA, thus making TABOUT another trajectory variable specified over the same domain as TRA.

Finally, domains are useful for yet another purpose. As stated previously, polynomial operations may be numerically ill-conditioned. It has been found that, in many cases, a better behaviour results when polynomials (or rational functions) are represented by a set of supporting values rather than by coefficients or roots.

```
FREQ = LOGDOM(.1,1000.,100)
```

generates a domain consisting of 100 values distributed logarithmically over the interval from 0.1 to 1000.

```
PP1 = TRAJEC(P1,FREQ)
```

computes a trajectory (matrix) of the polynomial (matrix) P1 by evaluating each polynomial (or each transfer-function, resp.) at each of the supporting values of FREQ. Obviously, this gives rise to a third representation of polynomials which often exhibits better numerical properties. (All primitive polynomial operations such as addition, multiplication, and inversion become trivial.)

```
Q1 = ROOTS(PP1)
```

reestablishes the factored representation of PP1. This can be obtained by a numerically well-behaving Fast Fourier Transform (/GEIG73/).

4. COMMAND LANGUAGE

For IMPACT, a very versatile command language has been developed using the command language of MATLAB as a base. However, IMPACT allows for a more structured input, in form of more general statements and the availability of four kinds of macros.

The requirement that novices as well as very experienced users should be able to use IMPACT is reflected in the design of the IMPACT command language. On one hand, the basic commands are extremely easy to learn, but still powerful enough to make all kinds of operation possible, although not necessarily in the most optimal way. On the other hand, more complicated language elements can be used to perform complex operations. In particular, a hierarchical structuring of a problem is possible through the use of macros.

Due to their very natural notation, the basic commands can be mastered in a few hours time. The most essential statement is the assignment statement, which uses a notation similar to that of a normal mathematical formula. If we, for example, want to determine whether or not a certain linear system is controllable, we first enter the state and input matrices:

```
A = <1,0,1
     0,1,2
     0,0,1>
B = <0; 0; 1>
```

Thereafter we will get our answer by calling a procedure CONTR with the command

```
CONTR(A,B)
```

The procedure CONTR writes the wanted result on the terminal.

For more complex problems, a full-fledged, structured input language is available, including IF..THEN..ELSE, FOR/WHILE-loops, and so on. Example: The heat-diffusion in a long metal bar can be approximately modelled through a set of N differential equations (t(1). denotes the derivative of the variable t(1)):

```
t(1). = -2*t(1) + t(2)
t(N). = -2*t(N) + t(N-1)
t(i). = -3*t(i) + t(i-1) + t(i+1)    ,  1 < i < N
```

The state-matrix of this model can be obtained through the statements

```
FOR i = 1:n DO
  FOR j = 1:n DO
    IF (j = i) THEN IF (i = 1) OR (i = n) THEN a(i,j) = -2
                                          ELSE a(i,j) = -3
                    ENDIF
                ELSIF abs(j-i)=1 THEN a(i,j) = 1
                                 ELSE a(i,j) = 0
    ENDIF
  ENDFOR
ENDFOR
```

When a sequence of statements like these are to be performed several times, the user should use a macro to avoid typing errors and to save time.

4.1 Macros

IMPACT provides the user with four different types of macros.

4.1.1 Function macros

If the previously described model of a metal bar is to be used several times, each time with a different value for N, the user can save time by defining a function macro returning the wanted state matrix:

```
FUNCTION bar_matrix(n)
  FOR i = 1:n DO
    ::
  ENDFOR
  RETURN a
ENDFUNCTION
```

4.1.2 Procedural macros

Example: We want to write a procedure to add a new MACRO to our private MACRO library (PRILIB.INT) or replace an old one by a newer update. This can be performed by:

```
PROCEDURE ADDMAC (FILNAM)
  LOAD('PRILIB');
  READ(FILNAM);
  SAVE('PRILIB',MACRO);
ENDPROCEDURE
```

This procedure is executed by

```
ADDMAC('NEWMAC.IMP')
```

Upon call, only one variable is known within the procedure, namely the variable FILNAM which is of type text-string and contains the name of the file which the new MACRO is currently stored in. LOAD('PRILIB') loads all variables from file PRILIB.INT containing the MACRO library. READ(FILNAM) reads the new MACRO and converts it to its internal representation. If such a MACRO variable was already in the library, this variable is now overwritten by the new definition. SAVE('PRILIB',MACRO) saves all currently accessible MACRO's (but not the text-string variable FILNAM) in a new cycle of the file PRILIB.INT. Upon return from the procedure, the old context is reestablished, and all previously visible variables are accessible again.

