THE DIFFERENT MODELING CAPABILITIES OF IMPACT.

Magnus Rimvall, Fredi Schmid

Institute for Automatic Control
Swiss Federal Institute
of Technology (ETH)
Zürich, Switzerland

François Cellier

Dept. of Electrical and
Computer Engineering
The University of Arizona
Tucson, Arizona, U.S.A.

ABSTRACT

Traditionally, two different classes of computer programs have served control engineers as tools for systems-modeling; nonlinear simulation packages and linear CACSD programs. In this paper, we show how nonlinear modeling and simulation facilities have been incorporated within the IMPACT linear control environment. A detailed example illustrates the flexibility of this system when dealing with systems having mixed linear and nonlinear components.

INTRODUCTION

During design of controllers for physical, nonlinear systems, some variant of the following iterative three-step overall scheme is often employed:

1/ the system is modeled and analyzed through nonlinear simulation

2/ the model is linearized for further analysis and linear design of the controller

3/ the thus obtained controller is tested on the nonlinear model through simulation.

Until recently, at least two different software packages were needed during these three stages.

- For step one and three, simulation languages such as ACSL [6] and CSSL-IV [8] are employed for modeling and simulation of nonlinear systems described through differential equations. These packages are not specifically developed for control-engineers, but to meet the simulation needs of a much more heterogeneous group of engineers; therefore they seldom include any control algorithms.

- For step two, linear analysis and design algorithms may be accessed through one of over 50 linear control-packages available [4]. However, few of these packages directly support a nonlinear modeling capability.

Because of the limited scope of each of these two classes of software, models and/or data have to be transferred several times between different programs. As this can become quite cumbersome and time-consuming, several control and/or simulation packages try to connect these two worlds. Hence, links have been constructed between nonlinear simulators and linear control-tools through linearization or eigenvalue algorithms. Unfortunately, such links do not remove the following drawbacks:

- the full expression power of an algorithmic, command-driven control environment can not be used to control nonlinear simulations.

- whereas modern control environments have a very natural and fast-to-use notation for entering and manipulating linear systems, it is much more awkward to enter linear (sub-)models in conventional nonlinear simulators [12].

- the user has to master two program interfaces.

- in control education with computer exercises, the limited time available makes it hard to introduce one complex software package during a lecture, let alone two. Thus, control exercises normally have to be limited to the linear part of a complete design cycle.

A better approach is the inclusion of nonlinear modeling and simulation facilities within a linear CACSD-package. In MATRIXx [13], a graphical editor (SYSTEM_BUILD) allow the user to enter nonlinear models. These models can also be invoked as fixed entities from the alphanumeric command-driven part for linearization and simulation. In IMPACT, nonlinear capabilities have been built into the interactive command-language itself. This makes nonlinear models better accessible from the linear part of the package, as will be shown in this paper.

Both the IMPACT and MATRIXx approach makes it easier to perform mixed linear/nonlinear modeling. Although this does not automatically enhance the operations we can perform on the different classes of systems, it certainly simplifies the handling of a system through the whole control cycle. Also, linear subparts can be described more easily than is normally the case in nonlinear simulators.

IMPACT

IMPACT (Interactive Mathematical Package for Automatic Control Theory) is an interactive CACSD-kernel system presently being implemented at the Swiss Federal Institute of Technology (ETH) [10], [11]. It belongs to the group of control environments which are, at least conceptually, based on the MATLAB package [7] [4]. Unlike other control environments, most of IMPACT has been implemented in Ada [1]. It is presently composed of more than 50 000 lines of Ada code for the kernel-system and a set of FORTRAN-coded numerical algorithms.

