State Space Computations Using MapleV

Ayowale B. Ogunye and Alexander Penlidis

The development of algorithms needed for the analytic solution of state space design equations using a computer algebra system (MapleV) is presented in this article. These calculations involve matrix manipulations, eigenvalue-eigenvector determinations, Laplace and Inverse Laplace transformations, Z and Inverse Z transformations, which are complex, tedious, and error-prone, even for simple examples. The process of programming with MapleV encourages appreciation of the important aspects of mathematical investigation, because there is one-to-one correspondence between the symbolic code and the mathematical algorithms being programmed. The use of MapleV has provided symbolic and numerical results, quickly and efficiently, with a tremendous gain in time and with minimal effort.

Introduction

MapleV [1] is a computer algebra system which can manipulate and solve complex mathematical expressions. It can perform symbolic and numeric computation, mathematical programming, and visualization. MapleV can be used by instructors to illustrate important concepts in process control courses. Tedium, repetitive, and complex calculations involved in process control exercises can be done much more easily using MapleV. Students have the opportunity to solve complex and non-trivial problems, exploring the different possible scenarios and asking or being asked "what-if" type questions.

We would like to stress the availability of software packages like Matlab [2] that produce solutions to numerical linear algebra problems; however, a computer algebra system such as MapleV is one of the few means by which a closed form symbolic result, similar to a hand calculation, may be obtained. Examples of advantages which a computer algebra system like MapleV offer include the determination of eigenvalues/eigenvectors, determinants, inverses, transposes, adjoints, etc., of a matrix containing symbolic elements, whereas an efficient linear algebra computation package like Matlab can only work with numeric entities. The advantages to having a closed-form symbolic solution include converting the resulting expression to a language like Fortran/Matlab for further numerical studies, an examination of limiting conditions in the resulting expression, etc.

The state space formulation has been a convenient basis for the development of advanced multivariable controller design methodologies [3,4], e.g. pole placement and assignment using state feedback, partial pole placement using static output feedback, optimal deterministic and stochastic control, state estimation, prediction and filtering, observer design, etc. This is because the underlying time-domain models (the most natural description of most problems of interest) can address a more general class of problem definition [5]. The state variable model of a system includes a description of the internal status of that system, in addition to the input-output behavior. The state space approach provides a convenient, compact notation and allows the application of powerful vector-matrix theory. The uniform notation for all systems regardless of order makes possible a uniform set of solution techniques and computer algorithms. The state space representation is in an ideal format for computer solution [3].

It is the purpose of this article to demonstrate how symbolic computation can be used to develop algorithms needed for the solution of state space design equations. In solving state space-related equations, the resulting matrix manipulations, eigenvalue-eigenvector determination, Laplace and Inverse Laplace transformations, Z and Inverse Z transformations, that have to be carried out are complex, tedious, and error-prone, even for simple cases. The widely available (general purpose) computer algebra systems (Maple [1], Mathematica [6], MACSYMA [7], Reduce [8], MuMath [9], and Scratchpad [10]) can perform these calculations. In this article, we use MapleV (Release 3); however, any of the listed computer algebra systems will suffice. MapleV contains a well-developed linear algebra package that contains the basic tools for performing the tedious state space analysis calculations, in addition to its symbolic, numerical, and graphical capabilities.

Recently, there has been an interest in the application of computer algebra to control design and analysis. Akhrif and Blankenship [11], Svensson [12], van Essen [13], and de Jager [14] describe collection of tools and packages to enable the analytic design and analysis of nonlinear control systems. Pas-
travanu et al. [15] describe the stages of a methodology employed to acquaint students in automatic control with the opportunities presented by symbolic calculus in formulating computerized versions for many notable theoretical results, which become powerful design tools. Forsman [16] describes the features and implementation of the Maple software package POLYCON which is intended to assist the control theorist in the analysis of nonlinear dynamical systems, in continuous and discrete time. POLYCON handles systems where all nonlinearities are polynomial or rational functions. Tsang [17] explores the use of computer algebra to synthesize neural network applications. The author can generate highly reliable and efficient codes (probably the most efficient possible) from a high-level neural network algebraic specification. Ogunye and Penlidis [18,19] discuss the use of MapleV to develop algorithms needed for the computation of observability and controllability graminians, computation of minimal balanced realizations, and the solution of matrix Lyapunov equations. Ogunye [20] demonstrates how symbolic computation is used to perform a variety of analytic computations that are part of introductory control courses.

