Version 2.5

The Polynomial Toolbox for MATLAB

Upgrade Information
for Version 2.5

```
P = 1 + .1s    .25 + s^3
    2s + s^2   .5 + s^2
```

February, 2001

PolyX, Ltd

E-mail info@polyx.com

Support support@polyx.com

Sales sales@polyx.com

Web www.polyx.com

Tel. +420-2-66052314

Fax +420-2-6884554

Jarni 4, Prague 6, 16000

Czech Republic

Polynomial Toolbox Manual

Printing history: February 2001. First printing.

Contents

1 Introduction

This document highlights the new features of Version 2.5 of the Polynomial Toolbox for MATLAB

How to use this document

If you are upgrading to Version 2.5 of the Polynomial Toolbox from …	Then read ...
Polynomial Toolbox Version 2.0	All sections of the present document.
Polynomial Toolbox Version 1.4 or1.5 for Matlab 4	The complete documentation of Version 2 and then all sections of the present document. It may also be necessary to update your MATLAB knowledge.

References to other documents

Throughout this document there are references to the Manual and the Commands volumes of Version 2.0 of the Polynomial Toolbox.

New installation instructions

The Polynomial Toolbox 2.5 may be installed in the following simple steps.

Windows platforms

- Delete any existing Polynomial Toolbox Version 2.0 (the existing folder `...\polynomial` and its contents).
- Copy the whole folder `\polynomial` including all its contents from the Polynomial Toolbox CD-ROM Version 2.5 to your PC, preferably next to other MATLAB toolboxes that all are placed in the folder `...\MATLABR12\toolbox` or `...\MATLABR11\toolbox` or `...\MATLAB\toolbox`.
- Add the folder `...\polynomial` to your MATLAB path (for instance by using the MATLAB Path Browser).
- If you use version 2 of SIMULINK then replace the file `...\polynomial\ polblock.mdl`, residing in the main Polynomial Toolbox directory, by the file `...\polynomial\simulink2\polblock.mdl`. The current version of SIMULINK can be checked by typing "`ver simulink`" in the MATLAB main window.

- You are recommended to add a new line to your `startup.m` file containing the command `PINIT`. With this modification the Polynomial Toolbox is automatically initialized at the beginning of every MATLAB session. If you do not do this then you will have to type `PINIT` manually each time you start a Polynomial Toolbox session.
- When using the Polynomial Toolbox for the first time after installation you will be asked to provide your personal license number.
- The standard configuration of the Polynomial Toolbox contains an Acrobat Reader placed in the folder `...\polynomial\Pdf-files\Acrobat 4.0`, which guarantees easy use of the on-line documentation by the `POLDESK` command. This configuration requires no action during the installation and it is recommended for most users. For more details on using the on-line documentation see the section Documentation.

UNIX platforms

- Delete any existing Polynomial Toolbox Version 2.0 (the existing directory `.../polynomial` and its contents).
- Copy the whole directory /polynomial including all its contents from the Polynomial Toolbox CD-ROM Version 2.5 to your system, preferably next to the other MATLAB toolboxes that all are placed in the directory `.../MATLABR12/ toolbox` or `.../MATLABR11/toolbox` or `.../MATLAB/toolbox`.
- Add the directory `.../polynomial` to your MATLAB path.
- If you use version 2 of SIMULINK then replace the file `...polynomial/ polblock.mdl`, residing in the main Polynomial Toolbox directory, by the file `.../polynomial/simulink2/polblock.mdl`. The current version of SIMULINK may be checked by typing "`ver simulink`" in the MATLAB main window.
- You are recommended to add a new line to your `startup.m` file containing the command `PINIT`. With this modification the Polynomial Toolbox is automatically initialized at the beginning of every MATLAB session. If you do not do this then you will have to type `PINIT` manually each time you start a new Polynomial Toolbox session.
- When using the Polynomial Toolbox for the first time after installation you will be asked to provide your personal license number.
- To access the Polynomial Toolbox on-line documentation by the command `POLDESK` your UNIX system is supposed to run Acrobat Reader by the usual command "`acroread`." If this is not the case then you must create such an alias, or ask your system administrator for help. For more details on using the on-line documentation see the section Documentation.

Upgrading instructions

Older versions of the Polynomial Toolbox 2.x may be upgraded to Version 2.5 by executing the following steps.

Windows platforms

- Make sure that your current folder `...\polynomial` and all its contents are not read-only. You can check this by right-clicking a few files and viewing the properties sheet. To disable the read-only attribute of all files and folders in the Polynomial Toolbox right-click the top level Polynomial Toolbox folder `...\ polynomial` and open the properties sheet. Uncheck the box “Read-only” and click on OK. In some versions of Windows you can now select the option “Apply changes to this folder, subfolders and files” and again click on OK. If this option is not available then repeat the procedure for all subfolders of `...\ polynomial`. Alternatively, you may open a DOS-box, change to the folder `...\ polynomial`, and type the command “`attrib -r *.* /s /d`” to disable the read-only attribute of all files and folders in the Polynomial Toolbox.
- Copy the entire contents of the folder `upgrade\polynomial` including all subfolders from the Polynomial Toolbox CD-ROM Version 2.5 to your PC over the contents of the existing folder `...\polynomial`.
- If you still use version 2 of SIMULINK then replace the file `...\polynomial\ polblock.mdl`, residing in the main Polynomial Toolbox directory, with the file `...\polynomial\simulink2\polblock.mdl`. You may check the current version of SIMULINK by typing “`ver simulink`” in the MATLAB main window.

Alternatively, you may delete the old Polynomial Toolbox version (the whole folder `...\polynomial` including its contents) and then install Version 2.5 following the “New installation instructions.” In this case you will be asked to provide your personal license number.

UNIX platforms

- Make sure that your current directory `.../polynomial` and all its contents are write enabled. Check this by moving to the directory `.../polynomial,` typing “`ll`” and inspecting the w flag. If the w flag is not present in the user access persissions for all files then type “`chmod u+w *`” to set it. Repeat this for all sub-directories.
- Copy the entire contents of the directory `upgrade/polynomial` including all subfolders from the Polynomial Toolbox CD-ROM Version 2.5 to your computer over the contents of the existing directory `.../polynomial`.
- If you still use version 2 of SIMULINK then replace the file `.../polynomial/ polblock.mdl`, residing in the main Polynomial Toolbox directory, with the file `.../polynomial/simulink2/polblock.mdl`. You may check the current version of SIMULINK by typing "`ver simulink`" in the MATLAB main window.

Alternatively, you may delete the old Polynomial Toolbox version (the whole folder `...\polynomial` including its contents) and then install Version 2.5 following the "New installation instructions." In this case you will be asked to provide your personal license number.

Documentation

Three document volumes are provided with the Polynomial Toolbox: Manual, Commands, and Version 2.5 Upgrade Information. The printed Manual and Version 2.5 Upgrade Information volumes are delivered with the Polynomial Toolbox CD-ROM. The printed Commands volume may be purchased separately (see the PolyX website or contact `info@polyx.cz`).

Ready-to-print electronic versions of the Manual, Commands and Version 2.5 Upgrade Information are also available. They may be found in `...\polynomial\Pdf-files` in the files `manual.pdf`, `commands.pdf` and `upgradeinfo25.pdf`. The files are all readable by Acrobat Reader. Users are welcome to print these files for their own use but should not distribute them any further. For more copyright details see the License Agreement.

On-line electronic versions of the Manual, Commands and Version 2.5 Upgrade Information are also provided. They are located in the folder `...\polynomial\Pdf-files` in the files `OnLineManual.pdf`, `OnLineCommands.pdf` and `OnLineUpgradeInfo25.pdf`. They are normally accessed by the Polynomial Toolbox command `POLDESK` but users are free to create other arrangements.

On Windows Platforms, `POLDESK` by default uses Acrobat Reader located in the Polynomial Toolbox folder `...\polynomial\Pdf-files\Acrobat 4.0` . This configuration is generally recommended. If an experienced user wishes to employ a different version of Acrobat Reader located elsewhere then the entire folder `...\polynomial\Pdf-files\Acrobat 4.0` may simply be deleted. During the next execution `POLDESK` will look for Acrobat Reader in the standard location `C:\Program Files\ Adobe\Acrobat 4.0\Reader\ AcroRd32.exe` or will ask the user to provide a valid path name.

On UNIX Platforms, `POLDESK` by default calls the command "`acroread`" that typically runs Acrobat Reader on a UNIX system. If this alias is not recognized then the user or a system administrator may create such an alias.

Alternatively, the user of each system may type `POLDESK RECOVER`. This opens a dialogue window where the user can type in a valid pathname.

A note for SIMULINK 3 users on Windows platforms

Under the MS Windows operating systems the way the "`simulink`" command is processed differs slightly in versions 2 and 3 or 4 of SIMULINK. The instructions in the Polynomial Toolbox 2.0 Manual (pages 83–84) refer to version 2 of SIMULINK. If you use SIMULINK 3 or 4 under Windows then please proceed in one of the two following ways:

1. Type

   ```
   » simulink
   ```

   to open the Simulink Library browser. The Polynomial Toolbox 2.0 Simulink library now is directly accessible within the browser along with the other Simulink libraries.

2. Type

   ```
   » simulink3
   ```

   to open the Simulink Library window. Follow the instructions in the Polynomial Toolbox 2.0 Manual, pages 83–84.

For further information consult the SIMULINK manual (Using Simulink, Version 3).

Compatibility with MATLAB version 6

The Polynomial Toolbox 2.5 works well with the MATLAB Release 12 products MATLAB 6 and Simulink 4. In fact, some functions are up to two times faster with MATLAB 6 than before.

MATLAB 6 users will see the Polynomial Toolbox icon in their MATLAB Launch Pad window among the other MATLAB toolboxes they may have. The Polynomial Toolbox help functions, demos, Polynomial Matrix Editor and PolyX web site may be directly accessed from the Launch Pad window.

Clicking a POL object icon in the MATLAB Workspace window does not open the object in an array editor. We hope to fix this shortcoming in the future but the related MATLAB code is not open for us currently. Instead, type PME in the command window and open the object in the Polynomial Matrix Editor.

2 What is New in Version 2.5?

Overview

Version 2.5 features the following enhancements.

- Bug fixes to Version 2.0
- Improved algorithms and other internal changes
- New display formats
- Several new functions
- Miscellaneous updates and modifications

Bug fixes

Version 2.5 includes a number of bug fixes. In particular, it includes all patches that were made available on the PolyX website since the release of Version 2.0.

Improved algorithms and other internal changes

Several algorithms have been improved in Version 2.5 to reflect recent research achievements. In particular, the linear polynomial matrix equation solvers `axb`, `axbyc`, `xab`, `xaybc`, and `axxa2b` perform faster, in particular for large matrices. These modifications have no impact on the way the functions are used and hence require no attention on the part of the user. In particular, no changes were made in the numbers of input and output arguments and their order.

New display formats

Version 2.5 includes several additional display formats for polynomial matrices.

`pformat rootr`	Format a polynomial or polynomial entry as a product of real first- and second-order factors
`pformat rootc`	Format a polynomial or polynomial entry as a product of first-order factors
`pdisp`	Display a polynomial matrix without printing the name

New functions

Several new functions were added in Version 2.5.

LaTeX formatting of polynomial matrices

The new routine `pol2tex` is a great help for authors who use LaTeX.

`pol2tex`	Formats a polynomial matrix for use in a LaTeX document

H2 optimization

Version 2.5 offers two new solutions for the standard H_2 problem under quite general conditions.

`h2`	Polynomial solution of the standard H_2 optimization problem
`dssh2`	Descriptor solution of the standard H_2 optimization problem

Interval polynomials

Version 2.5 adds the following new macros to the already impressive list of routines for testing the stability of interval polynomials

`jury`	Create the Jury matrix corresponding to a polynomial
`sarea, sareaplot`	Robust stability area for polynomials with parametric uncertainties
`spherplot`	Plot the value set ellipses for a spherical polynomial family
`tsyp`	Use the Tsypkin-Polyak function to determine the ℓ_∞ robustness margin for a continuous interval polynomial
`vset,vsetplot`	Value set of parametric polynomial. A tool for robust stability testing via Zero Exclusion Condition

State space systems

Version 2.5 includes two polynomial methods for state space systems

`psseig`	Polynomial approach to eigenstructure assignment for state-space system
`psslqr`	Polynomial approach to linear-quadratic regulator design for state-space system

Simulink routines

Two brand new routines allow the automatic conversion of SIMULINK block diagrams to LMF and RMF descriptions.

`sim2lmf`	Simulink-to-LMF description of a dynamic system
`sim2rmf`	Simulink-to-RMF description of a dynamic system

Numerical routines

Version 2.5 includes two upgrades of existing numerical utilities and a new numerical function.

`clements1`	Conversion to Clements standard form (upgrade of `clements`)
`dssreg`	“Regularizes” a standard descriptor plant (upgrade)
`gare`	Solution of the generalized algebraic Riccati equation

Polynomial matrix functions

The function `complete` is a new addition to the collection of polynomial matrix functions.

`complete`	Complete a non-square polynomial matrix to a square unimodular matrix

Demos and shows

Three new text based demos have been included in Version 2.5. They are self-explanatory and no documentation is available. Simply type the name of the demo in the command line.

`poldemo`	This demo reviews several of the functions and operations defined in the Polynomial Toolbox for polynomials and polynomial matrices
`poldemodebe`	Design of a dead-beat compensator
`poldemodet`	Comparison between numerical and symbolic computation of determinant of a polynomial matrix. This demo requires the Symbolic Toolbox to be installed

In addition two “shows” have been prepared that run in a graphical interface. Enter the name of the show in the command line to view the show. No additional documentation is available.

`poltutorshow`	Introduction into the basic operations with polynomials and polynomial matrices. This is a graphical version of the text based demo `poldemo`
`polrobustshow`	Overview of parametric robust control tools

Miscellaneous updates and modifications

This section lists modifications in various macros that were made after Version 2.0 was released. The changes leave the macros fully compatible with Version 2.0 and are all reflected in the on-line help.

axxab

There are a number of improvements in `axxab`.

- By default, the macro `axxab` now returns a solution with triangular leading coefficient matrix (in the continuous-time case) or triangular constant coefficient matrix (in the discrete-time case). The option 'tri' is no longer effective but still valid for compatibility reasons.
- By default, the macro now uses the sparse linear system solver and performs no preliminary rank check.
- The new option `'chk'` turns the preliminary rank check on and activates MATLAB's built-in standard (non-sparse) linear system solver.

cgivens1

The macro `cgivens1` differs from the implementation in Version 2.5 by the introduction of an optional tolerance `tol`. The default value of `tol` is 0. In the form

```
[c,s] = cgivens1(x,y,tol)
```

the routine sets `x` and `y` equal to zero if their magnitude is less than `tol`.

isstable

Unimodular polynomial matrices and constant non-polynomial matrices are now considered to be stable, and not unstable as in Version 2.0.

prand

The macro `prand` has two new options.

