Erable 3.024

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

July 8, 1998

Abstract

This is the manual of the `Erable` computer algebra system for the HP48 calculators. Section 1 describes the end user license. Section 2 describes how to install `Erable` on your HP48. Section 3 gives an overview of the capabilities of `Erable`, compared to other computer algebra systems available on calculators. Section 4 explains the user interface of `Erable`. The rest of the manual gives more details on `Erable` functions topic by topic and the appendix starting from a Frequently Asked Questions (section B) gives some references.

Contents

1 License **5**

2 Installation. **5**
- 2.1 Simplified installation for a GX. . . . 5
- 2.2 All other cases. . . . 6
  - 2.2.1 Getting the binaries from a computer. . . . 7
  - 2.2.2 Getting the binaries from another HP48. . . . 7
  - 2.2.3 Installing the binaries 7
  - 2.2.4 Installing the user keys redefinition. . . . 7
  - 2.2.5 Improve your installation 8
  - 2.2.6 Abstract of the installation commands 8

3 Introduction. **8**
- 3.1 Overview. . . . 8
- 3.2 Warnings. . . . 10
- 3.3 Erable and alg48. . . . 10
- 3.4 Implementation notes. . . . 11
- 3.5 Next upgrades. . . . 11

4 Getting started. **11**
- 4.1 Current variable. . . . 12
- 4.2 Complex and real modes 12
- 4.3 Main functions 12
- 4.4 The main menu 15

5 Simplifications. **18**
- 5.1 Rational simplifications instructions. . . . 18
- 5.2 Presimplifications instructions 18
  - 5.2.1 Linearization. . . . 18
  - 5.2.2 Development. . . . 19
  - 5.2.3 Trigonometry 19
  - 5.2.4 The `TSIMP` instruction 19
- 5.3 Recurse flag. . . . 19

6 Limits, Taylor and asymptotic series. **20**

7 Derivation and integration. **21**
- 7.1 Derivation 21
- 7.2 Integration 22
- 7.3 Integration by part 23

8 Ordinary differential equations. 24
8.1 Linear differential equations (systems) with constant coefficients. 24
8.1.1 Laplace transform. . . . 24
8.1.2 Inverse laplace transform. . . . 24
8.1.3 Linear differential equations systems with constant coefficients. . . . 25
8.2 First order equations. . . . 26

9 Substitution, change of variables: `EXEC`. 27

10 Arithmetic. 28
10.1 Complex arithmetic . . . 28
10.2 Integer and polynomial arithmetic . . . 29
10.3 Infinities. . . . 30
10.4 Modular arithmetic . . . 30

11 Factorization. Solving equations. 31
11.1 Summary of the instructions. . . . 31
11.2 A word about factorization. . . . 33

12 Linear algebra. 34
12.1 Building a matrix . . . 35
12.2 Operations . . . 35
12.3 Gauß -Jordan row reduction. . . . 35
12.3.1 Solving a linear system. . . . 36
12.3.2 Inversion . . . 37
12.3.3 Determinant . . . 37
12.4 Kernel and image of a linear application. . . . 37
12.4.1 Other examples. . . . 37
12.4.2 Stack input/output for reduction instructions. . . . 38
12.5 Diagonalization . . . 39
12.6 The MMULT— instruction. . . . 41

13 Multivariate analysis. 42

14 Quadratic forms. 43

15 Customization and other utilities. 45
15.1 Data types. . . . 45
15.2 User flags. . . . 45
15.3 Conversions . . . 45
15.4 Other functions . . . 47
15.5 Permutations . . . 47
15.6 Variables . . . 47
15.7 Differential geometry . . . 48

16 Final remarks. 48

A Frequently asked questions **49**

B All functions of Erable— listed in alphabetic order **50**

C User Keys. **53**

D User flags **55**

E Error codes for the SERIES command. **56**

F Thanks to ... **56**

1 License

Erable v3.024 © 07/1998 by Bernard Parisse, with integers routines from ALG48, © 1997 by Claude-Nicolas Fiechter and Mika Heiskanen, The S version is no more provided here. To compile it, get erablsrc.zip from:
ftp://fourier.ujf-grenoble.fr/pub/hp48 and follow instructions in the file ersolv.s.

The Erable package is made of a KERNEL.LIB library and the ERABLEG.LIB library which needs the kernel library. The kernel library is a work based on ALG48 (using long integer routines) hence is submitted to the license of ALG48 (see the file license.txt). See eqstk.doc in eqstk.zip for license information about eqstk and readme.txt for information about minwriter. The rest of the package is submitted to the ERABLE license (see the file copying.doc).

2 Installation.

This section describes the installation of Erable with the kermit software. You must choose one of the method described in section (2.1) or in section (2.2). The first method will install Erable and some stack handling improvements (a modified eqstk and the minwriter editor), it is recommended for new users. Choose the second method if you have a SX or if you want to install Erable in port 1 or higher. If you have a HP48G(X), you may use the X-modem protocol for faster transferts (look at section 2.2 and replace the command KGET by XGET except for the GXKEYS/SXKEYS file that you must download with kermit in binary mode).

2.1 Simplified installation for a GX.

Erable provides now a simplified installation procedure which should make it a lot easier for new users. Thanks to Mika Heiskanen and Claude-Nicolas Fiechter for letting me modify and distribute their eqstk library; to Jean-Yves Avenard (for the minwriter), and to Andre Schoorl (for ufl102) for letting me include their programs in the Erable distribution.

1. Go to the directory where you have unzipped erable.zip.
2. Run kermit on your computer. If you don't have kermit, you can get it at e.g. ftp://kermit.columbia.edu or at a mirror site. If your version of kermit does not work under Windows 95, restart your computer in MS-DOS mode and try again.
3. Set the line to the HP48, e.g. type
   set port 2
   if the HP is connected to the second serial port COM2 (for Linux type set line /dev/cua1).

4. To insure binary file transfer, type:
   `set file type binary`

5. Type the following command to set the speed of `kermit` to 9600 bauds:
   `set speed 9600`

6. Put `kermit` in server mode, type:
   `serv`

7. Now take your HP48. You must have at least 120K free in your HP48GX. If you do not have important data in your memory, press the `ON` key, then the `A` key, then the `F` key and release them, then press the `F` key to answer `NO` to `Try to recover memory`. Otherwise, if you are running a stack replacement like `java` or `eqstk`, switch to the HP48 built-in stack handler.

8. Type the left shift key followed by the 1 key, type the `B` menu key to select `IOPAR`, press the `A` menu key until line 1 looks:
   `IR/wire:  wire`

9. Press the α key, and type `S E T U P` then the space key then `K G E T`, release the α key and press the `ENTER` key. This will get the program `SETUP` from the computer to the HP. If an error occurs, verify the configuration of `kermit` and try again.

10. Press the `VAR` key, then the `A` menu key (for `SETUP`), this will get and install the binaries for you. If you don't have enough memory, an `Insufficient memory error` will occur. You may get a `Object in use` error if you are running stack replacement like `java` or `eqstk`, in this case, leave them (for `java`, type `JAVA` and for `eqstk`, type simultaneously `ON` and `C`). Otherwise, after about 10 minutes your HP48 will reboot.

11. After the reboot, press the `VAR` key, then the `E` menu key (again for `SETUP`). This will continue the installation and reboot the calculator after about 1 minute.

12. Eventually press the `VAR` key, then the `A` menu key (for `INIT`). This will launch the modified `eqstk` stack handler. Congratulations, you're done!!

The installation is completed, all should work smoothly. You can go to section (3)

2.2 All other cases.

You must first get the binary either from a computer or from another HP48 where `Erable` is installed. Then you have to install the binaries and eventually install the user keys redefinition.

2.2.1 Getting the binaries from a computer.

Connect your HP to your computer. Run `kermit` in server mode on your computer, and type either:
`{ KERNEL.LIB ERABLEG.LIB algbg GXKEYS } KGET` (HP48 GX)
or
`{ KERNEL.LIB erable.lib algb SXKEYS } KGET` (HP48SX)
If you have the `metakernel`, you should rename `UKEYS` to `GXKEYS` on your computer before the transfer and you should get `STARTEQW` as well to enable the use of some `Erable` commands in the `EQW` environment by hitting the `CST` key.

