System Representation

Åström, Karl Johan; Kreutzer, Wolfgang

1986

Document Version:
Publisher's PDF, also known as Version of record

CODEN: LUTFD2/(TFRT-7330)/1-6/(1986)

System Representation

Karl Johan Åström
Wolfgang Kreutzer

Department of Automatic Control
Lund Institute of Technology
October 1986

Department of Automatic Control
Lund Institute of Technology
P.O. Box 118
S-221 00 Lund Sweden

Document name
Report

Date of issue
October 1986

Document Number
CODEN: LUTFD2/(TFRT-7330)/1-6/(1986)

Author(s)
Karl Johan Åström
Wolfgang Kreutzer

Supervisor

Sponsoring organisation

Title and subtitle
System Representation

Abstract

The presentation of systems is a key issue in system theory and in computer aided control engineering. This paper discusses different ways to represent systems and suggests an approach based on object-oriented programming which admits hierarchical descriptions of system structure and behaviour. A prototype implementation of the ideas is described together with experiences of using it.

Key words

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title

ISBN

Language
English

Number of pages
6

Recipient's notes

Security classification

IEEE Third Symposium on Computer-Aided Control System Design, Arlington, Virginia

SYSTEM REPRESENTATIONS

Karl Johan Åström

Department of Automatic Control
Lund Institute of Technology
Lund, Sweden

Wolfgang Kreutzer

Department of Computer Science
University of Canterbury
Christchurch, New Zeeland

Abstract

The representation of systems is a key issue in system theory and in computer aided control engineering. This paper discusses different ways to represent systems and suggests an approach based on object-oriented programming which admits hierarchical descriptions of system structure and behaviour. A prototype implementation of the ideas is described together with experiences of using it.

1. INTRODUCTION

The notion of system is an essential element of control theory. The representation of systems is also a key issue in computer aided control engineering (CACE). Systems can be represented in many different ways. There are graphical representations like block diagrams, signal flow diagrams and bond graphs. There are also mathematical representations like state space models and input-output relations which come in many different forms, matrix fractions, impulse responses, frequency responses. When working with control systems it is frequently useful to use several different representations of a system.

Only fairly primitive ways of representing systems are used in current CACSD systems. Typical examples are the Matlab derivatives where systems are described by matrices. A slightly more sophisticated representation is used in the simulation language Simnon. This representation recognizes that a system has the properties, inputs, outputs and states. Simnon also allows a system to be described as an interconnection of subsystems. However only flat interconnections are allowed.

This paper presents a more flexible way to describe systems which is based on object-oriented programming. It is shown that a general structural description of hierarchically connected systems can be constructed from simple ingredients by making a system an object with the properties Name, Inputs, Outputs, Subsystems and Connections.

It is also necessary to add behavioural descriptions to obtain a useful tool. This is done by creating new objects which describe behaviour. A system can then inherit both structure and behaviour. Behaviour can be characterized in many different ways. A state description is one of the simpler alternatives. This can be covered by introducing the object StateBehaviour with the properties States, StateTransitionMap and OutputMap. The behavioural descriptions should also allow a given system to be described by models of different complexity. Apart from the detailed quantitative descriptions it is also useful to be able to deal with qualitative descriptions of behaviour.

The paper also describes a small prototype implementation which was designed to experiment with the ideas. This prototype which is written in Lisp admits hierarchical system representation and symbolic manipulation of the system descriptions. The experiments with the prototype indicate that the approach is one way towards implementation of powerful CACE systems.

The paper is organized as follows. Some system representations in current CACSD packages are described in Section 2. Requirements on system representations are given in Section 3. Sections 4 and 5 deal with representation of system structure and behavior. The prototype implementation is described in Section 7. Some conclusions are drawn in Section 8.

2. EXAMPLES

Some examples of system representations used in current CACSD packages are given in this chapter.

