Some MACSYMA Functions for Analysis of Multivariable Linear Systems

Holmberg, Ulf

1986

Document Version:
Publisher's PDF, also known as Version of record

CODEN: LUTFD2/(TFRT-7333)/1–040/(1986)

Some MACSYMA Functions for Analysis of Multivariable Linear Systems

Ulf Holmberg

Department of Automatic Control
Lund Institute of Technology
October 1986

Department of Automatic Control
Lund Institute of Technology
P.O. Box 118
S-221 00 Lund Sweden

Document name
Report

Date of issue
October 1986

Document Number
CODEN: LUTFD2/(TFRT-7333)/1–040/(1986)

Author(s)
Ulf Holmberg

Supervisor

Sponsoring organisation
The National Swedish Board of Technical Development (STU contract 85-4809)

Title and subtitle
Some Macsyma Functions for Analysis of Multivariable Linear Systems

Abstract

Macsyma is an interactive program for symbolic manipulations written in LISP. In Macsyma it is possible to create new user defined functions. This facility is used to create a package in Macsyma for analysis of multivariable linear systems described by polynomial matrices. The intention is that the package will be used in courses in automatic control.

The report gives a short description of 27 functions that can be used together with the standard Macsyma functions.

Key words
Multivariable systems, Polynomial matrices, Symbolic Manipulations, Macsyma.

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title

ISBN

Language
English

Number of pages
40

Recipient's notes

Security classification

1. INTRODUCTION

The framework of polynomial matrices is useful for analysis of multivariable linear systems, see [Kailath]. However, polynomial matrices are not easily manipulated by hand. It is thus very important that good analysis tools for polynomial matrices are available. This report attempts to fill the gap between theory and practice by illustrating the use of Macsyma as a powerful tool for manipulating polynomial matrices.

In Section 2 a list of the user defined functions is given. The structure of these functions is then described in Section 3. This structure makes is possible to display an information text, describing the function and its syntax. This information text is given in Section 4 for all the user defined functions. All these functions could be collected in a special file with file addresses for all the functions. Once this file is loaded into Macsyma a user defined function, not previously defined, will be automatically loaded into Macsyma when it is called. This "login file" for this user defined analysis package is given in Section 5. Three examples will show how to use these functions. The third more elaborate example is a Macsyma demo with matrix fraction decompositions, coprime factorizations, multivariable realizations etc. This will illustrate the beauty of symbolic manipulations. Finally, it is described how a representation of a multivariable linear system in Macsyma can be transfered to a text file. This text file could then be loaded into CTRL-C or filtered through a program that generates simulation code for the simulation language Simnon (see [Mårtensson (1986a)]. For a more detailed discussion about ideas how to transfer results between different packages like Macsyma, CTRL-C, Simnon and interface to automatic documentation, TEX and PostScript, see [Holmberg, Lilja, Mårtensson].

2. SUMMARY OF FUNCTIONS

The following functions are described:

Linearization

`LINEARIZE`	Linearizes the dynamical system $\dot{x} = f(x,u), \quad y = g(x,u)$

Stability analysis

`ROUTH`	Generates the stability conditions for a continuous time system
`JURY`	Gives the stability conditions of discrete time systems and the steady state output variance

Sampling

`SAMP`	Sampling from transfer function to pulse-transfer function
`SAMPSTATE`	Sampling from state space to state space

Geometry functions — state space

`HERMITE`	Gaussian elimination when applied to a constant matrix
`KER`	Computes the Kernel $\{X \mid AX = 0\}$
`INVERSE_IMAGE`	Calculates the inverse image $\{X \mid AX = B\}$ (A possibly singular)
`INTERSECTION`	Computes the intersection of two subspaces
`GRAM_SCHMIDT`	Calculates an orthogonal base for a subspace
`AINV`	Computes the maximal A-invariant subspace in a given subspace
`ABINV`	Computes the maximal (A,B)-invariant subspace in a given subspace

Factorization — Frequency domain

SMITH	Calculates the Smith form together with transformation matrices
SMITH_MCMILLAN	Calculates the Smith-McMillan form with transformation
HERMITE	Calculates the Hermite form
COLUMNREDUCE	Makes a denominator polynomial matrix column reduced
ROWREDUCE	Makes a denominator polynomial matrix row reduced
RMFD	Right Matrix Fraction Decomposition (MFD) of a transfer matrix
LMFD	Left MFD of a transfer matrix
RIGHTCOPRIME	Gives a right coprime MFD from a noncoprime MFD
LEFTCOPRIME	Gives a left coprime MFD from a noncoprime MFD
SS2TF	State space to transfer function conversion
MAKESYS	Makes a list of the A, B, C, D matrices to represent a system

Multivariable Realizations

CONTROLLER	Calculates a controller form realization
OBSERVER	Calculates an observer form realization
CONTROLLABILITY	Calculates a controllability form realization
OBSERVABILITY	Calculates an observability form realization

3. THE HELP FUNCTION

Syntax and information of the standard Macsyma functions are displayed by the command `DISPLAY(function)`. It is possible to get a similar help facility for user defined functions. It could be done in the following way.

```
Function_name(arg1,[arglist]):=Block([local variables],

if arg1=help then return((
  print("      Syntax and Information       "),
  print("             of                    "),
  print("   the function Function-name      ")),

/* Pick out the rest of the arguments from arglist */
if length(arglist)=0 then error("Wrong arguments"),
if length(arglist)=1 then arg2:part(arglist,1),
if length(arglist)=2 then (arg2:part(arglist,1),
                           arg3:part(arglist,2)),
.
.
Body)
```

If a function has the structure above we receive the help information text when the function is called with the argument `help`.

```
Function_name(help)
```

Perhaps it is preferable to write `help(function_name)` instead of the reverse `Function_name(help)`. In that case the function `help` can be defined:

$$\texttt{HELP(function)} := \texttt{function(help)}$$

To have the help of a function included in the function definition as above is regarded to be a good idea. If the function is modified or extended you can modify the information text appropriately at the same time. The help information in the functions is presented in Section 4.

4. SUMMARY OF FUNCTION CALLS

A brief description of the available user defined functions is now given.

Linearization

`LINEARIZE(f,g,x,x0,u,u0)`

Linearizes the nonlinear system

$$\begin{cases} \dfrac{dx}{dt} = f(x,u) \\ y = g(x,u) \end{cases} \quad \rightarrow \quad \begin{cases} \dfrac{dx}{dt} = Ax + Bu \\ y = Cx + Du \end{cases}$$

around $x = x_0$ and $u = u_0$. It is assumed that $f(x_0, u_0) = g(x_0, u_0) = 0$. The arguments may be scalars as well as vectors. The answer will have the form $[A = \cdots, B = \cdots, C = \cdots, D = \cdots]$

Stability analysis

`ROUTH(apol)`

Generates the stability conditions for a continuous time system using Routh's algorithm. The argument `apol` is the characteristic polynomial and the given answer is a list. The number of sign-shifts in the list is equal to the number of roots in the right half plane.

`JURY(apol,bpol)`

Gives the stability conditions for a discrete time system and the steady state output variance, see [Åström-Wittenmark]. The arguments `apol` and `bpol` correspond to the $A-$ and B-polynomial in the pulse-transfer function $G(z) = \frac{B(z)}{A(z)}$. If none of the a_0-coefficients is zero in Jury's scheme then the number of negative a_0-coefficients is equal to the number of roots outside the unit disc. These a_0-coefficients are given as a list. Also, the steady state output variance I is given in the answer (when the input is white noise). [$a_0$coeff = [..], I= ..].

Sampling

`SAMP(G,h)`

Zero order hold sampling of the scalar transfer function G(s). The sampling function uses the residue formula

$$H(z;h) = \sum_{s=s_i} \text{Res}\{\frac{z-1}{z-e^{sh}} \cdot \frac{G(s)}{s}\}$$

where the transform variables are s and z and the sampling period is h. Other choices of transform variables can be chosen as shown below.

`Samp(G,h)`	→	`H(z;h)`
`Samp(G,q,h)`	→	`H(q;h)`
`Samp(G,[p,q],h)`	→	`H(q;h)`

In the last case both transform variables are defined, i.e. `G(p)` → `H(q;h)`

`SAMPSTATE(A,B,h) or SAMPSTATE(A,B,`τ`,h)`

samples a continuous time state space description $\dot{x}(t) = Ax(t) + Bu(t-\tau)$ without or with a time delay, τ. The result is $[\Phi = \cdots, \Gamma = \cdots]$ and $[\Phi = \cdots, \Gamma_0 = \cdots, \Gamma_1 = \cdots, d = \cdots]$ respectively. The latter describe the discrete time system:

$$x(kh+h) = \Phi x(kh) + \Gamma_0 u(kh - dh + h) + \Gamma_1 u(kh - dh)$$

The sampling interval, `h`, is always the last argument.

Geometry functions — State space

Some functions working on subspaces will now be described. We represent a subspace by the column span of a matrix.

`KER(A)`

Computes the Kernel of the matrix A; $\{X|AX = 0\}$. The resulting matrix X defines a base for $\ker(A)$.

`INVERS_IMAGE(A,B)`

Calculates the inverse image of the mapping A, i.e. solves for X in the equation system $AX = B$. If A has full rank the inverse image is simply a matrix inversion. However, when A is singular a solution, if there exist any, is constructed by a partial and a homogenous solution. When B is a vector the answer has the form

$$X = [X_p, X_p + X_h^{(1)}, X_p + X_h^{(2)}, ...X_p + X_h^{(n-r)}]$$

where $n - r$ is the dimension of $\ker(A)$. When B is a matrix the answer is $X = [X_p, X_h]$.

`INTERSECTION(U,V)`

Computes the intersection between two subspaces U and V (matrices).

`GRAM_SCHMIDT(V)`

Calculates an orthogonal base from a given matrix V.

`AINV(A,V)`

Computes the maximal A-invariant subspace in V. The arguments A and V must be defined as matrices. The answer is also a matrix where the columns are spanning the resulting subspace.

`ABINV(A,B,V)`

Computes the maximal (A, B)-invariant subspace in V. The type of the arguments and the answer is as in `Ainv` above.