The development of IMPACT is made with the objective of serving a very heterogeneous group of users ranging from the undergraduate student to the skilled control researcher with programming experience. Therefore, the following basic concepts have guided the design of the IMPACT user-interface:

- the basic commands must be fast and easy-to-use. Thus, a parser accepting free-form input-data has been implemented. The used expression-syntax is an extension of the one used by MATLAB [7].
- an algorithmic interface is needed for flexibility. IMPACT accepts statements following a syntax similar to that of Ada. Functions and procedures may be interactively defined.
- the transition from basic to advanced use must be gradual. In IMPACT, the user calling complex functions can switch from the command-language interface to a question-and-answer interface (the query-facility) at will.
- different, but equivalent, entities should be treated equally. This is particularly true for the modeling of systems. IMPACT will therefore use the same language constructions to simulate, manipulate or interconnect system descriptions, regardless if these are linear or nonlinear, single-level or hierarchical.
- small and large systems should be equally treated by the user-interface.

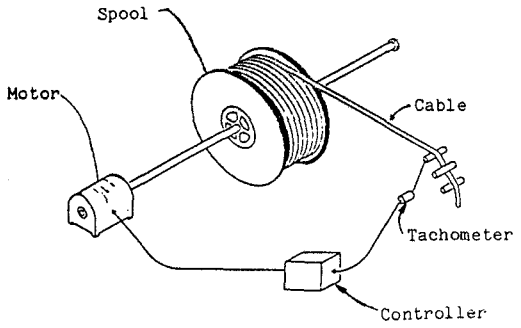


Figure 1. The treated cable system.

LINEAR SYSTEMS MODELING IN IMPACT

To illustrate the versatility of IMPACT in modeling control-related systems, the benchmark cable-spool system depicted in Figure 1 will be used. Linear approximations are used for the motor and tachometer, as shown in the block-diagram picture of Figure 2. The spool is nonlinear.

We will start by defining representations for all linear subsystems of Figure 2. As IMPACT supports polynomial matrices as well as rational matrices, we can enter the two transfer functions of the motor and the tachometer directly. We commence by defining the proportional controller KP and thereafter we define the two rational functions (">>" is the IMPACT prompt to indicate that it is waiting for further interactive input):

```
>> KP    = 1;

>> TACHO = 3/[1^0.5];

>> MOTOR = 7./[1^1];
```

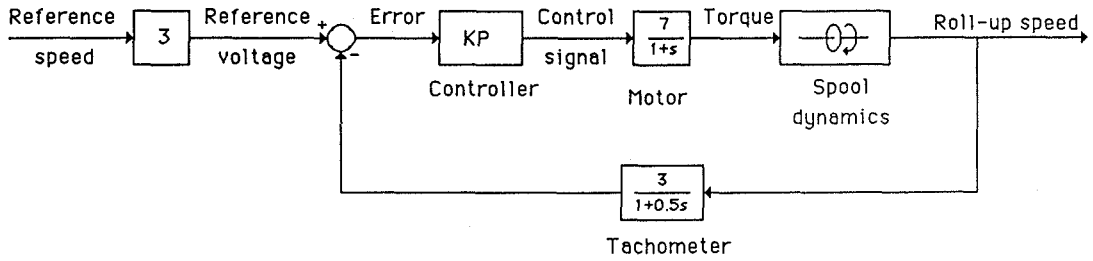


Figure 2. Block-diagram of the treated system.

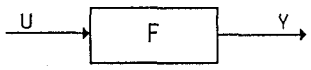
To illustrate how linear systems are connected and simulated, let us neglect the spool-nonlinearity caused by changing radius and mass of the roll. Then the spool transfer function is given by

```
>> SPOOL = 0.019/_s;
```

We have now entered all elements of the system in linear transfer-function form. As multiplication has been defined for rational functions in IMPACT, a representation of two cascaded systems is correctly obtained through this operation. In addition, the operator "\\" has been included for the feedback loop. Thus, the total transfer function of our system is

```
>> SYS = ( SPOOL * MOTOR * KP \\ - TACHO ) * 3;
```

where SYS of course again is a rational function scalar. Simulation has also been defined as a trivial operation. Given the structure



the output signal Y is obtained by multiplying the system-representation F with an input signal. In IMPACT, signals are represented as trajectories containing arbitrary values defined over a collection of independent points (domain). For simulations, these independent points of course must take increasing (decreasing) values on the real axis; for frequency responses they have to lie on the imaginary axes. Hence, if we wish to simulate the step-response for ten seconds with a 0.1 second resolution on the output, we define the time-domain

```
>> TIME = LINDOM(0,10,101);
```

and the trajectory (signal)

```
>> U = ONES(TIME);
```

The simulation is thereafter invoked as

```
>> VOUT = SYS * U
```

The result is graphically displayed and saved in the trajectory (signal) VOUT for later use.

