THE MATRIX ENVIRONMENT AS ENHANCEMENT TO MODELING AND SIMULATION.

Magnus Rimvall

Institute for Automatic Control

Swiss Federal Institute
of Technology (ETH)
Zürich, Switzerland

Francois Cellier

Dept. of Electrical and
Computer Engineering
The University of Arizona

Tucson, Arizona, U.S.A.

Abstract

In any future standard for continuous simulation languages, it is of primary importance that the data and program structures are made general and flexible enough. In this paper, the use of matrix structures in modeling and simulation and the impact of matrix environments on the design of simulation languages and their experimental frames will be discussed.

Introduction

Presently several efforts are undertaken to replace the CSSL'67 standard for continuous simulation languages with a more modern template. Since the publication of the CSSL report in 1967 [2] several successful commercial packages adhering to the standard have emerged, for example ACSL [10] and CSSL-IV [12]. Other languages (e.g. DSL/VS [6] and DARE [15]) differ syntactically from CSSL'67, yet retain many of its concepts. This has given the standard a quite remarkable life-span of almost twenty years in times of strong technological advancements, something which must be attributed to the open-endedness of the standard. Therefore, the products derived from CSSL'67 are not forced to be static, but have been enriched by assimilation of algorithms and concepts from other fields.

Despite this evolution of CSSL languages, there is now an imminent need for a new standard. Most of the hitherto developments within CSSL were of functional rather than structural nature, as any fundamental changes in the program/model structures would violate the standard. Consequently, the work of present standardization committees (e.g. in IMACS and SCi) concentrate on structural matters like the partition between model and experiment sections, the inclusion of discrete elements and the design of submodels facilities. In this contribution, we want to elaborate on some concepts which should be carefully considered during any development of a new standard: matrix data structures and ideas from interactive matrix/control environments.

Matrix environments

The term matrix environment designates a class of programs giving the user easy and interactive access to matrix manipulation algorithms. The first of these programs, MATLAB [11], was based upon the state-of-the-art numerical packages LINPACK [3] and EISPACK [4] [14]. Other programs (e.g. CTRL-C [8] and MATLAB-PC [9]) are derivatives of MATLAB, but also include algorithms for control theory and systems analysis (also called control environments). Yet another generation of packages (e.g. MATRIXx [16] and IMPACT [13]), retains the simple functionality of the previous programs, yet complements the matrix structure with structures like polynomial matrices, linear and nonlinear systems descriptions.

Common to all these packages are their ease of use. To illustrate this, let us calculate the stability of a linear system. We then enter the system matrix and calculate its eigenvalues:

```
>> A = < -1   -2    0
          1    0   -2
         -1   -1    2 >;

>> EIG(A)

ANSWER =

   -0.6348 + 0.6916i
   -0.6348 - 0.6916i
    2.2695 + 0.0000i
```

We note that one eigenvalue has a positive real part, indicating that the system is not stable.

With the same relative ease the matrix environment allows us to interactively define small algorithms. For example, it is possible to implement simple integration algorithms for small simulations. The following example illustrates the use of MATLAB to simulate a small linear system using the forward Euler algorithm:

```
//Simulation of Cedar Bog Lake
//----------------------------
// 1 //System order, final time
N = 5; TMAX = 10;
//State matrix
A = < -4.03   0.      0.      0.     0.
       0.48 -17.87    0.      0.     0.
       0.     4.85   -4.65    0.     0.
       2.55   6.12    1.95    0.     0.
       1.     6.9     2.7     0.     0. >
//Input vector
B = < 1 ; 0 ; 0 ; 0 ; 0 >
//Initial conditions
X0 = < 0.83 , 0.003 , 0.0001 , 0 , 0 >

// 2 //Eigenvalue computation
E = EIG (A), EEPS = EPS*ONES(E);
//Find eigenvalue determining crit. step size
F=NORM(((E.*CONJ(E))./(2*REAL(E)+EEPS)),'INF')

// 3 //Foreward Euler, small steps (dt=dtk/5)
DTK = 1/F;              DT = DTK/5
//Initial computations
X  = X0';               XST = <0,X0>;
AM = EYE(A) + DT*A;     BM  = DT*B;
NN = ROUND(TMAX/DT);    KK  = ROUND(NN/50);
K  = 1;                 IF  KK=0, KK=1; END

// 4 //Simulation
FOR I=1:NN,T=DT*I;U=95.9*(1+0.635*SIN(T-DT));...
       X = AM*X + BM*U; EXEC('STORE.MTL'); END

// 5 //Printout
EXEC('PRINTOUT.MTL')
```