This article is organized as follows. First, we present the solution of the linear time-invariant continuous-time state equations. The state transition matrix needed for this solution is computed from its fundamental definition and from diagonalizing the appropriate system matrix. Next, we discretize the linear continuous-time invariant state equation. Then, we present the solution of the linear time-invariant discrete-time state equation by the Z transform method. Finally, we obtain the controllable, observable and Jordan canonical state-space realizations of discrete-time systems represented by pulse transfer functions and continuous-time invariant state equation. Then, we present the solution of the linear time-invariant continuous-time state equation

\[ \frac{dX}{dt} = AX + BU \]  

(1)

\[ Y = CX + DU \]  

(2)

where \( A, B, C, D \) are matrices of dimensions \( n \times n, n \times r, m \times n, \) and \( m \times r \) respectively, and \( X \) and \( U \) are the state and input vectors of dimensions \( n \times 1 \) and \( r \times 1 \) respectively. The solution of Equation (1) is as follows:

\[ X(t) = e^{At}X(0) + \int_0^t e^{A(t-\tau)}BU(\tau)d\tau \]  

(3)

where \( \Phi(t) = e^{At} \) is the state transition matrix [21]. The state transition matrix can be fundamentally defined as:

\[ \Phi(t) = e^{At} = L^{-1} [sI - A]^{-1} \]

where \( L \) is the laplace transform operator. \( \Phi(t)S \) may also be obtained by diagonalizing matrix \( A \) if it has distinct eigenvalues, or by converting matrix \( A \) into its Jordan canonical form if the eigenvalues are repeated. If the eigenvalues of \( A \) are distinct, then:

\[ \Phi(t) = e^{At} = \begin{bmatrix} e^{\lambda_1 t} & 0 & \ldots & 0 \\ 0 & e^{\lambda_2 t} & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & e^{\lambda_n t} \end{bmatrix} P^{-1} \]  

(6)

where \( P \) is the diagonalizing matrix and \( \lambda_1, \ldots, \lambda_n \) are the distinct eigenvalues of \( A \). If the eigenvalues of \( A \) are repeated, then:

\[ \Phi(t) = e^{At} = J \]  

(7)

where \( J \) is the Jordan canonical form and \( S \) is the transformation matrix.

The computation of \( \Phi(t) \) is carried out in two ways. In the first method, we perform the operation in Equation (5). This is done succinctly using MapleV via procedure \texttt{expmatrix} (see Table 1).

The second method used to compute the state transition matrix is to utilize the matrix exponential function in Maple which performs the operations in Equations (6) and (7). This is done as follows:

\[
\text{> with(linalg, exponential):}
\text{> expmatrix := exponential(matrix, t);} \\
\text{where with(linalg, exponential) reads in the matrix exponential function to compute } \Phi(t) \text{ and exponential(matrix,t)} \text{ is the actual computation step.}
\]

It should be noted that the exponential function will only return a “symbolic” answer if the eigenvalues of the matrix can be found. If the characteristic polynomial of the matrix has a factor of degree greater than four, it may not be possible to express all the eigenvalues in terms of exact “radicals” (a radical is an expression involving a fractional power). In that situation, a floating-point approximation may be obtained by having one of the elements of the matrix being a floating-point number.