2.2.2 Getting the binaries from another HP48.

Check that both HP48 are in `IR` mode for `I/O` transferts.
If `Erable` is installed in a RAM card, check that the RAM card is writeable (otherwise the `RCL` function below will fail).
Put the recieving HP48 in server mode.
On the giving HP (where `Erable` is installed), call the `SENDIR` program if you have it or type the following little program:

```
<< :&:787 RCL :&:788 RCL RCLKEYS -> KERNEL.LIB ERABLEG.LIB GXKEYS
   << { KERNEL.LIB ERABLEG.LIB GXKEYS } SEND
   >>
>>
```

and `EVAL` it.

2.2.3 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`
or for port 1:
`'KERNEL.LIB' DUP RCL SWAP PURGE 1 STO`
The Erable library may be installed in any port, for example in port 2:
`'ERABLEG.LIB' DUP RCL SWAP PURGE 2 STO`
(or `'erable.lib' DUP RCL SWAP PURGE 2 STO` for a HP48SX)

2.2.4 Installing the user keys redefinition.

This step is 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 for the user mode.
- if you got `GXKEYS` (or `SXKEYS`) from another HP, type:
  `'GXKEYS' DUP RCL SWAP PURGE STOKEYS`
  (or `'SXKEYS' DUP RCL SWAP PURGE STOKEYS`)
  to assign keys for user mode.

2.2.5 Improve your installation

If you are short on memory, you can erase parts or the whole `algb` (respectively `algbg`) directory: keep `SENDIR` if you want to transfert `Erable` to another HP48. You can erase the `GXKEYS` (or `SXKEYS`) variable in the `{ HOME }` directory unless you want to customize the user key redefinition of `Erable`. If you have assigned `Erable` user key redefinition, you may decide to delete the main menu of the HP48, which is mainly useful for new users of the HP48GX: to do this, type:
`22.1 DELKEYS`

If you have a HP48GX with 128K, I recommend to install `EQSTK.LIB`, `MINWRT12` and `UFL3.LIB`. This will leave your HP48GX with 18K of RAM.

If you have enough memory, you can download the `other` directory and on-line help (`fr` or `us`).

If you install `Erable` user keys redefinition but you don't install the `MINWRT12`, you must unassign the downarrow key by typing
`35.1 DELKEYS`
(Otherwise hiting the downarrow key will return `'EDITM'`)

If you don't install `EQSTK.LIB`, you must unassign the left shift-downarrow key by typing
`35.2 DELKEYS`

2.2.6 Abstract of the installation commands

This is the list of commands in the `algb` or `algbg` directory:

- `GETALL`: downloads directories with some programs using `Erable`
- `INIT`: downloads the `Erable` libraries if they are not installed and reboots. Otherwise, installs `GXKEYS` or `SXKEYS` as user keys redefinition and reboots.
- `PURG`: purge the `algb` or `algbg` directory
- `SENDIR`: used to transfert the `Erable` libraries and the user keys assignement from one HP to another HP. Before calling `SENDIR`, you must check that both HP48 are in `IR mode` (in the `I/O` menu) and you have to put the recieving HP48 in server mode.
- `SETFR`: set some flags: e.g. European date display, or radian 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, limits, Taylor and asymptotic 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 problems solved by a TI92 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, diagonalization of matrices, change of variables,...

If you have both `alg48` and `Erable`, then you have the most complete computer algebra system currently available on a calculator (HP, TI and CASIO).

Examples:

$\frac{1+\sqrt{2}}{1+2\sqrt{2}}$	EXPA	$\frac{3}{7}+\frac{1}{7}\sqrt{2}$
$e^{i\pi/6}$	TSIMP	$\frac{\sqrt{3}}{2}+\frac{i}{2}$
$\ln(1+i)$	TSIMP	$\frac{\ln(2)}{2}+\frac{1}{4}\pi i$
$\frac{1}{e^x-1}$	RISCH	$\int \frac{1}{e^x-1}\,dx = \ln(e^x-1)-x$
$\lim_{x\to 0}\frac{y^x-1}{x}$	LIMIT	$\ln(y)$
$\begin{pmatrix} 1 & 1 & a \\ 1 & a & 1 \\ a & 1 & 1 \end{pmatrix}$	rref	$\begin{pmatrix} 2-a-a^2 & 0 & 0 \\ 0 & -2+a+a^2 & 0 \\ 0 & 0 & 2-a-a^2 \end{pmatrix}$

Examples not solved natively by a TI92:

$\sin(x)/(e^x-1), x, 4$	SERIES	$1-\frac{1}{2}x-\frac{1}{12}x^2+\frac{1}{12}x^3+O(x^4)$
$\cos(x)^2$	LAP	$\frac{1}{2x}+\frac{1}{4(x-2i)}+\frac{1}{4(x+2i)}$
$\frac{1}{(x^2+1)(x+1)}$	ILAP	$\frac{-1}{2}\cos(x)+\frac{1}{2}\sin(x)+\frac{1}{2}e^{-x}$
$y'=xy^2$	DSOLVE EXPA	$y=\frac{y_0}{e^{-1/2x^2+1/2x_0^2}}$
$\int_0^1 (1+2x^2)e^{x^2}\,dx$	EXPA	e
$\int_a^b \frac{e^x}{x}\,dx, \quad e^x=y$	EXEC	$\int_{e^a}^{e^b}\frac{y}{y\ln(y)}\,dy$
$\begin{pmatrix} 0 & 1 \\ -1 & 2 \end{pmatrix}$	JORDAN	Char :1:$(1,0)$; Eigen :1:$(-1,-1)$

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 math 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, large matrices (say e.g. 100×100), ... Before learning how to solve exactly a problem, I strongly recommend that you 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 most of the time slower (about 2 times I would say, this is only a mean, sometimes `alg48` is 3 times faster than `Erable`, sometimes `Erable` is 1 to 2 faster than `alg48`). Why? Because `Erable` handles complexes and square roots natively: e.g. `Erable` simplifies expressions like $(1+i)/(1+2i) = \frac{3-i}{5}$, $(1+\sqrt{2})/(1-2\sqrt{2}) = -\frac{5}{7} + \frac{-3}{7}\sqrt{2}$, $e^{5i\pi/6} = -\frac{1}{2}\sqrt{3} + \frac{i}{2}$. Hence arithmetic operations in `Erable` must be generic, that's why they are slower. On the other hand, `Erable` treats matrices as global objects for simplifications, as `alg48` simplifies matrices element by element, hence `Erable` may be faster if matrices are involved.

- `Erable` accepts strings embedded in symbolics, this means that if you `EXPA`nd $(5x+12)^{16}$ with `Erable` you'll get the exact answer. You may also do arithmetic operations on strings representing integers (e.g. try `"123456789123456789" DUP MULT`)

- `Erable` handles numerical data (reals which are not integers and complexes with non integral real and/or imaginary parts)

- `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 and then, 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), limits and 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 `ML` and `Sysrpl` with HP48 supported entries and standard instructions (and a few unsupported but static entries), 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 (RedHat 4.2), the GNU emacs editor, my patched version of the x48 emulator (with almost instantaneous file transfers) 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 (like Netscape), you may prefer to go to my professional homepage:
`http://www-fourier.ujf-grenoble.fr/~parisse/english.html`
or to my personal homepage:
`http://perso.wanadoo.fr/bernard.parisse/english.html`

4 Getting started.

This section explains the interface of `Erable`. It is assumed that you have installed the user keys redefinition as explained in section (2.2.4). Now you must switch your calc to user mode. Look at the status lines, if `USER` appears, then you are done, otherwise (or if `NORM` appears when using `JAVA`) you must type the left shift key followed by the α key once or twice (try once and type a key, if `USER` disappears then type the left shift- α key sequence twice).

