Erable 2.93.

Bernard Parisse
Institut Fourier (CNRS UMR 5582)
Université de Grenoble I
F-38402 St Martin d'Hères Cédex
Tél. (33 — 0) 4 76 51 43 14
Bernard.Parisse@ujf-grenoble.fr

April 27, 1997

Contents

1 License

2 Installation.

2.1 Getting the binaries from a computer.

Connect your HP to your computer. Run kermit in server mode on your computer, and type:
`{ kernel.lib erableg.lib ALGBG } KGET`
on your HP48 GX model or:
`{ kernel.lib erable.lib ALGB } KGET`
on your HP48 SX model.

2.2 Getting the binaries from another HP48.

Put your HP48 in server mode. On the other HP, type the following little program:

```
<< :&:787 RCL :&:788 RCL RCLKEYS -> kernel.lib erable.lib touches
   << { kernel.lib erable.lib touches } SEND
   >>
>>
```

and EVAL it.

2.3 Installing the binaries

The kernel library must be installed in port 0 or port 1, for port 0 type:
`'kernel.lib' DUP RCL SWAP PURGE 0 STO`
The erable library may be installed in any port, for example in port 2:
`'erable.lib' DUP RCL SWAP PURGE 2 STO`

Optional but recommended:

- if you got ALGB or ALGBG from your computer, go in the ALGB or ALGBG folder and hit INIT. This will assign keys in user mode and create a little CST menu.
- if you got touches from another HP, type:
  `'touches' DUP RCL SWAP PURGE STOKEYS`
  to assign keys in user mode.

3 Introduction.

3.1 Overview.

erable is a computer algebra system for the HP48. The main features are simplifications (including complexes and square roots), integration, first order differential equations, partial fraction decomposition, Laplace and inverse Laplace transform, Taylor series, row reduction to echelon form of matrices, linear system (including over and underdetermined), eigenvalues and eigenvectors, quadratic forms, permutations, variables substitution, ... With erable you will be able to solve most of the problems solved by a TI92 (but you'll have to wait a little bit more!) and some problems which are not solved by a TI92: some integrals (the

Risch algorithm is not implemented in the TI92), some Taylor series, arithmetic, diagonalisation of matrices, change of variables,...

Examples (# means not solved natively by the TI92):

$\dfrac{1+\sqrt{2}}{1+2\sqrt{2}}$	`EXPA`	$\dfrac{3}{7}+\dfrac{1}{7}\sqrt{2}$
$e^{i\pi/6}$	`TSIMP`	$\dfrac{\sqrt{3}}{2}+\dfrac{i}{2}$
$\ln(1+i)$	`TSIMP`	$\dfrac{\ln(2)}{2}+\dfrac{1}{4}\pi i$
$\int \dfrac{1}{e^x-1}\,dx$	`RISCH`	$\ln(e^x-1)-x$
$\# \int (1+2x^2)e^{x^2}\,dx$	`RISCH`	xe^{x^2}
$\# \int_a^b \dfrac{e^x}{x}\,dx, e^x=y$	`EXEC`	$\int_{e^a}^{e^b} \dfrac{y}{y\ln(y)}\,dy$
$\# \sin(x)/(e^x-1), x, 4$	`SERIES`	$1-\dfrac{1}{2}x-\dfrac{1}{12}x^2+\dfrac{1}{12}x^3+o(x^3)$
$\# \cos(x)^2$	`1 LAP`	$\dfrac{1}{2x}+\dfrac{1}{4(x-2i)}+\dfrac{1}{4(x+2i)}$
$\# \dfrac{1}{(x^2+1)(x+1)}$	`ILAP`	$\dfrac{-1}{2}\cos(x)+\dfrac{1}{2}\sin(x)+\dfrac{1}{2}e^{-x}$
$\# y'=xy^2$	`'X*SQ(Y(X))' DSOLVE EXPA`	$y=\dfrac{y_0}{e^{-1/2x^2+1/2x_0^2}}$
$\begin{pmatrix} 1 & 1 & a \\ 1 & a & 1 \\ a & 1 & 1 \end{pmatrix}$	`rref`	$\begin{pmatrix} a-1 & 0 & a^2-1 \\ 0 & a-1 & -a+1 \\ 0 & 0 & -a^2-a+2 \end{pmatrix}$
$\# \begin{pmatrix} 0 & 1 \\ -1 & 2 \end{pmatrix}$	`JORDAN`	char. vector $(1)(1,0)$; eigenvector $(1)(-1,-1)$

Note that this version `erable` is not the ultimate release of `erable`, see the last section (15, p. 24).

3.2 Warnings.

- Using a computer algebra system does not mean that you don't have to think. Most of the time, all works perfectly and you get the answer within 30 seconds. But sometimes, after 1 or 2 minutes, you don't get the answer or you get a Insufficient memory error. In this case, you should think "Is there a different way to get the answer? Is there a way which will be easier for the system?" And most of the time, there is a better way! Think of double integrals where you can reverse the integration order or define integrals where you may do a variable translation to have less variables, or linearity in inverse Laplace transform, ... You should learn maths and algorithms to get the best of any computer algebra system. And a system is never complete, you will need to program sometimes!
- Most of the problems in the real life don't have exact answers but can only be solved approximately. Think of integrals, differential equations, great matrices (say e.g. 100×100), ... Before learning how to solve exactly a problem, I strongly recommend to learn how to solve numerically a problem. Then for a real life problem, you will know when you must stop finding an exact solution and begin to use a numerical algorithm.

3.3 `erable` and `alg48`.

`erable` is partially derived from the `alg48` package. The arithmetic functions of `erable` are derived from those of alg48. erable and alg48 have some other common features like simplifications, partial fraction expansion or rational integration. The main differences are:

- erable is slower (about 3 to 4 times I would say). Why? Because erable handles complexes and square roots natively: e.g. erable simplifies expressions like $(1+i)/(1+2i)$, $(1+\sqrt{2})/(1-2\sqrt{2})$, $e^{5i\pi/6}$. Hence simplifications algorithms of erable are generic (-¿ slower).
- erable accepts strings embedded in symbolics, this means that if you EXPAnd $(5x+12)^{16}$ with erable you'll get the exact answer.
- erable handles numerical data (non integers real and complexes)
- erable has a partial implementation of the Risch integration algorithm: it handles most of the common integrals, not only rational fractions.
- alg48 (version 4) implements the complete factorization algorithm over the rationals, erable finds only first order factors of the square-free factorization. For 1-variable polynomials calls the numeric solver if necessary and tries to rebuild 2nd order factor. Hence, alg48 factors the expanded form of $(x^4+x^3+1)(x^4+x+1)$ but not x^4+1 and erable does not factor the first example but factors the second one.
- The main specific feature of alg48 is the Gröbner base computation. The main specifics features of erable are eigenvalues and eigenvectors of matrices, differential equations (first order: linear, Bernouilli, homogenous; linear with constant coefficients), Taylor series, quadratic forms, permutations, variable substitutions.

If you have enough memory, do like me: keep both on your calc and choose the right instruction!

3.4 Implementation notes.

This software is written in Sysrpl with HP48 supported entries and standard instructions (and a few unsupported but static entry), hence it should work on all versions of the HP48. Of course, you should backup your calc before trying it: no software is bug free!

This package was written on a 90Mhz Pentium PC running under Linux version 2, with XFree86 (X-Windows system), the GNU emacs editor, my patched version of the x48 emulator (with almost instantaneous file transferts) with JAZZ installed, the gtools package (HP48 GNU compiler) and kermit.

3.5 Next upgrades.

The latest versions are available by anonymous ftp in the directory:
ftp://fourier.ujf-grenoble.fr/pub/hp48/
If you have a WEB client, you may prefer to go to my professional homepage:
http://www-fourier.ujf-grenoble.fr/~parisse
or to my personal homepage:
http://perso.wanadoo.fr/bernard.parisse

4 Data and main functions of the library

4.1 Data types.

Data handled by erable have two representation: the user representation which you use most all the time and the internal representation. If the user flag 17 is cleared, erable is in user representation mode, otherwise, erable is in internal representation mode. Sometimes you will be astonished by the results of erable functions because you believe that you are in user representation mode but you are in fact in internal mode. **You can use the VER instruction to reset all flags in the usual state**.

List of data types:

True data	Example	User	Example	Internal	Example
Integer	5	real, hex, string	5	hex	#5
Float	5.02	real	5.02	long real	%% 5.02
Gauss integer	$1+2i$	symbolic	`'1+2*i'`	secondarie	`:: #1 #2 ;`
Complex	(1.1,2.3)	complex	`(1.1,2.3)`	long complex	`C%% 1.1 2.3`
Fractions	$\frac{2}{3}$	symbolic	`'2/3'`	symbolic	`'#2/#3'`
Irr. quadr.	$1+2\sqrt{5}$	symbolic	`'1+2*V3'`	program	`<< #1 #2 #5 >>`
Unknowns	a, x ...	variables	`A X`	list	*variable*
Symbolics	$a+x^2$	symbolic	`'A+X^2'`	list	*variable*
Lists	{ 1 i }	list	`{ 1 'i' }`	list	`{ #1 :: #0 #1 ; }`
Array	`[ 1 2 ]`	array	`[ [ 1 2 ] [ 3 4 ] ]`	array	`[ [ 1 2 ] [ 3 4 ] ]`
Symb. array	`{ 1 2 }`	list	`{ { 1 2 } { 3 4 }`	list	`{ { #1 #2 } { #3 #4 }`

Notes:

- You can use binary integer (Sysrpl hexstrings) with erable but no simplification routines are provided for this data type.
- Numeric and symbolic vectors are of course allowed (1-d arrays).
- List of lists are used for symbolic matrices (a symbolic matrice is like a numeric matrice, just replace [by and] by).
- For Sysrpl programmers: you can use the arithmetic operations on long real and system binary integers.

