Informatik-Fachberichte

Herausgegeben von W. Brauer
im Auftrag der Gesellschaft für Informatik (GI)

71

First European Simulation Congress ESC 83

Aachen, September, 12–16, 1983
Proceedings

Edited by W. Ameling

Springer-Verlag
Berlin Heidelberg New York Tokyo 1983

Contents:

COMPUTER AIDED CONTROL SYSTEMS DESIGN

Dr. François Cellier, CH-8092 Zürich

Mr. Magnus Rimvall, CH-8092 Zürich

Summary: This paper reflects our experience with Computer Aided Control System Design (CACSD). A first section briefly presents a CACSD-system for student use (INTOPS) which has been developed by our group some years ago. Our experience with this software system is outlined. A second section discusses a few state-of-the-art CACSD-systems developed by other research groups. In particular, the systems developed by K.J. Åström and his group are mentioned (SIMNON, IDPAC, MODPAC, POLPAC, and SYNPAC), as they seem to be among the most advanced systems currently available. A next chapter discusses the advents of a more general expression parser at hand of the MATLAB matrix manipulation program. It is shown that, though MATLAB was not really designed for the implementation of control algorithms, many control problems can be formulated and solved very elegantly by means of MATLAB. A new CACSD-system (IMPACT), which is currently under development by our group, is then presented. This system shall enhance the capabilities of MATLAB by encompassing additional operations and data structures particularly useful in control system analysis and design. A final chapter concludes this discussion by mentioning some perspectives for further development.

1. Introduction.

Control Theory has become a well established topic with a history of roughly 50 years and an extensive number of publications. The topic has meanwhile gained so much that recently a decision was taken to issue the first Encyclopedia of Control /SING84/.

However, there exists a large gap between the fairly sophisticated control algorithms being developed and the simple PID-controllers which are predominantly used in practice. One of the reasons for this gap may lie in the fact that modern controllers are quite difficult to design. Their evaluation requires in most cases large skill and a substantial number of sophisticated numerical algorithms. The use of a computer for this purpose is a conditio sine qua non.

In the early seventies, several research groups developed extensive libraries of FORTRAN subroutines implementing many numerical control algorithms. One of the largest collection of such subroutines is AUTLIB /CELL77/. AUTLIB contains more than 300 FORTRAN-coded subroutines for control system analysis and synthesis, and still there are many problems not tackled. In particular, we have only few programs for multivariable systems, and even less for nonlinear systems. Another good library has recently been developed at Kingston Polytechnic /DENH82/. Most of these libraries (AUTLIB not excluded!) suffer from the fact that the therein contained programs are coded predominantly by control engineers rather than by numerical mathematicians. Their numerical behaviour is not in all cases sound. It is only recently that control theory became sufficiently well reputed among numerical mathematicians (!), that also they started to look into these problems, hopefully with the long-time effect that gradually better numerical algorithms and computer programs will be made available, which shall be equally sound and elaborate as currently LINPACK (for linear matrix operations), EISPACK (for eigenvalue and eigenvector analysis) and IMSL (matrix and statistical operations). A first conference on Numerical problems in control was organized in 1980 jointly by K.J. Åström (Control Engineering) and G. Golub (Numerical Mathematics), two highly reputed researchers who were able to attract a rather unique collection of the best researchers from both fields to discuss these problems.

However, even the use of these subroutine libraries is a fairly complicated and time consuming undertaking. Let us assume that somebody who has never before written a program to compute the eigenvalues and eigenvectors of a matrix wants to do this by the use of such library routines. He shall start by asking the numerical group at his institution whether they know of any available and adequate software for his task. Being advised to use EISPACK, he shall have to consult the EISPACK documentation (/SMIT74/, /GARB77/), which in itself is fairly voluminous, to find out which subroutines to use. Next, he shall have to organize matrix input and also matrix output (for checking on input errors), then call one after the other somewhat like 5 different EISPACK subroutines (each with a fairly large number of parameters), and finally print out the results. The resulting FORTRAN program comprises of about 50 statements (including comments). Its coding shall, in the best of all cases, keep him busy for a couple of hours. In the design of an advanced control algorithm, this may be just a very little subproblem to be solved.

As this is very inconvenient, it is important that the control engineer is supported by the computer to a much larger extent. This can be achieved by developing interactive CACSD programs with interactive machine-readable documentation (the HELP-ducument) by use of which the control engineer can concentrate on the control aspects of his problems rather than on programming aspect. Let us now discuss briefly the requirements of such CACSD systems: A CACSD program should be easily learnable and easily useable; it should be optimally adapted to control problems, and at the same time be as flexible as possible. Obviously, these demands are in competition with each other. The most easily learnable CACSD program, and thus a program being optimally suited for student use, is one which uses a rigid question-and-answer format, in that "the computer" (that is: the program) poses the pertinent questions, and the user is asked to provide the answers. Whenever a user does not fully understand a question, he may type in a question mark to obtain a more comprehensive explanation. Obviously, the same program is not very flexible, as the program stays in control of the operation thoughout the session, that is: it is not possible to depart from the foreseen paths. A command driven program is certainly more flexible, but less easy to use. A program which offers few commands only is certainly easier to learn than one which provides the user with a large variety of commands. However, the same program may be much more difficult to use, as more complex operations might have to be composed from those few primitives of the former program by the user, whereas the latter program probably provides for the required operations directly. That is: CACSD means making compromises. The quality of a CACSD program is determined much more by its man-machine communication interface than by the operations it offers. In recent years, a noticeable progress has been achieved in this respect, as shall be shown in this paper.