Factorization — Frequency domain

The following functions are manipulating polynomial matrices. For terminology and a background the reader is referred to [Kailath], especially chapter 6.

`SMITH(A)`

Returns the Smith form of a polynomial matrix, A, in the following way:
`SMITH(A)`
→ [Smith form in s]
`SMITH(A,`λ`)`
→ [Smith form in λ]
`SMITH(A,transforms)`
→ [Smith form in s, row- and , column operations]
`SMITH(A,`λ`,transforms)`
→ [Smith form in λ, row- and , column operations]

`SMITH_MCMILLAN(G)`

Returns the Smith-McMillan form of a polynomial matrix G in the following way:
`SMITH_MCMILLAN(G)`
→ [Smith-McMillan form in s, McMillan degree]
`SMITH_MCMILLAN(G,`λ`)`
→ [Smith-McMillan form in λ, McMillan degree]
`SMITH_MCMILLAN(G,transforms)`
→ [Smith-McMillan form in s, row-, column operations]
`SMITH_MCMILLAN(G,`λ`,transforms)`
→ [Smith-McMillan form in λ, row- and , column operations]

`HERMITE(A)`

Returns the Hermite form of a polynomial matrix, A, in the following way:
`HERMITE(A)`
→ [Hermite form in s]
`HERMITE(A,`λ`)`
→ [Hermite form in λ]
`HERMITE(A,transform)`
→ [Hermite form in s, row operations]
`HERMITE(A,`λ`,transform)`
→ [Hermite form in λ, row operations]

`COLUMNREDUCE(D,N)`

performs elementary column operations such that the polynomial matrix $D(s)$ becomes column reduced (column proper). By default D is a polynomial matrix in s. In other cases the variable must be given as a second argument. Ex.: `COLUMNREDUCE(D,N,`λ`)`

`ROWREDUCE(D,N)`

performs elementary row operations such that the polynomial matrix $D(s)$ becomes row reduced (row proper). By default D is a polynomial matrix in s. In other cases the variable must be given as a second argument. Ex.: `ROWREDUCE(D,N,`λ`)`

`RMFD(G)`

Makes a right matrix fraction decomposition, $N_r(D_r)^{-1}$, of a rational transfer function matrix, $G(s)$. The extraction of the denominator polynomial matrix is made by placing the least common multiple of each column denominators in the diagonal. Note that this doesn't always lead to an irreducible MFD. By default G is a rational matrix in s. In other cases the variable must be specified as a second argument. Ex.: `RMFD(G,`λ`)`. The answer has the form $[D_r = \cdots, N_r = \cdots]$.

`LMFD(G)`

As `RMFD(G)` but makes instead a left MFD, $G(s) = (D_l(s))^{-1}N_l(s)$. The answer has the form $[D_l = \cdots, N_l = \cdots]$.

`RIGHTCOPRIME(D,N)`

Makes an irreducible right MFD out of a reducible right MFD. The answer also contains the extracted polynomial matrix factor, R_r, $[D_r = \cdots, N_r = \cdots, R_r = \cdots]$

`LEFTCOPRIME(D,N)`

As `RIGHTCOPRIME(D,N)` but makes instead a left irreducible MFD. $[D_l = \cdots, N_l = \cdots, R_l = \cdots]$

`SS2TF(SYS)`

State space to transfer function conversion. The argument `SYS` has the form $[A = \cdots, B = \cdots, C = \cdots]$ or $[A = \cdots, B = \cdots, C = \cdots, D = \cdots]$

`MAKESYS(A,B,C,D) or MAKESYS(A,B,C)`

makes a list of A, B, C (and D) matrices representing a system. $[A = \cdots, B = \cdots, C = \cdots, D = \cdots]$

Multivariable Realizations

`CONTROLLER(D,N)`

Makes a controller form realization of a right coprime MFD. Note that the denominator matrix D must be column-reduced. The answer has the form $[A = \cdots, B = \cdots, C = \cdots]$

`OBSERVER(D,N)`

Makes an observer form realization of a left coprime MFD. Note that the denominator matrix D must be row-reduced. The answer has the form $[A = \cdots, B = \cdots, C = \cdots]$

CONTROLLABILITY(D,N)

Similar to CONTROLLER(D,N) but yields a controllability form realization

OBSERVABILITY(D,N)

Similar to OBSERVER(D,N) but yields an observability form realization

5. USER DEFINED PACKAGE IN MACSYMA

It was shown in Section 3 how to write a user defined function. Using variable argument lists, a help-string could be displayed, describing the function and the syntax. It is a good habit to write this help information at the same time the rest of the code is written.

A large function is usually built up of smaller sub-functions. One example is the HERMITE-function which basically uses three sub-functions: row permutations, multiplication of a row by a scalar, and Euclides division algorithm. Other large similar functions may use the same sub-functions. In this case the SMITH-function uses some of the subfunctions from HERMITE. This may cause a problem if you don't know what files that had to be loaded before you can use a certain function.

If we use SETUP_AUTOLOAD we don't need to know if a function is calling another function or not. When an undefined function is called, Macsyma can get the file address for it in SETUP_AUTOLOAD. It will when be automatically loaded. Using this facility we can construct our own login file for this analysis package. This file, which we call LOGIN.MAC, is listed below.

```
/* ------------------------------------------------------------
THIS IS A KIND OF LOGIN FILE FOR MACSYMA.
INSTEAD OF LOADING AND BATCHING ALL FILES YOU THINK YOU NEED,
YOU JUST DO ONCE : LOAD("LOGIN.MAC").
THE FILES WILL THEN BE AUTOMATICALLY LOADED IF AND WHEN THEY ARE CALLED
------------------------------------------------------------ */
loadprint:false$

/* Linearization */
setup_autoload("/u/ulfh/packages/macsyma/linearize.mac",linearize)$

/* Stability analysis */
setup_autoload("/u/ulfh/packages/macsyma/routh.mac",routh)$
setup_autoload("/u/ulfh/packages/macsyma/jury.mac",jury)$

/* Sampling */
setup_autoload("/u/ulfh/packages/macsyma/samp.mac",samp)$
setup_autoload("/u/ulfh/packages/macsyma/sampstate.mac",sampstate)$

/* Geometry functions - Time domain */
```

```
/* (Gauss elimination is made by the general "Hermite" program) */
setup_autoload("/u/ulfh/packages/macsyma/geometry.mac",ker)$
setup_autoload("/u/ulfh/packages/macsyma/invimage.mac",Inverse_image)$
setup_autoload("/u/ulfh/packages/macsyma/geometry.mac",Intersection)$
setup_autoload("/u/ulfh/packages/macsyma/geometry.mac",Gram_Schmidt)$
setup_autoload("/u/ulfh/packages/macsyma/geometry.mac",ainv)$
setup_autoload("/u/ulfh/packages/macsyma/geometry.mac",abinv)$


/* Polynomial Matrices - Frequency domain */
setup_autoload("/u/ulfh/packages/macsyma/smith.mac",Smith)$
setup_autoload("/u/ulfh/packages/macsyma/smith.mac",Smith_McMillan)$
setup_autoload("/u/ulfh/packages/macsyma/hermite.mac",hermite)$
setup_autoload("/u/ulfh/packages/macsyma/hermite.mac",lcm)$
setup_autoload("/u/ulfh/packages/macsyma/reduce.mac",columnreduce)$
setup_autoload("/u/ulfh/packages/macsyma/reduce.mac",rowreduce)$
  /* Matrix Fraction Decompositions */
setup_autoload("/u/ulfh/packages/macsyma/mfd.mac",rmfd)$
setup_autoload("/u/ulfh/packages/macsyma/mfd.mac",lmfd)$
setup_autoload("/u/ulfh/packages/macsyma/mfd.mac",rightcoprime)$
setup_autoload("/u/ulfh/packages/macsyma/mfd.mac",leftcoprime)$
setup_autoload("/u/ulfh/packages/macsyma/mfd.mac",ss2tf)$
setup_autoload("/u/ulfh/packages/macsyma/mfd.mac",makesys)$
  /* Multivariable realizations */
setup_autoload("/u/ulfh/packages/macsyma/realizations.mac",controller)$
setup_autoload("/u/ulfh/packages/macsyma/realizations.mac",observer)$
setup_autoload("/u/ulfh/packages/macsyma/realizations.mac",controllability)$
setup_autoload("/u/ulfh/packages/macsyma/realizations.mac",observability)$
setup_autoload("/u/ulfh/packages/macsyma/realizations.mac",controlindices)$
setup_autoload("/u/ulfh/packages/macsyma/realizations.mac",Dhcmatrix)$

print_true:false$

help(x):=(x(help))$
```

---

6. EXAMPLES

The examples in this Section has been run with the typeset switch set to true and the resulting Troff/EQN code translated to TEX by the program MacEQ2TEX.

EXAMPLE 1.

Linearize the following nonlinear system

$$\begin{cases} \dot{x}_1 = x_1^2 - u_1 x_2 \\ \dot{x}_2 = u_1 x_2^2 + u_2 \end{cases}$$

$$\begin{cases} y_1 = x_1^2 + u_2 x_1 \\ y_2 = x_2^3 - u_1 \end{cases}$$

around $\{x_0 = (1 \quad 1)^T, u_0 = (1 \quad -1)^T\}$ and determine the poles and zeros of the resulting multivariable linear system.

```
(c2) load("login.mac")$
(c3) f:matrix([x1*x1-x2*u1],[x2*x2*u1+u2]);
```

(d3)
$$\begin{bmatrix} x_1^2 - u_1 x_2 \\ u_1 x_2^2 + u_2 \end{bmatrix}$$

```
(c4) g:matrix([x1*x1+x1*u2],[x2^3-u1]);
```

(d4)
$$\begin{bmatrix} x_1^2 + u_2 x_1 \\ x_2^3 - u_1 \end{bmatrix}$$

```
(c5) x:matrix([x1],[x2])$
(c6) x0:matrix([1],[1])$
(c7) u:matrix([u1],[u2])$
(c8) u0:matrix([1],[-1])$
(c9) linearize(f,g,x,x0,u,u0);
```

(d9)
$$\left[a = \begin{bmatrix} 2 & -1 \\ 0 & 2 \end{bmatrix}, b = \begin{bmatrix} -1 & 0 \\ 1 & 1 \end{bmatrix}, c = \begin{bmatrix} 1 & 0 \\ 0 & 3 \end{bmatrix}, d = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}\right]$$

```
(c10) ss2tf(%);
```

(d10)
$$\begin{bmatrix} -\frac{s-1}{(s-2)^2} & \frac{(s-3)(s-1)}{(s-2)^2} \\ -\frac{s-5}{s-2} & \frac{3}{s-2} \end{bmatrix}$$

```
(c11) smith_mcmillan(%);
```

(d11)
$$\left[smith - mcmillan = \begin{bmatrix} \frac{1}{(s-2)^2} & 0 \\ 0 & (s-6)(s-1) \end{bmatrix}, mcmillan\_degree = 2\right]$$

EXAMPLE 2.