Remarque 1

- *If flag 13 is cleared (real mode), all global names and all non rational functions are considered as real w.r.t. the instructions* `RE`, `IM`, `CONJ`. *This could lead to false simplifications if a global name stays for a complex, or if a non-rational inverse function is called with a usually forbidden real argument, like* `LN(-1)` *or* `ASIN(2)` *Solution: in the first case, replace your global name, say 'Z', by 'X+iY'. In the second case, set flag 13.*
- *In internal mode, symbolics returned by* `erable` *may contain lists. This is not allowed by the HP compiler/decompiler, hence the stack display may show something like:* `'UNKNOWN/2'` *or* `'UNKNOWN/UNKNOWN'` *or* `'UNKNOWN+X'` *(e.g. by typing* `{ #1 #2 } { #3 #4 } NDXF`*) Caution, you can* not *edit such objects. Solution: use* `FXND` *to know the numerator and denominator of the above fractions, or use another stack display like* `EQSTK` *(by Mika Heiskanen) or* `JAVA` *(by Richard Steventon and Andre Schoorl).*

4.2 User flags.

You may change the behaviour of `erable` by clearing or setting some user flags (and the system flag -27). Here are the most important flags of `erable`:

- flag 1: display flag (if set then verbose mode selected)
- flag 10: integer arithmetic (if set then `erable` expects integer arguments for instructions like `GCD3`, `ABCUV` or `DIV2`).
- flag 12: simplification flag (if set then `erable` calls the gcd algorithm if needed)
- flag 13: complex flag (if cleared, all expressions are assumed to be real and `erable` tries to return only real expressions)
- flag 15: real are integer flag (if set, real are assumed to be integer)
- flag 17: internal flag (if set, all data are assumed to be internal)
- flag 21: recurse flag (if set, the simplification algorithms `EXPA` and `TSIMP` will simplify in subexpressions)
- system flag -27: if set, then the user representation of complex numbers is symbolic.

To set a flag (say flag 12), type `12 SF`. To clear this flag, type `12 CF`.

4.3 Main functions.

The main functions of the erable library are assigned to keys in user mode. If you don't see the word USER in the status area (or with JAVA if you see NORM in the status area), you should go in USER mode: press once or twice the left shift key followed by the α key.

The redefined keys are most of the time preceded by the $\boxed{\alpha}$ and right-shift keys. For example, the add addition function of erable is obtained in USER mode by hiting threes keys: $\boxed{\alpha}$, $\boxed{\text{Right Shift}}$, $\boxed{+}$.

The same method applies for other operations. The $+$, $-$, $*$, $/$, y^x, $\sqrt{x}$, ∂, $\int$, $1/x$, $\pm$ keys are redefined (after $\boxed{\alpha}$ and $\boxed{\text{Right Shift}}$) as add, SUBT, MULT, DIV1, POWER, SQRT, CHS, der, RISCH, INVL. These commands extend the usual HP commands to all erable data types, except for the RISCH program, which tries to integrate with respect to the main variable (contained in 'VX': usually 'X').

The other α-right shift-redefined keys are:

- the 9 key calls EXPA which extends the EXPAN function of the HP. It fully expands rational expressions in one pass.
- the 6 keys is redefined as COLC which tries to factorize an expression (this has a similar effect to the COLCT instruction).
- the 3 keys calls TSIMP, the simplification program for transcendental functions (exponentials and logarithms).
- The 5 key calls the EXEC function which is used to execute something (e.g replace a variable by a value, or execute a program) on an object
- the 2 keys lets you switch from numeric to symbolic array representation of an array.
- the 7 key finds the roots and poles of a rational fraction (function FROOTS),
- the 4 key evaluates an object at stack level 1 and returns an approximation of the time needed (tEVAL function).
- the 1 key switches on or off the complex mode (i.e. user flag 13), the same key but α-left shifted switches on or off the internal flag 17. The new state is displayed. The R key has a similar effect for the recurse flag (21) (if α-right-shifted, for the α-left-shift-R this switches the $+/-$ integration flag on or off)

Let's finish this section by two redefined keys which are *not* α-right shifted:

- On SX models, the →Q and →NUM keys (not alpha shifted) are redefined to handle matrices. They toggle user flags 12, 14 and 15 (XNUM to clear and XQ to set).
- On a G/GX models, use α -$\boxed{\text{Right Shift}}$-Q or →NUM (not shifted).

5 Simplifications.

5.1 Main simplifications instructions.

Two kinds of simplifications are provided: full rational simplification (EXPA) and transcental simplification (TSIMP). Transcental simplification is used to detect relations between exponentials and logarithms. Since trigonometric functions are complex exponentials, they are simplified by TSIMP too. In addition erable operations perform automatic rational simplifications (if you are not in internal data representation mode).

The TSIMP output is affected by the state of the flag 13: if cleared, then complex logarithms and exponentials are converted in arctan and sin/cos functions.

The COLC instruction may be used as a simplification instruction, it tries to factorize a symbolic.

5.2 Other simplifications instructions

- `PFEXPA`: like `EXPA` but only in subexpressions between $+$ and $-$. Usefull before or after integration. The `PFCOLC` instruction behaves like `PFEXPA` but calls `COLC` on subexpressions.
- `EXPLN`: replace transcendental functions by exponentials and logarithms, tries to expand the result as a sum of exponentials.
- `TRIG`: replace complex exponentials and logarithms by trigonometric functions and tries to simplify sinus and cosinus fonctions together (applying $\sin^2 + \cos^2 = 1$).

5.3 Recurse flag.

If flag 21 is set, "variables" of an expression are simplified recursively (global name are evaluated, integrals are evaluated by a call to `RISCH`). Warning: derivatives of user-functions are not evaluated (you have to do an explicit substitution with `EXEC` for this).

6 Substitution, change of variables.

The syntax is `'old_name=expression' EXEC`. `oldname` may be a global name, an expression (in this case, the first global name in this expression will be isolated) or a user-defined function.

For multiples substitutions, the syntax is
`{ old_name_1 ... old_name_n } { expr_1 ... expr_n } EXEC`
In this case, `EXEC` does only substitutions.

Examples:

- `'X^2+2*X+5' 'X=1' EXEC`: evaluate an expression at $x = 1$.
- `'X=Y^2' EXEC`: change of variables, works in integrals too
- `'2*Z(X)-X*d1Z(X)' 'Z(X)=X^2 EXEC`: in a differential equation, replace the function $z(x)$ by x^2.
- `'Z(X)+d1Z(X)' 'Z(X)=EXP(-X)*Y(X)' EXEC`:
  change of function in a differential equation.
- `'X^2+X*COS(X)' 'X^2=1-Y'  EXEC`: replace x^2 by $1 - y$ and replace x by $\sqrt{1-y}$.
- `'SIN(X)^2+SIN(X)*COS(X)' { 'SIN(X)^2' } { '1-COS(X)^2' } EXEC`: replace $\sin(x)^2$ by $1-\cos(x)^2$ but does not replace $\sin(x)$ by $\sqrt{1-\sin(x)^2}$.

The `EXEC` programs checks the object type at stack level 1 and performs the corresponding action:

- *doall* function:
  if stack 1 is a program, `EXEC` executes program at stack level 1 recursively on the components of a list object at stack level 2. Example: `{ 1 2 3 } << NEG >> EXEC`
  is the same as `{ 1 2 3 } CHS`
- one algebraic substitution:
  If stack 1 is an equation (`'objA=objB'`), replace `objA` by `objB` in stack2.
- mulitple substitutions:
  If stack 1 and 2 are lists, replace each object of list2 in stack level 3 by the corresponding object of list1. Example:

```
'COS(X)+i*SIN(X)'
{ SIN COS }
{ << i * EXP DUP INV - i 2 * / >>
  << i * EXP DUP INV + 2 / >>
}
```

replace sin and cos by complex exponentials. If you call EXPA after you get e^{ix}.

- variable isolation:
Otherwise, EXEC tries to isolate stack level 1 in stack level 2. Example:

```
'X^2-5'
X
EXEC
```

returns 'X' $\sqrt{5}$.

7 Factorization.

7.1 Summary of the instructions.

- COLC: factorize a symbolic fraction (returns a symbolic). Factorization may be incomplete, but is squarefree and all first order factors are detected.
- FROOTS: given an object as input, outputs the list of var (stack level 3), the list polynomial (2), and the list of root/multiplicity (1) (each root is followed by its multiplicity). Examples:

```
* 'X^3-6*X^2+11*X-6'
  -> 3: { X }, 2:'X^3-6*X^2+11*X-6' , 1: { 2 1 3 1 1 1 }
* '1/X^2' -> 3: { X }, 2: '1/X^2', 1: { 0 -2 }
* 'X^2-Y^2' ->
3: { X Y }, 2: 'X^2+2*X*Y+Y^2', 1: { '-Y' 2 }
```

For a symbolic, FROOTS factorizes with respect to the variable contained in VX, or if the symbolic is independant of VX with respect to the first variable of LVAR applied to the symbolic.