Matrix languages

Linear timeinvariant systems can be described using arrays. Such systems are conveniently handled in some matrix language like MATLAB, (Moler 1981) or its derivatives MatrixX, (Walker et al. 1982), CTRL-C, (Little et al. 1984). A system is represented as a matrix quadruplet in CTRL-C. In MatrixX it is represented as a system matrix and an integer which gives the order of the system. It is, however, clear that it is not sufficient to only have matrices. A detailed discussion of this is found in Åström (1984). There are a few more data types like polynomials and transfer functions in Blaise (Delebecque and Steer, 1985), Impact, (Rimvall and Cellier 1984) and Eagles, (Gavel et al 1986). A more sophisticated data structure for systems was used in the Lund packages (Åström 1985). Our experiences indicate that it would be very useful to have even more flexible concepts.

Simnon

The system description used in the simulation language Simnon, (Elmqvist 1977), includes system descriptions for continuous and discrete time systems. A continuous system corresponds to a state models described by an ordinary differential equation like

$$\begin{aligned} \frac{dx}{dt} &= f(x, u, t) \\ y &= g(x, u, t) \end{aligned} \tag{1}$$

where x is the state vector, u the input vector and y the output vector. The Simnon representation is

```
CONTINUOUS SYSTEM <system identifier>
INPUT <list of inputs>
OUTPUT <list of outputs>
STATE <list of states>
DER <list of derivatives>
TIME <variable>
Computation of outputs
Computation of derivatives
Parameter assignment
Initial value assignment
END
```

The standard state space model for a discrete time system is

$$\begin{aligned} x(t_{k+1}) &= f(x_k, u_k, t_k) \\ y(t_k) &= g(x_k, u_k, t_k) \end{aligned} \tag{2}$$

where $\{t_k\}$ is a sequence of sampling points. In Simnon such a system is described as

```
DISCRETE SYSTEM <system identifier>
INPUT <list of inputs>
OUTPUT <list of outputs>
STATE <list of states>
NEW <list of new states>
TIME <variable>
TSAMP <variable>

Computation of outputs
Computation of new values of the states
Update the TSAMP-variable
Modify states in continuous subsystems
Parameter assignment
Initial value assignment
END
```

Notice that this description is analogous to continuous systems. There is however a new variable TSAMP which gives the next time that the system should be sampled. In Simnon it is also possible to connect subsystems by using a connecting system which is described by

```
CONNECTING SYSTEM <system identifier>
TIME <variable>
Computation of inputs
Parameter assignment
END
```

The notation in Simnon is very natural for a control engineer. Long experience of using it has also shown that it is very easy to teach and use.

Discussion

The matrix based languages lack a proper system concept. This means that it is difficult to implement operations which are naturally viewed as operations on a system. We cannot make the natural abstractions used in system theory. Low level matrix operations have to be used instead. Simnon has a notion of systems. A drawback with this notion is however that it only admits flat interconnections. Particularly when dealing with large systems it would be desirable to have an hierarchical interconnection of sybsystem. One possibility to do this was suggested in Åström (1985). A more flexible approach is suggested in this paper. Since Simnon is a simulation language there are also only a limited number of system operations which are supported.

3. REQUIREMENTS

We will now briefly discuss some key issues in system representation. An important requirement is that the descriptions introduced should admit hierarchical system representations. Another is that it should be convenient to express systems composed of regular patterns of similar components in a convenient way.

To discuss suitable system descriptions we must also know how they are typically used. Typical operations on a system may be to:

- Combine several subsystems into a new subsystem.
- Expand a system into its subsystem.
- Find interconnected loops.
- Compute steady state operating points.
- Compute steady state input-output relations.
- Simulate.
- Linearize.
- Describe region of validity of linearized model.
- Analyse stability, reachability and observability.
- Make a Kalman decomposition.
- Compute system inverses.
- Compute sensitivity functions.
- Compute well-conditioned linear representations.
- Find linear characteristics like:
  - poles and zeros,
  - transfer functions,
  - frequency curves.
- Transform system representations.
- Perform and validate control design.
- Make model reductions.
- Fit parameters to experimental data.
- Find graphical representations.

Some of these operations are conveniently done numerically. Others require formula manipulation. It is therefore essential that the system can support numerical as well as formal calculations.

