MATH LIBRARY version 2.10

An almost complete math package for dealing with matrices and polynomials

Written in SysRPL and Machine Language
Library number 1211

By Cesar Augusto Rorato Crusius - 1994

NOTICE

* The program MATH 2.10 and the documentation in this file are provided "as is", and are subject to change without notice. I give no warranty of any kind with regarded to the software or documentation, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Cesar Augusto Rorato Crusius shall not be liable for any error or for incidental or consequential damages in connection with the furnishing, performance, or use of this software and documentation.

* Sale of this material is not allowed without prior written permission of Cesar Augusto Rorato Crusius.

* Non commercial distribution allowed, provided that the original documentation is preserved unchanged.

* Modified versions are NOT allowed.

* Use of any part of MATH library code is not allowed for any purpose, except, of course, when used directly from the MATH library. You cannot reproduce any part of MATH library code for any purpose without prior written permission of Cesar Augusto Rorato Crusius. Use of MATH library in commercial or shareware programs is not allowed.

This program is Post-Cardware

* If you use and enjoy this excellent program you MUST send me a post-card. It will not cost too much, and I'll appreciate a lot ! Send it to the following address:

Cesar Crusius
Caixa Postal 5193
Trindade - Florianopolis - SC
ZIP CODE 88040-970

this address will work only 'til Jan 1996. So hurry up !

ABOUT

As we know, many times we need to perform calculations only with numbers. Many times we need to do polynomial calculus (multiplication, division and so on). Then, if we go one step ahead, we will need to do the same calculations with symbolic polynomials. As we go deeper, we will need many more things: symbolic matrices, matrices of polynomials, matrices of symbolic polynomials, matrices of matrices, et cetera.

MATH library is the way to do all those things! You can add a symbolic polynomial to a numerical matrix, and you'll get the correct answer: a symbolic matrix. Now you can get the inverse of the symbolic matrix, add a polynomial to this matrix and so on... Let's take an example. Let's do inv(sI-A) (familiar to engineers) in two ways (consider A=[[1 2][3 4]]):

First method: using the symbol 'S'

```
2:'S'            1:{{'S-1' -2}         2:{{'S-4' 2}
1:[[1 2]     MSUB   {-3 'S-4'}}  MINV     {3 'S-1'}}
  [3 4]]                           1:'(S-1)*(S-4)-6'
```

The second method uses [1 0] as a polynomial for 'S':

```
2:[1 0]            1:{{[1 -1] -2}       2:{{[1 -4] 2}
1:[[1 2]     MSUB   {-3 [1 -4]}} MINV     {3 [1 -1]}}
 [3 4]]                            1:[1 -5 -2]
```

As you can see, if you work with a lot of calculus THAT is the program you need.

LIMITATIONS and BUGS

This program is very difficult to debug because of its high complexity. You can do almost everything with the commands. If you try to do something you think is right and get a wrong answer, please notice me and I'll try to fix the bug. Any suggestions are welcome.

BE CAREFUL

There are times when the program has no way to discover what you are trying to do. As an example, take {{A 1 2}}. What is that? A 1x3 matrix or a zero-degree polynomial with the idependent coefficient equal to the symbolic polynomial {A 1 2}? There's no way to discover. The program will, in those cases, take the first choice. So DO NOT try to add {{A 1 2}} with {{1 2}}, because the program will think that you're trying to add two matrices. Do {0 {A 1 2}} + {0 {1 2}} instead. This will work correctly.

USER MODE

In the library commands description, sometimes I say that you can assign some command to some key. When I say that, I mean that you can STAY IN USER MODE and continue to use the keys as if none command were assigned to them if the arguments do not match those of the library command.

GX COMPATIBILITY

As discovered by Michael Guravage, MATH may not work correctly if installed on a RAM card that lies on the second slot of a GX calculator. So, if you have a GX, install MATH on the main memory or on the first slot.

MATH DATA TYPES

There are basically two kinds of data that you can use with MATH library: numerical and symbolic data. You represent numerical data with the "array" format of the HP48. Symbolic data is represented just replacing the "[]"s by "{}". Being that way, a numerical matrix for MATH have exactly the same format as for the calculator. Examples:

numerical 2x2 matrix: [[1 2][3 4]]
symbolic 2x2 matrix: {{ 1 2 }{ 3 A }}

Polynomials are represented as vectors, and the vector elements are the coefficients of the polynomial. Example: the polynomial x^3+2*x+3, for MATH, is [1 0 2 3]. Note that the polynomial variable is unknown! If you have symbolic coefficients you must use the symbolic format. Example: x^3+A*x^2+2*x+3 is represented as { 1 A 2 3 }.