4.1 Current variable.

When you start `Erable` (and each time you reset it by typing `VER` followed by the `ENTER` key), the current variable is set to `'X'`. The `DEL` key is assigned to `X`. It is therefore easy to enter a symbolic expression depending of `X` because you do not need to press the α key. You may need to modify the current variable. This is done by changing the content of the `VX` variable (in the `{ HOME }` directory). For example to set the current variable to `Y`, type:
`Y 'VX' STO`

4.2 Complex and real modes

`Erable` has two major modes: complex and real mode. The user flag 13 is set in complex mode and cleared in real mode (default state). This mode affects the way `Erable` outputs results. For example, partial fraction decomposition is made over $\mathbb{C}$ or over $\mathbb{R}$ depending of the current mode. If you see unwanted i on the stack, this means that `Erable` is probably in complex mode. For G/GX users, the key sequence α-right shift-`CST` provides an easy way to switch to complex or real mode using arrow keys to select the mode (type the `ENTER` key to confirm). HP48S/SX users can go to real mode either by typing `13 CF` or by typing `VER` (which reset all flags to their default state) and can switch to complex mode by typing `13 SF`.

4.3 Main functions

The keys redefined by `Erable` 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 hitting threes keys: $\boxed{\alpha}$, $\boxed{\text{Right Shift}}$, $\boxed{+}$.

The same method applies for other arithmetic operations. The $+$, $-$, $*$, $/$, y^x, $\sqrt{x}$, $\pm$, $1/x$, ∂, $\int$ keys are redefined (after $\boxed{\alpha}$ and $\boxed{\text{Right Shift}}$) as `add`, `SUBT`, `MULT`, `DIV1`, `POWER`, `SQRT`, `CHS`, `INVL`, `der1`, `RISCH`. These commands are `Erable` commands for addition, subtraction, multiplication, division, power, square root, change sign, derivation and integration with respect to the current variable [1].

All other `Erable` functions may be reached by the `MTH`-Erable menu[2] or by a key preceded by α-right shift. To launch the `MTH`-Erable menu, type the `MTH` key, use arrow keys or hit a number to select a topic and type the `ENTER` key. After that you select the desired command by hitting the white `A-F` keys or search further by hitting the `NXT` key.

Remark 1 *A main menu of the HP48GX is available by hitting the* `PRG` *key and may be useful for beginners. Note that the former* `MTH` *and* `PRG` *key definition are reached by* α*-right shift-*`MTH` *or* α*-right shift-*`PRG`*. If you want to delete the*

[1] This variable is contained in 'VX', it is by default set to 'X', note that you can easily type X by hitting the DEL key

[2] This menu is not available on HP48 S/SX

PRG *key assignement, type* 22.1 DELKEYS *(note that the* 0.BACK *option of the* MTH *menu will no longer work)*

You can directly reach a topic of Erable by hitting α-right shift followed by the number of the topic in the MTH-Erable menu. For example, if you hit α-right shift-1, you will get the BASE ALGEBRA menu, that is the most useful commands of Erable that are illustrated with an example below:

- EXPA (expand an expression):
  $\frac{(X+1)^4}{X^2-1}$ EXPA $\frac{X^3+3X^2+3X+1}{X-1}$
- COLC (factorize an expression):
  X^4-1 COLC $(X^2+1)(X+1)(X-1)$
- der1 (derive an expression with respect to the current variable):
  $\ln(\ln(X))^2$ der1 $2\frac{1}{X}\frac{1}{\ln(X)}\ln(\ln(X))$
- RISCH (integrate an expression with respect to the current variable):
  $e^X\sin(X)^2$ RISCH $e^X\frac{1}{2}+e^X((-\sin(2X)\frac{1}{5}+\frac{-1}{10}\cos(2X))$
- LIMIT (of the expression at stack level 2, you must specify the variable and the limit point at level 1, e.g. 'X=0'):
  $(1+\frac{1}{X})^X$, $X=\infty$ LIMIT e
- TAYLR (the built-in one):
  $\sin(X)$, X 4 TAYLR $X-\frac{1}{3!}X^3$
- EXEC (to make a substitution at stack level 2, you must specify the substitution at level 1, e.g. 'X=1+Y'):
  $\ln(X^2+1)+\arctan(X)$, $X=2$ EXEC $\ln(2^2+1)+\arctan(2)$
- SOLV (to isolate a variable at level 1 in an expression at level 2):
  X^4-1, X SOLV X, { 1 -1 'i' '-i' }
- FSIGN (sign of a rational fraction):
  X^2-X-2 FSIGN returns 'X': { + 1 - 2 + }. You read the sign starting from $-\infty$ at the left to $+\infty$ at the right. 1 and 2 are here zeroes from the rational fraction, in general the value appearing in the list are roots or poles.
- IPP (integration by part: see 7.3):
  $\int_1^2\ln(X)dX$, X IPP $\int_1^2\ln(X)\,dX=2\ln(2)-\int_1^2 1\,dX$
  (applies $\int uv'=[uv]-\int u'v$ where $\ln(X)$ is the function u and the argument X is the function v)

Some of these commands are directly assigned to a shortcut:

- the EXPA command is assigned to the α-right shift-SPC sequence,
- the EXEC command is assigned to the α-right shift-EVAL sequence,

- i is assigned to the `CST` key,
- X is assigned to the `DEL` key,
- ∞ or $+\infty$ is assigned to the α-right shift-`DEL` sequence,
- $-\infty$ is assigned to the α-left shift-`DEL` sequence,

Here is a brief description of all the functions of `Erable` which may be launched this way:

- α-right shift-1: basic algebra commands:
  `EXPA`, `COLC`, `der1`, `RISCH`, `LIMIT`, `TAYLR`, `EXEC`, `SOLV`, `FSIGN`, `IPP` (see above)
- α-right shift-2: complex:
  `re`, `im`, `conj`, `arg`, `abs` and i. Note that you can always get i if you hit the `CST` key.
- α-right shift-3: trigonometry:
  `TEXPA` (trigonometric expand), `TRIGLIN` (trig. linearization), `SINCOS` (convert complex exp and ln to trigonometric expressions), `TRIG` (applies $\sin^2 + \cos^2 = 1$ to simplify an expression), `TRIGCOS` (same as `TRIG`, returns only cosines if possible), `TRIGSIN` (same as `TRIGCOS` but returns only sines if possible), `HALFTAN` (convert to tan of the half-angle), `->SC2` (convert tan to sin and cos of the double angle), `T->SC` (convert tan to sin and cos of the same angle).
- α-right shift-4: matrices
  `rref` (row reduction to echelon form), `JORDAN` (diagonalization of matrices), `det` (determinant), `PCAR` (characteristic polynomial), `SYST` (solves a linear system), `SOLGEN` (returns all solutions of a linear system), `RDET` (determinant by row reduction), `RANG` (half-row reduction), `idn` (symbolic identity matrix), `LCXM` (to build a matrix), `HILBERT` and `VAND` to build Hilbert and Vandermonde matrices.
- α-right shift-5, conversions:
  `AXL` (numeric to symbolic matrices or conversely), `EXEC` (substitution), `SINCOS` (exponentials and logarithms to sines and cosines), `EXPLN` (sines and cosines to exponentials), `FXND` (fraction to numerator and denominator or equality to left and right handsides), `NDXF` (numerator and denominator to fraction), `AXQ` and `QXA` (matrix to symbolic representation of quadratic forms and conversely)
- α-right shift-6, integer and polynomial arithmetic:
  `DIV2` (euclidean division), `GDC1` (usual greatest common denominator), `GCD3` (extended gcd), `ABCUV` (Bézout identity), `LCM1` (least common multiple), `PF` (partial fraction decomposition), `COLC` (factorization), `DIVIS` (list of divisors), `SIMP2` (simplification of stack levels 2 and 1), `EULER` (Euler indicatrix), `fact` (factorial), `comb` ($\left(\begin{array}{c} n \\ p \end{array} \right)$).

- α-right shift-7, solve and factorization:
  `FROOTS` (roots and poles of a fraction with multiplicity), `FCOEF` (reverse of `FROOTS`), `SOLV` (isolate a variable in an equation), `LNCOLC` (collect logarithms), `COLC` (factorization), `EXEC` (substitution)

- α-right shift-8, exponentials and logarithms:
  `EXPLIN` (linearization of exponentials), `EXPLN` (convert sines and cosines to exponentials), `LNCOLC` (collect logarithms), `TEXPA` (expand logarithms and exponentials), `TSIMP` (transcendental presimplification)

- α-right shift-9, differential calculus:
  `der` (derivative and gradient), `RISCH` (integration), `IPP` (integration by part), `SERIES` (asymptotic series expansion), `LIMIT` (limits), `DSOLVE` (ordinary first order differential equation solver), `LDEC` (linear differential equation with constant coefficients solver), `LAP` (Laplace transform), `ILAP` (invert Laplace transform)

An easy way to configure `Erable` is to type the α-right shift-`CST` sequence (for `Erable MODES`) and select the mode you want to switch to.

