# POLMAT

## a library for MuPAD

polynomials
polynomial matrices
polynomial methods in control
multivariate polynomials

# REFERENCE MANUAL

by

Petr Augusta

**Abstract** This document gives basic informations about POLMAT. The POLMAT is a freely available package for the computer algebra system **MuPAD**. The POLMAT is intended for manipulating and computing with polynomials and polynomial matrices. It includes algorithms implemented for basic operations and computing, for solving equations with polynomials and polynomial matrices and others.

**Keywords** Polynomials; polynomial matrices; symbolic computation; polynomial methods in control.

20th November 2014

A part of this work was financially supported by the Czech Science Foundation
under the project GPP103/12/P494.

# Contents

## How to start?

Simply, download the POLMAT into your computer. In **MuPAD** type

```
>> package("polmat")
```

where `polmat` is full path to POLMAT directory in your computer, for example,

```
/home/myname/polmat
```

See **MuPAD**'s manual for details.

## polmat::new – definition of a new object

polmat::new defines a new object – a polynomial, a polynomial matrix, a racional function or a racional matrix.

**Call(s)**

```
polmat(p<, x>)
polmat(m, x)
```

**Parameters**

p – a polynomial or an expression
m – an element of a domain DOM_LIST, see the examples
x – an indeterminate

**Return Value**

an element of a domain or a type
Dom::Matrix(polmat::poly), Dom::Matrix(), DOM_EXPR, DOM_POLY,...

**Examples**

```
>> a:=polmat(x+10*c*x^2, x)
```

```
                                  2
                      poly((10 c) x  + x, [x])
```

```
>> A:=polmat([[d, -d+1],[5*d^2, d^2]], d)
```

```
                          +-                 -+
                          |    d,  - d + 1   |
                          |                  |
                          |     2       2    |
                          |  5 d ,     d     |
                          +-                 -+
```

```
>> a:=polmat(3)
```

```
                                  3
```

```
>> domtype(%)
```

```
                               DOM_INT
```

```
>> a:=polmat(3, x)
```

```
                            poly(3, [x])
```

```
>> domtype(%)
```

```
                              DOM_POLY
```

```
>> c:=polmat(1/z + 3/z^2)
```

$$\frac{1}{z} + \frac{3}{z^2}$$

```
>> c:=polmat(1/z + 3/z^2, 1/z)
```

$$\mathrm{poly}(3\,d^2 + d, [d])$$

## polmat::axbyc – solution to equation

polmat::axbyc finds a solution to linear equation with polynomials or polynomial matrices. For polynomials, the equation has the form

$$a\,x + b\,y = c,$$

where $a$, $b$, $c$ are given polynomials and $x$, $y$ are unknown polynomials, i. e., a solution to be determined. For polynomial matrices, the equation has the form

$$A\,X + B\,Y = C,$$

where $A$, $B$, $C$ are given polynomial matrices and $X$, $Y$ are unknown polynomial matrices, i. e., a solution to be determined.

**Call(s)**

```
polmat::axbyc(a, b, c, x<, "x-minimal">)
polmat::axbyc(a, b, c, x<, "y-minimal">)
polmat::axbyc(a, b, c, x<, "degy<dega">)
polmat::axbyc(A, B, C, x)
```

**Parameters**

`a, b, c` – the polynomials
`A, B, C` – the polynomial matrices of type `Dom::Matrix(polmat::poly)`
`x` – an indeterminate

**Return Value**

the polynomials $x$, $y$ of the type `DOM_POLY` or the matrix $X$, $Y$ of the type `Dom::Matrix(polmat::poly)`

**Related Functions**

`polmat::axybc`, `polmat::xaybc`, `polmat::gcd`

**Details**

- The general solution to the equation $a\,x + b\,y = c$ is

$$\begin{aligned} x &= p\,\frac{c}{d} + r\,t \\ y &= q\,\frac{c}{d} + s\,t, \end{aligned}$$

  where the polynomials $p$, $q$, $r$, $s$ are the solution to `polmat::gcd` and $t$ is an arbitrary polynomial.
- If the parameter `"x-minimal"` is given, the solution $x_1$, $y_1$ minimising the degree of $x$ is computed.
- If the parameter `"y-minimal"` is given, the solution $x_2$, $y_2$ minimising the degree of $y$ is computed.
- If the parameter `"degy<dega"` is given, the solution with $\partial y_3 < \partial a$ is computed. Generally, such a solution has the form

$$\begin{aligned} x_3 &= x_2 + \frac{b}{(a,\ b)}\,t \\ y_3 &= y_2 - \frac{a}{(a,\ b)}\,t, \end{aligned}$$

  where an arbitrary polynomial $t$ must satisfy $\partial t < \partial(a,\ b)$.

- The general solution to the equation $A\,X + B\,Y = C$ is

$$\begin{aligned} X &= P_2\,C_2 + R_2\,T \\ Y &= Q_2\,C_2 + S_2\,T, \end{aligned}$$

  where $T$ is an arbitrary matrix and

$$\begin{aligned} C &= D_2\,C_2, \\ A\,P_2 + B\,Q_2 &= D_2 \\ A\,R_2 + B\,S_2 &= 0. \end{aligned}$$

- Conditions of solvability of the above equations, number of solutions and other details can be found in [1, 2].

**Examples**

```
>> A:=polmat([[1, d, d^2],[0, d, d]], d)

                                +-           -+
                                |          2  |
                                |  1, d, d    |
                                |             |
                                |  0, d,  d   |
                                +-           -+
>> B:=polmat([[1+d, 0],[0, 1+d]], d)

                              +-               -+
                              |  d + 1,    0     |
                              |                  |
                              |    0,    d + 1   |
                              +-               -+
>> C:=polmat([1+d+d^2, 1+d], d)

                               +-             -+
                               |   2            |
                               |  d  + d + 1    |
                               |                |
                               |     d + 1      |
                               +-             -+
>> [X, Y]:=polmat::axbyc(A, B, C, d)

[

array(1..3, 1..1,
  (1, 1) = - d^2*t[1, 1] - d^2*t[3, 1] + 2*d^2 + d*t[1, 1] - d*t[2, 1] - d\
*t[3, 1] + 2*d - t[2, 1] + 1,
  (2, 1) = d t[3, 1] - d - t[1, 1] + t[3, 1] - 1,
  (3, 1) = t[1, 1]
```

```
)


      +-                          -+
      |          t[2, 1]           |
    , |                            | ]
      |  - d t[3, 1] + d + 1       |
      +-                          -+
>> A*X+B*Y-C

                                    +-    -+
                                    |  0   |
                                    |      |
                                    |  0   |
                                    +-    -+
```

## polmat::axbycd – solution to equation

polmat::axbycd finds a solution to linear equations with polynomials

$$a\,x^* + b^* y = c + d^*,$$

where $a$, $b$, $c$, $d$ are given polynomials and $x$, $y$ are unknown polynomials.

**Call(s)** polmat::axbycd(a, b, c, d_c, x<, "x-minimal">)

polmat::axbycd(a, b, c, d_c, x<, "y-minimal">)

**Parameters** a, b, c – polynomials, expressions or numbers
d_c – an expression
x – an indeterminate

**Return Value** polynomials $x$, $y$ of the type DOM_POLY

**Related Functions** polmat::axbyc, polmat::axxab

**Details**

- If the parameter "x-minimal" is given the solution minimising the degree of $x$ is found.
- If the parameter "y-minimal" is given the solution minimising the degree of $y$ is found.
- If neither "x-minimal" nor "y-minimal" is given the solution minimising the degree of $x$ is found.
- The algorithm was derived and described in [3].

**Examples**

```
>> a:=polmat(1+2*d)

                              poly(2 d + 1, [d])
>> b:=polmat(1+3*d)

                              poly(3 d + 1, [d])
>> c:=1

                                        1
>> d_c:=0

                                        0
>> polmat::axbycd(a, b, c, d_c, d, "x-minimal")

                     [poly(-1/5, [d]), poly(2/5 d, [d])]
>> polmat::axbycd(a, b, c, d_c, d, "y-minimal")

                     [poly(3/5 d, [d]), poly(-1/5, [d])]
```

## polmat::axxab – solution to equation

polmat::axxab finds a solution to symmetric linear equation with polynomials

$$a^*x + a\,x^* = b,$$

where $a$ is the given polynomial, $b$ is the given two-sided polynomial and $x$ is an unknown polynomial.

| | |
|---|---|
| **Call(s)** | polmat::axxab(a, b, x) |
| **Parameters** | a, b – the polynomials or expressions<br>x – an indeterminate |
| **Return Value** | the polynomial $x$ of the type DOM_POLY |
| **Related Functions** | polmat::axbyc, polmat::axbycd |

**Examples**

```
>> a:=polmat(1+d)
```

```
                                poly(d + 1, [d])
```

```
>> b:=polmat(1/d+2+d)
```

```
                                      1
                                 d + - + 2
                                      d
```

```
>> x:=polmat::axxab(a, b, d)
```

```
                                  poly(1, [d])
```

## polmat::axybc – solution to equation

polmat::axybc finds a solution to linear equation with polynomials or polynomial matrices. For polynomials, the equation has the form

$$a\,x + y\,b = c,$$

where $a$, $b$, $c$ are given polynomials and $x$, $y$ are unknown polynomials, i. e., a solution to be determined. For polynomial matrices, the equation has the form

$$A\,X + Y\,B = C,$$

where $A$, $B$, $C$ are given polynomial matrices and $X$, $Y$ are unknown polynomial matrices, i. e., a solution to be determined.

**Call(s)**
```
polmat::axybc(a, b, c, x)
polmat::axybc(A, B, C, x)
```

**Parameters**
`a, b, c` – the polynomials
`A, B, C` – the polynomial matrices of type `Dom::Matrix(polmat::poly)`
`x` – an indeterminate

**Return Value**
the polynomials $x$, $y$ of the type `DOM_POLY`, resp. the matrix $X$, $Y$ of the type `Dom::Matrix(polmat::poly)`

**Related Functions**
`polmat::axbyc`, `polmat::xaybc`

**Examples**

```
>> A:=polmat([[1, 0, -d, d^2],[0, -d+d^3, d^2-d^4, d-d^3],[1, -1+d^2,
   -d^3, 1]], d)
```

```
             +-                                    -+
             |                                2      |
             |  1,    0,        -d,        d       |
             |                                       |
             |      3          4    2     3          |
             |  0, d  - d, - d  + d , - d  + d       |
             |                                       |
             |      2             3                  |
             |  1, d  - 1,    - d ,        1         |
             +-                                    -+
>> B:=polmat([[0,1-d]], d)
```

```
                        +-           -+
                        | 0, - d + 1 |
                        +-           -+
```

```
>> C:=polmat([[1, 1+d],[0, -1-d+d^2+d^4],[1, 1+d^3]], d)

                    +-                       -+
                    |  1,       d + 1         |
                    |                         |
                    |       4    2            |
                    |  0, d  + d  - d - 1     |
                    |                         |
                    |            3            |
                    |  1,       d  + 1        |
                    +-                       -+
>> [X, Y]:=polmat::axybc(A, B, C, d)

[

array(1..4, 1..2,
                2
  (1, 1) = - d  w[2, 4, 2] + d w[2, 3, 2] + 1,
  (1, 2) = - d^2*w[2, 4, 1] + d*w[1, 1, 1] + d*w[2, 3, 1] + d\
 - w[1, 1, 1] + 1,
  (2, 1) = d w[2, 3, 2] + w[2, 4, 2],
  (2, 2) = d w[2, 3, 1] + w[1, 2, 1] + w[2, 4, 1],
  (3, 1) = w[2, 3, 2],
  (3, 2) = w[2, 3, 1],
  (4, 1) = w[2, 4, 2],
  (4, 2) = w[2, 4, 1]
)


     +-                                                          -+
     |                        w[1, 1, 1]                          |
     |                                                            |
     |      3    2                2                               |
   , |  - d  + d  w[1, 2, 1] - d  + d w[1, 2, 1] - 2 d - 1       |
     |                                                            |
     |       2                                                    |
     |    - d  + d w[1, 2, 1] - d + w[1, 1, 1] + w[1, 2, 1]       |
     +-                                                          -+

   ]
>> polmat::norm(A*X+Y*B-C)

                                0
```

## polmat::xaybc – solution to equation

polmat::xaybc finds a solution to linear equation with polynomials or polynomial matrices. For polynomials, the equation has the form

$$x\,a + y\,b = c,$$

where $a$, $b$, $c$ are given polynomials and $x$, $y$ are unknown polynomials, i. e., a solution to be determined. For polynomial matrices, the equation has the form

$$X\,A + Y\,B = C,$$

where $A$, $B$, $C$ are given polynomial matrices and $X$, $Y$ are unknown polynomial matrices, i. e., a solution to be determined.

**Call(s)**
```
polmat::xaybc(a, b, c, x)
polmat::xaybc(A, B, C, x)
```

**Parameters**
```
a, b, c – the polynomials
A, B, C – the polynomial matrices of type
  Dom::Matrix(polmat::poly)
x – an indeterminate
```

**Return Value** the polynomials $x$, $y$ of the domain DOM_POLY or the matrices $X$, $Y$ of the type Dom::Matrix(polmat::poly)

**Related Functions** polmat::axbyc, polmat::axybc, polmat::gcd

**Details**

- The general solution to the equation $X\,A + Y\,B = C$ is

$$\begin{aligned} X &= C_1\,P_1 + T\,R_1 \\ Y &= C_1\,Q_1 + T\,S_1 \end{aligned}$$

  where $T$ is an arbitrary matrix and

$$\begin{aligned} C &= C_1\,D_1, \\ P_1\,A + Q_1\,B &= D_1 \\ R_1\,A + S_1\,B &= 0. \end{aligned}$$

**Examples**

```
>> A:=polmat([[1, 1-d],[d, d-d^2]], d)
                        +-              -+
                        |  1,   - d + 1  |
                        |                |
                        |         2      |
                        |  d, - d  + d   |
                        +-              -+
```

```
>> B:=polmat([[1+d, 1-d],[0, 0]], d)
```

$$\begin{bmatrix} d+1, & -d+1 \\ 0, & 0 \end{bmatrix}$$

```
>> C:=polmat([[1, 1-d],[1+d, 1-d^2]], d)
```

$$\begin{bmatrix} 1, & -d+1 \\ 2 & \\ d+1, & -d+1 \end{bmatrix}$$

```
>> [X, Y]:=polmat::xaybc(A, B, C, d)
```

$$\left[\begin{bmatrix} -d\,t[1,1]+1, & t[1,1] \\ -d\,t[2,1]+d+1, & t[2,1] \end{bmatrix}, \begin{bmatrix} 0, & t[1,2] \\ 0, & t[2,2] \end{bmatrix}\right]$$

## polmat::axbycnD – solution to equation

polmat::axbycnD finds a solution to linear equation with multivariate polynomials of the form

$$a\,x + b\,y = c,$$

where $a$, $b$, $c$ are the given multivariate polynomials and $x$, $y$ are unknowns.

| | |
|---|---|
| **Call(s)** | polmat::axbycnD(a, b, c, x, n) |
| **Parameters** | a, b, c – polynomial expressions<br>x – an indeterminate<br>n – number of indeterminates |
| **Return Value** | the polynomial expressions $x$, $y$ |
| **Related Functions** | polmat::axbyc, polmat::axybc, polmat::xaybc, polmat::gcd |

**Examples**

```
>> a:=z[1]+z[2]*z[1]+z[3]

                          z[1] z[2] + z[1] + z[3]
>> b:=z[1]*z[2]

                                z[1] z[2]
>> c:=z[1]*z[2]*z[3]

                             z[1] z[2] z[3]
>> [x,y]:=polmat::axbycnD(a,b,c,z,3)

                                [0, z[3]]
>> norm(a*x+b*y-c)

                                    0
```

## polmat::spf – polynomial spectral factorization

polmat::spf computes the polynomial spectral factorization.