VERSION DIFFERENCES

Version 1.00 and 1.10

Now the library is GX compatible! When the program detects that it's being used on a GX, MRoot dispatches to the GX own PROOT.

In version 1.0 MINV did not work correctly for 1x1 matrices. Now it does.

The program A<->L now works correctly when there are complex numbers involved.

Some differences in MEVL, e.g. it can be used as PR|L for matrices. It also dispatches to the system EVAL when arguments are not matching.

Many commands now dispatch to some system commands when arguments are not matching. Here are the list of the commands: MRoot dispatches to SQRT, MEVL dispatches to EVAL, MFCTP dispatches to XROOT and MDER dispatches to "delta". I made this so I can assign these commands to the corresponding system keys and I can stay in user mode to perform normal calculations.

ADPOL accepts any combination of lists/polynomials and execute MFCTP on the arguments that are polynomials.

When MFCTP receives a list it does nothing.

A lot of changes in FRPRC. You better read the command description.

Version 1.10 and 1.20

MMLT and MDIV now works normally, i.e., there are no more "element-by-element" product and MDIV does not accept that Matrix/Array stuff.

MMLT now can multiply a symbolic polynomial by a matrix. In earlier versions this was not allowed.

MDIV now can divide a symbolic matrix by anything but an array. When dividing a polynomial by a zero-degree polynomial the answer is a single polynomial. If the remainder of a polynomial division is a zero-degree polynomial, then the remainder is only a number, and not a single-element array.

A correction to the old version manual brings up a new capability of MEVL (it was already implemented in version 1.10, but I forgot to mention it on the old version manual :-). If you have a matrix of polynomials you can execute MEVL exactly as if you have a polynomial, i.e., every polynomial in the matrix will be evaluated for the given point (and the point can be anything: a matrix, another polynomial and so on).

EVECT and EVALUES commands added.

MADD and MSUB return numerical arrays when you add or subtract a complex number from a matrix. In earlier versions they return symbolic matrices.

Now the following commands test the matrix dimensions: MADD, MSUB, MMLT, MDET and MINV. The test is not complete, but make this commands much safer to use.

MEVL now treats a real number as a zero-degree polynomial.

ZTRIM is now transparent to objects that are not polynomials. When used on a zero-degree polynomial, the answer is the zero-degree coefficient. ZTRIM can also be used on a matrix.

Version 1.20 and 1.21

Some machine code added.

A<->L now accepts a list with symbolic elements. The program computes the numerical value of the symbolic elements before the transformation.

Version 1.21 and 1.30

MADJ command added, so now you don't need to invert a matrix in order to compute its adjoint. MINOR command added to compute the minor matrix.

Some utility commands added: MGET, MPUT, Row-, Row+, Col- and Col+. These commands follows the same syntax as for SX/GX GET and PUT, and Row+- & Col+- follows the same syntax as for GX ROW+- and COL+-, so now you can make a LOT of programs to deal with symbolic matrices.

MINV is now much faster, because now the matrix determinant is computed in a very fast way. In fact, the determinant of the matrix M is equal to the first row of adj(M) times the first column of M!

Version 1.30 and 2.00

SPEED! Now MATH is the fastest polynomial math program for the HP48 series! A lot of numerical polynomial tasks are performed 99% in machine language. The speed is improved by 1000% in many operations, such as MADD and MMLT (take a look at MPWR). Because of the internal representation for polynomials (arrays of long complexes), the precision is greater too! BELIEVE IT OR NOT, MATH's MEVL running under a version D HP48-SX is MUCH FASTER THAN HP48-G's PEVAL !!! MADD for arrays is FASTER THAN HP'S "+" !!!

SIZE! MATH 2.00 keeps almost the same size as MATH 1.30, thanks to the new MRoot program. The new MRoot is a little bit slower than the old one (the old and good PROOT), but it's a good roots program too, and memory is at a premium in my "dinosaur" HP48-SX version D. The new MRoot can take the roots of polynomials with complex coefficients! If you have a G(X), then MRoot will dispatch to PROOT and change the output format to a list.

MEVL can evaluate the nth-derivative (for any "n") of a polynomial as well!