- The option `'mon'` generates a monic polynomial matrix.
- The option `'pos'` generates a polynomial matrix with the required number of zeros. In particular, the call `P = prand(degP,I,'pos'[,zeros_vector])` generates a square `I`-by-`I` polynomial matrix `P` but now `degP` means the required number of zeros, including multiplicities. Some zeros can be fixed a priori by the optional vector `zeros_vector`. Complex conjugate complex parts are added if necessary.

reverse

The function call

```
reverse(P)
```

with the single input argument P, reverses the order of the coefficients. Thus, if

$$P(s) = P_0 + P_1 s + \cdots + P_n s^n$$

then

```
Q = reverse(P)
```

returns

$$Q(s) = P_n + P_{n-1}s + \cdots + P_0 s^n$$

root2pol Zeroing management has been changed in this macro. Now, no zeroing is performed by default. However, an optional tolerance `tol` may be passed to the macro in one of the forms

```
P = root2pol(Z,K,tol)
```

or

```
P = root2pol(Z,K,tol,var)
```

In this case all coefficients of the resulting polynomial that are less than `tol` times the largest coefficient are neglected. Note that if the tolerance argument is included both the input argument `Z` and `K` needs to be present.

stabint The on-line help has been modified to emphasize that the routine does not work with complex polynomials.

3 New Functions in Version 2.5 of the Polynomial Toolbox

This chapter documents the new functions of Version 2.5 of the Polynomial Toolbox.

clements1

Purpose Transformation of a para-Hermitian pencil to Clements form

Syntax

```
[C,u,p] = clements1(P)
[C,u,p] = clements1(P,q)
[C,u,p] = clements1(P,q,tol)
```

Description The command

```
[C,u,p] = clements1(P,q,tol)
```

transforms the para-Hermitian nonsingular real pencil $P(s) = sE + A$ to Clements standard form C according to

$$C(s) = u(sE + A)u^T = se + a$$

The matrix u is orthogonal. The pencil C has the form

$$C(s) = se + a = \begin{bmatrix} 0 & 0 & se_1 + a_1 \\ 0 & a_2 & se_3 + a_3 \\ -se_1^T + a_1^T & -se_3^T + a_3^T & se_4 + a_4 \end{bmatrix}$$

The pencil $se_1 + a_1$ has size $p \times p$ and its finite roots have nonnegative real parts. The matrix a_2 is diagonal with the diagonal entries in order of increasing value.

If the optional input argument q is not present then a_2 has the largest possible size. If q is present and the largest possible size of a_2 is greater than $q \times q$ then – if possible – the size of a_2 is reduced to $q \times q$. Setting q = `Inf` has the same effect as omitting the second input argument.

The optional input parameter `tol` defines a relative tolerance with default value `1e-10.` It is used to test whether eigenvalues of the pencil are zero, have zero imaginary part, or are infinite, and for other tests. For compatibility with an earlier version of the macro a tolerance parameter of the form `[tol1 tol2]` is also accepted but only the first entry is used.

Compatibility This version is backward compatible with the earlier version (named `clement` in Version 2.0) but also handles singular pencils and pencils with roots on the imaginary axis. Because of certain modifications in the algorithm `clements` and `clements1` generally do not produce the same output for the same input.

Example We consider the computation of the Clements form of the para-Hermitian pencil

$$P(s) = \begin{bmatrix} 100 & -0.01 & s & 0 \\ -0.01 & -0.01 & 0 & 1 \\ -s & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

We first input this matrix as

```
P = [100 -0.01 s 0; -0.01 -0.01 0 1;-s 0 -1 0;0 1 0 0];
```

and next compute its Clements form:

```
[C,u,p] = clements(P);
```

We have

```
p, C = pzer(C)
p =
     1
C =
     0            0                     0                  -10 + s
     0           -1                     0           -3.5e-005 - 0.0007s
     0            0                     1             -0.014 + 0.00071s
   -10 - s   -3.5e-005 + 0.0007s   -0.014 - 0.00071s    99
```

We see that

$$se_1 + a_1 = s - 10, \qquad a_2 = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$$

Next we attempt to reduce a_2 to the smallest possible size:

```
[C,u,p] = clements(P,0);
p, C = pzer(C)
p =
     2
Polynomial matrix in s: 4-by-4,  degree: 1
C =
     0          0                    0                  -10 + s
     0          0                    1             -0.01 + 5e-006s
```

```
     0           1                     -0.01          -0.0099 + 0.001s
   -10 - s    -0.01 - 5e-006s   -0.0099 - 0.001s       99
```

We now have

$$se_1 + a_1 = \begin{bmatrix} 0 & s-10 \\ 1 & 5\times10^{-6}s - 0.01 \end{bmatrix}$$

while a_2 is the empty matrix.

Algorithm

The algorithm is described in Clements (1993) and in slightly more detail in Kwakernaak (1998). The extension to roots on the imaginary axis is described in Kwakernaak (2000).

References

Clements, D. J. (1993), "Rational spectral factorization using state-space methods." *Systems & Control Letters*, vol. 20, pp. 335–343.

Kwakernaak, H. (1998), "Frequency domain solution of the H_∞ problem for descriptor systems." In Y. Yamamoto and S. Hara, Eds., *Learning, Control and Hybrid Systems*, Lecture Notes in Control and Information Sciences, vol. 241, Springer, London, etc.

H. Kwakernaak (2000), "A Descriptor Algorithm for the Spectral Factorization of Polynomial Matrices." Third IFAC Symposium on Robust Control System Design ROCOND 2000, Prague, June 21–23, 2000.

Diagnostics

The macro displays error messages in the following situations:

- The input matrix is not a square pencil
- The input matrix is not real
- The input pencil is not para-Hermitian
- An eigenvalue on the imaginary axis cannot be deflated

A warning message is issued if the relative residue exceeds `1e-6`. The "relative residue" is the norm of the juxtaposition of the (1,1) and (1,2) blocks of *C* divided by the norm of *P*.

See also

`dsshinf`	H_∞-suboptimal compensators for descriptor systems
`gare`	Solution of generalized algebraic Riccati equations

complete

Purpose Complete a nonsquare polynomial matrix to a unimodular matrix

Syntax `[U,V] = complete(Q,[tol])`

Description If Q is a tall polynomial matrix then the command

```
[U,V] = complete(Q)
```

produces a unimodular matrix U of the form $U = [Q\ R]$. If Q is wide then the unimodular matrix U has the form $U = [Q; R]$. V is the inverse of U.

If Q does not have full rank or is not prime then no unimodular matrix U exists and an error message follows. Also if Q is square non-unimodular an error is reported.

The optional input argument `tol` is the tolerance used for the row or column reduction of Q that is part of the algorithm.

Compatibility This is a new function in the Polynomial Toolbox.

Example A tall polynomial matrix Q with column degrees 2 and 1 and dimensions 3×2 is generated by the command

```
Q = prand([2 1],3,2)
Q =
     1.6 - 1.1s - 0.026s^2    -1.1 + 0.75s
     0.5 - 0.52s - 0.56s^2    -0.75 + 0.93s
    -0.25 - 0.15s - 1.3s^2     0.31 + 2.7s
```

Q is completed to a unimodular matrix U by typing

```
[U,V] = complete(Q);
U
U =
     1.6 - 1.1s - 0.026s^2    -1.1 + 0.75s     0.2 + 0.00074s
     0.5 - 0.52s - 0.56s^2    -0.75 + 0.93s    0.4 + 0.016s
    -0.25 - 0.15s - 1.3s^2     0.31 + 2.7s     1 + 0.036s
```

It may be verified that U is unimodular and that V is its inverse by successively typing

```
det(U)
Constant polynomial matrix: 1-by-1
```

```
ans =
    -0.73
U*V
Constant polynomial matrix: 3-by-3
ans =
     1     0     0
     0     1     0
     0     0     1
```

Algorithm

Let Q be a full rank $n \times k$ polynomial matrix, with $n > k$. We wish to find an $n \times (n-k)$ polynomial matrix R such that $\begin{bmatrix} Q & R \end{bmatrix}$ is unimodular. Let U be a unimodular matrix which reduces Q to the extended row-reduced form

$$UQ = \begin{bmatrix} Q_o \\ 0 \end{bmatrix}$$

If the $k \times k$ matrix Q_o is a constant matrix then it is nonsingular and the desired unimodular completion exists. Otherwise, the completion does not exist. The row reduction algorithm also yields the inverse $V = U^{-1}$ of U. Redefine

$$U := \begin{bmatrix} Q_o^{-1} & 0 \\ 0 & I \end{bmatrix} U, \quad V := V \begin{bmatrix} Q_o & 0 \\ 0 & I \end{bmatrix}$$

and partition $V = \begin{bmatrix} V_1 & V_2 \end{bmatrix}$. Then

$$UQ = \begin{bmatrix} I \\ 0 \end{bmatrix}, \quad UV_2 = \begin{bmatrix} 0 \\ I \end{bmatrix}$$

Hence, the desired completion is

$$\begin{bmatrix} Q & V_2 \end{bmatrix}$$

and its inverse is U. If Q is not tall but wide then the algorithm is applied to the transpose of Q.

Diagnostics

The macro `complete` issues error messages if

- The input matrix is square non-unimodular
- The input matrix cannot be completed to a unimodular matrix because it is not prime
- The input matrix does not have full rank

See also

`colred, rowred` Reduction to column or row reduced form

dssh2

Purpose Descriptor solution of the H2 problem

Syntax `[Ak,Bk,Ck,Dk,Ek] = dssh2(A,B,C,D,E,nmeas,ncon,tol)`

Description The command

```
[Ak,Bk,Ck,Dk,Ek] = dssh2(A,B,C,D,E,nmeas,ncon,tol)
```

solves the H2 optimization problem for the standard plant

$$G(s) = C(sE - A)^{-1}B + D$$

with `nmeas` measured outputs and `ncon` control inputs. The optimal compensator is given by

$$K(s) = C_k(sE_k - A_k)^{-1}B_k + D_k$$

The optional parameter `tol` is a tolerance with default value `1e-10`.

Conditions on the input data: If D is partitioned as

$$D = \begin{bmatrix} D_{11} & D_{12} \\ D_{21} & D_{22} \end{bmatrix}$$

where D_{12} has `ncon` columns and D_{21} has `nmeas` rows, then D_{12} needs to have full column rank and D_{21} full row rank, and D_{22} should be the zero matrix. Use the command `dssreg` with the option `'D22'` to "regularize" the system if these conditions are not met.

Compatibility This function is new in the Polynomial Toolbox.

Example *H_2 design problem*

Consider the block diagram of Fig. 1. The plant is a MIMO system with transfer matrix

$$P(s) = \begin{bmatrix} \dfrac{1}{s^2} & \dfrac{1}{s+2} \\ 0 & \dfrac{1}{s+2} \end{bmatrix}$$

The controlled output is

$$z = \begin{bmatrix} z_{11} \\ z_{12} \end{bmatrix}$$

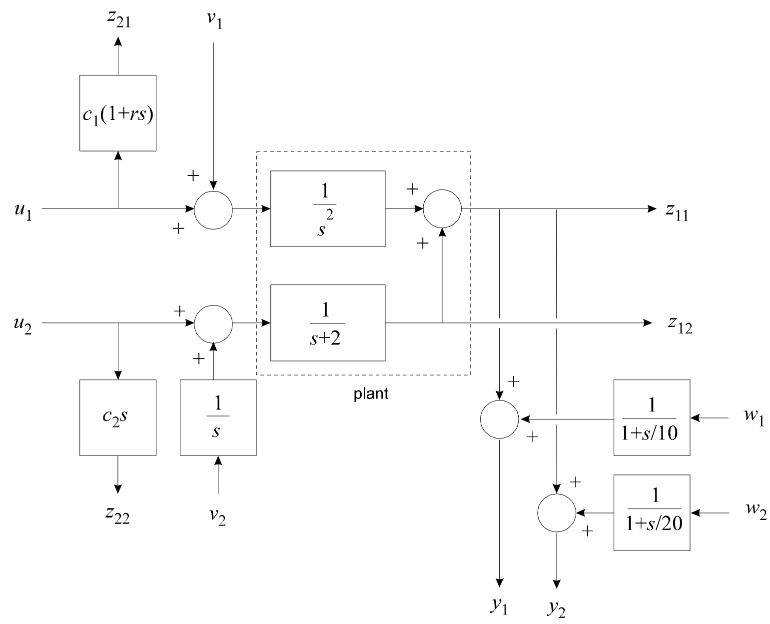

Fig. 1. Design problem

The measured output

$$y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

is corrupted by colored measurement noise generated by the two shaping filters with transfer functions

$$\frac{1}{1+s/10} \quad \text{and} \quad \frac{1}{1+s/20}$$

The second component of the disturbance

$$v = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

is passed through a shaping filter with transfer function $1/s$ to ensure integrating action on both input channels. The input

$$u = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

is weighted with dynamic weighting functions with transfer functions $c_1(1+rs)$ (to ensure sufficient high-frequency roll-off of the compensator) and $c_2 s$ (both for high-frequency roll-off and to allow integral control at the second input channel).

The generalized plant that defines the H_2 problem is given by

$$G(s) = \begin{bmatrix} G_{11}(s) & G_{12}(s) \\ G_{21}(s) & G_{22}(s) \end{bmatrix} = \left[\begin{array}{cccc|cc} \frac{1}{s^2} & \frac{1}{s(s+2)} & 0 & 0 & \frac{1}{s^2} & \frac{1}{s+2} \\ 0 & \frac{1}{s(s+2)} & 0 & 0 & 0 & \frac{1}{s+2} \\ 0 & 0 & 0 & 0 & c_1(1+rs) & 0 \\ 0 & 0 & 0 & 0 & 0 & c_2 s \\ \hline \frac{1}{s^2} & \frac{1}{s(s+2)} & \frac{1}{1+s/10} & 0 & \frac{1}{s^2} & \frac{1}{s+2} \\ 0 & \frac{1}{s(s+2)} & 0 & \frac{1}{1+s/20} & 0 & \frac{1}{s+2} \end{array}\right]$$

The transfer matrix may be entered in rational format, converted to a left polynomial matrix fraction by the command `rat2lmf` and after this converted to descriptor representation by the command `lmf2dss`:

```
c1 = 1; c2 = 1; r = 5;
Num = [ 1  1  0  0     1        1
        0  1  0  0     0        1
        0  0  0  0 c1*(1+r*s)   0
        0  0  0  0     0      c2*s
       -1 -1 -1  0    -1       -1
        0 -1  0 -1     0       -1 ];
Den = [ s^2 s*(s+2)   1      1    s^2  s+2
         1  s*(s+2)   1      1     1   s+2
         1     1      1      1     1    1
         1     1      1      1     1    1
        s^2 s*(s+2) 1+s/10   1    s^2  s+2
```

```
         1   s*(s+2)    1     1+s/20   1    s+2 ];
[N,D] = rat2lmf(Num,Den);
[a,b,c,d,e] = lmf2dss(N,D)
a =
Columns 1 through 7
0     1.0000   -1.7638          0          0          0          0
0          0         0          0          0          0          0
0          0   -2.0000     1.0000          0          0          0
0          0         0          0          0          0          0
0          0         0          0   -10.0000          0          0
0          0   -0.0000          0          0   -20.0000          0
0          0         0          0          0          0     1.0000
0          0         0          0          0          0          0
0          0         0          0          0          0          0
0          0         0          0          0          0          0
Columns 8 through 10
0          0         0
0          0         0
0          0         0
0          0         0
0          0         0
0          0         0
0          0         0
1.0000     0         0
0          1.0000    0
0          0         1.0000
b =
     0          0          0          0          0     0.3333
0.3333     0.3333          0          0     0.3333          0
```

```
     0         0         0         0         0    0.3780
     0    0.3780         0         0         0         0
     0         0   -0.5754         0         0         0
     0         0         0   -0.5769         0         0
     0         0         0         0         0         0
     0         0         0         0    5.0000         0
     0         0         0         0         0         0
     0         0         0         0         0    1.0000
c =
Columns 1 through 7
3.0000    0         0         0         0         0         0
0         0    2.6458         0         0         0         0
0         0         0         0         0         0   -1.0000
0         0         0         0         0         0         0
-3.0000   0         0         0   17.3781         0         0
0         0   -2.6458         0         0   34.6699         0
Columns 8 through 10
0         0         0
0         0         0
0         0         0
0   -1.0000         0
0         0         0
0         0         0
d =
     0     0     0     0     0     0
     0     0     0     0     0     0
     0     0     0     0     1     0
     0     0     0     0     0     0
     0     0     0     0     0     0
```

```
     0     0     0     0     0     0

e =

     1     0     0     0     0     0     0     0     0     0
     0     1     0     0     0     0     0     0     0     0
     0     0     1     0     0     0     0     0     0     0
     0     0     0     1     0     0     0     0     0     0
     0     0     0     0     1     0     0     0     0     0
     0     0     0     0     0     1     0     0     0     0
     0     0     0     0     0     0     0     1     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     1
     0     0     0     0     0     0     0     0     0     0
```