2. INTOPS : A Question-and-Answer Driven CACSD Program Suite.

INTOPS /AGAT79/ has been developed by our group a couple of years ago. In fact, INTOPS was conceived as an interactive front-end to our control library AUTLIB /CELL77/. The INTOPS package has been developed largely by students in their term- and diploma projects , and this unfortunately is reflected in the code. The modules do not always match very well together. Therefore, the maintenance of the program creates some headache. INTOPS consists of three programs:

a) POLOPS: for polynomial operations (comprising 35 commands),

b) MATOPS: for matrix operations (comprising 35 commands),

c) LTDOPS: for linear time-domain operations (comprising 17 commands).

INTOPS has been conceived from its beginning as a program to be used primarily by students. Therefore, a strict question-and-answer dialog is used.

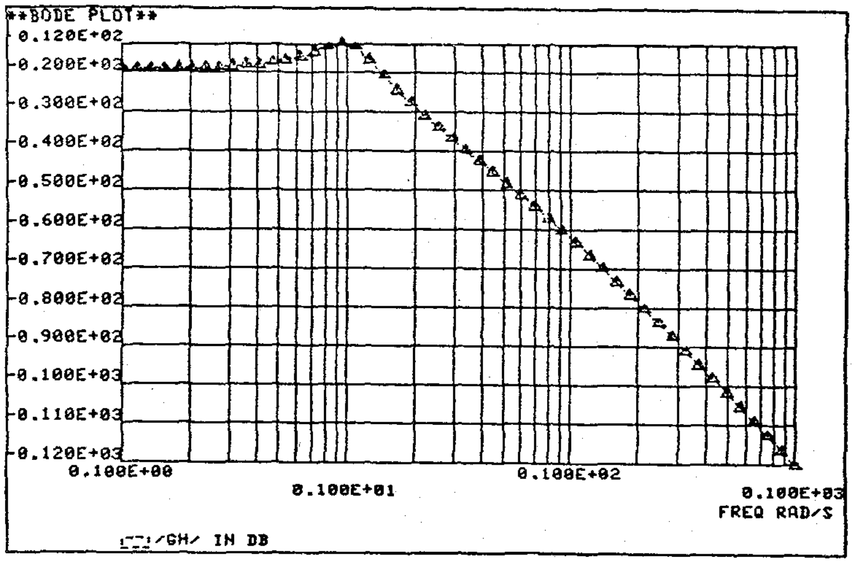
Let us illustrate the INTOPS dialog by means of a simple example. We want to compute the Bode diagram of the transfer function:

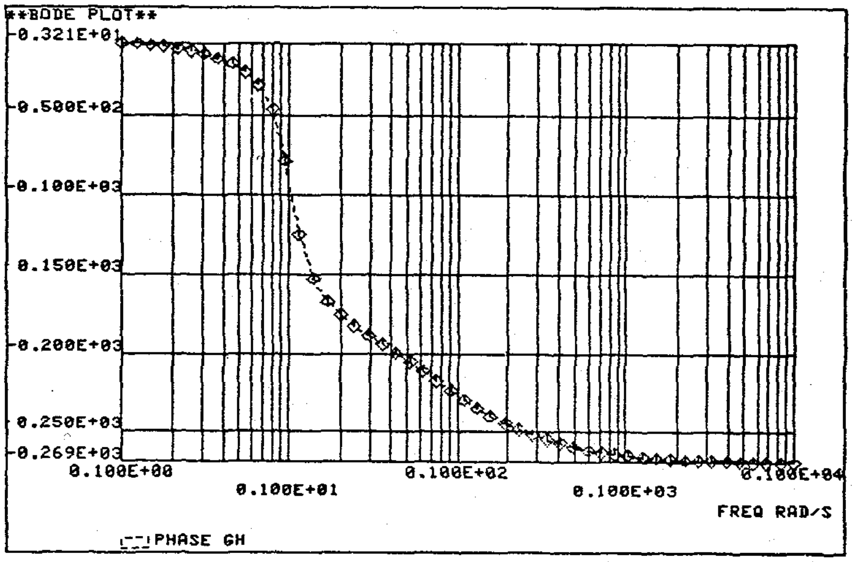
$$G(s) = \frac{1}{s^3 + 5s^2 + 9s + 5}$$

This can be performed by the following dialog (user input is underlined):

```
RUN_INTOPS

INTOPS>

I>      FOR SELECTING PROGRAM OPTIONS:

        OPTION = POLOPS (POLYNOMIAL OPERATIONS), OR
        OPTION = MATOPS (MATRIX OPERATIONS), OR
        OPTION = LTDOPS (LINEAR TIME DOMAIN OPERATIONS),

I>      OPTION = POLOPS

POLOPS>
P>      HELP IS AVAILABLE

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 = 1000
P>      OMEMAX = 1000
P>      OMEMIN = .1
```

finally resulting in:





For this particular problem, INTOPS proved fairly comfortable, as the requested path had been foreseen by us when implementing INTOPS. On the other hand, if one tries to find out (by use of INTOPS) whether or not a linear system is controllable (a question which at least still can be answered!), INTOPS proves extremely clumsy, because the requested path was not preprogrammed.