New commands added for numerical and symbolic matrices: RSwp, CSwp, MSIZE and MTRACE. EVECT has gone, and SVALUES (singular values) takes its place.

Some corrections on Row+ and Col+. MGET is now user-proof (i think :-). MDET internally improved. It will test for zero pivots, so if your matrix have many zeros on the first row the program will execute A LOT faster. MDIV now divides complex coefficient polynomials.

Some polynomial tasks will return simplified polynomials, if possible, by transforming complex polynomials into real ones. Example: If you execute MADD with [(1,1) (0,0)] and [(1,-1) (2,0)], you will get [2 2] instead of [(2,0) (2,0)].

MROOT now is MRoot (there is a MROOT command in G series). But don't worry, your programs will keep working, 'cause the XLIB number remains the same.

MGET and MPUT now accepts lists with symbolic elements. The programs will compute the numerical value of the symbolic elements.

Version 2.00 and 2.10

Bug in SVALUES fixed, and now it works :-)

MEVL now can compute the nth derivative of a symbolic polynomial too. MDER now passes automatically through symbolic matrices. MDIV now can divide a numerical polynomial by a symbolic element, and can divide complex coefficients polynomials.

Col+ now returns a numerical matrix if the input arguments are two numerical matrices.

All commands (except PR|L) now works correctly with LASTARG. MEVL will keep only the first element of the stack. However, this feature eated more than a half kbyte of your precious memory.

SPEED MARKS (version 2.00)

I made some speed measures to compare MATH 1.30 and 2.00, and here are the results. The tests were made on my HP48 version D with 32k. Only MATH was on the calc, so there were about 20k free. All LAST flags were enabled (thus slowing the speed). When two lines are presented for the same test, then the first line contains the results for real polynomial tests, and the second contains the results for complex polynomial tests. Note that my calculator is an SX, so if you have a GX you'll be even faster!

command	v1.30	v2.00	improv (%)
V2 V1 MADD	1.870	0.096	1847.92
	2.084	0.101	1963.37
V2 V1 MSUB	1.948	0.173	1026.01
	2.181	0.188	1060.17
V2 V1 MMLT	34.405	1.004	3326.79
	48.726	1.202	3953.74
V2 V1 MDIV	23.925	3.093	673.52
	xxxxxx	7.130	xxxxxx
V1 2 MPWR	103.658	0.575	17927.47
	145.514	0.820	17645.61
[1 1] 10 MPWR	25.690	0.617	4063.70
V2 (1+j) MEVL	1.709	0.177	865.54
	1.803	0.172	948.26
V2 MDER	1.033	0.707	46.11
FRPRC	87.067	18.385	373.57
V1 MRoot	10.051	11.151	-9.86
	xxxxxx	32.799	xxxxx
V1 MFCTP	10.278	11.388	-9.75
V1roots MCOEF	11.757	0.849	1284.81
M1 MDET	181.594	71.468	154.09
M1 MINV	903.567	196.638	359.74
M2 EVALUES	189.759	27.816	617.06

The vectors and matrices used for the test were:

```
V1real = [ 1 2 3 4 5 6 7 8 9 10 ]
V1comp = (1,1)*V1real
V2 = V1 ^ 2

   [[ 1      2  0  4  0]
    [ 5      3  4  0  6]
M2 =  [ 0    2  5  6  2]
    [ 0      3  2 -1 -1]
    [-5      3  1 -2  8]]

M1 = [ 1 0 ] - M2
```

FRPRC test: in normal mode, numerator = [100 0] and denominator = { [1 2 5] 3 [1 1] 3 [1 2] }

Speed improvement = 100*(1.30/2.00 - 1)

MATH speed versus HP speed :-)

Just kidding, I love HP calcs, but you must look at this! Here are some speed comparsions between MATH commands and the similar HP commands. The HP used for the test was a version R HP48-G. The polynomials used were:

```
real:  P=[1 1] 40 MPWR
cmplx: C=[1 1] 40 MPWR (1,1) *
```

All results are in seconds. Take a special look at the comparsion between PEVAL and MEVL.

	HP48G	MATH
P P + vs MADD	0.275	0.134
C C + vs MADD	0.649	0.147
P (1,1) PEVAL vs MEVL	1.080	0.218
C (1,1) PEVAL vs MEVL	1.338	0.227

Please note two things:

i. MATH's MADD is faster AND more versatile than HP's command "+" for vectors: when adding two complex vectors (polynomials for MATH), MADD will return a real one if possible.

ii. MATH's MEVL is faster AND more versatile than HP's command PEVAL: MEVL can evaluate the polynomial

derivatives AND will return a real value if possible
(i.e. if the imaginary part of the result is zero).