The descriptor representation does not satisfy the regularity assumptions, This is corrected with the help of the command

```
[a1,b1,c1,d1,e1] = dssreg(a,b,c,d,e,2,2,'D22');
```

Following this the solution of the H_2 problem follows by typing

```
[yd,xd] = dss2rmf(ak,bk,ck,dk,ek);
```

We suppress the rather copious output. The compensator may be converted to rational form by the commands

```
[Y,X]          =          dss2rmf(ak,bk,ck,dk,ek);
[NumK,DenK] = rmf2rat(Y,X);
```

The compensator is given by

$$K(s)=\begin{bmatrix} \dfrac{0.45+2.6s+2.8s^2+1.1s^3+0.08s^4}{4.4+13s+17s^2+14s^3+5.6s^4+s^5} & \dfrac{-0.033-0.81s-0.56s^2-0.14s^3-0.0058s^4}{s(4.4+13s+17s^2+14s^3+5.6s^4+s^5)} \\ \dfrac{0.94+4.4s+8s^2+6.9s^3+2.3s^4+0.17s^5}{4.4+13s+17s^2+14s^3+5.6s^4+s^5} & \dfrac{1.9+3.4s+3s^2+1.7s^3+0.42s^4+0.017s^5}{s(4.4+13s+17s^2+14s^3+5.6s^4+s^5)} \end{bmatrix}$$

Algorithm The solution is obtained by a mixed state space and polynomial matrix solution (Kwakernaak, 2000).

Reference

Kwakernaak, H. (2000), "H_2-Optimization — Theory and Applications to Robust Control Design," Plenary paper, *IFAC Symposium on Robust Control Design 2000*, 21–23 June 2000, Prague, Czech Republic.

Diagnostics

The macro `dssh2` issues error messages if

- The input data have inconsistent dimensions
- The matrix *D* does not satisfy the regularity conditions
- The plant has unstable fixed poles
  - The generalized plant has marginally stable fixed poles that cannot be cancelled
  - The closed-loop transfer matrix cannot be made strictly proper

See also

`dssreg`	Regularization of a descriptor system
`h2`	Polynomial solution of the standard H_2 problem
`gare`	Solution of Generalized Algebraic Riccati Equations
`dsshinf`	H_∞ suboptimal compensator for descriptor systems
`mixeds`	Solution of a SISO H_∞ mixed sensitivity problem
`plqg`	Polynomial solution of a MIMO LQG problem
`splqg`	Polynomial solution of a SISO LQG problem

dssreg

Purpose "Regularization" of a standard descriptor plant

Syntax

```
[a,b,c,d,e] = dssreg(A,B,C,D,E,nmeas,ncon)
[a,b,c,d,e] = dssreg(A,B,C,D,E,nmeas,ncon[,tol][,option1][,option2])
```

Description The commands

```
[a,b,c,d,e] = dssreg(A,B,C,D,E,nmeas,ncon)
[a,b,c,d,e] = dssreg(A,B,C,D,E,nmeas,ncon,tol)
```

transform the generalized plant

$$E\dot{x} = Ax + B\begin{bmatrix} w \\ u \end{bmatrix}$$

$$\begin{bmatrix} z \\ y \end{bmatrix} = Cx + D\begin{bmatrix} w \\ u \end{bmatrix}$$

where the dimension of y is `nmeas` and the dimension of u is `ncon`, into an equivalent generalized plant

$$e\dot{x} = ax + b\begin{bmatrix} w \\ u \end{bmatrix}$$

$$\begin{bmatrix} z \\ y \end{bmatrix} = cx + d\begin{bmatrix} w \\ u \end{bmatrix}$$

with

```
d = [d11 d12
     d21 d22]
```

such that `d12` has full column rank and `d21` has full row rank. "Equivalent" means that the two plants have the same transfer matrices.

The optional tolerance parameter `tol` is used in the various rank tests. It has the default value `1e-12`.

Two options may be included. The option `'D11'` modifies the representation so that the term with d11 is absent. The option `'D22'` removes the term with d22.

In verbose mode the routine displays a relative error based on the largest of the differences of the frequency response matrices of the transformed and the original plant at the frequencies 1, 2, ..., 10.

Compatibility This version of `dssreg` is backward compatible with the version of Version 2.0 of the Toolbox. The only difference is that the options `'D11' and D22'` have been added.

Examples In the Example section of the manual page for the Polynomial Toolbox command `dsshinf` the descriptor representation of a generalized plant is derived. When considering the subsystem

$$z_2 = c(1+rs)u$$

two pseudo state variables are defined as $x_3 = u, \quad x_4 = \dot{u}$, which leads to the descriptor equations

$$\begin{aligned} \dot{x}_3 &= x_4 \\ 0 &= -x_3 + u \end{aligned}$$

The output equation is rendered as

$$z_2 = c(1+rs)u = crx_4 + cu$$

The output equation, however, equally well could be chosen as

$$z_2 = c(1+rs)u = cx_3 + crx_4$$

This brings the generalized plant in the form

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}}_{E} \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & 0 \end{bmatrix}}_{A} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \underbrace{\begin{bmatrix} \sqrt{2} & 0 \\ 1 & 1 \\ 0 & 0 \\ 0 & 1 \end{bmatrix}}_{B} \begin{bmatrix} w \\ u \end{bmatrix}$$

$$\begin{bmatrix} z_1 \\ z_2 \\ y \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & c & cr \\ -1 & 0 & 0 & 0 \end{bmatrix}}_{C} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \underbrace{\begin{bmatrix} 1 & 0 \\ 0 & 0 \\ -1 & 0 \end{bmatrix}}_{D} \begin{bmatrix} w \\ u \end{bmatrix}$$

For this plant we have

$$D_{12} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \qquad D_{21} = -1$$

so that D_{12} does not have full rank. We apply `dssreg` to this plant for $c = 0.1$, $r = 0.1$.

```
c = 0.1; r = 0.1;
E = [1 0 0 0; 0 1 0 0; 0 0 1 0; 0 0 0 0];
```

```
A = [0 1 0 0; 0 0 0 0; 0 0 0 1; 0 0 -1 0];
B = [sqrt(2) 0; 1 1; 0 0; 0 1];
C = [1 0 0 0; 0 0 c c*r; -1 0 0 0];
D = [1 0; 0 0; -1 0];
ncon = 1; nmeas = 1;
```

We now apply `dssreg`.

```
[a,b,c,d,e] = dssreg(A,B,C,D,E,nmeas,ncon)
a =
     0     1     0     0
     0     0     0     0
     0     0     0     1
     0     0    -1     0
b =
    1.4142         0
    1.0000    1.0000
         0    1.0000
         0    1.0000
c =
    1.0000         0         0         0
         0         0    0.1000    0.0100
   -1.0000         0         0         0
d =
    1.0000         0
         0    0.0100
   -1.0000         0
e =
     1     0     0     0
     0     1     0     0
     0     0     1     0
```

```
         0         0         0         0
```

We now have

$$D_{12} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad D_{21} = -1$$

so that the transformed plant is "regular."

As a second example we consider the standard plant

$$\dot{x} = x + \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} w \\ u \end{bmatrix}$$

$$\begin{bmatrix} z \\ y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} x$$

for which neither D_{12} nor D_{21} has full rank. We obtain the following result.

```
E = 1; A = 1; B = [1 1]; C = [1; 1]; D = [0 0;0 0];
nmeas = 1; ncon = 1;
[a,b,c,d,e] = dssreg(A,B,C,D,E,nmeas,ncon)
a =
     1     0     0
     0     1     0
     0     0     1
b =
     1     1
     0     1
     1     0
c =
     1     1     0
     1     0     1
d =
     0     1
     1     0
e =
```

```
     1     0     0
     0     0     0
     0     0     0
```

Instead of a state representation of dimension 1 we now have a 3-dimensional descriptor representation, which, however, is “regular.”

Finally, consider the system

$$\dot{x} = u + v, \quad z = v, \quad y = u$$

Accordingly, we let

```
» E
E =
     1
» A
A =
     0
» B
B =
     1     1
» C
C =
     0
     0
» D
D =
     1     0
     0     1
```

We successively have

```
» [a,b,c,d,e] = dssreg(A,B,C,D,E,1,1); length(a),d
ans =
     3
```

```
d =
     1     1
     1     1
» [a,b,c,d,e] = dssreg(A,B,C,D,E,1,1,'D11'); length(a),d
ans =
     4
d =
     0     1
     1     1
» [a,b,c,d,e] = dssreg(A,B,C,D,E,1,1,'D11','D22'); length(a),d
ans =
     5
d =
     0     1
     1     0
```

Algorithm

First consider the case that D_{21} does not have full row rank.

Let the rows of the matrix N span the left null space of E, so that $NE = 0$. Then by multiplying the descriptor equation $E\dot{x} = Ax + B_1 w + B_2 u$ on the left by N we obtain the set of algebraic equations $0 = NAx + NB_1 w + NB_2 u$. By adding suitable linear combinations of the rows of this set of equations to the rows of the output equation $y = C_2 x + D_{21} w + D_{22} u$ the rank of D_{21} may be increased without increasing the dimension of the pseudo state x.

If after this operation D_{21} still does not have full row rank then we apply a suitable transformation to the output equation $y = C_2 x + D_{21} w + D_{22} u$ so that it takes the form

$$y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} C_{21} \\ C_{22} \end{bmatrix} x + \begin{bmatrix} D_{211} \\ 0 \end{bmatrix} w + \begin{bmatrix} D_{221} \\ D_{222} \end{bmatrix} u$$

where D_{211} has full row rank. It is easy to construct a matrix D_{212} so that

$$\begin{bmatrix} D_{211} \\ D_{212} \end{bmatrix}$$

has full row rank. Following this we redefine the output equation as

$$y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} C_{21} \\ C_{22} \end{bmatrix} x + \begin{bmatrix} 0 \\ I \end{bmatrix} x' + \begin{bmatrix} D_{211} \\ D_{212} \end{bmatrix} w + \begin{bmatrix} D_{221} \\ D_{222} \end{bmatrix} u$$

where x' is an additional component of the pseudo state. This component is accounted for by adding the algebraic equation

$$0 = x' + D_{212} w$$

to the descriptor equation. This increases the dimension of the pseudo state, of course.

If D_{12} does not have full rank then the procedure as described is applied to the "dual" system.

If, say, D_{22} is nonzero then we use singular value decomposition to write $D_{22} = USV$, where S is square nonsingular. Adding the equation $x' = SVu$ to the descriptor equations we may now rewrite the equation for y as

$$y = C_1 x + D_{21} v + D_{22} u = C_1 x + Ux' + D_{21} v$$

If needed, D_{11} is similarly removed.

Diagnostics

The macro `dssreg` displays error messages in the following situations.

- The input parameters have inconsistent dimensions.
- D_{12} is not tall or D_{21} is not wide.
- The relative error exceeds `1e-6`. The relative error is computed on the basis of the largest of the differences of the frequency responses of the system before and after regularization at the frequencies 1, 2, …, 10.

In verbose mode the relative error is always reported.

See also

`dssrch`	H_∞ optimization for a descriptor plant
`dssmin`	dimension reduction of a descriptor system
`dssh2`	H_2 optimization of a descriptor system

gare

Purpose Generalized algebraic Riccati equation

Syntax `[X,F] = gare(A,B,C,D,E,Q,R,tol);`

Description This routine computes the solution X of the generalized algebraic Riccati equation

$$X^T A + A^T X + C^T QC - (X^T B + C^T D)R^{-1}(B^T X + D^T C) = 0$$
$$X^T E = E^T X$$

and the gain

$$F = R^{-1}(B^T X + D^T C)$$

such that the feedback law $u = -Fx$ stabilizes the descriptor system

$$E\dot{x} = Ax + Bu$$

and makes the impulsive modes non-impulsive. Finite closed-loop poles on the imaginary axis are allowed. If the descriptor system is not stabilizable or impulse controllable then X is returned as a matrix filled with `Infs`. In this case F still has a well-defined solution. The corresponding feedback law stabilizes the stabilizable modes and makes the controllable impulsive modes non-impulsive.

The optional tolerance `tol` is used by the routine `clements` and also to test whether the GARE has a finite solution. Its default value is `1e-12`.

Compatibility This function is new in the Polynomial Toolbox.

Example Let

$$E = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad A = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}, \quad B = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$
$$C = \begin{bmatrix} 2 & 1 \end{bmatrix}, \quad D = 1$$

$Q = 1$, $R = 1$. The system has an uncontrollable mode with eigenvalue 2. Accordingly, we obtain

```
» [X,F] = gare(A,B,C,D,E,Q,R)
X =
    Inf    Inf
    Inf    Inf
F =
      2      0
```

```
» roots(s*E-A+B*F)
ans =
    2.0000
   -1.0000
```

On the other hand,

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

$$C = \begin{bmatrix} 0 & 1 \end{bmatrix}, \qquad D = 1$$

is controllable. We now have

```
» [X,F] = gare(A,B,C,D,E,Q,R)
X =
  1.0e-015 *
         0          0
   -0.0754    -0.1601
F =
   -0.0000     1.0000
» roots(s*E-A+B*F)
ans =
   -1.0000
```

Algorithm The algorithm for the solution of the GARE relies on transforming the associated Hamiltonian pencil to Clements form (Kwakernaak, 2000).

Reference Kwakernaak, H. (2000), “H_2-Optimization — Theory and Applications to Robust Control Design,” Plenary paper, *IFAC Symposium on Robust Control Design 2000*, 21–23 June 2000, Prague, Czech Republic.

Diagnostics The macro `gare` issues error messages if the input data have inconsistent dimensions.