IMPACT supports not only frequency-domain, but also time-domain models. Thus, having entered the A, B, C and (optionally) D-matrix of a linear state-space representation, a system is formed by:

```
>> SYS = LINSYS(A,B,C);
```

As in the frequency-domain, state-space system interconnections can be made using the operations "*", "+" and "\\" for series, parallel and feedback structures, respectively. Moreover, the more general connection mechanism of nonlinear systems presented in the next section can be used on linear systems as well.

NONLINEAR SYSTEM REPRESENTATIONS IN CACSD

Apart from the obvious advantage of avoiding transportation of data between different software tools, the main advantage of incorporating nonlinear modeling capabilities into interactive, command-driven CACSD-packages is that the full expression-power of the CACSD-package can be used. This simplifies the incorporation and use of nonlinear tools such as

- nonlinear time-simulators. It should be possible to use some model-external driving function. Results of the simulation should be available in trajectory form for further usage. package.
- algorithms for nonlinear controller design.
- nonlinear sensitivity analysis either through linearization algorithms or over a worst-case multiple-simulation approach.
- a linearizer creating models for the linear operations of the control package. If a linear controller-design is made, it must be possible to use this linear controller within the nonlinear model for verificational simulations.
- tools for hierarchical modeling of larger nonlinear systems using modular subsystems.

Although a nonlinear (sub-)system is described through a fixed set of equations, these equations are accessed completely differently in each of these tools, requiring a model to have alternative outward interfaces. In the most general case, all of the following interfaces may be needed for a single model definition:

- parameter constants. During the modeling and/or design phase, individual physical parameters of a system may not be known or not yet determined. In particular, the definition of predefined submodels with free, changeable parameters should be supported.
- input and output signals. In control environments where simulations are invoked as simple operations (e.g. as multiplications between a system and a trajectory), inputs and outputs must be definable for each (sub-)system.
- initial conditions for simulation and equilibrium points for linearization.
- submodel connections. For simple submodel interconnections, the overloaded operations for parallel, series and feedback connections suffice. However, for a general modular modeling, possible interconnections ("cuts") between submodels must be defined in each subsystem. These cuts are then be connected to each other in the hierarchically higher system.

Hence, the descriptive language for nonlinear models must be quite versatile. In particular, to be able to submit a submodel to all the mentioned nonlinear tools, generic model-interfaces have to be defined.

NONLINEAR MODELING IN IMPACT

We will in this section show how all nonlinear operations can be invoked without increased complexity compared to the linear case. The nonlinear model for the spool of our example is:

$$v = r * av$$

$$\frac{dav}{dt} = \frac{torque}{inertia}$$

$$inertia = cable * r^4 + roll$$

$$\frac{dr}{dt} = -k1 * av$$

$$k1 = \frac{d * d}{2 * \pi * w}$$

$$d = 0.03$$
$$w = 0.6$$
$$cable = 23.5$$
$$roll = 2.1$$

where "v" is the roll-on/-off speed, "r" the instantaneous radius of the roll and "av" the angular velocity of the spool. The two constants "d" and "w" denote the cable diameter and spool width. Typical values of "w", "d" and the inertia constants "cable" and "roll" are indicated to the right.