LIBRARY COMMANDS DESCRIPTION

In this section a complete description of the library commands will be given. You can use any of the library commands to make your own programs in user or SysRPL. What I'm trying to say is that I will NOT change the library ID or the XLIB numbers. The only command that can have it's XLIB number changed is ABOUT.

MADD

Takes two arguments and try to add them. If one argument is a matrix M and the other is anything but a matrix (x), the program will perform the sum xI+M. If x is a real number and M is a numerical matrix, the answer will be a numerical matrix, as expected. Anything is allowed (matrices of matrices of matrices of polynomials, and so on...).

If you assign this program to the "+" key, be aware that the following operation won't be carried out in the USER mode:

```
{ 1 2 3 } 3 + ---> { 1 2 3 3 }
```

Instead, you'll get { 1 2 6 }.

MSUB

Takes two arguments and try to subtract them. If one argument is a matrix M and the other is anything but a matrix (x), the program will perform xI-M or M-xI (it will do what you asked for). Anything is allowed.

This program can be assigned to the key "-".

MMLT

Multiplication in numerical/symbolic form. You can do almost anything you want, e.g. multiply a numerical matrix by a symbolic polynomial, or a numerical polynomial by a symbolic polynomial. This program can be assigned to the key "*".

MDIV

Divides level 2 object by level 1 object. You cannot divide a matrix by an array, and level 1 object cannot be a symbolic matrix. If the arguments are not matching, the command dispatches to the HP own "/". When dividing a polynomial by another, the answer will be the result of the division on level 2 and the remainder on level 1. If the remainder is a zero-degree polynomial, it will be given as a number. If you divide a polynomial by a zero-degree polynomial, the answer will be a single polynomial (obviously the remainder will be zero).

MPWR

Takes an object in level 2 and raises it to the nth power, where n is the value of the real number in level 1. This program can be assigned to the user key "y^x" and you won't need to leave the user mode for the normal operation of the key.

MRoot

Takes a numerical polynomial in level 1 with real or complex coefficients and returns a list with the polynomial roots. If the polynomial have zero degree, then MRoot returns an empty list.

You can assign this command to the user key SQRT.

MFCTP

Takes a polynomial (like MRoot, but only real coefficients are allowed) in level 1 and factorizes it, returning the list of the polynomials. If the argument is a list the program will simply do nothing. Note that the leading coefficient of the factored polynomial will be ONE no matter the value of the leading coefficient of the original polynomial.

You can assign this command to the key XROOT.

MCOEF

Takes a list of roots in level 1 and returns the polynomial that have these roots. All the roots must be numerical, and they must correspond to a real coefficient polynomial. The leading coefficient of the resulting polynomial will be always one.

MEVL

Takes a polynomial (or a matrix of polynomials) P in level 2 (matrices of matrices of polynomials are allowed, and so on) and any object x in level 1 and computes P(x). P can be symbolic or numerical, and x can be anything (another polynomial or even a matrix). If the object on level 2 is a real number, MEVL will simply drop the level 1 object.

If you want to evaluate the nth-derivative of a polynomial, add a hex-string with the value of "n" in the arguments. Example:

```
3: [ 1 2 3 4 5 ]
2: 12
1: # 3d
```

MEVAL will compute the value of the 3rd derivative of the polynomial [1 2 3 4 5] at the point 12. It will return, in this case,

```
1: 300
```

You can assign this command to the key EVAL, but be aware -> this is a "strange" command. Strange things can happen if you try to use MEVL as the HP EVAL when there are more than one element in the stack (e.g., a list of variables on level 2 and anything on level 1), because MEVL will think that the list is a polynomial, and so on...

If you have a G(X), please note that MEVL is FASTER THAN PEVAL.

MDER

Takes a polynomial in level 1 and computes its derivative. The polynomial can be symbolic.

You can assign this command to the derivative key.

ZTRIM

Removes the left zeros of a polynomial or a matrix of polynomials. If the object is not a polynomial, nothing happens. If the result is a zero-degree polynomial, then ZTRIM will return only the first element of this polynomial.

MDET

Calculate the determinant of a square matrix. The matrix can contain anything: polynomials, symbolic elements or even matrices(!).

