Control System Professional Suite

Béla Paláncz, Zoltán Benyó, and Levente Kovács

Control System Professional Suite (CSPS), released in the middle of 2003 by Wolfram Research, Inc., provides an object-oriented environment for solving control and system engineering problems with state-of-the-art algorithms. This product consists of the Control System Professional v.2 (CSP2) and Advanced Numerical Methods (ANM) application packages, which are integrated into Mathematica and utilize Mathematica's numeric and symbolic capabilities.

This review employs a tutorial case study to introduce some of the main features of CSPS and compare them with those of some of the MATLAB toolboxes. We focus on the application of symbolic algebra to control engineering and demonstrate the use of symbolic computation for defining control objects, deriving their linearized versions, and designing control laws in the time and frequency domains.

The features, role, and importance of a symbolic computational environment in control engineering have been widely recognized (see, for example, the special features edition of the *Computing and Control Engineering Journal* on the use of symbolic algebra computing in control engineering [5]).

Our computations were performed with Mathematica 5, CSPS applications CSP2 v. 2.0.1 and ANM v. 1.0.1, and the MATLAB Control Toolbox version 6.5. To compare Mathematica and MATLAB features, the detailed Mathematica and MATLAB codes are shown.

System Model

We consider a simplified model of a liquid storage vessel called the one tank system with a time-dependent inlet flow rate $Qin(t)$. The level $h(t)$ is controlled by the outlet flow $Qout(t)$ regulated by a control valve $c(t)$ as the actuator (see Figure 1). Extension of this problem to multiple cascaded tanks is considered in [6].

We derive the model equations using Mathematica. From the mass balance, the derivative of $h(t)$ is given by

$$\mathrm{In}[1] := \dot{\mathbf{h}}[\mathbf{t}] = \frac{\mathbf{Qin}[\mathbf{t}] - \mathbf{Qout}[\mathbf{t}]}{\mathbf{A}}.$$

The outlet flow rate can be expressed in terms of the valve setting and the water level as

$$\mathrm{In}[2] := \mathbf{Qout}[\mathbf{t}] = \mathbf{c}[\mathbf{t}]\sqrt{\mathbf{h}[\mathbf{t}]}.$$

Combining these equations yields a nonlinear differential equation for the state-variable $h(t)$ given by

$$\mathrm{In}[3] := \dot{\mathbf{h}}[\mathbf{t}]$$

$$\mathrm{Out}[3] = \frac{-c[t]\sqrt{h[t]} + Qin[t]}{A}.$$

In terms of the general nonlinear model

$$\dot{x} = f(x, u),$$
$$y = g(x, u),$$

we have

$$\mathrm{In}[4] := \mathbf{f} = \dot{\mathbf{h}}[\mathbf{t}]$$

$$\mathrm{Out}[4] = \frac{-c[t]\sqrt{h[t]} + Qin[t]}{A},$$

and

$$\mathrm{In}[5] := \mathbf{x} = \{\mathbf{h}[\mathbf{t}]\}; \quad \mathbf{y} = \{\mathbf{h}[\mathbf{t}]\};$$
$$\mathbf{u} = \{\mathbf{Qin}[\mathbf{t}], \quad \mathbf{c}[\mathbf{t}]\}; \mathbf{g} = \mathbf{y};$$

where, in Mathematica, the curly braces { } denote vectors (or lists; matrices in Mathematica are lists of lists).

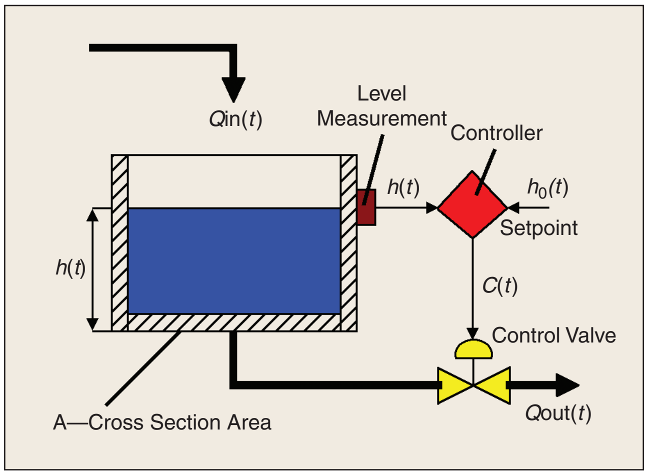

Figure 1. *Block diagram of the One Tank System. The level h(t) of the liquid storage tank is influenced by the inlet flow rate Qin(t) as a disturbance and by the outlet flow rate Qout(t), which is controlled by the control valve setting c(t) as the control input. The time constant of the measuring device is assumed to be small compared to that of the process.*

0272-1708/05/$20.00©2005IEEE

Linearization of the Model

To design a controller for the system, we first linearize the model. We start by loading the CSP2 application

$$\text{In}[6] := \ll \textbf{ControlSystems}'.$$

CSPS Commands

Using the function **Linearize**, we express the model as a linear control object in the vicinity of the steady state $(h0,\ Qin0,\ c0)$ using the commands