Although this example is not very interesting from a simulation point of view, it shows the flexibility of the MATLAB command language and the relative ease with which new algorithms can be developed and tested. Note that only the first of the five sections is needed in environments with predefined integration algorithms (like CTRL-C or IMPACT).

Both of the preceding small examples illustrate the close conceptual and algorithmic connection between matrix environments and simulation languages. We should therefore ask ourselves the following questions:

- Several simulation languages (e.g. ACSL) allow the user to calculate the eigenvalues/vectors of a system by automatically linearizing the system from the Jacobian matrix around a working-point and returning the result in matrix form. Are there any other uses for matrices in general simulation languages?

- It is in matrix environments possible to perform rudimentary simulations, and thereby utilize the interactive environment as a flexible experimental frame. Would a similar interactive environment also be useable in simulation languages?

Matrix constructions as a modeling instrument

Most simulation languages allow for the declaration and use of matrices within the model description. However, although these matrices can be used at the discrepancy of the user, few languages utilize them to enhance their modeling capabilities. This is very unfortunate, as matrices can be used to simplify the solution to several modeling problems.

Consider the normal table-driven function. Although this is a good example of a multi-dimensional structure, practically all simulation languages force the user to enter the function-values as a linear list. A more readable approach, which also preempts all hard-to-find dimensional errors in the specification, would be to use a somewhat modified matrix-structure (internally represented as a matrix):

```
FUNCA = [ 0.0 | 0.0
          0.5 | 0.1
          1.0 | 0.33
          1.5 | 0.67
          2.0 | 0.9
          2.5 | 1.0 ];
```

This could easily be extended to two dimensions:

```
FUNC_2 = [     | 0.0,    0.5,    1.0
'comment  -----+--------------------'
          0.0 | 0.0,    0.1,    0.9
          0.5 | 0.1,    0.5,    1.3
          1.0 | 0.6,    1.2,    2.1 ];
```

Although no longer displayable in one plane, three-dimensional tables could use a similar notation by concatenating two-dimensional tables with the value of the third dimension in the free upper left hand field.

Scientists working in systems theory and automatic control are used to represent their linear systems in the time domain by a set of matrices:

$$A = \begin{bmatrix} 3, & 2 \\ 0, & 1 \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

are used to describe the system

$$\dot{x}\_1 = 3*x\_1 + 2*x\_2$$
$$\dot{x}\_2 = \qquad\quad x\_2 + u$$

where u is some input signal. On the other hand, simulationists traditionally describe their, sometimes identical, systems through differential equations and integrational blocks. Using the matrix structures from control theory as an extension to the traditional simulation elements, we enhance the readability and thereby also gain a very nice overview of the (sub-)system interconnections:

```
DIMENSION X(2), XO(2), A(2,2), B(2)

A = [ 3, 2
      0, 1 ];
B = [ 0
      1 ];

X = INTEG(A*X+B*U,XO)
U = ...
```

In addition to describing systems in the time-domain by matrices, control scientists also often represent systems in the frequency domain by transfer function matrices (rational functions containing polynomials). Correspondingly, most simulation languages provide block functions for elements described in the frequency domain. For example, the transfer function

$$G(s) = \frac{1.0}{1 + b*s}$$

is generally available as a function REALPL:

```
y = REALPL(b,x,ic)
```

However, some control environments allow the construction of general transfer functions. For example, to define the transfer function

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

IMPACT [13] allows the user to enter

```
G = (1/[9^5^9^1])
```

This notation could also be used in simulation languages, giving a much more general approach to frequency domain elements:

```
G = (1/[9^5^9^1]);
y = G*x;
```

(in which case the initial condition for x must be specified in the INITIAL section). For multi-dimensional systems, y and x would be vectors and G a transfer-function matrix.