**Example 1.** Consider the following example [21, page 525], with the following state space representation:

\[ A = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix} \]  

\[ B = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \]  

\[ X_0 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \]  

\[ U = 1 \]

\[
\text{expmatrix} := \text{proc}(A, \text{transmatx}) \\
\text{local n1, mA, mA, mT, s, t; }
\text{readlib(laplace); }
\text{n1 := coldim(A); }
\text{mA := array(identity, 1..n1, 1..n1); }
\text{mA := map(simplify, inverse(s*mA-A)); }
\text{mT := map(invlaplace, mA, s, t);} \\
\text{transmatx := map(simplify, mT);} \\
\text{end;} \\
\]

| Table 1. Procedure expmatrix |
We now use the two procedures to obtain the state transition matrix. Using procedure `expmatrix` (Table 1) we obtain

\[
\Phi_b = \begin{bmatrix}
-e^{-2t} + 2e^{-t} & e^{-3t} - e^{-2t} \\
2e^{-2t} - 2e^{-t} & 2e^{-3t} - e^{-2t}
\end{bmatrix}
\]

Using the second method (the reader should note that the arrow in the following MapleV commands is essential. The arrow notation enables the computed state transition matrix to be made a function of (t):

\[
\text{with(linalg, exponential)};
\]

\[
\text{phia} := (t) \rightarrow \text{exponential}(A, t);
\]

\[
\text{lindif} := \text{proc}(A, B, U, x_{to}, t_{to}, solution) \text{local } mA, mA0, BC, mE; \text{with(linalg, exponential)}; mA := \text{exponential}(A, t); \text{mA0} := \text{subs}(t=t-tt, eval(mA)); \text{mA} := \text{evalm}(mA0 \times B \times U); \text{mA} := \text{mapl}(\text{int}, mA, tt=tTo..t); \text{mE} := \text{evalm(subs}(t=t-tt, \text{eval}(mA0) \times x_{to}); \text{solution} := (\text{evalm}(mA + mE)); \text{end};
\]

Table 2. Procedure `lindif`

```
lindif := proc(A, B, U, xto, tto, solution) local mA, mA0, mB, mC, mD, mE; with(linalg, exponential); mA := exponential(A, t); mA0 := subs(t = t - tt, eval(mA)); mC := evalm(mB * E * U); mD := mapl(int, mC, tt = tTo..t); mE := evalm(subs(t = t - tt, eval(mA0) * xto)); solution := (evalm(mA + mE)); end;
```

![Fig. 1. System response for Example 2.](image)

The solution, matrix \(X_t\), obtained using `lindif`, is identical to the one obtained in [21]. A plot of the time response of the system (Fig. 1) is generated as follows:

```
p1 := plot(Xt[1, 1], t = 0 .. 10, axes = boxed, style = line, thickness = 0);
> p2 := plot(Xt[2, 1], t = 0 .. 10, axes = boxed, style = point, thickness = 3);
> with(plots);
> t1 := textplot([4, 7, 'Xt[1, 1]', align = {ABOVE, RIGHT}]);
> t2 := textplot([4, 1, 'Xt[2, 1]', align = {ABOVE, RIGHT}]);
> display([p1, t1, p2, t2]);
```

Discretization of Continuous-Time State Space Equations

Discretizing the continuous-time state space Equation (1), by introducing fictitious sample and hold devices (zero-order hold sampling) [21], we obtain

\[
X((k + 1)T) = e^{\Lambda T}X(kT) + \int_0^T e^{\Lambda \tau}BU(kT)d\tau
\]

where \(\Lambda = T - t, t = kT,\) and \(T\) is the sampling time. If we define

\[
G(T) = e^{\Lambda T}
\]

(10a)

Solution of the Linear Time-Invariant State Equation

We are now in a position to obtain the solution of Equation (1) which is given by

\[
X(t) = e^{A(t-t_0)}X(t_0) + \int_{t_0}^{t} e^{A(t-\tau)}BU(\tau)d\tau
\]
Example 4. To demonstrate the use of procedure `dlindif` to compute the state transition and pulse transfer function matrices, state matrix $X(k)$ and output matrix $Y(k)$, consider example 5-3 from [21] with system matrices given as:

$$
G = \begin{bmatrix} 0 & 1 \\ -16 & -1 \end{bmatrix}, \quad H = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad C = \begin{bmatrix} 1 \\ 0 \end{bmatrix} 
$$

$$
DD = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad XX0 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, \quad Uz = z/(z-1)
$$

Using procedure `dlindif`:

```latex
\begin{align*}
\text{dlindif}(G, H, C, DD, XX0, Uz, 'Gk', 'Xk', 'Yk', 'Fk'):
> 'G[1]' &= \text{eval}(Gk); \\
> 'F[1]' &= \text{eval}(Fk);
\end{align*}
```

The routine computing $G(T)$ and $H(T)$ is given in procedure `discon` (See Table 3).