**Call(s)**

```
polmat::spf(b, x)
polmat::spf(Z, x)
```

**Parameters**

b – a polynomial
Z – a polynomial matrix of type Dom::Matrix(polmat::poly)
x – an indeterminate

**Return Value**

an element of DOM_LIST which contains a polynomial or an expression, number of iteration and toleration for polynomial as input, an element of DOM_LIST which contains two matrices of type Dom::Matrix(polmat::poly) for matrix as input

**Details**

- For the given polynomial $b$ the algorithm solves the equation

  $$a\,a^* = b,$$

  where $a^*(s) = a(-s)$ in the continue-time case, $a^*(z) = a(z^{-1})$ in the discrete-time case. The algorithm is based on a Newton-Raphson iterative scheme and polmat::axxab. The Newton-Raphson method is stopped when $|a_i - a_{i-1}| <$ polmat::spf::tol or number of iterations = polmat::spf::noi.
- For the given polynomial matrix $Z$ the algorithm solves the equation

  $$Z = P^* J P,$$

  where $P^*(s) = P^{\mathrm{T}}(-s)$ and $J$ is the signature matrix having the form

  $$J = \begin{pmatrix} I_1 & 0 & 0 \\ 0 & -I_2 & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

  The algorithm is based on a diagonalization and scalar version of polmat::spf.
- The default value of polmat::spf::tol is $10^{-8}$.
- The default value of polmat::spf::noi is 30.
- The implemented algorithms are described in [4, 5, 6].

**Examples**

```
>> a:=polmat(-s^6-s^4+s^2+1, s)

                                  6     4     2
                          poly(- s   - s   + s   + 1, [s])
>> [x,n,eps]:=polmat::spf(a,s)
```

```
                          3    2
                 [poly(s  + s  + s + 1, [s]), 2, 0.0]
>> b:=polmat(-d^3+5*d^2-16*d+44)

                           3      2
                  poly(- d  + 5 d  - 16 d + 44, [d])
>> [x,n,eps]:=polmat::spf(b,d):
>> float(expr(x)), n, eps

             2                                      3
0.7493581013 d  - 2.292460189 d - 0.1618875929 d  + 6.177125636, 4,

   4.418332438e-11
>> Z:=polmat([[0, 1-s],[1+s, 1-s^2]], s)

                      +-                  -+
                      |    0,    - s + 1   |
                      |                    |
                      |            2       |
                      |  s + 1, - s  + 1   |
                      +-                  -+
>> [P, J]:=polmat::spf(Z, s)

  -- +-                                       -+               --
  |  |                        2                 |  +-       -+  |
  |  |  1/2 s + 1/2, - 1/4 s  - 1/2 s + 5/4     |  |  1,  0  |  |
  |  |                                          |, |         |  |
  |  |                        2                 |  |  0, -1  |  |
  |  |  1/2 s + 1/2, - 1/4 s  - 1/2 s - 3/4     |  +-       -+  |
  -- +-                                       -+               --
>> polmat::norm(polmat::conjugatetranspose(P, s)*J*P-Z)

                                0
```

## polmat::care – solution to algebraic Riccati equation with polynomials

polmat::care solves the continuous-time algebraic Riccati equation with scalar polynomials

$$(a + a^*)\,x - b\,b^* x^2 + q = 0,$$

where $a(z_1, \ldots, z_n)$ and $b(z_1, \ldots, z_n)$ are given two-sided polynomials with symmetric expansion, $q(z_1, \ldots, z_n)$ is a two-sided polynomial with symmetric expansion positive for all $|z_1| = 1, \ldots, |z_n| = 1$ and $x(z_1, \ldots, z_n)$ is an unknown two-sided polynomial with symmetric expansion positive for all $|z_1| = 1, \ldots, |z_n| = 1$.

| | |
|---|---|
| **Call(s)** | `polmat::care(a, b, q, vars, N<, Numeric>)` |
| **Parameters** | `a, b, q` – two-sided multivariate polynomials of the type DOM_EXPR<br>`vars` – a list of indeterminates<br>`N` – an integer<br>`Numeric` – numerical algorithm is used instead of symbolical one |
| **Return Value** | a list of two values of the type DOM_EXPR |
| **Related Functions** | `polmat::dare` |

### Details

- A polynomial with symmetric expansion has generally form

$$p(z_1, z_2, \ldots, z_n) = \sum_{k_1=-d_1}^{d_1} \sum_{k_2=-d_2}^{d_2} \cdots \sum_{k_n=-d_n}^{d_n} g_{k_1,k_2,\ldots,k_n}\, z_1^{k_1}\, z_2^{k_2} \cdots z_n^{k_n},$$

  where coefficients $g_{k_1,k_2,\ldots,k_n} \in \mathbb{R}$ are symmetric about the origin, i. e., $g_{k_1,k_2,\ldots,k_n} = g_{-k_1,-k_2,\ldots,-k_n}$ and similarly.
- The algorithm uses **MuPAD** implementation of the fast Fourier transform (FFT).
- The parameter $N$ is equal to the number of sampling points, $N = 2^m, m \in \mathbb{Z}$. If $n > 1$, the number of sampling points can be defined for each variable separately. In this case $N$ is a list containing $n$ integers.
- If the parameter `Numeric` is given then the algorithm performs the numerical FFT else the algorithm performs the symbolical FFT.
- The algorithm is based on [7, 8] and is implemented for $1 \le n \le 3$.

**Examples**

Finite solution exists

```
>> a:=w+5+1/w: b:=1: q:=4*w+24+4/w:

>> [x1,x2]:=polmat::care(a,b,q,[w],2^2)

                          --       2          --
                          |  2 w + - + 12, -2  |
                          --       w          --
```

Infinite solution exists only

```
>> a:=w+5+1/w: b:=1: q:=1:

>> [x1,x2]:=polmat::care(a,b,q,[w],2^2)

--                  1.978107965   0.004413305606                    2
|  1.978107965 w + ------------ + --------------- + 0.004413305606 w  -
|                        w               2
--                                      w

   0.0009104273771                     3
   --------------- - 0.0009104273771 w  + 10.10750124,
           3
          w

                    0.0218920347   0.004413305606
   0.0218920347 w + ------------ - --------------
                          w               2
                                         w

                   2   0.0009104273771                      3
   0.004413305606 w  + --------------- + 0.0009104273771 w  -
                              3
                             w

                --
   0.1075012369  |
                 |
                --
```

## polmat::dare – solution to algebraic Riccati equation with polynomials

polmat::dare solves the discrete-time algebraic Riccati equation with scalar polynomials

$$(a + a^*)\,x - x - \frac{a\,a^* b\,b^* x^2}{1 + b\,b^* x} + q = 0,$$

where $a(z_1, \ldots, z_n)$ and $b(z_1, \ldots, z_n)$ are given two-sided polynomials with symmetric expansion, $q(z_1, \ldots, z_n)$ is a two-sided polynomial with symmetric expansion positive for all $|z_1| = 1, \ldots, |z_n| = 1$ and $x(z_1, \ldots, z_n)$ is an unknown two-sided polynomial with symmetric expansion positive for all $|z_1| = 1, \ldots, |z_n| = 1$.

**Call(s)** polmat::dare(a, b, q, vars, N<, Numeric>)

**Parameters**
- a, b, q – two-sided multivariate polynomials of the type DOM_EXPR
- vars – a list of indeterminates
- N – an integer
- Numeric – numerical algorithm is used instead of symbolical one

**Return Value** a list of two values of the type DOM_EXPR

**Related Functions** polmat::care

### Details

- A polynomial with symmetric expansion has generally form

$$p(z_1, z_2, \ldots, z_n) = \sum_{k_1=-d_1}^{d_1} \sum_{k_2=-d_2}^{d_2} \cdots \sum_{k_n=-d_n}^{d_n} g_{k_1,k_2,\ldots,k_n}\, z_1^{k_1}\, z_2^{k_2} \cdots z_n^{k_n},$$

  where coefficients $g_{k_1,k_2,\ldots,k_n} \in \mathbb{R}$ are symmetric about the origin, i. e., $g_{k_1,k_2,\ldots,k_n} = g_{-k_1,-k_2,\ldots,-k_n}$ and similarly.
- The algorithm uses **MuPAD** implementation of the fast Fourier transform (FFT).
- The parameter $N$ is equal to the number of sampling points, $N = 2^m, m \in \mathbb{Z}$. If $n > 1$, the number of sampling points can be defined for each variable separately. In this case $N$ is a list containing $n$ integers.
- If the parameter Numeric is given then the algorithm performs the numerical FFT else the algorithm performs the symbolical FFT.
- The algorithm is based on [8] and is implemented for $1 \leq n \leq 3$.

## polmat::decomp – right and left decomposition of a matrix

polmat::decomp finds the right or the left decomposition

$$S = B_1 A_1^{-1}, \qquad S = A_1^{-1} B_1.$$

of a given matrix $S$. The matrix $S$ must be given as

$$S = \frac{B}{a},$$

where $B$ is a matrix and $a$ is a polynomial.

**Call(s)** polmat::decomp(B, a, x<, *left*>)

**Parameters**
B – a matrix of a type Dom::Matrix(polmat::poly)
a – a polynomial of a domain DOM_POLY
x – an indeterminate

**Return Value** the matrices $A_1, B_1, A_2, B_2$ of the domain Dom::Matrix

**Details**

- If the parameter *left* is given the left decomposition is computed else the right decomposition is computed.

**Examples**

```
>> B:=polmat([[d-d^2, d-d^2],[1-2*d+d^2, -d^2],[d-d^2, d^2]], d)
```

```
                          +-                          -+
                          |       2           2       |
                          |    - d  + d,   - d  + d   |
                          |                           |
                          |   2                  2    |
                          |  d  - 2 d + 1,   - d      |
                          |                           |
                          |       2            2      |
                          |    - d  + d,      d       |
                          +-                          -+
```

```
>> a:=polmat(1-d)
```

```
                               poly(- d + 1, [d])
```

```
>> polmat::decomp(B, a, d, left)
```

$$\begin{bmatrix} 0, & 1, & 1 \\ 1, & -d, & -d \\ d, & -2\,d^2+d, & -2\,d^2+2\,d-1 \end{bmatrix}, \begin{bmatrix} 1, & 0 \\ 0, & d \\ 0, & 0 \end{bmatrix}$$

## polmat::diagonalization – matrix diagonalization

For a given square matrix $A$, polmat::diagonalization computes such uni-modular matrices $P$ and $Q$ that $PAQ = S$ is a diagonal matrix.

**Call(s)** polmat::diagonalization(A, x<, *all*>)

**Parameters** A – a matrix of the domain Dom::Matrix
x – an indeterminate

**Return Value** the matrices $P$, $Q$ and $S$ of the domain Dom::Matrix

### Details

- If the parameter *all* is given all three matrices $P$, $Q$ and $S$ are returned else only the matrix $S$ is returned.

### Examples

```
>> A:=polmat([[1-d^2, 1-d],[0, 1+d]], d)
                      +-                   -+
                      |     2                |
                      |  - d  + 1, - d + 1   |
                      |                      |
                      |      0,     d + 1    |
                      +-                   -+
>> [P, S, Q]:=polmat::diagonalization(A, d, all)
--
|  +-                    -+
|  |     1,        1      |
|  |                       |
|  |  d         d          |,
|  |  - + 1/2, - - 1/2     |
|  |  2         2          |
-- +-                    -+
                                                  +-                -+ --
   +-                                        -+   |  0,      1       |  |
   |  2,                 0                    |   |                  |  |
   |                                          |   |       2          |  |
   |             3         2                  |,  |      d           |  |
   |  0, - 1/2 d  - 1/2 d  + 1/2 d + 1/2      |   |  1, -- - 1/2     |  |
   +-                                        -+   |       2          |  |
                                                  +-                -+ --
>> polmat::norm(P*A*Q-S)

                                0
```

## polmat::hermiteForm – matrix reduction to Hermite form

For a given matrix $A$, polmat::hermiteForm computes such matrices $P$ and $Q$ that $PA$ and $AQ$ is in row and column Hermite form, respectively.

| | |
|---|---|
| **Call(s)** | polmat::hermiteForm(m, x<, *col*>) |
| **Parameters** | m – a matrix of the domain Dom::Matrix<br>x – an indeterminate |
| **Return Value** | list of two matrices of the domain<br>Dom::Matrix(polmat::poly) |
| **Related Functions** | polmat::smithForm |

### Details

- If the parameter *col* is given column Hermite form is computed else row Hermite form is computed.
- The algorithm is adopted from [9].

### Examples

```
>> A:=polmat([[s^2, 0],[0, s^2], [1, s+1]], s)
```

$$\begin{bmatrix} s^2 & 0 \\ 0 & s^2 \\ 1 & s+1 \end{bmatrix}$$

```
>> [X, P]:=polmat::hermiteForm(A, s)
```

$$\left[\begin{bmatrix} 1 & s+1 \\ 0 & s^2 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & s+1 & -s \end{bmatrix}\right]$$

## polmat::isreduced – a test of a matrix reduction

polmat::isreduced checks whether a given matrix is reduced.

**Call(s)**
polmat::isreduced(m, x<, *col*>)
polmat::isreduced(m, x<, *row*>)

**Parameters**
m – a matrix of the domain Dom::Matrix
x – an indeteminate
*col* – checks the column reduction
*row* – checks the row reduction

**Return Value**
a value of the type DOM_BOOL

**Related Functions**
polmat::reduce

### Details

- If neither *col* nor *row* in the parameters is given a test of the column reduction is performed.

### Examples

```
>> A:=polmat([[d, -d+1],[5*d^2, d^2]], d)
```

$$\begin{pmatrix} d & -d+1 \\ 5d^2 & d^2 \end{pmatrix}$$

```
>> polmat::isreduced(A, d, col)

                                  FALSE
>> polmat::isreduced(A, d, row)

                                  TRUE
```

## polmat::popovForm – Popov form

polmat::popovForm reduces a given polynomial matrix to the *Popov form* (*polynomial-echelon form*), see [9] for details.