If stack 1 is a real integer, FROOTS computes the roots of a list polynomial with numeric coefficient using Bairstow method (real coefficients) or Laguerre method (complex coefficients). During iterations, you can modify some parameters:

  - E*10: ($\varepsilon \times 10$) multiplies the test value by 10, use this when there are multiple roots.
  - E/10: divides the test value/10, for accurate precision (use this after you have found all multiple roots)
  - RAND: reset current iteration (restart with random initial value)
  - STOP abort iteration (for the next one or two roots)

Displayed are the last found roots and the current test value (to compare with the ε value). Before starting the program, you must specify the ε value and the number of test-successfully iterations using the following stack input:

  - 3: list polynomial,
  - 2: test value (positive real),
  - 1: iteration number (real integer)

Example:

```
{ 1 -21 183 -847 2196 -3024 1728 }
1E-4
3
FROOTS
-> approximatively { 3 3 3 4 4 4 }
```

The result is bad since 3 and 4 are multiple roots.

- `FACTO`: same stack as `FROOT` but returns a list of "prime" factors instead of roots. Example:

```
* 'X^3-6*X^2+11*X-6'
  -> 3: { X }, 2: 'X^3-6*X^2+11*X-6', 1: { '(X-2)' 1 '(X-3)' 1 '(X-1)' 1 }
* 21 -> 3: { }, 2: 21,  1: { 3 1 7 1 }
```

- `FCOEF`: input is a list of roots/multiplicity, output is a fraction or polynomial with leading coefficient 1 having this roots (and poles). Example:
  `{ 1 1 2 1 3 1 } -> 'X^3-6*X^2+11*X-6'`
  `{ A -1 2 1 } -> '(X-2)/(X-A)'`

7.2 A word about factorization.

Factorization of polynomial is very important in several mathematical functions, like symbolic integration or matrix diagonalization. It is important to understand the mechanism used by `erable` to perform this tasks.

Let's begin by recalling some mathematicals facts:

- Theorem (d'Alembert):
  A polynomial of degre n has exactly n complex roots (counted with multiplicity).
- Formula exists to get the solution of polynomials up to order 4 but Galois proved the following theorem last century:
  There is no formula for solving a generic polynomial of degre ≤ 5 (by algebraic operations and extraction of n-th roots)

This means that you can not root a multivariate polynomial of order ≥ 5 (for such polynomials, systems like Maple, Reduce, Axiom Mathematica or Mupad use algebraic extension), and that you can only root numerically a univariate polynomial of order ≥ 5. Note that the generic solution of a polynomial of order 3 is still complicated and of order 4 very complicated. I think that it is not possible to handle the generic solution of polynomials of order 3 or 4 on the HP48 in a raisonable amount of time. Hence, only polynomial of order 2 are generically solved by `erable`.

However, in some situations, you can root exactly polynomials of order ≤ 3, by searching multiple roots and by finding obvious roots (or obvious factor). The rooting algorithm of `erable` search first multiple roots by computing the `GCD` of the polynomial and his first derivative (this is the `SQFFext` algorithm in the source of `erable`). Of course flag 12 must be set for this step to be done. If a univariate polynomial has only integer (or rational) coefficients, you can find all rational solutions of this polynomial by testing a finite set of rationals (of the form numerator/denominator where numerator is a divisor of the constant coefficient and denominator a divisor of the leading coefficient). This is implemented in `erable` by the nullnamed `XLIB EVIDENText` which is called if flag 14 is set. Hence, `erable` detects all 1st order factors of a symbolic.

If exact solving fails, `erable` calls the numeric solver for univariate polynomials (which is the HP48GX `PROOT` function in `erableg.lib` or the Bairstow or Laguerre algorithm in `erable.lib`) and tries to find second order polynomial with integer coefficients by coupling 2 approximate solutions (this was an idea of Mika Heiskanen implemented in `POLYLIB`). Hence `erable` should find all rationals and quadratic irrationals roots of a univariate polynomial (unless the polynomial is bad conditionned).

For multivariate polynomials, the two first steps are achieved (`EVIDENText` and `SQFFext`). `erable` should find all rational multivariate roots of a polynomial (1st order factors). Unfortunately, `erable` does not implement the exhaustiv search of all 2nd order (or greater) multivariate rationals factors. This can be performed using the `FCTR` function of the `ALG48` library.

Abstract of the Sysrpl `XLIB`s (include `erextdec.h` and `erhash.h` in your source code to use them) to factorize:

- `EVIDENText`: finds rational roots
- `SQFFext`: finds square-free factorization of a polynomial

- `SOLVext`: roots a univariate polynomial numerically and tries to rebuild quadratic irrationals roots

Abstract of the users commands to factorize:

- `COLC`:
  for symbolic input calls `SQFFext`, then `EVIDENText`, does **not** call `SOLVext`,
  for list input calls only `SQFFext`
- `FROOT`: calls `SQFFext` then `EVIDENText` then `SOLVext`,
- `FACTO`: calls `SQFFext` then `EVIDENText`, does not call `SOLVext`

Flags 12 and 14 may be cleared to skip respectively `SQFFext` and `EVIDENText`.

8 Derivation and integration.

The `erable` derivation instruction is `der`, it computes the derivative of a (list of) function(s) like the built-in instruction but does not evaluate numeric expressions (like $\sqrt{2}$ or $\frac{1}{2}$). If level 1 is a list, `der` returns the gradient of level 2:

```
2: 'X^2+2*X*LN(Y)-1/Y', 1: { X Y }
  ->  { '2*X+2*LN(Y)' '2*X*(1/Y)+1/Y^2' }
```

`der` returns `d`j`Z(X,Y,...)` for the derivative of the user-defined function `Z(X,Y,...)` with respect to the j-th variable of $z(x, y, ...)$. *Example:*
Suppose that $x \to z(x)$ is the primitive of $\sqrt{x^3-1}$. Type `'Z(X)' X der`, you get $\partial_1 z(x)$ on the stack. Enter $\sqrt{x^3-1}$ and hit `=` then enter `DEFINE`. Now, you can type `'Z(X^2)' X der EVAL` and get $2x\sqrt{(x^2)3-1}$. Now try `'Z(X,X^2)' X der`.

The main integration command is `RISCH`. The `LIN` command may be used to linearize trigonometric expressions, it is called by `RISCH` if polynomials of $\sin(x)$ and $\cos(x)$ are integrated. The `RISCH` program accepts functions or integrals as input and (tries to) return the primitive or the evaluated primitive.

Some examples for `RISCH`:

- $\frac{1}{x^2-4} \to \frac{1}{4}\ln(x-2) - \frac{1}{4}\ln(x+2)$
- $x\ln(x) \to \frac{1}{2}x^2\ln(x) - \frac{1}{4}x^2$
- $\sqrt{x^2-1} \to \frac{1}{2}\ln(-x+\sqrt{x^2-1}) + \frac{x}{2}\sqrt{x^2-1}$:
- $1/(\sin(x)+2) \to \frac{-2}{3}\sqrt{3}\arctan(\frac{-2\tan(x/2)-1}{3}\sqrt{3})$

The `RISCH` program should be used in conjonction with the `TSIMP` function to get "weak normalization". It is a partial implementation of the Risch algorithm: it works with pure transcendental extensions (i.e. square root are generically not allowed), and exponential polynomial part must not contain logarithms or other exponentials. Examples:

$$\ln\ln(x), \quad \frac{1}{e^{x^2+1}-1}, \quad x^3 e^{\left(\frac{x+1}{x+2}\right)}$$

are allowed (and returned since they do not have a antiderivative which may be expressed with elementary functions), but:

$$\sqrt{\ln(x)^2-1}, \quad e^{\ln(x)^2+1}$$

are not allowed as input. Trigonometric expressions are converted in exponentials or logarithms (if linearisation is not the direct solution). Fractions of the type $F(x, \sqrt{ax^2+bx+c})$ are detected.

In conjonction with `TSIMP` and `EXPA`, you should obtain some non trivial results: e.g.

$$\int_1^2 \frac{1}{x^3+1} = \frac{\ln(3)-2\ln(2)}{6} + \frac{\pi}{18}\sqrt{3}$$

(try `RISCH` followed by `TSIMP` and `EXPA`).

9 Ordinary differential equations.

9.1 Linear differential equations (systems) with constant coefficients.

The most efficient tool for these equations is the Laplace transform defined by:

$$L(y)(s) = \int_0^\infty e^{-st} y(t)\, dt$$

Example: solve $y' + 2y = \cos(x)$. Apply L, since:

$$L(y')(s) = sL(y)(s) - y(0)$$

we get:

$$(s+2)L(y) = L(\cos(x))(s) + y(0)$$

hence:

$$y(t) = L^{-1}\left(\frac{\frac{s}{s^2+1} + y(0)}{s+2}\right) = L^{-1}\left(\frac{\frac{s}{s^2+1}}{s+2}\right) + y(0)L^{-1}(\frac{1}{s+2})$$

since $L(\cos(x))(s) = s/(s^2+1)$.

9.1.1 Laplace transform.

The program `LAP` takes 2 arguments: a function f at level 2 and a divisor g at level 1 (usually the characteristic equation of the linear o.d.e.) and returns $L(f)/g$ (Laplace transform is performed with respect to the variable contained in `VX`). If you need the Laplace transform alone, type 1 for g. For the above example, you would type $\cos(x)$ and $x+2$ then `LAP`.