See also

`clements` Clements transformation of a matrix pencil

h2

Purpose H2-optimization

Syntax `[Y,X,clpoles,fixed] = H2(N,D,nmeas,ncon[,tol][,'check'])`

Description Given a continuous-time generalized plant of the form

$$\begin{bmatrix} z \\ y \end{bmatrix} = \underbrace{\begin{bmatrix} G_{11} & G_{12} \\ G_{21} & G_{22} \end{bmatrix}}_{G} \begin{bmatrix} w \\ u \end{bmatrix}$$

with G represented in the left coprime polynomial matrix fraction form $G = D^{-1}N$, the command

```
[Y,X,clpoles,fixed] = H2(N,D,ncon,nmeas)
```

computes the compensator $u = Ky$ in right polynomial matrix fraction form $K = YX^{-1}$ that minimizes the 2-norm $\|H\|_2$ of the closed-loop transfer matrix

$$H = G_{11} + G_{12}(I - KG_{22})^{-1} KG_{21}$$

from v to z. The norm is defined by

$$\|H\|_2^2 = \frac{1}{2\pi} \text{tr} \int_{-\infty}^{\infty} H^T(-j\omega) H(j\omega)\, d\omega$$

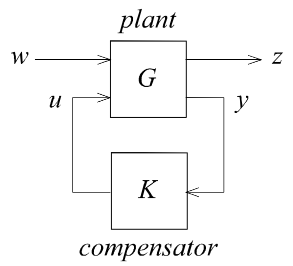

Fig. 2 . Generalized plant

The input parameter `ncon` is the number of control inputs and `nmeas` is the number of measured outputs. The output parameter `clpoles` contains the (non-fixed) closed-loop poles and `fixed` the fixed-plant poles.

In the optional forms

```
[Y,X,clpoles,fixed] = H2(N,D,ncon,nmeas,tol)
[Y,X,clpoles,fixed] = H2(N,D,ncon,nmeas,'check')
[Y,X,clpoles,fixed] = H2(N,D,ncon,nmeas,tol,'check')
```

the parameter `tol` is a tolerance. Its default value is 1e-8. If the option `'check'` is present then the routine checks whether the H_2 optimization problem has a solution, and exits if no solution exists. If the option is not invoked then the routine produces a solution even if none exists. In the latter case the closed-loop transfer matrix either has poles on the imaginary axis or is not strictly proper.

Nonproper generalized plants are allowed. Fixed open-loop poles (that is, uncontrollable or unobservable poles) cannot have strictly positive real parts but may be located on the imaginary axis.

Compatibility This function is new in the Polynomial Toolbox.

Examples Example 1: H_2 *design problem*

Consider the block diagram of Fig. 1. The plant is a MIMO system with transfer matrix

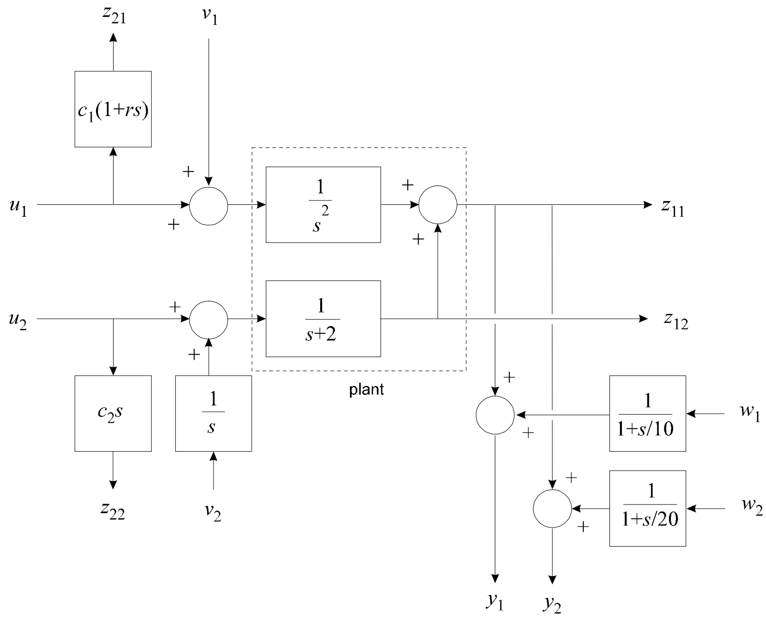

Fig. 3. Design problem

$$P(s)=\begin{bmatrix}\frac{1}{s^2} & \frac{1}{s+2}\\ 0 & \frac{1}{s+2}\end{bmatrix}$$

The controlled output is

$$z=\begin{bmatrix}z_{11}\\ z_{12}\end{bmatrix}$$

The measured output

$$y=\begin{bmatrix}y_1\\ y_2\end{bmatrix}$$

is corrupted by colored measurement noise generated by the two shaping filters with transfer functions

$$\frac{1}{1+s/10} \quad \text{and} \quad \frac{1}{1+s/20}$$

The second component of the disturbance

$$v=\begin{bmatrix}v_1\\ v_2\end{bmatrix}$$

is passed through a shaping filter with transfer function $1/s$ to ensure integrating action on both input channels. The input

$$u=\begin{bmatrix}u_1\\ u_2\end{bmatrix}$$

is weighted with dynamic weighting functions with transfer functions $c_1(1+rs)$ (to ensure sufficient high-frequency roll-off of the compensator) and $c_2 s$ (both for high-frequency roll-off and to allow integral control at the second input channel).

The generalized plant that defines the H_2 problem is given by

$$G(s)=\begin{bmatrix} G_{11}(s) & G_{12}(s) \\ G_{21}(s) & G_{22}(s) \end{bmatrix}=\left[\begin{array}{cccc|cc} \frac{1}{s^2} & \frac{1}{s(s+2)} & 0 & 0 & \frac{1}{s^2} & \frac{1}{s+2} \\ 0 & \frac{1}{s(s+2)} & 0 & 0 & 0 & \frac{1}{s+2} \\ 0 & 0 & 0 & 0 & c_1(1+rs) & 0 \\ 0 & 0 & 0 & 0 & 0 & c_2 s \\ \hline \frac{1}{s^2} & \frac{1}{s(s+2)} & \frac{1}{1+s/10} & 0 & \frac{1}{s^2} & \frac{1}{s+2} \\ 0 & \frac{1}{s(s+2)} & 0 & \frac{1}{1+s/20} & 0 & \frac{1}{s+2} \end{array}\right]$$

The transfer matrix may be entered in rational format and then converted to a left polynomial matrix fraction by the command `rat2lmf`:

```
c1 = 1; c2 = 1; r = 5;
Num = [ 1  1  0  0      1        1
        0  1  0  0      0        1
        0  0  0  0 c1*(1+r*s)   0
        0  0  0  0      0      c2*s
       -1 -1 -1  0     -1       -1
        0 -1  0 -1      0       -1 ];
Den = [ s^2 s*(s+2)    1       1    s^2   s+2
         1  s*(s+2)    1       1     1    s+2
         1     1       1       1     1     1
         1     1       1       1     1     1
        s^2 s*(s+2) 1+s/10     1    s^2   s+2
         1  s*(s+2)    1    1+s/20   1    s+2 ];
[N,D] = rat2lmf(Num,Den)
N =
     0      0      0       0       0.2 + s      0
     0      0      0       0       0            s
     1            20 + s          0        0      0            20 + s
```

The solution of the H_2 problem follows by typing

```
[Y,X,clpoles,fixedpoles] = H2(N,D,2,2)
Y =
     0.044 + 0.25s + 0.023s^2     0.014 + 0.042s + 0.0022s^2
     0.54 + 0.4s + 0.04s^2       -0.3 - 0.15s - 0.0065s^2
X =
   0.41 + s + 0.76s^2 + 0.27s^3      0.13 - 0.12s - 0.064s^2
  -0.21 + 0.16s + 0.09s^2 - 0.27s^3  -0.065 - 0.83s - s^2 - 0.38s^3
clpoles =
  -1.8586
  -1.8477
  -0.7541 + 0.6556i
  -0.7541 - 0.6556i
  -0.2991 + 0.4534i
  -0.2991 - 0.4534i
  -0.8073
  -0.5388
  -0.4545
fixedpoles =
         0
  -20.0000
  -10.0000
```

The compensator may be converted to rational form by the command

```
[NumK,DenK] = rmf2rat(Y,X)
```

The compensator is given by

$$K(s) = \begin{bmatrix} \dfrac{0.45+2.6s+2.8s^2+1.1s^3+0.08s^4}{4.4+13s+17s^2+14s^3+5.6s^4+s^5} & \dfrac{-0.033-0.81s-0.56s^2-0.14s^3-0.0058s^4}{s(4.4+13s+17s^2+14s^3+5.6s^4+s^5)} \\ \dfrac{0.94+4.4s+8s^2+6.9s^3+2.3s^4+0.17s^5}{4.4+13s+17s^2+14s^3+5.6s^4+s^5} & \dfrac{1.9+3.4s+3s^2+1.7s^3+0.42s^4+0.017s^5}{s(4.4+13s+17s^2+14s^3+5.6s^4+s^5)} \end{bmatrix}$$

Inspection shows that integrating action is included in the second input channel as intended.

Example 2: *Wiener filtering problem*

Wiener filtering problems may be defined as follows. A message signal *x* is given by

$$x = H_1(s)v$$

where *v* is a standard white noise process. The observed signal *y* is related to the message process by

$$y = H_2(s)v$$

H_1 and H_2 are stable rational transfer matrices. It is desired to estimate the message signal *x* by filtering the observed signal *y*.

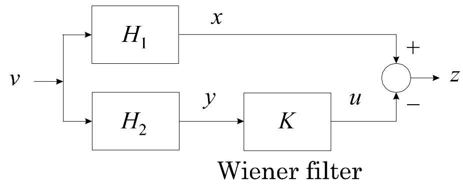

Fig. 4. Wiener filter configuration

Fig. 4 shows the system configuration. Inspection shows that the generalized plant that defines the H_2-problem is given by

$$\begin{bmatrix} z \\ y \end{bmatrix} = \underbrace{\begin{bmatrix} H_1 & -I \\ H_2 & 0 \end{bmatrix}}_{G} \begin{bmatrix} v \\ u \end{bmatrix}$$

By way of example, suppose that *x* and *y* are related as

$$y = x + n$$

where the observation noise *n* is independent of the message signal *x*. The message signal is generated by the shaping filter

$$x = \frac{1}{(s+1)^2} v_1$$

with v_1 white noise, and the noise is given by

$$n = \frac{\omega_o^2}{s^2 + 2\zeta\omega_o s + \omega_o^2} \sigma v_2$$

where the white noise v_2 is independent of v_1. We let $\omega_o = 1$, $\zeta = 0.01$ and $\sigma = 0.1$ so that the measurement noise is not very large but has a relatively sharp peak at the cut-off frequency of the message signal. This defines

$$H_1(s) = \begin{bmatrix} \frac{1}{(s+1)^2} & 0 \end{bmatrix}$$

$$H_2(s) = \begin{bmatrix} \frac{1}{(s+1)^2} & \frac{\omega_o^2 \sigma}{s^2 + 2\zeta\omega_o s + \omega_o^2} \end{bmatrix}$$

so that

$$G(s) = \left[\begin{array}{cc|c} \frac{1}{(s+1)^2} & 0 & -1 \\ \hline \frac{1}{(s+1)^2} & \frac{\omega_o^2 \sigma}{s^2 + 2\zeta\omega_o s + \omega_o^2} & 0 \end{array}\right]$$

The following commands solve this problem:

```
d1 = (s+1)^2; n1 = 1;
omo = 1; zeta = 0.01; sigma = 0.1;
d2 = s^2+2*zeta*omo*s+omo^2; n2 = omo^2*sigma;
Num = [ 1  0 -1
        1  n2   0 ];
Den = [ d1 1  1
        d1 d2 1 ];
[N,D] = rat2lmf(Num,Den);
[Y,X,clpoles] = H2(N,D,1,1)
```

The solution is returned as

```
Y =
     0.91 + 0.018s + 0.91s^2
X =
     1 + 0.2s + s^2
clpoles =
   -0.1000 + 0.9950i
```

```
   -0.1000 - 0.9950i
```

The command

```
bode(pol2mat(Y),pol2mat(X))
```

produces the Bode plot of the filter of Fig. 5. The filter is a notch filter that removes the colored measurement noise as best as it can.

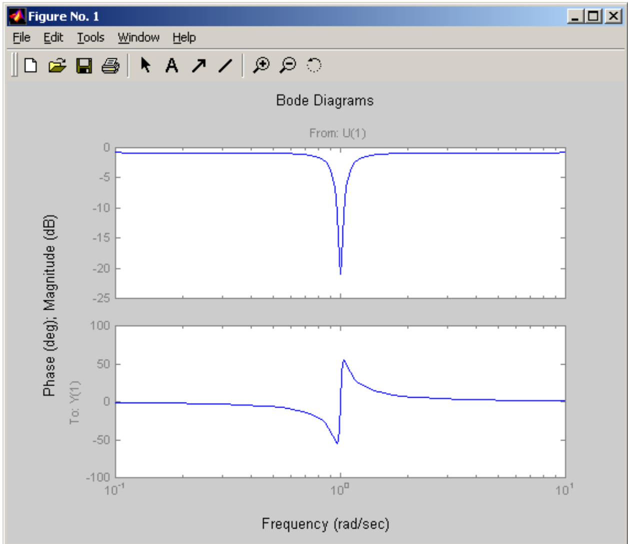

Fig. 5. Bode diagram of the Wiener filter

Algorithm

The solution is obtained by Wiener-Hopf optimization (Kwakernaak, 2000).

Reference

Kwakernaak, H. (2000), “H_2-Optimization — Theory and Applications to Robust Control Design,” Plenary paper, *IFAC Symposium on Robust Control Design 2000*, 21–23 June 2000, Prague, Czech Republic.

Diagnostics

The macro `h2` issues error messages if

- G_{12} does not have full column rank or G_{21} does not have full row rank
- The generalized plant has unstable fixed poles

If the option `'check'` is activated then error messages are issued if

- The plant has fixed poles on the imaginary axis that cannot be canceled
- The closed-loop system transfer matrix cannot be made strictly proper

Warning messages are issued if

- N_{21} or $\bar{N}_{12}$ have zeros on the imaginary axis
- The closed-loop system has one or more poles on the imaginary axis

The polynomial matrices N_{21} and $\bar{N}_{12}$ occur in the left en right coprime fractional representations

$$\begin{bmatrix} G_{21} & G_{22} \end{bmatrix} = D_{22}^{-1} \begin{bmatrix} N_{21} & N_{22} \end{bmatrix}, \qquad \begin{bmatrix} G_{12} \\ G_{22} \end{bmatrix} = \begin{bmatrix} \bar{N}_{12} \\ \bar{N}_{22} \end{bmatrix} \bar{D}_{22}^{-1}$$

If these polynomial matrices have roots on the imaginary axis then the two spectral factorizations will also involve roots on the imaginary axis, which may make the factorizations fail.