$\text{In}[7] := \textbf{ControlObjectSS} =$
$\textbf{Linearize}[\textbf{f},\ \textbf{g},\ \{\{\textbf{h}[\textbf{t}],\ \textbf{h0}\}\},\ \{\{\textbf{Qin}[\textbf{t}],\ \textbf{Qin0}\},\ \{\textbf{c}[\textbf{t}], \textbf{c0}\}\}],$
$\text{Out}[7] =$

$$\text{StateSpace}\left[\left\{\left\{-\frac{c0}{2\ A\sqrt{h0}}\right\}\right\},\ \left\{\left\{\frac{1}{A}, -\frac{\sqrt{h0}}{A}\right\}\right\},\ 1,\ 0,\ 0\right].$$

The linearized state-space system can be represented as

$\text{In}[8] := \textbf{ControlObjectSS}/\textbf{EquationForm}$
$\text{Out}[8]//\text{EquationForm} =$

$$\dot{x} = \left(-\frac{c0}{2\ A\sqrt{h0}}\right)x + \left(\frac{1}{A}\quad -\frac{\sqrt{h0}}{A}\right)u$$
$$y = (1)x + (0\ 0)u.$$

Here, the variables $x(t)$ and $u(t)$ are deviations from the equilibrium.

In the MATLAB environment, the linearization can be carried out numerically in SIMULINK using the function **linmod**. This procedure requires the following steps:

- Define an S-function OneTankSFunC.m to describe the nonlinear model and give the steady state point.
- Use an S-function block in SIMULINK to create a model OneTankSystemSFunc.mdl whose input is an IN block and whose output is an OUT block.
- In MATLAB, apply the function **linmod** to the above model using

$$[A, B, C, D] = linmod(\text{‘}OneTankSystemSFunC\text{’}, x0),$$

where $x0$ is the steady-state value.

The commands needed to perform these steps are given below.

Step 1. Create the S-Function OneTankSFunC.m for the Nonlinear System

```
function[sys, x0, str, ts] = OneTankSFunC(t, x, u, flag)
% SFUNTMPL General M-file S-function template
% The following is an outline of the general structure of an
% S-function:

switch flag,
  case 0,         % Initialization
        [sys, x0, str, ts] = mdlInitializeSizes;
  case 1,         % Derivatives
        sys = mdlDerivatives(t, x, u);
  case 3,         % Update
        sys = mdlOutputs(t, x, u);
  case{2, 4, 9}   % GetTimeOfNextVarHit
  sys = [];       % Unused flags
  otherwise       % Unexpected flags
        error([‘Unhandled flag = ’, num2str(flag)]);
end
% end sfuntmpl

% mdlInitializeSizes-Return the sizes, initial
% conditions, and sample times for the S-function.

function [sys,x0,str,ts]=mdlInitializeSizes

sizes = simsizes;

sizes.NumContStates = 1;
sizes.NumDiscStates = 0;
sizes.NumOutputs = 1;
sizes.NumInputs = 2;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;

sys = simsizes(sizes);

% initialize the initial conditions
x0  = [9];

% str is always an empty matrix
str = [];

% initialize the array of sample times
ts  = [0 0];
% end mdlInitializeSizes

% mdlDerivatives—Return the derivatives for the
% continuous states.
function sys = mdlDerivatives(t, x, u)
  a=10;           % the cross-section of the tank
  sys(1) = (-u(1) * (x(1)^0.5) + u(2))/a;
% end mdlDerivatives

% mdlUpdate—Handle discrete state updates, sample
% time hits, and major time step requirements.
function sys = mdlUpdate(t, x, u)
% end mdlUpdate
% mdlOutputs—Return the block outputs.
function sys = mdlOutputs(t, x, u)
   C = [1];
   sys = C * x;
% end mdlOutputs

% mdlGetTimeOfNextVarHit—Return the time of the
% next hit.
function sys = mdlGetTimeOfNextVarHit(t, x, u)
   sampleTime = 1; % Ex. set the next hit to be
   % one second later.
   sys = t + sampleTime;
% end mdlGetTimeOfNextVarHit

% mdlTerminate—Perform any end of simulation tasks.
function sys = mdlTerminate(t, x, u)
   sys = [];
% end mdlTerminate
```

Step 2. Create the SIMULINK Model for the S-Function

On the SIMULINK side, the S-function block calls the S-function model OneTankSystemSFunc.mdl created in MATLAB (Figure 2). In this model, In1 is an input block and Out1 is an output block. The OneTankSFunC is also a block, which calls the function created in Step 1.

Step 3. Apply the Linmod Function

Finally, in MATLAB, the linearized model is computed by calling the **linmod** function with the name of the SIMULINK model (without extension) and the steady-state value ($x0 = 9$) where the system is linearized as

$$[A, B, C, D] = linmod(\text{'OneTankSystemSFunc'}, 9).$$

Control Object Representation

The ability to create different representations of a control object is an important feature of Mathematica. With this tool, it is possible to change the form of the control object, as we now illustrate.