We conclude that MATRIX CONSTRUCTIONS CAN CONSIDERABLY ENHANCE THE READABILITY, SECURITY AND/OR FLEXIBILITY OF A SIMULATION MODELING LANGUAGE.

Overloaded operations as modeling tool

In this chapter we will discuss how the "overloading" of operators can make modeling environments simpler yet more flexible. The term "overloading" derives from the new computer language Ada [1], where it describes the possibility to write several functions/procedures having the same name and differing only in their parameters. This, for example, enables the user to define a structure matrix and a function "+" containing a matrix multiplication algorithm. Thereafter the Ada-operator "+" is overloaded to work on matrices as well as scalars.

As present simulation languages do not support overloading, a user wishing to perform e.g. a matrix multiplication is left with two choices: to program a multiplication algorithm within the modeling environment or to call an external numerical routine for the operation. In CSSL-IV, the only CSSL simulation language including matrix manipulation algorithms, this call would take the form

```
"DECLARATION OF MATRICES"
       DIMENSION A(4,4), B(4,4), RES(4,4)
"PERFORM MATRIX MULTIPLICATION"
       RES = MMULTM(A,B,4,4)
```

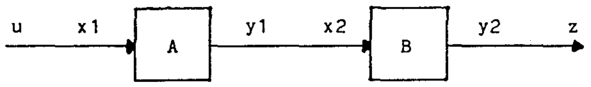
However, this complicated construction is completely unnecessary. As the model description is translated (normally into FORTRAN), simpler constructions could be used. The translator knows the dimensions of A, B and RES and thereby the parameters of the call to MMULTM, making the dimension declarations of the call redundant:

```
RES = A*B
```

This overloading of the multiplication operator also preempts user errors in the parameter specification.

Modern simulation languages provide an operator (often called INTEG) for scalar integration and another (INTVC) for vector operations. With the same argument as by the matrices, one of these operators is redundant (see the example using INTEG in the previous chapter).

We know that given two systems in transfer-function form, the cascading of these systems is described by a multiplication of the transfer-functions in reverse order. Generalizing this rule, a reverse-order multiplication of two systems (using an arbitrary representation) corresponds to a cascading of the systems and an addition of two systems describes a parallel connection. These rules can then be used to describe subsystem interconnections whenever these dynamic subsystems are described in modular form. For example, let us assume that A and B are two modules with one input and one output each, and the output of subsystem A (y1) is connected to the input of subsystem B (x2). The following program then describes the interconnection:



```
SUBSYSTEM A (y1 = x1)
...
END A

SUBSYSTEM B (y2 = x2)
...
END B

z = B * A * u
```

Matrix constructions can also be used to simplify more complex modeling structures. Standard continuous simulation languages are sometimes (mis-)used to simulate systems described by partial differential equations. A simple, one-dimensional diffusion problem can thereby be described by:

```
 DIMENSION u(50), dudt(50), uic(50)
 ..
 u = INTVC(dudt,uic)
 PROCEDURAL (dudt=u)
   DO 5 i=2,49
   dudt(i) = c*(u(i-1) - 2.0*u(i) + u(i+1))
5..CONTINUE
   dudt(1) = c*(leftu - 2.0*u(1) + u(2))
   dudt(50) = c*(u(49) - 2.0*u(50) + rightu)
 END
```

However, using matrix manipulations and the predefined, automatically scaled structures ONES (matrix with all ones), EYE (unity matrix), DIAG(V,K) (diagonal matrix with all elements of the vector V on the K'th diagonal) and E(I) (the I'th unity vector), the whole diffusion model can be reduced to:

```
INITIAL
  n = 50
  d = ONES(n-1,1)
  a = DIAG(d,-1) - 2*EYE(n) + DIAG(d,1)
  ..
DERIVATIVE
  u = INTEG(dudt,uic)
  dudt = c*( a*u + E(1)*leftu + E(n)*rightu )
  ..
```