**Call(s)**
```
polmat::popovForm(A, x<, col>)
polmat::popovForm(A, x, row)
```

**Parameters**
A – a matrix of the domain Dom::Matrix(polmat::poly)
x – an indeterminate
*col* – a reduction to the column Popov form
*row* – a reduction to the row Popov form

**Return Value** a matrix of the domain Dom::Matrix

**Related Functions** polmat::hermiteForm, polmat::isreduced, polmat::reduce, polmat::smithForm

**Details**

- A polynomial matrix $A(s)$ is in the Popov form form if
  1. it is column reduced, with its column degrees arranged in ascendings order, say
     $$k_1 \leq k_2 \leq \cdots k_m,$$
  2. for column $j$, $1 \leq j \leq m$, there is a so-called *pivot index* $p_j$ such that
     (a) $a_{p_j j}(s)$ has degree $k_j$,
     (b) $a_{p_j j}(s)$ is monic,
     (c) $a_{p_j j}(s)$ is the last (or lowest) entry of degree $k_j$ in the $j$-th column; i. e., $\partial a_{ij}(s) < k_j$ if $i > p_j$,
     (d) if $k_i = k_j$ and $i < j$, then $p_i < p_j$; i. e., the pivot indices are arranged to be increasing,
     (e) $a_{p_j i}(s)$ has degree less than $k_j$ if $i \neq j$.

**Examples**

```
>> A:=polmat([[-3*s, s+2],[1-s, 1]], s)
```

```
                        +-                  -+
                        |    (-3) s, s + 2   |
                        |                    |
                        |  - s + 1,    1     |
                        +-                  -+
>> polmat::popovForm(A, s, col)
```

```
                         +-                -+
                         |  s + 2,    -6    |
                         |                  |
                         |    1,    s - 4   |
```

```
                          +-                -+
>> B:=polmat([[-3*s, s+2, 3],[1-s, 1, s^2], [1,s, 9]], s)

                      +-                       -+
                      |   (-3) s, s + 2,  3    |
                      |                        |
                      |                      2 |
                      |  - s + 1,   1,    s    |
                      |                        |
                      |     1,       s,    9   |
                      +-                       -+
>> polmat::popovForm(B, s, row)

                     +-                          -+
                     |  s + 1/3, -2/3,    2     |
                     |                          |
                     |     1,      s,     9     |
                     |                          |
                     |                    2     |
                     |    4/3,    1/3, s  + 2   |
                     +-                         -+
```

## polmat::reduce – a reduction of polynomial matrix

polmat::reduce performs a reduction to so-called reduced form or reduction by a right or a left divisor.

**Call(s)**

```
polmat::reduce(A, x<, row>)
polmat::reduce(A, x, col)
polmat::reduce(C, D, x<, right>)
polmat::reduce(C, D, x, left)
```

**Parameters**

A, C, D – matrices of the domain Dom::Matrix(polmat::poly)
x – an indeterminate
*left* – reduction by a left divisor will be performed
*right* – reduction by a left divisor will be performed
*row* – reduction to row reduced form will be performed
*col* – reduction to column reduced form will be performed

**Return Value**

a matrix of the domain Dom::Matrix, list of two matrices of the domain Dom::Matrix

**Related Functions**

polmat::gcd, polmat::isreduced

### Details

- For a given matrix $A$, polmat::reduce(A, x<, *row*>) and polmat::reduce(A, x, *col*) return row and column reduced matrix, respectively. The algorithm computes such matrices $A_r$, $A_c$ and such uni-modular matrices $U$, $V$ that

  $$A_r = UA, \qquad A_c = AV,$$

  respectively. The algorithm performs elementary operations on the polynomial matrix.
- If the optional parameter *col* is given the column reduction of $A$ is computed. If no optional parameter is given the row reduction of $A$ is computed.
- For given matrices $C$ and $D$, the procedures polmat::reduce(C, D, x<, *right*>) and polmat::reduce(C, D, x, *left*) compute such polynomial matrices $C_1$ and $C_2$ that

  $$C = C_1 D,$$

  where $D$ is the right divisor of $C$, and

  $$C = D C_2,$$

  where $D$ is the left divisor of $C$. The divisors $D$ must be in the triangular matrix form.
- If the optional parameter *left* is given the reduction by left divisor is performed. If no optional parameter is given the reduction by right divisor is performed.

**Examples**

```
>> A:=polmat([[d, -d+1],[5*d^2, d^2]], d)
```

$$\begin{bmatrix} d, & -d+1 \\ 5d^2, & d^2 \end{bmatrix}$$

```
>> [Ar, U]:=polmat::reduce(A, d)
```

$$\left[\begin{bmatrix} d, & -d+1 \\ 5d^2, & d^2 \end{bmatrix}, \begin{bmatrix} 1, & 0 \\ 0, & 1 \end{bmatrix}\right]$$

```
>> polmat::isreduced(Ar, s, row)
```

TRUE

```
>> polmat::norm(U*A-Ar)
```

0

```
>> [Ac, V]:=polmat::reduce(A, d, col)
```

$$\left[\begin{bmatrix} d, & 1 \\ 5d^2, & 6d \end{bmatrix}, \begin{bmatrix} 1, & 1 \\ 0, & 1 \end{bmatrix}\right]$$

```
>> polmat::isreduced(Ac, s, col)
```

TRUE

```
>> polmat::norm(A*V-Ac)
```

0

```
>> C:=polmat([[1+d, 1+d, 1-d^2],[1+d, 1-d, 1],[2, 0, 2-d]], d)
```

$$\begin{bmatrix} d+1, & d+1, & -d^2+1 \\ d+1, & -d+1, & 1 \\ 2, & 0, & -d+2 \end{bmatrix}$$

```
>> De:=polmat([[1+d, 0],[1,d],[1 ,1]], d)
```

$$\begin{bmatrix} d+1, & 0 \\ 1, & d \\ 1, & 1 \end{bmatrix}$$

```
>> polmat::reduce(C, De, d, left)
```

$$\begin{bmatrix} 1, & 1, & -d+1 \\ 1, & -1, & 1 \end{bmatrix}$$

## polmat::smithForm – matrix reduction to Smith form

For a given matrix $A$, polmat::smithForm computes such uni-modular matrices $P$ and $Q$ that

$$PAQ = \mathrm{diag}_{l,m}[a_1,\ a_2,\ \ldots,\ a_r,\ 0,\ \ldots,\ 0].$$

| | |
|---|---|
| **Call(s)** | `polmat::smithForm(A, x)` |
| **Parameters** | `A` – a matrix of the domain `Dom::Matrix`<br>`x` – an indeterminate |
| **Return Value** | the matrices $P$, $Q$ and $PAQ$ of the domain `Dom::Matrix`, rank $r$ of $A$ of the type `DOM_INT` |
| **Related Functions** | `polmat::hermiteForm` |

**Examples**

```
>> A:=polmat([[1-d^2, 0, 1-d],[0, 1+d, 0]], d)
```

```
                        +-                            -+
                        |     2                        |
                        |  - d  + 1,    0,    - d + 1  |
                        |                              |
                        |       0,    d + 1,     0     |
                        +-                            -+
```

```
>> [P, B, Q, r]:=polmat::smithForm(A, d)
```

```
--
|
|  +-                    -+  +-                        -+
|  |     1,        1       |  |  2,        0,       0   |
|  |                       |  |                          |
|  |  d         d          |, |           2              |,
|  |  - + 1/2, - - 1/2     |  |  0, 1/2 d  - 1/2, 0      |
|  |  2         2          |  +-                        -+
|  +-                    -+
|
--
```

```
   +-                          -+     --
   |  0,      0,         1      |      |
   |                            |      |
   |        d                   |      |
   |  1, - - + 1/2,     0       |      |
   |        2                   |, 2   |
   |                            |      |
   |        d                   |      |
   |  1, - - - 1/2, - d - 1     |      |
   |        2                   |      |
   +-                          -+     --
```

```
>> polmat::norm(P*A*Q-B)

                                  0
```

## polmat::svd – singular value decomposition

For a given matrix $A \in \mathbb{R}^{m,n}$, `polmat::svd` computes matrices $U \in \mathbb{R}^{m,m}$, $S \in \mathbb{R}^{m,n}$, $V \in \mathbb{R}^{n,n}$ that $A = U S V^{\mathrm{T}}$.

| | |
|---|---|
| **Call(s)** | `polmat::svd(A,`<*Numeric*>`)` |
| **Parameters** | `A` – a matrix of a domain `Dom::Matrix()`<br>*Numeric* – choice of a computing method |
| **Return Value** | a list of matrices of the domain `Dom::Matrix()` |

### Details

- The classical and numerically unstable algorithm is implemented. It is based on computing a symmetric matrix $B = \hat{A}^{\mathrm{T}}\hat{A}$ where $\hat{A}_{n,n}$ is formed by joining the matrices $A_{m,n}$ and $\mathbf{0}_{n-m,n}$ vertically, where we consider $m \leq n$. Then $S = \mathrm{diag}_{m,n}[\sigma_i]$, $\sigma_i = \sqrt{\lambda_i}, i = 1,2,\ldots,n$, where $\lambda_1,\ldots,\lambda_n$ are eigenvalues of $B$ sorted by value. The columns of a matrix $V$ are eigenvectors of $B$ coresponding to $\lambda_1,\ldots,\lambda_n$ and the matrix $U$ is given by the relation

$$u_i = \frac{1}{\sigma_i} A\, v_i.$$

- If the parameter *Numeric* is given algorithm computes SVD by numerical function of MuPAD library.

### Examples

```
>> A:=matrix([[1, 2],[2, 1]])
```

$$\begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix}$$

```
>> [U, S, V]:=polmat::svd(A)
```

$$\left[\begin{pmatrix} \frac{1}{2}\sqrt{2} & \frac{1}{2}\sqrt{2} \\ \frac{1}{2}\sqrt{2} & -\frac{1}{2}\sqrt{2} \end{pmatrix}, \begin{pmatrix} 3 & 0 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} \frac{1}{2}\sqrt{2} & \frac{1}{2}\sqrt{2} \\ \frac{1}{2}\sqrt{2} & -\frac{1}{2}\sqrt{2} \end{pmatrix}\right]$$

```
>> polmat::norm(U*S*linalg::transpose(V)-A)
```

$$0$$

## polmat::adj – adjoint matrix and determinant

For a given matrix $A$, `polmat::adj` computes the adjoint matrix $\operatorname{adj} A$ and the determinant $\det A$.

**Call(s)** `polmat::adj(A, x)`

**Parameters** `A` – object of the domain `Dom::Matrix(polmat::poly)`
`x` – an indeterminate

**Return Value** the matrix $\operatorname{adj} A$ of the domain `Dom::Matrix`, $\det A$ of the domain `polmat::poly`

**Related Functions** `polmat::inverse`

**Examples**

```
>> A:=polmat([[0, 0, 1-d],[1, 1-d, 0],[1, -1-d, -1-d]], d)
```

```
                          +-                        -+
                          |  0,    0,    - d + 1    |
                          |                         |
                          |  1, - d + 1,    0       |
                          |                         |
                          |  1, - d - 1, - d - 1    |
                          +-                        -+
>> [adj, det]:=polmat::adj(A, d)
```

```
             -- +-                                  -+          --
             |  |   2       2          2             |           |
             |  |  d  - 1, d  - 1, - d  + 2 d - 1    |           |
             |  |                                    |           |
             |  |   d + 1,  d - 1,     - d + 1       |, 2 d - 2  |
             |  |                                    |           |
             |  |    -2,      0,          0          |           |
             -- +-                                  -+          --
```

## polmat::coeff – coefficients of a polynomial or polynomial matrix

polmat::coeff computes coefficients of a given polynomial or polynomial matrix.

**Call(s)**
```
polmat::coeff(A, x<, n>)
polmat::coeff(p, x<, n>)
```

**Parameters**
A – a matrix of the domain Dom::Matrix
p – a polynomial of the type DOM_POLY or an expression
x – an indeterminate
n – a number of the domain DOM_INT

**Return Value**
an element of the domain or the type DOM_TABLE or DOM_EXPR, Dom::Matrix, DOM_INT, DOM_RAT, ...

### Details

- If a number n is given the function returns the coefficient of the term $x^n$.

### Examples

```
>> a:=polmat(s+v*s^2, s)
```

$$\mathrm{poly}(v\,s^2 + s, [s])$$

```
>> polmat::coeff(a, s)
```

```
table(
  1 = 1,
  2 = v
)
```

```
>> A:=polmat([[d, d+1],[5*d, d+m]], d)
```

$$\begin{pmatrix} d & d+1 \\ 5\,d & m+d \end{pmatrix}$$

```
>> polmat::coeff(A, d)
```

```
table(
                +-       -+
                |  1, 1   |
          1 =   |         |,
                |  5, 1   |
                +-       -+
                +-       -+
                |  0, 1   |
          0 =   |         |
                |  0, m   |
                +-       -+
)
```

## polmat::conjugate – conjugate polynomial

polmat::conjugate computes the complex conjugate of a given polynomial.

| | |
|---|---|
| **Call(s)** | polmat::conjugate(p, x) |
| **Parameters** | p – a polynomial of the type DOM_POLY<br>x – an inderterminate |
| **Return Value** | an expression |
| **Related Functions** | polmat::reverse |

### Details

- In the discrete-time case, the polynomial $a$ of the indeterminate $d$ or $z$ is considered in the form $a(d) = \alpha_0 + \alpha_1 d + \cdots + \alpha_n d^n$, $a(z) = \alpha_0 + \alpha_1 z + \cdots + \alpha_n z^n$, respectively, where $\alpha_i \in \mathbb{C}$. The conjugate polynomial is defined as $\bar{a}(d) = \bar{\alpha}_0 + \bar{\alpha}_1 d^{-1} + \cdots + \bar{\alpha}_n d^{-n}$, $\bar{a}(z) = \bar{\alpha}_0 + \bar{\alpha}_1 z^{-1} + \cdots + \bar{\alpha}_n z^{-n}$, respectively.
- In the continuous-time case, the polynomial $a$ of the indeterminate $s$ is considered in the form $a(s) = \alpha_0 + \alpha_1 s + \cdots + \alpha_n s^n$, where $\alpha_i \in \mathbb{C}$. The conjugate polynomial is defined as $\bar{a}(s) = \bar{\alpha}_0 + \bar{\alpha}_1 (-s) + \cdots + \bar{\alpha}_n (-s)^n$.

### Examples

```
>> u:=polmat(s^3+(5-I)*s^2+(18+7*I)*s-15)

                    3            2
               poly(s  + (5 - I) s  + (18 + 7 I) s - 15, [s])
>> polmat::conjugate(u, s)

                           2                    3
                     (5 + I) s  - (18 - 7 I) s - s  - 15
>> v:=subs(u, s=d)

                    3            2
               poly(d  + (5 - I) d  + (18 + 7 I) d - 15, [d])
>> polmat::conjugate(v, d)

                        18 - 7 I   5 + I    1
                        -------- + ----- + -- - 15
                            d         2     3
                                      d     d
```

## polmat::conjugatetranspose – conjugate transposition of a matrix

polmat::conjugatetranspose returns the conjugate transposition of the polynomial matrix.