CSPS Commands

In addition to the equation form demonstrated above, CSPS can display a state-space system in the traditional textbook form. Our model is given by

In[9] := **TraditionalForm[ControlObjectSS]**

Out[9]//TraditionalForm =

$$\left(\begin{array}{c|cc} -\frac{c0}{2A\sqrt{h0}} & \frac{1}{A} & -\frac{\sqrt{h0}}{A} \\ \hline 1 & 0 & 0 \end{array}\right)^{S},$$

where S in the exponent stands for state space. This form can be produced by clicking the ControlFormat button in the ControlFormat palette of CSP2. To represent a state-space object in the frequency domain, we construct its transfer function by means of

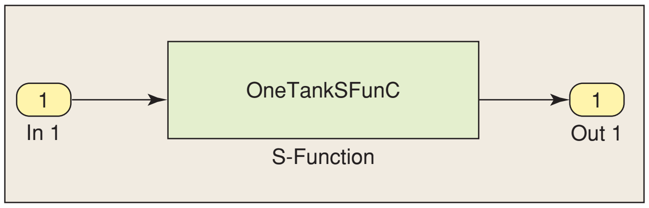

Figure 2. *Flow chart of the OneTankSystemFunc.mdl model as an S-function in SIMULINK.*

In[10] := **ControlObjectTF = TransferFunction[s, ControlObjectSS]//Simplify**

Out[10] = TransferFunction

$$\left[s, \left\{\left\{\frac{2\sqrt{h0}}{c0 + 2\ A\ \sqrt{h0}\ s}, -\frac{2\ h0}{c0 + 2\ A\ \sqrt{h0}\ s}\right\}\right\}\right].$$

The same object can be represented in the traditional form, where τ in the exponent stands for frequency domain, as

In[11] := **ControlObjectTF//TraditionalForm**

Out[11]//TraditionalForm =

$$\left(\frac{2\sqrt{h0}}{c0 + 2\ A\ \sqrt{h0}\ s} \quad -\frac{2\ h0}{c0 + 2\ A\ \sqrt{h0}\ s}\right)^{\tau}.$$

We can obtain a state-space realization of this transfer function by entering

In[12] :=**StateSpace[ControlObjectTF]//EquationForm**

Out[12]//EquationForm =

$$\dot{x} = \begin{pmatrix} -\frac{c0}{2A\sqrt{h0}} & 0 \\ 0 & -\frac{c0}{2A\sqrt{h0}} \end{pmatrix} x + \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} u$$

$$y = \begin{pmatrix} \frac{1}{A} & -\frac{\sqrt{h0}}{A} \end{pmatrix} x.$$

MATLAB Commands

State-space-to-transfer-function and transfer-function-to-state-space conversions in MATLAB can be implemented numerically using the functions **ss2tf(A, B, C, D)** and **tf2ss(num, den)**. However, custom programs are needed in the symbolic case. To be understood by the Symbolic Toolbox embedded in MATLAB, each symbolic variable must be declared separately as a symbolic object (**syms**) in the beginning of the program.

To convert from a state-space realization to a transfer function, one can implement the relation $W(s) = C(sI - A)^{-1}B + D$ using the Symbolic Toolbox of MATLAB. This conversion can be carried out using the commands

```
syms s real      % Symbolic variables
syms c0 real
syms a real
syms h0 real
A = [-c0/(2*a*sqrt(h0))];
B = [1/a  -sqrt(h0)/a];
C = [1];
D = [0 0];
% SS to transfer function symbolically
ControlObjectTF = simplify(C*inv(s*eye(1) - A)*B)
% two transfer functions are obtained
```

To convert from a transfer function to a state-space realization, one can use the function **sym2poly**, with the commands

```
[numyuos, denyuos] = numden(ControlObjectTF(1, 1));
% as illustration, the inverse conversion is made only for
% one transfer function
num = numyuos
denyuo = sym2poly(denyuos)
% Transfer function to SS symbolically
n = 2;
% rank of the system
den = denyuo(2 : n)./denyuo(1);
a = [-den; eye(n - 2, n - 1)]
b = eye(n - 1, 1)
% c = num(:, 2 : n) - num(:, 1) * den;
% general formula for C matrix,
% now C is 1x1 matrix (C = [1]).
```

Control Design in State Space

Let us start by designing a simple proportional controller in state-space using pole placement. Using Mathematica, we perform the design in symbolic form.

CSPS Commands

Let p be the desired location of the pole. Then, the controller gains are given by

In[13] := **k = StateFeedbackGains[ControlObjectSS, {p}, ControlInput → 2]**

$$\text{Out[13]} = \left\{\{0\}, \left\{\frac{c0 + 2A\sqrt{h0}\,p}{2\,h0}\right\}\right\}.$$

By specifying **ControlInput → 2**, only the second component of the control vector $u(t)$, namely $c(t)$, depends on the state vector $x(t)$. The first component of the control vector $Q\text{in}(t)$, which represents the disturbance, does not depend on $x(t)$.

