International Federation of Automatic Control

COMPUTER AIDED DESIGN IN CONTROL SYSTEMS

Preprints of the IFAC Symposium, Swansea, UK
15-17 July 1991

Edited by

H.A. BARKER
University of Wales, Swansea, UK

PREPRINTS

PERGAMON PRESS

NUMERICAL PROPERTIES OF TRAJECTORY REPRESENTATIONS OF POLYNOMIAL MATRICES

François E. Cellier and SungDo Chi
Department of Electrical and Computer Engineering
The University of Arizona
Tucson, Arizona 85721
U.S.A.

Abstract. In the early 1970s, several researchers reported results relating to the design of multivariable linear systems represented by polynomial matrices. In particular, the book by Wolovich (1974) found widespread resonance. In the sequel, however, the success of these techniques was rather limited since a manual application of the proposed algorithms is atrocious for all but the most trivial systems, whereas appropriate CACSD tools that would make use of these techniques were not available. The main reasons for this deficiency were twofold: (i) polynomial matrix operations require symbolic processing, a computational technique that was still in its infancy in the 1970s, and (ii) the numerical properties of frequency domain operations were considered dubious. In this paper, the numerical properties of frequency domain operations are analyzed. The two classical data representations (coefficients and roots) are reviewed, and two new data representations, trajectories and coefficient spectra, are proposed.

Keywords. Canonical forms; computer–aided system design; control system design; data structures; minimum–data representation; numerical methods; polynomial matrices.

THE CURSE OF DIMENSIONALITY

The design of controllers for high dimensional systems is a numerically difficult problem. The problem will be illustrated by means of an example. An optimal controller for a linear system with quadratic performance index is to be designed. This problem leads naturally to the solution of a Matrix–Riccati equation.

The following experiment was performed. In an arbitrary SISO system of the type:

$$\dot{\mathbf{x}} = \mathbf{A} \cdot \mathbf{x} + \mathbf{b} \cdot u \tag{1}$$

the elements of **A** and **b** were picked at random using random numbers uniformly distributed between 0.0 and 1.0. Obviously, this system is unstable. In the performance index:

$$PI = \int_{t_0}^{t_f} \mathbf{x}^T \cdot \mathbf{Q} \cdot \mathbf{x} + r \cdot u^2 \; dt \tag{2}$$

Q was chosen as a positive definite symmetric random matrix of dimensions $n \times n$ and r as a random scalar. The dimension of the system, n, was kept variable. The system was coded in CTRL–C (SCT, 1989).

Two different Riccati algorithms were chosen to compute the feedback vector:

$$u = -\mathbf{k}^T \cdot \mathbf{x} \tag{3}$$

which determines the optimal control. On the one hand, the algorithm provided as a system function in CTRL–C was used and on the other hand, the eigenvalue/eigenvector algorithm described by MacFarlane (1963) was chosen. The CTRL–C version of the latter algorithm is presented here:

```
// [k,p] =LQR2(a,b,q,r)
//
// Compute the solution to the
// algebraic Matrix–Riccati equation
//
[n,m] =SIZE(b);
h = [ a , -(b/r)*b' ; -q , -a' ];
[v,l] =EIG(h);
k = 0;
for i = 1:2*n, ...
    if REAL(l(i,i)) < 0 then ...
        k = k+1; ...
        v(:,k) = v(:,i); ...
    end, ...
end
v1 = v(1:n,1:n);
v2 = v(n+1:2*n,1:n);
p = v2/v1;
k = (r\(b'))*p;
k =REAL(k);
return
```

The resulting $\mathbf{k}^T$ vectors were computed using these two algorithms for various orders n. The infinity norm of the difference between the two resulting $\mathbf{k}^T$ vectors was then determined as a function of the system order, and an exponential regression metamodel for the error was also computed. The CTRL–C code used to numerically compare the two LQR algorithms is presented here:

```
//[err,erl,ra,rb] =RTESTLQR(n)
//
// This function tests LQR2 versus LQR
// up to any order n
//
```

```
deff lqr2 -c
TERM ='4100';
HARD ='tekf';
//
// Loop over all orders
//
ord = (1 : n)';
for i = ord',...
    a =RAND(i); ...
    b =RAND(i,1); ...
    aux =RAND(i); ...
    q = aux' * aux; ...
    r =RAND(1); ...
    k1 =LQR(a,b,q,r); ...
    k2 =LQR2(a,b,q,r); ...
    err(i) =NORM(k1 - k2,'inf'); ...
    if err(i) = 0, err(i) = eps; end; ...
end
erl = err/eps;
//
// Perform a regression analysis
//
rx = [ ord , ONES(ord) ];
ry =LOG(err);
coef = rx\ry;
ra = coef(1);
rb = coef(2);
erb =EXP(rb);
reg = erb * (EXP(ord) * *ra);
//
// Plot the results
//
erase
window('221')
plot(ord,[err,reg],'logy,grid')
title('Comparison : LQR2 and LQR')
xlabel('SystemOrder')
ylabel('Error')
pause
redhard > rtestlqr.tekf
replot
redhard -close
erase
//
return
```