Sample the following time-delayed system with a sampling period h and determine how the delay influences the discrete time system.

$$\dot{x}(t) = \begin{pmatrix} 0 & 1 \\ 0 & -1 \end{pmatrix} x(t) + \begin{pmatrix} 0 \\ 1 \end{pmatrix} u(t-\tau) \qquad \text{where} \quad \tau = ah \quad \text{and} \quad 1 < a < 2$$

Note line (c4) where we specify the variations of the parameter a.

```
(c1) load("login.mac");
```

(d1) *login.mac*

```
(c2) aa:matrix([0,1],[0,-1]);
```

(d2) $$\begin{bmatrix} 0 & 1 \\ 0 & -1 \end{bmatrix}$$

```
(c3) bb:matrix([0],[1]);
```

(d3) $$\begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

```
(c4) assume(a > 1 AND a < 2);
```

(d4) $[a > 1, a < 2]$

```
(c5) sampstate(aa,bb,a*h,h);
```

$$x(kh+h) = \Phi x(kh) + \Gamma_0 u(kh-dh+h) + \Gamma_1 u(kh-dh)$$

(d5)
$$\left[\Phi = \begin{bmatrix} 1 & 1-e^{-h} \\ 0 & e^{-h} \end{bmatrix}, \Gamma_0 = \begin{bmatrix} \left((2h-ah)\,e^{2h-ah} - e^{2h-ah} + 1\right) e^{ah-2h} \\ \left(e^{2h-ah} - 1\right) e^{ah-2h} \end{bmatrix},\right.$$
$$\left.\Gamma_1 = \begin{bmatrix} e^{h-ah}\left((ah-h)\,e^{ah-h} - e^{ah-h} + 1\right) + e^{h-ah}\left(1 - e^{ah-2h}\right)\left(e^{ah-h} - 1\right) \\ e^{-h}\left(e^{ah-h} - 1\right) \end{bmatrix}, d = 2\right]$$

EXAMPLE — Polynomial Matrix Manipulations

The following example is a Macsyma *Demo* that illustrate the use of polynomial matrices for analysis of multivariable linear systems. The cumbersome manipulations are done by the factorization functions in Section 4. The Demo starts with a transfer function matrix, describing a multivariable linear system. The description is transfered into a matrix fraction decomposition, MFD, i.e. a polynomial matrix description. Extraction of different polynomial matrix factors of the MFD are made. Also, different multivariable realizations are presented. For terminology and a background the reader is referred to [Kailath], especially chapter 6.

```
(c1) load("login.mac")$
(c2) demo("realizations.dem");
/* This demo describes a couple of examples in Kailath, chapter 6.
Example 6.2-1. Alternative MFDs for a Tranfer Function. p. 368-9.
Example 6.4-1. Controller-Form Realization of a Right MFD. p. 407-8.
Example 6.4-2. Observer-Form Realization of a Left MFD. p. 416
Example 6.4-6. Constructing Canonical Controllability Forms. p. 433-4. */
(c3) g:matrix([s/((s+1)*(s+2))^2,s/(s+2)^2],[-s/(s+2)^2,-s/(s+2)^2]);
```

(d3)
$$\begin{bmatrix} \frac{s}{(s+1)^2(s+2)^2} & \frac{s}{(s+2)^2} \\ -\frac{s}{(s+2)^2} & -\frac{s}{(s+2)^2} \end{bmatrix}$$

```
/* Example 6.2-1. Alternative MFDs for a Tranfer Function. p. 368-9. */
(c4) rmfd(g);
```

(d4)
$$\left[dr = \begin{bmatrix} (s+1)^2(s+2)^2 & 0 \\ 0 & (s+2)^2 \end{bmatrix}, nr = \begin{bmatrix} s & s \\ -s(s+1)^2 & -s \end{bmatrix}\right]$$

```
(c5) ev(rightcoprime(dr,nr),%);
```

(d5)
$$\left[dr = \begin{bmatrix} (s+1)^2(s+2)^2 & -(s+1)^2(s+2) \\ 0 & s+2 \end{bmatrix}, nr = \begin{bmatrix} s & 0 \\ -s(s+1)^2 & s^2 \end{bmatrix}, rr = \begin{bmatrix} 1 & 1 \\ 0 & s+2 \end{bmatrix}\right]$$

```
(c6) ev(columnreduce(dr,nr),%);
```

(d6)
$$\left[dr = \begin{bmatrix} 0 & -(s+1)^2(s+2) \\ (s+2)^2 & s+2 \end{bmatrix}, nr = \begin{bmatrix} s & 0 \\ -s & s^2 \end{bmatrix}, u = \begin{bmatrix} 1 & 0 \\ s+2 & 1 \end{bmatrix}\right]$$

```
/* Example 6.4-1. Controller-Form Realization of a Right MFD. p. 407-8. */
(c7) ev(real:controller(dr,nr),%);
```

(d7)
$$\left[a = \begin{bmatrix} -4 & -4 & 0 & -1 & -2 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -4 & -5 & -2 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}, b = \begin{bmatrix} 0 & 1 \\ 0 & 0 \\ -1 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}, c = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 \end{bmatrix}\right]$$

```
/* Example 6.4-6. Constructing Canonical Controllability Forms. p. 433-4.
*/
/* Search by Crate 2 */
(c8) ev(controllability(a,b,c),%);
```

$$(d8)\quad \left[a=\begin{bmatrix}0&0&-2&0&0\\1&0&-5&0&0\\0&1&-4&0&0\\0&0&2&0&-4\\0&0&1&1&-4\end{bmatrix},b=\begin{bmatrix}1&0\\0&0\\0&0\\0&1\\0&0\end{bmatrix},c=\begin{bmatrix}0&0&1&1&-4\\-1&4&-12&-1&4\end{bmatrix}\right]$$

```
/* Search by Crate 1 */
(c9) crate_nr:1;
```

(d9) 1

```
(c10) ev(controllability(a,b,c),real);
```

$$(d10)\quad \left[a=\begin{bmatrix}0&0&0&-4&2\\1&0&0&-12&5\\0&1&0&-13&4\\0&0&1&-6&1\\0&0&0&0&-2\end{bmatrix},b=\begin{bmatrix}1&0\\0&0\\0&0\\0&0\\0&1\end{bmatrix},c=\begin{bmatrix}0&0&1&-6&1\\-1&4&-12&32&-1\end{bmatrix}\right]$$

```
/* Example 6.4-2. Observer-Form Realization of a Left MFD. p. 416 */
(c11) lmfd(g);
```

$$(d11)\quad \left[dl=\begin{bmatrix}(s+1)^2(s+2)^2&0\\0&(s+2)^2\end{bmatrix},nl=\begin{bmatrix}s&s(s+1)^2\\-s&-s\end{bmatrix}\right]$$

```
(c12) ev(leftcoprime(dl,nl),%);
```

$$(d12)\quad \left[dl=\begin{bmatrix}(s+1)^2(s+2)^2&0\\(s+1)^2(s+2)&s+2\end{bmatrix},nl=\begin{bmatrix}s&s(s+1)^2\\0&s^2\end{bmatrix},rl=\begin{bmatrix}1&0\\-1&s+2\end{bmatrix}\right]$$

```
(c13) ev(rowreduce(dl,nl),%);
```

$$(d13)\quad \left[dl=\begin{bmatrix}0&-(s+2)^2\\(s+1)^2(s+2)&s+2\end{bmatrix},nl=\begin{bmatrix}s&s\\0&s^2\end{bmatrix},u=\begin{bmatrix}1&-(s+2)\\0&1\end{bmatrix}\right]$$

```
(c14) ev(observer(dl,nl),%);
```

(d14)
$$\left[a=\begin{bmatrix}-4 & 1 & 0 & 0 & 0\\ -4 & 0 & 0 & 0 & 0\\ 0 & 0 & -4 & 1 & 0\\ 1 & 0 & -5 & 0 & 1\\ 2 & 0 & -2 & 0 & 0\end{bmatrix},b=\begin{bmatrix}1 & 1\\ 0 & 0\\ 0 & 1\\ 0 & 0\\ 0 & 0\end{bmatrix},c=\begin{bmatrix}0 & 0 & 1 & 0 & 0\\ -1 & 0 & 0 & 0 & 0\end{bmatrix}\right]$$

7. GENERATION OF A MIMO SYSTEM TEXT FILE

It will now be demonstrated how a MIMO system in Macsyma can be transfered into a text file of a special form. The special form of the text file is chosen to be the same as the print format from CTRL-C. This makes it possible to load results form Macsyma into CTRL-C. It should also be mentioned that there is a Pascal program written by [Mårtensson (1986a)] that generates Simnon code from this text file representation. The generation of the text file from Macsyma is made by the function TOMIMO. This is a LISP program and consequently we have to enter the LISP mode before we apply it to our MIMO system.

```
(c1) load("login.mac")$
(c2) a:matrix([1,2],[3,4])$
(c3) b:matrix([5,6],[7,8])$
(c4) c:matrix([9,10],[11,12])$
(c5) d:matrix([13,14],[15,16])$
(c6) sys:makesys(a,b,c,d);
```

(d6)
$$\left[a=\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix},b=\begin{bmatrix}5 & 6\\ 7 & 8\end{bmatrix},c=\begin{bmatrix}9 & 10\\ 11 & 12\end{bmatrix},d=\begin{bmatrix}13 & 14\\ 15 & 16\end{bmatrix}\right]$$

```
(c7)
Break Entering Lisp:
<1>: (load "tomimo.l")
t
<1>: (tomimo $sys 'abcd.mim)
t
```

The MIMO system has now been written in the file `ABCD.mim`. This file looks as follows:

```
nmp =

2 2 2

a =
```

```
1 2
3 4

b =

5 6
7 8

c =

9 10
11 12

d =

13 14
15 16
```

---

8. ACKNOWLEDGEMENTS

The work has been supported by the Swedish Board of Technical Development in the project "Combination of formula manipulation and numerics" (STU-85-4809). A special thank is given to Dr. Bengt Mårtensson whose program MacEQ2TEX made it possible to "TEX-ify" the Macsyma log file.

