MATRIXx: A Data Analysis, System Identification, Control Design and Simulation Package

Robert Walker, Charles Gregory, Jr., and Sunil Shah
Integrated Systems Inc., 151 University Ave., Ste. 400, Palo Alto, CA 94301

Abstract

MATRIXx is designed as a software system to perform all the steps in the control design cycle starting from system modeling to data analysis, identification, control synthesis and simulation. The package features a powerful matrix interpreter, a user-friendly environment with device independent graphics, state-of-the-art numerical algorithms for reliable computations and user-transparent file management. The program is implemented in ANSI-77 FORTRAN, and is designed to run on any allowing interactive execution of FORTRAN programs.

Introduction

Computer-aided-design tools have a significant role to play in the future of control design practice. Each step of the control design cycle starting from system modeling to data analysis, system identification, model reduction, control design, simulation and implementation can be made more efficient with software tools. MATRIXx offers classical as well as modern approaches to control design.

MATRIXx provides a comprehensive set of capabilities in a single integrated package with uniform data and file formats. Most "bookkeeping" chores are performed by the software, leaving the control designer free to tackle control problems. Graphics allow the designer to rapidly visualize information. The system-build capability lets the designer see the schematic block diagram as it is built.

MATRIXx is built on reliable numerical algorithms drawn from LINPACK, EISPACK and recent research in numerical analysis. Numerical stability and robustness are always important, but particularly for system dimensions exceeding approximately 20. The designer can handle systems with large dimensions in MATRIXx, relying on superior numerical software with comprehensive reporting and control of numerical errors.

Perhaps the most remarkable aspect of MATRIXx is its simplicity. Commands are natural, simple, yet powerful. Online documentation with HELP commands, extensive, direct, jargon-free diagnostics and reasonable defaults make MATRIXx easy to use. The interpreter can execute higher level constructs called MACROS and command files. They allow personalization of MATRIXx commands, eliminating superfluous interactions with the designer. The command files are typically less than ten lines long and consist of interpreter statements.

Creation and modification of specific design and analysis procedures are easy because of the rich set of primitives and the hierarchical structure.

MATRIXx inherited many of its capabilities from its predecessor MATLAB [6], developed by Cleve Moler, which was developed as a pedagogical aid for numerical analysis. In contrast, MATRIXx was designed to serve as an engineering design tool. The code-length of MATRIXx (although code-length is not necessarily a measure of capability) is five times that of MATLAB. MATRIXx is significantly more efficient in memory usage and computations. Capabilities to perform control synthesis, system identification, signal processing and simulation required a number of new numerical algorithms. MATRIXx also incorporates a state-of-the-art device independent graphics package which conforms to the 1979 core system defined by Graphics Standard Planning Committee of ACM/SIGGRAPH.

0272-1708/82/1200-0030$00.75 © 1982 IEEE
Control Design and System Analysis

MATRIXX offers a variety of approaches to control design, classical Bode, Nyquist and Root-locus plots, modern Linear-Quadratic-Gaussian (LQG) techniques, geometric approaches, as well as others. Rather than advocating a particular approach, MATRIXX provides the underlying numerical algorithms in a user-friendly environment, allowing the designer to choose the approach.

The following example illustrates typical classical design capabilities (see Ogata [8], pp. 515-517). The system is represented by \( y = \frac{20}{s(s+1)(s+2)}u \), or equivalently \( x = Fx + Gu \), \( y = Hx + Ju \) where

\[
F = \begin{bmatrix} 0 & 1 & 0 \\ 0 & -2 & 1 \\ 0 & 0 & -1 \end{bmatrix}, \quad G = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad H = [1 \ 0 \ 0], \quad J = [0]
\]

To enter \( F, G, H \) and \( J \), the designer types

\[
\{F = \begin{bmatrix} 0 & 1 & 0 \\ 0 & -2 & 1 \\ 0 & 0 & -1 \end{bmatrix}; \quad \{G = \begin{bmatrix} 0 \\ 0 \end{bmatrix}; \quad \{H = [1 \ 0 \ 0]; \quad \{J = [0]\}
\]

MATRIXX prompt \( \{ \) indicates the beginning of a command line. Numerical values are entered in free format. The square brackets \( [ \) indicate definition of a new matrix, to be entered row-wise. The semicolons ‘;’ or end-of-lines separate two rows inside square brackets. Spaces or commas separate individual elements. A convenient way of representing system is by a system matrix

\[
S = \begin{bmatrix} F & G \\ H & J \end{bmatrix}
\]