MINV

Calculate the inverse of a square matrix. You can have anything on the matrix. The program returns a numerical matrix if the input is a numerical matrix. If the input is a symbolic matrix, the program returns, in level 2, the adjoint matrix and, in level 1, the determinant. As we know, the inverse is: adjoint matrix divided by the determinant. Uses Faddeev method for symbolic matrices.

You can assign this command to the key "1/x".

MTRN

Calculate the transposed matrix. The matrix can be either symbolic or numerical.

MIDN

Creates an identity matrix with the dimension of level 1 and with the diagonal elements equal to the level 2 element.

A<->L

Transform a symbolic/numerical poly/matrix into a numerical/symbolic poly/matrix. The program will pass the HP-command ->NUM to every element on a symbolic representation, so now you can do things like

```
{ 1 '1/3' } ---> A<->L ---> [ 1 .3333... ]
```

PR|L

Given a list on level 2 and a program on level 1, executes the program on every element of the list. This program has a maximum depth of 2. What does it means? The "depth" of PR|L is the "maximum recursivity depth" of the program. So a "depth" of 2 means that PR|L will pass the program through lists that are in the original lists, but not through lists into these ones. See the following examples:

```
{ 2 }     << 2 + >> PR|L { 4 }
{{ 2 }}   << 2 + >> PR|L {{ 4 }}
{{{ 2 }}} << 2 + >> PR|L {{{ 2 2 }}}
```

If you want to override this depth limit, simply add one more parameter (a binary integer or a hex string) containing the maximum depth wanted. Example:

```
{{{ 2 }}} << 2 + >> #3 PR|L {{{ 4 }}}
```

FRPRC

There are two types of partial fractions: the normal type and the "Z-type". In the normal type, the numerators' degrees are smaller than the denominators' degrees. In the "Z-type", the numerators can have the same degree as the denominators (very useful for Z-transforms). User flag 31 control the current mode of operation. If flag 31 is SET, then the normal type is used, otherwise "Z-type" is used.

Level 1 can be a numerical polynomial or a list of numerical polynomials representing Q. Example: if Q=(x+1)(x+2)x^2 the list of level 1 could be:

```
{ [ 1 1 ] [ 1 2 ] [ 1 0 ] 2 } or
[ 1 3 2 0 0 ]
```

You must note that the leading coefficient of the denominator polynomial MUST BE ONE! If this is not the case, you must re-arrange the polynomials to get the correct denominator polynomial. Complete example in normal type (don't forget to set flag 31): calculate the partial fractions of

```
(x+1)/((x+4)^2 * (x+2) * x)

2: [ 1 1 ]
1: { [ 1 4 ] 2 [ 1 2 ] [ 1 0 ] }
```

the output of FRPRC is

```
1: {{ [ -.15625 ] [ 1 4 ] 1 }
  { [ -.375 ]   [ 1 4 ] 2 }
  { [ .125 ]    [ 1 2 ] 1 }
  { [ .03125 ]  [ 1 0 ] 1 }}
```

so the answer is

```
-.15625       .375    .125    .03125
-------- - -------- + ----- + ------
 (x+4) (x+4)^2     x+2      x
```

If you want to use the "Z-type" partial fractions, you must first clear flag 31. To use this mode of operation there is one constraint to the numerator: it must have a degree higher than zero, and the zero-order coefficient must be zero. If you think that that's too bad, remember that when you calculate the step response in Z you must multiply the numerator by Z. So, if you use this mode to work in Z-domain, this constraint will be not a problem at all. The answer will always have its numerators with the zero-order coefficients equal to zero (better for inverse Z-transforms). Complete example: calculate the partial fractions of

```
z/(z-1)(z-.5)
```

With flag 31 cleared, the stack must be

```
2: [ 1 0 ]
1: { [1 -1] [1 -.5] }
```

The program FRPRC will give you the answer:

```
1: {{ [ 2 0 ] [ 1 -1 ] 1 }
  { [ -2 0 ] [ 1 -.5 ] 1 }}
```

So the answer is

```
 2z     2z
---- - ----
z-1    z-.5
```

SOME DIFFERENCES FROM MATH 1.00: If you already have MATH 1.00, you'll soon notice that complete answers are always given. I did that because it seems to be better for the user and certainly it is better to make programs that use FRPRC. The old style FRPRC that takes a binary integer in level 1 is not allowed anymore.