INTOPS certainly fulfils the task it was designed for. Our students are extremely happy with this software. There is no need for an INTOPS manual, the only thing the student needs to know is how to sign on to the machine and how to start the program. Afterwards, all operations are self-explanatory, and the little help still required is available by means of "on-line" help support.

Where then lie the deficiencies of INTOPS:

A) Shortcomings of INTOPS from a users' point of view:

A1) INTOPS is not user extendable. There exists no possibility at the dialog- (that is: INTOPS-) interface to combine a sequence of operations into a new command. Addition of new algorithms at the implementation- (FORTRAN-) interface proves unnecessarily complicated.

A2) Once the user has entered an incorrect code, he has no possibilities to correct it, but is forced to strictly follow the programmed path, and wait for the end of this now senseless dialog until he can restart again.

A3) After a while, the question-and-answer scheme gets rather boring, and comparatively time-consuming.

A4) Although it is possible to stash INTOPS variables away on a file for reuse in another session, this data interface has not been properly designed. It is not always possible to retrieve data produced by one program module from within another program module, and there is hardly any chance that one might read INTOPS-produced data into another program.

B) Shortcomings of INTOPS from a software engineer's point of view:

B1) The heart of each larger software system should be a set of carefully designed data structures which are used consistently throughout all program modules. There can hardly happen anything worse to a software engineer than being forced to modify the central data structures of an existing large software system. In INTOPS, such a central data concept does not even exist. Each student who contributed to INTOPS has designed his own data structures with the effect that INTOPS is difficult to maintain and update. This deficiency shines through to the user in shortcoming A4.

B2) No CACSD program can be made entirely machine independent. There does, for instant, not yet exist a standard graphics interface. First steps towards the development of such an interface (SIGGRAPH, GKS /GKS83/) are not yet accepted enough to make them really useable. (The only widely used interfaces are the CalComp subroutines used to drive most plotters, and the PLOT10 software used to drive most graphical terminals. Both "standards" are, however, entirely insufficient in any respect.) For this reason, most graphical software systems use their own kernel of virtual graphics primitives, leaving it up to the implementer to map these primitives into those which are available at his installation (a fairly simple task though). INTOPS has no centralized graphics interface. Instead, each module basically uses its own plot routine, partly by calling PLOT10 routines, and partly by utilizing the HP-2648 graphics capabilities. This makes INTOPS badly portable.

B3) It makes sense to provide an interface to a relational data base in which all permanent data are stored for reuse (preferrably, also the graphics system should make use of this interface). Such a design strategy ensures that data are easily exchangeable between all program modules as well as with the outer world. Unfortunately, INTOPS does not make use of such a concept. Instead, it uses a private (badly designed) direct access data file. This shortcoming comes through to the user in A4. For the same reason, a central data administration is hardly imaginable in INTOPS, and there exists no well-defined interface between the different program modules.

Altogether is to be said that, although INTOPS is certainly a useful program, it no longer represents the state-of-the-art in CACSD-programs.

3. State-of-the-art CACSD programs

Beside INTOPS, there exist quite a few other CACSD-programs on the "program market". D.K. Frederick has recently compiled an impressive list of such systems /FRED82/, which is still far from being complete. Most of these CACSD programs are comparable to INTOPS both in complexity and sophistication. They vary largely with respect to the types and numbers of operations (and thus algorithms) implemented, but little in their man-machine interface. Most of them suffer from similar shortcomings and a lack of systematic design methodology as INTOPS. Among the best currently available CACSD programs are those developed by K.J. Åström and his group. While in many research groups the CACSD programs were developed as side products - often by less qualified people which were considered not bright enough to conduct theoretical work (!), Åström early realized the potential impact of CACSD, and put highly qualified software engineers and professional programmers on the task. These efforts resulted in a suite of five CACSD programs:

1) SIMNON /ELMQ77/, a simulation program for continuous systems with discrete-time regulators (sampled data systems),

2) IDPACK /WISL80a/, a program system for data analysis and identification of linear deterministic or stochastic multi-input, single-output systems,

3) SYNPAC /WISL80b/, a state-space oriented control systems design program,

4) POLPAC, a frequency-domain oriented design program, and

5) MODPAC /WISL80c/, a program for transformations between different control system representations.

A short overview of these five programs can be found in Åström /ASTR83/. Many of the previously mentioned shortcomings of INTOPS are eliminated in these programs:

A1) All programs are command-driven. A MACRO feature (INTRAC /WISL78/) exists in all of these programs, providing for a unified means of generating additional operators at the CACSD interface level. INTRAC contains the expression parser which is, therefore, common to all five programs. This makes the programs more uniform and better maintainable. At the implementation interface level (FORTRAN), some of the packages (e.g. SIMNON) allow the user to add new algorithms in a standardized (although not necessarily convenient!) manner.