To verify the correctness of the expression for the control gain k, we extract the matrices A and B of the linearized equation from the state-space object using the commands

In[14] :=**a = ControlObjectSS[[1]]**

$$\text{Out[14]} = \left\{\left\{-\frac{c0}{2A\sqrt{h0}}\right\}\right\},$$

In[15] :=**b = ControlObjectSS[[2]]**

$$\text{Out[15]} = \left\{\left\{\frac{1}{A}, -\frac{\sqrt{h0}}{A}\right\}\right\}.$$

Indeed, the eigenvalue of the closed-loop system is given by

In[16] :=**Eigenvalues[a − b.k]//Simplify**

Out[16] $=\{p\}$.

The functions **StateFeedbackGains** and **Eigenvalues**, which are based on Ackermann's formula, can be used for higher-order systems but only with one input variable. In the case of multiple inputs, numerical solution is possible by employing the iterative robust algorithm of Kautsky-Nichols-Van Dooren implemented in CSPS.

MATLAB Commands

In MATLAB, the pole placement problem must be solved numerically because the corresponding functions **place()** (for MIMO systems) and **acker()** (for single-input systems) require the system matrices A and B and the pole specification in numerical form. The necessary commands are

```
c0 = 3;
a = 10;          % the cross section of the tank
h0 = 9;
p = -1;          % the desired pole
A = [-c0/(2*a*sqrt(h0))];
B = [1/a  -sqrt(h0)/a];
C = [1];
D = [0 0];
% Pole placement design
Ka = place(A, B, p).
```

Control Design in the Frequency Domain

To eliminate the deviation between the steady-state level and the desired level, we employ an integrator in the controller. Starting with the transfer function form of the linearized model, we design a PI controller for the storage level control in the frequency domain.

CSPS Commands

The transfer function of the linearized model is given by

$$\begin{aligned}&\text{In[18]} := \textbf{ControlObjectTF} \\ &\quad = \textbf{TransferFunction}[\textbf{s}, \textbf{ControlObjectSS}] \\ &\quad //\textbf{Simplify}, \\ &\text{Out[18]} = \text{TransferFunction} \\ &\quad \left[s, \left\{\left\{\frac{2\sqrt{h0}}{c0 + 2A\sqrt{h0}\, s}, -\frac{2\, h0}{c0 + 2A\sqrt{h0}\, s}\right\}\right\}\right].\end{aligned}$$

The transfer function of the PI controller is

$$\text{In[19]} := \textbf{pi} = \textbf{TransferFunction}\left[\textbf{s},\ \mathbf{k_p} + \frac{\mathbf{k_i}}{\mathbf{s}}\right],$$

where k_p and k_i are proportional and are integrator gains. These two units are connected according to Figure 3. Therefore, the transfer function of the closed-loop system in the traditional form is given by

$$\begin{aligned}&\text{In[20]} := \textbf{CLControlObjectTF} = \textbf{FeedbackConnect} \\ &\quad [\textbf{ControlObjectTF}, \textbf{pi}, \{\textbf{1}\}, \{\textbf{2}\}]//\textbf{Simplify}; \\ &\text{In[21]} := \textbf{CLControlObjectTF}//\textbf{TraditionalForm} \\ &\text{Out[21]}//\text{TraditionalForm} = \\ &\left(\frac{2\sqrt{h0}\, s}{s(c0 + 2A\sqrt{h0}\, s - 2h0\, k_p) - 2h0\, k_i} \quad \frac{2h0\, s}{2h0\, k_i - s(c0 + 2A\sqrt{h0}\, s - 2h0\, k_p)}\right)^{\tau},\end{aligned}$$

where the **FeedbackConnect( )** parameter values 1 and 2 mean that the first process output is fed back to the second process input.

To determine the gains k_p and k_i, we equate the coefficients of like powers of s in the denominator of the first term and the target polynomial with two poles. This polynomial is

$$\begin{aligned}&\text{In[22]} := \textbf{Denominator}[\%[\textbf{s}]][[\textbf{1},\ \textbf{1}]] \\ &\text{Out[22]} = -2\, h0\, k_i + s(c0 + 2A\sqrt{h0}\, s - 2\, h0\, k_p).\end{aligned}$$

In Mathematica, the % character denotes the result of the previous evaluation.

Normalizing this polynomial we get

$$\begin{aligned}&\text{In[23]} := \mathbf{d_1} = \textbf{Sum}[\textbf{Coefficient}[\%, \textbf{s}, \textbf{j}]\ / \\ &\quad \textbf{Coefficient}[\%, \textbf{s}, \textbf{2}]\ \mathbf{s^j}, \{\textbf{j}, \textbf{0}, \textbf{2}\}] \\ &\text{Out[23]} = s^2 - \frac{\sqrt{h0}\, k_i}{A} + \frac{s(c0 - 2\, h0\, k_p)}{2\, A\, \sqrt{h0}}.\end{aligned}$$