We conclude: IN A SIMULATION LANGUAGE WITH PREDEFINED MATRIX DATA STRUCTURES AND MODULAR PROGRAM ELEMENTS, STANDARD OPERATORS CAN BE OVERLOADED. THIS ENHANCES THE READABILITY AND SIMPLIFIES THE LANGUAGE BY MAKING DUPLICATE OPERATORS FOR EACH STRUCTURAL ELEMENT REDUNDANT.

An algorithmic experimental frame

Early simulation packages (e.g. CSMP [5]) were designed for batch operations only. A "simulation job" normally consisted of a deck of punched cards describing not only the model to be simulated (in form of differential/algebraic equations) but also the experiment to be performed (length of simulation, variables to be printed/plotted and so on). When performing complex experiments on a given model, this environment is not flexible enough; the whole simulation (including translating and linking) had to be repeated after every, however small, change (for example by each rescaling of the plots or after changing of a single parameter value).

In more modern packages, a "run-time monitor" allows the user to interactively invoke simulations, change parameter values, create numerical or graphical output, and so on. However, the run-time commands available are normally too primitive (exception: DESCTOP [7]); no structural language elements are available (except for rudimentary macros) and, therefore, no procedural programming can be done on this interactive level.

Contrasting this rather limited interactive interface, the command interfaces of the matrix environments are considerably more flexible. As we already have seen in the introductory examples, small programs (algorithms) can be interactively defined and executed. Tests and decisions can be made reading the commands either from the terminal (interactive use), from a file (batch) or a combination of both (interactive execution using predefined macros/procedures).

As a first example on the use of procedural statements in an experimental frame for simulation, let us consider the problem of performing a *sensitivity analysis on a non-linear system.* As result we want to see the envelope of all time-histories when a parameter is changed over a range of values. Using a mixture of IMPACT and ACSL syntax, this could be described by:

```
   range_of_a = LINDOM(10.,20.,0.5); -- 21 points
   PREPARE(result,RUNS=all)
   FOR par IN range_of_a
    LOOP
      parameter_a = par;
      SIMULATE;
    END LOOP;
   PLOT(result,FORM=envelope)
```

In a conventional CSSL simulation language, this envelope could have been obtained only if the user had included the corresponding code in the model description. However, often the need for a sensitivity analysis like this rises only after some test runs have implied a high sensitivity, and then a reprogramming becomes necessary.

As another example of high-level run-time control, let us consider a system where we try to minimize an error parameter sys_error over the parameter par_a using a golden section algorithm. This small optimization problem is easily solved:

```
glob_lo = 0.0; glob_hi = 100.0;
new_lo  = glob_lo + .382*(glob_hi-glob_lo);
new_hi  = glob_lo + .618*(glob_hi-glob_lo);
par_a   = new_hi; SIMULATE; error_hi = sys_err;
par_a   = new_lo; SIMULATE; error_lo = sys_err;
WHILE ABS( error_hi - error_lo ) > 1.0e-3
 LOOP
  IF (error_hi < error_lo)
   THEN
     glob_lo = new_lo; new_lo = new_hi;
     new_hi = glob_lo + .618*(glob_hi-glob_lo);
     error_lo = error_hi;
     par_a = new_hi; SIMULATE; error_hi = sys_err;
   ELSE
     glob_hi = new_hi; new_hi = new_lo;
     new_lo = glob_lo + .382*(glob_hi-glob_lo);
     error_hi = error_lo;
     par_a = new_lo; SIMULATE; error_lo = sys_err;
  END IF;
 END LOOP;
par_a = (new_lo + new_hi) / 2.0
```

or the user can utilize any optimization algorithm already incorporated into the package in the standard fashion used in other procedural languages:

```
   FUNCTION error(x);
   BEGIN
     par_a = x; simulate; RETURN sys_error;
   END error;
   start = 50;
   result = NLP(START=start,FUNCTION=error, ...
                ERROR=1.0e-3)
```

Further cases where a procedural experimental frame can be used include statistical replication analysis, interactive specification of input signals and calculation and manipulation of steady-state and eigenvector/eigenvalue information. Also, it should be possible to directly connect the simulation system to a control environment.