References

ÅSTRÖM, K. J. and B. WITTENMARK (1984): *Computer Controlled Systems—Theory and Design*, Prentice-Hall Inc., Englewood Cliffs, New Jersey.

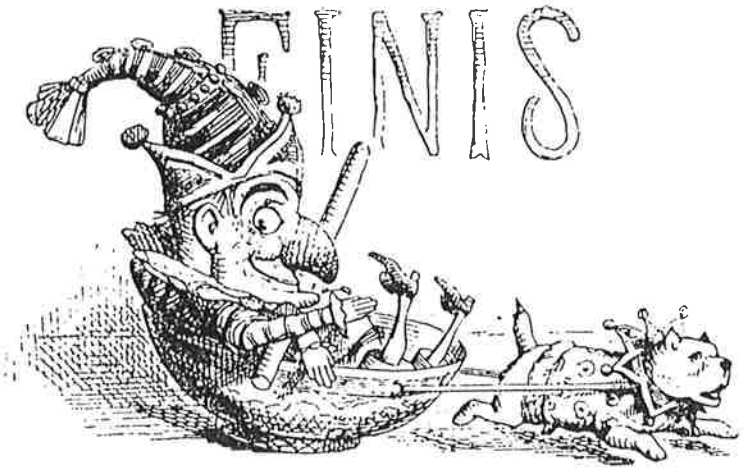
HOLMBERG, LILJA, MÅRTENSSON (1986):, "Integrating Different Symbolic and Numeric Tools for Linear Algebra and Linear System Analysis". To appear in proceedings of the SIAM Conference on *Linear Algebra in Signals, System and Control*. Aug. , Boston, USA..

KAILATH, THOMAS (1980): *Linear Systems*, Prentice-Hall Information and System Sciences Series., Englewood Cliffs, New Jersey.

MÅRTENSSON (1986b):, "CODEGEN—Automatic Simnon Code Generator for Multivariable Linear Systems", Report CODEN: LUTFD2/(TFRT-7323)/1-8/(1986), Dept. of Automatic Control, Lund Institute of Technology, Lund, Sweden..

MÅRTENSSON (1986c):, "MACEQ2TeX and S2TeX—Automatic TEX-code generation from Macsyma and CTRL-C", Dept. of Automatic Control, Lund Institute of Technology, Lund, Sweden..

THE MATHLAB GROUP LABORATORY FOR COMPUTER SCIENCE (1983): *MACSYMA Reference Manual*, M.I.T., Cambridge, MA..