See also

`dsshinf`	H_∞ suboptimal compensator for descriptor systems
`mixeds`	Solution of a SISO H_∞ mixed sensitivity problem
`plqg`	Polynomial solution of a MIMO LQG problem
`splqg`	Polynomial solution of a SISO LQG problem

jury

Purpose Create the Jury matrix corresponding to a polynomial

Syntax

```
J = jury(p,k)
J = jury(p,k,'rev')
```

Description The command

```
J = jury(p,k))
```

creates the constant Jury matrix J of dimension $(k-1)\times(k-1)$ corresponding to the polynomial p. If

$$p(v) = p_0 + p_1 v + \cdots + p_d v^d$$

and $k \le d$ then

$$J(p) = \begin{bmatrix} p_k & p_{k-1} & p_{k-2} & \cdots & p_3 & p_2 \\ 0 & p_k & p_{k-1} & \cdots & p_4 & p_3 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & 0 & p_k & p_{k-1} \\ 0 & 0 & 0 & 0 & 0 & p_k \end{bmatrix} - \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 & p_0 \\ 0 & 0 & 0 & \cdots & p_0 & p_1 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & p_0 & p_1 & \cdots & p_{k-4} & p_{k-3} \\ p_0 & p_1 & p_2 & \cdots & p_{k-3} & p_{k-2} \end{bmatrix}$$

The default value of k is d.

With the syntax

```
J = jury(p,k,'rev')
```

the coefficients $p_0, p_1, \ldots, p_k$ are reversed.

The Jury matrix is quite useful when analysing the robust stability of discrete-time systems by polynomial methods. In the Polynomial Toolbox the function is called by the function `stabint` when computing stability interval.

Compatibility This function is new in the Polynomial Toolbox.

Examples The Jury matrix of the polynomial

```
P = 1+z+2*z^2+3*z^3+4*z^4+5*z^5
p =
     1 + z + 2z^2 + 3z^3 + 4z^4 + 5z^5
```

simply is

```
J = jury(p)
```

```
J =

     5     4     3     1
     0     5     3     2
     0    -1     4     2
    -1    -1    -2     2
```

A version of reduced size is obtained by typing

```
J4 = jury(p,4)
J4 =

     4     3     1
     0     3     2
    -1    -1     2
```

Algorithm The macro uses standard MATLAB operations.

References R. Barmish: *New Tools for Robustness of Linear Systems.* Macmillan Publishing Company. New York, 1994.

Diagnostics The function displays no error messages.

See also

`hurwitz`	Hurwitz matrix for a polynomial
`stabint`	robust stability interval

pdisp

Purpose Display a polynomial or polynomial matrix without its name

Syntax `pdisp(M)`

Description The command

```
pdisp(M)
```

displays the polynomial matrix M without printing the name.

Example Typing

```
M = [s+1 s^2+s]
M =
       1 + s       s + s^2
```

displays the matrix M including its name. The name is suppressed by typing

```
pdisp(M)
   1 + s       s + s^2
```

Compatibility This command is new in the Polynomial Toolbox.

Algorithm The macro uses standard MATLAB commands.

pformat rootr, pformat rootc

Purpose Format a polynomial or polynomial entry

Syntax

```
pformat rootr
pformat rootc
```

Description The Polynomial Toolbox can display polynomial matrices in various formats under the control of the command `pformat`. In addition to the formats available in Version 2.0 (see the Manual for details) two more options were included in Version 2.5 that allow to display polynomials in terms of their roots.

By specifying the format `rootr` a polynomial with real coefficients is expressed as a product of first and second order factors. Every real root results in real factor of degree 1 while a pair of complex conjugate roots results in a real factor of degree 2.

The other new display format `rootc` returns a product of factors of degree 1 only. A pair of complex conjugate roots now results in a pair of first degree factors with complex coefficients.

For polynomials with complex coefficients the two new formats display identical results with factors of degree 1 only.

For polynomial matrices the formats apply to each of the entries.

Compatibility In Version 2.0 use of the options results in an error message.

Examples By way of example, create a simple polynomial

```
P = (s-1)*(s+2)*(s+3*i)*(s-3*i);
```

In the default display format `symbs` the polynomial is displayed as

```
p
p =
    -18 + 9s + 7s^2 + s^3 + s^4
```

Changing the display format to `rootr` results in

```
pformat rootr
p
p =
     (s+2.0000)(s^2+9.0000)(s-1)
```

The other new display format `rootc` returns a product of factors of degree 1 only. A pair of complex conjugate roots now results in a pair of first degree factors with complex coefficients:

```
pformat rootc
p
p =
     (s+2.0000)(s+3.0000i)(s-3.0000i)(s-1)
```

To view the effect on polynomial matrices consider

```
pformat rootr
[s^2+2*s+1 s+s^2]
ans =
     (s+1)(s+1)     s(s+1)
```

Algorithm The routine uses standard MATLAB operations.

See also `pformat` Control the display format of polynomials and polynomial matrices

pol2tex

Purpose Conversion of a polynomial object into LaTeX code

Syntax

```
Tex_str = pol2tex(A1,A2,…,AN)
Tex_str = pol2tex(A1,A2,…,AN,'File_name')
```

Description The command

```
Tex_str = pol2tex(A1,A2,…,AN)
```

converts the polynomial matrices or standard MATLAB matrices `A1,A2,…,AN` into a string `Tex_str` in LaTeX code to be used in LaTeX source files. LaTeX is a well-known document preparation system that is especially effective for text containing many mathematical formulas including matrices [1]. The output string comprises a sequence of LaTeX commands to create an array surrounded by bracket delimiters in display mathematical mode. The user is expected to copy this string to a LaTeX source file.

Alternatively, the command

```
Tex_str = pol2tex(A1,A2,…,AN,'File_name')
```

appends `Tex_str` to the existing file `File_name.tex`. If the file does not exist then the output string is saved in a newly created TEX file. This file, however, does not contain any LaTeX preamble and hence cannot be compiled by LaTeX as it is. Instead, it can be related to another TEX file using LaTeX command `input` or the `include` statement.

The macro allows any number of input arguments. The resulting format is given by the currently active display format, which is controlled by the functions `pformat` and `format`.

Compatibility This function is new in the Polynomial Toolbox.

Examples The following examples illustrate how the command should be used.

Example 1

Consider a polynomial matrix C given by

```
» C=[-8+s 1-6*s 6+6*s; 0 2 1; -1+4*s-3*s^2 2.1e-5 11-s]
C =
      -8 + s            1 - 6s       6 + 6s
       0                2            1
      -1 + 4s - 3s^2    2.1e-005     11 - s
```

which should be included in a LaTeX based document. Calling `pol2tex` creates the string

```
» pol2tex(C)
```

```
ans =
$$
C=
\left[ \begin{array}{lll}
-8+s & \;\;\;1-6s & \;\;\;6+6s \\
\;\;\;0 & \;\;\;2 & \;\;\;1 \\
-1+4s-3s^{2} & \;\;\;2.1*10^{-5}  & \;\;\;11-s
\end{array} \right]
$$
```

This string may be further edited if necessary. If the string is copied into an existing LaTeX file and compiled by LaTeX then one gets the fairly nice result

$$C = \left[\begin{array}{ccc} -8+s & 1-6s & 6+6s \\ 0 & 2 & 1 \\ -1+4s-3s^2 & 2.1\times 10^{-5} & 11-s \end{array}\right]$$

Example 2

As another example consider a constant matrix B

```
B =
    0.2200   -0.3333    0.1222    4.0000
   -0.6364    8.0000    0.0927    0.4000
```

and change the format to *rational*

```
» format rat
» B
B =
    11/50        -1/3         11/90          4
    -7/11          8          29/313        2/5
```

Then

```
» pol2tex(B)
ans =
$$
```

```
B=
\left[ \begin{array}{llll}
 \;\;\; \frac{11}{50} & -\frac{1}{3} &  \;\;\; \frac{11}{90} &
\;\;\; 4 \\ \\
-\frac{7}{11} &  \;\;\; 8 &  \;\;\; \frac{29}{313} &  \;\;\;
\frac{2}{5}
\end{array} \right]
$$
```

LaTeX returns this as

$$B = \left[\begin{array}{llll} \;\;\; \frac{11}{50} & -\frac{1}{3} & \;\;\; \frac{11}{90} & \;\;\; 4 \\ \\ -\frac{7}{11} & \;\;\; 8 & \;\;\; \frac{29}{313} & \;\;\; \frac{2}{5} \end{array} \right]$$

Algorithm The macro uses standard MATLAB 5 operations.

Diagnostics The macro displays error messages if

- There are not enough input arguments.
- The class of the input argument is inappropriate.

References Leslie Lamport, *LaTeX: A Document Preparation System.* Addison-Wesley, Reading, Massachusetts, 1994.

See also

`char`	Convert a polynomial object to a string
`pformat`	Set the output format for a polynomial object

psseig

Purpose Polynomial approach to eigenstructure assignment for a state-space system

Syntax `L = psseig(F,G,P[,TOL])`

Description Given a linear system

$$\dot{x} = Fx + Gu$$

where F is an $n \times n$ constant matrix and G is an $n \times m$ constant matrix, and a set of polynomials $P = \{p_1(s), p_2(s), \ldots, p_r(s)\}$, $r \le m$, the command

```
L = psseig(F,G,P)
```

returns, if possible, a constant matrix L such that the closed-loop matrix of the controlled system

$$\dot{x} = (F - GL)x$$

has invariant polynomials $q_1(s), q_2(s), \ldots, q_n(s)$, where

$$\begin{aligned} q_1(s) &= p_1(s)q_2(s) \\ q_2(s) &= p_2(s)q_3(s), \\ &\vdots \\ q_r(s) &= p_r(s), \\ q_{r+1}(s) &= \cdots = q_n(s) = 1 \end{aligned}$$

Such a matrix exists if and only if the fundamental degree inequality

$$\deg q_1 + \deg q_2 + \cdots \deg q_k \ge c_1 + c_2 + \cdots c_k$$

holds for all $k = 1, 2, \ldots, r$, where $c_1 \ge c_2 \ge \cdots \ge c_r$ are the controllability indices of the pair (F,G). Moreover, equality must hold for $k = r$. If the input polynomials P do not satisfy these conditions then the macro issues an error message.

A tolerance `TOL` may be specified as an additional input argument. Its default value is the global zeroing tolerance.

Compatibility This function is new in the Polynomial Toolbox.

Examples The dynamics of an inverted pendulum linearized about the equilibrium position are described by the equation

$$\dot{x} = Fx + Gu$$

where

$$F=\begin{bmatrix}0 & 1 & 0 & 0\\ 10.7800 & 0 & 0 & 0\\ 0 & 0 & 0 & 1\\ -0.9800 & 0 & 0 & 0\end{bmatrix},\quad G=\begin{bmatrix}0\\ -0.2000\\ 0\\ 0.2000\end{bmatrix}$$

The desired closed-loop poles are selected as

$$-1\pm j$$
$$-2\pm 2j$$

This yields the invariant polynomial

$$\psi_1(s)=s^4+6s^3+18s^2+24s+16$$

Since $m=1$, one has

$$\psi_2(s)=\psi_3(s)=\psi_4(s)=1.$$

We aim to find a feedback gain matrix L so that the state feedback law $u=-Lx$ assigns these invariant polynomials to the closed loop system matrix $F-GL$. The corresponding code is as follows:

```
F=[0,1,0,0;10.78,0,0,0;0,0,0,1;-0.98,0,0,0]
F =
         0    1.0000         0         0
   10.7800         0         0         0
         0         0         0    1.0000
   -0.9800         0         0         0
G=[0;-0.2000;0;0.2000]
G =
         0
   -0.2000
         0
    0.2000
P = s^4 + 6*s^3 + 18*s^2 + 24*s + 16;
L=psseig(F,G,P)
Constant polynomial matrix: 1-by-4
L =
```

```
    -1.5e+002    -42    -8.2    -12
det(s*eye(4)-F+G*L)-P
Zero polynomial matrix: 1-by-1,  degree: -Inf
ans =
     0
```

Algorithm The algorithm is fully described in reference [1]. It may be summarized as follows:

1. A coprime matrix polynomial fraction description $A_r(s), B_r(s)$ is computed for the system with the macro `ss2rmf`.
2. The controllability indices (the column degrees of $A_r(s)$) are sorted and the fundamental degree condition for invariant polynomials assignment is checked.
3. A polynomial matrix $C_r(s)$ featuring the controllability indices and the desired invariant polynomial factors is built.
4. The Diophantine equation $X_L(s)A_r(s) + Y_l(s)B_r(s) = C_r(s)$ is solved for a constant solution X_L, Y_l with the macro `xaybc`.
5. The constant feedback matrix $L = X_L^{-1}Y_l$ $ is constructed.

References

[1] V. Kucera, M. Sebek, D. Henrion: “Polynomial Toolbox and State Feedback Control.” *Proceedings of the IEEE International Symposium on Computed-Aided Control* System Design, IEEE, pp. 380–385, Kohala Coast, Hawaii, August 1999.

Diagnostics The macro produces error messages if

- the input matrices have incompatible dimensions
- there is an incorrect number of invariant polynomials
- some invariant polynomial is zero
- the fundamental degree condition is not satisfied

See also

`psslqr`	Polynomial approach to linear-quadratic regulator design for state-space systems

psslqr

Purpose Polynomial approach to linear-quadratic regulator design for state-space systems

Syntax

```
L = psslqr(F,G,H,J[,TOL])
```

Description Given a linear system

$$\dot{x} = Fx + Gu$$

where F is an $n \times n$ constant matrix and G is an $n \times m$ constant matrix, and a regulated variable

$$z = Hx + Ju$$

where H is a $p \times n$ constant matrix and J is a $p \times m$ constant matrix, the command

```
L = psslqr(F, G, H, J)
```

returns a constant matrix L such that the control function $u = -Lx$ minimizes the L_2-norm of z for every initial state $x(0)$.

It is assumed that

$$J^T H = 0, \;\; J^T J = I$$

A tolerance `TOL` may be specified as an additional input argument. Its default value is the global zeroing tolerance.

Compatibility This function is new in the Polynomial Toolbox.

Examples The linearized model of the vertical-plane dynamics of an AIRC aircraft is described by the equations

$$\dot{x} = Fx + G_L v$$
$$y = Hx + J_L v$$

where

$$F = \begin{bmatrix} 0 & 0 & 1.1320 & 0 & -1 \\ 0 & -0.0538 & -0.1712 & 0 & 0.0705 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0.0485 & 0 & -0.8556 & -1.013 \\ 0 & -0.2909 & 0 & 1.0532 & -0.6859 \end{bmatrix},$$

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

We want to design a linear Gaussian filter for the covariance matrices given by

$$G_L = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ -0.1200 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 4.4190 & 0 & -1.6650 & 0 & 0 & 0 \\ 1.5750 & 0 & -0.0732 & 0 & 0 & 0 \end{bmatrix},$$

$$J_L = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

The corresponding code is as follows:

```
F = [0,0,1.1320,0,-1;0,-0.0538,-.1712,0,0.0705;0,0,0,1,0;
0,0.0485,0,-0.8556,-1.0130;0,-0.2909,0,1.0532,-0.6859]
F =
         0         0    1.1320         0   -1.0000
         0   -0.0538   -0.1712         0    0.0705
         0         0         0    1.0000         0
         0    0.0485         0   -0.8556   -1.0130
         0   -0.2909         0    1.0532   -0.6859
H = [1,0,0,0,0;0,1,0,0,0;0,0,1,0,0]
H =
     1     0     0     0     0
     0     1     0     0     0
     0     0     1     0     0
GL = [0,0,0,0,0,0;-
0.12,1,0,0,0,0;0,0,0,0,0,0;4.4190,0,1.665,0,0,0;
1.575,0,-0.0732,0,0,0]
GL =
         0         0         0         0         0         0
   -0.1200    1.0000         0         0         0         0
         0         0         0         0         0         0
```

```
    4.4190         0    1.6650         0         0         0
    1.5750         0   -0.0732         0         0         0
JL = [0,0,0,1,0,0;0,0,0,0,1,0;0,0,0,0,0,1]
JL =
     0     0     0     1     0     0
     0     0     0     0     1     0
     0     0     0     0     0     1
L = psslqr(F',H',GL',JL')
Constant polynomial matrix: 3-by-5
L =
     1          0.066    -0.21     -0.45     -0.81
     0.066      0.94     -0.069    -0.053    -0.25
    -0.21      -0.069     1.8       1.6       2.2
```

Algorithm The algorithm is fully described in reference [1]. It may be summarized as follows:

1. A coprime matrix polynomial fraction description $A_r(s), B_r(s)$ is computed for the system with the macro `ss2rmf`.
2. A polynomial matrix $C_r(s)$ with L_2 -optimal eigenstructure is computed with the spectral factorization macro `spf`.
3. The Diophantine equation $X_L(s)A_r(s) + Y_l(s)B_r(s) = C_r(s)$ is solved for a constant solution X_L, Y_l with the macro `xaybc`.
4. The constant feedback $L = X_L^{-1}Y_l$ is constructed.