To describe systems it is thus necessary to have a rich structure which makes it possible to describe hierarchical interconnections of subsystems where each subsystem in turn is composed of subsystems. The subsystems may be of different types. They may be described in terms of state models, as input-output relations like impulse responses or transfer functions. We also have a need for descriptions of different complexity.

4. SYSTEM STRUCTURE

Representation of system structure is a key element when dealing with complex systems. Graphical representations, like block diagrams, signal flow diagrams and bond graphs are common for this purpose. They can be used to present details of subsystems as will as to give an overview of systems. In a block diagram description a subsystem is represented by a box and interconnections by lines between the boxes. See Figure 1. A line can represent a simple connection which tells that the variables at the connections are the same. It can also represent a more complex situation where several variables are involved. It is common practice to introduce arrows to indicate causality when this is possible. There may also be special symbols to denote simple operations like addition and multiplication of signals. There are many related descriptions like signal flow graphs and bond graphs.

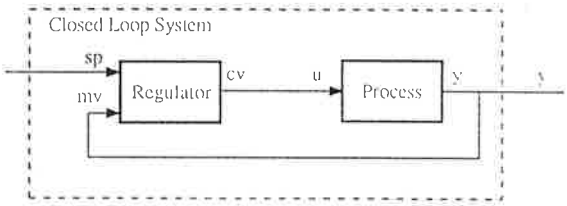

Figure 1. Example of a hierarchical block diagram. The ClosedLoopSystem is composed of two subsystems Process and Regulator.

To capture descriptions like block diagrams it is necessary to introduce the notions of subsystems and interconnections. In this paper we will only consider systems with well defined inputs and outputs. Such a system can be regarded as an abstraction of a box with input and output terminals. It can be represented as an object with the variables

```
Name
Inputs
Outputs
Subsystems
Connections
```

The inputs and outputs can be simple variables but they could also be objects with properties like units, range, etc. A primitive connection is a pair of input and output terminals. The connection implies that the corresponding terminal variables are the same. To avoid ambiguity the name of the associated system is also given. The notation (Regulator sp) denotes the terminal sp of the regulator. The system in Figure 1 can be represented as follows

```
Name ClosedLoopSystem
Input r
Outputs y
Subsystems Regulator Process
Connections r (Regulator sp)
(Regulator cv) (Process u)
(Regulator mv) (Process y)
y (Regulator y)
```

The regulator has the representation

```
Name Regulator
Input mv sp
Outputs cv
Subsystems nil
Connections nil
```

It thus has two inputs, mv the measured value and sp the set point. It has one output the controlled variable cv. The regulator has no subsystems and consequently no connections. In such a case the corresponding variables will be not be given.

The process in Figure 1 has the representation

```
Name Process
Input u
Outputs y
```

This notation is simple, natural and quite powerful.

Methods

A system structure has a number of associated operations. Examples of basic low level constructor and destructor function are

```
MakeSystem
AddInputs
DelInputs
```

Basic query and selector functions of the type

```
Inputs?
Inputs
```

are also needed to work with a system structure. The function Inputs? returns true if the subsystem has inputs and the function Input returns all inputs to a given subsystem. These functions operate on one level only. There are also primitive display functions like

```
ShowSystem
```

and a system editor which admits a structured editing of a system.

The functions discussed so far relate to a particular system only. For a system with subsystems it is also of interest to find all attributes of the system and of all associated subsystems. This is done by the functions

```
AllInputs
AllOutputs
AllSubsystems
AllConnections
```

It is sometimes also desirable to show the attributes hierarchically corresponding to the the subsystem hierarchy. This is done by the function.

```
HierarchyOfInputs
HierarchyOfOutputs
HierarchyOfSubsystems
HierarchyOfConnections
```

Several functions are useful in order to explore the structure of a system. There are some auxiliary functions which are useful to explore the connections. The functions

```
InputsConnectedTo
OutputsConnectedTo
```

return the systems which are connected to the input and output terminals of a given system. The function

```
ContainedIn
```

gives all the systems which contain a given system as a subsystem.