The results of this experiment are shown in Fig. 1 using logarithmic format for the dependent axis.

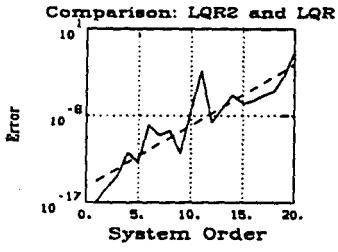

Figure 1. Numerical comparison of two LQR algorithms.

The solid line represents the actual difference between the two algorithms, whereas the dashed line represents the exponential metamodel. The code used to generate Fig. 1 is listed in the paper since, in this way, the presented results are fully reproducible and since the code is sufficiently short to warrant its being listed.

CTRL–C (SCT, 1989) uses double–precision arithmetics for all computations. Consequently, the smallest number that can be additively distinguished from 1.0 on a VAX is $\epsilon = 1.3878 \cdot 10^{-17}$. For low orders n, the numerical error between the two feedback vectors is of the same order of magnitude. However, already for a 20^{th}–order system, the error is roughly 0.1, and thus, basically represents numerical noise. Approximately one decimal per order is lost in this algorithm.

While the results shown in Fig. 1 certainly depend on the algorithm that is used, they nevertheless reflect a general truth. The same methodology was applied to a number of different control problems and in most cases, similar patterns were found. A loss of one decimal per order seems to be the "going rate." This is why all respectable CACSD programs are coded in double–precision arithmetics. In single precision, numerical noise dominates the results in controller designs for all systems of higher than approximately sixth order. In double precision, operations can be safely performed on systems up to approximately 14^{th} order.

High–dimensional systems are numerically problematic. Most control engineers apply model reduction techniques to find a lower order model for any high–dimensional system and apply the controller design to the low–order model instead of the original high–order system. It is essential that the control engineer know about these problems, since in any event the design of high–dimensional systems is a numerically difficult task.

The question is: Is it possible to find a data representation in which the curse of dimensionality does not exist? This paper provides a partial answer to that question.

MINIMUM–DATA REPRESENTATIONS AND REDUNDANCY

It is well known that a linear system can be represented in many different ways in a state–space form. That is, the state–space representation of a linear system contains redundancy that can be exploited to improve the numerical behavior of operations performed on the model. The trick is to minimize the largest parameter sensitivity in the model, which is equivalent to saying that the system behavior should be made about equally sensitive to variations in all model parameters.

Minimum–data representations are representations that have exactly as many parameters as the system has degrees of freedom (Cellier and Rimvall, 1988). Since the transfer function of a SISO system is unique, it is evident that an n^{th}–order system has exactly $2 \cdot n$ degrees of freedom and therefore, any state–space representation that contains exactly $2 \cdot n$ variable parameters is a minimum–data representation. Typical examples of minimum–data representations are the controller–canonical form (a lower companion form) in which only the last row of the system matrix (n parameters) and the output vector (n parameters) are variable, and the Jordan–canonical form in which only the diagonal elements of the system matrix (n parameters) and the output vector (n parameters) are variable.

Minimum–data representations are appealing since they minimize the number of parameters to be identified in the model. Unfortunately, none of the minimum–data representations is numerically attractive. The controller– (or observer–) canonical forms are closely related to the transfer function represented in a defactorized form. Given the system:

$$G(s) = \frac{b_0 + b_1 \cdot s + b_2 \cdot s^2 + \ldots + b_{n-1} \cdot s^{n-1}}{a_0 + a_1 \cdot s + a_2 \cdot s^2 + \ldots + a_{n-1} \cdot s^{n-1} + s^n} \qquad (4)$$

its controller–canonical representation can immediately be written down:

$$\dot{\mathbf{x}} = \begin{pmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \\ -a_0 & -a_1 & -a_2 & \dots & -a_{n-1} \end{pmatrix} \cdot \mathbf{x} + \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix} \cdot u \quad (5a)$$

$$y = \begin{pmatrix} b_0 & b_1 & b_2 & \dots & b_{n-1} \end{pmatrix} \cdot \mathbf{x} \quad (5b)$$