**Call(s)** polmat::conjugatetranspose(A, x)

**Parameters** A – a matrix of the domain Dom::Matrix()
x – an indeterminate

**Return Value** a matrix of the domain Dom::Matrix()

**Related Functions** polmat::transpose

**Examples**

```
>> A:=polmat([[(1+I)*d, 8],[12-5*I, 2*d]], d)
```

```
                          +-                  -+
                          |  (1 + I) d,   8    |
                          |                    |
                          |   12 - 5 I, 2 d    |
                          +-                  -+
```

```
>> B:=polmat::conjugatetranspose(A, d)
```

```
                          +-                    -+
                          |  1 - I               |
                          |  -----, 12 + 5 I     |
                          |    d                 |
                          |                      |
                          |              2       |
                          |    8,        -       |
                          |              d       |
                          +-                    -+
```

## polmat::degree – degrees of a polynomial matrix

polmat::degree computes degrees of a polynomial matrix.

**Call(s)**

```
polmat::degree(A, x<, all>)
polmat::degree(A, x<, col>)
polmat::degree(A, x<, row>)
```

**Parameters**

A – a matrix of the domain Dom::Matrix
x – an indeterminate
*all* – computes a degree of every polynomial in a polynomial matrix
*col* – computes the greatest degrees in the columns
*row* – computes the greatest degrees in the rows

**Return Value**

a matrix of the domain Dom::Matrix

### Details

- The default method is a computing the degree of every polynomial in a polynomial matrix.

### Examples

```
>> A:=polmat([[d, 4*d^2-d+1],[5+1/d, d]], d)
```

$$\begin{pmatrix} d & 4d^2 - d + 1 \\ 5 + \frac{1}{d} & d \end{pmatrix}$$

```
>> polmat::degree(A, d)
```

$$\begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix}$$

```
>> polmat::degree(A, d, row)
```

```
+-   -+
|  2  |
|     |
|  1  |
+-   -+
```

## polmat::det – polynomial matrix determinant

polmat::det computes the determinant of a given polynomial matrix.

| | |
|---|---|
| **Call(s)** | polmat::det(A, x<, *Numeric*>) |
| **Parameters** | A – a matrix of the domain Dom::Matrix(polmat::poly)<br>x – an indeterminate |
| **Return Value** | an object of the domain DOM_POLY |

**Details**

- If the parameter *Numeric* is given the numerical computing method based on the fast Fourier transform is used for the computation. This algorithm is described in [10].
- If the parameter *Numeric* is not given the symbolical computing method based on the matrix elementary operations is used for the computation. This algorithm is described in [1].

## polmat::dimen – matrix dimension

polmat::dimen returns the dimension of a given matrix.

**Call(s)** polmat::dimen(A)

**Parameters** A – a matrix of the domain Dom::Matrix

**Return Value** an object of the domain DOM_LIST

**Examples**

```
>> A:=polmat::rand(3,4,3,d):
>> polmat::dimen(A)
```

```
[4, 3]
```

## polmat::divide – quotient of two polynomials

polmat::divide computes the quotient of given polynomials *a* and *b*.

**Call(s)** polmat::divide(a, b, x)

**Parameters** a, b – polynomials of the domain DOM_POLY
x – an indeterminate

**Return Value** an object of the domain DOM_LIST of two polynomials, the first one is the quotient, the second one is the remainder

**Examples**

```
>> a:=polmat(4*x+x^2+20)
```

$$\mathrm{poly}(x^2 + 4\,x + 20, [x])$$

```
>> b:=polmat(x+1)
```

$$\mathrm{poly}(x + 1, [x])$$

```
>> [q, r]:=polmat::divide(a, b, x)
```

$$[\mathrm{poly}(x + 3, [x]), \mathrm{poly}(17, [x])]$$

## polmat::gcd – greatest common divisor of polynomials

polmat::gcd computes the greatest common divisor of given polynomials $a$ and $b$ or polynomial matrices $A$ and $B$.

**Call(s)**

```
polmat::gcd(a, b, x)
polmat::gcd(a, b, x<, all>)
polmat::gcd(A, B, x<, all>)
polmat::gcd(A, B, x<, right<, all>>)
polmat::gcd(A, B, x<, left<, all>>)
```

**Parameters**

`a, b` – polynomials of the type `DOM_POLY`
`A, B` – matrices of the domain
`Dom::Matrix(polmat::poly)`
`x` – an indeterminate

**Return Value**

polynomials $d$, $p$, $q$, $r$, $s$ or matrices $D$, $P$, $Q$, $R$, $S$

**Details**

- For given polynomials $a$ and $b$, polmat::gcd solves the system of equations

$$\begin{aligned} a\,p - b\,q &= d \\ a\,r - b\,s &= 0. \end{aligned}$$

  If the parameter *all* is given all polynomials $d$, $p$, $q$, $r$, $s$ are returned else only the polynomial $d$ is returned.
- For given matrices $A$ and $B$, polmat::gcd solves the system of equations

$$\begin{aligned} P\,A + Q\,B &= D \\ R\,A + S\,B &= \mathbf{0} \end{aligned} \quad \text{or} \quad \begin{aligned} A\,P + B\,Q &= D \\ A\,R + B\,S &= \mathbf{0}. \end{aligned}$$

  If the parameter *all* is given all matrices $D$, $P$, $Q$, $R$, $S$ are returned else only the matrix $D$ is returned.

**Examples**

```
>> a:=polmat(2-d-2*d^2+d^3)

                              3      2
                       poly(d  - 2 d  - d + 2, [d])
>> b:=polmat(-2*d+d^2)

                                 2
                           poly(d  - 2 d, [d])
>> dd:=polmat::gcd(a, b, d)

                          poly(- d + 2, [d])
>> [dd,p,q,r,s]:=polmat::gcd(a, b, d, all)
```

```
[poly(- d + 2, [d]), poly(1, [d]), poly((-1) d, [d]), poly(d, [d]),

           2
   poly(- d  + 1, [d])]
>> A:=polmat([[d-d^2, 2*d-d^2]], d)

                          +-   2         2      -+
                          | - d  + d, - d  + 2 d |
                          +-                    -+
>> B:=polmat([[d-1, d-1],[d-1, d-2]], d)

                             +-              -+
                             |  d - 1, d - 1  |
                             |                |
                             |  d - 1, d - 2  |
                             +-              -+
>> Dd:=polmat::gcd(A, B, d, right)

                                 +-              -+
                                 |  d - 1, d - 1  |
                                 |                |
                                 |    0,     -1   |
                                 +-              -+
>> [Dd, P, Q, R, S]:=polmat::gcd(A, B, d, right, all)

         -- +-              -+  +-   -+  +-       -+                    --
         |  |  d - 1, d - 1  |  |  0  |  |   1, 0  |  +- -+  +-    -+  |
         |  |                |, |     |, |         |, | 1 |, | 0, d |  |
         |  |    0,     -1   |  |  0  |  |  -1, 1  |  +- -+  +-    -+  |
         -- +-              -+  +-   -+  +-       -+                    --
```

## polmat::gensylv – Sylvester matrix

polmat::gensylv computes the Sylvester matrix.

**Call(s)**
```
polmat::gensylv(A, x, col, k)
polmat::gensylv(A, x, row, k)
```

**Parameters**
A – a matrix of the domain Dom::Matrix
x – an indeterminate
*col*, *row* – a column, row Sylvester matrix
k – number of the given columns or rows

**Return Value**
a matrix of the domain Dom::Matrix

**Examples**

```
>> A:=polmat([[1+z, 2+2, 3+z],[4, 5+2*z, 6]], z)
```

$$\begin{pmatrix} z+1 & 4 & z+3 \\ 4 & 2z+5 & 6 \end{pmatrix}$$

```
>> polmat::gensylv(A, z, col, 1)
```

$$\begin{pmatrix} 1 & 4 & 3 & 0 & 0 & 0 \\ 4 & 5 & 6 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 4 & 3 \\ 0 & 2 & 0 & 4 & 5 & 6 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 2 & 0 \end{pmatrix}$$

## polmat::hurwitz – Hurwitz matrix

For a given polynomial or polynomial matrix $A(x)$, polmat::hurwitz returns the corresponding Hurwitz matrix of order $k$.

**Call(s)** polmat::hurwitz(A, x<, k>)

**Parameters**
A – an object of DOM_POLY or DOM_EXPR or Dom::Matrix
x – an indeterminate
k – an integer number

**Return Value** an object of the domain Dom::Matrix

**Examples**

```
>> a:=polmat(s^3+2*s^2-4*s+8)

                        3      2
                 poly(s  + 2 s  - 4 s + 8, [s])
>> H1:=polmat::hurwitz(a, s)

                          +-           -+
                          |  2,  8, 0  |
                          |            |
                          |  1, -4, 0  |
                          |            |
                          |  0,  2, 8  |
                          +-           -+
>> H2:=polmat::hurwitz(a, s, 4)

                        +-               -+
                        |  1, -4,  0, 0  |
                        |                |
                        |  0,  2,  8, 0  |
                        |                |
                        |  0,  1, -4, 0  |
                        |                |
                        |  0,  0,  2, 8  |
                        +-               -+
```

```
>> A:=polmat([[s, s+1],[s^2-9, 7]], s)
```

$$\begin{bmatrix} s, & s+1 \\ s^2-9, & 7 \end{bmatrix}$$

```
>> H3:=polmat::hurwitz(A, s)
```

$$\begin{bmatrix} 1, & 1, & 0, & 0 \\ 0, & 0, & 0, & 0 \\ 0, & 0, & 0, & 1 \\ 1, & 0, & -9, & 7 \end{bmatrix}$$

## polmat::inverse – the inverse of the matrix

polmat::inverse returns the inverse of a given matrix.

| | |
|---|---|
| **Call(s)** | polmat::inverse(A, x) |
| **Parameters** | A – a matrix of the type Dom::Matrix(polmat::poly)<br>x – an indeterminate |
| **Return Value** | a matrix of the domain Dom::Matrix() |
| **Related Functions** | polmat::adj |

### Details

- The algorithm computes the inverse by

$$A^{-1} = \frac{\operatorname{adj} A}{\det A}.$$

### Examples

```
>> A:=polmat([[0, 0, 1-d],[1, 1-d, 0],[1, -1-d, -1-d]], d)
```

```
                          +-                        -+
                          |  0,     0,     - d + 1   |
                          |                          |
                          |  1, - d + 1,     0       |
                          |                          |
                          |  1, - d - 1, - d - 1     |
                          +-                        -+
>> polmat::inverse(A, d)
```

```
                  +-                                      -+
                  |     2          2              2       |
                  |    d  - 1     d  - 1  2 d - d  - 1     |
                  |   -------,  -------, ------------      |
                  |   2 d - 2   2 d - 2     2 d - 2        |
                  |                                        |
                  |    d + 1      d - 1       - d + 1      |
                  |   -------,  -------,     -------      |
                  |   2 d - 2   2 d - 2     2 d - 2        |
                  |                                        |
                  |       2                                |
                  |  - -------,     0,          0          |
                  |    2 d - 2                             |
                  +-                                      -+
```

## polmat::isPos – test of positiveness of a polynomial

polmat::isPos tests whether a $n$-variable symmetric polynomial is positive on the $n$-circle. The algorithm is implemented for $1 \leq n \geq 3$.

**Call(s)** polmat::isPos(p, vars, N <, Numeric, all>)

**Parameters**
p – a two-sided multivariate polynomial of the type DOM_EXPR
vars – a list of indeterminates
N – an integer or a list of integers
Numeric – numerical algorithm is used instead of symbolical one
all – an array of all samples is returned

**Return Value** a value of the type DOM_BOOL and an object of the type DOM_ARRAY

### Details

- A polynomial with symmetric expansion has generally form

$$p(z_1, z_2, \ldots, z_n) = \sum_{k_1=-d_1}^{d_1} \sum_{k_2=-d_2}^{d_2} \cdots \sum_{k_n=-d_n}^{d_n} g_{k_1,k_2,\ldots,k_n} z_1^{k_1} z_2^{k_2} \cdots z_n^{k_n},$$

  where coefficients $g_{k_1,k_2,\ldots,k_n} \in \mathbb{R}$ are symmetric about the origin, i. e., $g_{k_1,k_2,\ldots,k_n} = g_{-k_1,-k_2,\ldots,-k_n}$ and similarly.
- The algorithm uses **MuPAD** implementation of the fast Fourier transform (FFT).
- The parameter $N$ is equal to the number of sampling points, $N = 2^m, m \in \mathbb{Z}$. If $n > 1$, the number of sampling points can be defined for each variable separately. In this case $N$ is a list containing $n$ integers.
- If the parameter Numeric is given then the algorithm performs the numerical FFT else the algorithm performs the symbolical FFT.
- If the parameter all is given then positiveness as well as an array of all samples of the polynomial on the unit $n$-circle are returned.

### Examples

```
>> p:=(z+2)*(1/z+2)
                                  / 1      \
                                  | - + 2  | (z + 2)
                                  \ z      /

>> polmat::isPos(p,[z],2^3)

                                     TRUE

>> polmat::isPos(p,[z],2^2,all)

                                  +-           -+
```

```
                          TRUE, | 9, 5, 1, 5 |
                                +-          -+
```

```
>> polmat::isPos(p,[z],2^3,all)

         +-      1/2                 1/2             1/2         1/2    -+
   TRUE, | 9, 2 2    + 5, 5, 5 - 2 2   , 1, 5 - 2 2   , 5, 2 2    + 5 |
         +-                                                            -+
```

```
>> polmat::isPos(p,[z],2^3,Numeric,all)

                              array(1..8,
                                (1) = 9.0,
                                (2) = 7.828427125,
                                (3) = 5.0,
                         TRUE,  (4) = 2.171572875,
                                (5) = 1.0,
                                (6) = 2.171572875,
                                (7) = 5.0,
                                (8) = 7.828427125
                              )
```

```
>> p:=1+(z+1)*(1/z+1)*(v+2)*(1/v+2)*(w+1/2)*(1/w+1/2)

 / / 1     \ / 1     \ / 1       \     \
 | | - + 2 | | - + 1 | | - + 1/2 | + 1 | ((v + 2) (z + 1) (w + 1/2) + 1)
 \ \ v     / \ z     / \ w       /     /
```

```
>> polmat::isPos(p,[z,v,w],2^2,all)

                                    array(1..4, 1..4, 1..4,
                                      (1, 1, 1) = 113,
                                      (1, 1, 2) = 53,
                                      (1, 1, 3) = -7,
                                      (1, 1, 4) = 53,
                                      (1, 2, 1) = 62,
                                      (1, 2, 2) = 30,
                                      (1, 2, 3) = -2,
                                      (1, 2, 4) = 30,
                                      (1, 3, 1) = 11,
                                      (1, 3, 2) = 7,
                                      (1, 3, 3) = 3,
                                      (1, 3, 4) = 7,
                                      (1, 4, 1) = 62,
                                      (1, 4, 2) = 30,
                                      (1, 4, 3) = -2,
                                      (1, 4, 4) = 30,
                                      (2, 1, 1) = 105/2,
```

```
                                    (2, 1, 2) = 53/2,
                                    (2, 1, 3) = 1/2,
                                    (2, 1, 4) = 53/2,
                                    (2, 2, 1) = 59/2,
                                    (2, 2, 2) = 31/2,
                                    (2, 2, 3) = 3/2,
                                    (2, 2, 4) = 31/2,
                                    (2, 3, 1) = 13/2,
                                    (2, 3, 2) = 9/2,
                                    (2, 3, 3) = 5/2,
                                    (2, 3, 4) = 9/2,
                                    (2, 4, 1) = 59/2,
                                    (2, 4, 2) = 31/2,
                                    (2, 4, 3) = 3/2,
                          FALSE,    (2, 4, 4) = 31/2,
                                    (3, 1, 1) = -8,
                                    (3, 1, 2) = 0,
                                    (3, 1, 3) = 8,
                                    (3, 1, 4) = 0,
                                    (3, 2, 1) = -3,
                                    (3, 2, 2) = 1,
                                    (3, 2, 3) = 5,
                                    (3, 2, 4) = 1,
                                    (3, 3, 1) = 2,
                                    (3, 3, 2) = 2,
                                    (3, 3, 3) = 2,
                                    (3, 3, 4) = 2,
                                    (3, 4, 1) = -3,
                                    (3, 4, 2) = 1,
                                    (3, 4, 3) = 5,
                                    (3, 4, 4) = 1,
                                    (4, 1, 1) = 105/2,
                                    (4, 1, 2) = 53/2,
                                    (4, 1, 3) = 1/2,
                                    (4, 1, 4) = 53/2,
                                    (4, 2, 1) = 59/2,
                                    (4, 2, 2) = 31/2,
                                    (4, 2, 3) = 3/2,
                                    (4, 2, 4) = 31/2,
                                    (4, 3, 1) = 13/2,
                                    (4, 3, 2) = 9/2,
                                    (4, 3, 3) = 5/2,
                                    (4, 3, 4) = 9/2,
                                    (4, 4, 1) = 59/2,
                                    (4, 4, 2) = 31/2,
                                    (4, 4, 3) = 3/2,
                                    (4, 4, 4) = 31/2
                                )
```

## polmat::jury – Jury matrix

For a given polynomial or polynomial matrix $A(x)$, polmat::jury returns the Jury matrix of order $k$.