Example 3. Consider the following example from [4, page 46], with the system matrices given as:

$$
A = \begin{bmatrix} -1 & 0 \\ 1 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 1 \\ 0 \end{bmatrix}
$$

We now use procedure `discon` to obtain their discrete time equivalent representations as:

```latex
\begin{align*}
\text{discon}(A, B, h, 'G', 'H'):
> 'H' &= \text{eval}(H), \quad 'G' = \text{eval}(G); \\
H &= \begin{bmatrix} 1 - e^{-h} \\ h + e^{-h} - 1 \end{bmatrix}, \quad G = \begin{bmatrix} e^{-h} & 0 \\ 1 - e^{-h} & 1 \end{bmatrix}
\end{align*}
```

The sampling time is $h$ and the system matrices $G$ and $H$ are identical to those obtained in [4].

Solution of the Discrete-Time State Space Equation

Consider the discrete-time state space system described by Equation (11). The solution to this equation is given in [21] as:

$$
X(k) = Z^{-1} [(cI - G)^{-1} zX(0) + Z^{-1} [(cI - G)^{-1} HU(z)] 
$$

The state transition matrix is

$$
G^k = Z^{-1} [(cI - G)^{-1} z] 
$$

Equation (13) can also be expressed as:

$$
X(k) = Z^{-1} [(cI - G)^{-1} zX(0) + HU(z)] 
$$

The pulse transfer function matrix (matrix relating the output to the input of the process) is

$$
F(k) = Z^{-1} [C(cI - G)^{-1} H + D] 
$$

The output may be expressed in terms of the pulse transfer function matrix as:

$$
Y(k) = \sum_{j=0}^{k} F(k-j)U(j) 
$$

All the foregoing computations are carried out in procedure `dlindif` (see Table 4).

Table 3. Procedure `discon`

<table>
<thead>
<tr>
<th>Procedure <code>discon</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>discon((A, B, h, 'G', 'H'))</code></td>
</tr>
<tr>
<td><code>H = eval(H), G = eval(G)</code></td>
</tr>
<tr>
<td><code>H = [1 - e^{-h}, h + e^{-h} - 1], G = [e^{-h}, 0; 1 - e^{-h}, 1]</code></td>
</tr>
</tbody>
</table>

Table 4. Procedure `dlindif`

<table>
<thead>
<tr>
<th>Procedure <code>dlindif</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>dlindif(GH, HH, CC, DD, X0U0, UZ, Gk, Xk, Yk, Fk)</code></td>
</tr>
<tr>
<td><code>G = map(convert, GH, rational), H = map(convert, HH, rational), C = map(convert, CC, rational), D = map(convert, DD, rational), X0 = map(convert, X0, rational), nl = coldim(G)</code></td>
</tr>
<tr>
<td><code>ii = array(identity, 1..nl, 1..nl)</code></td>
</tr>
<tr>
<td><code>Gkz = evalmap((z*ii-G)^invztrans)</code></td>
</tr>
<tr>
<td><code>Gkz = map(simplify, Gkz)</code></td>
</tr>
<tr>
<td><code>Gk = map(invert, Gkz)</code></td>
</tr>
<tr>
<td><code>Gk = map(convert, eval(map(Z*Gkz, z, k)), z, k)</code></td>
</tr>
<tr>
<td><code>Xk0 = evalmap(Gkz*Z*X0H+UZ)</code></td>
</tr>
<tr>
<td><code>Xkz = map(simplify, Xkz)</code></td>
</tr>
<tr>
<td><code>Xk = map(invert, Xkz)</code></td>
</tr>
<tr>
<td><code>Yk = eval((C*Xk)+(D*UZ))</code></td>
</tr>
<tr>
<td><code>Fkz = map(simplify, eval(map((C*Xk)+(D*UZ)), (C*Xk)+(D*UZ)))</code></td>
</tr>
<tr>
<td><code>Fk = map(invert, Fkz)</code></td>
</tr>
</tbody>
</table>