Let's finish by 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) and system flag 2.

- On a G/GX models, use α - Right Shift -`Q` or →`NUM` (not shifted).

4.4 The main menu

If you hit the `PRG` key in user mode, you will get a main menu for the HP48, not only for `Erable`. This menu will help new users of the HP48. Experienced users can remove this key assignment, saving some memory (using the command `22.1 DELKEYS`). Here is a brief description of this main menu:

1. `EDITORS`:
   - `EDIT LEVEL 1`:
     To modify the object at stack level 1 (shortcut: downarrow)
   - `VIEW LEVEL 1`:
     To view (not modify) the object at stack level 1 (shortcut: left shift-downarrow)
   - `EDIT STACK`:
     To Delete, copy or move object from one stack level to another stack level (shortcut: uparrow or right shift-uparrow)
   - `SPEC. CHAR.`:
     To get special characters (shortcut: right shift-`PRG`)

- NEW EQUATION:
  To enter a new equation in the EquationWriter environment (shortcut: left shift-ENTER)
- NEW TEXT:
  Enter a new string with the Minwriter editor. See the Minwriter documentation for editing keys.
- NEW MATRIX:
  Enter a numeric array in the MatrixWriter environment (shortcut: right shift-ENTER)
- NEW LIST:
  Enter a new list with the Minwriter editor.
- NEW PROGRAM:
  Enter a new program with the Minwriter editor.
- PICTURE:
  Modify the graphic object (e.g. the graph of a function) in the PICTURE environment (shortcut: left shift-rightarrow)

2. VAR, I/O:

   - MEMORY:
     The memory handler of the HP48: see your variable and their contents (shortcut: right shift-VAR)
   - SEND/GET:
     The Input/Output menu of the HP48, to exchange files with a computer or another HP48 (shortcut: right shift-1)
   - PORTS:
     Management of your RAM/ROM card extensions (shortcut left shift-1)

3. PROGRAMS:

   - USER PROG:
     The programmation menu of the HP48 (shortcut in user mode: α-right shift-PRG)
   - LIBRARIES:
     The libraries installed (shortcut rightshift-2)

4. PHYSICS

   - EQ. LIBRARY:
     The Equation Library of the HP48 (shortcut: right shift, 3)
   - CONSTANTS:
     Physical constants and other utilities (shortcut left shift-3). Choose COLIB (B menu key) to get either the library of constants (CONLI) or the CONST function (convert e.g. 'c' at level 1 to the speed of light)

- DATE & TIME:
  To set or modify the date and the time, and compute with dates and time (shortcut: left shift-4)

5. SETUP

   - CALC MODES:
     Modes of the HP48: system flags, angle format, clock display, ... (shortcut right shift-CST)
   - RESET ERABLE:
     Like the VER command of Erable: restore standard state (no shortcut)
   - REAL MODE:
     Put Erable in real mode, like the command 13 CF
   - COMPLEX MODE:
     Put Erable in complex mode, like the command 13 SF
   - INTEGER ARIT:
     Assumes now that you are doing arithmetic on integers, not polynomials, like the command 10 SF
   - POLYN. ARIT:
     Assumes now that you are doing arithmetic on polynomials, not integers, like the command 10 CF
   - NUMERIC MODE:
     Asumes now that your integer input are not exact but numeric, like the command XNUM
   - SYMBOLIC MODE:
     Assumes now that you integer input are exact, not numeric approximations, like the command XQ
   - TIME SETUP:
     The TIME menu of the HP48 to set, browse alarms and date/time (shortcut right shift-4)

6. STATISTICS
   The STAT menu of the HP48: single-var, frequencies, fit data, summary stats (shortcut: right shift-5)

7. UNITS:
   The UNITS menu of the HP48: length, area, volume, time, speed, mass, force, energy, power, pressure, temperature, electricity, angle, light, radioactivity, viscosity (shortcut: right shift, 6)

8. NUMERIC
   The built-in numerical functions of the HP48:

- `MATH FNCS`
  vectors, matrices, list, hyperbolic, real fonctions, base probability, Fast Fourier Transform, complex, constants: the built-in `MTH` menu (shortcut in user mode: α-right shift-`MTH`)
- `SOLVE EQN`
  The `SOLVE` menu of the HP48: numeric or semi-numeric solutions for equations, differential equations, polynomial, linear systems, financial problems (shortcut: left shift-`7`)

9. `GRAPH`
   If you want to plot the function at stack level 1, choose `PLOT LEVEL 1`, otherwise choose `PLOT MENU`

10. `ERABLE`
    To launch the `Erable` main menu if you don't remember that the `MTH` key does it directly. See description above.

5 Simplifications.

Two kinds of simplifications are provided: full rational simplification (`EXPA`) and transcental presimplification (`TEXPA`, `EXPLIN`, `TRIGLIN`, `TRIGSIN`, `TRIGCOS`, ..., `LNCOLC`, `TSIMP`). In many situations, full rational simplification achieve the whole simplification, but sometimes you will need to detect relations between exponentials and logarithms; in this situation you should call `TEXPA`, `TRIGLIN`, ..., `TSIMP`, followed by `EXPA` or `COLC` to finish the simplification.

5.1 Rational simplifications instructions.

`EXPA` does a complete simplification of an expression viewed as a rational fraction, `COLC` tries to factorize a symbolic. For convenience, arithmetic operations of `Erable` perform automatic rational simplifications: e.g. `add` (shortcut: α-right shift-+) is equivalent to `+ EXPA`.

5.2 Presimplifications instructions

To simplify non rational expressions, you will most of the time apply identities like $\ln(xy) = \ln(x) + \ln(y)$ or conversely and after you will call `EXPA`.

5.2.1 Linearization.

Exponential and trigonometric linearization are implemented via:

- `EXPLIN`: $e^x e^y \to e^{x+y}$ and for integral powers $(e^x)^n = e^{nx}$
- `TRIGLIN`: $\sin(x)\sin(y) = \frac{1}{2}(\cos(x-y) - \cos(x+y))$ and similar identities, $\sin(x)^n$ and $\cos(x)^n$ for n integer

5.2.2 Development.

The `TEXPA` instruction applies the following identities:

- $e^{x+y} = e^x e^y$ and $e^{nx} = (e^x)^n$ for n integer,
- $\ln(xy) = \ln(x) + \ln(y)$.
  Warning: this identity is only valid module $2i\pi$
- $\sin(x+y) = \sin(x)\cos(y) + \sin(y)\cos(x)$ and $\cos(x+y) = \cos(x)\cos(y) - \sin(x)\sin(y)$

5.2.3 Trigonometry

The remaining trigonometric simplifications instructions are:

- `EXPLN` and `SINCOS` to apply Euler idendities in both directions
- `TRIGCOS`, `TRIGSIN`: replace $\sin^2$ [resp. $\cos^2$] by $1 - \cos^2$ [resp. $1 - \sin^2$]
- `TRIG`: replace complex logarithms with arctan functions, then does `TRIGCOS` or `TRIGSIN` (according to the last call to one of these functions)
- `TAN2SC`: replace tan by sin / cos
- `TAN2SC2`: replace $\tan(x)$ by $\sin(2x)/(1+\cos(2x))$ or by $(1-\cos(2x))/\sin(2x)$.
- `HALFTAN`: replace $\sin(x)$, $\cos(x)$ and $\tan(x)$ in terms of $\tan(x/2)$

5.2.4 The `TSIMP` instruction

`TSIMP` is used to minimize the number of rational "variables". It may be used if `RISCH` fails because it returns an expression which is "weak-normalized". Note that `TSIMP` considers trigonometric functions as complex exponentials, and simplifies them this way and that the output of `TSIMP` is affected by the state of the flag 13 (complex flag): if flag 13 is cleared, then complex logarithms and exponentials are converted to arctan and sin/cos functions.

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 Limits, Taylor and asymptotic series.