A loop is a closed path obtained by scanning a connection of subsystems in the direction defined by the input-output causality. The functions

```
Loop?
Loop
AllLoops
```

tell if a given subsystem is contained in a loop, gives a loop associated with a given subsystem and all loops associated with a given system. These functions can be used to trace a given collection of subsystems.

System Operations

It is convenient to have a number of operations which act on a system and generate new systems. This is accomplished by the functions

```
Aggregate
Disaggregate
```

The function Aggregate applies to a collection of subsystems and gives a new subsystem. The appropriate connections are generated from the aggregated subsystems.

There are also a number of other system operations which are useful to form composite systems from simple ingredients. Typical examples are

```
Invert
ParallelConnect
SeriesConnect
FeedbackConnect
```

These operations will have to operate on many different properties of a system. They will create new system with the appropriate properties.

5. SYSTEM BEHAVIOR

Only topological properties of a system i.e. structure and interconnections, have been discussed so far. To describe a system it is also necessary to describe how it behaves. Systems behavior is a very rich field. Examples of categories of behavior are

```
Static
Qualitative
StateSpace
StochasticStateSpace
StochasticInputOutput
LinearStateSpace
TransferFunction
TransientResponse
DescribingFunction
```

These categories can be described as objects. The static behavior can be described by a nonlinear function. Several methods are needed to work with static behavior e.g.

```
FindOutput
FindInput
Linearize
MaxGain
MinGain
DegreeOfLinearity
OperatingRange
```

Qualitative behavior attempts to describe some gross properties of a system like gain, time constants, and estimate of largest dynamic gain, some measure of nonlinearity, and some measure of how deterministic a system is. Associated with these properties we also need methods to obtain these properties from the more detailed representations. The ideas developed for automatic tuning of regulators are quite useful for this purpose. See Åström and Hägglund (1984). It is quite useful to allow qualitative system descriptions because it allows qualitative reasoning about system properties. In large systems composed of many subsystems we may e.g. neglect an interconnection if a system with a very low gain is connected in parallel with a system with a very high gain. In a simulation where we are exploring phenomena in a time scale of minutes we may use static models for subsystems whose time constants are less that one tenth of a second.

The state space and the transfer function behavior are well described in standard texts on control engineering. In this paper we will only use some simple forms of nonlinear state space behavior.

It is also very useful to introduce the ValidityRange property to indicate the region of validity of the model. This can be described as a subset of the product of the input spaces and the state spaces. With such a feature it is possible to write a simulation program which will raise an exception if the state of the system goes outside the region of validity during a simulation.

6. A PROTOTYPE IMPLEMENTATION

A small prototype program was written to test the ideas described in the previous sections. The program was written in ExperLisp on the Macintosh. The main goal was to experiment with descriptions of system structure. A secondary goal was to try some formula manipulation. The prototype can only handle a simple type of system behavior namely continuous time state model behavior. The version of ExperLisp that we used does unfortunately only support object-oriented programming to a limited degree. Our implementation was based on the `defstruct` function which can be used to generate a structure.

A system was implemented as structure with seven named components (slots)

```
Name
Inputs
Outputs
Subsystems
Connections
States
Behavior
```

The first five slots are used to describe system structure and the last two to describe system behavior. The system behavior is characterized in terms of the functions f and g in equation (1). The function f gives the rate of change of the state and the function g is the output map. These functions are given as Lisp functions.

The prototype has primitive constructor, destructor, query, selector and display functions for the system properties. The higher level operations include functions like

```
ShowSubsystemHierarchy
GetAllSubsystems
```

for all system variables. There are also operations to explore the system structure and tools for linearization of system behavior.

An Example

Consider the model following control system shown in the block diagram in Figure 2. The system named S1 has three subsystems Model, FF and S2. The system S2 has also three subsystems Reg, Proc and Sensor. The inputs, outputs and subsystems are shown in the figure. The connections can also be read from the figure. State variables are introduced to describe the behavior of the systems. These are also shown in the figure. The behavior of each system is described by the functions f and g in the model (1). The properties of the subsystems are given below.