A2) This drawback is automatically removed by the command-driven dialog.

A3) The beginner cannot run any of these programs without having studied a manual beforehand. Unfortunately, those are the weakest parts of these software systems! However, the MACRO-concept allows to make the dialog as convenient as possible. (It is even feasible to implement a question-and-answer dialog by use of these macros.)

A4) Not optimally resolved.

B1) All programs use the same data structure concept.

B2) Not resolved.

B3) The data interface between the different program modules is better than in INTOPS, although still far from optimal.

It has to be stated that all these programs stem from a time when computer power was still expensive, while programmers were still cheap (!). Accordingly, the design goal weighed execution efficiency higher than software portability, maintainability and updateability. This decision must certainly today be considered wrong.

4. MATLAB - A Matrix Laboratory.

A somewhat different approach was taken by C. Moler in the development of MATLAB /MOLE80/, an interactive program for all sorts of matrix operations. MATLAB is a full-fledged interpretive language using complex, double precision matrices (including vectors and scalars as special cases) as its only data type. On these data, all kind of operations can be performed by use of a very natural notation.

Matrices are specified by use of angular brackets:

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

where "," or SPACEs separate column elements, while ";" or the continuation on a new row part each of the rows. Each matrix element may be a submatrix in itself, and may be compiled as an expression of almost unlimited complexity.

Let us assume that we want to know whether or not the system

$$\dot{x} = Ax + Bu$$

with

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -3 & -4 & -5 & -6 \end{bmatrix} \qquad b = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

is controllable. This can e.g. be computed in MATLAB by the statements :

```
A = <<0;0;0>,EYE(3);<-3,-4,-5,-6>>
B = <0;0;0;1>
QS = <B,A*B,A*A*B,A**3*B>
RANK(QS)
```

where EYE(3) denotes a 3 by 3 unity matrix. If the same operation is to be executed many times for systems with different matrices A and B as well as different system orders N, we might want to store onto the file CONTR.MTL the following statements:

```
QS = B; AUX=B;
FOR I=2:N, AUX=A*AUX; QS = <QS,AUX>; END
RANK(QS)
RETURN
```

which can then be executed by:

```
A=<...>; B=<...>; N=...;
EXEC('CONTR.MTL')
```

Why is MATLAB that much easier and more comfortable to use than all the other programs which have been specially designed for control operations? We assume that the reason has something to do with the enhanced capabilities of the highly recursive expression parser. Even the CACSD programs developed by Åström are far less advanced in this respect. This commonly found limitation has presumably to do with the fact that most CACSD programs are coded in FORTRAN which makes the use of recursions cumbersome. (Also MATLAB is coded in even portable FORTRAN'66, but its expression parser is rather sophisticated and certainly not a FORTRAN program which a second-class control engineer is likely to develop (!).)

Let us show at a somewhat more elaborate example that MATLAB is quite useful for involved control applications. Let us compute for the linear system

$$\dot{x} = Ax + Bu \ ; \ x \in R^n$$

a state feedback such that

$$\int_0^\infty (x'Qx + u'Ru)dt \overset{!}{=} \min.$$

According to the eigenvalue method proposed by P. van Doren /DORE80/, we can solve this Riccati problem by means of the following algorithm:

a) Check on controllability of the system. If the system is not controllable return with an error message.

b) Compute the Hamiltonean:

$$H = \begin{bmatrix} A & -BR^{-1}B' \\ -Q & -A' \end{bmatrix}$$

c) Compute the eigenvalues and eigenvectors of the Hamiltonean (2*N). As the system is controllable, the eigenvalues will be symmetrical to the imaginary axis, and have all their real parts different from zero.

d) Take those eigenvectors belonging to negative eigenvalues (dimension: 2N*N), and split this reduced eigenspace into equally sized upper and lower parts:

$$V = <V1;V2>$$

e) Now the Riccati feedback can be computed as

$$K = -R^{-1}B'P$$

where

$$P = Re(V_2 * V_1^{-1})$$

For this algorithm, we may write the following MATLAB "program" (file: RICC.MTL):

```
EXEC('CONTR.MTL')
IF ANS <> N, SHOW ('SYSTEM NOT CONTROLLABLE'), RETURN, END
<V,D>=EIG(<A,-B*(R\B');-Q,-A'>);
D=DIAG(D); K=0;
FOR J=1:2*N, IF D(J)<0, K=K+1; V(:,K)=V(:,J); END
P=REAL(V(N+1:2*N,1:K)/V(1:N,1:K));
K=-R\B'*P
D=<>; V=<>;
RETURN
```

which is a reasonably compound way of specifying a fairly complex algorithm.

Where then lie the deficiencies of MATLAB with respect to control applications? These are still manifold:

1) The EXEC-file concept is no valid replacement for a true MACRO facility. For example, there is no way to pass actual parameters during a macro call. INTRAC is much better in this respect.

2) Although there exists the possibility in MATLAB to store data away for later reuse (SAVE and LOAD commands), one soon ends up with a large number of unstructured data files. A clean data base access mechanism would be much better.