APPENDIX: List of Functions

```
LINEARIZE(f,[y]):=Block([n,m,a,b,c,d,k,i,x,x0,u,u0],
if f=help then return((
print("Linearize(f,g,x,x0,u,u0) Linearizes the nonlinear system"),
print("      dx/dt = f(x,u)"),
print("      y     = g(x,u)"),
print("around the vectors x = x0 and u = u0"),
print("The arguments may be scalars as well as vectors"),
print("The answer will have have the form [A=..,B=..,C=..D=..]"),
"-------------------------------------------------------------")),

/* Matrixify the arguments */
g:part(y,1),
x:part(y,2),
x0:part(y,3),
u:part(y,4),
u0:part(y,5),
if not matrixp(x) then (x:matrix([x]),x0:matrix([x0])),
if not matrixp(u) then (u:matrix([u]),u0:matrix([u0])),
if not matrixp(f) then f:matrix([f]),
if not matrixp(g) then g:matrix([g]),
n:length(x),
m:length(u),
p:length(g),

/* A matrix */
a:zeromatrix(n,n),
for k:1 thru n do (for i:1 thru n do
a:setelmx(diff(f[k,1],x[i,1]),k,i,a)),

/* B matrix */
b:zeromatrix(n,m),
for k:1 thru n do (for i:1 thru m do
b:setelmx(diff(f[k,1],u[i,1]),k,i,b)),

/* C matrix */
c:zeromatrix(p,n),
for k:1 thru p do (for i:1 thru n do
c:setelmx(diff(g[k,1],x[i,1]),k,i,c)),

/* D matrix */
d:zeromatrix(n,m),
for k:1 thru p do (for i:1 thru m do
d:setelmx(diff(g[k,1],u[i,1]),k,i,d)),

for i:1 thru n do ( a:subst(x0[i,1],x[i,1],a),
                    b:subst(x0[i,1],x[i,1],b),
```

```
                    c:subst(x0[i,1],x[i,1],c),
                    d:subst(x0[i,1],x[i,1],d) ),
for i:1 thru m do ( a:factor(subst(u0[i,1],u[i,1],a)),
                    b:factor(subst(u0[i,1],u[i,1],b)),
                    c:factor(subst(u0[i,1],u[i,1],c)),
                    d:factor(subst(u0[i,1],u[i,1],d))),

if n>1 then ['a= a,'b= b,'c= c,'d= d]
       else ['a= a[1,1],'b= b[1,1],'c= c[1,1],'d=d[1,1]])$

Routh(apol):=Block([n,k,a,b,i,firstcol],

/* Routh's algorithm for the stability conditions
of a continuous time system */

n:hipow(apol,s),
a:makelist(coeff(apol,s,i),i,0,n),
a:reverse(a),
if evenp(n) then (k:n/2,
                  b:makelist(a[2*i],i,1,k),
                  b:append(b,[0]),
                  a:makelist(a[2*i+1],i,0,k))
           else (k:(n-1)/2,
                  b:makelist(a[2*i],i,1,k+1),
                  a:makelist(a[2*i+1],i,0,k)),
                  firstcol:append([first(a)],[first(b)]),
                  print("Number of signshift = Number of roots in the
RHP"),
                  Routh_aux(a,b))$

Routh_aux(a,b):=Block([n,i],
n:length(a),
if n=1 then return(firstcol),
a:makelist(a[i]-a[1]*b[i]/b[1],i,2,n),
firstcol:append(firstcol,[first(a)]),
if last(b)#0 then
    a:append(a,[0])
else
    b:reverse(rest(reverse(b))),
Routh_aux(b,a))$

Jury(apol,bpol):=Block([n,a,b,a0,a0coeff,i],

/* Stability conditions using Jury's scheme and
calculation of the steady state variance */

n:hipow(apol,z),
a:makelist(coeff(apol,z,i),i,0,n),
a0:last(a),
```

```
b:makelist(coeff(bpol,z,i),i,0,n),
a0coeff:[],
a0coeff:ev(a0coeff,Jury_aux(a,b)),
I:0,
I:ev(I,Jury_aux(a,b))/a0,
I:ratsimp(I,a0),
print("Stability condition : a0coeff>0"),
print("Steady State Variance :I"),
['a0coeff= a0coeff,'I= factor(I)])$

Jury_aux(a,b):=Block([ar,alfa,beta],
ar:reverse(a),
alfa:first(a)/first(ar),
beta:first(b)/first(ar),
I:I+first(b)*beta,
a:a-alfa*ar,
b:b-beta*ar,
a:rest(a),
b:rest(b),
if not member(first(ar),a0coeff) then a0coeff:append([first(ar)],a0coeff),
if a=[] then return(['a0coeff= a0coeff,'I= I])
        else Jury_aux(a,b))$

SAMP(g,[y]):=Block([sum,i,x,s,z],
if g=help then return((
print("SAMP(G,h) Zero order hold sampling of the scalar transfer
function"),
print("G(s). The sampling function uses the Residue formula "),
print('Sum('Res[s[i]],i,1,n),(z-1)/(z-exp(s*'h))*'G/s),
print("Examples:"),
print("          Samp(G,h)       ->    H(z\;h) "),
print("          Samp(G,q,h)     ->    H(q\;h) "),
print("          Samp(G,[p,q],h) ->    H(q\;h) "),
print("In the last example both transform variables are defined, i.e."),
print(" G(p) -> H(q\;h)"),
print("----------------------------------------------------------------"))),

/* Pick out arguments */
if length(y)=1 then h:part(y,1)
  else (if atom(part(y,1)) then z:part(y,1) else
(s:part(y,1,1),z:part(y,1,2)),
        h:part(y,2)),

/* Calculate poles */
x:g/s,
pol:transpose(solve(x^-1,s)),

/* Put the poles in a vector */
for i:1 thru length(pol) do pol:setelmx(part(pol,i,1,2),i,1,pol),
```

```
/* Calculate the sum of all residues of : */
x:((z-1)/(z-exp(s*h)))*x,

sum:0,
for i:1 thru length(pol) do sum:residue(x,s,pol[i,1])+sum,
sum:factor(sum))$

SAMPSTATE(A,[y]):=Block([B,n,Fi,Gamma,Gamma0,Gamma1,tau1,hh,t,d],
if A=help then return((
print("SAMPSTATE(A,B,h) or SAMPSTATE(A,B,tau,h) samples a continuous
time"),
print("state space description (A,B) without or with a time delay, tau."),
print("The result is [Phi=..,Gamma=..] and
[Phi=..,Gamma0=..,Gamma1=..,d=..]"),
print("respectively. The latter describe the discrete time system:"),
print("------------------------------------------------------ "),
print("x(k*h+h)=Phi*x(k*h)+Gamma0*u(k*h-d*h+h)+Gamma1*u(k*h-d*h)"),
print("------------------------------------------------------ "),
print("The sampling interval, h, is always the last argument"),
print("------------------------------------------------------------")))),
/* Pick out arguments */
B:part(y,1),
if length(y)=2 then
  (tau:0,
   h:part(y,2))
 else (tau:part(y,2),
       h:part(y,3)),

assume(h>0 and hh>0),
n:length(A),
Fi:ilt((s*ident(n)-A)^^-1,s,t),
Gamma:factor(integrate(Fi.B,t,0,hh)),
tau1:tau,
tau1:delay(tau1),
d:1+(tau-tau1)/h,
Gamma0:factor(ev(Gamma,hh=h-tau1)),
Gamma1:factor(ev(Fi,t=h-tau1).ev(Gamma,hh=tau1)),
Fi:factor(ev(Fi,t=h)),
if length(y)=2 then
       (print('x(k*h+h)='Phi*x(k*h)+'Gamma*u(k*h)),
        ['Phi= Fi,'Gamma= Gamma0])
  else
(print('x(k*h+h)='Phi*x(k*h)+'Gamma0*u(k*h-'d*h+h)+'Gamma1*u(k*h-'d*h)),
        ['Phi= Fi,'Gamma0= Gamma0,'Gamma1= Gamma1,'d= d]))$

delay(tau1):=(if tau1>h then (tau1:tau1-h,delay(tau1)) else tau1)$

/* Functions to analyse system geometry
```

```
------------------------------------ */

KER(x) := Block([n,m,v,i,j,lastpivot,k,pivot,np],
if x=help then return((
print("Ker(A) Computes the Kernel of the matrix A. {X:AX=0}"),
print("The resulting matrix X defines a base for Ker(A)"),
"-----------------------------------------------------------")),

x:part(Hermite(x),1,2),
n:length(transpose(x)),
m:length(x),
v:zeromatrix(n,1),

e(j):=col(ident(n),j),

newvector(x,i,k):= Block([vec,j],
vec:zeromatrix(n,1),
for j:1 thru i do vec:vec + x[j,k]*e(pivot[j,1]),
vec:e(k) - vec),

np:rank(x),
pivot:zeromatrix(np,1),
pivot[1,1]:findcol(x,1,1),
for i:1 thru pivot[1,1]-1 do v:addcol(v,e(i)),
for i:2 thru np do pivot[i,1]:findcol(x,i,pivot[i-1,1]+1),
for i:1 thru np-1 do
  (for k:pivot[i,1]+1 thru pivot[i+1,1]-1 do
     (v:addcol(v,newvector(x,i,k)))),
lastpivot:pivot[np,1],
for k:pivot[np,1]+1 thru n do
  (v:addcol(v,newvector(x,np,k))),
submatrix(v,1))$

/* ------------------------------------------------------ */

INTERSECTION(U,[V]):=Block([Ut,Vt],
if U=help then return((
print("Computes the intersection between two subspaces U and V
(matrices)"),
"-------------------------------------------------------------------------
")),
V:part(V,1),

Ut:transpose(U),
Vt:transpose(V),
ker(transpose(addcol(ker(Ut),ker(Vt)))))$

/* ------------------------------------------------------ */
```

```
GRAM_SCHMIDT(p):=Block([m,n,x,sum,i,projold,norm],
if p=help then return((
print("Gram_Schmidt(V) Calculates an orthogonal base from a given matrix
V"),
"-----------------------------------------------------------")),
m:length(p),
n:length(transpose(p)),
x:col(p,1),
for i:2 thru n do
 (sum:zeromatrix(m,1),
  for j:1 thru i-1 do
   (norm:transpose(col(x,j)).col(x,j),
    projold:transpose(col(p,i)).col(x,j)/norm,
    sum:sum+projold*col(x,j)),
  x:addcol(x,col(p,i)-sum)),
x)$

/* ---------------------------------------------------------- */

Ainv(A,[V]):=Block([U,sV],
if A=help then return((
print("Computes the maximal A-invariant subspace in V. The arguments"),
print("A and V must be defined as matrices. The answer is also a matrix"),
print("where the columns are spanning the resulting subspace.       "),
"-------------------------------------------------------------------")),
V:part(V,1),

U:inverse_image(A,V),
sV:Intersection(U,V),
if sV[1]=[] then return(sV),
if rank(sV)=rank(V) then V
  else Ainv(A,sV))$

/* ---------------------------------------------------------- */

ABinv(A,[arglist]):=Block([B,V,U,sV],
if A=help then return((
print("Computes the maximal (A,B)-invariant subspace in V."),
print("The arguments must be defined as matrices. The answer is also a
matrix"),
print("where the columns are spanning the resulting subspace.       "),
"-------------------------------------------------------------------")),
B:part(arglist,1),
V:part(arglist,2),

U:inverse_image(A,addcol(V,B)),
sV:Intersection(U,V),
if sV[1]=[] then return(sV),
if rank(sV)=rank(V) then V
```

```
  else ABinv(A,B,sV))$

INVERSE_IMAGE(Amatrix,[Bmatrix]):=Block(
[n,rank,List,LA,LB,LAR,R,B1,B2,A1,Xh,Xp,p,X1,X,L],

if Amatrix=help then return((
print("INVERS_IMAGE(A,B) calculates the inverse image of the mapping A,"),
print("i.e. solves for X in the equation system AX = B. If A has full  "),
print("rank the inverse image is simply a matrix inversion. However,   "),
print("when A is singular a solution, if there excist any, is
constructed"),
print("by a partial and a homogenous solution. The answer has the form "),
print("X = [ Xp , X + Xh(1), X + Xh(2), ... X + Xh(n-r)] when B is a
vector"),