You may wonder why I added the g function? The reason is that inverse Laplace transform of rational fractions is obtained by partial fraction decomposition, and it is easier to decompose expressions with subexpressions separated by $+$ or $-$, hence it is faster to divide by the characteristic equation each term of the Laplace transform of f.

9.1.2 Inverse laplace transform.

The program `ILAP` performs inverse Laplace transform of rational fractions.
Example: `'X/(X^2+1)' 'X+2' / ILAP` and you get the answer:

$$y(t) = \frac{1}{5}(2\cos(x) + \sin(x)) + \frac{-2}{5}e^{-2x}$$

Important remark: The name of the Laplace variable is the same name as the normal variable (and is contained in `VX`).

9.1.3 Linear differential equations systems with constant coefficients.

Example: suppose we want to solve the following system:

$$\left\{\begin{array}{lllllll} y_1'(x) & = & y_1(x) & - & y_2(x) & + & 1 \\ y_2'(x) & = & 2y_1(x) & + & 4y_2(x) & + & e^x \end{array}\right.$$

with initial data $y_1(0) = y_2(0) = 0$.

For scalar linear differential equations with constant coefficients, it is a well-known procedure to use Laplace transform, e.g. to solve

$$ay'' + by' + cy = f(x)$$

one would perform the L-transform and get:

$$(ap^2 + bp + c)L(y) = L(f)(p) + a(pf(0) + f'(0)) + bf(0)$$

where $f(0)$ and $f'(0)$ are the initial consitions at $x = 0$. If $L(f)$ is a rational fraction, `ILAP` allows you to recover y by inverse Laplace transform of

$$\frac{L(f)(p) + a(pf(0) + f'(0)) + bf(0)}{ap^2 + bp + c}$$

But this method may be applied to systems of linear differential equations, the aim of `LDEC` is to help you solving 1st order systems (note that higher order systems may always be rewritten as 1st order systems). Let $y = (y_1, ..., y_n)$ be a vector of functions of x and suppose that we want to solve:

$$y' = Ay + b$$

where A is a $n \times n$ constant matrix and b a vector of n functions of x. Denote by $L(b)$ the vector of the n Laplace transform of the n functions of b , and by $y(0)$ the vector of n initial data at $x = 0$. Then

$$(pId - A)L(y) = L(b) + y(0)$$

and:

$$L(y) = (pId - A)^{-1}(L(b) + y(0))$$

The purpose of `LDEC` is to compute

$$(pId - A)^{-1}(L(b) + y(0))$$

given A and $L(b) + y(0)$. The stack should be prepared as:
`stk2:` A,
`stk1:` $L(b) + y(0)$
then type `LDEC` and you will get at stack level 1:

$$(pId - A)^{-1}(L(b) + y(0))$$

Stack level 3 is the comatrix, stack level 2 is the determinant of A.

Back to the example above: The matrix A is then:
`[ [ 1 -1 ] [ 2  4 ] ]`
and since Laplace(1)=$1/p$ and Laplace(e^x)=$1/(p-1)$, we put $L(b) + y(0)$ on the stack:
`{ '1/X' '1/(X-1)' }`
(here `VX` is set as usual to `'X'`). After calling `LDEC`, we get:
`{ '(X^2-6*X+4)/(X^2-X)/((X-3)*(X-2))' '(X+2)/X/((X-3)*(X-2))' }`
which is the { Laplace(y_1) Laplace(y_2) }. Now to recover y_1 or/and y_2 just hit `EVAL` and `ILAP` for each coordinate. You'll get:

$$\left\{ \begin{array}{lcc} y_1 & = & -1/2e^x - 2/3 + 2e^{2x} - 5/6e^{3x} \\ y_2 & = & 1/3 - 2e^{2x} + 5/3e^{3x} \end{array} \right.$$

To compute the solution of the `LDEC` with initial data $y_1(0) = 1, y_2(0) = 2$ just replace `{ '1/X' '1/(X-1)' }` by `{ '1/X+1' '1/(X-1)+2' }`..

9.2 First order linear equation.

The `DSOLVE` program recognizes and solves the following equations types:

- $y'(x) = f(y(x))$,
- $y'(x) = f(x, y(x))$ with f homogenous,
- $y'(x) = g(x)y(x) + h(x)y(x)^\alpha, \quad \alpha \in \mathbb{R}$ (Bernouilli type)
- $y'(x) = f(x)g(y)$ (separable, if f and g are rationals fractions)
- $y'(x) = f(x)y(x) + g(x)$ (linear)

The input is a symbolic $f(x, y(x))$. Examples:

```
Y(X)^2+Y(X) DSOLVE (incomplete)
X*Y(X)+1-X^2 DSOLVE (linear)
(Y(X)-X)/(Y(X)+X) DSOLVE (homogenous)
Y(X)^2+X*Y(X) DSOLVE (Bernouilli)
```

The output may be y as a function of x or x as function of y or x and y as a function of t (parametric solution) for an homogenous ode.

10 Taylor series.

The program `SERIES` is very similar to the built-in instruction. In addition, Taylor expansion at any point is made easier by the ability to type at level 2 an equation, e.g. `'X=1'` instead of `'X'` if you want to know a Taylor expansion at $x = 1$ instead of at $x = 0$. Moreover, `SERIES` of `erable` can solve some undetermined forms (like the third example)

Examples:

```
* 'EXP(SIN(X))' 'X' 3 SERIES
  -> 3: 0
     2: { 1 1 1/2 0 }
     1: '1+X+1/2*X^2'
* '\v/X' 'X=1' 4 SERIES
  -> 3: 0
     2: { 1 1/2 -1/8 1/16 -5/128 }
     1: '1+1/2*(X-1)-1/8*(X-1)^2+1/16*(X-1)^3-5/128*(X-1)^4'
* 'EXP(1/X*LN(1+X)) 'X' 4 SERIES
  -> 3: 0
     2: { 'e' '-e/2' '11*e/24' }
     1: 'e+-e/2*X+11*e/24*X^2'
```

Note that the last example can not be computed using the inbuild `TAYLR` instruction.

At level 1, you see the Taylor expansion (algebraic form) and at levels 2 and 3 the list form of the Taylor expansion (level 3 is a real: the valuation).

11 Linear algebra.

11.1 Building a matrix

To build a matrix, you may type it as usual with { and } instead of [and] or you may use one of the following instructions:

- `idn`: to build a symbolic identity matrix
- `LCXM`: to build a matrice $A = (a_{ij})_{1 \le i \le l, 1 \le j \le c}$. The program takes 3 arguments: l, c and a program building a_{ij} from i and j. Example: `2 4 << SQ + >> LCXM` returns a 2×4 matrix with $a_{ij} = i + j^2$.
- `VAND` and `HILB` return Vandermonde and Hilbert matrices given respectively a list of objects or an integer.

11.2 Operations

`erable` provides the arithmetic usual operations on matrices and vectors (`add`, `SUBT`, `MULT`, `CHS`) and:

- `STUDMULT`: (`MATR` directory) student multiplication of matrices (term by term)
- `TR`: trace of a matrix

- `TRAN`: transposed of a matrix (true transposed, no conjugation)
- `XY`: scalar produc of two vectors
- `cross`: cross product of two 3-d vectors.

11.3 Gauß -Jordan row reduction.

Summary of the instructions:

- `rref`: row reduction to echelon form. At level 2, the list of pivoting coefficients is given, this is usefull to treat particular cases.
- `RANG`: like `rref` but creates 0 only under the diagonal.
- `det` and `RDET`: determinant (using the $O(n*n!)$ algorithm or row reduction)
- `INVL`: inverse of a matrix using row reduction
- `LU2`: given a square matrix, returns L^{-1} and U s.t. $A = LU$ (i.e. $A =$`stk2`$^{-1}\times$`stk1`) where L and U are lower and upper triangular (maybe w.r.t. to a permutation matrix, this means that computing the inverse of L or U is trivial). For comparison, the built-in `LU` returns three matrices L, U and P s.t. $A = PLU$.
- `SYST` and `SOLGEN`: solution of a linear system.

11.3.1 Solving a linear system.

Suppose you want to find (x, y) s.t.:

$$\left\{ \begin{array}{rcrcr} mx & + & y & = & -2 \\ mx & + & (m-1)y & = & 2 \end{array} \right.$$

where m is a parameter. Enter the following matrix `{ {M 1 -2} { M 'M-1' 2 } }`
and type the `ENTER` key to have a copy, then type `rref`, you get at level 1:
`{ { '-M^2+2*M' 0 '2*M' } { 0 '2-M' -4 } }`.
This means that:

$$(2m - m^2)x = 2m, \qquad (2-m)y = -4.$$

This is the case iff all the coefficients in the list at level 2 are non 0. You should have at level 2:
`{ -1 'M+-2'}`
The second coefficient vanishes if $m = 2$. You have to solve for this particular case again. Recall the original matrix from the stack and type:
`'M=2' EXEC`
This replace all occurences of `M` by 2 in the original matrix. Now type `rref` again, you get:
`{ { 2 1 -2 } { 0 0 4 } }`
The last line means that:

$$0x + 0y = 4$$

which is clearly impossible, the system has no solution.

Another way to solve the system is the `SYST` instruction. Put the `MATRIX` on the stack, type the list of unknown followed by -1 ($+1$ means that the constant row is *before* the $=$ sign, -1 means *after* the $=$ sign):
`{ X Y -1}`
then type `SYST`, you get the solutions (value tagged by the corresponding unknown). At level 2, you get the list of cases for which you need a specific study:
`{ 'M^2-2*M' '-M+2' -1 'M+-2'`
This means that we have to solve again for $m = 0$ and $m = 2$.