3) Control engineers want to see nice graphs. The output facilities offered by MATLAB are insufficient in this respect. (A data base interface would soften that problem as well, in that the stored curves could then at least be analyzed by a separate graphics package outside MATLAB which is linked to the same data base.)

4) Although there exist currently few numerically sound algorithms for polynomial operations, control engineers want to have the possibility to analyze and synthesize their systems in the frequency domain as well. This requires an extention to the data structures offered by MATLAB.

5) Even linear systems often require nonlinear controllers (e.g. when using wind-up techniques in otherwise classical PID-controllers, or when using adaptive controllers). Therefore, an extention to nonlinear system descriptions is required.

6) A library of control algorithms should be provided with the system. (This final request is the one which is easiest to supply!)

Also some other control engineers have meanwhile realized that MATLAB has a great potential as a basis for an excellent CACSD program. At least one extension of MATLAB (MATRIX$_x$ /WALK82/) is currently available on the software market. This product, which provides for linear system descriptions (in the time domain), for some frequency domain operations (such as Bode diagrams), and for graphical output, has just two major disadvantages:

a) It just solves a few of the previously mentioned demands, namely numbers 3 and 6. In particular, no new data structures have been provided. MATRIX$_x$ makes use of the fact that a linear system

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx + Du \end{aligned} \qquad x \in R^n \; ; \; u \in R^m \; ; \; y \in R^p$$

can unequivocally be represented by the matrix:

$$S = \begin{bmatrix} A & B \\ C & D \end{bmatrix}$$

a matrix with dimension (N+P)*(N+M), when the system-order N is known. Unfortunately, such a system description is not extendable to encompass nonlinear structures as well, and neither is a decomposition into subsystems or a composition from subsystems achievable.

b) MATRIX$_x$ is unaffordable to universities!

At least one of the other CACSD programs (CLADP /MACI82/) shares some of the advantages of MATLAB and MATRIX$_x$ by providing a similar (though less appealing) exression parser. CLADP is particularly useful for the analysis of multivariable systems in the frequency domain.

5. IMPACT, Interactive Mathematical Program for Automatic Control Theory

After we had realized that INTOPS was no longer up-to-date, and as we felt that - on the basis of MATLAB - an excellent CACSD program could be produced, we decided to start with the IMPACT project which shall in the sequel be briefly presented.

5.1 Polynomial Operations

The MATLAB double precision complex matrices are extended to double precision, complex tensors, where the third dimension takes either polynomial coefficients (defactorized form) or roots of polynomials (factorized form). In this way, polynomial matrices can be formulated, and even transfer function matrices (rational function matrices) can be represented as a set of two such tensors.

Let us discuss once more the Bode example. This can be solved in IMPACT by one single statement:

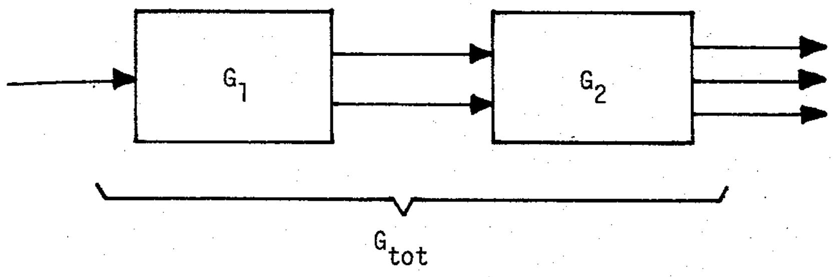
```
BODE(1/[5^9^5^1])
```

or somewhat more verbous (but better readable) by

```
S = [^1]
NUME = 1;
DENO = S**3 + 5*S*S + 9*S + 5;
G = NUME/DENO;
BODE(G)
```

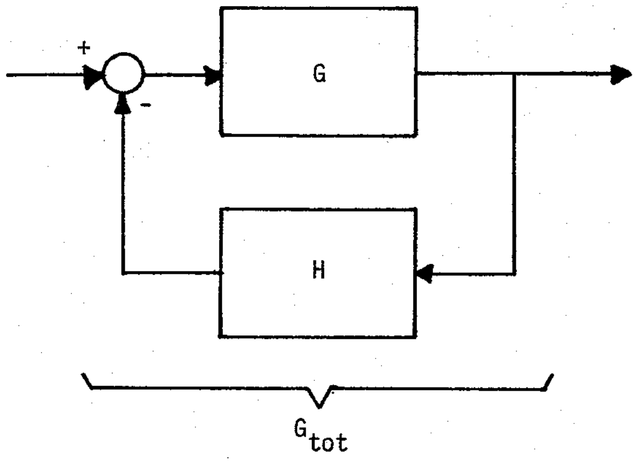
The "^"-operator separates polynomial coefficients.