**Call(s)** polmat::jury(A, x<, k>)

**Parameters**
A – an object of the type DOM_POLY or DOM_EXPR or Dom::Matrix
x – an indeterminate
k – an integer

**Return Value** an object of the domain Dom::Matrix

**Examples**

```
>> a:=z^4+3*z^3-6*z^2+z-9
```

```
                                  2       3    4
                            z - 6 z  + 3 z  + z  - 9
```

```
>> polmat::jury(a, z)
```

```
                                +-           -+
                                |  1,  3, 3   |
                                |             |
                                |  0, 10, 2   |
                                |             |
                                |  9, -1, 7   |
                                +-           -+
```

## polmat::lcoeff – a leading coefficient

polmat::lcoeff returns a leading coefficient of a polynomial or polynomial matrix.

**Call(s)**
polmat::lcoeff(p, x)
polmat::lcoeff(A, x)

**Parameters**
A – a matrix of the domain Dom::Matrix
p – a polynomial of the type DOM_POLY or an expression
x – an indeterminate

**Return Value**
an object of the type or the domain DOM_EXPR, Dom::Matrix, DOM_INT, DOM_RAT, ...

**Related Functions**
polmat::tcoeff, polmat::ldeg, polmat::tdeg

**Examples**

```
>> a:=polmat(x+10*c*x^2, x)
```

```
                                  2
                      poly((10 c) x  + x, [x])
```

```
>> polmat::lcoeff(a, x)
```

```
                                 10 c
```

```
>> A:=polmat([[d, -d+1],[5*d^2, d^2]], d)
```

```
                          +-                -+
                          |    d,  - d + 1   |
                          |                  |
                          |     2       2    |
                          |  5 d ,     d     |
                          +-                -+
```

```
>> polmat::lcoeff(A, d)
```

```
                              +-      -+
                              |  0, 0  |
                              |        |
                              |  5, 1  |
                              +-      -+
```

## polmat::ldeg – a leading degree

polmat::ldeg returns a leading degree of a polynomial or polynomial matrix.

**Call(s)**
```
polmat::ldeg(p, x)
polmat::ldeg(A, x)
```

**Parameters**
A – a matrix of the domain Dom::Matrix
p – a polynomial of the type DOM_POLY or an expression
x – an indeterminate

**Return Value** an object of the type DOM_EXPR or DOM_INT, DOM_RAT, ...

**Related Functions** polmat::lcoeff, polmat::tcoeff, polmat::tdeg

**Examples**

```
>> a:=polmat(x+10*c*x^2+3-2/x, x)
```

$$x - \frac{2}{x} + 10\,c\,x^2 + 3$$

```
>> polmat::ldeg(a, x)
```

$$2$$

```
>> A:=polmat([[d, -d+1],[5*d^2+4/3, d^3]], d)
```

$$\begin{pmatrix} d, & -d+1 \\ 5\,d^2+4/3, & d^3 \end{pmatrix}$$

```
>> polmat::ldeg(A, d)
```

$$3$$

## polmat::longdivide – expansion of transfer function to series

polmat::longdivide computes the expansion of transfer function $c = \frac{b}{a}$ to series.

**Call(s)**
```
polmat::longdivide(a, b, x, k)
polmat::longdivide(c, x, k)
```

**Parameters**
a, b – polynomials of the domain DOM_POLY
c – an expression
x – an indeterminate
k – a number of the computed members

**Return Value**
polynomial of the domain DOM_POLY

**Examples**

```
>> a:=polmat(2-d-2*d^2+d^3)

                              3      2
                       poly(d  - 2 d  - d + 2, [d])
>> b:=polmat(-2*d+d^2)

                                  2
                           poly(d  - 2 d, [d])
>> polmat::longdivide(b, a, 5)

                                    3
                           poly(- d  - d, [d])
>> c:=polmat(1/(1-2*d))

                                     1
                                  -------
                                  1 - 2 d
>> polmat::longdivide(c, 5)

                          4      3      2
                 poly(16 d  + 8 d  + 4 d  + 2 d + 1, [d])
```

## polmat::minreal – a minimal realization

For a given transfer function

$$S = \frac{B}{a},$$

polmat::minreal computes a minimal realisation, that is, matrices $F, G, H, J$ of state-space equations.

**Call(s)** polmat::minreal(B, a, x)

**Parameters**
B – a matrix of the type Dom::Matrix(polmat::poly)
a – a polynomial of the type DOM_POLY
x – an indeterminate

**Return Value** matrices $F, G, H, J$ of the domain Dom::Matrix

**Examples**

```
>> B:=polmat([[d-d^2, d-d^2],[1-2*d+d^2, -d^2],[d-d^2, d^2]], d)
```

```
                          +-                          -+
                          |       2           2       |
                          |    - d  + d,   - d  + d   |
                          |                           |
                          |   2                  2    |
                          |  d  - 2 d + 1,   - d      |
                          |                           |
                          |       2            2      |
                          |    - d  + d,      d       |
                          +-                          -+
```

```
>> a:=polmat(1-d)
```

```
                             poly(- d + 1, [d])
>> [F, G, H, J] := polmat::minreal(B, a, d)
```

```
  -- +-          -+  +-       -+  +-            -+  +-       -+ --
  |  |  0, 0, 0  |  |  1, 0   |  |   1, 1,  0   |  |  0, 0   |  |
  |  |           |  |         |  |              |  |         |  |
  |  |  0, 0, 0  |, |  0, 1   |, |  -1, 0, -1   |, |  1, 0   |  |
  |  |           |  |         |  |              |  |         |  |
  |  |  0, 1, 1  |  |  0, 0   |  |   1, 0,  1   |  |  0, 0   |  |
  -- +-          -+  +-       -+  +-            -+  +-       -+ --
```

## polmat::norm – norm of polynomial matrix

polmat::norm returns the largest singular value of a constant matrix

$$\begin{pmatrix} P_0 & P_1 & \cdots & P_n \end{pmatrix}$$

for a polynomial matrix

$$P = P_0 + P_1\, var + \cdots + P_n\, var^n.$$

**Call(s)** polmat::norm(P)

**Parameters** P – a polynomial matrix of the type Dom::Matrix(polmat::poly)

**Return Value** an expression

**Examples**

```
>> A:=polmat([[s, s^2+2],[9, 5*s-8]], s)
```

$$\begin{pmatrix} s & s^2+2 \\ 9 & 5s-8 \end{pmatrix}$$

```
>> polmat::norm(A)
```

$$2^{1/2}\,(1745^{1/2}+44)^{1/2}$$

## polmat::rank – a rank of a matrix

polmat::rank computes rank of a matrix.

**Call(s)**
polmat::rank(A, x<, *horn*>)
polmat::rank(A, x<, *sylv*>)

**Parameters**
A – a matrix of the domain Dom::Matrix
x – an indeterminate
*horn*, *sylv* – a computing method

**Return Value**
an integer number

### Details

- If the parameter *horn* is given the rank is computed by replacing an indeterminate by values. This is the default method.
- If the parameter *sylv* is given the rank is computed by the Sylvester matrix algorithm [11].

### Examples

```
>> A:=polmat([[1+z, 2+2, 2*z+2],[4, 5+2*z, 8], [z, 3, 2*z]], z)

                        +-                          -+
                        |  z + 1,    4,    2 z + 2  |
                        |                           |
                        |    4,   2 z + 5,    8     |
                        |                           |
                        |    z,      3,      2 z    |
                        +-                          -+
>> polmat::rank(A, z)

                                    2
>> polmat::rank(A, z, sylv)

                                    2
```

## polmat::reverse – a reciprocal of a polynomial

polmat::reverse returns the reciprocal of a given polynomial.

| | |
|---|---|
| **Call(s)** | polmat::reverse(p, x) |
| **Parameters** | p – a polynomial of the type DOM_POLY<br>x – an indeterminate |
| **Return Value** | a polynomial of the type DOM_POLY |
| **Related Functions** | polmat::conjugate |

### Details

- For a polynomial $a(d) = \alpha_0 + \alpha_1 d + \cdots + \alpha_n d^n$, where $\alpha_i \in \mathbb{C}$, the reciprocal polynomial $\tilde{a}(d)$ is defined as $\tilde{a}(d) = \alpha_n^* + \alpha_{n-1}^* d + \cdots + \alpha_0^* d^n$.

### Examples

```
>> u:=polmat(23*x^3+(18+7*I)*x-15)

                              3
                   poly(23 x  + (18 + 7 I) x - 15, [x])
>> polmat::reverse(u, x)

                                    2        3
                         (18 - 7 I) x  - 15 x  + 23
>> y:=polmat(c*x+b*x^2+a, x)

                                 2
                          poly(b x  + c x + a, [x])
>> polmat::reverse(y, x)

                                _  2   _      _
                                a x  + c x + b
```

```
>> A:=polmat([[0, 0, 1-d],[1, 1-d, 0],[1, -1-d, -1-d]], d)

                          +-                       -+
                          |  0,    0,    - d + 1   |
                          |                        |
                          |  1, - d + 1,    0      |
                          |                        |
                          |  1, - d - 1, - d - 1   |
                          +-                       -+
>> B:=polmat::reverse(A, d)

                          +-                     -+
                          |  0,    0,     d - 1   |
                          |                       |
                          |  d,  d - 1,     0      |
                          |                       |
                          |  d, - d - 1, - d - 1   |
                          +-                     -+
```

## polmat::routh – Routh table

polmat::routh constructs the Routh table for a given polynomial.

**Call(s)** polmat::routh(a, x)

**Parameters** a – a polynomial of the type DOM_POLY
x – an indeterminate

**Return Value** a matrix of the domain Dom::Matrix

**Related Functions** polmat::augRouth, polmat::isstable

**Examples**

```
>> a:=polmat(s^4+s^3+3*s^2+2*s+1)

                          4    3      2
                    poly(s  + s  + 3 s  + 2 s + 1, [s])
>> polmat::routh(a, s)

                                +-          -+
                                |  1, 3, 1  |
                                |           |
                                |  1, 2, 0  |
                                |           |
                                |  1, 1, 0  |
                                |           |
                                |  1, 0, 0  |
                                |           |
                                |  1, 0, 0  |
                                +-          -+
```

## polmat::tcoeff – a trailing coefficient

polmat::tcoeff returns a trailing coefficient of a polynomial or polynomial matrix.

**Call(s)**
polmat::tcoeff(p, x)
polmat::tcoeff(A, x)

**Parameters**
A – a matrix of the domain Dom::Matrix
p – a polynomial of the type DOM_POLY or an expression
x – an indeterminate

**Return Value**
an object of the type or the domain DOM_EXPR, Dom::Matrix, DOM_INT, DOM_RAT, ...

**Related Functions**
polmat::lcoeff, polmat::ldeg, polmat::tdeg

**Examples**

```
>> a:=polmat(x+10*c*x^2, x)
```

```
                                  2
                      poly((10 c) x  + x, [x])
```

```
>> polmat::tcoeff(a, x)
```

```
                                 1
>> A:=polmat([[d, -d+1],[5*d^2, d^2]], d)
```

```
                          +-                -+
                          |    d,  - d + 1   |
                          |                  |
                          |     2       2    |
                          |  5 d ,     d     |
                          +-                -+
```

```
>> polmat::tcoeff(A, d)
```

```
                              +-      -+
                              |  0, 1  |
                              |        |
                              |  0, 0  |
                              +-      -+
```

## polmat::tdeg – a trailing degree

polmat::tdeg returns a trailing degree of a polynomial or polynomial matrix.

**Call(s)**
```
polmat::tdeg(p, x)
polmat::tdeg(A, x)
```

**Parameters**
A – a matrix of the domain Dom::Matrix
p – a polynomial of the type DOM_POLY or an expression
x – an indeterminate

**Return Value**
an element of the type DOM_EXPR or DOM_INT, DOM_RAT, ...

**Related Functions**
polmat::lcoeff, polmat::tcoeff, polmat::ldeg

**Examples**

```
>> a:=polmat(x+10*c*x^2+3-2/x, x)
```

$$x - \frac{2}{x} + 10\,c\,x^2 + 3$$

```
>> polmat::tdeg(a, x)
```

$$-1$$

```
>> A:=polmat([[d, -d+1],[5*d^2+4/3, d^3]], d)
```

$$\begin{pmatrix} d, & -d+1 \\ 5\,d^2+4/3, & d^3 \end{pmatrix}$$

```
>> polmat::tdeg(A, d)
```

$$0$$

## polmat::tf – transfer function

polmat::tf creates transfer functions

$$G(s) = \frac{b(s)}{a(s)}.$$

**Call(s)** polmat::tf(b, a)

**Parameters** a, b – polynomials of the type DOM_POLY

**Return Value** an element of the domain DOM_EXPR

**Examples**

```
>> a:=polmat(8*s+s^2, s)
```

$$\text{poly}(s^2 + 8\,s, [s])$$

```
>> b:=polmat(s^3+6*s^2-4*s+5)
```

$$\text{poly}(s^3 + 6\,s^2 - 4\,s + 5, [s])$$

```
>> polmat::tf(b, a)
```

$$\frac{6\,s^2 - 4\,s + s^3 + 5}{8\,s + s^2}$$

## polmat::tsp – two-side polynomial

polmat::tsp creates the two-side polynomial or polynomial matrix.