For systems, the SOLGEN program provides another way of writing the solution as an affine space of solutions. Recall the matrix on the stack (simply hit SYSTEM), type:
'M=0' EXEC EVAL
and SOLGEN, you get at level 1:
If { }, { X Y }=:{ X -2 }
This means that $(x, -2)$ is solution for every x. The If statement shows necessary conditions for the system to have solutions (here no condition, but if we try $m = 2$ instead of $m = 0$ the system has no solution: the If statement is If { '0=-1'} never fulfilled).

Remarque 2 SYST *and* SOLGEN *order the unknown so that principal unknown are followed by auxiliary unknown. For the $m = 0$ case, the list of unknown returned is* { Y -1 X }*, this means that* Y *is a principal unknown and* X *an auxiliary one.*

11.3.2 Inversion

The INVL implements the Gauß method to invert matrices.

```
{ { '1/2' -1 }
  { 1    '2/3' } }
INVL
-> { { '1/2' '3/4'  }
     { '-3/4' '3/8' } }
```

11.3.3 Determinant

The RDET instruction implements Gauss row reduction to compute determinant.

```
{ { 1 T T T }
  { 1 K T T }
  { 1 T K T }
  { 1 T T K } }
RDET -> '(-T+K)*(-T+K)*(-T+K)'
```

11.3.4 Other examples.

- *LU* decomposition example:

  $$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

  LU2 returns:

  $$l = \begin{pmatrix} 1 & 0 \\ -3 & 1 \end{pmatrix} \qquad U = \begin{pmatrix} 1 & 2 \\ 0 & -2 \end{pmatrix}$$

  We have $A = L^{-1}U$.

- Rank of a matrix:

  $$\begin{pmatrix} 1 & 2 & 4 & 6 \\ -1 & 3 & 5 & 7 \\ 2 & 1 & 0 & 1 \\ 2 & 6 & 9 & 14 \end{pmatrix},$$

  hit RANG, and look at the matrix, the rank is the number of non zero lines (BTW you get a half reduced matrix)

- Linear relations between vectors
  Suppose we want to know the rank and linear relations existing between $v_1(1, 2, 0)$, $v_2(-2, -1, 1)$, $v_3(0, 3, 1) \in I\!R^3$:

```
{ { 1 2 0 V1 }
  { -2 -1 1 V2 }
  { 0 3 1 V3 } }
```

then `RANG`, we get:

```
{ { 1 2 0 V1 }
  { 0 3 1 '2*V1+V2' }
  { 0 0 0 '-(2*V1)-V2+V3' } }
```

The family is of rank 2 (the 3rd line is 0) and $-2v_1 - v_2 + v_3 = 0$.

11.3.5 Stack input/output for reduction instructions.

Program	Input	Output
RANG	1: matrix	2: pivots, 1: redcued matrix
rref	1: matrix	2: pivots, 1: rref-ed matrix
RDET	1: matrix	3: pivots, 2: reduced matrix , 1:determinant
INVL	1: matrix	1: inverse
SYST	2: matrix, 1: list of unknowns	4: list unknowns, 3: list of principals and auxiliary unknowns 2: list of pivots 1: list of tagged algebraics
SOLGEN	2: matrix, 1: list of unknowns	5: pivots, 4: list of unknowns, 3: list of principal and auxiliary unknowns 2: list of tagged algebraics, 1: result

11.4 Diagonalisation

The diagonalisation instructions are:

- `MAD`: given a square matrix, returns determinant, formal inverse, a list polynomial (cf. infra) and the characteristic polynomial.
- `PCAR`: characteristic polynomial using `det`
- `JORDAN`: compute eigenvalues and eigenvectors (cf. infra)

Given a square matrix A, `JORDAN` returns 7 levels:

- 7: $\det(A)^{-1}$
- 6: A^{-1}
- 5: a list polynomial $P(A)$ with matrix coefficient defined by the relation

  $$(xI_n - A)P(A) = M(x)I_n = M(x)I_n - M(A)$$

  where M denotes the minimal polynomial of A. (this is the list polynomial returned by `MAD` if characteristic and minimal polynomial coincides)
- 4: list of eigenvalues (with multiplicities)
- 3: characteristic polynomial
- 2: minimal polynomial M (it divides the characteristic polynomial)
- 1: list of characteristic spaces tagged by the corresponding eigenvalue (either a vector or a list of Jordan chains, each of them ending by a "Eigen:"-tagged eigenvector)

Examples:

1.

$$A = \begin{pmatrix} 1 & -1 & 0 \\ 0 & 1 & -1 \\ -1 & 0 & 1 \end{pmatrix}$$

returns:

```
7: 0
6: { { '1/0' '1/0' '1/0' }
     { '1/0' '1/0' '1/0' }
     { '1/0' '1/0' '1/0' } }
5: { {{1 0 0}  {{-2 -1 0}  {{1 1 1}
      {0 1 0}   {0 -2 -1}   {1 1 1}
      {0 0 1}   {-1 0 -2}}  {1 1 1}} }
4: {0 1 '3/2+i/2*V3' 1 '3/2-i/2*V3' 1 }
3: 'X^3-3*X^2+3*X'
2: 'X^3-3*X^2+3*X'
1: {  :0: {1 1 1}
:'3/2+i/2*V3': {1 '-1/2-i/2*V3' '-1/2+i/2*V3'}
      :'3/2-i/2*V3': {1 '-1/2+i/2*V3' '-1/2-i/2*V3'} }
```

This means that A has 3 eigenvalues $\frac{3 \pm \sqrt{3}i}{2}$, and a basis of eigenvectors is:

$$\{(1,1,1),(1,\frac{-1 \mp i\sqrt{3}}{2},\frac{-1 \pm i\sqrt{3}}{2})\}$$

corresponding to $0,(3+\sqrt{3}i)/2,(3-\sqrt{3}i)/2$. The characteristic and minimal polynomial are identical (this is generically the case) $X^3 - 3X^2 + 3X$. The matrix is not invertible (`'1/0'` is infinite) and has a 0 determinant.

2. For the identity matrix I_2, we get:

```
7: 1
6: { { 1 0 } { 0 1 } }
5: { {{1 0}
      {0 1}} }
4: {1 2}
3: 'X^2-2*X+1'
2: 'X-1'
1: { :1, Eigen: { 0 1 } :1, Eigen: { 1 0 } }
```

The minimal polynomial is $X-1$, different form the characteristic polynomial $(X-1)^2 = X^2 - 2X + 1$.

3.

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

4. A formal example:

```
{ { 1 A }
  { A 1 } }
```

5. In dimension greater than 2, the factorisation routines may fail. For this reason, you may call `MAD`, factor the characteristic polynomial (e.g. by trying the `FCTR` instruction of `ALG48`) before calling `JORDAN`. If you have `ALG48` installed, try this:

```
{ { 1 1 A }
  { 1 A 1 }
  { A 1 1 } }
MAD FCTR JORDAN
```

   Note that this example is solved by typing `JORDAN` directly but it may fail in other situations.

6. Jordan cycles example:

$$A = \begin{pmatrix} 3 & -1 & 1 \\ 2 & 0 & 1 \\ 1 & -1 & 2 \end{pmatrix},$$

   returns:

```
7: -4
6: : { { '1/4' '1/4' '-1/4' }
       { '-3/4' '5/4' '-1/4' }
       { '-1/2' '1/2' '1/2'  } }
5: { [[ 1 0 0]  [[ -2 -1  1]  [[ 1 1 -1]
     [ 0 1 0]   [  2 -5  1]   [-3 5 -1]
     [ 0 0 1]]  [  1 -1 -3]]  [-2 2  2]] }
4: { 2 2 1 1}
3: 'X^3-5*X^2+8*X-4'
2: 'X^3-5*X^2+8*X-4'
1: { :2, Char: { 2 2 1 }  :2, Eigen:{ 1 1 0 }  :1: { 0 1 1 } }
```

   This means that 2 has multiplicity 2, but the corresponding eigenspace is only 1-dimensional (spanned by $(1,1,0)$ the last vector of the Jordan chain). The first vector $(2,2,1)$ is only a characteristic vector, his image by $(A-2I)$ is the eigenvector $(1,1,0)$.

Remarque 3 *If flag 12 is set (normal state), then eigenvectors and characteristic vectors are divided by their common greatest divisor. In case of Jordan chains, this means that the image of the i-th vector of the chain by $A-\lambda I_n$ is not necesseraly equal (but always colinear) to the $i+1$-th vector of the chain.*

11.5 The `MMULT` instruction.

This multiplication takes 3 arguments: 2 objects at levels 3 and 2, and a real at level 1: the product type:

- 0: matrix, matrix
- 1: matrix, vector
- 2: matrix, scalar,
- 3: vector, scalar
- 6: scalar, matrix
- 7: scalar, vector

It is useless in interactive mode (if you plan to write your own program over `erable`, you may need `MMULT` to switch to internal mode data representation for speed).