Entering

\[
\{S = \begin{bmatrix} F & G \\ H & J \end{bmatrix}
\]

defines the S matrix. \( S \) is printed out as below because the semicolon did not end the command.

\[
S = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -2 & 1 & 0 \\ 0 & 0 & -1 & 20 \\ 1 & 0 & 0 & 0 \end{bmatrix}
\]

To obtain the frequency response of the open-loop system \( S \), with 3 states, from \( \omega = 10^{-2} \) to \( 10^{1} \) rad/s, one enters

\[
\{RANG = [-2 \ 1]; \quad \{NS = 3;
\]

\[
[F1, FR] = \text{FREQ}(S, NS, RANG);
\]

FR is now the complex frequency response evaluated at frequency points in \( F1 \). Magnitude and phase of FR are obtained with the following:

\[
\{\text{OMAGN} = \text{ABS}(FR); \quad C = 180/\pi;
\]

\[
\{\text{PHASE} = \text{IMAG}(\text{LOG}(FR))*C;
\]

To plot MAGNitude against frequency \( F1 \), both on log scales, on the upper part of the screen, enter

\[
\{\text{PLOT}(F1, \text{MAGN}, \text{'LOG UPPER'})
\]

To plot PHASE against \( F1 \), on lower part of the screen with log scale on X-axis, keeping the upper plot, enter

\[
\{\text{PLOT}(F1, \text{PHASE}, \text{'LOGX LOWER KEEP'})
\]

As seen in Fig. 1. the same variables used to invoke the PLOT are used as default labels. The Bode plot clearly indicates instability with a negative phase margin.

Now suppose we want to cascade the plant with a lead-lag network that has the transfer function \( (s+.15)/(s+.015) \cdot (s+.7)/(s+7) \). The lag portion of the transfer function \( (s+.15)/(s+.015) = 1 + .135/(s+.015) \) is represented by

\[
S1 = \begin{bmatrix} -.015 & 1 \\ .135 & 1 \end{bmatrix}
\]

It is easy to verify that if

\[
S1 = \begin{bmatrix} F1 & G1 \\ H1 & J1 \end{bmatrix}
\]

their cascade connection, from \( u_1 \) to \( y_2 \), is represented by

\[
S2 = \begin{bmatrix} F2 & G2 \\ H2 & J2 \end{bmatrix}
\]

We can write a small MACRO or a command file to perform the cascade connection; however, to keep the presentation simple, we just identify terms in the above expression and form the matrices

\[
S12 = \begin{bmatrix} F1 & 0 & G1 \\ G2*H1 & F2 & G2*J1 \\ J2*H1 & H2 & J2*J1 \end{bmatrix}
\]

Fig. 1.
explicitly \((s+.7)/(s+7) = -.63 + 1/(s+7)\) so that

\[
S2 = \begin{bmatrix}
-7 & 1 \\
-6.3 & 1
\end{bmatrix}
\]

\[
\phi_F = [-.015 0; .135 -7];
\]

\[
GC = [1; 1]; HC = [.135 -6.3];
\]

\[
JC = [1];
\]

\[
S12 = [FC GC; HC JC];
\]

The frequency response of the lead-lag network is given by

\[
\phi[F1, FRC] = FREQ(S12, 2, RANG);
\]

In order to save typing PLOT options we define

\[
\phiBODM = 'LOG UPPER YLAB/MAGNITUDE/XLAB/FREQUENCY';
\]

\[
\phiBODP = 'LOGX LOWER KEEP YLAB/PHASE/XLAB/FREQUENCY';
\]

and then

\[
\phiPLOT(F1, ABS(FRC), BODM),
\]

\[
PLOT(F1, C*IMAG(LOG(FRC)), BODP)
\]

generates Fig. 2.

Similarly, for Nyquist plots

\[
\phiONYQP = XLAB/REAL PART/YLAB/IMAGINARY PART/TITLE/NYQUIST PLOT/UPPER RIGHT';
\]

The compensated system is given by

\[
FF = \begin{bmatrix}
FC & 0 \\
G*HC & F
\end{bmatrix}, GF = \begin{bmatrix}
GC \\
G
\end{bmatrix}
\]

\[
HF = [J*HC H]; JF = [0]
\]

\[
\phiFF = [FC O*ONES(2, 3); G*HC F]; HF = [J*HC H];
\]

\[
JF = 0; GF = [GC; G];
\]