The program SERIES computes Taylor series, asymptotic development and limit at finite and infinite points. It should cover a lot of weird limits, even some that are not handled by the TI92 (not surprising!) nor by maple (more surprising!) like:

$$\lim_{x \to 0} sin(1/x + x) - sin(1/x)$$

The LIMIT instruction may be used if you need only the limit. Note that SERIES handles more limits than LIMIT, but is a lot slower for trivial cases. SERIES can not be used with non-exact arguments (like 0.1) and should not be used with parameters. LIMIT may give strange results with parameters. If you see a warning message, you will get a binary integer as answer, this means that SERIES or LIMIT was not successful, the binary integer is an error code.

Syntax of SERIES:
Put on the stack the following arguments in this order

- the function $f(x)$
- the variable if the limit point is 0 or an equation $x = a$ if the limit point is a (and the variable is x). This entry is optional if the stack as only 1 argument.
- the order for series expansion (optional), by default 4 (minimum 2, maximum 20). If the order is a positive integer, the series expansion is made from the right, if the order is a negative integer from the left. For bidirectional series expansions, give the order as a binary integer (e.g. #5d).

Type SERIES, this computes the bidirectional limit at level 3. At level 2, you get a list of two elements: the series expansion and the rest order. They are expressed in terms of a small parameter h. At level 1, h is expressed in terms of the initial variable (hence calling EXEC would return the series expansion and the rest in terms of the initial variable).

Remark 2
- *Note that the series expansion is not always fully truncated, don't forget to look at the rest. If you need to truncate the series expansion, split the list on the stack (hit the* EVAL *key) and call the* TRUNC *function.*
- *Sometimes* SERIES *will not be successful and returns an error code. You can look at E for more details about the failure.*
- *For mono-directional series expansion, either precise the order as a positive or negative integer, or for default order, put an equation $x = a + 0$ for a left-directional series expansion at a or $x = a - 0$ for a right-directional series expansion.*
- *For limits at infinity: you may use the ∞ symbol (from keyboard type α-right shift-*DEL *in user mode or α-right shift-I in normal mode).* To get $X = +\infty$, type $X = \infty$, then hit EVAL.

Examples:
$1/x\ x=\infty$ `SERIES`
$1/x\ x=+\infty$ `SERIES`
$1/x\ x=-\infty$`SERIES`
$\sin(x)/x\ x$ `SERIES`
$\sin(x)/x\ x=+\infty$`SERIES`
$\sqrt{(2+x)}\ x\ 5$ `SERIES`
$\sin(1/x+x)-\sin(x)\ x$ `SERIES`
$(\ln(-\ln(x+x^2))-\ln(-\ln(x)))*\ln(x)/x\ x=0+0$ `SERIES`

The syntax of `LIMIT` is similar: put the function and the `'variable=limit_point'` equation on the stack. Note that you can not force the order for series expansions and `LIMIT` handles only bidirectional limits (except at infinity). `LIMIT` returns only the limit at stack level 1.

`Erable` can handle relatively complex limits, like the example above (extracted from the `Mupad` on-line help):

$$\frac{\exp\left(\frac{e^{-x}}{e^{-x}+e^{\frac{-2x^2}{x+1}}}x\right)-e^x}{x}$$

$X=\infty$ `LIMIT` returns $-e^2$.

In addition `Erable` provides the:

- `TRUNC` instruction which truncates a series expansion at level 2 with respect to the rest at level 1.
- `DIVPC` instruction which make a division in ascending power up to an integer order. The numerator is at level 3, the denominator at level 2 and the order at level 1.

These instructions may be used to understand series expansions without cumbersome calculations.

7 Derivation and integration.

7.1 Derivation

The `Erable` derivation instructions are `der` and `der1`, they compute the derivative of a (list of) function(s) like the built-in instruction but do not evaluate numeric expressions (like $\sqrt{2}$ or $\frac{1}{2}$). `der1` is used for derivation with respect to the variable contained in `VX` and takes only one argument (the function to derive). `der` is used with 2 arguments: the (list of) function(s) to derive at level 2 and the variable with respect to which you want to derive at level 1. 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 $z(x, y, ...)$.
Examples:
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 $2x\sqrt{(x^2)3-1}$.
Try `'Y(X,X^2)' X der`.

7.2 Integration

The main integration command are `RISCH` and `EXPA`. The `RISCH` program accepts functions as input and (tries to) return the primitive. `EXPA` should be called for symbolic expressions which contains the $\int$ symbol. The last computed antiderivative is stored in the variable `PRIMIT`. The variable `ERABLEMSG` contains additionnal information if `RISCH` returns an unevaluated antiderivative (with a $\int$ sign).

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 must sometimes be used in conjunction with the `TSIMP` function to get "weak normalization". If you get `No closed form` in `ERABLMSG`, try `TSIMP` and `RISCH` again, if you get again the message `No closed form`, this does not mean that `RISCH` failed, but that your input does not admit an antiderivative which may be expressed in terms of elementary functions.

Remark 3 • `RISCH` *is only a partial implementation of the Risch algorithm: it works with pure transcendental extensions (i.e. square root are generically not allowed), and exponential polynomial parts 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.

In addition to this partial implementation, `RISCH` *can integrate fractions of the type* $F(x, \sqrt{ax^2+bx+c})$.

- *You can not use the name of the current variable as a parameter name in an integral, for example if* `VX` *is set to* `X`*, evaluation of:*

$$\int_0^T (X^2 - Y^2)\ dY$$

does not return a correct answer, because X in the integral is a parameter. You can however use X as integration variable, e.g.

$$\int_0^T (X^2 - Y^2)\ dX$$

works.

For integrals with bounds, the right instruction is `EXPA`.
Example of `EXPA` usage (in real and symbolic mode):

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

If you have still computed the antiderivative e.g. with `RISCH`, you can evaluate it between two bounds using `PREVAL`. Arguments of `PREVAL` are a function $f(x)$ at level 3, lower and upper bounds a and b at level 2 and 1. It returns $f(b)-f(a)$ (x is the variable contained in `VX`).

Remark 4 *Warning:* `EXPA` *does not detect discontinuities of the antiderivative. It blindly computes the value at both end of the integration interval (by a call to* `LIMIT`*, hence infinite bounds are allowed) and returns the difference. For example, $\int_0^{2\pi} \frac{1}{\sin(x)+2}$ returns 0. You should always check the answer numerically and if the answers are not the same, you have to study the antiderivative for discontinuities.*

7.3 Integration by part

Integration by part is implemented via the `IPP` command. You have to put a defined integral $\int_a^b f(t)\ dt$ at level 2 and a function $u(t)$ at level 1. Let $v = f/u'$, then `IPP` returns

$$\int_a^b f(t)\ dt = \int_a^b u'v(t)\ dt = [uv(t)]_a^b - \int_a^b uv'(t)\ dt$$

You may call `IPP` twice or more. In this case, the transformation applies on the first integral of the second member of the equality.
Example:

- $$\int_0^x \arcsin(t)^2\ dt$$

  `'T' IPP` $\sqrt{1-T^2}$ `IPP EXPA` returns the antiderivative of $\arcsin(x)^2$.

- Try:
$$\int_0^x \exp(t)\sin(2t)\ dt$$
with $\exp(t)$ `IPP` twice. The resulting equality may be used to compute the antiderivative (`RISCH` may be called directly on this example, but `IPP` gives a pedagogical approach)

8 Ordinary differential equations.

8.1 Linear differential equations (systems) with constant coefficients.

The most efficient tool for these equations is the Laplace transform defined by:
$$Y(s) = \mathcal{L}(y)(s) = \int_0^\infty e^{-st}y(t)\ dt$$
Example: solve $y' + 2y = \cos(x)$. Apply $\mathcal{L}$, since:
$$\mathcal{L}(y')(s) = s\mathcal{L}(y)(s) - y(0)$$
we get:
$$(s+2)\mathcal{L}(y)(s) = \mathcal{L}(\cos(x))(s) + y(0)$$
hence:
$$y(x) = \mathcal{L}^{-1}\left(\frac{\frac{s}{s^2+1} + y(0)}{s+2}\right) = \mathcal{L}^{-1}\left(\frac{\frac{s}{s^2+1}}{s+2}\right) + y(0)\mathcal{L}^{-1}(\frac{1}{s+2})$$
since $\mathcal{L}(\cos(x))(s) = s/(s^2+1)$. This method is implemented by the `LDEC` instruction. It takes the second member at level 2 (here $\cos(x)$), and the characteristic equation at level 1 (here $x+2$) and returns the solution vanishing at the origin at level 1, the characteristic equation at level 2, and 1 at level 3. Be sure that the current variable name contained in `VX` is the right one!