4.1.3 String macros

Until now, we have considered macros which look and work as functions or procedures. In this chapter, we will extend our macro definitions to include a general macro concept.

When we allow ourself a slight simplification, a macro definition connects a string of characters (possibly divided onto several lines) with a macro name. Each time this macro is called, the corresponding string is inserted at the point of call with each formal parameter being replaced by its actual value. In the more general case, the string of a macro could be inserted not only as a factor within any type of expressions (function macros returning one variable) or as a statement (procedure macro changing the values of one or several variables), but anywhere in an IMPACT input.

As a trivial example, let us consider a user who, for estetical reasons, dislikes the element-by-element operations '.*' . Such a user could avoid this symbol through defining a new string macros :

```
MACRO ELMULT
  .*
ENDMACRO
```

which thereafter can be used in statements as

```
C =  A ELMULT B          -- Equivalent to C=A.*B
```

Generally, the string macro is a very versatile instrument. It can for example be used to dynamically define new functions, and will certainly

be used to shorten other macros through the use of "tricky" string operations. However, inexperienced users are warned not to use the string macro.

4.1.4 System macros

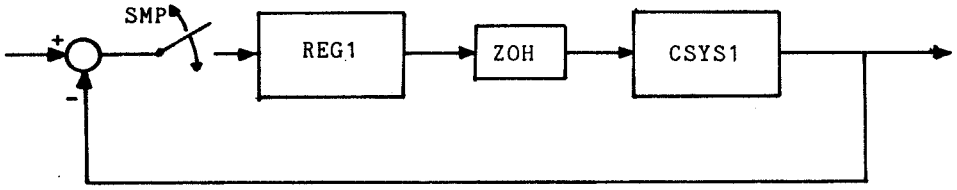
As we live in an imperfect world, control scientists usually have to use non-linear models to describe a real system. IMPACT provides the so-called system macro for this modelling. Consider the following example describing a discrete PI-regulator :

```
SYSTEM discr_regulator(kp,ki,dt)
  DSTATE int
  INITIAL int0=0;
  INPUT err
  OUTPUT u
    NEXT.int = int + ki*err*dt;
    u = kp*err + int;
ENDSYSTEM
```

This definition of a discrete system with one difference equation can be used to create a variable of the same type discr_regulator:

```
REG1 = discr_regulator(1,1,0.1)
```

The thus created system variable can then be used in any mathematical operations to construct parallel and/or concatenated systems, perform simulations, and so on. Given the predefined systems SMP (sampler) and ZOH (Zero-order-hold), a sampled data system using REG1 as regulator can be created through the single statement:




```
STOT = (CSYS1*ZOH*REG1*SMP(.1))\\(-1)
```

5. IMPLEMENTATION CONSIDERATIONS

Although MATLAB, as most other larger scientific software-projects of the sixties and seventies, is coded in FORTRAN, and although most available algorithm-libraries are FORTRAN-coded, it has been decided that IMPACT shall be coded in ADA. There are manifold reason for this /BIRD83/:

- ADA allows almost any types of data-structures to be directly defined, avoiding the hazzle of redefining all structures into arrays (the only structure available in FORTRAN). Furthermore, through the use of discriminants, ADA allows for the dynamic sizing of arrays, which means that no unnecessary space has to be reserved, as would be the case in a language like PASCAL.

- ADA, due to recursiveness, allows for a much more elegant coding of the IMPACT expression parser than FORTRAN would do. In this way, IMPACT shall be easier maintainable and updateable than MATLAB.

- ADA provides for a unique means of exception handling which shall prove very useful for our task.

- ADA is per definition portable, there may not exist any sub- and/or super-set of ADA with that name.

- ADA is highly structured, making a modular programming possible, resulting in reliable and easily maintainable code. Furthermore through the use of visibility rules, all system-dependencies can be hidden from the user as well as from most of the people involved in the development of IMPACT.