Reference [1] V. Kucera, M. Sebek, D. Henrion: "Polynomial Toolbox and State Feedback Control." *Proceedings of the IEEE International Symposium on Computed-Aided Control* System Design, IEEE, pp. 380–385, Kohala Coast, Hawaii, August 1999.

Diagnostics The macro produces error messages if

- the input matrices have incompatible dimensions
- the orthogonality condition on covariance matrices does not hold

See also `psseig` Polynomial approach to eigenstructure assignment for state-space system

sarea, sareaplot

Purpose Robust stability area for polynomials with parametric uncertainties

Syntax

```
S = sarea(q1,…,qm,ExpressionString,p0,p1,…,pn[,tol])
sareaplot(q1,S,[,PlotType][,'new'])
sareaplot(q1,q2,S,[,PlotType][,'new'])
sareaplot(q1,q2,q3,S,[,PlotType][,'new'])
```

Description The command

```
S = sarea(q1,…,qm,ExpressionString,p0,p1,…,pn[,tol])
```

investigates the robust stability of a family of polynomials depending on m uncertain parameters. By gridding the parameter space the family is split into a number of standard polynomials that are separately checked for stability. The m-dimensional grid is defined by the vectors of parameter values $q_1,\ldots,q_m$, and the results are stored in an m-dimensional array S structured accordingly. As expected, in S 1s stand for "stable" and 0s for "unstable."

For details on checking the stability of a single polynomial please read the description of macro `isstable`. The parameters $p_0,\ldots,p_n$ are given fixed polynomials that serve to define the uncertainty structure. Note that the input arguments representing both the parameters and the fixed polynomials must be written using their names (rather than values) in the function call.

The uncertainty structure of the polynomial family is defined by the string variable `ExpressionString`. This string may contain any MATLAB-like expression composed of the parameter names (acting here as scalars) and of the names of the fixed polynomials.

The procedure is better explained in the examples below. For three or more uncertain parameters dense gridding may result in slow performance. Typing

```
verbose yes
```

before the run activates an on-line info on the macro performance.

Once the array S is available, it may be plotted by typing one of

```
sareaplot(q1,S,[,PlotType][,'new'])
sareaplot(q1,q2,S,[,PlotType][,'new'])
sareaplot(q1,q2,q3,S,[,PlotType][,'new'])
```

for the case of one, two or three parameters, respectively. As before the parameters q_1,q_2,q_3 must be typed by names and not by values. The optional argument `PlotType` specifies the type of plot. It may be a surface plot (default or `PlotType='surf'`), a point plot (`PlotType='points'`), or a combination of the two (`PlotType='both'`). The surface plot is usually nicer but may miss some details, while the point plot is always complete. With the input string argument `'new'` the plot is displayed in a new window.

Compatibility

These functions are new in the Polynomial Toolbox.

Examples

The following examples illustrate how the command should be used.

Example 1

Consider an uncertain polynomial

$$p(s,q_1,q_2) = p_0(s) + (q_1 + q_2)p_1(s) + \sqrt{|q_2|}\,p_2(s)$$

composed of three fixed polynomials

$$p_0 = 4 + 8s + 5s^2 + s^3$$
$$p_1 = 1 - s + s^2$$
$$p_2 = s + s^4$$

and two real parameters $q_1 \in [-6, 12]$ and $q_2 \in [-5, 15]$. Suppose you want to check which values of q_1 and q_2 give rise to a stable $p(s,q_1,q_2)$. As there are two parameters and the uncertainty structure is quite complicated there is hardly any theoretical method known to help. Nevertheless, simple gridding can do the job in a reasonable time.

To start, insert the data

```
p0 = 4+8*s+5*s^2+s^3; p1=1-s+s^2; p2=s+s^4;
```

and choose an appropriate grid, such as

```
q1 = -6:.1:12; q2=-5:.1:15;
```

Then construct the stability area array by typing

```
S = sarea(q1,q2,'p0+(q1+q2)*p1+sqrt(abs(q2))*p2',p0,p1,p2);
```

and plot it with the help of

```
sareaplot(q1,q2,S)
```

What you get is the really nice picture displayed in Fig. 1. It shows which combinations of parameter values yield a stable polynomial.

It is a must here to use names rather than values as the input arguments for both the parameters and the polynomials. Violation of this rule causes an error message:

```
S=sarea(-6:.1:12,q2,'p0+(q1+q2)*p1+sqrt(abs(q2))*p2',p0,p1,p2);
??? Error using ==> sarea
```

```
The input argument of parameter vector or polynomial must be a
named variable.
```

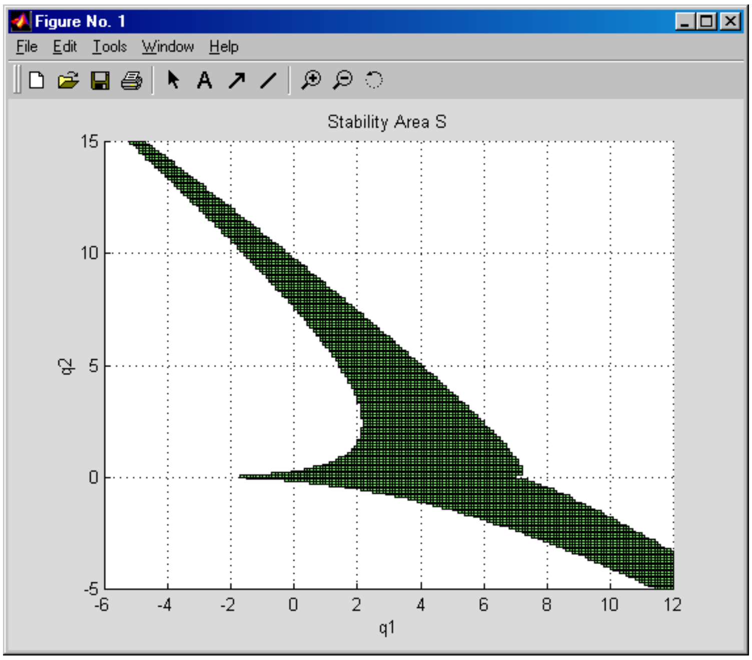

Fig. 6. Stability area of Example 1

Example 2

For the same three fixed polynomials, but a different uncertainty structure

$$p(s,\lambda_1,\lambda_2) = p_0(s) + \lambda_1\lambda_2 p_1(s) + \lambda_2^3 p_1(s)$$

and parameters $\lambda_1 \in [-20,20]$ and $\lambda_2 \in [-10,10]$, we may use the grid

```
lambda1 = -20:.1:20; lambda2 = -10:.1:10;
```

and type

```
expr = 'p0+lambda1*lambda2*p1+lambda2^3*p1';
S2 = sarea(lambda1,lambda2,expr,p0,p1,p2);
sareaplot(lambda1,lambda2,S2)
```

This results in the amusing picture shown in Fig. 7.

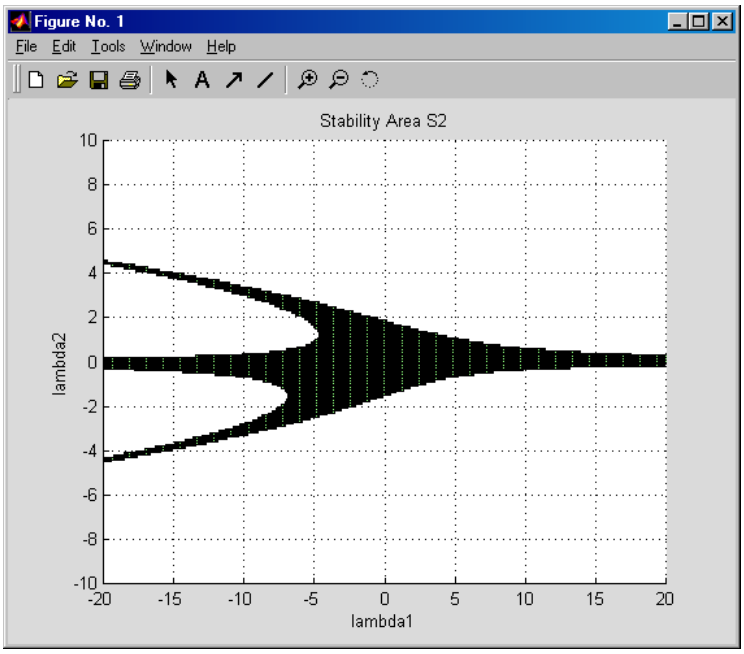

Fig. 7. Stability area for Example 2

Example 3

3-D examples are even nicer but, of course, more time consuming. Consider a three-parameter uncertain polynomial

$$p(s,q_1,q_2,q_3) = p_0(s) + (q_1 + q_1 q_3) q_3 p_1(s) + q_1^2 q_2^2 p_2(s)$$

with

$$p_0(s) = 2 + 4s + 3s^2 + s^3$$

$$p_1(s) = -1.7 + 0.13s + 0.29s^2$$

$$p_2(s) = 1.2 + 1.2s - 0.038s^2$$

and $q_1, q_2, q_3 \in [-20, 20]$. When inputting the data

```
p0 = 2+4*s+3*s^2+s^3;
p1 = -1.7+0.13*s+0.29*s^2;
p2 = 1.2+1.2*s-0.038*s^2;
q1 = -20:.5:20;q2=q1;q3=q1;
```

```
expr = 'p0+(q1+q1*q2)*q3*p1+(q1^2*q2^2)*p2';
```

the function called by

```
S3 = sarea(q1,q2,q3,expr,p0,p1,p2);
```

needs more than one hour on an average PC. The command

```
sareaplot(q1,q2,q3,S3)
```

results in Fig. 8. Such a 3-D plot can of course be zoomed or rotated by mouse in the standard MATLAB manner.

Example 4

We consider another 3-D example of uncertainty structure

$$p(s,q_1,q_2,q_3) = p_0 + (q_1^2 - q_3)p_1 + (q_2 + q_3)p_2$$

with

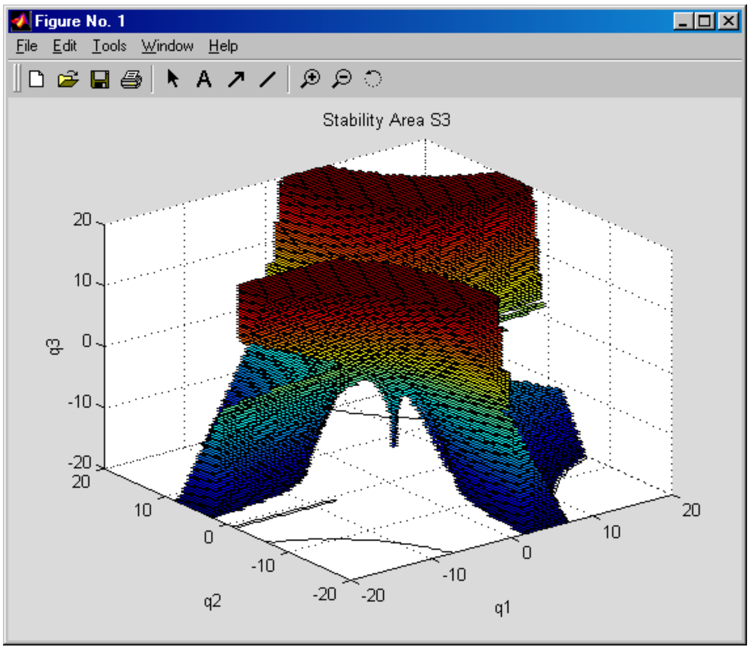

Fig. 8. Stability area for Example 3

$$p_0(s) = 2 + 4s + 3s^2 + s^3$$
$$p_1(s) = 0.5 - 1.5s - s^2$$
$$p_2(s) = 0.02 - 2s + s^2$$
$$q_1 \in [-7,7], q_2 \in [-40,2], q_3 \in [0,40]$$

We enter the data

```
p0 = 2+4*s+3*s^2+s^3;
p1 = 0.5-1.5*s-s^2;
p2 = 0.02-2*s+s^2;
expr = 'p0+(q1^2-q3)*p1+(q2+q3)*p2';
q1 = -7:.1:7; q2=-40:2; q3 = 0:0.5:40;
```

and run the macros

```
S4 = sarea(q1,q2,q3,expr,p0,p1,p2);
sareaplot(q1,q2,q3,S4)
```

to obtain Fig. 9.

Algorithm The method is trivial: It directly runs a stability test step by step for each particular point of the grid.

Diagnostics The macro `sarea` displays an error messages if

- There are not enough input arguments
- An argument corresponding to parameter or polynomial is not a named variable
- An invalid argument is encountered
- The expression string cannot be evaluated (in which case the error message is generated by `lasterr` and hence its text may vary according to the situation encountered).

The macro `sareaplot` displays an error messages if

- An invalid argument or option is encountered
- There are more than three vectors representing uncertain parameters
- Input arguments have inconsistent dimensions

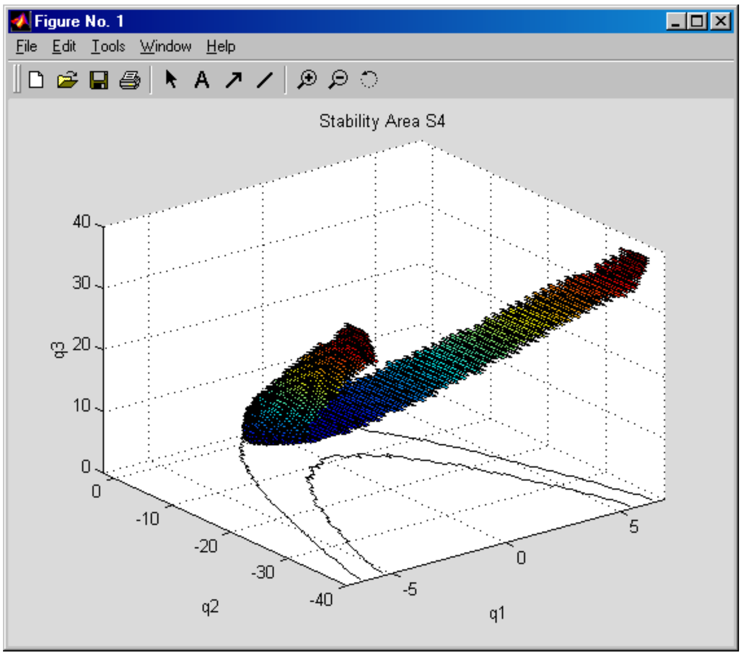

Fig. 9. Stability area for Example 4

See also

`isstable`	Stability test for a polynomial matrix
`vset, vsetplot`	Value set plot for a parametric polynomial family

sim2lmf, sim2rmf

Purpose LMF and RMF description of a SIMULINK model.