As the model we wish to enter contain more than a dozen lines, the risk of making errors during a line-by-line direct entry is relatively large. We therefore invoke the systems editor from within IMPACT and define the system using this editor.

```
SYSTEM Spool(r0,d,w    : SCALAR)
             IN torque : SCALAR
             RETURN v  : SCALAR IS
  --
  r       : STATE := r0;
  av      : STATE := 0.0;
  inertia : SCALAR;
  cable   : SCALAR := 23.5;
  roll    : SCALAR := 2.1;
  k1      : SCALAR := d*d/(2*_pi*w);
BEGIN
  v       = r*av;
  inertia = cable*r**4 - roll;
  av`     = torque/inertia;
  r`      = -k1*av;
END Spool;
```

This system definition declares a spool-template with not yet determined initial condition "r0" and system constants "d" and "w". These free parameters of the defined model must be specified by all operations on the spool. The system-declaration header also includes information on input and output parameters to the system, which are implicitly used when we for example perform a simulation using commands identical to those used for linear systems:

```
>> VOUT=(SPOOL(1.2,0.03,0.6)*MOTOR*KP\\-TACHO)*3*U
```

In the shown examples, default integration parameters have been used during simulation. For complete simulation control, each trajectory may contain additional information on integrational methods, step sizes, error conditions et cetera.

If we wish to "freeze" the parameter-values and initial conditions, we may do so by creating a new nonlinear system as:

```
>> MYSPOOL = SPOOL(1.5,0.03,0.6);
```

Whenever a model is invoked without the correct number of parameters and if specially formatted textual information has been included in the model definition (see [11] for details), the powerful IMPACT query facility will jump into action. For example, if we perform a simulation without specifying the free parameters, as in

```
>> VOUT = ( SPOOL * MOTOR * KP \\ - TACHO ) *3*U
```

IMPACT will ask for the missing parameters (user input is underlined):

```
S>>The nonlinear system SPOOL has been invoked
S>>with missing parameters. This system models
S>>the roll-off of a cable from a spool.
S>>Please enter missing parameters.
S>>
S>>Initial roll-diameter R0  (NO DEFAULT): 1.2
S>>Diameter of the cable D   (NO DEFAULT): 0.03
S>>Width of the spool     W  (NO DEFAULT): 0.6
```

If the user is uncertain on the meaning of a particular parameter, he can enter a HELP for further information.

```
S>>Initial roll-diameter R0  (NO DEFAULT): HELP
S>>This parameter indicate the thickness of the
S>>layers of cable on the spool at the outset
S>>of the simulation (this thickness includes
S>>the radius of the spool axis).
S>>
S>>Initial roll-diameter R0  (NO DEFAULT): 1.2
```

The query facility is particularly useful for systems with many parameters and for cases wnere the user and the constructor of a system are different persons. Especially students using predefined models appreciate this question-and-answer mode as a complement to the faster, but more complex, command-driven input.

The other operations working on nonlinear systems, with the exception of the hierarchical modeling, are invoked through commands implemented in the normal command-language of IMPACT. For example, a linear state-space representation of SPOOL may be calculated by symbolic linearization [9]:

```
>> LINSPOOL = LINEARIZE(SPOOL(1.2,0.6,0.03));
```

Also here, the query facility would jump in wnen either the function LINEARIZE or system SPOOL (or both) was missing a parameter.

Yet another example illustrates the use of the IMPACT command language as an experimental frame for simulation runs. Assume that the parameters "r0" "d" and "w" are known with 10% precision and that we wish to plot the envelope of the step-responses from all worst-cases of this parameter-variation [2]. We would then create a matrix where each row corresponds to a set of parameters:

```
>> MEAN = [1.2, 0.03, 0.6];
>> ERR  = [0.1, 0.1,  0.1];
>> RUNS = PERMUTE(MEAN,ERR,"RELATIVE")
```

resulting in

```
RUNS =
   1.0800    0.0270    0.5400
   1.0800    0.0270    0.6600
   1.0800    0.0330    0.5400
   1.0800    0.0330    0.6600
   1.3200    0.0270    0.5400
   1.3200    0.0270    0.6600
   1.3200    0.0330    0.5400
   1.3200    0.0330    0.6600
```