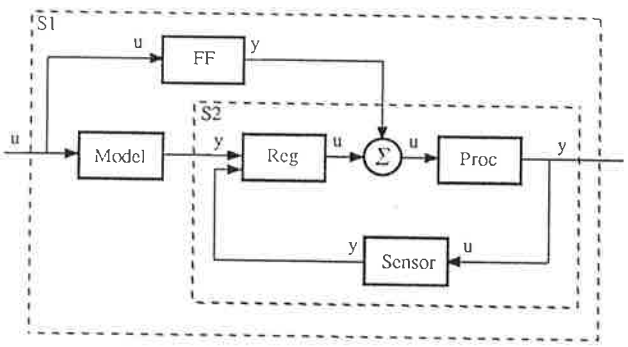

Figure 2. Block diagram of a model following control system.

Model Desired closed behavior.

```
Input u
Output c
States x1 x2
Behaviour
```

$$\frac{dx_1}{dt} = \omega x_1$$

$$\frac{dx_2}{dt} = \omega(-x_1 - 2\varsigma x_2 + c)$$

$$y = x_1$$

FF Feedforward compensation.

```
Input u
Output y
State x
Behaviour
```

$$\frac{dx}{dt} = -ax + (b-a)x$$

$$y = k(x+u)$$

Reg Simple PID regulator.

```
Input r y
Output u
State i d
Behaviour
```

$$\frac{dd}{dt} = \frac{N}{T_d}(y-d)$$

$$\frac{di}{dt} = \frac{1}{T_i}(r-y)$$

$$u = k(r - y + i + N(d-y))$$

Process Process dynamics.

```
Input u
Output y
State x1 x2
Behaviour
```

$$\frac{dx_1}{dt} = -a\sqrt{x_1} + bu$$

$$\frac{dx_2}{dt} = a\sqrt{x_1} + c\sqrt{x_2}$$

$$y = x_2$$

Sensor Sensor dynamics.

```
Input u
Output y
State x
Behaviour
```

$$\frac{dx}{dt} = \frac{1}{T}(u-x)$$

$$y = \frac{x}{1+x^2}$$

In our prototype we have not put much effort into the user interface. We are simply using the ordinary Lisp functions.

Sample Dialog

The simple dialog below illustrates how the system works. The function call (**ShowSystem** S1) generates the following listing of the description of the system S1

```
==============================
SYSTEM: S1
==============================
Inputs :  (c)
Outputs :  (y)
States :  nil
*** Subsystems ***
++++++++++
(S2 Model FF)
*** Connections ***
++++++++++++
(c S1)
(S1 y)
*** Behaviour ***
+++++++++
+++ no state equations defined +++
+++ no output equations defined +++
nil
```

There are several tools to explore the system structure. The function call

(AllSubsystems S2)

returns all subsystems of S2 i.e.

```
(Reg Proc Sensor)
```

If the function is applied to a system whose subsystems also have subsystems all lower level subsystems are returned. An example is

(AllSubsystems S1)

which returns

```
(S2 Model FF Reg Proc Sensor)
```

The function **AllSubsystems** AllSubsystems can also be applied to a list of several systems as in

(GetAllSubsystems '(S1 S2))

which returns

```
(S2 Model FF Reg Proc Sensor)
```

The function **GetSubsystemHierarchy** can be used to get the detail of the subsystem structure. An example is

(GetSubsystemHierarchy '(S1))

which returns

```
(S1 (S2 Model FF) S2 (Reg Proc Sensor) Reg nil
Proc nil Sensor nil Model nil FF nil)
```

To list the hierarchy of subsystems we can use the function

(ShowSubsystemHierarchy S1)

which generates

```
++++++++++++++++++++++++++++++++++++++++
Subsyststems of:  S1
++++++++++++++++++++++++++++++++++++++++
S2
  Reg
  Proc
  Sensor
Model
FF
nil
```

There are similar functions to investigate the other system variables. The function call