A bug or a limitation? :-) Try to, whenever possible, use the list format for the denominator. If you don't, *strange* things may happen, as discovered by Tyson Leistiko. Example: try to perform, in the normal mode, the following partial fractions:

```
(a) -> [ 1 ] / [ 1 8 13 6 ]
(b) -> [ 1 ] / { [ 1 6 ] [ 1 1 ] 2 }
```

As you will note, the answers are different. Why? Because MRoot isn't precise enough to identify the multiple roots in -1 in the polynomial [1 8 13 6].

ADPOL

Make a FRPRC list from lists/polynomials.
Examples:

```
2: { [ 1 0 ] }
1: [ 1 0 ]      -> ADPOL -> 1: { [ 1 0 ] 2 }

2: [ 1 2 1 ]
1: [ 1 2 5 ]    -> ADPOL -> 1: { [ 1 2 5 ] 1 [ 1 1 ] 2 }
```

Note: one argument at least must be a numerical polynomial.

STDL

Prepares a list to be used by FRPRC (the program FRPRC automatically calls this program, so you will only use it if you will build your own programs).

Example:

```
1: { [ 1 0 ] [ 1 2 ] 2 [ 1 0 ] 3 [ 1 2 ] 5 [ 1 4 ] }
```

STDL gives

```
1: { [ 1 0 ] 4 [ 1 2 ] 7 [ 1 4 ] 1 }
```

EVALUES

Takes a numerical square matrix (real or complex) on level 1 and returns the list of the Eigenvalues of the matrix. Example:

```
1: [[  3 -1  1 ]
  [ -1  5 -1 ]
  [  1 -1  3 ]]
```

EVALUES will return

```
1: { 2 3 6 }
```

This program uses Faddeev's method to find the characteristic polynomial of the matrix and then executes MRoot. This program is slower than GX's EGVL. So, if you have a GX, you may want to use GX's own command. The fact is that an SX owner can build his programs using MATH's EVALUES and the programs will run without modification on any GX with MATH 1.20 or newer installed.

SVALUES

Takes a numerical matrix (real or complex) on level 1 and returns the list of the singular values of the matrix. Example:

```
1: [[  3 -1  1 ]
  [ -1  5 -1 ]]
```

SVALUES will return

```
1: { 2.63787896258 5.57149841414 }
```

MADJ

Computes the adjoint matrix. One more time, the input matrix can have anything as its elements. Uses Faddeev's method.

MINOR

Computes the minor matrix. You must put in level 3 the matrix, in level 2 the row and in level 1 the column to be removed. Example:

```
3: { { 1 2 } { c d } { f 5 } }
2: 2
1: 2
```

MINOR will give you

```
1: { { 1 } { f } }
```

MGET

Pick an element from a matrix/polynomial. As in MPUT, symbolic matrices can have ANY NUMBER OF dimensions! The command follows the same syntax as the HP command GET. Example:

```
2: { { { { A B } } { { C D } } } }
1: { 1 2 }
```

MGET will give

```
1: { { C D } }
```

MPUT

Puts an element into a matrix/polynomial. Note that only numbers can be putted into numerical matrices. Another important thing to note is that symbolic matrices can have ANY NUMBER OF dimensions, and not only two! The command follows the same syntax as the HP command PUT. Let's do an example with a symbolic matrix with four dimensions:

```
3: { { { { 1 2 } } { { 3 4 } } } }
2: { 1 2 1 2 }
1: A
```

MPUT will give

```
1: { { { { 1 2 } } { { 3 A } } } }
```

WARNING: In this version MPUT does not perform ANY tests on the giving index! If you give an invalid index you may (and certainly will) loose your memory!

Row-

Removes a row from a matrix. The program returns in level 2 the modified matrix and, in level 1, the removed row. If the original matrix has only one row, the program will return the unity matrix [[1]] or {{1}}. Example:

```
2: { { A B } { C D } { E F } }
1: 2
```

Row- will give you

```
2: { { A B } { E F } }
1: { C D }
```

Row+

Puts one or more rows into a matrix. You must put in level 3 the original matrix, in level 2 the row (or matrix of rows) and in level 1 the line where the rows will be inserted. If the number in level 1 is greater than the matrix dimensions, then the row(s) will be inserted at the end of the matrix. Example:

```
3: [ [ 1 2 ] [ 3 4 ] ]
2: { A B }
1: 2
```

Row+ will give

```
1: { { 1 2 } { A B } { 3 4 } }
```

Another example:

```
3: { { 1 2 } { C 4 } { x y } }
2: { { a b } { c d } }
1: 3
```

Row+ will give

```
1: { { 1 2 } { C 4 } { a b } { c d } { x y } }
```

Note that any combination of symbolic/numerical matrices are allowed.