12 Quadratic forms.

The main program is GAUSS to perform reduction of a quadratic form q. There are two ways to use GAUSS:

- symbolic input:
  Input: a quadratic form q (symbolic) at level 1 or the quadratic form q at level 2 and the list of variables at level 1.
  Output:

  - stk5: D the list of diagonal coefficients (only the number of positive and negative coefficients is characteristic of q)
  - stk4: P (the columns vectors of P^-1 form a q-orthogonal basis of A at level 3)
  - stk3: A (A is the matrix of q in the dual base of the coordinates-forms at level 2, we have $A = P^t DP$ where P^t denotes the transposed of P)
  - stk2: list of variables
  - stk1: symbolic as a sum of independent squares

  Examples:

```
Example 1:
'X^2+4*X*Y-2*X*Z+4*Y^2+6*Y*Z+7*Z^2' GAUSS
5: { 1 '-25/6' '1/6' }
4: { { 1 2 -1 } { 0 1 0 } { 0 5 6 } }
3: { { 1 2 -1 } { 2 4 3 } { -1 3 7 } }
2: { X Y Z }
1: '1/6*(6*Z+5*Y)^2+ -25/6*Y^2+(-Z+2*Y+X)^2'

Example 2: same example but with variable in the reverse order
'X^2+4*X*Y-2*X*Z+4*Y^2+6*Y*Z+7*Z^2' { Z Y X } GAUSS
5: { '1/7' '7/19' '-25/19' }
4: { { 7 3 -1 } { 0 '19/7' '17/7' } { 0 0 1 } }
3: { { 7 3 -1 } { 3 4 2 } { -1 2 1 } }
2: { Z Y X }
1: '-25/19*X^2+7/19*(17/7*X+19/7*Y)^2+1/7*(-X+3*Y+7*Z)^2

Example 3: if you want to orthogonalize with parameter, you need to
enter the list of variables of the quadratic form
'X^2+2*A*X*Y' { Y X } GAUSS
5: { '-A^2' 1 }
4: { { 1 0 } { A 1 } }
3: { { 0 A } { A 1 } }
2: { Y X }
1: '(X+A*Y)^2-A^2*Y^2'
```

- matrix input:
  Input (stack level 1): the formal matrix A of the quadratic form q
  Output: at stack level 2 D the diagonal coefficients list and at stack level 1 the transition matrix P. We have $A = P^t DP$ where P^t denotes the transposed of P. Note that to obtain a q-orthogonal basis, one can take the columns of the inverse P^{-1} of P).

  Example:
  The matrix of q defined by $q(x, y) = 4x^2 + 2xy - 3y^2$ is:

$$A = \begin{pmatrix} 4 & 1 \\ 1 & -3 \end{pmatrix},$$

(to get the matrix of q, enter `'4*X^2+2*X*Y-3*Y^2'`, then the list of variables `{ X Y}` and hit `QXA`). Call `GAUSS` which returns:

```
2: { '1/4' '-13/4' }
1: { { 4 1 } { 0 1 } }
```

this means that:

$$A = \begin{pmatrix} 4 & 0 \\ 1 & 1 \end{pmatrix} \times \begin{pmatrix} '1/4' & 0 \\ 0 & '-13/4' \end{pmatrix} \times \begin{pmatrix} 4 & 1 \\ 0 & 1 \end{pmatrix}.$$

This means that:

$$q(x,y) = 4x^2 + 2xy - 3y^2 = \frac{1}{4}(4x+y)^2 - \frac{13}{4}y^2.$$

The other utilities are `QXA` and `AXQ` to switch from algebraic to matricial representation of a quadratic form (quadratic as symbolic to array). `QXA` accepts an optional list of variables at level 1.

Remarque 4 *If you want to save time, use numeric mode (instruction* `XNUM`*)!*

13 Arithmetic.

You may force integer arithmetic by setting flag 10. Otherwise, polynomial arithmetic is assumed. This is important for instructions like `GCD3` or `ABCUV`.

- `GCD1`:
  returns the greatest common divisor d of two objects a and b (integers, Gauß integers, polynomials). Examples:
  `'X^2+2*X+1' 'X^2+3*X+2' GCD1 -> 'X+1'`
  `25 15 GCD1 -> 5`
  If flag 12 is clear returns 1.

- `LCM1`: lowest common multiple ($\texttt{GCD1}(a,b) \times \texttt{LCM1}(a,b) = a \times b$). Examples:
  `'X^2+2*X+1' 'X^2+3*X+2' LCM1 -> '(X^2+2*X+1)*(X+2)'`
  `25 15 LCM1 -> 75`

- `GCD3`: extended gcd algorithm, given x and y returns d , u and v s.t.:

  $$ux + vy = d$$

  (d is a multiple of the `GCD` of x and y by an invertible, i.e. an integer in the univariate case)

- `ABCUV`: (Bezout identity)
  solve the equation c=ax+by *Examples:*

  ```
  'X^2+2*X+1' 'X^2+2*X+3' 1 -> -1 1 1
  'X^2+2*X+1' 'X^2+2*X+3' 1 -> 0
  ```

  This means for the first case that:

  $$(X+1) = -(X^2+2X+1) * (-1) + (X^2+3X+2) * 1$$

  as in the second case there is no solution because the *gcd* of $x^2 + 1 * x + 1$ and $x^2 + 3 * x + 2$ does not divide 1.

- `LGCD`: returns the *gcd* of a list of objects.

- `SIMP2`: simplifies two objects by dividing them by their GCD. Sets flags 12, 14 and 15. Ex:
  `stk2: 9, stk1: 6 SIMP2 -> stk2: 3, stk1: 2`

- DIVIS: gives a list of divisors of an object. Example:
  `21 -> { 1 7 3 21 }`
- fact and comb: like the built-in FACT and COMB instructions but for long integers.
- EULER: Euler indicatrix
  Given an integer n, returns an integer e: the number of integers lower than and prime with n. Example for $n = 25$, $e = 20$ because 1,2,3,4,6,7,8,9,11,12,13,14,16,17,18,19,21,22,23,24 are prime together with 25.
- XFRC:
  Same as →Q but handles quadratic irrationals (recognize a quadratic irrational if its expansion in contined fraction is ultimately periodic of period less or equal to 3) *Examples:*

$$1.20710678118 \rightarrow \frac{1}{2} + \frac{1}{2}\sqrt{2} \quad 1.5 \rightarrow \frac{3}{2}$$

- ORND:
  round object at stack level 2, stack level 1 is the expected denominator of all rationals of the object. For example, $.49999999999 + .2000000001 * x$ 10 ORND returns $.5 + .2 * x$.
- re: real part of a fraction, global names are always considered as reals
- im: imaginary part of a fraction,
- conj: conjugate of a fraction,

14 Other utilities.

14.1 Data representation.

- AXL: array ↔ list conversion.
- SXL:
  - VX variable-fraction representation conversion. Switches from algebraic to list-polynomials or fractions. Ex:
    `'(X+1)/(3*X-2)' <--> '{1 1}/{3 -2}'` (displayed as 'UNKNOWN/UNKNOWN')
    `'X+3' <--> {1 3}`
  - General stack object conversion. Ex:

```
'X+3*SIN(X)'
{ { 1 '5*X' } { 'SIN(X)' 1 } }
{ 'X^2+7*X' '3*SIN(X)' }
#3h  SXL ->
{ 'SIN(X)' X }
{ 3 { 1 0 } }
{ { 1 { { 5 0 } } } { { 1 0 } 0 } }
{ { { 1 7 0 } } { 3 0 } }
```

    To go back, type `{ #0 #1 #2 } SXL`

The SXL may be used in programs for speed since you may spare user-internal conversions. Write and test your program in user mode, then add SXL at the beginning and at the end of your program. Warning: some instructions of erable can only handle user data (e.g. RISCH), but most of the basic instruction may be used in internal mode.

- `S2L`: convert an algebraic polynomial in a list polynomial. Ex:
`'1+2*A' A S2L -> { 2 1 }` Accepts lists. Ex:
`{ '1+A' '2*A-3' } A S2L -> { { 1 1 } { 2 -3} }`

- `L2S`: converts back a list polynomial to an algebraic. `L2S` may be used for multiple variable polynomial evaluation. Ex:
`{ { 1 2 3} {4 5 6} } { X Y } L2S  ->  '(Y^2+2*Y+3)*X+(4*Y^2+5*Y+6)'`

- `EPSX0`: strip leadings zeros in list-polynomials, replace objects by 0 if their absolute value is less than `EPS`.

- `FXND`: splits a fraction in numerator (stack 2) and denominator (stack 1). Ex:
`'(X+1)/A' FXND -> 2:X+1, 1:A`
Works as well for fractions of lists polynomials.

- `NDXF`: reverse of `FXND`. Ex:
`1 2 NDXF -> '1/2'`
Works for all data types (you can get strange symbolics with `NDXF`).

- `XNUM`: like the in-build →`NUM`, but accepts lists (this was not the case on S/SX models). Clears flags 12, 14 and 15.

- `XQ`: like the in-build →`Q`. Sets flags 12, 14 and 15.

- `idn`: like the built-in `IDN` but returns a symbolic identity matrix.

14.2 Polynomials

- `HORN` executes an Horner scheme. The syntax is:

```
stk2: P
stk1: r
-> 3: P div (X-r), 2: r, 1: P(r)
Ex: 'X^2+2*X+3', 5 -> 'X+7', 5, 38
```

  This means that $X^2 + 2 * X + 3 = (X + 7)(X - 5) + 38$.

- `PTAYL`: Taylor development (for polynomials only):
`2: P(X), 1: r -> P(X-r)`
Example:
`'X^3+2*X' 2 PTAYL -> 'X^3+6*X^2+14*X+12'`
means that $X^3 + 2X = (X - 2)^3 + 6(X - 2)^2 + 14(X - 2) + 12$

- `LEGENDRE` and `TCHEB`: given an integer n, returns a list of $n+1$ polynomials, respectively Legendre and Tchebycheff.