Syntax

```
[N,D] = sim2lmf('model')
[N,D] = sim2lmf('model',X0)
[N,D] = sim2lmf('model',X0,U0)
[N,D] = sim2rmf('model')
[N,D] = sim2rmf('model',X0)
[N,D] = sim2rmf('model',X0,U0)
```

Description The command

```
[N,D] = sim2lmf('model')
```

returns the LMF description for the linearization of the SIMULINK scheme called `'model'`. The initial conditions for inputs and internal states of related observer-form realization by default are supposed to be zero but may be specified as additional input arguments:

```
[N,D] = sim2lmf('model',X0)
[N,D] = sim2lmf('model',X0,U0)
```

Similarly, the commands

```
[N,D] = sim2rmf('model')
[N,D] = sim2rmf('model',X0)
[N,D] = sim2rmf('model',X0,U0)
```

compute the RMF description of the SIMULINK file `'model'`.

Compatibility These functions are new in the Polynomial Toolbox.

Examples Consider the SIMULINK nonlinear model `'pendm'` of an undamped simple pendulum depicted in Fig. 10. The `sim2lmf` command may be employed to obtain its linearization:

```
[N,D] = sim2lmf('pendm')
Constant polynomial matrix: 1-by-1
N =
    -1
D =
    -9.8 - s^2
```

Without specifying any initial conditions we obtain the linearization around the lower stable position of the pendulum. The linear model of an inverted pendulum can be found using the same SIMULINK scheme by prescribing the initial angle $\varphi_0 = \pi$:

```
[N,D] = sim2lmf('pendm', [pi 0])
Constant polynomial matrix: 1-by-1
N =
    -1
D =
     9.8 - s^2
```

The command `sim2rmf` will of course give the same result in this SISO example.

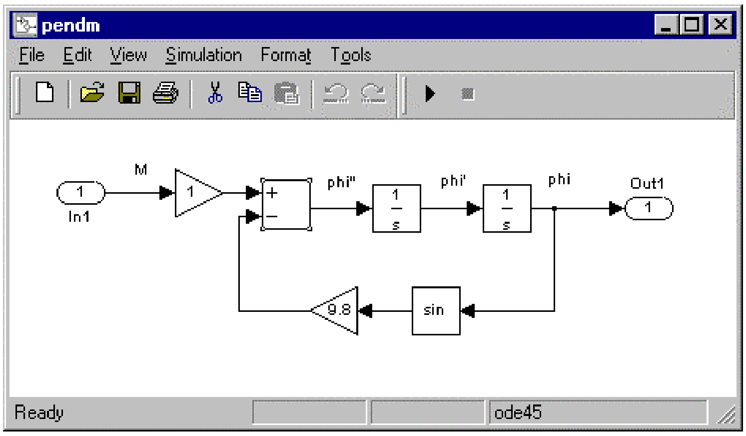

Fig. 10. SIMULINK model of a simple undamped pendulum

Algorithm The standard SIMULINK command `linmod` is utilized along with the Polynomial Toolbox macros `ss2lmf` and `ss2rmf`.

Diagnostics The macros `sim2lmf` and `si2rmf` display error messages if

- The specified SIMULINK model does not exist
- The length of the initial conditions vector does not match the model dimension
- An invalid argument is encountered

See also

`ss2lmf, ss2rmf`	State-space to LMF and RMF conversion
`polblock`	Polynomial Toolbox block for SIMULINK

spherplot

Purpose

Plot the value set of a polynomial family with a spherical uncertainty set and independent uncertainty structure for a range of frequencies.

Syntax

`spherplot(p0,omega,r,W)`

spherplot(p0,omega,r)

`spherplot(p0,omega)`

Description

This is a tool for testing robust stability using the Zero Exclusion Condition. A family of polynomials $P = \{p(\cdot,\mathbf{q}) : \mathbf{q} \in Q\}$ is said to be spherical if $p(\cdot,\mathbf{q})$ has an independent uncertainty structure and the uncertainty set Q is an ellipsoid. The command

```
spherplot(p0,omega,r,w)
```

plots the value sets for the spherical polynomial family, where `p0` is a nominal polynomial, `omega` is a vector of generalized frequencies, `r` is a robustness bound and `weight` is a vector of diagonal entries of the weighting matrix `W`. If the family has an independent uncertainty structure then the polynomial family can be expressed in the centered form

$$p(s,\mathbf{q}) = p_0(s) + \sum_{i=0}^{n} q_i s^i$$

where the weighted Euclidian norm of the vector of the uncertain parameters is bounded by

$$\|\mathbf{q}\|_{2,W} \le r$$

The command

```
spherplot(p0,omega,r)
```

assumes that the weighting matrix `w` is the unit matrix. The command

```
spherplot(p0,omega)
```

assumes that the weighting matrix is the unit matrix and the robustness margin `r` equals 1. The vector of uncertain parameters is then bounded by

$$\|\mathbf{q}\|_2 \le 1$$

As with other tools based on the Zero Exclusion Condition it is necessary to make sure that there is at least one stable member of the polynomial family. Also remember that if you enter the `weight` parameter you only assign the vector of diagonal entries and not the whole matrix.

Compatibility

This function is new in the Polynomial Toolbox.

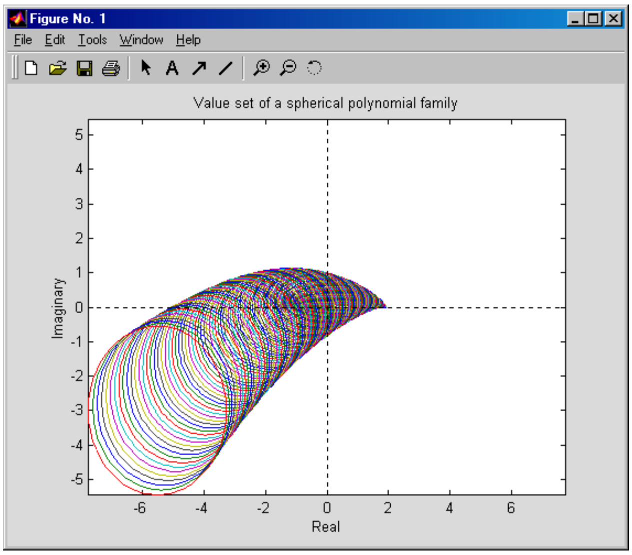

Fig. 11. Value set for Example 1

Examples

Example 1

Consider the uncertain polynomial

$$p(s,q) = (0.5 + q_0) + (1 + q_1)s + (2 + q_2)s^2 + (4 + q_3)s^3$$

with the uncertainty bound $\|\mathbf{q}\|_{2,\mathbf{W}} \leq 1$ and the weighting matrix $W = \mathrm{diag}(2, 5, 3, 1)$, that is,

$$2q_0^2 + 5q_1^2 + 3q_2^2 + q_3^2 \leq 1$$

Use the graphical method of the Zero Exclusion Principle to test for the robust stability of the given uncertain polynomial. First we express the given polynomial in the centered form

$$p(s,\mathbf{q}) = 0.5 + s + 6s^2 + 4s^3 + \sum_{i=0}^{3} q_i s^i$$

with the uncertainty bound unchanged. Now type

```
p0 = 0.5+s+6*s^2+4*s^3;
weight = [2,5,3,1];
r = 1; omega = 0:.01:1;
isstable(p0)
```

```
ans =
      1
```

The graphical representation of the value set for the given range of frequencies is generated by

```
spherplot(p0,omega,r,weight)
```

and shown in Fig. 11. It can be seen that the Zero Exclusion Condition is violated so we conclude that the given polynomial family is not robustly stable.

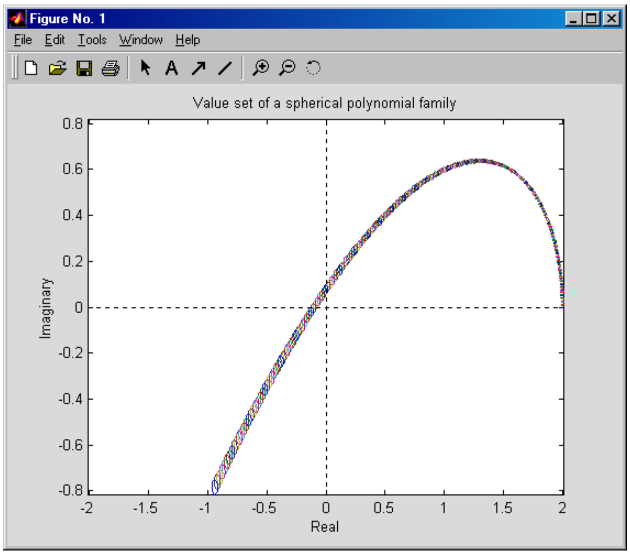

Fig. 12. Value set for Example 2

Example 2

Similarly to the previous example, test the following polynomial [1, pp.268] for robust stability

$$p(s,\mathbf{q}) = (2+q_0) + (1.4+q_1)s + (1.5+q_2)s^2 + (1+q_3)s^3$$

with the uncertain parameters subject to

$$\|\mathbf{q}\|_2 \le 0.011$$

We type

```
p0 = 2+1.4*s+1.5*s^2+s^3; r = 0.011; omega = 0:0.005:1.4;
isstable(p0)
```

```
ans =
     1
spherplot(p0,omega,r)
```

This results in Fig. 12. In this case, the origin is excluded from the value set and we conclude that the polynomial family is robustly stable.

Algorithm

The value set at each frequency is characterized [1, p. 270] by an ellipse centered at nominal $p_0(j\omega)$ and with principal axis in the real direction having length

$$R_0 = 2r\sqrt{\left(\sum_{i\,even} w_i^2 \omega^{2i}\right)}$$

and principal axis in the imaginary direction having length

$$I_0 = 2r\sqrt{\left(\sum_{i\,odd} w_i^2 \omega^{2i}\right)}$$

The number r is a bound on the Euclidean norm of the vector of uncertain parameters, ω is a frequency, and W a weighting matrix given by

$$W = diag\left(w_1^2, w_2^2, \ldots, w_n^2\right).$$

References

[1] R. Barmish: *New Tools for Robustness of Linear Systems.* Macmillan Publishing Company. New York, 1994.

Diagnostics

The macro returns error messages if the input arguments are incompatible.

See also

`khplot`	Value set for an interval polynomial.
`ptopplot`	Value set for a polytope of polynomials.
`vsetplot`	Value set for polynomials with general uncertainty structure

tsyp

Purpose

Use the Tsypkin-Polyak function to determine the ℓ_∞ robustness margin for a continuous interval polynomial.

Syntax

```
R = tsyp(p0,w,epsilon)
R = tsyp(p0,w)
R = tsyp(p0)
R = tsyp(p0,[],epsilon)
[R,W] = tsyp(p0)
[R,W] = tsyp(p0,w,epsilon)
[R,W] = tsyp(p0,w)
[R,W] = tsyp(p0)
[R,W] = tsyp(p0,[],epsilon)
```

Description

Given the nominal polynomial `p0` the macro finds a robustness margin `R` such that the resulting interval polynomial

$$p_R(s,q) = p_0(s) + R\sum_{i=0}^{n}[-\varepsilon_i, \varepsilon_i]s^i$$

is robustly stable. The command

```
R = tsyp(p0,w,epsilon)
```

computes the robustness margin for an interval polynomial `p0` at frequencies given by the vector `w` and with scale factors given by the vector `epsilon`. The command

```
R = tsyp(p0,w)
```

implicitly uses the coefficients of the nominal polynomial `p0` as scale factors. The command

```
R = tsyp(p0)
```

implicitly uses the coefficients of the nominal polynomial `p0` as scale factors and supplies its own vector of frequencies. The command

```
R = tsyp(p0,[],epsilon)
```

uses the supplied scale factors but computes its own frequency vector . The commands

```
[R,W] = tsyp(p0)
```

and

```
[R,W] = tsyp(p0,[],epsilon)
```

return the computed vector of frequencies as the second output for possible use with the function `khplot`.

If no output is specified then the graphical output of Tsypkin-Polyak function is generated. Also shown is the robustness margin square, which is the largest possible square inscribed inside the plot of the Tsypkin-Polyak function. Its size is the robustness margin `R`.

Compatibility

This function is new in the Polynomial Toolbox.

Examples

Example 1

We consider the interval polynomial family P_r with the nominal polynomial given by

$$p_o(s) = 676 + 1365s + 1019s^2 + 420s^3 + 104s^4 + 15s^5 + s^6$$

and scaling factors $\varepsilon_0 = 676$, $\varepsilon_1 = 682.5$, $\varepsilon_2 = 509.5$, $\varepsilon_3 = 210$, $\varepsilon_4 = 52$, $\varepsilon_5 = 15$, $\varepsilon_6 = 0$. Find a robustness margin `R` such that the resulting interval polynomial is robustly stable. Typing

```
p0 = pol([676 1365 1019 420 104 15 1],6);
w = 1:0.01:10;
epsilon = [676 682.5 509.5 210 52 15 0];
tsyp(p0,w,epsilon)
ans =
    0.2344
```

results in Fig. 13. We obtain the robustness margin `R` = 0.2344, which may be viewed as size of the largest possible square inscribed inside the plot of the Tsypkin-Polyak function.

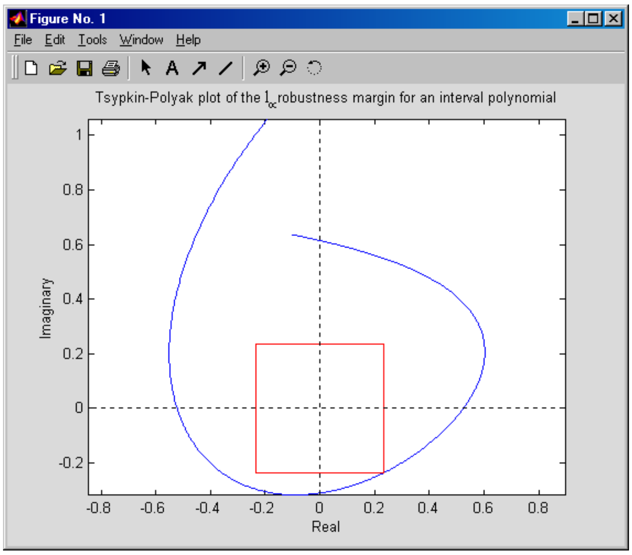

Fig. 13. Output for Example 1

Example 2 — simple feedback

The nominal pitch control system ([1], pp.101) is described in Fig. 14. Find the robustness margin for $K = 4$.