Cascading of systems is performed by the normal multiplication applied in reverse order



```
GTOT = G2*G1
```

Parallel connection is performed by addition, and the "\\"-operator has been introduced to represent the feedback loop:



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

More complex operators are performed on these structures as easily as in MATLAB. E.g. can the transfer function matrix of a linear system:

$$\dot{x} = Ax + Bu$$
$$y = Cx$$

be computed as:

```
G = C*INV(S*EYE(N)-A)*B
```

5.2 Macros:

Four types of macros are provided in IMPACT:

5.2.1 Function macros

Example: We want to expand a transfer function matrix such that all denominators are equal. This can be performed by the function:

```
        FUNCTION EQLIZ(G)
            H = REDUCE(G);
            L = LCD(DENOM(H));
            D = REDUCE( (ONES(G) * L ) ./ DENOM(H) );
            EQLIZ = (D .* NUM(H) ./ L);
        ENDFUNCTION
```

where REDUCE cancels common factors of the numerator and the denominator; LCD computes the least common divisor of a polynomial matrix; DENOM extracts the denominator polynomial matrix of a rational function matrix; ONES(G) produces a matrix with the same dimensions as G whose elements are all scalars equal to 1; and NUM extracts the numerator polynomial matrix. For numerical reasons, it is recommended to apply this FUNCTION only to transfer function matrices in factorized form.

5.2.2 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.

There exist two special procedures (LOGIN.MTL) on the system account as well as on the user's account. These are executed automatically upon call to IMPACT. The users LOGIN-file could look like:

```
PROCEDURE LOGIN
  SHOW ('What the hell are YOU doing on my account number?')
  PI = 4*atan(1);
  WHO
ENDPROCEDURE
```

All variables or MACRO's created during the execution of the LOGIN-file are read-only.

5.2.3 String MACROS

String macros (paranthesized by the keywords "MACRO" and "ENDMACRO") are general purpose MACRO's for all types of applications. These MACRO's are more versatile than FUNCTIONS or PROCEDURE's, but they are also more dangerous to apply as they allow for less error checking. For that reason, the use of string MACRO's is not recommended to beginners.

5.2.4 System MACROS

EXAMPLE: Let us formulate a system description for the Van-der-Pol oscillator

$$\ddot{x} - \mu(1-x^2)\dot{x} + x = 0$$

This can be performed by:

```
SYSTEM VANDERPOL(MY)
   STATE X(2)
   INITIAL X0=<0;0>;
   OUTPUT Y
      X(1). = X(2);
      X(2). = MY*(1-X(1)*X(1))*X(2)-X(1);
      Y=X(1);
ENDSYSTEM
```

A new variable of type VANDERPOL may now be created by:

```
SYS1 = VANDERPOL(2.)
```

or

```
SYS2=VANDERPOL(.5 //X0=<.75;-.75>)
```

The "//"-operator is used to overwrite defaulted parameters which, in IMPACT, are named rather than positional parameters to avoid long parameter lists.

There exist many predefined SYSTEM MACRO's, e.g.

```
SYS3=LCSYS(A,B,C)
```

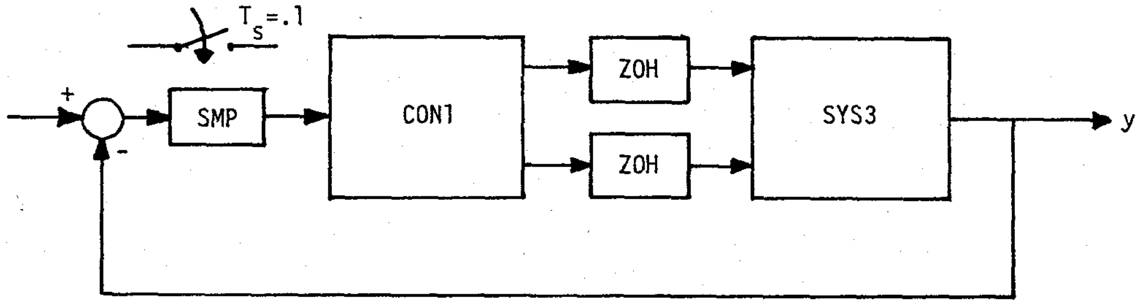
to denote linear continuous systems, or

```
    CON1=LDSYS(F,G,H,TS)
```

to denote linear discrete-time systems (with sampling interval TS).

5.3 Time-Domain Operations.

Example: Let us create a system description for the compound system:



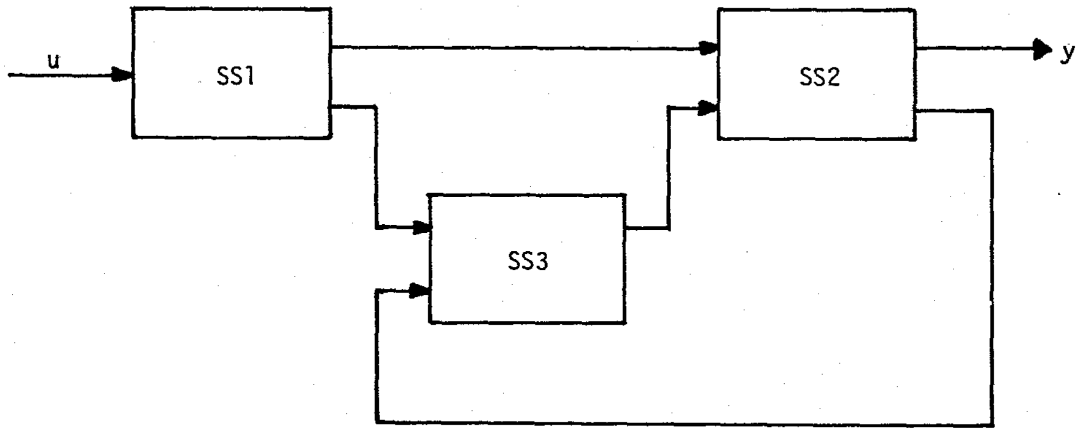
where ZOH denotes the predefined system type zero-order-hold, and SMP(TS) denotes the predefined system type sampling.