The variable parameters of the controller– (and observer–) canonical forms are the coefficients of the transfer function. This unravels immediately the numerical problems associated with controller– (and observer–) canonical forms. If the 20^{th}–order polynomial:

$$Q(s) = a_0 + a_1 \cdot s + a_2 \cdot s^2 + a_{19} \cdot s^{19} + s^{20} \quad (6)$$

is to be evaluated for $s = 10$ (or any complex vector with an absolute value of 10), the higher powers are obviously dominant since 10^{20} is a very large number. Consequently, the value of Q is very sensitive to variations in a_{n-1}, while it is not sensitive at all to variations in a_0. On the other hand, if the same polynomial is evaluated for $s = 0.1$ (or any complex vector with an absolute value of 0.1), it can be concluded that just the opposite is true since 0.1^{20} is negligibly small. Consequently, the value of Q is now much more sensitive to a_0 than to a_{n-1}. It has thus become evident that polynomial coefficients are extremely poorly balanced with respect to their sensitivities and that is the numerical crux of this data representation.

The parameters of the Jordan–canonical form are the eigenvalues and residua of the system. This data representation is related to the factorized form of the transfer function, or more precisely, to the partial fraction expansion of the transfer function. The data representation is well balanced with respect to parameter sensitivities. Moreover, since the Jordan–canonical form decouples a high–dimensional system into its individual modes, the curse of dimensionality vanishes in this representation. Unfortunately, it has been shown by Golub and Wilkinson (1976) that the transformation into and out of this data representation is extremely ill–conditioned. Thus, if a system happens to be naturally specified in this form, that may be the best thing that can occur, but if the system is initially specified in any other form, it may not be advisable to convert it to Jordan form and, in fact, this may not be feasible at all without committing numerical suicide on the way.

Consequently, minimum–data representations may not be as attractive a proposition as the initial glance does suggest. Since polynomials (and transfer functions) were traditionally represented either in defactorized (coefficient) or factorized (root) form, which are two minimum–data representations, numerical operations on such polynomials (or polynomial matrices in the multivariable case) are somewhat dubious. However, it is important to realize at this point that what makes the operations dubious is the chosen data representation and not the fact that the model is analyzed in the frequency domain. It is therefore a legitimate question to ask whether it is possible to find other data representations in the frequency domain that are not minimum–data representations and that defuse the time bomb inherent in all minimum–data representations, i.e., the curse of dimensionality. This is the subject matter of this paper.

TRAJECTORY REPRESENTATION

An n^{th}–order polynomial can be completely characterized by any $n+1$ supporting values. For example, the polynomial:

$$P(s) = 3 \cdot (s+1) \cdot (s+2) = 3 \cdot s^2 + 9 \cdot s + 6 \quad (7)$$

can be completely characterized by Table 1:

TABLE 1 Trajectory Representation of $P(s)$

s	$P(s)$
−1.0	0.0
0.0	6.0
+1.0	18.0

Thus, the three data structures:

$$P_{coef} = (\,6.0 \quad 9.0 \quad 3.0\,) \quad (8a)$$

$$P_{root} = (\,3.0 \quad -1.0 \quad -2.0\,) \quad (8b)$$

$$P_{traj} = \begin{pmatrix} -1.0 & 0.0 & +1.0 \\ 0.0 & 6.0 & 18.0 \end{pmatrix} \quad (8c)$$

can all be used to characterize the polynomial $P(s)$ if interpreted correctly. P_{coef} stores the coefficients of $P(s)$ in the order of ascending powers of s, P_{root} stores the gain value of $P(s)$ followed by its roots in an arbitrary sequence, and P_{traj} stores an arbitrary number of supporting s values in the first row and the corresponding $P(s)$ values in the second.