We summarize: AN INTERACTIVE RUN-TIME MONITOR SUPPORTING PROCEDURAL LANGUAGE CONSTRUCTIONS AND MATRIX DATA STRUCTURES NOT ONLY SIMPLIFIES THE CONTROL OVER THE MODEL BUT ALSO OPENS UP A RANGE OF INTERESTING EXPERIMENTAL POSSIBILITIES.

Conclusions

By the development of new simulation standards, the possible inclusion of matrix elements as standard building elements in the modeling language should be seriously considered. Moreover, the experimental frame of a modern simulation language should be as flexible and versatile as the command languages of matrix/control environments.

References

[1] ANSI/MIL-STD 1815 A, Reference manual for the Ada programming language (January 1983).

[2] Augustin, D.C., J.C. Strauss, M.S. Fineberg, B.B. Johnson, R.N. Linebarger and F.J. Sansom. The SCi Continuous System Simulation Language (CSSL). Simulation, 9, (1967), p. 281-303.

[3] Dungorra, J.J, J.R. Bunch, C.B. Moler, and G.W. Stewart. LINPACK Users' Guide. Society for Industrial and Applied Mathematics (1979).

[4] Garbow, B.S., et alia. Matrix Eigensystem Routines, EISPACK Guide Extensions. Springer, Lecture Notes in Computer Science, 51 (1977).

[5] IBM. Continuous System Modeling Program III (CSMP III) Program Reference Manual, Program Number 5734-XS9, Form SH19-7001-2, IBM Canada Ltd., Program Product Centre, 1150 Eglington Ave. East, Don Mills 402, Ontario. (1972).

[6] IBM. Dynamic Simulation Language/VS (DSL/VS). Language Reference Manual, Program Number 5798-PXJ, Form SH20-6288-0, IBM Corporation, Dept. G12/Bldg. 141, 5600 Cottle Road, San Jose, CA 95193. (1984).

[7] Korn, G.A. DESCTOP Reference Manual, Version V2.0. University of Arizona, Tucson, AZ 85721. (1985).

[8] Little, J.N., et alia. CTRL-C and matrix environments for the computer aided design of control systems, in Proc. 6'th International Conference on Analysis and Optimization (INRIA), (Lecture notes in Control and Information Sciences 63, Springer Verlag, 1984).

[9] Little, J.N. and C. Moler. PC-MATLAB User's Guide. The MATH WORKS Inc. 124 Foxwood Rd. Portula Valley, CA 94025 (1985).

[10] Mitchell and Gauthier, Assoc. ACSL: Advanced Continuous Simulation Language - User Guide / Reference Manual, P.O.Box 685, Concord, Mass. (1981).

[11] Moler, C., MATLAB, User's Guide, Department of Computer Science, University of New Mexico, Albuquerque, USA, (1980).

[12] Nilsen, R.N. The CSSL-IV Simulation Language, Reference Manual. Simulation Services, 20926 Germain Street, Chatsworth, California. (1984).

[13] Rimvall, M., Cellier, F.C., A Structural Approach to CACSD. In Jamshidi, M. and C. Herget (Eds.), Advances in Computer-Aided Control systems Engineering. North Holland Press (1985).

[14] Smith, B.T. et alia. Matrix Eigensystem Routines, EISPACK Guide. Springer, Lecture Notes in Computer Science, 6 (1974).

[15] Wait, J. V. and Clarke III, D., DARE P User's Manual, Dept. of Electrical Engineering, University of Arizona, Tucson. (December 1976).

[16] Walker, R. et alia, MATRIX , A Data Analysis, System Identification, Control Design, and Simulation Package, IEEE Control Systems Magazine (December 1982).