print("and the form X= [ Xp , Xh ] when B is a matrix"),
"-------------------------------------------------------------------
")),

Bmatrix : part(Bmatrix,1),

/* Trivial Cases */
n:length(Amatrix),
rank:rank(Amatrix),
if rank=n then return((Amatrix^^-1).Bmatrix),
if rank=0 then return("AAAAAAH.... About your IQ?"),

/* The partial solution is calculated as follows.

AX = B

Gauss elimination:
                        [ 0 * ... ]
LAX = LB    where LA =  [     * ..]  Upper Staircase
                        [  0    0 ]
------------------------------------------------- */
List:Hermite(Amatrix,transform),
LA:ev('Hermite,List),
L:ev('UL,List),
LB:L.Bmatrix,
/* ------------------------------------------------ */

/* Pick out the base variables: X = RX
                                     1

                [ A  ]
such that LAR = [  1 ] , A  = Triangular and full rank = r
                [ 0  ]     1

                     [ B  ]
```

```
Partition LB:     LB = [  1 ]
                       [ B  ]
                          2

If B  = 0 then there is a solution to AX = B
    2
-------------------------------------------------- */
B2:LB,
for i:1 thru rank do B2:submatrix(1,B2),
if B2#zeromatrix(n-rank,length(transpose(LB))) then return("Empty Space"),
/* ------------------------------------------------------------------ */

/* Calculatation of LAR and R ------------ */
p:1,
LAR:zeromatrix(n,1),
R:zeromatrix(n,1),
for i:1 thru rank do
  (p:findcol(LA,i,p), /* find column#0 */
   LAR:addcol(LAR,col(LA,p)),
   R:addcol(R,col(ident(n),p))),
LAR:submatrix(LAR,1),
R:submatrix(R,1),

/* The partitioning A1 and B1 ------------ */
A1:LAR,
B1:LB,
for i:n step -1 thru rank+1 do
  (A1:submatrix(i,A1),
   B1:submatrix(i,B1)),

/* ----------------------------------------
Solve the quadratic system A X  = B
                            1 1    1
---------------------------------------- */
X1:(A1^^-1).B1,
/* ----------------------------------------
Then the partial solution is X  = RX
                              p     1
---------------------------------------- */
Xp:R.X1,

/* ----------------------------------------
The homogenous solution Xh = Ker(A) = Ker(LA)
---------------------------------------- */
Xh:ker(LA),

/* ----------------------------------------
Finally add the homogenous solution  X  = ker(A)
                                     h
```

```
X = [ X , X + X (1), X + X (2), ... X + X (n-r)]  or X = [X , X ]
      p   p   h      p   h          p   h                 p   h
---------------------------------------- */
X:Xp,
if matrixp(X) then return(addcol(X,Xh)),
for i:1 thru n-rank do X:addcol(X,Xp + col(Xh,i)),
X)$


SMITH(x,[y]):=Block([ly],
if x=help then return((
print("Returns the SMITH form of a polynomial matrix  X                "),
print("in the following way:                                           "),
print("   Smith(x)                                                     "),
print("     ->[Smith form in  s ]                                      "),
print("   Smith(x,lambda)                                              "),
print("     ->[Smith form in lambda]                                   "),
print("   Smith(x,transforms)                                          "),
print("     ->[Smith= Smith form in s,Ul= row- ,Ur= column operations] "),
print("   Smith(x,lambda,transforms)                                   "),
print("     ->[Smith= Smith form in lambda,Ul=..,Ur=..]                "),
" ------------------------------------------------------------------
")),
ly:length(y),
if ly=0 then return(Smith_detdividor(x)),
if ly=1 then
  (if part(y,1)=transforms
     then return(Smith_transforms(x))
     else return(Smith_detdividor(x,part(y,1)))),
if ly=2 then
  (if part(y,2)=transforms
     then return(Smith_transforms(x,part(y,1)))
     else print("The second argument should be TRANSFORMS")))$

SMITH_detdividor(x,[y]) := Block([r,m,n,i,j,size,detdividor,invpol,s],
if length(y)=1 then s:part(y,1),
r:rank(x),
m:length(x),
n:length(transpose(x)),

detminor(i,j,size) := Block([k,subdet],
subdet:x,
for k:1 thru i-1 do subdet:submatrix(1,subdet),
for k:i+size thru m do subdet:submatrix(size+1,subdet),
for k:1 thru j-1 do subdet:submatrix(subdet,1),
for k:j+size thru n do subdet:submatrix(subdet,size+1),
determinant(subdet)),

detdividor:zeromatrix(r,r),
```

```
for size:1 thru r do
  (for i:1 thru m-size+1 do
     (for j:1 thru n-size+1 do

(detdividor[size,1]:gcd(detdividor[size,1],detminor(i,j,size),s)))),
invpol:ident(r),
invpol[1,1]:detdividor[1,1],
for i:2 thru r do invpol[i,i]:detdividor[i,1]/detdividor[i-1,1],
if m>r then invpol:addrow(invpol,zeromatrix(m-r,r)),
if n>r then invpol:addcol(invpol,zeromatrix(m,n-r)),
factor(invpol))$

/* ------- SMITH FORM WITH TRANSFORMATIONS ----------------------- */

/* Smith form construction of a matrix */
SMITH_transforms(x,[y]):=Block([xlis,xper,ul,ur,ly,s],
   ly:length(y),
   if ly=1 then s:part(y,1),

   xlis : smithreduce(x),
   xper : permutediag(ev('x,xlis)),
   ul   : ev(vl,xper).ev(ul,xlis),
   ur   : ev(ur,xlis).ev(vr,xper),
   ['Smith= ev('x,xper),'UL= ul,'UR= ur])$

/* Row and column operations for Smith form construction of a matrix */
SMITHREDUCE(x):=Block([n,m,i,j,p,lp,pp,ready,r1,rr,c1,xr,ul,ur,xlis,xtmp],
  n:length(transpose(x)),
  m:length(x),
  ul:ident(m),
  ur:ident(n),

  ready:false,
  for j:1 while not ready do
    (x : ratexpand(x),
     pp: 1,
     lp: maxpower(x[1,1]),
     for p:1 thru n do
       for i:1 thru m do
         if lp>maxpower(x[i,p]) then (lp : maxpower(x[i,p]), pp : p ),
     if pp>1 then
       (x : addcol( col(x,pp)  , submatrix(x,pp)  ),
        ur: addcol( col(ur,pp) , submatrix(ur,pp) ))
     else if zerobelow(x,1,1) and zerobelow( transpose(x) ,1,1) then
       ready:true,

     if not ready then
       (if m>1 then
         (xtmp : reducebelow( addcol(x,ul) ,1,1),
```

```
          x: col(xtmp,1),
          for p:2 thru n do x : addcol(x,col(xtmp,p)),
          ul: col(xtmp,n+1),
          for p:2 thru m do ul: addcol(ul,col(xtmp,n+p))),
        if n>1 then
          (x : transpose(addrow(x,ur)),
           x : reducebelow(x,1,1),
           xtmp : transpose(x),
           x: row(xtmp,1),
           for i:2 thru m do x : addrow(x,row(xtmp,i)),
           ur: row(xtmp,m+1),
           for i:2 thru n do ur: addrow(ur,row(xtmp,m+i)))))),

  if print_true then print("x,ul,ur",x,ul,ur),

  if n>1 and m>1 then
    (r1  : row(x,1),
     rr  : submatrix(1,x),
     c1  : col(rr,1),
     xr  : submatrix(rr,1),
     xlis: smithreduce(xr),
     x   : addrow( r1 , addcol( c1 , ev(x,xlis) ) ),
     ul
:addrow(ematrix(1,m,1,1,1),addcol(zeromatrix(m-1,1),ev(ul,xlis))).ul,
     ur
:ur.addrow(ematrix(1,n,1,1,1),addcol(zeromatrix(n-1,1),ev(ur,xlis)))),

  ['X= x,'UL= ul,'UR= ur])$

maxpower(xii):=if xii#0 then hipow(xii,s) else 10^6$

/* Permutation of a matrix according to degree of diagonal element*/
PERMUTEDIAG(x):=Block([mn,m,n,i,j,lpow,xii,v,vl,vr,rj],
  m :length(x),
  n :length(transpose(x)),
  mn:min(m,n),
  x : ratexpand(x),
  vl: ident(mn),

  for i:1 thru mn do
    (lpow:10^8,
     for j:i+1 thru mn do
       if lpow>maxpower(x[j,j]) then (lpow : maxpower(x[j,j]), rj : j ),
     if lpow<maxpower(x[i,i]) then
       (xii:x[i,i],
        x:setelmx(x[rj,rj],i,i,x),
        x:setelmx(xii,rj,rj,x),
        v:ident(mn),
        v:setelmx(0,i,i,v),
```

```
        v:setelmx(0,rj,rj,v),
        v:setelmx(1,rj,i,v),
        v:setelmx(1,i,rj,v),
        vl:v.vl)),

  vr:transpose(vl),
  if m>n then
    (vl:addrow( addcol( vl                , zeromatrix(n,m-n) ) ,
                addcol( zeromatrix(m-n,n) , ident(m-n)        ) ) ),
  if n>m then
    (vr:addrow( addcol( vr                , zeromatrix(m,n-m) ) ,
                addcol( zeromatrix(n-m,m) , ident(n-m)        ) ) ),

  ['X= x,'VL= vl,'VR= vr])$

/* ------- SMITH-MCMILLAN FORM ------------------------------------ */

SMITH_MCMILLAN(x,[y]):=Block([ly,s],
if x=help then return((
print("Returns the SMITH-MCMILLAN form of a polynomial matrix  x       "),
print("in the following way:                                          "),
print("   Smith_McMillan(x)                                           "),
print("     ->[Smith_McMillan form in  s, McMillan degree]            "),
print("   Smith_McMillan(x,lambda)                                    "),
print("     ->[Smith-McMillan form in lambda, McMillan degree]        "),
print("   Smith_McMillan(x,transforms)                                "),
print("     ->[Smith-McMillan form in s,row- and ,column operations]  "),
print("   Smith_McMillan(x,lambda,transforms)                         "),
print("     ->[Smith-McMillan form in lambda,row- and ,column op.]    "),
" ----------------------------------------------------------------
")),
ly:length(y),
if ly=0 then return(Smith_McMillan_detdividor(x,s)),
if ly=1 then
  (if part(y,1)=transforms
     then return(Smith_McMillan_transforms(x,s))
     else return(Smith_McMillan_detdividor(x,part(y,1)))),
if ly=2 then
  (if part(y,2)=transforms
     then return(Smith_McMillan_transforms(x,part(y,1)))
     else print("The second argument should be TRANSFORMS")))$

SMITH_MCMILLAN_detdividor(H,var):=Block([m,p,i,j,den,tmp,s,sm],
  s   : var,
  p   : length(h),
  m   : length( transpose(h) ),
  den : 1,
  for i:1 thru m do
     for j:1 thru p do
```

```
        den:lcm(den,denom( h[j,i] ) , s ),
   sm  : factor(smith(factor(den*h))/den),
   deg : sum(hipow(expand(denom(sm[i,i])),s),i,1,max(p,m)),
   ['Smith-McMillan= sm,'McMillan_degree= deg])$