A trajectory with the results from the eight simulation using with the above parameter sets is constructed through the commands

```
>> FOR INDEX IN RUNS(I,:) LOOP
-> S(I) = (SPOOL(RUNS(I,1..3))*MOTOR*KP\\-TACHO)*3*U;
-> END LOOP;
```

The envelope of these nine time-responses is then obtained through the command

```
>> PLOT(S,"ENVELOPE");
```

GENERAL MODEL CONNECTIONS IN IMPACT

The feedback structure in Figure 2 is common enough to warrant the introduction of a special feedback operator "\\". However, for general topologies, such as the one shown in Figure 3, our connection operators "*", "+" and "\\" do not suffice. To cover these cases, a more general, hierarchical interconnection facility is available. Assuming that all the systems in Figure 3 had linear representations (and thereby positional rather than named input-output signals), we would define a connectivity system through

```
SYSTEM TOTAL(A,B,C : SYSTEM) IS
  CONNECT A.IN(1)    = C.OUT(1) = OUT,
          A.IN(2)    = IN,
          B.IN       = A.OUT(3),
          C.IN(1..2) = A.OUT(1..2),
          C.IN(3)    = B.OUT;
BEGIN
  NULL;
END TOTAL;
```

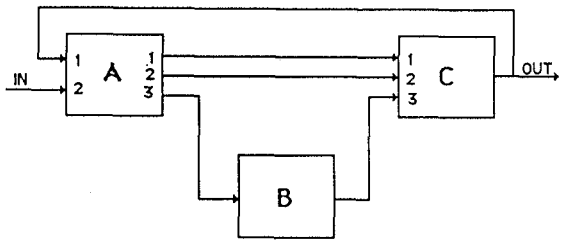


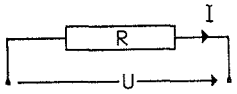
Figure 3. Non-trivial interconnection.

Note that the body of this system is empty as all dynamic equations are contained within the subsystems. "IN" and "OUT" are reserved words denoting if an input or output is connected. The total subsystem is thereafter invoked through the call

```
>> TOT = TOTAL(F,G,H);
```

where "F", "G" and "H" must be linear systems with the correct number of inputs and outputs. The thereby created system TOT is also a linear system of the same type as the systems "F" through "H". All physically realizable interconnections can be processed using an algorithm described in [3].

Nonlinear systems may be connected in the same manner, using either positional or named signal specification. The result is another nonlinear system. However, whereas linear systems in transfer-function or state-space form have well defined inputs and outputs, this must not be the case for systems described in algebraic and/or differential equation form. Even a model of the simplest of systems, a resistor, may be used in two ways, depending on its surrounding connections:



U = R*I or I = U/R

Hence, for a modular design of large systems in IMPACT, the interface concepts of Elmqvist will be used. In his pioneer work, Elmqvist [5] allows the system equations to be entered in any form, for example would Elmqvists program DYMOLA accept the equation set

```
torque  = av`*inertia;
av      = v/r;
inertia = cable*r**4 - roll;
r`      = -k1*av;
```

just as well as our original equations. This allows the user to enter equations as they were first ootained e.g. from physical laws. DYMOLA, as well as a not yet finished version of IMPACT, will sort the equations into correct order ("vertical" sorting) and also sort each equation so that the correct variable is on the left-hand-side ("horizontal" sorting). The sorter will also check the equations for completeness and consistency and detect any algebraic loops requiring special treatment.