8.1.1 Laplace transform.

The program `LAP` takes the function f as argument and returns $\mathcal{L}(f)$ (Laplace transform is performed with respect to the variable contained in `VX`).

8.1.2 Inverse laplace transform.

The program `ILAP` performs inverse Laplace transform of rational fractions.
Example: for $x/((x^2+1)(x+2))$, type `'X/(X^2+1)' 'X+2' / ILAP` to get the answer:
$$y(x) = \frac{1}{5}(2\cos(x) + \sin(x)) + \frac{-2}{5}e^{-2x}$$

Remark 5 *The name of the Laplace variable is the same name as the normal variable (and is contained in* `VX`*).*

8.1.3 Linear differential equations systems with constant coefficients.

Example: suppose we want to solve the following system:

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

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 $\mathcal{L}$-transform and get:

$$(as^2 + bs + c)\mathcal{L}(y) = \mathcal{L}(f)(s) + a(sf(0) + f'(0)) + bf(0)$$

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

$$\frac{\mathcal{L}(f)(s) + a(sf(0) + f'(0)) + bf(0)}{as^2 + bs + c}$$

But this method may be applied to systems of linear differential equations as well. 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 $\mathcal{L}(b)$ 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

$$(sI - A)\mathcal{L}(y) = \mathcal{L}(b) + y(0)$$

(I denotes the identity matrix), hence:

$$\mathcal{L}(y) = (sI - A)^{-1}(\mathcal{L}(b) + y(0))$$

and:

$$y = \mathcal{L}^{-1}\left[(sI - A)^{-1}(\mathcal{L}(b))\right] + \mathcal{L}^{-1}\left[(sI - A)^{-1}y(0)\right] \quad (1)$$

The purpose of `LDEC` is to compute

$$y = \mathcal{L}^{-1}((sI - A)^{-1}(\mathcal{L}(b)))$$

the solution with initial data $y(0) = 0$, given A and b. The stack should be prepared as:
`stk2:` b
`stk1:` A,

then type `LDEC` and you will get y at stack level 1. Stack level 3 is the comatrix, stack level 2 is the determinant of $(sI - A)$.

Let's do it for the example above:
First put b on the stack: `{ 1 'EXP(X)' }` (here `VX` is set as usual to `'X'`). Now put the matrix A on the stack:
`{ { 1 -1 } { 2 4 } }`
After calling `LDEC`, we get:

$$\begin{cases} y_1 &= -1/2e^x - 2/3 + 2e^{2x} - 5/6e^{3x} \\ y_2 &= 1/3 - 2e^{2x} + 5/3e^{3x} \end{cases}$$

At level 2, we have $\det(sI - A)$ (with $s = X$):

$$\det(sI - A) = (2 - s)(3 - s)$$

and level 3 is the comatrix of $sI - A$:

$$\begin{pmatrix} -4 + s & -1 \\ 2 & -1 + s \end{pmatrix}$$

Level 2 and 3 will be useful if we want to compute the solution of the `LDEC` with non-zero initial data , e.g. $y_1(0) = 1, y_2(0) = 2$. If we look at Equation (1), we see that we have to add $\mathcal{L}^{-1}\left[(sI - A)^{-1}y(0)\right]$ to the previous solution. To do this, type `{ 1 2 }`, multiply by the comatrix from level 3, multiply by the inverse of the determinant from level 2 (you can note divide a vector, you must first invert the determinant and then multiply):

$$(sI - A)^{-1}(1, 2) = (\frac{-6 + s}{6 - 5s + s^2}, \frac{2s}{6 - 5s + s^2})$$

call `ILAP`:

$$(4e^{2x} - 3e^{3x}, -4e^{2x} + 6e^{3x})$$

and add the result to the previous solution:

$$\begin{cases} y_1 &= -1/2e^x - 2/3 + 6e^{2x} - 23/6e^{3x} \\ y_2 &= 1/3 - 6e^{2x} + 23/3e^{3x} \end{cases}$$

8.2 First order equations.

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

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

The input is the function $f(x, y(x))$ or an equation like `'d1Y(X)+Y(X)=2'`. 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. The equation type is stored in the `ODETYPE` variable.

9 Substitution, change of variables: `EXEC`.

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

- one algebraic substitution:
  If stack 1 is an equation (`'objA=objB'`), replace `objA` by `objB` in stack2.
  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.
  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 $\sqrt{1-y}$.
- multiple substitutions:
  If stack 1 and 2 are lists, replace each object of list2 in stack level 3 by the corresponding object of list1. The syntax is
  `{ old_name_1 ... old_name_n } { expr_1 ... expr_n } EXEC`
  Note that here `EXEC` does only substitutions.
  Examples:
  - `'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 $\sqrt{1-\sin(x)^2}$.
  - ```
    'COS(X)+i*SIN(X)'
    { SIN COS }
    { << i * EXP DUP INV - i 2 * / >>
      << i * EXP DUP INV + 2 / >>
    ```
    ```