This task can be accomplished in IMPACT by

```
    STOT = (SYS3 * <ZOH,0;0,ZOH> * CON1 * SMP(.1)) \\ (-1)
```

involving a little compiler to generate a new system macro (of a hidden type) and, thereafter, create the variable STOT of this hidden system type.

The more complex structure:



could be handled by writing:

```
SYSTEM STOTTYPE(SS1,SS2,SS3)
   IMPORT SS1,SS2,SS3
   INPUT U
   OUTPUT Y
      SS1.U = U;
      SS2.US = SS1.Y1;
      SS2.U2 = SS3.Y;
      SS3.U1 = SS1.Y2;
      SS3.U2 = SS2.Y2;
      Y = SS2.Y1;
ENDSYSTEM
```

As one can see, the system description operators in IMPACT are defined such that operations in the time-domain resemble those in the frequency domain as much as possible. However, due to the possibility of describing also nonlinear and sampled data systems, they are even more versatile.

Let us assume, the three systems above were specified in the frequency domain. In that case, one possible solution to achieve the interconnection would be to write

```
S1 = DFORM(G1);
S2 = DFORM(G2);
S3 = DFORM(G3);
STOT = STOTTYPE(S1,S2,S3);
SLIN = LINEAR(STOT);
GTOT = TRANS(SLIN);
```

whereby DFORM transforms the systems into a time-domain description in controllability canonical form (companion matrix representation), LINEAR transforms STOT back into a system description of type LCSYS, and TRANS computes again a transfer function matrix.

5.4 Simulation

A new concept has been proposed to simplify the simulation of (linear as well as nonlinear) models. To show how this concept works, we have to introduce two more data types:

5.4.1 The Time-Domain

A time-domain is a collection of points on the independent axis which form a table. E.g. would

```
TIME = DOMDEF(0,100,0.5);
```

create a table with 201 elements starting with 0, and using increments of 0.5. Time-domains can also be concatenated by use of the '&'-operator. Thus, another legal time domain would be:

```
TAU = 1 & 2 & 5 & 10 & 20 & 50 & 100;
```

In the vocabulary of current CSSL's, the entries in a time-domain are the communications points.

5.4.2 The Trajectory

Most operations on time-domains result in variables of type trajectory. E.g. would

```
TR1 = SIN(TIME);
```

create a two-column table with the SINE-value computed for each entry in the time-domain and stored as the second column. Also on trajectories, operations can be performed. E.g. would

```
PLOT(TR1 //TITLE='SINE FUNCTION')
```

graph the sine function versus time.

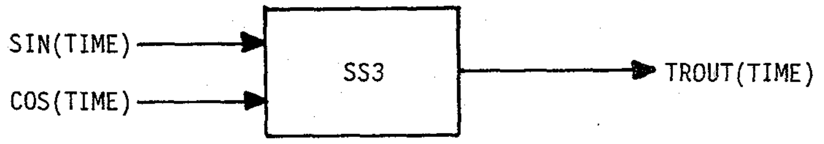
```
TRIN = <TR1;COS(TIME)>;
```

creates a column vector with two elements, each being a trajectory. That is, beside from scalars, polynomials, and systems, also trajectories may be elements of matrices (they must be specified over the same time-domain though).

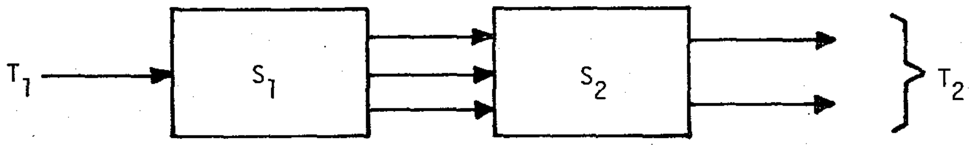
With this concept, an elegant way of formulating a simulation problem can be found.

```
TROUT = SS3*TRIN;
```

applies the two trajectories to the two inputs of the system SS3, performs a spline interpolation, executes a simulation of system SS3, and stores as a result the output trajectory in variable TROUT, sampled at the same communication points. That is: TROUT is a trajectory over the same time-domain.



It is now interesting to see whether the normal algebraic rules for the '*'-operator apply to this new definition as well. Let us check this at hand of the following example:



```
T2 = S2*S1*T1;
   = (S2*S1)*T1;
   ≟ S2*(S1*T1);
```

In the first formulation, our little system compiler is involved to create a new system for S2*S1, then one simulation is performed on this new system by use of the input trajectory T1. In the latter case, a simulation is performed on system S1 only, and the three resulting trajectories are stored in an intermediate trajectory vector. Then, this vector is once more interpolated to perform a second simulation, this time on system S2. It is quite clear that the results will be numerically different (in particular, if the time-domain is not dense enough to keep track of the high frequencies of system S1). However, it is quite evident as well that conceptually the equality holds.