February 1996
\[
F[k] = \begin{bmatrix}
\frac{25}{4} & \Delta(k) + \frac{5}{12} & -\frac{4}{5} & -\frac{20}{3} & -\frac{1}{5} \\
-\frac{5}{3} & \frac{1}{5} & \frac{1}{3} & \frac{3}{12} & \frac{3}{5} \\
\end{bmatrix}
\]

\[
Y[k] = \begin{bmatrix}
\frac{25}{4} & 62(-1)^k & 61 & -4 & -4 \\
\frac{27}{5} & 45 & 5 & -2 & -1 \\
\frac{31}{5} & 90 & 5 & 24 & -4 \\
\frac{54}{5} & 90 & 5 & 24 & -4 \\
\end{bmatrix}
\]

\[
X[k] = \begin{bmatrix}
\frac{25}{4} & 31(-1)^k & 61 & -4 & -4 \\
\frac{27}{5} & 45 & 5 & -2 & -1 \\
\frac{31}{5} & 90 & 5 & 24 & -4 \\
\frac{54}{5} & 90 & 5 & 24 & -4 \\
\end{bmatrix}
\]

\[\Delta(k) = \begin{cases} 
1 & \text{if } k = 0 \\
0 & \text{if } k \neq 0
\end{cases}
\]

The pulse transfer function matrix \(F[k]\), the state transition matrix \(G[k]\), the state matrix \(X[k]\) and output matrix \(Y[k]\) obtained from procedure \(\text{dlindif}\) are identical to those obtained in [21].

**State Space Representations of Discrete-Time Systems**

Consider the discrete-time system described by

\[
y(k + 1) = a_1 y(k) + a_2 y(k - 1) + \cdots + a_n y(k - n) + b_0 u(k) + b_1 u(k - 1) + \cdots + b_{n-1} u(k - n) + b_n u(k)
\]

(18)

where \(u(k)\) is the input and \(y(k)\) is the output of the system at the \(k\)th sampling instant. Note that some of the coefficients \(a_i\) and \(b_j\) may be zero. Equation (18) can be rewritten in the form of the pulse transfer function as

\[
G(z) = \frac{Y(z)}{U(z)} = \frac{b_0 z^n + b_1 z^{n-1} + \cdots + b_n}{z^n + a_1 z^{n-1} + \cdots + a_n}
\]

(19)

where \(Y(z)\) is the \(\mathcal{Z}\) transform of \(y(k)\) and \(U(z)\) is the \(\mathcal{Z}\) transform of \(u(k)\). In this section, we present three methods by which state space realizations of Equations (18) and (19) may be obtained. The methods are: (1) the controllable canonical method, (2) the observable canonical method, and (3) the Jordan or diagonal canonical method.

**Controllable Canonical Method**

Consider the pulse transfer function given by Equation (19). The controllable canonical state space realization is given by the following equations:

\[
\begin{bmatrix}
x_1(k+1) \\
x_2(k+1) \\
\vdots \\
x_{n-1}(k+1) \\
x_n(k+1)
\end{bmatrix} =
\begin{bmatrix}
0 & 1 & 0 & \cdots & 0 \\
0 & 0 & 1 & \cdots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & 0 & \cdots & 1 \\
-a_n & -a_{n-1} & -a_{n-2} & \cdots & -a_1 + \phi(k)
\end{bmatrix}
\begin{bmatrix}
x_1(k) \\
x_2(k) \\
\vdots \\
x_{n-1}(k) \\
x_n(k)
\end{bmatrix} +
\begin{bmatrix}
0 \\
0 \\
\vdots \\
0 \\
1
\end{bmatrix} u(k)
\]

(20)

\[
x(k+1) = G X(k) + H u(k)
\]

(21)

\[
y(k) = C X(k) + D u(k)
\]

(22)

The controllable canonical state space realization for a single-input, single-output system described by Equation (19) is computed in MapleV using procedure \(\text{contcano}\) (see Table 5).