**Call(s)**
```
polmat::tsp(p, x)
polmat::tsp(A, x)
```

**Parameters**
p – a polynomial
A – a matrix of the domain Dom::Matrix(polmat::poly)
x – an indeterminate

**Return Value**
an expression or matrix of the type Dom::Matrix()

**Examples**

```
>> A:=polmat([[1, d],[d^2-d+4, 0]], d)
```

```
                              +-                 -+
                              |        1,      d  |
                              |                   |
                              |   2               |
                              |  d  - d + 4, 0    |
                              +-                 -+
```

```
>> polmat::tsp(A, d)
```

```
                        +-                                -+
                        |                              1   |
                        |            1,           d + -   |
                        |                             d   |
                        |                                 |
                        |   1   1         2               |
                        |  -- - - - d + d  + 4,    0      |
                        |   2   d                         |
                        |  d                              |
                        +-                                -+
```

## polmat::transpose – transposition of a matrix

polmat::transpose returns the transposition of a matrix.

**Call(s)** polmat::transpose(A)

**Parameters** A – a matrix of the domain Dom::Matrix()

**Return Value** a matrix of the domain Dom::Matrix()

**Related Functions** polmat::conjugatetranspose

**Examples**

```
>> A:=polmat([[(1+I)*d, 8],[12-5*I, 2*d]], d)
```

```
                          +-                  -+
                          |  (1 + I) d,  8    |
                          |                   |
                          |   12 - 5 I, 2 d   |
                          +-                  -+
>> B:=polmat::transpose(A, d)
```

```
                       +-                       -+
                       |  (1 - I) d, 12 + 5 I    |
                       |                         |
                       |      8,         2 d     |
                       +-                       -+
```

## polmat::value – value of an object

polmat::value replaces an indeterminate in an object by a value.

**Call(s)**
```
polmat::value(p, x, v)
polmat::value(m, x, v)
```

**Parameters**
m – a matrix of the domain Dom::Matrix
p – a polynomial of the type DOM_POLY or an expression
x – an indeterminate
v – a value

**Return Value** an element of the type DOM_EXPR or DOM_INT, DOM_RAT, ...

### Details

- For the expression in number is used *Horner scheme*.

### Examples

```
>> a:=polmat(4*x+x^2+20, x)
```

$$\mathrm{poly}(x^2 + 4\,x + 20, [x])$$

```
>> polmat::value(a, x, 5)
```

$$65$$

```
>> a:=polmat(m*x+n*x^2+l, x)
```

$$\mathrm{poly}(n\,x^2 + m\,x + l, [x])$$

```
>> polmat::value(a, x, 0)
```

$$l$$

```
>> A:=polmat([[s+1, s-1],[s^2+2/5, s]], s)
```

$$\begin{pmatrix} s + 1, & s - 1 \\ s^2 + 2/5, & s \end{pmatrix}$$

```
>> polmat::value(A, s, 1+I)
```

$$\begin{bmatrix} 2 + I, & I \\ 2/5 + 2\,I, & 1 + I \end{bmatrix}$$

## polmat::var – symbols contained in an expression

polmat::var returns all symbols contained in an object.

**Call(s)**
polmat::var(p)
polmat::var(A)

**Parameters**
A – a matrix of the domain Dom::Matrix
p – a polynomial of the type DOM_POLY or an expression

**Return Value**
an element of the type DOM_EXPR or DOM_INT, DOM_RAT, ...

**Related Functions**
polmat::lcoeff, polmat::tcoeff, polmat::ldeg

**Examples**

```
>> a:=polmat(m*x+n*x^2+l, x)

                                  2
                      poly(n x  + m x + l, [x])
>> polmat::var(a)

                             n, x, m, l
```

## polmat::augRouth – augmented Routh table

polmat::augRouth constructs the augmented Routh table of

$$G(s) = \frac{b(s)}{a(s)} = \frac{b_1\, s^{n-1} + \cdots + b_n}{a_0\, s^n + a_1\, s^{n-1} + \cdots + a_n}, \quad a_0 > 0.$$

**Call(s)** polmat::augRouth(b, a, x)

**Parameters** a, b – polynomials of the type DOM_POLY
x – an indeterminate

**Return Value** matrices of the domain Dom::Matrix

**Related Functions** polmat::isstable, polmat::routh

**Examples**

```
>> a:=polmat(s^4+s^3+3*s^2+2*s+1)

                           4    3      2
                    poly(s  + s  + 3 s  + 2 s + 1, [s])
>> b:=polmat(s^3+2*s^2+5*s+6)

                          3      2
                     poly(s  + 2 s  + 5 s + 6, [s])
>> [R, Q1, Q2]:=polmat::augRouth(b, a, s)

                 -- +-          -+  +-       -+  +-       -+ --
                 |  |  1, 3, 1  |  |  1, 5  |  |  0, 0  |  |
                 |  |           |  |        |  |        |  |
                 |  |  1, 2, 0  |  |  1, 2  |  |  2, 6  |  |
                 |  |           |  |        |  |        |  |
                 |  |  1, 1, 0  |, |  3, 0  |, |  1, 1  |  |
                 |  |           |  |        |  |        |  |
                 |  |  1, 0, 0  |  |  1, 0  |  |  4, 0  |  |
                 |  |           |  |        |  |        |  |
                 |  |  1, 0, 0  |  |  0, 0  |  |  1, 0  |  |
                 -- +-          -+  +-       -+  +-       -+ --
```

## polmat::h2norm – $\mathcal{H}_2$ norm computation

polmat::h2norm computes $\mathcal{H}_2$ norm of a given strictly stable system

$$G(s) = \frac{b(s)}{a(s)} = \frac{b_1 s^{n-1} + \cdots + b_n}{a_0 s^n + a_1 s^{n-1} + \cdots + a_n}, \qquad a_0 > 0.$$

| | |
|---|---|
| **Call(s)** | polmat::h2norm(b, a, x) |
| **Parameters** | a, b – a polynomial of the type DOM_POLY<br>x – an indeterminate |
| **Return Value** | a number or an expression |
| **Related Functions** | polmat::augRouth, polmat::nehari, polmat::Routh, polmat::schmidtPair |

### Details

- The algorithm is based on computation of the augmented Routh table of $G(s)$. This method is described in [12].

### Examples

```
>> b:=polmat(s^3+2*s^2+5*s+6)

                              3      2
                       poly(s  + 2 s  + 5 s + 6, [s])
>> a:=polmat(s^4+s^3+3*s^2+2*s+1)

                            4    3      2
                     poly(s  + s  + 3 s  + 2 s + 1, [s])
>> polmat::h2norm(b, a, s)

                                    1/2
                                   15
```

## polmat::Impulse – impulse response of LTI models

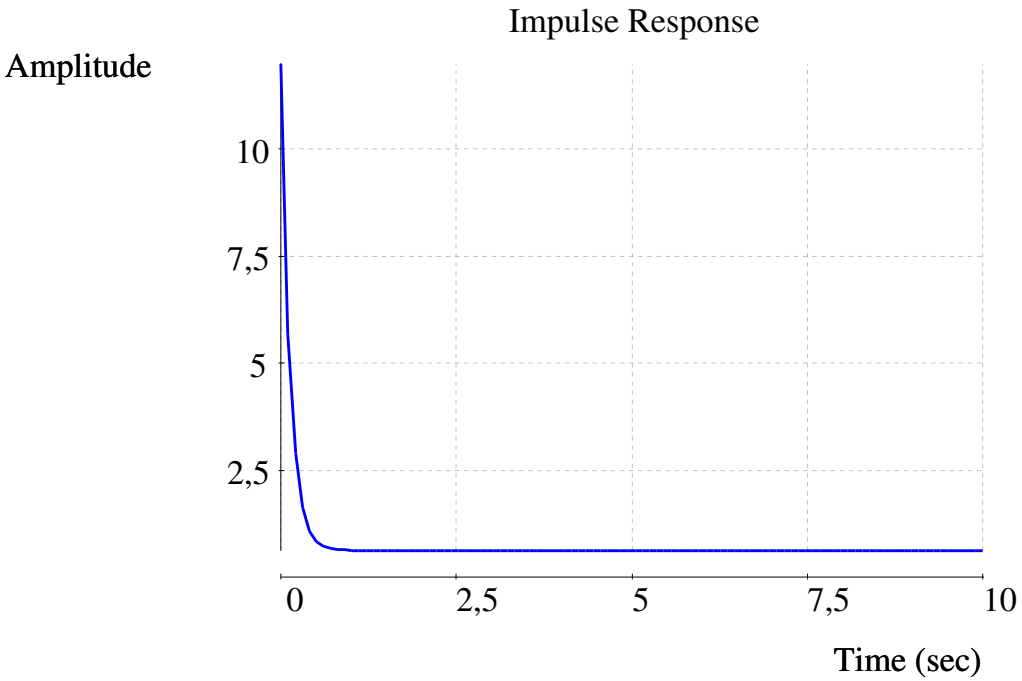
polmat::Impulse plots the impulse response of an LTI model.

**Call(s)** polmat::Impulse(G, tfinal)

**Parameters** G – an expression of the type DOM_EXPR
tfinal – a real number

**Examples**

```
>> a:=polmat(8*s+s^2, s):
>> b:=polmat(s^3+6*s^2-4*s+5):
>> G:=polmat::tf(b, a):
>> polmat::Impulse(G, 10)
```



## polmat::iscontrollable – controllability test

polmat::iscontrollable tests whether a system defined by

$$x(k+1) = F\,x(k) + G\,u(k)$$

or

$$\dot{x}(t) = F\,x(t) + G\,u(t)$$

is controllable.

| | |
|---|---|
| **Call(s)** | polmat::iscontrollable(A, B) |
| **Parameters** | A, B – matrices of the domain Dom::Matrix |
| **Return Value** | a value of the type DOM_BOOL |
| **Related Functions** | polmat::isobservable |

### Details

- The algorithm generates the controllability matrix by

$$C = \begin{pmatrix} B & AB & A^2B & \cdots & A^{n-1}B \end{pmatrix}$$

  and computes its rank.
- In the continuous-time case the controllability and the reachability is the same. In the discrete-time case, this function performs the reachability test.

### Examples

```
>> F:=matrix([[2, 0],[1, 2]])
```

```
+-      -+
|  2, 0  |
|        |
|  1, 2  |
+-      -+
```

```
>> G:=matrix([1, 0])
```

```
+-   -+
|  1   |
|      |
|  0   |
+-   -+
```

```
>> [isc, C]:=polmat::iscontrollable(F, G)
```

$$\left[\begin{array}{c} \text{TRUE} \end{array},\ \begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix}\right]$$

## polmat::isobservable – observability test

polmat::isobservable tests whether a system defined by

$$\begin{aligned} x(k+1) &= F\,x(k) + G\,u(k) \\ y(k) &= H\,x(k) \end{aligned}$$

or

$$\begin{aligned} \dot{x}(t) &= F\,x(t) + G\,u(t) \\ y(t) &= H\,x(t) \end{aligned}$$

is observable.

| | |
|---|---|
| **Call(s)** | polmat::isobservable(A, B) |
| **Parameters** | A, B – a matrix of the domain Dom::Matrix |
| **Return Value** | a value of the type DOM_BOOL |
| **Related Functions** | polmat::iscontrollable |

### Details

- The algorithm generates the observability matrix by

$$O = \begin{pmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{n-1} \end{pmatrix}$$

  and computes its rank.

### Examples

```
>> F:=matrix([[2, 0],[1, 2]])
```

```
                                  +-      -+
                                  |  2, 0  |
                                  |        |
                                  |  1, 2  |
                                  +-      -+
```

```
>> H:=matrix([[1, 0.001]])
```

```
                                 +-        -+
                                 | 1, 0.001 |
                                 +-        -+
```

```
>> [isd, Ob]:=polmat::isobservable(F, H)
```

```
--        +-                  -+ --
|         |    1,    0.001    |  |
|  TRUE,  |                    |  |
|         |  2.001, 0.002     |  |
--        +-                  -+ --
```

## polmat::isproper – test of properness of a polynomial fraction

polmat::isproper tests whether a polynomial fraction

$$G(s) = \frac{b(s)}{a(s)}$$

is proper.

| | |
|---|---|
| **Call(s)** | polmat::isproper(G, var) |
| **Parameters** | G – an expression<br>var – an indetermite |
| **Return Value** | a value of the type DOM_BOOL |
| **Related Functions** | polmat::properness |

### Details

- The transfer function is proper if $\partial b \leq \partial a$.

### Examples

```
>> G:=(s+a)/(b*s^2+c*s-d)
                                  a + s
                            --------------
                                        2
                            c s - d + b s
>> polmat::isproper(G, s)
                                  TRUE
```

## polmat::isstable – test of stability of a polynomial

polmat::isstable tests whether a polynomial is stable.

**Call(s)** polmat::isstable(p)

**Parameters** p – a polynomial of the type DOM_POLY

**Return Value** a value of the type DOM_BOOL

**Examples**

```
>> a:=polmat(s^4+s^3+3*s^2+2*s+1)

                          4    3      2
                    poly(s  + s  + 3 s  + 2 s + 1, [s])
>> polmat::isstable(a)

                                  TRUE
>> b:=polmat(2-2*d^2+d^3-d^4)

                           4    3      2
                    poly(- d  + d  - 2 d  + 2, [d])
>> polmat::isstable(b)

                                  FALSE
>> c:=polmat(2-2*z^2+z^3-z^4)

                           4    3      2
                    poly(- z  + z  - 2 z  + 2, [z])
>> polmat::isstable(c)

                                  FALSE
```

## polmat::nehari – Nehari problem

polmat::nehari for

$$G(s) = \frac{b(s)}{a(s)}$$

finds $Q(s) \in \mathcal{H}_\infty$ that minimises

$$\|G(-s) - Q(s)\|_\infty.$$

| | |
|---|---|
| **Call(s)** | polmat::nehari(b, a, x) |
| **Parameters** | a, b – a polynomial of the type DOM_POLY<br>x – an indeterminate |
| **Return Value** | an expression |
| **Related Functions** | polmat::augRouth, polmat::Routh,<br>polmat::h2norm, polmat::schmidtPair |

### Details

- The algorithm is based on computation of the augmented Routh table of $G(s)$. This method is described in [12].

### Examples

```
>> a:=polmat(s^2+sqrt(2)*s+1)

                          2    1/2
                  poly(s  + 2    s + 1, [s])
>> b:=polmat(2*sqrt(2)*s+4)

                              1/2
                   poly((2 2   ) s + 4, [s])
>> polmat::nehari(b,a,s)

                        1/2       1/2
                   s - 2    + s 2    + 3
                   --------------------
                           s + 1
```

## polmat::qnorm – the quadratic norm of stable series

polmat::qnorm computes the quadratic norm of stable series $e = \frac{b}{a}$.