```
}
EXEC
```

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

- variable isolation:
  If stack 1 is a symbolic but not an equality, `EXEC` tries to isolate stack level 1 in stack level 2. Example:

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

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

- *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`

10 Arithmetic.

10.1 Complex arithmetic

- `re`: real part
- `im`: imaginary part
- `conj`: conjugate
- `abs`: absolute value (modulus)
- `arg`: argument. Warning: for expressions containing variables, the returned argument is only valid modulo π.

Remark 6 *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: either replace your global name, say `'Z'`, *by* `'X+iY'` *or set flag 13 (shortcut* α*-right shift-*`CST` *or* `13 SF`*)*

10.2 Integer and polynomial arithmetic

You may force integer arithmetic by setting flag 10 (shortcut α-right shift-`CST` or `10 SF`). Otherwise, polynomial arithmetic is assumed. This is important for instructions like `GCD3` or `ABCUV`.

- `DIV2`: euclidean division. Stack 2 is the quotient, stack 1 the remainder.
- `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` returns `'X+1'`
  `25 15 GCD1` returns `5`
  If flag 12 is clear returns `1`.
- `LCM1`: lowest common multiple (`GCD1`$(a,b)\times$`LCM1`$(a,b) = a\times b$). Examples:
  `'X^2+2*X+1' 'X^2+3*X+2' LCM1` returns `'(X^2+2*X+1)*(X+2)'`
  `25 15 LCM1` returns `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+3*X+2' 'X+1' -> -1 1 1
  'X^2+2*X+1' 'X^2+3*X+2' 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+x+1 and x^2+3x+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 with 25.

- `PA2B2` (`kernel` library):
  Given a prime p which is 1 modulo 4, returns a complex $z = a + ib$ such that $|z|^2 = a^2 + b^2 = p$. Used for factorization of Gauß integers.

- `XFRC`:
  Same as →`Q` but handles quadratic irrationals (recognize a quadratic irrational if its expansion in continued fraction is ultimately periodic of period less or equal to 3) *Examples:*

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

- `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 $.5 + .2 * x$.

10.3 Infinities.

The arithmetic operations of `Erable` accept infinity arguments. The $+\infty$ symbol may be obtained from the keyboard by α-right shift-I in normal mode or α-right shift-`DEL` in user mode followed by `ENTER`. The $-\infty$ symbol is obtained by hitting the quote key, the - key and α-right shift-I or α-right shift-`DEL` followed by `ENTER`. ∞ and $-+\infty$ are understood as unsigned infinity (with the following exception: ∞ is understood as $+\infty$ in a bound of an integral or in a `LIMIT` instruction). The ? symbol means that the result of an operation is undefined. All operations involving ? will return ? after simplifications.

Remark 7 *The current release returns* `?` *for some expressions which are not undetermined, like* $(+\infty)^{-\infty}$.

Be aware of the fact that arithmetic operations on infinities may return false answers because `Erable` *can not check for nullity if non rational expressions are encoutered. For example* $(\sin(2x) - 2\sin(x)\cos(x))/0$ *will return* ∞ *instead of 0. This remark applies to the* `LIMIT` *instruction too (call* `SERIES` *if the answer of* `LIMIT` *seems false).*

10.4 Modular arithmetic

You must first store an integer n in the `MODULO` variable (by default n is set to 2). All computations are made modulo this integer. The `kernel` library provide the following commands `MODADD`, `MODSUBT`, `MODMULT`, `MODDIV`, `MODPOW`

and MODINV for the usual operations. MODADD, MODSUBT, MODMULT, MODDIV now accepts polynomials and Gauß integers as well as integers as input. MODPOW accept polynomial as first argument, the second argument must be a positive integer.

Remark for alg48 users: these functions are analog to AADD, ASUB, ... applied to mod-polynomials (like 'MOD(X^2+3*X+7,11)') but without MOD notation. I reserve the MOD notation for a future use with a polynomial as second argument so that it will be possible to compute e.g. in $\mathbb{Z}/3\mathbb{Z}[X]/(X^2+1)$.

11 Factorization. Solving equations.

11.1 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.
- LNCOLC: collect logarithms, e.g. $\ln(2)+\ln(3)$ LNCOLC returns $\ln(2\times 3)$.
- SOLV: tries to isolate a variable name at stack level 1 in the symbolic expression located at level 2. Returns the variable name at level 2 and a list of solutions at level 1. Examples:
  - 'X^4-1' X SOLV returns { 1 -1 'i' '-i' } at level 1 (and X at level 2)
  - '2*SIN(X)^2-3*SIN(X)+1' X SOLV returns

    ```
    { '-(12*pi*n2+-5*pi)/6' '(12*pi*n2+pi)/6'
    '-(4*pi*n2+-pi)/2' '(4*pi*n1+pi)/2' }
    ```

    where n1 and n2 represent integers.

  SOLV will be successful if the symbolic expression is a polynomial of a function of the variable or a product of polynomials. Otherwise it will fail. Hence, it may be useful to rewrite the symbolic expression using LNCOLC or/and COLC before calling SOLV. Example:
  'LN(X-1)+LN(X+2)=2' X SOLV doesn't work
  but 'LN(X-1)+LN(X+2)=2' LNCOLC X SOLV works

- FSIGN: returns the sign of a monovariate rational fraction as a list of signs + or - separated by roots or poles of the rational fraction, starting from $-\infty$ at the left tp $+\infty$ at the right of the list.
  Example:
  '(X^2-1)/(X^3-7*X^2+16*X-2)' FSIGN
  returns:
  'X': { - -1 + 1 - 2 - 3 + }
  hence $\frac{x^2-1}{x^3-7x^2+16x-2}$ is negative for $x\in(-\infty,-1)\cap(1,2)\cap(2,3)$, positive for $x\in(-1,1)\cap(3,+\infty)$, and zero or infinite for $x\in\{-1,1,2,3\}$.

- 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+2*X*Y+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.

(For the SX version only):
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.
(End of specificity of the SX version)

- `FACTO`: same stack as `FROOTS` 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)*(X-2)*(X-1)'`
`{ A -1 2 1 } -> '(X-2)/(X-A)'`

11.2 A word about factorization.

You should skip this section for a first reading. 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 degree 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 degree ≥ 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 reasonable 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 rational and quadratic irrationals roots of a univariate polynomial (unless the polynomial is badly 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 exhaustive search of all 2nd order (or greater) multivariate rational 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 user commands to factorize:

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

12 Linear algebra.

List of lists are used to represent symbolic matrices, in other words a symbolic matrix is entered like a numeric matrix, replacing [by { and] by }. Symbolic vectors are allowed as well (represented as lists).

12.1 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 I_n (if n is at level 1)
- `LCXM`: to build a matrix $A = (a_{ij})_{1 \leq i \leq l, 1 \leq j \leq c}$. The command 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 `HILBERT` return Vandermonde and Hilbert matrices given respectively a list of objects or an integer.

12.2 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 product of two vectors
- `cross`: cross product of two 3-d vectors.

12.3 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 useful to treat particular cases.
- `RANG`: like `rref` but creates 0 only under the diagonal.
- `det` and `RDET`: determinant (using respectively 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.

Note that all instructions using row reduction show intermediate steps if flag 1 is set (`1 SF`). If flag 1 is cleared (`1 CF`), you get directly the results.

12.3.1 Solving a linear system.

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

$$\begin{cases} mx & + & y & = & -2 \\ mx & + & (m-1)y & = & 2 \end{cases}$$

where m is a parameter. `SYST` instruction. Type a list containing the linear equations and as last element put the list of unknowns. Here:
`{ 'M*X+Y=-2' 'M*X+(M-1)*Y=2' { X Y } }`
Then call `SYST` or `SOLGEN`. For `SYST`, you get the solution at level 1, the list of particular cases at level 2 and the original system at level 3. For `SOLGEN` you get the same results but at level 2, 3 and 4 and the paramatrized solution at level 1.

On the above example, we get at level 1:
`:X:'-2/(M-2)' :Y:'4/(M-2)'`
At level 2, you get the list of pivots. The result returned by `SYST` and `SOLGEN` is incorrect if one of the pivot is 0. Here level 2 is:
`{ 'M^2-2*M' '-M+2' -1 'M+-2'}`
Using `M SOLV`, we see that we have to solve for the particular cases $m = 0$ and $m = 2$. The commands `SYST` and `SOLGEN` create a variable named `SYSTEM` to help solving particular cases. To solve for $m = 2$, recall `SYSTEM` on the stack, type `'M=2' EXEC`, and call `SYST`.

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 SOLGEN`
you get at level 2:
`If { }, { X Y }=:{ X -2 }`
(level 1 is the same as the result of `SYST`). 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).

Another way to solve the system is the enter the matrix of the system
`{ {M 1 -2} { M 'M-1' 2 } }`
and call `rref` to reduce it. You get at level 1:
`{ { 'M^2-2*M' 0 '-2*M' } { 0 'M-2' 4 } }`.
This means that:

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

The reduction is correct 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. To do this, you can use the variable named `MATRIX` (which is created if the argument contains at least one parameter). Recall this matrix 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.

12.3.2 Inversion

The `INVL` implements the Gauß method to invert matrices.

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

12.3.3 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)'
```

12.4 Kernel and image of a linear application.

To get the kernel of a linear application f with matrix A, enter the matrix A and type `KERN`. This will return the parametrized equations of the kernel like `SOLGEN`.

To get a basis of the image of f, enter the matrix A, type:
```
<< TRAN rref >>
```
the basis is made of the non-zeros lines of this matrix.

12.4.1 Other examples.

- LU decomposition example:

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

  `LU2` returns:

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

  We have $A = LU$.

- 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:

$$\begin{pmatrix} 1 & 2 & 4 & 6 \\ 0 & 5 & 9 & 13 \\ 0 & 0 & -13 & -16 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

the rank is 3, 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 $-2v_1 - v_2 + v_3 = 0$.

12.4.2 Stack input/output for reduction instructions.

Program	Input	Output
LU2	1: matrix $A = LU$	3: pivots, 2: L^-1, 1: U
RANG	1: matrix	2: pivots, 1: half-reduced matrix
rref	1: matrix	2: pivots, 1: `rref`-ed matrix
RDET	1: matrix	1:determinant
INVL	1: matrix	1: inverse
SYST	1: { equations { unknown } }	3: original system 2: list of pivots 1: result as a list of tagged algebraics
SOLGEN	1: { equations { unknown } }	4: original system 3: list of pivots 2: result, 1: list of tagged algebraics,

12.5 Diagonalization

The diagonalization instructions are:

- `MAD`: given a square matrix, returns determinant, formal inverse, a list polynomial and the characteristic polynomial. The list polynomial P_A is a matrix coefficient polynomail defined by the relation:

$$(xI_n - A)P_A(x) = M(x)I_n = M(x)I_n - M(A) \tag{2}$$

  where M denotes the characteristic polynomial of A.