5.5 Status of the Project.

IMPACT has been designed by the use of a general purpose LL(1) parser /BONG79/. Apart from a formal syntax, there also exists a preliminary version of the IMPACT User's Guide /RIMV83/. As IMPACT enhances the data structures of MATLAB significantly, it has been decided, not to extend the MATLAB expression parser, but to recode the IMPACT scanner and parser from scratch. This shall not be done by use of FORTRAN (as in the case of MATLAB), but rather by the use of ADA. IMPACT shall be coded as portably as possible. However, we shall not try to keep the code sufficiently small to make it implementable on a 16 bit machine without virtual memory. A typical machine to implement IMPACT on shall be a 32 bit machine with virtual memory (e.g. a VAX).

6. Perspectives for Further Development.

Although we are convinced that IMPACT shall mean a large step forward in CACSD software, we are aware of the fact that not even IMPACT can be the final answer to all problems. In particular, the I/O still creates some headache. There does not yet exist an exeptable standard for graphical I/O (for graphical <u>input</u> even less than for graphical <u>output</u>). The proposed database concept reduces this problem to some extent, as a large amount of advanced I/O-operations can be separated from IMPACT and placed in one or several separate programs linked to the same data base. These programs need not be implemented in the first version of IMPACT. Some very interesting results concerning graphical input were presented recently by H. Elmquist /ELMQ82/, more shall certainly follow, and we are convinced that these techniques shall once more revolutionize the CACSD software in the future. This, together with the rapid development in computer hardware, shall provide us with cost-effective implementations of sophisticated CACSD programs prior to the end of this decade.

There exists already a book on CACSD by H.H. Rosenbrock (/ROSE74/). The software engineering aspects of CAD systems are dealt with in a book by J.J. Allan (/ALLA77/). However, these books may already be a little too old, as publications in this field outdate very quickly. Two very interesting reports on the expected perspectives of CACSD software have been written recently by K.J. Åström /ASTR83/ and D. Birdwell /BIRD83/. An interesting Symposium on CACSD was organized not long ago by M. Mansour (/MANS79/). A brand new state-of-the-art of CACSD software has been edited by C.J. Herget and A.J. Laub (/HERG82/) as a special issue of the new IEEE Control Systems Magazine.

7. 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.

/ALLA77/ Allan, J.J.; CAD Systems; North-Holland; 1977.

/ASTR83/ Åström, 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.

/CELL77/ Cellier, F.E., et alia; "Educational Aspects of Development and Application of a Subprogram-Package for Control"; in Proc. of the IFAC Symposium on Trends in Automatic Control Education, Barcelona, Spain; Pergamon Press, London; 1977.

/DENH82/ Denham, M.J.; "Development of a Software Library and an Interactive Design Environment for Computer-Aided Control System Design"; IEEE Control Systems Magazine, December 1982.

/DORE80/ Van Doren, P.; "A Generalized Eigenvalue Approach for Solving Riccati Equations"; in Informal Proceedings, Conference on Numerical Algorithms in Control, Department of Automatic Control, Lund Institute of Technology, Sweden; 1980.

/ELMQ77/ Elmqvist, H.; "SIMNON, An Interactive Simulation Program for Nonlinear Systems"; in Proc. of SIMULATION'77, Montreux, Switzerland; Acta Press, P.O.Box 354, CH-8053 Zurich, Switzerland; 1977.

/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.

/FRED82/ Frederick, D.K.; "Software Summaries"; IEEE Control Systems Magazine; December 1982.

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

/GKS83/ GKS; Special Issue on Graphics Software, Informatik Spektrum, 6; 1983.

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

/MACI82/ Maciejowski, J.M. and G.J. MacFarlane; "CLADP, The Cambridge Linear Analysis and Design Programs"; IEEE Control Systems Magazine, December 1982.

/MANS79/ Mansour, M.; (Editor); Proc. First IFAC Symposium on CAD of Control Systems; Pergamon Press; 1979.

/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.

/ROSE74/ Rosenbrock, H.H.; Computer Aided Control System Design; Academic Press; 1974.

/SING84/ Singh, M.; (Chief Editor); Encyclopedia for Systems and Control; Pergamon Press; 1984.

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

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

/WISL78/ Wislander, J. and H. Elmqvist; INTRAC, A Communication Module for Interactive Programs, Language Manual; Report CODEN: LUTFD2/(TFRT-3149), Department of Automatic Control, Lund Institute of Technology, Sweden; 1978.

/WISL80a/ Wislander, J.; IDPAC Commands, User's Guide; Report CODEN: LUTFD2/(TFRT-3157), Department of Automatic Control, Lund Institute of Technology, Sweden; 1980.

/WISL80b/ Wislander, J.; SYNPAC Commands, User's Guide; Report CODEN: LUTFD2/(TFRT-3159), Department of Automatic Control, Lund Institute of Technology, Sweden; 1980.

/WISL80c/ Wislander, J.; MODPAC Commands, User's Guide; Report CODEN: LUTFD2/(TFRT-3158), Department of Automatic Control, Lund Institute of Technology, Sweden; 1980.