This polynomial with poles p_1 and p_2 can be expressed as

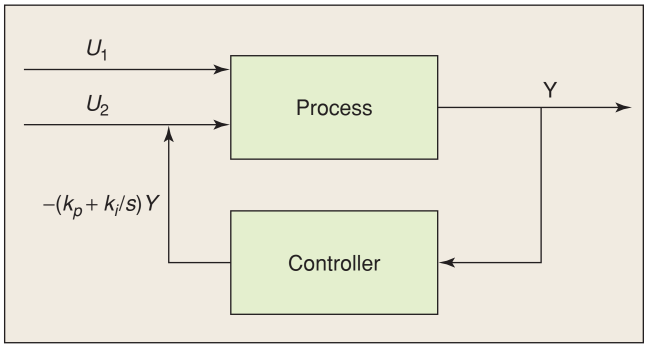

Figure 3. *Closed-loop model with PI controller in the frequency domain. U1(s) represents the disturbance (inlet flow rate), while U2(s) is the control input (control valve setting) and Y(s) is the controlled variable (liquid level).*

$$\text{In[24]} := \mathbf{d_2} = (\mathbf{s} - \mathbf{p_1})\ (\mathbf{s} - \mathbf{p_2}).$$

This solution for the integral and proportional gains is expressed through the desired location of poles of the closed-loop system, which is given in symbolic form as

$$\begin{aligned}&\text{In[25]} := \textbf{Solve}[\textbf{CoefficientList}[\mathbf{d_1} - \mathbf{d_2},\ \textbf{s}] == \textbf{0}, \\ &\quad \{\mathbf{k_i},\ \mathbf{k_p}\}]//\textbf{Simplify} \\ &\text{Out[25]} = \\ &\left\{\left\{k_i \to -\frac{Ap_1p_2}{\sqrt{h0}},\ k_p \to \frac{c0 + 2A\sqrt{h0}\, p_1 + 2A\sqrt{h0}\, p_2}{2\, h0}\right\}\right\}.\end{aligned}$$

In the case $p_2 = 0$, we obtain the same result from a state-space design of the proportional controller (see Out[13]). For higher-order systems, see [6].

MATLAB Commands

In MATLAB, the numerator and denominator can be separated by the function **numden( ),** allowing at most one symbolic variable (such as s), while a function analogous to **Coefficient** is not available. Therefore, solutions for k_i and k_p can be obtained only if the model parameters are given numerically. Consequently, PI control design must be carried out numerically. This procedure requires iteration by computing the phase margin using the Bode plot. In Mathematica, this information is incorporated in the proper selection of the poles p_1 and p_2.

Design of the Optimal Controller

In this section, we design a linear quadratic regulator by employing the CSP2 functions for solving the Riccati equation. In addition, different solution techniques implemented in Advanced Numerical Methods (ANM) are also applied.

As we have seen, the nonlinear model of the process is given by

In[27] := **Clear**[**"Global′ ∗"**]

$$\text{In}[28] := \mathbf{f} = \frac{-\mathbf{c}[\mathbf{t}]\sqrt{\mathbf{h}[\mathbf{t}]} + \mathbf{Qin}[\mathbf{t}]}{\mathbf{A}}.$$

The input and output state variables are

$$\text{In}[29] := \mathbf{x} = \{\mathbf{h}[\mathbf{t}]\};\ \mathbf{y} = \{\mathbf{h}[\mathbf{t}]\};$$
$$\mathbf{u} = \{\mathbf{Qin}[\mathbf{t}], \mathbf{c}[\mathbf{t}]\};\ \mathbf{g} = \mathbf{y}.$$

The linearized form of the model, in the equation form, is

```
In[30] :=ControlObjectSS
        =Linearize[f, g, {{h[t], h0}},
           {{Qin[t], Qin0}, {c[t], c0}}];
In[31] :=ControlObjectSS//EquationForm
Out[31]//EquationForm =
```

$$\dot{x} = \left(-\frac{c0}{2A\sqrt{h0}}\right)x + \left(\frac{1}{A} \quad -\frac{\sqrt{h0}}{A}\right)u$$
$$y = (1)x + (0\ 0)u.$$

We check the controllability of the system using the controllability matrix

```
In[32] := ControllabilityMatrix[ControlObjectSS]
```

$$\text{Out}[32] := \left\{\left\{\frac{1}{A},\ -\frac{\sqrt{h0}}{A}\right\}\right\}.$$

For a controllable system, the rank of this matrix must be equal to that of the system matrix; that is,

```
In[33] :=MatrixRank[%] ==
           MatrixRank[ControlObjectSS[[1]]]
Out[33] := True.
```

Therefore the system is controllable. Alternatively, controllability can be checked by using the built-in function

```
In[34] :=Controllable[ControlObjectSS]
Out[34] := True.
```

Since linear-quadratic-regulator design cannot be performed symbolically, we introduce numerical values

$$\text{In}[35] : = \mathbf{numericalValues} = \{\mathbf{A} \to \mathbf{10},\ \mathbf{c0} \to \mathbf{3}, \mathbf{Qin0} \to \mathbf{9}, \mathbf{h0} \to \mathbf{9}\}.$$

We choose matrices q and r such that the resulting gain matrix is close to the pole assignment design with $p = -1$. Suitable values are given by

```
In[36] :=q = {{1.}};
In[37] :=r = {{1000, 0}, {0, 0.0745}}.
```