For illustration, the three representations are repeated here for another polynomial:

$$Q(s) = 2 \cdot (s+3) = 2 \cdot s + 6 \quad (9)$$

which can be represented as:

$$Q_{coef} = (\,6.0 \quad 2.0\,) \quad (10a)$$

$$Q_{root} = (\,2.0 \quad -3.0\,) \quad (10b)$$

$$Q_{traj} = \begin{pmatrix} -1.0 & 0.0 & 1.0 \\ 4.0 & 6.0 & 8.0 \end{pmatrix} \quad (10c)$$

It can be noticed that Q_{traj} is no longer a minimum–data representation, since two points would have been enough to characterize this first–order polynomial. In the coefficient representation, addition of $P(s)$ and $Q(s)$:

$$A(s) = P(s) + Q(s) \quad (11)$$

is easily accomplished. Q_{coef} is simply expanded from the right with a zero and coefficients of the same power are added:

$$A_{coef} = (\,12.0 \quad 11.0 \quad 3.0\,) \quad (12)$$

The multiplication of $P(s)$ with $Q(s)$:

$$M(s) = P(s) \cdot Q(s) \quad (13)$$

is a little more tricky. The coefficient vectors must be convolved:

$$M_{coef} = \mathrm{CONV}(P_{coef}, Q_{coef}) = (\,36.0 \quad 66.0 \quad 36.0 \quad 6.0\,) \quad (14)$$

In the second data representation, addition is an impossibly difficult task. In order to add two polynomials represented by their roots, both polynomials must first be defactorized, then they must be added using the previously advocated algorithm, and finally they must be refactorized ... and the numerical disaster is already programmed.

$$A_{root} = (\,3.0 \quad -0.4583 + 0.1998\mathrm{i} \quad -0.4583 - 0.1998\mathrm{i}\,) \quad (15)$$

However, multiplication is trivial. The gain factors are simply multiplied and the roots are concatenated to each other:

$$M_{root} = (\,6.0 \quad -1.0 \quad -2.0 \quad -3.0\,) \quad (16)$$

In the third data representation, both operations are easy to perform if the two polynomials are sampled over the same domain (as this is the case in the preceding example). To add two polynomials $P(s)$ and $Q(s)$, the individual values of $P(s)$ and $Q(s)$ at the sampling points are simply added:

$$A_{traj} = \begin{pmatrix} -1.0 & 0.0 & +1.0 \\ 4.0 & 12.0 & 26.0 \end{pmatrix} \tag{17}$$

To multiply two polynomials $P(s)$ and $Q(s)$, the individual values of $P(s)$ and $Q(s)$ at the sampling points are simply multiplied:

$$M_{traj} = \begin{pmatrix} -1.0 & 0.0 & +1.0 \\ 0.0 & 36.0 & 144.0 \end{pmatrix} \tag{18}$$

The trajectory representation has four striking algebraic properties:

(1) It is not a minimum–data representation. Already in the preceding example, $Q(s)$ was represented by more data points than necessary. As many points can be added as are deemed suitable and this redundancy may be exploited to improve the numerical behavior.

(2) The primitive operations addition and multiplication can both be performed within the data representation and are extremely simple. Since most operations useful for linear algebra can be reduced to series of additions and multiplications, this is an important asset.

(3) Addition and multiplication can easily be implemented on a parallel processor using an SIMD architecture. This makes the redundancy less painful from a computational perspective.

(4) The curse of dimensionality has vanished.

The last property deserves to be explained in a little more depth. If all polynomials are sampled over the same domain, additions and multiplications affect only one supporting value at a time. Errors can accumulate only within that supporting value. For example, if a numerical error is made while two polynomials are added at the supporting value -1.0, this error will never infect the polynomial at the supporting values 0.0 or $+1.0$. Thousands of additions and multiplications can be performed in a row. Errors made in the addition or multiplication of two numbers relating to one supporting value will never spread across the insurmountable barrier between supporting values. Consequently, the errors made in thousand additions and multiplications of polynomials of order 1 and the errors made in the same thousand additions and multiplications of polynomials of order 50 are exactly the same. As long as all computations are performed within the trajectory data representation scheme, the order of the polynomials has become immaterial, and therefore, the curse of dimensionality has been successfully banned.

Although it would be feasible to operate directly on transfer functions, i.e., on rational functions or rational function matrices, this is not recommended due to the numerically detrimental effects of singularities. It is better to operate on polynomial matrices using the ideas and algorithms advocated by Wolovich (1974).

THE DARTBOARD DOMAIN

The numerical properties of algorithms converting polynomials into and out of their trajectory representation must still be analyzed. If polynomials are initially represented in a different form, e.g. in coefficient form, or if the final results are wanted in another form, it is important that the errors be analyzed that are picked up on the transitions from and to the trajectory representation.