**Example 5.** A controllable canonical state space realization of a pulse transfer function will now be computed using procedure \(\text{contcano}\) (see Table 5). Consider the pulse transfer function given by

\[
y(k) = 2.5 + 62(-1)^k y(k) - 21 + 45 y(k-1) - 25 + 31(-1)^k y(k-2)
\]

(23)
A controllable canonical state space realization is computed as follows:
\[
G = \begin{bmatrix}
0 & 1 & 0 \\
1 & 0 & -a_1 \\
0 & 1 & 0
\end{bmatrix}, \quad H = \begin{bmatrix} 0 \\
1 \\
0
\end{bmatrix}
\]
\[
C = \begin{bmatrix}
b_3 & a_2 & b_2 & a_1 & b_1 & a_0
\end{bmatrix}, \quad DD = \begin{bmatrix} b_0 \end{bmatrix}
\]

The computed state space realization is correct. This can be seen by comparing with Equations (20) and (22).

**Example 6.** Consider the following pulse transfer function [21, page 517]:
\[
G(z) = \frac{z+3}{z^2+3z+2}
\]

A controllable canonical state space realization is computed as follows:
\[
G = \begin{bmatrix}
0 & 1 & 0 \\
-2 & -3 & 1
\end{bmatrix}, \quad H = \begin{bmatrix} 0 \\
1 \\
0
\end{bmatrix}, \quad C = \begin{bmatrix} 3 \\
1 \\
0
\end{bmatrix}, \quad DD = \begin{bmatrix} 0 \\
0 \\
0
\end{bmatrix}
\]

The computed results are identical to those obtained in [21].

**Observe Canonical Form**

The state space realization of the pulse transfer function system given by Equation (19) in the observe canonical form is
\[
x_1(k+1) = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & -a_n \\
0 & 1 & 0 & \cdots & 0 & -a_{n-1} \\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\
0 & 0 & \cdots & 1 & 0 & -a_2 \\
0 & 0 & \cdots & 0 & 1 & -a_1 \\
x_n(k+1) & 0 & 0 & \cdots & 0 & 0
\end{bmatrix} x_n(k) + \begin{bmatrix} 0 \end{bmatrix}
\]
\[
x_2(k+1) = \begin{bmatrix} b_n - a_n b_0 \\
b_{n-1} - a_{n-1} b_0 \\
\vdots \\
b_2 - a_2 b_0 \\
b_1 - a_1 b_0 \\
b_0
\end{bmatrix} u(k)
\]
\[
x_1(k), \quad x_2(k), \quad \cdots, \quad x_n(k), \quad y(k) = \begin{bmatrix} x_1(k) \\
x_2(k) \\
\vdots \\
x_n(k)
\end{bmatrix} + b_0 u(k)
\]

**Table 6. Procedure obsvcano**

**Example 7.** To demonstrate the use of procedure obsvcano (Table 6) to compute an observable canonical state space realization, consider the pulse transfer function \(G(z)\) of Example 5. An observable canonical realization is computed as follows:
\[
G = \begin{bmatrix} b_3 & a_2 & b_2 & a_1 & b_1 & a_0
\end{bmatrix}, \quad DD = \begin{bmatrix} b_0 \end{bmatrix}
\]

The computed results are in agreement with Equations (24) and (25).

**Example 8.** An observable canonical state space realization for the pulse transfer function of Example 6 is computed as follows:
\[
G = \begin{bmatrix} b_3 & a_2 & b_2 & a_1 & b_1 & a_0
\end{bmatrix}, \quad DD = \begin{bmatrix} b_0 \end{bmatrix}
\]
The computed results are identical to those obtained in [21].

Jordan or Diagonal Canonical Method

A Jordan or diagonal canonical state space realization may be obtained by effecting a change of states of an observable or controllable canonical state space realization. Consider the discrete-time state space system given by Equations (11) and (12).