CSPS Commands

The regulator gains are given by

```
In[38] := LQRegulatorGains
           [ControlObjectSS/.numericalValues, q, r]
Out[38] := {{0.0000869372}, {−3.50082}},
```

where "/." denotes the Mathematica function **ReplaceAll**. Because only the second input is controlled (the first one is the disturbance), the first gain can be eliminated using

```
In[39] :=Chop[%, 0.001]
Out[39] := {{0}, {−3.50082}}.
```

The closed-loop system is thus

```
In[40] := StateFeedbackConnect[ControlObjectSS/
           .numericalValues, %];
```

and its equation form is

```
In[41] := % //EquationForm
Out[41]//EquationForm =
```

$$\dot{x} = (-1.10025)x + (0.1\ \ -0.3)u.$$
$$y = (1.)x + (0.\ 0.)u.$$

CSPS provides several methods for finding the optimal gains. Specifically, eigensystem and Schur methods are included in the ANM package, which can be loaded by entering

```
In[42] := << NumericalControlMethods'.
```

For example, the **GeneralizedSchurDecomposition** command is invoked by

```
In[43] := LQRegulatorGains[ControlObjectSS/
           .numericalValues, q, r, SolveMethod
           → GeneralizedSchurDecomposition]
Out[43] = {{0.0000869372}, {−3.50082}},
```

or the **MatrixSign** method

```
In[44] := LQRegulatorGains[ControlObjectSS/
           .numericalValues, q, r, SolveMethod
           → MatrixSign]
Out[44] = {{0.0000869372}, {−3.50082}}.
```

The function **LQRegulatorGains** calls the Riccati equation solver and passes the user-supplied option, which in this case is **SolveMethod**. At the same time, the user also has access to the underlying solvers and can find the optimal gains in a step-by-step fashion. To demonstrate this feature, we extract the system matrices from the state-space representation of the control object by means of

In[45] :=**ControlObjectSS**//**TraditionalForm**

Out[45]//TraditionalForm =

$$\left(\begin{array}{c|cc} -\dfrac{c0}{2A\sqrt{h0}} & \dfrac{1}{A} & -\dfrac{\sqrt{h0}}{A} \\ \hline 1 & 0 & 0 \end{array}\right)^{S},$$

In[46] :=**a** = % [[**1**]]

$$\text{Out[46]} = \left\{\left\{-\frac{c0}{2A\sqrt{h0}}\right\}\right\},$$

In[47] :=**b** = %% [[**2**]]

$$\text{Out[47]} := \left\{\left\{\frac{1}{A},\ -\frac{\sqrt{h0}}{A}\right\}\right\}.$$

The numerical values of these matrices are given by

In[48] :=**aa** = **a**/.**numericalValues**

Out[48] = {{−0.05}},

In[49] :=**bb** = **b**/.**numericalValues**

Out[49] = {{0.1, −0.3}}.

The Riccati equation can be solved by means of a number of methods. For example,

In[50] := **RiccatiSolve**[**aa**, **bb**, **q**, **r**,
SolveMethod → **SchurDecomposition**]

Out[50] = {{0.869372}},

In[51] := **RiccatiSolve**[**aa**, **bb**, **q**, **r**, **SolveMethod**
→ **GeneralizedSchurDecomposition**]

Out[51] = {{0.869372}},

In[52] := **RiccatiSolve**[**aa**, **bb**, **q**, **r**,
SolveMethod → **MatrixSign**]

Out[52] = {{0.869372}},

In[53] := **w** = **RiccatiSolve**[**aa**, **bb**, **q**, **r**,
SolveMethod → **Newton**,
InitialGuess → {{**1**.}}]

Out[53] = {{0.869372}},

In[54] := **RiccatiSolve**[**aa**, **bb**, **q**, **r**, **SolveMethod**
→ **GeneralizedEigendecomposition**]

Out[54] = {{0.869372}},

which gives the same result.

Even though the built-in CSPS functions are designed to solve the Riccati equations numerically, for this low-order system one can use the standard Mathematica function **Solve** to produce a symbolic solution using

In[55] := **Clear**[**w**].

The Riccati equation is given by

In[56] := **req** = **Transpose**[**a**].{{**w**}} + {{**w**}}.**a**−
{{**w**}}.**b**.**Inverse**[**r**].**Transpose**[**b**].{{**w**}}
+ **q** //**Simplify**

Out[56] =

$$\left\{\left\{1. - \frac{c0\ w}{A\sqrt{h0}} + \frac{(-0.001 + 0.\ \sqrt{h0} - 13.4228\ h0)w^2}{A^2}\right\}\right\}.$$

The two solutions are given by

In[57] := **Solve**[**reg** == **0**, **w**]

Out[57] =

$$\left\{\left\{w \to \frac{0.5A^2\left(-\frac{1.\ c0}{A\sqrt{h0}} - \frac{1.\ \sqrt{1.\ c0^2+0.004\ h0+53.6913\ h0^2}}{A\sqrt{h0}}\right)}{0.001 + 13.4228\ h0}\right\},\right.$$
$$\left.\left\{w \to \frac{0.5A^2\left(-\frac{1.\ c0}{A\sqrt{h0}} + \frac{1.\ \sqrt{1.\ c0^2+0.004\ h0+53.6913\ h0^2}}{A\sqrt{h0}}\right)}{0.001 + 13.4228\ h0}\right\}\right\}.$$