(AllStates 'Model)

returns all states of a system i.e.

```
((x1 x2) nil nil nil)
```

and the function

(GetAllStates '(Reg Proc Sensor))

returns all states of a list of systems

```
((i d) nil ((x1 x2) nil ((x) nil nil)))
```

The function call

(TableOfConnections Proc)

returns the table of connections for the system Proc i.e.

```
((u Proc)(Proc y))
```

The connections are printed using

(ShowConnectionTable (TableOfConnections Proc))

which generates

```
(u Proc)
(Proc y)
nil
```

The function call

(StateEqns (Behaviour proc))

returns the function f which gives the rate of change of the state, i.e.

```
(((- (a * (x1 ** (1 / 2 )))) + (b * u))
((a * (x1 ** (1 / 2 ))) - (a * (x2 ** (1/ 2 )))))
```

The output map g of the system proc is obtained by

(OutputEqns (Behaviour Proc))

which returns

```
((x2))
```

The functions f and g which give the behaviour of the system are listed by the function call

(ShowBehaviourList (Behaviour Proc))

which results in

```
State-Eqns
----------
((- (a * (x1 ** (1 / 2 )))) + (b * u))
((a * (x1 ** (1 / 2 ))) - (a * (x2 ** (1 / 2 ))))
Output-Eqns
-----------
(x2)
nil
```

A system can be linearized through

(ShowBehaviourTable (Linearize Proc))

The system is then first linearized using the the function Linearize which generates a list of linearized equations. This list is then printed using the command ShowBehaviourTable. The result is as follows

```
System - Matrix
---------------
(a * ((1 / 2 ) * (x1 ** ((1 / 2 ) - 1 )))) 0;
(a * ((1 / 2 ) * (x1 ** ((1 / 2 ) - 1 ))))
(- (a * ((1 / 2 ) * (x2 ** ((1/ 2 )- 1 )))))
Input - Matrix
--------------------------
b
0
Measurement - Matrix
--------------------
0 1
nil
```

7. CONCLUSIONS

The main purpose of this paper has been to explore new ways to describe interconnected systems using object-oriented programming. It is relatively easy to implement a system like the prototype in Lisp. Implementation is much easier in a programming environment which supports object-oriented programming with a powerful inheritance mechanism.

Our experiences indicate that the system descriptions proposed are natural and easy to work with. A complete system for describing system structure can be implemented following the ideas outlined in the paper. A few additional functions are needed to explore system structure. The extension to systems with bidirectional interaction is straightforward. The user interface can be improve considerably by incorporating graphics of the type described in Elmqvist and Mattsson (1986). Work along these lines is under way.

Much more work is required to obtain a system which can describe system behavior in a reasonably complete way. Our experiments indicate that such systems are much easier to implement in an object-oriented environments like Flavors, Loops or Object Lisp which support inheritance.

Acknowledgements

The results of this paper were obtained as part of the project Computer Aided Control Engineering (CACE) at Lund Institute of Technology. We are grateful to the National Swedish Board of Technical Development (STU) who has supported this project under contract 85-4808. We also would like to thank Sven Erik Mattsson and Karl Erik Årzén for many useful discussions.

REFERENCES

ÅSTRÖM, K.J. (1983): "Computer Aided Modelling, Analysis and Design of Control Systems—A perspective," *IEEE Control Systems Magazine*, Vol. 3, No. 2, May 1983, 4–16.

ÅSTRÖM, K.J. and HÄGGLUND, T. (1984): "Automatic Tuning of Simple Regulators with Specifications on Phase and Amplitude Margins," *Automatica* 20, No. 5, 645–651.

ÅSTRÖM, K.J. (1985): "Computer Aided Tools for Control System Design—A perspective," in M. Jamshidi and C.J. Herget (Eds.): *Computer-Aided Control Systems Engineering*, North-Holland.

DELEBECQUE, F. AND S. STEER (1985): "The Interactive System Blaise for Control Engineering," CADCE '85, 3rd IFAC/IFIP Symposium on Computer Aided Design in Control and Engineering Systems, Lyngby, Denmark.