The frequency response of the compensated system \(SF\) which has 5 states is given by the element by element multiplication.*

\[
\phiFRF = FR.*FRC
\]

\[
\phiPLOT(F1, ABS(FRF), BODM),
\]

\[
PLOT(F1, C*IMAG(LOG(FRF)), BODP)
\]

\[
\phiFRF = FR.*FRC
\]

\[
\phiFRING = [ABS(FRF), ABS(FR)];
\]

\[
\phiPHAS = C*IMAG([LOG(FRF), LOG(FR)]);
\]

\[
\phiPLOT(F1, MAG, BODM),
\]

\[
PLOT(F1, PHAS, BODP)
\]

generates the Bode plot for the compensated system. To get uncompensated and compensated plant Bode plots we use multichannel plotting. Each column of the first two parameters in the PLOT command can be a separate channel.

\[
\phiFRF = FR.*FRC
\]

\[
\phiPLOT(F1, ABS(FRF), BODM),
\]

\[
PLOT(F1, C*IMAG(LOG(FRF)), BODP)
\]

generates Fig. 3. The solid lines correspond to the compensated plant.
The two Nyquist plots, limited to a region around origin, are given by
\[
\text{OFRR} = \text{REAL}([\text{FRF} \ \text{FR}]);
\]
\[
\text{FRI} = \text{IMAG}([\text{FRF}, \text{FR}]);
\]
\[
\text{NYQP} = \text{'XLAB/REAL PART/YLAB/IMAGINARY PART/TITLE/NYQUIST PLOT/FULL GRID...}
\]
\[
\text{XMAX} = 1 \quad \text{XMIN} = -10
\]
\[
\text{DX} = 1 \quad \text{YMAX} = 1
\]
\[
\text{YMIN} = -10 \quad \text{DY} = 1
\]
then Fig. 4 is generated by
\[
\text{OPLOT(FRR, FRI, NYQP)}
\]
The time response is computed with the TIMR primitive, which generates the impulse response of a linear system. To obtain the step response in the closed-loop, we define
\[
\text{STP} = [(FF - GF*HF) GF O*GF O*HF 0 1]
\]
\[
\begin{bmatrix}
0 & 1 \\
HF & 0 & 0
\end{bmatrix}
\]
STP represents an integrator in cascade with the unity feedback closed-loop system.
\[
[T, TR] =
\]
\[
= \text{TIMR(STP, 6, [30, 100])};
\]
\[
\text{PLOT(T, TR,'GRID')}
\]
generates Fig. 5.
The next example illustrates the Linear-Quadratic-Gaussian (LQG) approach to control design. The following matrices describe a linearized model of an F-4 aircraft longitudinal dynamics model, flying at Mach = .9 at 15,000 feet.

\[
\begin{bmatrix}
\frac{d}{dt}
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3 \\
x_4 \\
x_5
\end{bmatrix}
= \begin{bmatrix}
-.0068 & .0015 & -.6594 & -.32 & 1 \\
.0011 & -1.98 & 14.484 & -.0145 & 0 \\
.341 & -1.98 & -.488 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & -1
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3 \\
x_4 \\
x_5
\end{bmatrix}
\]

\[
+ \begin{bmatrix}
.0321 & 0 \\
-.706 & 0 \\
-15.99 & 0 \\
0 & 0 \\
0 & .1
\end{bmatrix}
\begin{bmatrix}
u_1 \\
u_2
\end{bmatrix}
\]

\[
\begin{bmatrix}
y_1 \\
y_2
\end{bmatrix}
= \begin{bmatrix}
1.022 & 0 & 0 & 0 & 0 \\
0 & .0689 & 0 & 0 & 0
\end{bmatrix}
\begin{bmatrix}
x
\end{bmatrix}
\]

Eigenvalues of the open-loop system are given by entering
\[
\text{OEIG(F)}
\]
\[
\text{ANS} = -0.4879 + 5.3757i
\]
\[
-0.4879 - 5.3757i
\]
\[
-0.0065 + 0.0422i
\]
\[
-0.0065 - 0.0422i
\]
\[
-0.1000 + 0.0000i
\]
Defining cost matrices on states and control as
\[
\text{cost} = \int_0^\infty [x'u']
\]
\[
\begin{bmatrix}
\text{QXX} & \text{QXU} \\
\text{QXU}' & \text{QUU}
\end{bmatrix}
\begin{bmatrix}
x \\
u
\end{bmatrix}
\text{dt},
\]
December 1982
the entire cost matrix $Q$ can be built up, starting with the outputs as

$$Q_{XX} = H^*H;$$

weighting the inputs nominally with the identity matrix by,

$$Q_{UU} = \text{EYE}(2);$$

achieving no cross-costs between states and the controls with,

$$Q_{XU} = O*\text{ONES}(5, 2);$$

$$Q = [Q_{XX}, Q_{XU}; Q_{XU}', Q_{UU}];$$

Now, entering RICC with no left hand side,

$$\text{ORICC}(S, Q, 5)$$
generates the closed-loop eigenvalues:

$$\text{ANS} = -1.7793 + 5.6435i$$

$$-1.7793 - 5.6435i$$

$$-0.2677 + 0.2119i$$

$$-0.2677 - 0.2119i$$

$$-0.0527 + 0.0000i$$

The closed-loop eigenvalues for exponentially decreasing cost on control are generated and stored column-wise in EV by entering

$$\text{FOR } JJ = 1:20,$$

$$\text{EIG}(F - KF*H)$$
generates frequency-response FRCL evaluated at frequency points in FR, ranging from 1E-2 to 1E3 rads/sec. Each row of FRCL corresponds to a particular frequency. The corresponding frequency-response matrix is stored row-wise from a single row of FRCL.

$$\text{OEV(:, 20)}$$

$$\text{ANS} = -86.2818 + 81.0192i$$

$$-86.2818 - 81.0192i$$

$$-5.5634 + 5.5558i$$

$$-5.5634 - 5.5558i$$

$$-0.0010 + 0.0000i$$

The system poles have shifted substantially with low cost on control. The transmission zero near the origin attracts one of the closed-loop poles.

$$\text{OEV(:, 20)}$$

$$\text{ANS} = -1.7793 + 5.6435i$$

$$-1.7793 - 5.6435i$$

$$-0.2677 + 0.2119i$$

$$-0.2677 - 0.2119i$$

$$-0.0527 + 0.0000i$$

The system is nonminimum phase with a zero at +0.0010. Such systems are known to be difficult to control.

The first set of eigenvalues are probably reasonable. To get the control gain
impulse in process noise (gust),

\[ \mathcal{O}(TS, \text{TRCM}) = \text{TIMR(SCOM 10, [60, 200])} \];
\[ \text{OPLOT}(TS, \text{TRCM(:, 1)}, 'YLAB*Y1*UPPER LEFT') \]
\[ \text{OPLOT}(TS, \text{TRCM(:, 2)}, 'YLAB*Y2*UPPER RIGHT KEEP') \]
\[ \text{OPLOT}(TS, \text{TRCM(:, 3)}, 'YLAB*Y3*LOWER LEFT KEEP') \]
\[ \text{OPLOT}(TS, \text{TRCM(:, 4)}, 'YLAB*Y4*LOWER RIGHT KEEP') \]
generates Fig. 7. The two top plots are impulse response of \( Y1 \) to process noise in the two channels respectively. The bottom plots are for \( Y2 \).

Meaningful extensions to LQG methods require inclusion of dynamics of reference, disturbances, sensors and actuators. Appending of dynamics in frequency-shaped control design or model-following techniques involves forming augmented equations, which is easily accomplished with \( \text{MATRIX}_X \) primitives. Use of frequency-shaped costs, with singular value plots for robustness evaluation, allows incorporation of engineering judgment in the control design.

More advanced application of \( \text{MATRIX}_X \) would use MACROS and command files so as to tailor control design procedures such as the session above. Advanced system theoretic procedures, such as computing Kronecker indices, suprema\( (A,B) \) invariant subspaces in the kernel of \( C \) and computing minimal realizations have been implemented in \( \text{MATRIX}_X \) as procedures. Such command files and MACROS use \( \text{MATRIX}_X \) primitives like the QZ, SVD, QR and PVA. For lack of space these advanced procedures will not be described here.

**Modeling, System Identification and Data Analysis**

**System Building Capability**

Graphical display and interactive specification of a complex system using simpler subsystems or components is often very useful. Simulation and identification of nonlinear systems require the system building capability as a basic tool.

A set of basic components allows the user to build a model by specifying the individual component characteristics and the topological structure. System building allows mixed continuous and discrete-time systems, multi-rate systems and a variety of nonlinear elements. As specification of each component is completed, the model can be stored or used in simulation and identification. An individual component can also be a previously specified model. A recently developed differential algebraic system solver (Gear and Petzold [2]) can handle systems that previously could not be simulated. The implicit equation solver can also be used to automatically find a consistent set of initial conditions—often not an easy task. Fixed-step-size Runge-Kutta can also be used if a sample hold is used that gives discontinuous inputs.

Phase-flow plots of the vector field defining a nonlinear differential equation can be valuable for looking at the behavior of nonlinear systems without accurately solving for the entire time history. One can demand to look at any two-dimensional plane of the phase-space at any point.

**Data Analysis and System Identification**

Data display, detrending, censoring and filtering can be accomplished easily with \( \text{MATRIX}_X \). Flexible graphics is a very important part of this package.