- `COSN`: compute $\cos(nx)$ and $\sin(nx)$ as polynomial of $\cos(x)$ and $\sin(x)$.
Input: n (a real integer)
Output: 2: $\sin(nx)$, 1: $\cos(nx)$ as polynomials of c ($\cos(x)$) and s ($\sin(x)$)
Remark that $\cosh(nx)$ and $\sinh(nx)$ as polynomials of $\cosh(x)$ and $\sinh(x)$ have the same expression.
Example:

$$3 \to (-1 + 4c^2)s \quad - 3c + 4c^3$$

  This means that $\sin(3x) = (-1 + 4\cos(x)^2)\sin(x)$ and $\cos(3x) = -3 * \cos(x) + 4 * \cos(x)^3$.

14.3 Permutations

A permutation is represented as a list of images of $[1..n]$ e.g. { 5 1 2 4 3 } means $\sigma(1) = 5$, $\sigma(2) = 1$, $\sigma(3) = 2$, $\sigma(4) = 4$ and $\sigma(5) = 3$. The `P2C` instruction converts this representation to the cycle decomposition, here { { 1 5 3 2 } { 4 } } (stack level 2) and computes the signature of p (stack level 1). `C2P` converts back cycle decomposition to usual representation of permutations. `CIRC` compose 2 permutations in the usual representation.

14.4 Variables

- `LVAR`: returns the list of "variables" of an algebraic. The list is sorted by reverse alphabetic order. Example:
  `'SIN(A)+B*X+1' -> { X B 'SIN(A)' }`
- `LIDNT`: list of global names of an algebraic Example:
  `'SIN(A)+B*X+1' -> { X B A }`

14.5 `ALGB(G)` directory.

This directory contains the following programs:

- `tEVAL`: evaluate object 1 and returns the time it tooks to evaluate it. Not as accurate as `TIM` of the hacker library.
- `LATEX`: converts a symbolic to a string, the LaTeX traduction of the symbolic. To `tex` it on a computer, you must include the string in a math. environment (in `$ $` or in `\[ \]` or in an `equation` environment, and you must include the file `hp48.tex`).

15 Final remarks.

Remaining things to do:

- implement the `LIMIT` instruction for limits and improve the `SERIES` instruction,
- rewrite the Gauß reduction to include the Bareiss method,
- rewrite the extended GCD algorithm with the subresultant method,
- extend the Risch algorithm to multiples exponentials, and return an unevaluated integrals when there is no closed form,
- improve the factorization algorithm.

I wil probably build a contrib directory (or library) if I have sufficient material to do it, your RPL and SysRPL programs are welcomed. Otherwise, I would say that `erable` is almost finished since I have probably reached the speed limit! (`ALG48` will perhaps have in the future all the functionnalities of `erable`, but I don't expect a speed factor greater than 4, here we would need a FASTER CPU but is there a successor to the HP48?). Anyway, it is certainly sufficient to help maths students in their studies, which was my first aim.

I will probably switch to another project like working on C(++) on a fast CPU (it would be interesting to have a sysrpl implementation written in C(++), I would only have to rebuild a kernel.lib to have a competitive `erable`!).

A All functions of `erable` listed in alphabetic order

The following symbols will be used:

- %: real
- C%: complex
- n: integer (real integer)
- [ ]: numeric array
- { l }: list
- { m }: symbolic array
- p: polynomial ({ p } for a list-polynomial),
- { v }: list of variables
- s: symbolic object
- v: variable (global name or irrational symbolic)
- f: a fraction
- N, D: numerator and denominator of a fraction
- o: object

List of all global variables and programs in `HOME`, `ALGB` or `ALGBG`, or in subdirectories:

Name	Function	Arguments	Returns
CST	Cst Menu	nothing	{ l }
fr	French short doc	nothing	string
GETALL	Get All Modules	nothing	nothing
INIT	Initialization	nothing	nothing
INVLAP	Last inverse Laplace	nothing	s
LATEX	LaTeX conversion	1: s	1: string
MATRIX	Last matrix	nothing	m
PRIMIT	Last primitive	nothing	s
PURG	Purge erable	nothing	nothing
SCST	CST-Menu string	nothing	string
SETFR	Set French Flags	nothing	nothing
SYSTEM	Last system	nothing	$\{mv\}$
tEVAL	Execution time	..., 1: o	EVAL(o), 1: time
UKEYS	User keys string	nothing	string
us	English short doc	nothing	string
VX	integration variable	Rien	v

The 90 functions of the `erable` library:

Name	Function	Arguments	Returns
ABCUV	Bezout $ax+by=c$	3,2,1:a, b, c	1:1 [3,2: x, y] or 1: 0
AXL	array $\leftrightarrow$ list	[ ] ou { m }	{ m } or [ ]
AXQ	array to s quadratic form	{ m }	s
		2: { m }, 1: { v }	s
C2P	Cycles $\rightarrow$ permutation	{ cycles }	p
CHS	Change signe	o	$-o$
COLC	Factorization	s	s
COSN	$\cos, \sin(nx) \rightarrow P(\cos x, \sin x)$	$n > 0$	2: s, 1: s
		$n < 0$	2: { p }, 1: { p }
CIRC	Compose 2 permutations	2:p_2, 1:p_1	$p_2 \circ p_1$
DEGRE	Order	{ p }	n
DIV1	Usual division	2: o_2, 1: o_1	o_2/o_1
DIV2	Euclidean division	2: o_2, 1: o_1	2: o_2 div o_1,1: o_2 mod o_1
DIVIS	List of divisors	o	{ l }
DSOLVE	Solve $y'(x) = f(y(x), x)$	$f(y(x), x)$	$y(x)$
EPSX0	Strip expression	o	o
EULER	Euler indicatrix	n	$\varphi(n)$
EXEC	Substitution or *doall*	2: { l }, 1: program	1: { l }
		2: s, 1: $o_1 = o_2$	s
		3: s, 2: { l1 } 1: { l2 }	s
EXPA	Simplification	o	o'
EXPLN	Conversion en $\exp, \ln$	s	s
FACTO	Factorization	o	3: { v } 2: f , 1: { $f_1 n_1 f_2 n_2$ ... }
FCOEF	roots/poles $\rightarrow$ Fraction	{ $r_1 n_1 r_2 n_2$ ... }	f
FROOTS	Factorisation	o	3: { v } 2: f , 1: { $s_1 n_1 s_2 n_2$ ... }
FXND	Split a fraction	$f = N/D$	2: N, 1: D
GAUSS	Gauß quadratic form reduction	1: A	2: D, 1: P
		s	5: D,4: P, 3: A, 2: { v },1: s
		2: s, 1: { v }	5: D,4: P, 3: A, 2: { v },1: s
GCD1	Greatest common divisor	2: o_2, 1: o_1	GCD(o_2,o_1)
GCD3	GCD (solves $au + bv = d$)	2,1: a , b	GCD$(a, b) = d$, u, v
HILB	Hilbert matrix	integer r	$r \times r$ matrix
HORN	Horner scheme	2:p , 1: r	3: $p/(X-r)$, 2: r, 1: $P(r)$
ILAP	Inverse laplace transform	s	$L^{-1}(s)$
INVL	Inversion	o	o^{-1}
JORDAN	Diagonalisation	endomorphism	7 à 1: *cf.* section 11
LAP	Laplace transform	2:f, 1:g	$L(f)/g$
L2S	Evaluation	2: { p }, 1:v	$p(v)$
		2: { p },1: { v }	$p(v)$
LCM1	Least common multiple	2: o_2, 1: o_1	LCM(o_2,o_1)
LCXM	Matrix creation	3: r, 2: c, 1: prog	1: $r \times c$ matrix
LDEC	Linear Diff. Equ. Systems	2: { m }, 1: { v }	3,2: $(m-x)^{-1}$, 1: $(m-x)^{-1}v$
LEGENDRE	Polynomials	integer r	list of $r+1$ polynomials
LGCD	GCD of a list	{ l }	o=GCD
LIDNT	List of variables	s	2: s, 1: { v }
LIMIT	To be implemented!		
LIN	Linearization	C%, s ou { p }	s
LU2	LU decomposition	M	$L^{-}1, U$
LVAR	list of variables	o	{ v }
MAD	inverse, char. polyn., etc.	o	4: det, 3: $1/o$, 2: { p },1: { p }
MMULT	special product	3: o_2, o_1, n	"$o_2 \times o_1$"
MULT	product	2: o_2, o_1	$o_2 \times o_1$
NDXF	create a fraction	2: N, 1: D	$f = N/D$
ORND	Round an object	2: o, 1: D	o