**Call(s)** polmat::qnorm(a, b)

**Parameters** a, b – polynomials of the type DOM_POLY

**Return Value** a value of the type DOM_INT, DOM_RAT, ...

**Examples**

```
>> a:=polmat(2-d)
                                poly(- d + 2, [d])
>> b:=polmat(2+2*d)
                                poly(2 d + 2, [d])
>> polmat::qnorm(a, b)
                                          4
```

## polmat::plotrootcurve – plot of root curves

polmat::plotrootcurve plots root curves of a three-variable polynomial.

**Call(s)** polmat::plotrootcurve(a, x<, "d">)

**Parameters**
a – a polynomial
x – an indeterminate

**Details**

- If the parameter "d" is given then root curves of the discrete-time polynomial are plotted else root curves of the continue-time polynomial are plotted.

**Examples**

```
>> p:=4+z[1]+z[2]+z[3]^2:
>> polmat::plotRootCurve(p,z,"d")
```

## polmat::properness – test of the properness of polynomial fraction

polmat::properness returns a properness of transfer function

$$G(s) = \frac{b(s)}{a(s)}.$$

| | |
|---|---|
| **Call(s)** | polmat::properness(G, var) |
| **Parameters** | G – an expression<br>var – an indeterminate |
| **Return Value** | a value of the type DOM_STRING |
| **Related Functions** | polmat::isproper |

### Details

- The procedure returns "strictly proper" if $\partial b < \partial a$.
- The procedure returns "biproper" if $\partial b = \partial a$.
- The procedure returns "nonproper" if $\partial b > \partial a$.

### Examples

```
>> G:=(s+a)/(b*s^2+c*s-d)
```

$$\frac{a + s}{c\,s - d + b\,s^2}$$

```
>> polmat::properness(G, s)
```

```
"strictly proper"
```

## polmat::pzmap – map of poles and zeros

polmat::pzmap plots poles and zeros in the complex plane for a LTI system given by

$$G(x) = \frac{b(x)}{a(x)}.$$

**Call(s)** polmat::pzmap(b<, a>)

**Parameters** a, b – objects of DOM_POLY or DOM_EXPR

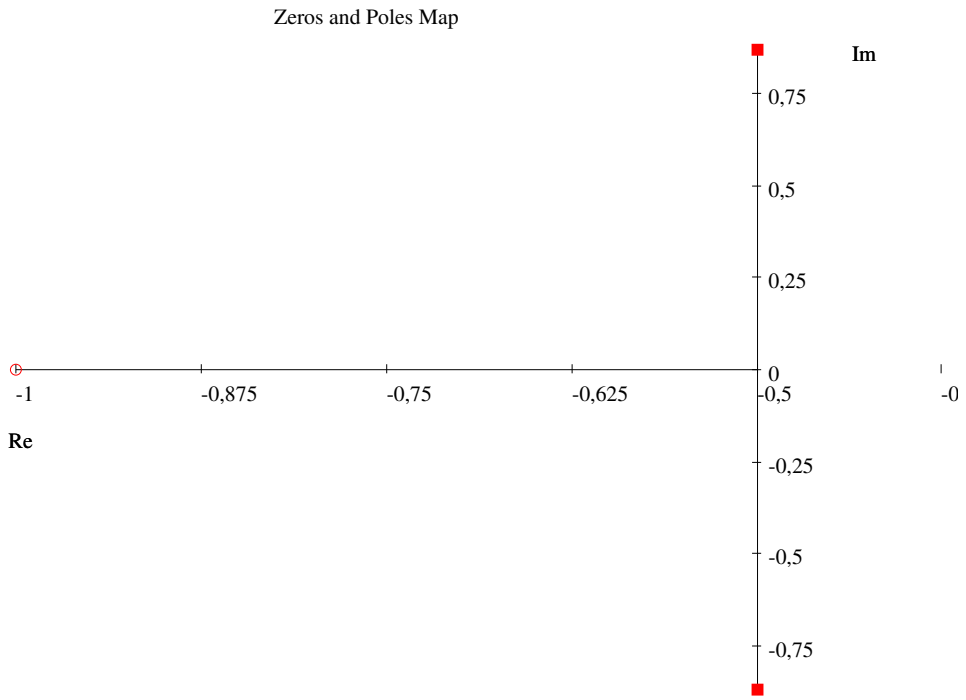
**Examples**

```
>> polmat::pzmap(s^2+s+1)
```

```
>> polmat::pzmap((s+1)/(s^2+s+1))
```

```
>> polmat::pzmap(s+1, s^2+s+1)
```

```
>> polmat::pzmap(poly(s+1), poly(s^2+s+1))
```



## polmat::schmidtPair – Schmidt pairs

polmat::schmidtpair computes Schmidt pairs for a given transfer function

$$G(s) = \frac{b(s)}{a(s)}.$$

| | |
|---|---|
| **Call(s)** | polmat::schmidtPair(b, a, x) |
| **Parameters** | a, b – a polynomial of the type DOM_POLY<br>x – an indeterminate |
| **Return Value** | a table of arrays of expressions |
| **Related Functions** | polmat::augRouth, polmat::h2norm, polmat::nehari, polmat::Routh |

**Details**

- The algorithm is based on computation of the augmented Routh table of $G(s)$. This method is described in [12].

**Examples**

```
>> a:=polmat(s^2+sqrt(2)*s+1)

                          2    1/2
                   poly(s  + 2    s + 1, [s])
>> b:=polmat(2*sqrt(2)*s+4)

                             1/2
                    poly((2 2   ) s + 4, [s])
>> polmat::schmidtPair(b,a,s)

table(
  2 = [array(1..1, 1..1, (1,1) = 1/2*2^(5/4)/(s^2 + s*2^(1/2) + \
1) - 1/2*s*2^(5/4)/(s^2 + s*2^(1/2) + 1)), array(1..1, 1..1, (1,\
1) = 2^(3/4)*(2^(1/2) - 2)/(2*2^(1/2) - 2)/(s^2 + s*2^(1/2) + 1)\
 + s*2^(3/4)*(- 2^(1/2) + 2)/(2*2^(1/2) - 2)/(s^2 + s*2^(1/2) + \
1))],
  1 = [array(1..1, 1..1, (1,1) = 1/2*2^(5/4)/(s^2 + s*2^(1/2) + \
1) + 1/2*s*2^(5/4)/(s^2 + s*2^(1/2) + 1)), array(1..1, 1..1, (1,\
1) = 2^(3/4)*(2^(1/2) + 2)/(2*2^(1/2) + 2)/(s^2 + s*2^(1/2) + 1)\
 + s*2^(3/4)*(2^(1/2) + 2)/(2*2^(1/2) + 2)/(s^2 + s*2^(1/2) + 1)\
)]
)
```

## `polmat::Step` – step response of LTI models

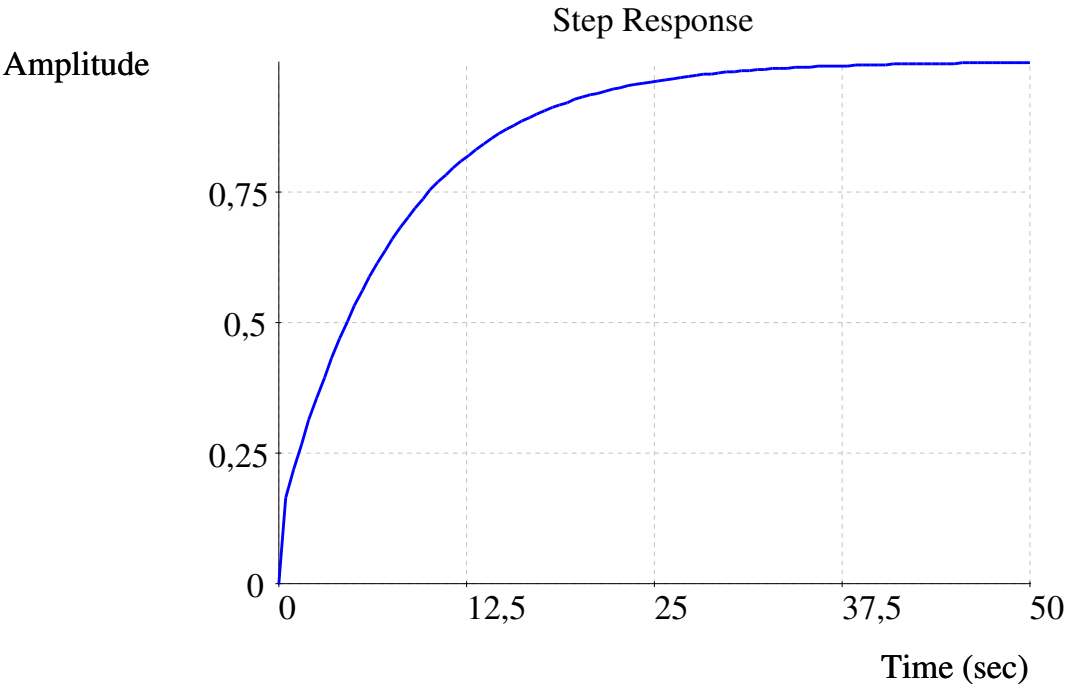
`polmat::Step` plots the step response of a given LTI model.

**Call(s)** `polmat::Step(G, tfinal)`

**Parameters** `G` – an expression of the type `DOM_EXPR`
`tfinal` – a real number

**Examples**

```
>> a:=polmat(1+8*s+s^2, s):
>> b:=polmat(s+1):
>> G:=polmat::tf(b, a):
>> polmat::Step(G, 50)
```



## polmat::charitonov – Kharitonov's polynomials

polmat::charitonov returns Kharitonov's polynomials for an interval polynomial

$$a(s) = \left[a_n^{\min}, a_n^{\max}\right] s^n + \left[a_{n-1}^{\min}, a_{n-1}^{\max}\right] s^{n-1} + \cdots + \left[a_1^{\min}, a_1^{\max}\right] s + \left[a_0^{\min}, a_0^{\max}\right]$$

and performs robust stability test.

**Call(s)** polmat::charitonov(amin, amax<, *all*>)
polmat::charitonov(Q<, *all*>)

**Parameters** amin, amax – objects of the type DOM_POLY or DOM_EXPR
Q – a list of lists of two real numbers (for example [[anmin, anmax],...,[a1min, a1max], [a0min, a0max]])

**Return Value** Boolean value or, if a parameter *all* is given, list of boolean value and list of four polynomials.

**Related Functions** polmat::chplot

### Details

- Kharitonov's polynomials for an interval polynomial $a(s)$ are defined

$$\begin{aligned}
K_1(s) &= a_0^{\min} + a_1^{\min} s + a_2^{\max} s^2 + a_3^{\max} + a_4^{\min} s^4 + a_5^{\min} s^5 + a_6^{\max} s^6 + \cdots \\
K_2(s) &= a_0^{\max} + a_1^{\max} s + a_2^{\min} s^2 + a_3^{\min} + a_4^{\max} s^4 + a_5^{\max} s^5 + a_6^{\min} s^6 + \cdots \\
K_3(s) &= a_0^{\max} + a_1^{\min} s + a_2^{\min} s^2 + a_3^{\max} + a_4^{\max} s^4 + a_5^{\min} s^5 + a_6^{\min} s^6 + \cdots \\
K_4(s) &= a_0^{\min} + a_1^{\max} s + a_2^{\max} s^2 + a_3^{\min} + a_4^{\min} s^4 + a_5^{\max} s^5 + a_6^{\max} s^6 + \cdots
\end{aligned}$$

- An interval polynomial is robust stable if all Kharitonov's polynomials are stable.

### Examples

```
>> pmi:=polmat(11+9*s+7*s^2+5*s^3+3*s^4+1*s^5)

                              5      4      3      2
                       poly(s  + 3 s  + 5 s  + 7 s  + 9 s + 11, [s])
>> pma:=polmat(12+10*s+8*s^2+6*s^3+4*s^4+2*s^5)

                              5      4      3      2
                     poly(2 s  + 4 s  + 6 s  + 8 s  + 10 s + 12, [s])
>> polmat::charitonov(pmi, pma)

                                          FALSE
>> polmat::charitonov(pmi, pma, all)

               5      4      3      2
```

```
[FALSE, [poly(s  + 3 s  + 6 s  + 8 s  + 9 s + 11, [s]),

           5      4      3      2
   poly(2 s  + 4 s  + 5 s  + 7 s  + 10 s + 12, [s]),

           5      4      3      2
   poly(2 s  + 3 s  + 5 s  + 8 s  + 10 s + 11, [s]),

         5      4      3      2
   poly(s  + 4 s  + 6 s  + 7 s  + 9 s + 12, [s])]]
>> polmat::charitonov([[1,2],[3,4],[5,6],[7,8],[9,10],[11,12]], all)

               5      4      3      2
[FALSE, [poly(s  + 3 s  + 6 s  + 8 s  + 9 s + 11, [s]),

           5      4      3      2
   poly(2 s  + 4 s  + 5 s  + 7 s  + 10 s + 12, [s]),

           5      4      3      2
   poly(2 s  + 3 s  + 5 s  + 8 s  + 10 s + 11, [s]),

         5      4      3      2
   poly(s  + 4 s  + 6 s  + 7 s  + 9 s + 12, [s])]]
```

## polmat::chplot – plots Kharitonov's rectangles

polmat::chplot plots Kharitonov's rectangles for an interval polynomial

$$a(s) = \left[a_n^{\min}, a_n^{\max}\right] s^n + \left[a_{n-1}^{\min}, a_{n-1}^{\max}\right] s^{n-1} + \cdots + \left[a_1^{\min}, a_1^{\max}\right] s + \left[a_0^{\min}, a_0^{\max}\right].$$

**Call(s)** polmat::chplot(amin, amax, omega)
polmat::chplot(Q, omega)

**Parameters** amin, amax – objects of the type DOM_POLY or DOM_EXPR
Q – a list of lists of two real numbers (for example [[anmin, anmax],...,[a1min, a1max], [a0min, a0max]])
omega – values of complex frequency which are evaluated

**Related Functions** polmat::charitonov

### Details

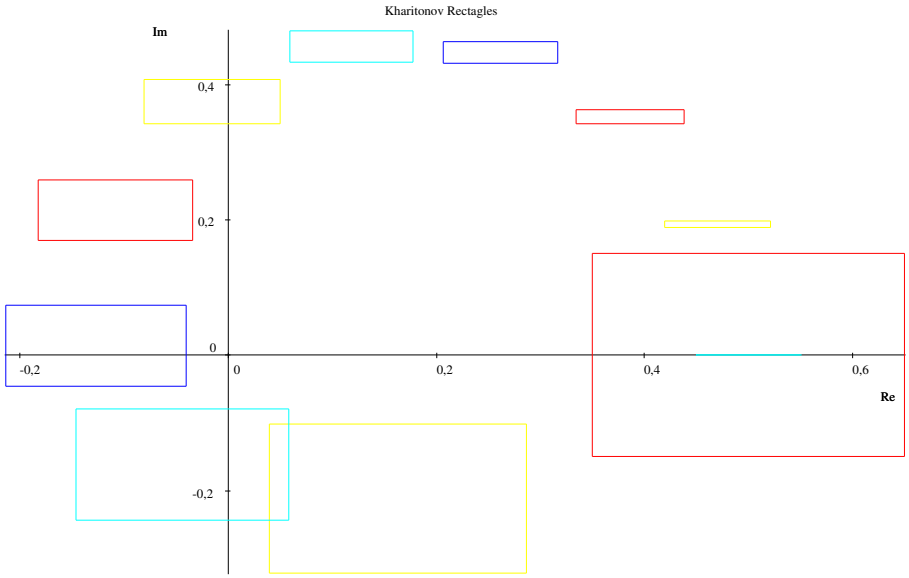
- With use of polmat::charitonov, the function plots Kharitonov polynomials evaluated in values omega.