Some PLOT capabilities were illustrated previously. Coupled with the powerful matrix arithmetic operations and matrix functions, the graphical capabilities provide an excellent tool for data display.

Batch procedures include stepwise regression, state-space and nonlinear maximum likelihood procedures. Non-parametric batch and semi-batch methods using Fast Fourier Transform give auto- and cross-spectra and auto- and cross-correlations. Non-parametric methods use a modified version of "overlap add" method so that past data can be discounted exponentially. After each block of data is processed, the results can be examined and blocks of data added to the previous data without recomputing all the past results.

Recursive identification methods such as the Extended Kalman Filter for linear state-space-system with Ljung's modification [5] and recursive prediction error algorithm are available. There is a U-D measurement update as primitive operator. A variety of recursive identification and adaptive control algorithms can be implemented with U-D update.

**Numerical Algorithms**

A short description of some of the important algorithms in \( \text{MATRIX}_X \) follows.

**Generalized Eigenvalue Problem**

QZ primitive solves the generalized
Recursive Prediction Error Methods (RPEM) and U-D Update (UDUP)

RPEM allows identification of ARMAX (autoregressive moving average with exogenous inputs) models from data in a recursive manner. The algorithm uses Schur-Cohn stability check and projection of the parameters into the region of stability of the predictor. The covariance update is in the U-D factored form with regularization (Ljung [4]). UDUP performs rank-one update of covariance, as well as measurement update of the Kalman filter. It uses U-D update algorithms of Bierman [1]. Adaptive control algorithms. Extended Kalman Filter equations, and Maximum Likelihood identification algorithms can be implemented effectively with UDUP.

References

Software Summaries

This section contains twenty-three summaries on Computer-Aided Control System Design software packages as compiled by Prof. Dean K. Frederick. Prof. Frederick has established a file on CACSD packages, and he would like to receive information on new packages or receive updates whenever significant changes to those already on file occur. The CSM will publish a section on software summaries on a regular basis. If you would like to participate, please submit information on your software package, following the same format used in this issue, to: Prof. Dean K. Frederick, Electrical, Computer, and Systems Engineering Department, Rensselaer Polytechnic Institute, Troy, NY 12181.

Package or program name: EASY5
Dynamic Analysis and Design System

Principal developer: Dr. John D. Burnoughs

Software capabilities: Nonlinear simulation/time history generation, steady state analysis, optimal control synthesis, full state feedback/Kalman state estimator/reduced order controller, linear model generation, linear control system analysis, eigenvalues/stability margins/root locus/frequency response, linear simulation/time history generation, modular modeling language is used, continuous and discrete time models (single and multirate sampling), nonlinear simulations and linear control analysis in one integrated package, system of order 150 has been handled satisfactorily, selective analysis options may handle 500th order system

Interactive capabilities: Interactive batch (immediate batch execution)

Programming language used: FORTRAN

Computers and terminals on which available: MAINSTREAM-EKS (BCS Nationwide network), VAX 11/780, CYBER (Large Scale), IBM (Large Scale)

Documentation: EASY5 Users Guide, BCS Document #10208-127

Memory and disk requirements: 100K - 300K Octal 60 bit words, (Depends on size of model), on CDC, comparable on other hosts

State of development: Production code

Availability of code: Object Code may be leased or EASY5 may be accessed via MAINSTREAM-EKS

Person to contact for details: Ronald A. Hammond, 565 Andover Park West, M/S 9C-01, Tukwila, WA 98188; (206) 575-5092

Principal developer: Dr. P. J. Fleming, School of Electronic Engineering Science, University College of North Wales, Bangor, Gwynedd, U.K.

Software capabilities: Design of controllers for linear optimal regulators and suboptimal linear regulators for linear time-invariant continuous systems and sampled-data systems. State-space input specification. Typically will handle 15-20 varying gain parameters and overall system sizes of at least order 30 satisfactorily. Numerical and graphical evaluation tools for controller designs including comprehensive graphics display facilities (GHOST)

Interactive capabilities: Question and answer: help commands

Programming language used: FORTRAN-10 (easily modified for other versions of FORTRAN)

Computers and terminals on which available: Any computer which supports FORTRAN, and for graphics version, any terminal which GHOST graphics package supports

Documentation: User’s Manual, implementation documentation, commented source

Memory and disk requirements: Memory requirement is dependent on user specification. Typically an overall system of order 30 will require 190 K bytes of executable code (excluding graphics)

State of development: Complete version for continuous-time systems currently available. Version which includes sampled-data systems due