Define a new state vector \( \mathbf{X}(k) \) by

\[
\mathbf{X}(k) = \mathbf{P}^{-1}\mathbf{x}(k) \tag{26}
\]

where \( \mathbf{P} \) is the transformation matrix corresponding to the Jordan form (Matrix \( \mathbf{J} \)) such that \( \mathbf{J} = \mathbf{P} \mathbf{G} \mathbf{P}^{-1} \). Then, by substituting Equation (26) into Equations (11) and (12) we obtain

\[
\dot{\mathbf{X}}(k+1) = \mathbf{G}\mathbf{X}(k) + \mathbf{H}\mathbf{U}(k) \tag{27}
\]

\[
\mathbf{Y}(kT) = \mathbf{C}\mathbf{X}(k) + \mathbf{D}\mathbf{U}(k) \tag{28}
\]

where \( \mathbf{G} = (\mathbf{PG}\mathbf{P}^{-1}) \), \( \mathbf{H} = (\mathbf{PH}) \), \( \mathbf{C} = (\mathbf{CP}^{-1}) \), and \( \mathbf{D} = \mathbf{D} \). Equations (27) and (28) are now in the diagonal canonical form, if the eigenvalues of \( \mathbf{G} \) are distinct, or in the Jordan canonical form, if some of the eigenvalues of \( \mathbf{G} \) are repeated. In addition, the diagonal elements of \( \mathbf{G} \) are the eigenvalues of \( \mathbf{G} \).

Procedure \texttt{jordcano} (see Table 7) computes a Jordan or diagonal canonical state space realization.

**Example 9.** By way of an example, we illustrate the use of procedure \texttt{jordcano} (Table 7) to compute a diagonal or Jordan canonical state space realization by using the system representation given in Example 8 as follows:

\[
\texttt{cano}(\mathbf{G}, \mathbf{H}, \mathbf{C}, \mathbf{D}, \mathbf{GG}, \mathbf{HH}, \mathbf{CC}, \mathbf{DD});
\]

\[
\texttt{GG} = \begin{bmatrix} -1 & 0 \\ 0 & -2 \end{bmatrix}, \quad \mathbf{HH} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}, \quad \mathbf{CC} = \begin{bmatrix} 1 & -1 \end{bmatrix}, \quad \mathbf{DD} = [0]
\]

The computed results are identical to those obtained in [21] and the \( \mathbf{GG} \) matrix contains the eigenvalues of \( \mathbf{G} \) which are distinct. Therefore, we have a diagonal canonical state space realization.

State Space Representations of Continuous-Time Systems

The ideas here are totally analogous to what has already been described with regard to discrete-time systems. Tables 5 to 7 can be employed for the different state space realizations of a continuous system. One could simply replace the indeterminate \( z \) with \( s \) (of the continuous transfer function).

Concluding Remarks

In this article, we have shown how to develop algorithms needed for the solution of state space design equations using MapleV. These calculations involve matrix manipulations, eigenvalue-eigenvector determinations, Laplace and Inverse Laplace transformations, and complex, tedious, and error-prone, even for simple examples.

One can carry out these computations with MapleV symbolically, easily, and efficiently. The students have the opportunity to manipulate a variety of different forms of algebraic expressions in almost the same way as would be done using paper and pencil.

It has also been shown that one can save time and effort, and hence tackle more realistic problems. This would enable the students to become active participants in mathematical exploration, rather than passive recipients of a fixed body of facts and
algorithms. The procedures also replace tables of functions, series and integrals, using exact arithmetic to produce accurate answers.

It has been shown that the easiness with which MapleV provides analytical results allows for a student to focus on the ideas being developed rather than overcoming calculational difficulties. The very process of programming with MapleV encourages appreciation of the important aspects of mathematical investigation, because there is a one to one correspondence between the symbolic code and the mathematical algorithms being programmed.

The symbolic, numerical, graphical, and programming capabilities of MapleV offer exciting possibilities in different areas of the engineering and process control curriculum. We feel that more applications are forthcoming.

References