SMITH_MCMILLAN_transforms(H,var):=Block([m,p,i,j,den,tmp,sm,ul,ur,s,x],
   s   : var,
   p   : length(h),
   m   : length( transpose(h) ),
   den : 1,
   for i:1 thru m do
      for j:1 thru p do
        den:lcm(den,denom( h[j,i] ) , s ),
   tmp : smith_transforms(factor(den*h),s),
   sm  : factor(ev('x,tmp)/den),
   ul  : ev(ul,tmp),
   ur  : ev(ur,tmp),
   ['Smith-McMillan= sm,'UL= ul,'UR= ur])$

print_true:false$

Hermite(x,[y]):=Block([s,p,n,m,i,np,ly,UL,Hermite],
if x=help then return((

print("----- HERMITE FORM ------------------------------------------ "),
print("------------------------------------------------------------- "),
print("Returns the Hermite form of a polynomial matrix  x            "),
print("in the following way:                                         "),
print("   Hermite(x)                                                 "),
print("     ->[Hermite form in  s ]                                  "),
print("   Hermite(x,lambda)                                          "),
print("     ->[Hermite form in lambda]                               "),
print("   Hermite(x,transform)                                       "),
print("     ->[Hermite form in s,row operations]                     "),
print("   Hermite(x,lambda,transform)                                "),
print("     ->[Hermite form in lambda,row operations]                "),
" ------------------------------------------------------------------
")),
ly:length(y),
n:length(transpose(x)),
m:length(x),
np:rank(x),

if ly=2 then
  (if part(y,2)=transform
     then (lis:Hermite(addcol(x,ident(m)),part(y,1)),
           UL:ev('Hermite,lis),
           Hermite:UL,
           for i:1 thru n do UL:submatrix(UL,1),
```

```
            for i:n+m step -1 thru n+1 do Hermite:submatrix(Hermite,i),
            return(['Hermite= Hermite, 'UL= UL]))
      else print("The second argument should be TRANSFORM")),

if ly=1 then
  (if part(y,1)=transform
     then (lis:Hermite(addcol(x,ident(m))),
           UL:ev('Hermite,lis),
           Hermite:UL,
           for i:1 thru n do UL:submatrix(UL,1),
           for i:n+m step -1 thru n+1 do Hermite:submatrix(Hermite,i),
           return(['Hermite= Hermite, 'UL= UL]))
     else s:part(y,1)),

p:1,
for i:1 thru np do
  (p:findcol(x,i,p),
   x:reducebelow(x,i,p),
   if i#1 then x:reduceabove(x,i,p)),
['Hermite= x])$

findcol(x,i,p):=block([n],
n:length(transpose(x)),
if p>n then n,
if not zerobelow(x,i,p) then p else
(if x[i,p]#0 then p else findcol(x,i,p+1)))$

reducebelow(x,i,p):=block([down],
x:rowperm(x,i,p),       if print_true then print("rowperm below row",i,x),
x:monic(x,i,p),         if print_true then print("monic leadcoef row",i,x),
x:Euclid(x,i,p,down),   if print_true then print("Euclid div below
row",i,x),
if not zerobelow(x,i,p) then reducebelow(x,i,p) else x)$

reduceabove(x,i,p):=block([powi,pow,k,up],
x:Euclid(x,i,p,up),     if print_true then print("Euclid div above
row",i,x),
powi:hipow(part(x,i,p),s),
for k:i-1 step -1 thru 1 do
   (pow:hipow(part(x,k,p),s),
    pow:if x[k,p]=0 then -1 else pow,
    if not pow<powi then x:reduceabove(x,i,p)),
x:simplify(x,p))$

zerobelow(x,i,p):=Block([k,j],
k:col(x,p),
for j:1 thru i do k:if length(k)=1 then 0 else submatrix(1,k),
if k.k=0 then true else false)$
```

```
monic(x,i,p):=Block([f,e,pow],
f:leadcoef(x,i,p),
x:setelmx(1/f,i,i,ident(length(x))).x,
x:simplify(x,p))$

leadcoef(x,i,p):=Block([e,pow,res],
x:ratexpand(x),
e:part(x,i,p),
pow:hipow(e,s),
res:coeff(e,s,pow),
if res=0 then x[i,p] else res)$

rowperm(x,i,p):=Block([m,k,ei,e,powi,pow,L],
m:length(x),
x:ratexpand(x),
ei:part(x,i,p),
powi:if ei#0 then hipow(ei,s) else 10^6,
for k:i+1 thru m do
    (e:part(x,k,p),
     pow:if e#0 then hipow(e,s) else 10^6,
     if pow<powi then (L:setelmx(0,i,i,ident(m)),
                       L:setelmx(1,i,k,L),
                       L:setelmx(0,k,k,L),
                       L:setelmx(1,k,i,L),
                       x:L.x)),
x)$

Euclid(x,i,p,upordown):=Block([m,k,q,L,begin,end,pow,powi],
m:length(x),
if upordown=down then (begin:i+1,end:m)
                 else (begin:1,end:i-1),
powi:hipow(part(x,i,p),s),
for k:begin thru end do
    (pow:hipow(part(x,k,p),s),
     if pow>=powi then
         (q:leadcoef(x,k,p)*s^(pow-powi),
          L:setelmx(-q,k,i,ident(m)),
          x:L.x)),
x:ratexpand(x))$

simplify(x,p):=Block([i,m],
m:length(x),
x:ratsimp(x),
for i:1 thru m do x:setelmx(multthru(x[i,p]),i,p,x),
x)$

Ibar(k):=Block([x,i],
x:zeromatrix(k,k),
for i:1 thru k do x:setelmx(1,k-i+1,i,x),
```

```
x)$

lcm(p1,p2,s):=p1*p2/gcd(p1,p2,s)$

print_true:false$

perm_col(x,c):=Block([k,m,i,j,U],
/* Permutation of the columns in x such that
   the controlindices c rearranges in increasing order. */

m:length(c),
k:c.ident(m), /* make matrix out of list */
R:ident(m),
for i:1 thru m-1 do
  (for j:i+1 thru m do
    (if k[1,j]<k[1,i] then
       (U:ident(m),
        U:setelmx(1,j,i,U),
        U:setelmx(0,i,i,U),
        U:setelmx(1,i,j,U),
        U:setelmx(0,j,j,U),
        R:R.U,
        k:k.U))),
k:k[1], /* make list out of matrix */
['x= x.R,'k= k])$

COLUMNREDUCE(y,[p]):=Block([D,N,s,Dr,U],
if y=help then return((
print("COLUMNREDUCE(D,N) performs elementary column operations "),
print("such that the polynomial matrix D(s) becomes column   "),
print("reduced (column proper). By default D is a polynomial "),
print("matrix in s. In other cases the variable must be given"),
print("as a third argument. Ex.: COLUMNREDUCE(D,N,lambda)    "),
print("The answer is a list including the column proper D,   "),
print("the corresponding nominator matrix N in the MFD and   "),
print("the unimodular matrix, U, that made the transform     "),
print("[ Dr= .., Nr= .., U= ..] "),
"-------------------------------------------------------------")),

D:y,
N:part(p,1),
if length(p)=2 then s:part(p,2),

Dr:Columnreduce_iter(D,s),
U:factor((D^^-1).Dr),
['Dr= factor(Dr), 'Nr= factor(N.U), 'U= U])$

COLUMNREDUCE_iter(y,[p]):=Block([D,k,Dhc,m,r,n,e,f,U,perm,s,f_list,i,j],
if length(p)=1 then s:part(p,1),
```

```
D:expand(y),
n:length(D),
k:controlindices(D),
perm:perm_col(D,k),
D:part(perm,1,2),
k:part(perm,2,2),
if print_true then (display(D),display(k)),
Dhc:Dhcmatrix(D,k),
f:matrix([f1]),
W:col(Dhc,1),
for i:1 while rank(W)=i and i<n do W:addcol(W,col(Dhc,i+1)),
r:rank(W),
m:length(transpose(W)),
if print_true then display(r,m),
if r=n then return(D),
for j:2 thru r do f:addrow(f,matrix([concat('f,j)])),
e:W.addrow(f,matrix([1])),
e:makelist(e[j,1],j,1,n),
if print_true then display(e),
f_list:linsolve(e,transpose(f)[1]),
if print_true then display(f_list),
U:ident(n),
for j:1 thru r do
  (f[j,1]:part(f_list,j,2),
   U:setelmx(f[j,1]*s^(k[m]-k[j]),j,m,U)),
if print_true then display(U),
columnreduce_iter(D.U,s))$

ROWREDUCE(y,[p]):=Block([s,D,N,Dl,U],
if y=help then return((
print("ROWREDUCE(D,N) performs elementary row operations  "),
print("such that the polynomial matrix D(s) becomes row   "),
print("reduced (row proper). By default D is a polynomial "),
print("matrix in s. In other cases the variable must be given"),
print("as a second argument. Ex.: ROWREDUCE(D,N,lambda)    "),
print("The answer is a list including the row proper D,    "),
print("the corresponding nominator matrix N in the MFD and "),
print("the unimodular matrix, U, that made the transform   "),
print("[ Dl= .., Nl= .., U= ..] "),
"-------------------------------------------------------------")),

D:y,
N:part(p,1),
if lenght(p)=2 then s:part(p,2),
Dl:transpose(columnreduce_iter(transpose(y),s)),
U:factor(Dl.(D^^-1)),
['Dl= factor(Dl), 'Nl= factor(U.N), 'U= U])$

/* Make a right MFD  */
```

```
RMFD(H):=Block([m,p,i,j,dr,nr,den],
if H=help then return((
print("Makes a right matrix fraction decomposition, Nr(Dr^^-1), of a"),
print("rational transfer function matrix, G(s). The extraction of the"),
print("denominator polynomial matrix is made by placing the least
common"),
print("multiple of each column denominator in the diagonal. Note that
this"),
print("doesn't always lead to an irreducible MFD. By default G is a
rational"),
print("matrix in s. In other cases the variable must be specified as a"),
print("second argument. Ex.: RMFD(G,lambda). The answer has the form"),
print("[Dr=..,Nr=..]"),
"-------------------------------------------------------------------")),

   p:length(h),
   m:length( transpose(h) ),
   dr:zeromatrix(m,m),
   for i:1 thru m do
     (den:1,
      for j:1 thru p do den:lcm(den,denom( h[j,i] ) , s ),
      dr:setelmx(den,i,i,dr) ),
   dr:factor(dr),
   nr:factor(h.dr),
   ['Dr= Dr,'Nr= Nr])$

/* Make a left MFD */
LMFD(H):=Block([m,p,i,j,dl,nl,den],
if H=help then return((
print("LMFD(H) works as RMFD(G) but makes instead a left MFD,"),
print("G(s)=(Dl^^-1)*Nl, The answer has the form [Dl=..,Nl=..]"),
"-----------------------------------------------------------")),

   p:length(h),
   m:length( transpose(h) ),
   dl:zeromatrix(p,p),
   for j:1 thru p do
     (den:1,
      for i:1 thru m do den:lcm(den,denom( h[j,i] ) , s ),
      dl:setelmx(den,j,j,dl) ),
   dl:factor(dl),
   nl:factor(dl.h),
   ['Dl= Dl,'Nl= Nl])$

/* Extracting a common right factor */
RIGHTCOPRIME(D,[N]):=Block([rrr,rr,i,dr,nr],
if D=help then return((
print("Makes an irreducible right MFD out of a reducible MFD. The
answer"),
```

```
print("also contains the extracted polynomial matrix factor, Rr, "),
print("[Dr=..,Nr=..,Rr=..]"),
"-------------------------------------------------------------------
")),
N:part(n,1),

   rrr:part(hermite(addrow(d,n)),1,2),
   rr:row(rrr,1),
   for i:2 thru length(d) do  rr:addrow(rr , row(rrr,i) ),