- ADA-libraries of algorithms are expected to emerge on the market in the near future. Therefore, IMPACT is to contain a well-defined interface for later incorporation of new algorithms.

At the present state of development, an IMPACT users' manual /RIMV83/ exists which describes the command language as well as each available function (algorithm).

To simplify the construction of the expression parser needed in IMPACT, the syntax of the IMPACT command language has been defined using an extended Backus-Naur form. This syntax has then been tested for consistency using a general purpose parser /BONG79/. LL(1) parsibility

rules have been applied wherever applicable.

The actual coding of IMPACT has commenced late 1983 using one of the first, almost complete ADA compilers. It is expected that a first subset of IMPACT will be available during 1984.

6. REFERENCES

/AGAT79/ Agathoklis, P., et alia; "Educational Aspects of Using Computer-Aided Design in Automatic Control"; in Proc. of the IFAC Symposium on Computer-Aided Design of Control Systems, Zurich, Switzerland; Pergamon Press, London; 1979.

/ASTR83/ Åstrom, K.J.; Computer-Aided Modeling, Analysis and Design of Control Systems, A Perspective; Report CODEN: LUTFD2/(TFRT-7251), Department of Automatic Control, Lund Institute of Technology, Sweden; 1983.

/BIRD83/ Birdwell, J.D.; "Future Directions in Computer-Aided Control System Design, Software Development"; IEEE Control Systems Magazine, February 1983.

/BONG79/ Bongulielmi, A.P. and F.E. Cellier; "On the Usefulness of Using Deterministic Grammars for Simulation Languages"; Proc. of the SWISSL Workshop, St. Agata, Italy; to appear in Simuletter; 1979.

/CELL83/ Cellier, F.E. and M. Rimvall; "Computer Aided Control Systems Design"; Proc. First European Simulation Conference ESC'83, (W. Ameling, Ed.); Informatik Fachberichte, Springer Verlag; 1983.

/CUEN79/ Cuenod, M.A.; (Editor); Proc. First IFAC Symposium on CAD of Control Systems; Pergamon Press; 1979.

/DUNG79/ Dungorra, J.J, Bunch, J.R., Moler, C.B., Stewart, G.W.; LINPACK Users' Guide; Society for Industrial and Applied Mathematics; 1979.

/ELMQ82/ Elmqvist, H.; "A Graphical Approach to Documentation and Implementation of Control Systems"; Proc. 3rd IFAC/IFIP Symposium on Software for Computer Control, SOCOCO'82, Madrid, Spain; 1982.

/GEIG81/ Geiger, P.; Nullstellenbestimmung bei Polynomen und allgemeinen analythischen Funktionen als Anwendung der schnellen Fouriertransformation. Diss.Math.ETH 6759; 1981.

/GARB77/ Garbow, B.S., et alia; Matrix Eigensystem Routines, EISPACK Guide Extensions; Springer, Lecture Notes in Computer Science, 51; 1977.

/HERG82/ Herget, C.J. and A.J. Laub; Special Issue on Computer-Aided Control System Design Programs; IEEE Control Systems Magazine, December 1982.

/IMSL82/ IMSL Library Reference Manual, Edition 9; IMSL, 1982.

/INFO79/ Infotech state of the art report : Man/Computer Communication, Vol 1-2, 1979.

/KAIL80/ Kailath, T.; Linear Systems; Prentice-Hall; 1980.

/MOLE80/ Moler, C.; MATLAB, Users' Guide; Department of Computer Science, University of New Mexico, Albuquerque, USA; 1980.

/RIMV83/ Rimvall, M.; IMPACT, Interactive Mathematical Program for Automatic Control Theory, A Preliminary User's Manual; Institute for Automatic Control, ETH Zurich, Switzerland; 1983.

/SMIT74/ Smith, B.T. et alia; Matrix Eigensystem Routines, EISPACK Guide; Springer, Lecture Notes in Computer Science, 6; 1974.

/SSP68/ System/360 Scientific Subroutine Package; Version III Programmers Manual, IBM, 1968.

/WALK82/ Walker, R., et alia; "MATRIX$_x$, A Data Analysis, System Identification, Control Design, and Simulation Package"; IEEE Control Systems Magazine, December 1982.