Col-

Removes a column from a matrix. The program returns in level 2 the modified matrix and, in level 1, the removed column. If the original matrix has only one column, the program will return the unity matrix [[1]] or {{1}}. Example:

```
2: { { A B } { C D } { E F } }
1: 2
```

Col- will give you

```
2: { { A } { C } { E } }
1: { B D F }
```

Col+

Puts one or more columns into a matrix. You must put in level 3 the original matrix, in level 2 the column (or matrix of columns) and in level 1 the line where the columns will be inserted. If the number in level 1 is greater than the matrix dimensions, then the column(s) will be inserted at the end of the matrix. Example:

```
3: [ [ 1 2 ] [ 3 4 ] ]
2: { A B }
1: 2
```

Col+ will give

```
1: { { 1 A 2 } { 3 B 4 } }
```

Another example:

```
3: { { 1 2 } { C 4 } { x y } }
2: { { a b } { c d } { e f } }
1: 3
```

Col+ will give

```
1: { { 1 2 a b } { C 4 c d } { x y e f } }
```

Note that any combination of symbolic/numerical matrices are allowed.

RSwp

Exchange (swap) two rows of a matrix. The level 3 must contain the matrix and the levels 2 and 1 the index of the rows to be swapped.

CSwp

Exchange (swap) two columns of a matrix. The level 3 must contain the matrix and the levels 2 and 1 the index of the columns to be swapped.

MSIZE

No more SIZE EVAL for polynomials and vectors! MSIZE will give you the size of the object in level 1. If the object is a polynomial (or a vector), then it will return just a number. If it is a matrix, MSIZE will return a list with the number of rows and columns.

MTRACE

Computes the trace of a square matrix, i.e., the sum of the elements of the matrix diagonal. Of course it will work for symbolic matrices ;-)

ABOUT

Shows my name !!! :-)

UTILITY PROGRAMS

Included with the package are some utility programs. Here is the description:

MVID

If you do not use my Control Package, then this program can be useful. Execute CNVID once or twice, until "(S/Z)MOD" appear on the top of the screen. The annunciators indicate what is the current type of partial fractions and the current MATH version.

MFLAG

Use MFLAG to change the type of partial fractions performed without have to (re)set the flag 31 by hand. Again, if you have Control Package you'll not need this program.

APPLICATION NOTES

MEVL hints and suggestions

This program can be used as PR|L if you have a symbolic matrix in level 2 and a program in level 1. In this case, the program will be evaluated for every matrix element, just as PR|L does.

This program can be very useful for other tasks. Example: if you have the polynomial x^3 + 2*x^2 + 3 and want to make the variable substitution x=s+1, you can get the polynomial in s by typing:

```
2: [ 1 2 0 3 ]
1: [ 1 1 ]
```

Now you execute MEVL (evaluates the polynomial when x is another polynomial) and get the answer:

```
1: [ 0 1 5 7 6 ]
```

Of course you can get the polynomial in x back by evaluating the s polynomial at [1 -1]

```
1: [ 0 0 1 2 0 3 ]
```

The leading zeros appears due to the evaluating algorithm. The answer is right anyway...

Another thing that you can do with MEVL is to transform a numerical polynomial into a symbolic polynomial. Example:

```
2: [ 1 2 3 ]
1: 'S'
```

MEVL will return '(S+2)*S+3'. Beside the fact that that's an "unusual" format for symbolic polynomials, the answer is right. And, as you see, this format is faster to evaluate and give better results than the "standard" format.

Interesting application of A<->L

A suggestion: when you get the inverse of some matrix, use A<->L and evaluate the program << ->Q >> over it (if you have a G series calculator you can simply execute the command ->Q). As an example, try to do this with the matrix [[3 1][-4 1]]. The symbolic matrix will be much better to visualize.

As you may note, the inverse procedure can be useful when entering matrices like [[1/3 1] [0 1/7]]. Simply enter {{ '1/3' 1 }{ 0 '1/7' }} (some may prefer the matrix editor to this method). After that, use A<->L and get the numerical matrix.