ELMQVIST, H. (1975): "SIMNON - User's Manual," Report CODEN: LUTFD2/TFRT-3091, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

ELMQVIST, H. (1978): "A Structured Model Language for Large Continuous Systems," Report CODEN: LUTFD2/TFRT-1015, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

ELMQVIST, H. (1985): *LICS - Language for Implementation of Control Systems*, Report CODEN: LUTFD2/TFRT-3179, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

ELMQVIST, H. AND S.E. MATTSSON (1985): "A Simulator for Dynamical Systems Using Graphics and Equations for Modelling," *Proc. of the IEEE Control Systems Society Third Symposium on Computer-Aided Control System Design (CACSD)*, Arlington, Virginia.

GAVEL, D.T., C.J. HERGET, AND B. S. LAWYER (1986): "The M Language—An Interactive Tool for Manipulating Matrices, Systems and Signals.," Dynamics and Controls Group, Engineering Research Division, Lawrence Livermore National Laboratory, Livermore, California.

INTEGRATED SYSTEMS, INC (1984): "Matrix-X Users's Guide, Matrix-X Reference Guide, Matrix-X Training Guide, Command Summary and On-line Help," Integrated Systems, Inc. 101 University Avenue, Palo Alto, California.

LAWVER, B. and P. POGGIO (1985): "EAGLES Requirements," Computer Systems Research Group, Engineering Research Division, Lawrence Livermore National Laboratory, Livermore, California.

LITTLE, J.N., A. EMAMI-NOEINI, and S.N. BANGERT (1984): "CTRL-C and Matrix Environments for the Computer-Aided Design of Control Systems," in Bensoussan and Lions (Eds.): *Analysis and Optimization of Systems*, Lecture Notes in Control and Information Sciences, Springer-Verlag, Berlin, Also in Jamshidi, M. and C.J. Herget (eds) Computer-Aided Control Systems Engineering, North-Holland, Amsterdam, The Netherlands, 111–124.

MOLER, C.B. (1980): "MATLAB - User's Guide," Department of Computer Science, University of New Mexico, Albuquerque, USA.

RIMVALL, M. (1983): "IMPACT, Interactive Mathematical Program for Automatic Control Theory, User's Guide," Department of Automatic Control, Swiss Federal Institute of Technology, ETH-Zentrum, Zurich, Switzerland.

RIMVALL, M and F. CELLIER (1984): "IMPACT Interactive Mathematical Program for Automatic Control Theory," in Bensoussan and Lions (Eds.): *Analysis and Optimization of Systems*, Springer Lecture Notes in Control and Information Sciences, Springer, Berlin.

RIMVALL, M. and F.E. CELLIER (1985): "A Structural Approach to CACSD," *Computer-Aided Control Systems Engineering*, North-Holland, Amsterdam, The Netherlands, pp. 149–158, In Jamshidi, M. and C.J. Herget (Eds.).

SHAH, S.C., M.A. FLOYD AND L.L. LEHMAN (1985): "MATRIX-X: Control and Model Building CAE Capability," in Jamshidi, M. and C.J. Herget (Eds.): *Computer-Aided Control Systems Engineering*, North-Holland, Amsterdam, The Netherlands, pp. 181–207.

SYSTEMS CONTROL TECHNOLOGY (1984): "CTRL-C A Language for the Computer-Aided Design of Multivariable Control Systems, Users Guide," Systems Control Technology, 1801 Page Mill Road, Palo Alto, CA.

VANBEGIN, M. and P. VAN DOOREN (1985): "MATLAB-SC, Appendix B: Numerical Subroutines for Systems and Control Problems," Technical Note N168, Philips Research Laboratories, Bosvoorde, Belgium.

WALKER, R., C. GREGORY, JR. AND S. SHAH (1984): "MATRIX-X: A Data Analysis, System Identification, Control Design and Simulation Package," Integrated Systems, Inc., Palo Alto, California.

WALKER, R., S.C. SHAH AND N.K. GUPTA (1984): "Computer-Aided Engineering (CAE) for System Analysis," *Proc. of the IEEE*. 72, No. 12, 1732–1745.