The first problem is: How are the supporting values chosen? If $s = 10$ is used as a supporting value, and if a polynomial of high order is evaluated, and if $P(s = 10)$ is nevertheless decently small, it is obvious that $P(s)$ must be the result of a cancellation of relatively large terms with opposite signs. Thus, a lot of accuracy is lost right there. Consequently, it makes sense to select the domain values (supporting values) equidistantly spaced along the unit circle of the complex plane. With this choice, $|s| = 1$ for all values of s and therefore, the sensitivity of $P(s)$ will be balanced with respect to all coefficients.

The inverse transformation can be written as a regression problem. If the polynomial:

$$W(s) = a_0 + a_1 \cdot s + a_2 \cdot s^2 + \ldots + a_{n-1} \cdot s^{n-1} + s^n \tag{19}$$

is known for k values of s:

$$\begin{aligned} W_1 &= a_0 + a_1 \cdot s_1 + a_2 \cdot s_1^2 + \ldots + a_{n-1} \cdot s_1^{n-1} + s_1^n && (19a) \\ W_2 &= a_0 + a_1 \cdot s_2 + a_2 \cdot s_2^2 + \ldots + a_{n-1} \cdot s_2^{n-1} + s_2^n && (19b) \\ \ldots &= \ldots && \ldots \\ W_k &= a_0 + a_1 \cdot s_k + a_2 \cdot s_k^2 + \ldots + a_{n-1} \cdot s_k^{n-1} + s_k^n && (19k) \end{aligned}$$

where $k \geq n+1$, Eqs. (19a–k) can be rewritten in a matrix form as:

$$\begin{pmatrix} W_1 \\ W_2 \\ \vdots \\ W_k \end{pmatrix} = \begin{pmatrix} s_1^0 & s_1^1 & s_1^2 & \ldots & s_1^n \\ s_2^0 & s_2^1 & s_2^2 & \ldots & s_2^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ s_k^0 & s_k^1 & s_k^2 & \ldots & s_k^n \end{pmatrix} \cdot \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ 1 \end{pmatrix} \tag{20}$$

and therefore, the coefficient vector can be found by solving a linear regression problem involving a Vandermonde matrix spanned over the domain vector. In MATLAB (MathWorks, 1987) or CTRL–C (SCT, 1989) this linear regression problem can be solved using the "\" operator:

$$Coef = \mathrm{VDM}(s) \backslash W \tag{21}$$

where s denotes the known vector of supporting values and W denotes the known vector of polynomial values at the given supporting values. $Coef$ is the resulting coefficient vector after solving the linear regression problem in a least square's sense.

Furthermore, if the domain vector is always chosen equidistantly spaced along the unit circle and if the same number of supporting values (e.g., 64) is maintained all the time, a (pseudo)inverse of the Vandermonde matrix can be computed once and for all, and the coefficients can be regenerated from the polynomial values by means of a simple multiplication. In reality, the algorithm is a little more tricky than that, since it is necessary to somehow keep track of the polynomial order.

Unfortunately, there are more problems. For illustration, the following polynomial will be considered:

$$R(s) = 5.0 \cdot (s + 99.0) \cdot (s + 100.0) \cdot (s + 101.0) \tag{22}$$

Once the polynomial $R(s)$ has been evaluated at 64 values of s equidistantly spaced along the unit circle, the three roots can never again be regained from this trajectory representation with any degree of accuracy, since the regression algorithm performs well in the case of interpolation but not extrapolation. Roots can be found directly (without evaluating the coefficients first), e.g., by use of barycentric interpolation, but the basic problem remains the same. Thus, a unit circle domain may not be the best choice for all types of problems.

For this reason, the dartboard domain is proposed. A dartboard domain is a domain consisting of several concentric circles around the origin. The domain values are equidistantly spaced along each of these circles. For ex-

ample, 32 points can be used along each one of five circles with the radii 0.01, 0.1, 1.0, 10.0, and 100.0. In the forward translation from either coefficients or roots to trajectories, large errors may be introduced for some of these points, and small errors for others. However, these errors will always remain constrained to the point where they were first generated. They can never spread across to other points ... until the time of the backtransformation from the trajectory representation to either coefficient or root representations. At that time, it is possible to select a subset of the available points only, and several different subsets can be investigated separately to check how much variation is obtained in the resulting parameters. Thus, the redundancy of the trajectory representation can be exploited to select those domain values for the backtransformation that produce the smallest errors. It may be a little tricky to fully automate this procedure, but this can be done.