Rank of a matrix

If you have a GX you have a command to compute the rank. But if you haven't, you can compute the rank by checking the number of non-zero singular values of the matrix. The following program will return the rank of a matrix:

```
<< SVALUES ZTRIM SIZE >>
```

Computing the determinant in a faster way

MDET now tests for zero pivots. So, if you have, on your matrix, a row with many zeros, swap (using RSwp) this row and the first. The determinant will be computed in a faster way. But don't forget to negate the result accordingly !

Successive divisions

This program is very useful, for example, if you want to simulate a digital system. It takes the numerator from the 3rd level, the denominator from the 2nd level and the number of divisions to carry out in the first level. The program will return a list with the values of the divisions.

```
<< OVER SIZE [ 0 ] SWAP RDM 4 ROLL MADD -> Q N P
 << { } 1 N
    START
          P Q MDIV [ 1 0 ] MMLT
          'P' STO 1 GET DUP 1 DISP +
    NEXT
 >>
>>
```

Ackermann's formula

Suppose you have a SISO system dx=Ax+Bu and you want to use the control law u=Kx to place the poles of the closed-loop system at a certain location. Then you can use Ackermann's formula to compute the gain matrix K. This program computes this gain matrix using Ackermann's formula and the following data:

```
3: A
2: B
1: { desired poles (MCOEF format) }
```

The program will give you the gain matrix K:

```
<< MCOEF -> A B P
 << B DUP 2 A SIZE 1 GET
    START A ROT * SWAP OVER 1000 Col+
    NEXT INV P A MEVL * SWAP TRN 0 * DUP
    SIZE -1 PUT SWAP *
 >>
>>
```

SPLIT

The following program, "SPLIT", is a perfect example of how much work can be saved by using MATH and a little programming. Suppose you have the polynomial fraction P(s)/Q(s). Now you make s=jw, and you want to split the original fraction into the real and imaginary parts, Pr(w)/Qr(w) + j*Pi(w)/Qi(w). The following program will do the task: it takes P and Q from the stack and returns on level two the list { Pr Qr } and on level one the list { Pi Qi }

```
<< [ (0;1) (0;0) ] ROT OVER
 MEVL ROT ROT MEVL ZTRIM
 DUP CONJ ROT OVER MMLT ROT
 ROT MMLT RE OVER RE ZTRIM
 OVER 2 ->LIST ROT IM ZTRIM
 ROT 2 ->LIST
>>
```

Example: P(s)/Q(s) = 1/s(s+1)^2

```
2: [ 1 ]
1: [ 1 2 1 0 ]
```

After SPLIT you'll get

```
2: { [ -2 0 0 ] [ 1 0 2 0 1 0 0 ] }
1: { [ 1 0 -1 0 ] [ 1 0 2 0 1 0 0 ] }
```

So the decomposition is:

```
P(jw)     -2            w^2 - 1
----- = -------------- + j*---------------
Q(jw)   w^4 + 2*w^2 + 1   w^5 + 2*w^3 + w
```

A NOTE ON THE METHODS

MRoot uses Laguerre's method only. It works VERY well for polynomials with distinct roots, and is satisfactory for polynomials with multiple roots. If you want to know more about this method try the book "A SURVEY OF NUMERICAL MATHEMATICS", by Young and Gregory. If you are using a G(X), then MRoot is simply your PROOT with some modifications for interfacing with MATH.

I use Faddeev's method a lot of times in the program. The method can be used to compute a lot of things of a matrix: the characteristic polynomial, the adjoint, the determinant and all the Eigenvectors. If you want to know more about Faddeev's method try the book "MATRIX THEORY", by Gantmacher.

THANKS TO

I want to thank the following people for their support and suggestions:

Andre Hentz (andre@lcmi.ufsc.br)
Arie Leib Bukinsky (bukinsky@jct.al.il)
David Peterson (18084DEP@msu.edu)
Hoa Van Lai (imach@cc.utexas.edu)
Joe Horn (joehorn@hpcvbbs.external.hp.com)
Keith Maddock (madd0118@nova.gmi.edu)
Marcelo Rodrigues (marcelor@acs.bu.edu)
Marcus Kindel (kindel@inf.ufrgs.br)
Mario Mozgy (mozgy@diana.zems.etf.hr)
Michael Guravage (Michael.Guravage@cwi.nl)
Mika Heiskanen (mheiskan@delta.hu.fi)
Sean McNamee (seanmc@u.washington.edu)
Tyson Leistiko (eleistik@elee.calpoly.edu)
Vinicius Vasconcellos (cello@vortex.ufrgs.br)