   dr:factor(d.(rr^^-1)),
   nr:factor(n.(rr^^-1)),
   ['dr= dr,'nr= nr,'rr= rr])$

/* Extracting a common left factor */
LEFTCOPRIME(D,[N]):=Block([rrr,rr,i,dr,nr,tmp,dl,nl],
if D=help then return((
print("As RIGHTCOPRIME(D,N) but makes instead a left irreducible MFD."),
print("[Dl=..,Nl=..,Rl=..]"),
"-------------------------------------------------------------------")),
N:part(n,1),

   dr:transpose(d),
   nr:transpose(n),
   tmp:rightcoprime(dr,nr),
   dl:transpose(part(tmp,1,2)),
   nl:transpose(part(tmp,2,3)),
   rl:transpose(part(tmp,3,2)),
   ['dl= dl,'nl= nl,'rl= rl])$

/* State-space to Transfer-function conversion */
SS2TF(sys):=Block([A,B,C,D,n],
if sys=help then return((
print("SS2TF(SYS) State space to transfer function conversion."),
print("The artument SYS has the form [A=..,B=.. C=..]"),
print("or [A=..,B=..,C=..,D=..]"),
"-----------------------------------------------------------")),

A:part(sys,1,2),
B:part(sys,2,2),
C:part(sys,3,2),
if length(sys)=4 then D:part(sys,4,2) else D:0,
n:length(A),

factor(C.((s*ident(n)-A)^^-1).B + D))$

/* Make list of A B C matrices representing a system */
MAKESYS(A,[x]):=Block([B,C],
if A=help then return((
print("MAKESYS(A,B,C,D) or MAKESYS(A,B,C) makes a list of A,B,C (and D)"),
```

```
print("matrices representing a system. [A=..,B=..,C=..,D=..]"),
"-----------------------------------------------------------")),

B:part(x,1),
C:part(x,2),
if length(x)=3 then ['a=a,'b=b,'c=c,'d=part(x,3)] else ['a=a,'b=b,'c=c])$

/* ---------------   REALIZATIONS ---------------------- */

/* --------- Help functions in CONTROLLER --------------- */
controlindices(x):=Block([n,k,j,i],
n:length(x),
k:makelist(0,i,1,n),
for j:1 thru n do
   (for i:1 thru n do k[j]:max(k[j],hipow(expand(x[i,j]),s))),
k)$

Dhcmatrix(x,k):=Block([n,i,Dhc],
n:length(x),
Dhc:zeromatrix(n,1),
for i:1 thru n do Dhc:addcol(Dhc,coeff(expand(col(x,i)),s,k[i])),
Dhc:submatrix(Dhc,1))$

lcmatrix(x,k):=Block([n,i,j,lc],
n:length(x),
lc:zeromatrix(n,1),
for i:1 thru n do
    (for j:1 thru k[i] do
         (lc:addcol(lc,coeff(expand(col(x,i)),s,k[i]-j)))),
lc:submatrix(lc,1))$

Core(k):=Block([m,n,i,j,a,b],
m:length(k),
n:sum(k[i],i,1,m),
if n=1 then A:zeromatrix(n,n) else
  (A:ident(n-1),
   for i:1 thru m-1 do A[sum(k[j],j,1,i),sum(k[j],j,1,i)]:0,
   A:addrow(zeromatrix(1,n),addcol(A,zeromatrix(n-1,1)))),
B:zeromatrix(n,m),
B[1,1]:1,
for i:1 thru m-1 do B[sum(k[j],j,1,i)+1,i+1]:1,
[A,B])$
/* ------------------------------------------- */

CONTROLLER(x,[y]):=Block([D,N,s,k,Dhc,Dlc,Core,A,B,C],
if x=help then return((
print("CONTROLLER(D,N) makes a controller form realization of a right"),
print("coprime MFD. Note that the denominator matrix D must be"),
print("column-reduced. By default D is a polynomial matrix in s, D(s)."),
```

```
print("In other cases the variable must be given as a third argument."),
print("Ex.: CONTROLLER(D,N,lambda)"),
print("The answer has the form [A= .., B= .., C= ..]"),
"-----------------------------------------------------------------")),

/* Pick out the arguments */
D:x,
N:part(y,1),
if length(y)=2 then s:part(y,2),

k:controlindices(D),
Dhc:Dhcmatrix(D,k),
if determinant(Dhc)=0 then return("D-matrix not column-reduced"),
Dlc:lcmatrix(D,k),
Core:Core(k),
A:Core[1],
B:Core[2],
A:A-B.(Dhc^^-1).Dlc,
B:B.(Dhc^^-1),
C:lcmatrix(N,k),
['A= A,'B= B,'C= C])$

OBSERVER(x,[y]):=Block([D,N,s,tmp,k,Dhc],
if x=help then return((
print("OBSERVER(D,N) makes a observer form realization of a left"),
print("coprime MFD. Note that the denominator matrix D must be"),
print("row-reduced. By default D is a polynomial matrix in s, D(s)."),
print("In other cases the variable must be given as a third argument."),
print("Ex.: OBSERVER(D,N,lambda)"),
print("The answer has the form [A= .., B= .., C= ..]"),
"-----------------------------------------------------------------")),

/* Pick out the arguments */
D:x,
N:part(y,1),
if length(y)=2 then s:part(y,2),

k:controlindices(transpose(D)),
Dhc:Dhcmatrix(transpose(D),k),
if determinant(Dhc)=0 then return("D-matrix not row-reduced"),
tmp:controller(transpose(d),transpose(n)),
A:transpose(part(tmp,1,2)),
C:transpose(part(tmp,2,2)),
B:transpose(part(tmp,3,2)),
['A= A,'B= B,'C= C])$

CONTROLLABILITY(x,[y]):=Block(
[D,N,s,dhc,ccc,k,a,b,c,col,A_Ab,i,j,T,chain,m,nrcols,A_A,test],
if x=help then return((
```

```
print("CONTROLLABILITY(D,N) makes a controllability form realization of"),
print("a right coprime MFD. Note that the denominator matrix D must be"),
print("column-reduced. By default D is a polynomial matrix in s, D(s)."),
print("In other cases the variable must be given as a third argument."),
print("Ex.: CONTROLLABILITY(D,N,lambda)"),
print("It is also possible to have an arbitrary (controllable!)"),
print("realization as input, CONTROLLABILITY(A,B,C)"),
print("A global variable 'crate_nr' (default=2) makes a choice "),
print("between 'Search by crate 1 or 2 diagram' (Young diagram)"),
print("The answer has the form [A= .., B= .., C= ..]"),
"-----------------------------------------------------------------")),

/* Pick out the arguments */
if length(y)=1 then (D:x,N:part(y,1)),
if length(y)=2 then
 (if matrixp(part(y,2)) then (A:x,B:part(y,1),C:part(y,2))
 else s:part(y,2)),
if A#x then
 (k:controlindices(D),
  Dhc:Dhcmatrix(D,k),
  if determinant(Dhc)=0 then return("D-matrix not column-reduced"),
  ccc:controller(d,n),
  A:part(ccc,1,2),
  B:part(ccc,2,2),
  C:part(ccc,3,2)),

/* Search by Crate 1 */
m:length(A),
if crate_nr=1 then
 ( T:zeromatrix(m,1),
   for i:1 while rank(T)<m do
     ( col:col(B,i),
       T:addcol(T,col),
       A_Ab:A.col,
       for j:1 while rank(addcol(T,A_Ab))>rank(T) do
         ( T:addcol(T,A_Ab),
           A_Ab:A.A_Ab ) ),
   T:submatrix(T,1) )

/* Search by Crate 2 */
else
 ( /* find controllability indices - chains */
   nrcols:length(transpose(B)),
   chain:zeromatrix(nrcols,1)^0,
   A_A:ident(m),
   if rank(B)<nrcols then return("B matrix not full rank") else T:B,
   for i:1 while rank(T)<m do
    ( A_A:A_A.A,
      for j:1 thru nrcols do
```

```
        ( test:addcol(T,A_A.col(B,j)),
          if rank(test)>rank(T) then (T:test,chain[j,1]:chain[j,1]+1))),
    /* Permutate the crate search to get the transform */
    T:zeromatrix(m,1),
    for i:1 thru nrcols do
     ( A_Ab:col(B,i),
       T:addcol(T,A_Ab),
       for j:1 thru chain[i,1]-1 do
        ( A_Ab:A.A_Ab,
          T:addcol(T,A_Ab))),
    T:submatrix(T,1)),

A:(T^^-1).A.T,
B:(T^^-1).B,
C:C.T,
['A= A,'B= B,'C= C])$

OBSERVABILITY(x,[y]):=Block(
[D,N,s,dhc,ccc,k,a,b,c,col,A_Ab,i,j,T,chain,m,nrcols,A_A,test],
if x=help then return((
print("OBSERVABILITY(D,N) makes an observability form realization of"),
print("a left coprime MFD. Note that the denominator matrix D must be"),
print("row-reduced. By default D is a polynomial matrix in s, D(s)."),
print("In other cases the variable must be given as a third argument."),
print("Ex.: OBSERVABILITY(D,N,lambda)"),
print("It is also possible to have an arbitrary (observable!)"),
print("realization as input, OBSERVABILITY(A,B,C)"),
print("A global variable 'crate_nr' (default=2) makes a choice "),
print("between 'Search by crate 1 or 2 diagram' (Young diagram)"),
print("The answer has the form [A= .., B= .., C= ..]"),
"-----------------------------------------------------------------")),

/* Pick out the arguments */
if length(y)=1 then (D:x,N:part(y,1)),
if length(y)=2 then
 (if matrixp(part(y,2)) then (A:x,B:part(y,1),C:part(y,2))
 else s:part(y,2)),
if A#x then
 (k:controlindices(transpose(D)),
  Dhc:Dhcmatrix(transpose(D),k),
  if determinant(Dhc)=0 then return("D-matrix not row-reduced"),
  ccc:observer(D,N),
  A:part(ccc,1,2),
  B:part(ccc,2,2),
  C:part(ccc,3,2)),

tmp:controllability(transpose(A),transpose(C),transpose(B)),
['A= transpose(part(tmp,1,2)),'B=
transpose(part(tmp,3,2)),'C=transpose(part(tmp,2,2))])$
```

```
(defun print-matrix (name value f)
  (patom name f)
  (patom " = " f)
  (terpr f)
  (terpr f)
  (mapcar
    '(lambda (x)
       (mapcar
         '(lambda (y)
            (if (symbolp y)
                (patom (implode (cdr (explode y))) f)
                (patom y f))
            (patom " " f))
          x)
       (terpr f))
     value)
  (terpr f))

(defun TOMIMO (system file)
  (let ((a (a-matrix-name system))
        (b (b-matrix-name system))
        (c (c-matrix-name system))
        (d (d-matrix-name system))
        (a-value (a-matrix system))
        (b-value (b-matrix system))
        (c-value (c-matrix system))
        (d-value (d-matrix system))
        (f (outfile file)))
    (terpr f)
    (patom "nmp = " f)
    (terpr f)
    (terpr f)
    (patom (length (car a-value)) f)
    (patom " " f)
    (patom (length (car b-value)) f)
    (patom " " f)
    (patom (length c-value) f)
    (patom " " f)
    (terpr f)
    (terpr f)
    (print-matrix a a-value f)
    (print-matrix b b-value f)
    (print-matrix c c-value f)
    (print-matrix d d-value f)
    (close f)))

(defun a-matrix-name (system)
```

```
  (implode (cdr (explode (cadadr system)))))

(defun b-matrix-name (system)
  (implode (cdr (explode (cadaddr system)))))

(defun c-matrix-name (system)
  (implode (cdr (explode (cadadddr system)))))

(defun d-matrix-name (system)
  (implode (cdr (explode (cadaddddr system)))))

(defun a-matrix (system)
  (mapcar 'cdr (cdaddadr system)))

(defun b-matrix (system)
  (mapcar 'cdr (cdaddaddr system)))

(defun c-matrix (system)
  (mapcar 'cdr (cdaddadddr system)))

(defun d-matrix (system)
  (mapcar 'cdr (cdaddaddddr system)))
```