The stabilizing solution is given by

In[58] := **w1** = {{**w**/. % [[**1**]]}}

Out[58] =

$$\left\{\left\{\frac{0.5A^2\left(-\frac{1.\ c0}{A\sqrt{h0}} - \frac{1.\ \sqrt{1.\ c0^2+0.004\ h0+53.6913\ h0^2}}{A\sqrt{h0}}\right)}{0.001 + 13.4228\ h0}\right\}\right\},$$

and the antistabilizing solution is

In[59] := **w2** = {{**w**/. %% [[**2**]]}}

Out[59] =

$$\left\{\left\{\frac{0.5A^2\left(-\frac{1.\ c0}{A\sqrt{h0}} + \frac{1.\ \sqrt{1.\ c0^2+0.004\ h0+53.6913\ h0^2}}{A\sqrt{h0}}\right)}{0.001 + 13.4228\ h0}\right\}\right\}.$$

To choose between these solutions, we use the numerical values

```
  In[60] :=w1n = w1/. numericalValues
Out[60] = {{−0.952149}},
  In[61] :=w2n = w2/. numericalValues
Out[61] = {{0.869372}}.
```

The second solution, which gives the positive value, is stabilizing. This solution is returned by CSPS.

As soon as the symbolic solution of the Riccati equation is known, the gain matrix can be computed in symbolic form using

$$K = R^{-1}\, B^T\, W,$$

which is obtained with

```
In[62] := Inverse[r].Transpose[bb].w2
Out[62] =
```

$$\left\{\left\{\frac{0.00005 A^2\left(-\frac{1.\,c0}{A\sqrt{h0}}-\frac{1.\,\sqrt{1.\,c0^2+0.004\,h0+53.6913\,h0^2}}{A\sqrt{h0}}\right)}{0.001+13.4228\,h0}\right\},\right.$$
$$\left.\left\{\frac{0.00005 A^2\left(-\frac{1.\,c0}{A\sqrt{h0}}+\frac{1.\,\sqrt{1.\,c0^2+0.004\,h0+53.6913\,h0^2}}{A\sqrt{h0}}\right)}{0.001+13.4228\,h0}\right\}\right\},$$

or numerically,

```
  In[63] :=Inverse[r].Transpose[bb].w2n
Out[63] = {{0.0000869372}, {−3.50082}}.
```

This gain matrix is also produced by the **LQRegulatorGains** function. Of course, it is not generally possible to solve the Riccati equation for higher-order systems in symbolic form.

MATLAB Commands

Using the Control Toolbox, the Mu–Toolbox, and the Symbolic Toolbox, the LQ control also can be realized in MATLAB numerically and symbolically with at most one symbolic variable.

Linear-quadratic-regulator design for continuous-time systems can be carried out using the Control Toolbox function **lqr**. This function returns a triple consisting of the optimal gain matrix K, the solution of the Riccati equation P, and the vector E containing the eigenvalues of the closed-loop system. The **lqr** function requires the system be in numeric or symbolic form (with at most one symbolic variable) as well as the matrices q and r. Using the function **lqry** from the Mu-Toolbox, the linear-quadratic-regulator design can be carried out with output weighting.

The main advantage of these MATLAB functions is that the triple (K, P, E) is obtained using a single function. The MATLAB commands are given by

```
syms s real
% Declaring symbolic variable s (if we want to work
% symbolically)
c0 = 3;
a = 10;          % the cross section of the tank
h0 = 9;
A = [−c0/(2 ∗ a ∗ sqrt(h0))];
B = [1/a  − sqrt(h0)/a];
C = [1];
D = [0 0];
% LQR design
q = [1];
r = [1000 0;  0 0.0745];
ControlObjectTF = ss(A, B, C, D);
[KLQ, P, E] = lqr(ControlObjectTF, q, r)
[KLQy, Py, Ey] = lqry(ControlObjectTF, q, r)
% for weighted output.
```

Additional Features of CSPS

There are many other features of CSPS that are not considered in this review due to limited space. For example, CSPS includes interconnections of arbitrary complex systems, observability matrices and Gramians, observable subspaces, Kalman and Jordan canonical forms, minimal and balanced realizations, pole-zero cancellation, similarity transformations, and Kalman filtering.

Additional features in ANM include system identification in the time and frequency domains, Schur and Hessenberg-Schur methods for Lyapunov and Sylvester equations, controller-Hessenberg and observer-Hessenberg forms, constrained and Lyapunov feedback stabilization, full- and reduced-order state estimation, and model reduction using the Schur and square-root methods. These features are discussed and demonstrated in [1] and [2].

The MATLAB toolboxes include some functions that work with numerical and a single symbolic variable using the Symbolic Toolbox, such as **lyap, h2norm, hinfnorm, hinfsyn, hinffi, tustin, ltru, ltry,** and **frsp**. The Mu-Toolbox offers control design capabilities based on H_2/H_∞ spaces and works mostly numerically, as do the System Identification Toolbox, Predictive Toolbox, and Nonlinear Control Design Block Set.