To fully utilize the freedom of vertical and horizontal sorting over different submodels, it must be possible to define modules without having to specify if a connection variable is an input or output signal. Elmqvist has shown that there generally exist two kinds of connecting variables; "ACROSS" variable which are set equal at the interfaces (as the voltages at the connection of three resistors) and "THROUGH" variables which are summed to zero at the interface (as the currents at the same connection). Returning to our example, if we wish to include the inertia of the motor in our system, the motor would have to be modelled in greater detail and our spool model could be defined having the

torque as through and angular velocity as across variable at the interface to the motor:

```
SYSTEM Spool(r0,d,w    : SCALAR)
            IN torque : SCALAR
            RETURN v  : SCALAR IS
  CUT axis(torque : THROUGH;
           av     : ACROSS);
      cable(v     : ACROSS);
  --
```

Note that this system has in/return parameter and cut declarations. This allows us to use the same model either as previously with fixed inputs/outputs or in a hierarchical model as follows: assuming the motor has a similar cut axis, then the two systems can be combined in a hierarchical system having the header

```
  SYSTEM Motor_spool(spool, motor : SYSTEM)
                     IN     uin   : SCALAR
                     RETURN vout  : SCALAR IS
    CONNECT spool.axis = motor.axis;
    CONNECT vout = spool.cable;
    CONNECT uin  = motor.uin
  END Motor_spool;
```

This model can then again be used to form the system of Figure 2.

CONCLUSIONS

It is a cruel side of reality that most control algorithms work only on linear systems whereas all physical systems are nonlinear, forcing many control engineers to work in the linear as well as nonlinear domain. However, the same engineers are often extra punished by having to work with different, incompatible software packages, giving them problems of transferring data from one package (domain) to another.

In this paper, we have shown how a nonlinear modeling and simulation environment can be fully integrated into a command-driven control environment. This allows the user to work with a unified user-interface during the whole design cycle. It also allows for a complete mixture between nonlinear (symbolic) and linear (numeric) models, something particularly useful when linear controllers are designed for nonlinear systems. Moreover, the algorithmic interface of IMPACT can be used as a flexible simulation environment for invoking complex simulation operations.

ACNOWLEDGEMENTS

We would like to thank Professor Mansour and Professor Schaufelberger for their long-time support of the IMPACT project.

REFERENCES

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

[2] Cellier, F.E.; "Enhanced Run-Time Experiments in Continuous System Simulation Languages." In: Proceedings of the 1986 SCSC Multiconference, (Cellier, F.E., ed.), San Diego, CA, pp.78-83.

[3] DeCarlo, R.A. and Saeks, R.; "Interconnected Dynamical Systems". Marcel Dekker Inc, New York, 1981.

[4] "ELCS - The Extended List of Control Software"; Number 2; June 1986 (Frederick, D.K., Herget, C.J, Kool, R. and Rimvall, M., Eds.); Contact M. Rimvall (Europa) or D. Frederick (RPI, Troy, NY, USA) for details.

[5] Elmqvist, H.; "A Structured Model Language for Large Continuous Systems". PhD Thesis; Report CODEN:LUTFD2/(TRFT-1015), Dept. of Automatic Control, Lund Institute of Technology, Lund, Sweden, 226 p.

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

[7] Moler, C.; "MATLAB, User's Guide". Department of Computer Science, University of New Mexico, Albuquerque, USA, 1980.

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

[9] Rall, L.B.; "Automatic Differentiation, Techniques and Applications". Lecture Notes in Computer Science, Vol. 120. Springer Verlag, 1981.

[10] 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.

[11] Rimvall, M. and Bomholt, L.; "A Flexible Man-Machine Interface for CACSD Applications". In: Proceedings of the 3rd IFAC symposium on Computer Aided Design in Control and Engineering Systems, Copenhagen, July 31 - August 2 1985, Pergamon Press, in press.

[12] Rimvall, M. and Cellier, F.E.; "The Matrix Environment as Enhancement to Modeling and Simulation". In: Proceedings of the 11th IMACS World Conference on System Simulation and Scientific Computation, Oslo, August 5-9, 1985. North Holland, in press.

[13] Shah, S.C., Floyd, M.A. and Lehman, L.L; "MATRIXx : Control Design and Model Building CAE Capability". In Jamshidi, M. and C. Herget (Eds.), Advances in Computer-Aided Control systems Engineering. North Holland Press, 1985.