### Examples

```
>> amin:=polmat(s^6 + 3.95*s^5 + 3.95*s^4 + 5.95*s^3 + 2.95*s^2 +
1.95*s + 0.45)

       6         5         4         3         2
 poly(s  + 3.95 s  + 3.95 s  + 5.95 s  + 2.95 s  + 1.95 s + 0.45, [s])
>> amax:=polmat(s^6 + 4.05*s^5 + 4.05*s^4 + 6.05*s^3 + 3.05*s^2 +
2.05*s + 0.55)

       6         5         4         3         2
 poly(s  + 4.05 s  + 4.05 s  + 6.05 s  + 3.05 s  + 2.05 s + 0.55, [s])
>> polmat::chplot(amin, amax, x/10 $ x=0..10)
```


Kharitonov Rectagles
Im
0,4
0,2
0
-0,2
-0,2
0
0,2
0,4
0,6
Re

## polmat::ptopex – extreme polynomials

polmat::ptopex creates $2^n$ extreme polynomials for the polytope of polynomials $A = A_0 + Q_1 A_1 + Q_2 A_2 + \cdots + Q_n A_n$ defined by polynomials $A_0, A_1, A_2, \ldots A_n$ and bounds $Q_1, Q_2, \ldots, Q_n$.

**Call(s)** polmat::ptopex(A0, <A, Q>)

**Parameters**
A0 – an object of the type DOM_POLY or DOM_EXPR or Dom::Matrix
A – a list of objects of the type DOM_POLY or DOM_EXPR or Dom::Matrix (for example [A1, A2,..., An])
Q – a list of lists of two real numbers (for example [[Q1min, Q1max], [Q2min, Q2max],...,[Qnmin, Qnmax]])

**Return Value** an element of the type DOM_TABLE, which contains the polynomials or the expressions or the elements of the domain Dom::Matrix

**Related Functions** polmat::ptopplot

**Examples**

```
>> f:=3 + 2*s + s^2
```

$$2\,s + s^2 + 3$$

```
>> g:= -1 - 1*s
```

$$-s - 1$$

```
>> polmat::ptopex(f, [g], [[0,4]])
```

```
table(
                2
  2 = s  - 2 s - 1,
                  2
  1 = 2 s + s  + 3
)
```

```
>> p0:=polmat(s^4*(78*s+13*s^2+s^3+234))
```

$$\mathrm{poly}(s^7 + 13\,s^6 + 78\,s^5 + 234\,s^4, [s])$$

```
>> p1:=polmat(s*(496*s+240*s^2+207*s^3+205))
```

$$\mathrm{poly}(207\,s^4 + 240\,s^3 + 496\,s^2 + 205\,s, [s])$$

```
>> p2:=polmat(s^2+1)
```

$$2$$

```
                              poly(s  + 1, [s])
>> polmat::ptopex(p0, [p1, p2], [[0,1], [0.05, 5]])

table(
                    2         3         4        5        6    7
  4 = 205 s + 501 s  + 240 s  + 441 s  + 78 s  + 13 s  + s  + 5,
          2         4        5        6    7
  3 = 5 s  + 234 s  + 78 s  + 13 s  + s  + 5,
                       2         3         4        5        6    7
  2 = 205 s + 496.05 s  + 240 s  + 441 s  + 78 s  + 13 s  + s  + 0.05,
            2         4        5        6    7
  1 = 0.05 s  + 234 s  + 78 s  + 13 s  + s  + 0.05
)
```

## polmat::ptopplot – plots polynomial values sets

polmat::ptopplot plots polynomial values sets of the polytope of polynomials $A = A_0 + Q_1 A_1 + Q_2 A_2 + \cdots + Q_n A_n$ defined by polynomials $A_0, A_1, A_1, \ldots A_n$ and bounds $Q_1, Q_2, \ldots, Q_n$.

**Call(s)** polmat::ptopplot(A0, A, Q, omega)

**Parameters**

A0 – an object of the type DOM_POLY or DOM_EXPR or Dom::Matrix

A – a list of objects of the type DOM_POLY or DOM_EXPR or Dom::Matrix (for example [A1, A2,..., An])

Q – a list of lists of two real numbers (for example [[Q1min, Q1max], [Q2min, Q2max],...,[Qnmin, Qnmax]])

omega – the values of complex frequency what are evaluated

**Related Functions** polmat::ptopex

### Examples

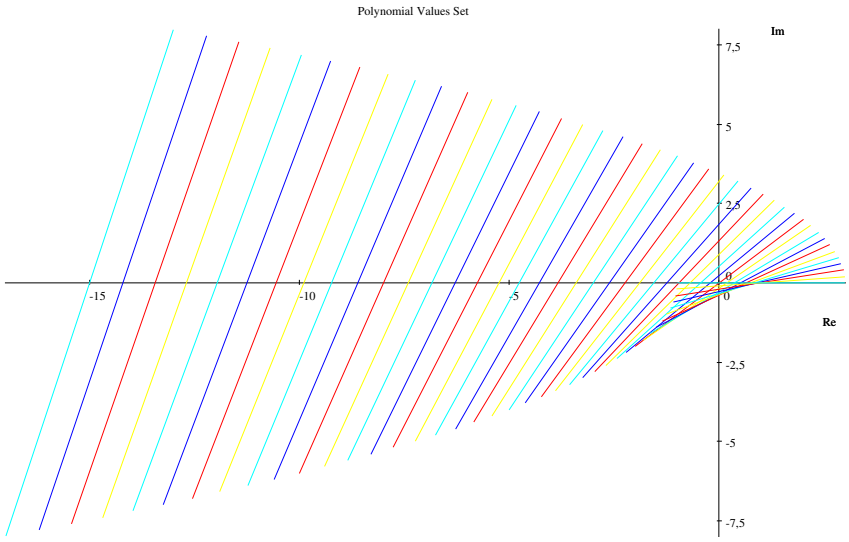
```
>> f:=3 + 2*s + s^2
```

$$2\,s + s^2 + 3$$

```
>> g:= -1 - 1*s
```

$$-s - 1$$

```
>> polmat::ptopplot(f, [g], [[0,4]], .1*x*I $ x=0..40)
```



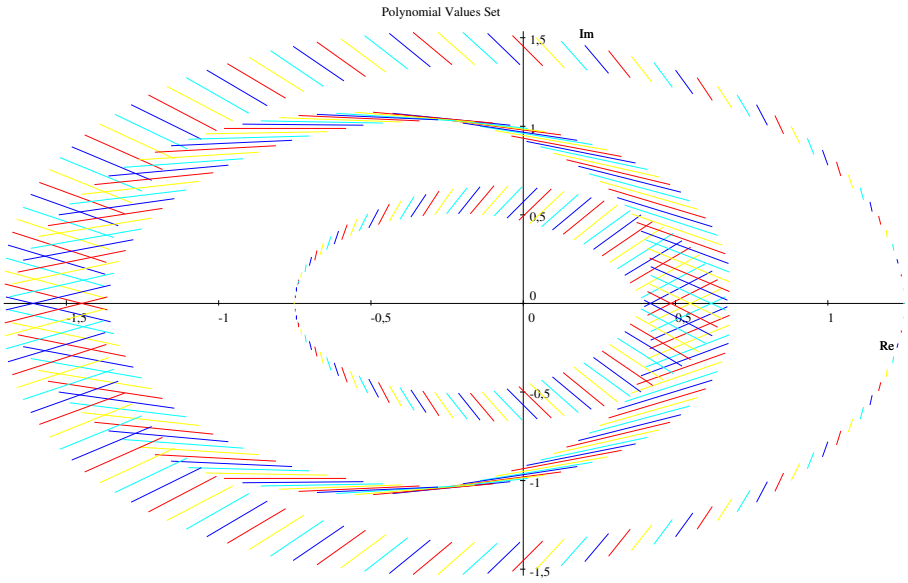
```
>> p0:=-1/4+1/2*z^2+z^3
```

$$\frac{z^2}{2} + z^3 - \frac{1}{4}$$

```
>> p1:=z^2-1
```

$$z^2 - 1$$

```
>> polmat::ptopplot(p0, [p1], [[-.1, .1]], exp(-I*x/50.0) $ x=0..160)
```



## polmat::stabint – stability interval of a polynomial matrix

polmat::stabint returns a stability interval $(q_{\min}, q_{\max})$ of a polynomial matrix

$$A = A_0 + q A_1 + q^2 A_2 + \cdots + q^n A_n.$$

**Call(s)** polmat::stabint(A0, <A1, A2,...,An>)

**Parameters** A0, A1, A2,...,An – objects of the type DOM_POLY or DOM_EXPR or Dom::Matrix

**Return Value** list of two real numbers

**Examples**

```
> p0:=s^4+6*s^3+12*s^2+10*s+3
```

$$10\,s + 12\,s^2 + 6\,s^3 + s^4 + 3$$

```
>> p1:=s^3+s
```

$$s + s^3$$

```
>> polmat::stabint(p0, p1, s)
```

$$\left[-\frac{128}{33^{1/2}+17}, \infty\right]$$

## polmat::list2poly – generates polynomial from a list

polmat::list2poly generates an element of the domain DOM_POLY for a given element of DOM_LIST.

**Call(s)** polmat::list2poly(l)

**Parameters** l – an element of the type DOM_LIST

**Return Value** a polynomial of the type DOM_POLY

**Related Functions** polmat::poly2list

**Examples**

```
>> l:=[1, 0, 4]

                                [1, 0, 4]
>> polmat::list2poly(l, s)

                                   2
                            poly(s  + 4, [s])
>> a:=polmat::list2poly([1, 2, 0, 5], d)

                              3      2
                      poly(d  + 2 d  + 5, [d])
```

## polmat::polmat2poltbx – generates a Polynomial Toolbox matrix

polmat::polmat2poltbx returns a Matlab code for definition polynomial matrix in Polynomial Toolbox for Matlab.

**Call(s)** polmat::polmat2poltbx(p)
polmat::polmat2poltbx(M)

**Parameters** p – a polynomial
M – a matrix of the domain Dom::Matrix

**Return Value** an object of the type DOM_NULL

**Related Functions** polmat::list2poly, polmat::poly2list

**Examples**

```
>> a:=polmat(s^4+3*s^2-8*s+1)

                          4      2
                 poly(s  + 3 s  - 8 s + 1, [s])
>> polmat::polmat2poltbx(a)

                       3*s^2 - 8*s + s^4 + 1

>> B:=polmat::rand(1,2,2,s)

                +-                                  -+
                |  - 1436 s + 567,  260 s - 2997     |
                |                                    |
                |  - 2235 s + 349, - 1312 s - 593    |
                +-                                  -+
>> polmat::polmat2poltbx(B)

[- 1436*s + 567, 260*s - 2997; - 2235*s + 349, - 1312*s - 593]
```

## polmat::poly2list – generates list from a polynomial

polmat::poly2list generates an element of the domain DOM_LIST for an element of the domain DOM_POLY.

**Call(s)** polmat::poly2list(p)

**Parameters** p – a polynomial of the type DOM_POLY

**Return Value** an element of the type DOM_LIST

**Related Functions** polmat::list2poly

**Examples**

```
>> p:=polmat(s^2+4)
```

$$\mathrm{poly}(s^2 + 4, [s])$$

```
>> polmat::poly2list(p)
```

$$[1, 0, 4]$$

## polmat::rand – random matrix

polmat::rand returns a random polynomial matrix.

**Call(s)** polmat::rand(d, m, n, x)

**Parameters**
d – degree of the polynomial matrix
m – number of rows
n – number of columns
x – an indeterminate

**Return Value** a matrix of the type Dom::Matrix(polmat::poly)

**Examples**

```
>> A:=polmat::rand(1, 2, 3, s)
```

```
            +-                                                  -+
            |  - 2199 s + 160, - 777 s + 354, - 431 s + 850     |
            |                                                    |
            |   1116 s + 351,  - 52 s + 1779, - 938 s + 219     |
            +-                                                  -+
```

## List of symbols

| | |
|---|---|
| $a$ | polynomial, number |
| $\partial a$ | degree of a polynomial |
| $a^*$ | complex conjugate of the polynomial |
| $\tilde{a}$ | reciprocal of the polynomial |
| | |
| $A$ | matrix, polynomial matrix |
| $I$ | identity matrix |
| $A_{l,m}$ | matrix of the dimension $l \times m$ |
| $A^{-1}$ | inverse of the matrix |
| $\mathrm{adj}\, A$ | adjungate matrix |
| $\det A$ | determinant of the matrix |
| $\mathrm{diag}_{l,m}[.]$ | diagonal matrix |
| | |
| $(.,.)$ | greatest common divisor |

# References

[1] V. Kučera. *Discrete linear control*. John Wiley and Sons, 1979.

[2] J. Ježek. New algorithm for minimal solution of linear polynomial equations. *Kybernetika*, 18(6):505–516, 1982.

[3] J. Ježek. Conjugate and symmetric polynomial equations – II: Discrete-time systems. *Kybernetika*, 19(3):196–211, 1983.

[4] Z. Vostrý. New algorithm for polynomial spectral factorization with quadratic convergence I. *Kybernetika*, 11(6):415–422, 1975.

[5] Z. Vostrý. New algorithm for polynomial spectral factorization with quadratic convergence II. *Kybernetika*, 12(4):248–259, 1976.

[6] H. Kwakernaak and M. Šebek. Polynomial J-spectral factorization. *IEEE Transactions on Automatic Control*, 39(2):315–328, 1994.

[7] Petr Augusta. An algorithm to solve algebraic riccati equations with polynomials. In *Proceedings of the 17th International Conference on Methods and Models in Automation and Robotics*, pages 409–414, 2012.

[8] Petr Augusta and Petra Augustová. Algorithm for solving polynomial algebraic riccati equations and its application. *Cybernetics and Physics*, 1(4):237–242, 2012.

[9] T. Kailath. *Linear systems*. New York : Prentice Hall, 1980.

[10] M. Hromčík and M. Šebek. New algorithm for polynomial matrix determinant based on FFT. In *European Control Conference ECC'99*, Karlsruhe, Germany, September 1999.

[11] D. Henrion. *Reliable algorithms for polynomial matrices*. PhD thesis, Institute of Information Theory and Automation, Academy of Sciences of the Czech Republic, Prague, 1998.

[12] L. Qiu. What can Routh table offer in addition to stability? *Journal of Control Theory and Applications*, 1(1):9–16, 2003.

# Index