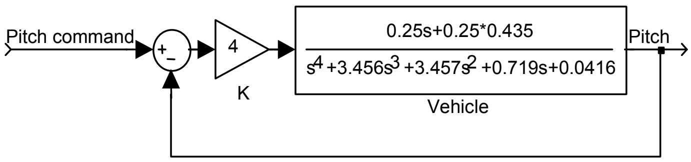

Fig. 14. Pitch control system

```
K = 4;
num = pol([0.25*0.435 0.25],1);
den = pol([.0416 .719 3.457 3.456 1],4);
p0 = den + K*num;
```

```
[R,W] = tsyp(p0); R
R =
    0.2741
pminus = p0 - R*p0;
pplus = p0 + R*p0;
khplot(pminus, pplus, W)
```

The output is shown in Fig. 15. Restricting the frequency range to lower frequencies (or zooming) by typing

```
khplot(pminus, pplus, W(1:round(length(W)/3)))
```

leads to Fig. 16. Thus we have found the robustness margin `R` and now it is easy to find the uncertainty bounds on the coefficients of the polynomial:

```
Qbounds = [pminus{:}' pplus{:}']
Qbounds =
    0.3460    0.6072
    1.2478    2.1902
    2.5093    4.4047
    2.5086    4.4034
    0.7259    1.2741
```

If the coefficients remain within these intervals then the polynomial is guaranteed to be stable.

Algorithm

The algorithm is based on the Tsypkin-Polyak function $G_{TP}(\omega)$ described in [1], pp.97. It finds a robust margin `R` such that the condition $\left\|G_{TP}(\omega)\right\|_\infty > R$ is satisfied for all frequencies (recall that $\|z\|_\infty = \max\{|\mathrm{Re}(s)|,|\mathrm{Im}(z)|,\ z \in \mathbf{C}\}$ and no degree drop occurs. It uses the standard MATLAB minimization routine `fminbnd`.

References

R. Barmish, *New Tools for Robustness of Linear Systems.* Macmillan Publishing Company. New York, 1994.

Diagnostics

Since the quality of the result of the minimization routine depends considerably on the initial guess, the proper choice of the frequency range is important. The program automatically validates its result by the testing stability of the four Kharitonov polynomials. If these are not robustly stable then the following error message appears:

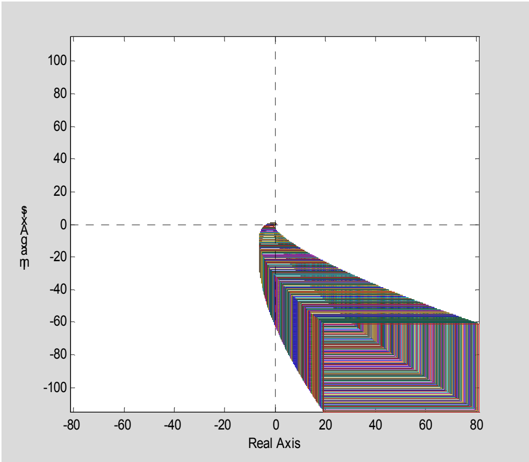

Fig. 15. Output for Example 2

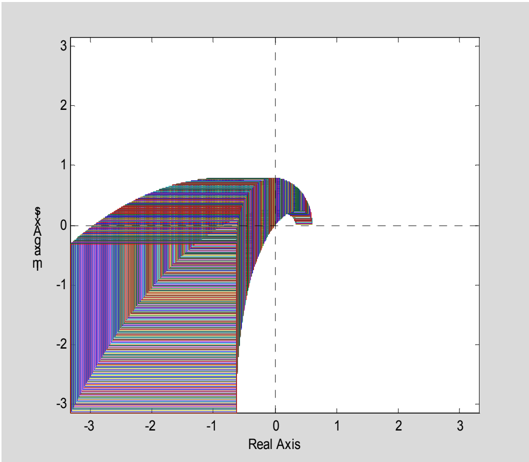

Fig. 16. Zoomed output for Example 2

```
Warning: Resulting margin does not guarantee robust stability
of the interval polynomial. Run again with extended frequency
range and/or denser gridding.
```

Also use the graphical output to assess the acceptability of the result.

See also

`khplot`	Value set for an interval polynomial.
`kharit`	Return the Kharitonov polynomials

vset, vsetplot

Purpose Value set of a parametric polynomial

Syntax

```
V = vset(q1,...,qm,ExpressionString,p0,p1,…,pn[,omega][,qType])
vsetplot(V[,PlotType][,'new'])
```

Description This is another tool for robust stability testing with the help of the Zero Exclusion Condition. The command

```
V = vset(q1,...,qm,ExpressionString,p0,p1,…,pn[,omega[,qType]])
```

computes the values at the generalized frequencies given by the vector ω of a family of polynomials depending on m independent parameters. The parameter values that are selected are given by the vectors $q_1,\ldots,q_m$ and the results are stored in a matrix V of complex numbers. The values at the various frequencies are organized column wise.

The arguments $p_0,\ldots,p_n$ are given fixed polynomials that define the family. The uncertainty structure is described by the string variable `ExpressionString.` This string is a MATLAB-syntax expression for $a_0(q_1,\ldots,q_m)p_0+\cdots+a_n(q_1,\ldots,q_m)p_n$ that is composed of the parameter names and the names of the fixed polynomials. The "coefficients" $a_i(q_1,\ldots,q_m)$ are given by any MATLAB-syntax expression consisting of the parameter names acting here as scalar symbols.

Note that the input arguments representing both the parameters and the fixed polynomials must already exist in the current workspace and, moreover, must be written using their names (rather than values) in the function call. The use of the command is further explained in the examples below.

Once the value matrix V is available one can plot it by typing

```
vsetplot(V[,PlotType][,'new'])
```

The plot consists of the sets $V(\omega_i)$ of values for the generalized frequencies. Depending on the optional argument `PlotType` they can be composed of lines (default or `PlotType = 'lines'`) or points (`PlotType = 'points'`). With the input string argument `'new'` the plot is displayed in a new window.

By default or with the string argument `qType = 'r'` the grid consists of combinations of entries in the vectors $q_1,\ldots,q_m$. When `qType='e'` the grid consists of l points defined by their coordinates in m-dimensional space; all the $q_1,\ldots,q_m$ must be of the same length l..

Scope This pair of macros tests robust stability of the polynomial family by the Zero Exclusion Condition [1]. If the family contains a stable member and if the value set for all generalized frequencies on the stability region boundary excludes the point 0 then the family is concluded to be robustly stable (stable for all parameters ranging given intervals). For more details, see [1] or another robust control textbook.

To perform the robust stability test we first find a stable member in the family. Typically, the nominal value is stable or we proceed by trial and error. Once a stable member is found we substitute into the family several generalized frequencies from the stability boundary and plot the corresponding value sets. It is important to use frequencies leading to value sets close to the point 0. If none of the sets contains or touches the critical point then robust stability is verified.

To plot value sets for special uncertainty structures such as polytopic or even interval uncertainty more efficient macros are available, in particular `ptopplot` and `khplot`, respectively.

Compatibility These functions are new in the Polynomial Toolbox

Examples To understand the use of command, go through the following simple examples.

Example 1: Continuous-time case

Consider an uncertain polynomial

$$p(s,q_1,q_2) = p_0(s) + q_1 p_1(s) + q_2 p_2(s) + q_1 q_2 p_{12}(s)$$

composed of four fixed polynomials

$$p_0 = 1.853 + 3.164s + 2.871s^2 + 2.56s^3 + s^4$$

$$p_1 = 3.773 + 4.841s + 2.06s^2 + s^3$$

$$p_2 = 1.985 + 1.561s + 1.561s^2 + s^3$$

$$p_{12} = 4.032 + 1.06s + s^2$$

and check its robust stability for $q_1 \in [0,1]$ and $q_2 \in [0,3]$. To this end, first enter the data

```
p0 = pol([1.853 3.164 2.871 2.56 1],4);
p1 = pol([3.773 4.841 2.06 1],3);
p2 = pol([1.985 1.561 1.561 1],3);
p12 = pol([4.032 1.06 1],2);
```

describe the uncertainty structure

```
expr = 'p0+q1*p1+q2*p2+q1*q2*p12'
```

and define a reasonable grid for the parameter intervals

```
q1 = 0:1/50:1; q2=0:3/50:3;
```

As the polynomials are of continuous-time nature it is necessary to plot value sets for several critical frequencies on the imaginary axis. Hence, choose ω_i = 1.3, 1.4, 1.6, 1.6 and type

```
V = vset(q1,q2,expr,p0,p1,p2,p12,j*[1.3:.1:1.6]);
vsetplot(V,'points')
```

to obtain the plot of Fig. 17. Note that the value sets are not convex. This typically happens whenever the uncertainty structure is multilinear or more complex.

As one of the value sets (that for $\omega_i = 1.4$) seems to include the critical point 0 we zoom the plot in to that of Fig. 18 to see more details. It is evident that $0 \in V(1.4)$ and, hence, the family is *not* robustly stable.

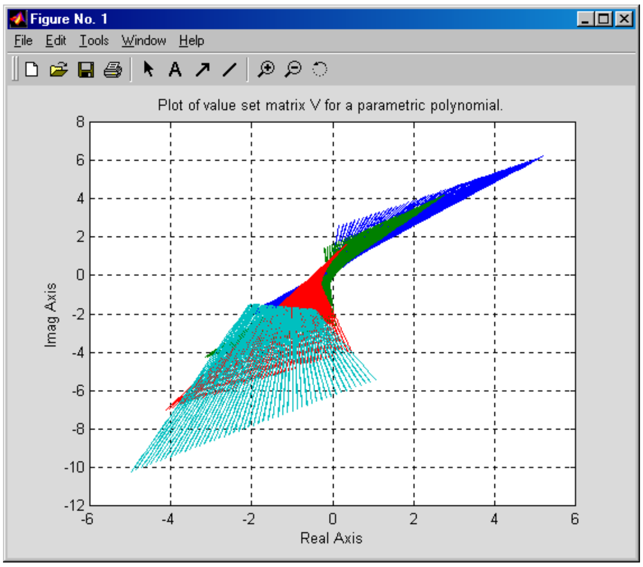

Fig. 17. Value set for Example 1

Example 2: Discrete-time case

Now consider a family of discrete-time polynomials with quite complicated uncertainty

$$p(z^{-1},k,l,m) = e(z^{-1}) + \sin(k) f(z^{-1}) - \cos(k) k g(z^{-1}) + l^2 h(z^{-1})$$

where

$$e(z^{-1}) = (z^{-1} - 1.5)(z^{-1} + 2)(z^{-1} - 2)$$

$$f(z^{-1}) = 1$$

$$g(z^{-1}) = z^{-1}$$

$$h(z^{-1}) = z^{-2}$$

and $k,l,m \in [-1,1]$. Here the data to be entered are

```
e = (zi-1.5)*(zi+2)*(zi-2);f=1; g=zi; h=zi^2;
uncrty = 'e+sin(k)*f-cos(m)*k*g+(l^2)*h';
```

and, say,

```
k = -1:.1:1; l = k; m = k;
```

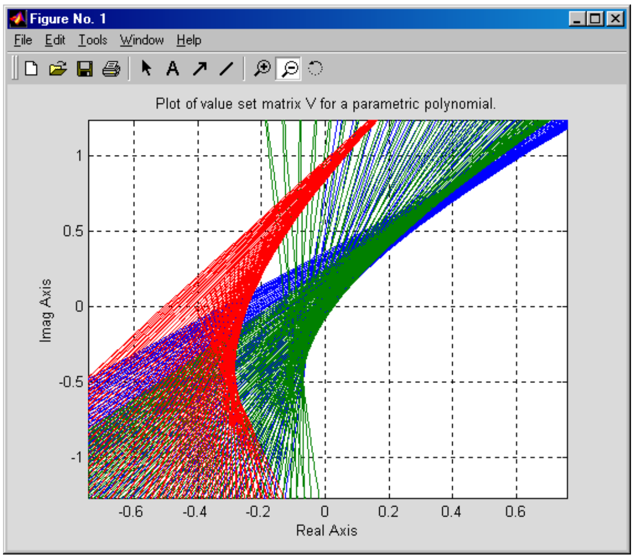

Fig. 18. Zoomed plot

Before using the Zero Exclusion Condition to test robust stability we must check that the family contains at least one stable member. Indeed, the nominal polynomial $p(z^{-1},0,0,0) = e(z^{-1})$ is stable:

```
isstable(e)
ans =
      1
```

Now we evaluate and plot value sets at 40 generalized frequencies evenly spread around unit circle:

```
V = vset(k,l,m,uncrty,e,f,g,h,exp(j*(0:2*pi/40:2*pi)));
vsetplot(V)
```

and obtain the picture of Fig. 19. As all the sets are far enough to the right of the critical point robust stability is verified.

Example 3: Incorrect calls

The user must not forget about calling the function with named variable arguments.

Even if the parameters

```
q0 = 1:5;
```

already exist in the workspace it must be represented by its name. The following call is definitely incorrect

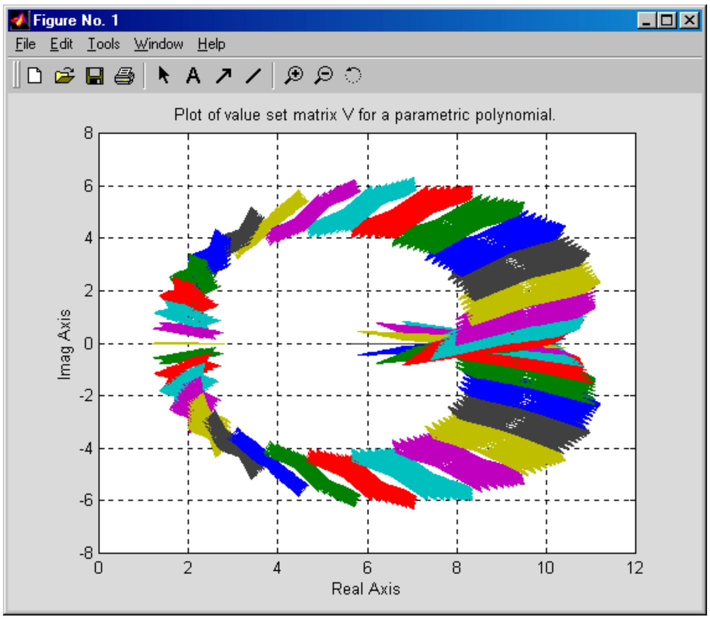

Fig. 19. Value set for Example 2

```
vset(1:5,'q0*p',p,j)
??? Error using ==> vset
Undefined function or variable 'q0'.
```

Algorithm

The method is quite easy. The overall picture is composed of the value sets for the generalized frequencies. Each set is obtained by substituting the frequencies into the uncertainty formula for all parameter values achieved by gridding the parameter set.

References

R. Barmish: *New Tools for Robustness of Linear Systems.* Macmillan Publishing Company. New York, 1994.

Diagnostics

The macro `vset` displays an error message if

- The set of generalized frequencies is not a non-empty vector
- There are not enough input arguments
- The expression string cannot be correctly evaluated. Here the error message is returned by `lasterr` and hence its text may vary according to the inconsistency encountered

The macro `vsetplot` displays an error message if

- The value set matrix is not a non-empty 2-dimensional double.
- An inappropriate input string argument is used.

See also

`khplot`	Value set for an interval polynomial.
`ptopplot`	Value set for a polytope of polynomials.

Index