COEFFICIENT SPECTRA

However, yet another new data representation for polynomial matrix operations will be proposed. In the coefficient representation, addition is performed by adding the coefficient vectors, whereas multiplication is performed by convolving the coefficient vectors. This leads naturally to the following idea. In Fig. 2, the coefficient values of the polynomial

$$V(s) = 1 + 2 \cdot s^2 + 2 \cdot s^3 + s^4 - s^5 + s^6 \qquad (23)$$

are plotted as a function of their respective orders:

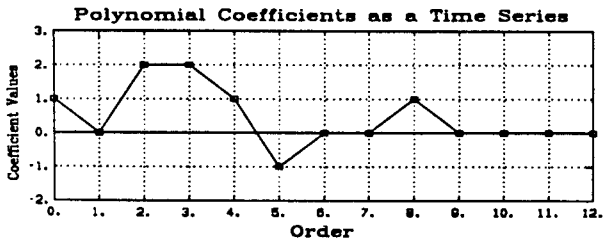

Figure 2. Coefficients represented as a time series.

Figure 2 can be thought of as a "time series" of which the FFT can be computed. Zero padding from the right will be used to at least double the number of memory cells and fill it up to the next power of 2. Thus, if the largest considered polynomial order is 32, a choice of 64 as the length of the coefficient vectors is adequate.

After this transformation, polynomials can still be added by adding their spectral lines, but now, polynomials can also be multiplied by multiplying their spectral lines, since in the FFT the convolution is mapped into a multiplication.

Consequently, the algebraic structure of the coefficient spectra representation is exactly the same as that of the trajectory representation. The coefficient spectra representation shares the four algebraic properties mentioned earlier for the trajectory representation. In particular, the curse of dimensionality is also banned from this new data representation scheme.

Furthermore, also the numerical properties of the trajectory representation and the coefficient spectra representation are remarkably similar. Nevertheless, the coefficient spectra representation has been found to be slightly better since:

(1) it is not necessary to select a domain,

(2) the FFT and IFFT operations are numerically more benign than the evaluation of a polynomial and the solution of the linear regression problem, and finally

(3) the FFT and IFFT operations lend themselves to an efficient parallel implementation on an SIMD processor architecture.

POLPAC

POLPAC is an experimental toolkit for polynomial matrix operations making use of the newly proposed data representation schemes. POLPAC is currently available as a CTRL–C function library. A corresponding MATLAB toolbox is under development. It is planned to implement POLPAC as a MATLAB toolbox on a machine with a DSP chip. Most of the POLPAC operations can be elegantly implemented on the DSP chip. It hasn't been decided yet what physical processor configuration will be selected. The major problem is the overhead created by shipping long data vectors back and forth between the CPU and the DSP chip. The NEXT machine might provide for an attractive architecture, but MATLAB has not yet been made available for the NEXT.

CONCLUSION

In this paper, two new data representation schemes for polynomial matrix operations were proposed. It has been shown that these data representation schemes do not suffer from the well known curse of dimensionality. Experimental software that incorporates the proposed data structures has also been implemented. Unfortunately, the current implementation is executing too slowly for practical applications. Only the principles have thus been proven.

Much research is still needed. One student is currently working on implementing several of the algorithms from Wolovich (1974) in POLPAC so that a better feel for the numerical behavior of complex control algorithms using the proposed data structures can be obtained. However, due to the interpretive nature of the current implementation, only small problems can be tackled at this point.

REFERENCES

Cellier, F.E., and C.M. Rimvall (1988). "Computer–Aided Control Systems — Techniques and Tools," In: *Systems Modeling and Computer Simulation* (N.A. Kheir, Ed.), Marcel Dekker, New York, pp. 631–679.

Golub, G.H., and J.H. Wilkinson (1976). "Ill–Conditioned Eigensystems and the Computation of the Jordan Canonical Form," *SIAM Review*, 18(4), pp. 578–619.

MacFarlane, A.G.J. (1963). "An Eigenvector Solution of the Optimal Linear Regulator," *J. Electron. Control*, 14, pp. 643–654.

MathWorks (1987). *Pro–MATLAB with System Identification Toolbox and Control System Toolbox — User Manual*, The MathWorks, Inc., 21 Eliot St., South Natick MA 01760.

SCT (1989). *CTRL–C User's Guide*, Version 4.3, Systems Control Technology, Inc., 2300 Geng Rd., Palo Alto CA 94303.

Wolovich, W.A. (1974). *Linear Multivariable Systems*, Springer–Verlag, New York.