In addition, MATLAB can perform interactive simulation using SIMULINK, where systems can be specified in block diagram form. SIMULINK also includes applications that demonstrate the capabilities of different toolboxes, such as the simulation of a one-tank system.

Conclusions

CSPS provides a unique and effective symbolic computation environment for solving control engineering problems, particularly in the case of linear systems. On the contrary, in MATLAB, symbolic algebra can be used only in a limited way, mostly for one variable.

MATLAB approaches control theory problems in a technical way, while Mathematica offers a didactic approach. Although both systems are object-oriented, the basic element of MATLAB is a matrix, while Mathematica can handle arbitrary objects using functional programming.

Mathematica works step by step, like a mathematical reasoning machine. This style makes Mathematica programs more readable and solutions easier to obtain. Solutions of control problems are also easier to understand. In addition, the structure of CSPS allows users and developers to readily customize and extend the built-in algorithms.

In our experience, Mathematica's CSPS has advantages for education and research, particularly for developing new control formulas [4]. It is especially useful in real-time applications, where fast evaluation is important (see, for example, [3]), whereas MATLAB has advantages for numerical engineering applications.

Acknowledgments

The authors would like to thank the Hungarian National Research Fund Grant T042990 and Hungarian Ministry of Education Grant FKFP 200/2001 for supporting this work.

References

[1] I. Bakshee, *Mathematica-Advanced Numerical Methods,* 1st ed. Champaign, IL: Wolfram Research, Inc., 2003.

[2] I. Bakshee, *Mathematica-Control System Professional*, 2nd ed. Champaign, IL: Wolfram Research, Inc., 2003.

[3] Z. Benyó, B. Paláncz, L. Kovács, and L. Szilágyi, "A fully symbolic design and modelling of nonlinear glucose control with CSPS of Mathematica," in *Proc. World Congress on Medical Physics and Biomedical Engineering*, Sydney, Australia, 2003 [Online]. Available: http://library.wolfram.com/infocenter/MathSource/5043/

[4] L. Kovács and B. Paláncz, "Linear and non-linear approach of the glucose-insulin control using Mathematica," *Periodica Politechnica, Trans. Autom. Control Compute. Sci.,* vol. 49, no. 63, pp. 65–70, 2004 [Online]. Available: http://bio.iit.bme.hu/obmk/

[5] N. Munro, "Symbolic algebra computing in control engineering," *Comput. Contr. Eng. J.*, vol. 8. no. 2, pp. 50–53, 1997.

[6] B. Paláncz, "Control of cascade system of N noninteracting tanks using CSPS" [Online]. Available: http://bio.iit.bme.hu/obmk/

Béla Paláncz (palancz@epito.bme.hu) is an associate professor of computer science at the Technical University of Budapest, Hungary. He received his D.Sc. from the Hungarian Academy of Sciences in 1993 and has a background in education and research of mathematical modeling and numeric-symbolic computation, including experience at RWTH (Aachen, Germany), Imperial College (London), and Wolfram Research (USA). He can be contacted at the Department of Photogrammetry and Geoinformatics, Budapest University of Technology and Economics, H-1111 Budapest, Muegyetem rkp.3., Hungary.

Zoltán Benyó received a degree in electrical engineering from the Technical University of Budapest, Hungary, in 1961. Since that time, he has taught at the Technical University of Budapest, first as an assistant professor, then as senior lecturer. Since 1994, he has been a professor with the Department of Control Engineering and Information Technology. He teaches courses in process control theory and biomedical engineering. He has published 180 papers and is the author of 20 textbooks. He is a member of the IFAC National Committee and the IEEE EMBS International Program Committee.

Levente Kovács received the M.Sc. degree in electrical engineering from the University "Politechnica" of Timisoara, Romania, in 2000. He is currently pursuing a Ph.D. degree in biomedical engineering at the Department of Control Engineering and Information Technology, Budapest University of Technology and Economics, Hungary. His research interests include control theory and mathematical modeling of physiological systems.

ELEVATED ADVICE

Many people influence our careers, but perhaps none more than our advisors in graduate school. Such is the case with me, and as I reflect on influential people and the advice I've benefited from over the years, I recall my advisor Mike Sain relating encounters he had had with various individuals in our field. I was lucky to experience such an encounter one day in a chance meeting with a giant in our field. When I was a young professor attending a control conference, I was surprised to find myself sharing an elevator ride with Nick Nichols, the renowned inventor of the Nichols chart. Although nervous in the presence of greatness, I couldn't let this once-in-a-lifetime opportunity pass by without seeking some gem of wisdom. So I struck up a brief conversation with Dr. Nichols and then mustered the courage to ask him the following question: "If you had one piece of advice for me to teach my control engineering students, what would it be?" For what seemed like endless seconds we rode upward without any reply coming from the legendary master. Just as we reached the end of our ride he finally spoke, and this is what he said to me: "Tell your students this: When you build your controllers, make sure you do it so that when you turn the knob clockwise, the gain goes up."

—CSS Past President Stephen Yurkovich

© EYEWIRE