P2C	Permutation $\to$ cycles	p	3: p, 2: cycles, 1: signature
PCAR	Characteristic polynomial	endomorphism	s
PF	Partial fraction	f	$\sum_i f_i$
PFCOLC	`COLC` between + and −	$\sum_i f_i$	$\sum_i f_i$
PFEXPA	`EXPA` between + and −	$\sum_i f_i$	$\sum_i f_i$
POWER	integral power	2: o, 1: n	o^n
PREVAL	Evaluation	3: primitive, 2,1:bornes	s
PTAYL	Taylor for polynomials	2: $P(X)$, 1: o	$P(X-o)$
QXA	s quadratic form to array	2: s, 1: $\{ v \}$	$\{ m \}$
		s	$\{ m \}$
RANG	Réduction sous-diagonale	$\{ m \}$	2: spec. cases, 1:$\{ m \}$
RDET	Determinant (rref)	endomorphism $\{ m \}$	2: $\{ m \}$, 1: determinant
RED	Gauß reduction	matrix or system	*cf.* section 11
RISCH	Symbolic integration	s	s
S2L	Symbolic to list	2: o, 1: $\{ v \}$	2: $\{ v \}$,1:$\{ p \}$
		2: o, 1: v	$\{ p \}$
SERIES	Taylor	3: s, 2: v, 1: n	3: n, 2: $\{ p \}$, 1: s
SIMP2	Simplification	2: o_2, 1: o_1	2: o_2', 1: o_1'
SOLGEN	Solves a linear system	2:$\{ m \}$, 1: $\{ v \}$	*cf.* section 11
SQRT	Square root	n or C% or s	n or C% or s
STUDMULT	"students" $\times$ of matrices	M, M'	"$M\dot{M}'$ "
SUBT	Substraction	2: o_2, 1: o_1	$o_2 - o_1$
SXL	Conversion	Internal [user]	User [internal]
SYST	Solves a linear system	2:$\{ m \}$, 1: $\{ v \}$	*cf.* section 11
TCHEB	Polynomials	integer r	list of $r+1$ polynomials
TR	trace	[ ] or $\{ m \} = (a_{ij})_{1\leq i,j\leq n}$	$\sum_{i=1}^n a_{ii}$
TRAN	transposed	[ ] or $\{ m \}$	[ ] ou $\{ m \}$
TRIG	Trigonometry: $\to \sin, \cos, \arctan$	s	s
TSIMP	Simplification (transcendental)	s	s
VAND	Vandermonde matrix	list of objects	matrix
VER	Version	rien	% 2.93
XFRC	To quadratic irrational	o	o
XNUM	$\to$ Numeric	o	o
XQ	$\to$ Rational	o	o
XY	Scalar product of 2 vectors	2: x 1: y	$x.y$
add	Addition	2: o_2, 1: o_1	$o_2 + o_1$
comb	Combinaisons	2: n, 1: n'	$C_n^{n'}$
conj	Conjugate	o	$\overline{o}$
cross	Wedge product	2: x, 1: y	$x \wedge y$
der	derivative	2: s, 1: v	1: s
det	Determinant (expand)	endomorphism	determinant
fact	Factorielle	n	$n!$
idn	identity	real integer or matrix	identity matrix
im	imaginary part	o	$\Im(o)$
re	real part	o	$\Re(o)$
rref	Row reduction	M	$\{ s \}$, reduced matrix

Functions of the `kernel` library (see `ALG48` documentation too):

Name	Function	Arguments	Returns
`{kernel.lib}`	(0:788)		
MOD+	Modular addition	3: n_1, 2:n_2, 1: n	$(n_1 + n_2)$ mod n
MOD-	Modular substraction	3: n_1, 2:n_2, 1: n	$(n_1 - n_2)$ mod n
MOD*	Modular multiplicatin	3: n_1, 2:n_2, 1: n	$(n_1 * n_2)$ mod n
MOD/	Modular division	3: n_1, 2:n_2, 1: n	(n_1/n_2) mod n
MODPOW	Modular power	3: n_1, 2:n_2, 1: n	$n_1^{n_2}$ mod n
MODINV	Modular inversion	2: n_1, 1: n	n_1^{-1} mod n
PRIM?	Primality test	1: n	1: 1 or 0
PRIM+	Next prime	1: n	1: n'
PRIM-	Previous prime	1: n	1: n'

B User Keys.

α - Right Shift -ed keys:

Princ. Key	Second. Key	Function
EVAL		L2S
	R (recursive)	sets/clears flag 21
SIN	Derivative	der
COS	Integration	RISCH
$\sqrt{}$	$\sqrt{}$	SQRT
y^x	y^x	POWER
$1/x$	$1/x$	INVL
$\pm$	pm	CHS
7	SOLVE	FROOTS
8	PLOT	EXPLN
9	ALGEBRA	EXPA
Division	Division	DIV1
4	TIME	TIM
5		EXEC
6		COLC
$\times$		MULT
1	(complex flag)	clears/sets flag 13
2		AXL
3	UNITS	TSIMP
-		SUBT
.		TRIG
SPC		rref
+	+	add

Other redefined keys:

- On S/SX: 33.2 (`XQ`) and 33.3 (`XNUM`)
- On G/GX: 33.2 (`XNUM`) and 35.6 (`XQ`).
- α- Left Shift -R (on both models): sets/clears flag 23 (used by `RISCH`)
- α- Left Shift -1 (both models): sets/clears internal flag

C User flags

List of the flags used by `erable` (* if cleared by `VER`, # if set by `VER`):

- 01: if set then verbose mode else quiet mode.
- 10: internal use, if set then `erable` performs integer arithmetic otherwise `erable` performs polynomial arithmetic
- # 11: internal use, cleared if a non-rational algebraic is found
- # 12: if clear then `GCD` returns always 1 ($\rightarrow$ no simplification in algebraics and no search of multiple roots of polynomials)
- # 13: if set then complex mode, else real mode (modifies the way of simplifying expressions with `re`, `im` and `conj` and the way of rooting polynomials)
- # 14: if set then searchs formal first order factors
- # 15: if set enables construction of integer fractions and square roots of integers
- * 16: internal use, if set then inverse Laplace transform
- * 17: cleared to use user data representation, set to use internal data representation.
- * 18: internally used by `INT`, if set then `INT` integrates a fraction of sin / cos
- * 19: internally used by `INT`, if set then integration else partial fraction decomposition.
- * 20: internally used by `DIAG`, if set then force multiplication of lists to be matrice times polynomials in Horner scheme
- # 21: if set then recursive simplification for `EXPA` and `TSIMP`
- * 22: if set then the rule $i^2 = -1$ is not applied
- * 23: if set then `RISCH` does not try linearity
- * 24: if set then positivity of expressions are tested at $x = 0$ instead of $x \rightarrow +\infty$.
- * 25: if set then the rule $\sqrt{x}^2 = x$ is not applied

You should only modify flags 12,13,14,15,17 and 21 to 25.

D Thanks to ...

Many people helped me during the creation and diffusion of `erable`:

- Claude-Nicolas Fiechter and Mika Heiskanen for letting me use their long integer routines for `erable`. Special thanks to Mika for explanations about the source code of `ALG48`.
- Some of my students and netsurfers tested various versions of `erable` and encouraged me to improve it: particularly Christophe Burdin, Craig Clifford, Jerome Coss, David Czinczenheim, Ludovic Dumaine, Frederic Hermann, Stephane Monboisset, Lionel Pilot, Eric Saya, Quan Tong Duc, Samy Venin, John Wilson ... Special thanks to Gilles Virone who showed me first what an HP28/48 is able to do.
- all anonymous ftp sites administrators, particularly those of fourier.ujf-grenoble.fr (André Voutier), ftp.funet.fi, cbs.cis.com, hplyot.obspm.fr, hpcvbbs.cv.hp.com and wuarchive.wustl.edu,
- I used the following softwares to create `erable`: the `EQSTK` and `JAVA` stack displays ([9], [13]), the `JAZZ` debugger ([10]), various compilers (`JAZZ`, the `HP` tools ([1]), the `RPL` based tools ([12]) and eventually the GNU `tools` ([11]).
- I looked at the following book and softwares: [6], [2], [5], [3], [4], [7], [8], [?] . One of the best reference is certainly [3] and references therein. M. Heiskanen WWW-homepage has a lot of interesting math links.

References

[1] H. P. Corvallis. `TOOLS.EXE`. `hpcvbbs.cv.hp.com ftp.funet.fi wuarchive.wustl.edu`, 1991.

[2] P. Courbis and S. Lalande. *Voyage au centre de la HP 48 S/SX*. Angkor, 1993.

[3] J. Davenport, Y. Siret, and E. Tournier. *Calcul formel: Systèmes et algorithmes de manipulations algébriques*. Masson, 1989.

[4] J. Ferrard. *"Mathez" la HP 48 G/GX,* `Mathemati92`. D31 Diffusion (HP48), `http://www.ti-calc.org` (TI92), 1993, 1997.

[5] C. Ferraro. `POLY46SX, POLY46GX, SMATH, SMATHGX`. `ftp.funet.fi ftp.cis.com hplyot.obspm.fr`, 1993.

[6] C. N. Fiechter and M. Heiskanen. `ALG48V40.ZIP`. `http://www.hut.fi/ mheiskan`, 1997.

[7] F. Gantmacher. *Théorie des matrices*, volume 1. Dunod, 1966.

[8] M. Heiskanen. `POLYLIB.ZIP`. `http://www.hut.fi/ mheiskan`, 1992,1995.

[9] M. Heiskanen. `EQSTK60.ZIP`. `http://www.hut.fi/ mhesikan`, 1996.

[10] M. Heiskanen. `JAZZV6x.ZIP`. `http://www.hut.fi/ mheiskan`, 1996.

[11] M. Mikocevic. `GNUTOOLS`. `ftp://srcm1.zems.fer.hr:/pub/hp48/tools2.0.4.zip`, 1995.

[12] D. Müeller and R. Hellstern. `RPL48V20.ZIP`. `ftp.funet.fi cbs.cis.com hplyot.obspm.fr`, 1993.

[13] R. Steventon and A. Schoorl. `JAVA28.ZIP`. `//ftp:ftp.cis.com/pub/hp48g/uploads/java28.zip`, 1996.