- `PCAR`: characteristic polynomial using `det`
- `JORDAN`: compute eigenvalues and eigenvectors (cf. infra)

Given a square matrix A, `JORDAN` returns 6 levels:

- 6: $\det(A)^{-1}$
- 5: A^{-1}
- 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:

```
6: 0
5: { { inf inf inf }
     { inf inf inf }
     { inf inf inf } }
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 and has a 0 determinant.

2. For the identity matrix I_2 (`2 idn`), we get:

```
6: 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 from 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. An example with 1 parameter:

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

5. In dimension greater than 2, the factorization routines may fail. For this reason, you may have to 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:

```
6: -4
5: : { { '1/4' '1/4' '-1/4' }
       { '-3/4' '5/4' '-1/4' }
       { '-1/2' '1/2' '1/2'  } }
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 $(1, 1, 0)$.

Remark 8 *You can not use the current variable name as a parameter of a symbolic matrix that you want to diagonalize. This would lead to incorrect results. For example, is* `VX` *is set to* `X`*, you can not diagonalize the following matrix:*

```
{ { 1 1 X }
  { 1 X 1 }
  { X 1 1 } }
```

Workaround: make a change of variable, e.g. `'X=A' EXEC`.

12.6 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).

13 Multivariate analysis.

`Erable` implements the following functions:

- `der` with a list of variables at level 1 returns the gradient of the expression at level 2 with respect to these variables.
  Example:
  `'X+2*Y' { X Y } der` returns `{ 1 2 }`= $(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y})$.

- `DIV` returns the divergence of a list-vector at level 2 with respect to a list of variables at level 1.
  Example:
  `{ 'X+2*Y' 'X^2+3*Y^3'} { X Y } DIV` returns `'1+9*Y^2'`=$\frac{\partial f}{\partial x} + \frac{\partial f}{\partial y}$.

- `CURL` returns the rotationnal (same arguments as `DIV`)

- `LAPL` returns the laplacian of a symbolic expression at level 2 with respect to a list of variables at level 1 (same arguments as `der`, `LAPL` is simply a shortcut for `der DIV`)

- `HESS` returns at level 1 the hessian of a symbolic expression with respect to a list of variables (same arguments as `der`). Level 2 is the gradient. This is useful to find local minima and local maxima of a function: you find first the solutions of gradient=0 (you may use the Gröbner basis program of `ALG48` to simplify this system and use the `SOLV` instruction to find all solutions), then you compute with `EXEC` the hessian at these critical points, and you find the signature of the critical point using `GAUSS` (in the `other` directory: see section 14).
  Example:
  $$f(X,Y) = X^4 + XY + Y^3$$
  `{ X Y } HESS` returns at level 2:
  $$(4X^3 + Y, X + 3Y^2) = (\frac{\partial f}{\partial X}, \frac{\partial f}{\partial Y})$$
  and at level 1:
  $$\begin{pmatrix} 12X^2 & 1 \\ 1 & 6Y \end{pmatrix}$$
  To find critical points, you have to solve level 2=(0,0):
  $$(4X^3 + Y, X + 3Y^2) = (0,0)$$
  hence $X = -3Y^2$. Swap level 2 and 1, type `'X=-3*Y^2' EXEC`, then `1 GET` to have the first coordinate:
  $$4(-3Y^2)^3 + Y$$

then `Y SOLV`. This equation has two real solutions: 0 and approximately 0.392026340842 giving two critical points. For $(0,0)$, the hessian is:

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

hence $(0,0)$ is not an extremum (signature $(1,1)$). For the second point ($Y = 0.392026340842$, $X = -3Y^2$), the hessian is:

$$\begin{pmatrix} 1.84421582297 & 1 \\ 1 & -2.76632373445 \end{pmatrix}$$

hence is not an extremum.

14 Quadratic forms.

The main program is `GAUSS` (located in the `other` directory) 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 $A = P^tDP$ 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.

15 Customization and other utilities.

15.1 Data types.

Data handled by `Erable` have two representations: the user representation which you use most all the time and the internal representation (used internally).

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 } }`

15.2 User flags.

You may change the behavior of `Erable` by clearing or setting some user flags (and the system flag -27). Here are the most important flags of `Erable` (see section (D) for a complete list):

- 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 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 (e.g. flag 13), type `13 SF`. To clear this flag, type `13 CF`.

15.3 Conversions

- `AXL`: array $\leftrightarrow$ list conversion (transforms { } to [ ] and conversely).
- `EPSX0`: strip leading 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

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

- XNUM: convert level 1 to a numeric format like the build-in →NUM, but accepts lists (this was not the case on S/SX models). Clears flags 12, 14 and 15.

- XQ: convert level 1 to rational format, like the build-in →Q. Sets flags 12, 14 and 15.

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

- SXL (other directory, obsolete): Used for conversion to internal data type representation)

  - 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

- S2L (other directory, obsolete): 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 (other directory, obsolete): 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)'

15.4 Other functions

- `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`: fast Taylor development for polynomials:
  `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` [resp. `HERMITE` and `TCHEB`]: given an integer n, returns the n-th degree Legendre [resp. Hermite and Tchebycheff] polynomial.

- `PFEXEC`: execute the program at level 1 in subexpressions between all $+$ and $-$ of the symbolic expression at level 2. For example, try:
  `'1/2/(X^2-1)+1/4/(X^2-4)' << COLC >> PFEXEC`

- `XPRG`: explodes a program on the stack or creates a program with *stack level 1* components.

- `tEVAL` (`other` directory): 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`).

15.5 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 (returns $\sigma_{\text{level 2}} \circ \sigma_{\text{level 1}}$).

15.6 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 }`

15.7 Differential geometry

There is currently only one program available in the `other` directory, written in UserRPL by John Wilson, that I have translated into SysRPL, it is named `TNBA` for tangent, normal, binormal, acceleration. It takes a 3-dimensional vector (as a list) at level 1 (e.g. `{ '2*T' 1 'T^2' }`) and returns the position, tangent, normal and binormal vectors as well as the tangential and normal acceleration.

16 Final remarks.

Remaining things to do:

- adapt and integrate the polynomial division routines of `alg48` for speed,
- rewrite the Gauß reduction to include the Bareiss method,
- extend the Risch algorithm to multiples exponentials, and return an unevaluated integrals when there is no closed form,
- improve the factorization algorithm (Berlekamp method over $\mathbb{Z}[i]$).
- ...

I will 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` over the Saturn architecture is almost finished since we have probably reached the microprocessor limits. 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 Frequently asked questions

- When I call `SETUP`, I get the error `STO Error` with 0 at level 1. Why?
  You did not use `kermit` to download files to your HP48 and your software is case-insensitive, or your filesystem is case-insensitive and translated uppercase names to lowercase names. Please use `kermit`, and unzip the `Erable` archive over a case-sensitive filesystem or a filesystem where filenames are uppercase names. `MS-DOS` should work, as well as `ext2` (Linux), but not `VFAT` (`Windows 9x` filesystem under Linux).

- When I call `INIT`, I get the `RCL Error: Undefined Name` with `GXKEYS` at level 1 on the stack. Why?
  The reason is the same as above, but there is a simple workaround in this case. Type the following command line:
  `'gxkeys' DUP RCL 'GXKEYS' STO PURGE INIT`

- How can I launch the `eqstk` stack replacement?
  Type `ASTK`.

- `ASTK` does not launch `eqstk`. Why?
  If you want to shutdown your calc, you must press the right shift followed by the `ON` key. If you press the `ON` key too fast, this will stop `eqstk` unexpectedly and `ASTK` will not work.
  Workaround: type `ON` and `C` simultaneously, then again `ASTK`. If this doesn't work, you must reinstall `eqstk`.

- In previous versions of `Erable`, a `CST` menu was created. Is it gone?
  Yes, the user interface is now completely handled by the user key redefinition, because this is faster. You may create you own `CST` menu if you want by compiling the `SCST` file from the original package.

- How can I simplify $\sqrt{x^2}$?
  Type `28 SF` to simplify $\sqrt{x^2} = |x|$. Otherwise $\sqrt{x^2}$ remains not modified. This flag setting is not the default because setting flag 28 means that subexpressions are always simplified and this slow down most simplifications.

- How can I simplify $|x|$ if I know that $x > 0$?
  Type `29 SF`. If flag 29 is set, `Erable` will try to guess the sign of your expression: either at $x = 0^+$ or at $x = +\infty$ depending of the status of flag 24. The default is $x = +\infty$ but be aware that calling `SERIES` (directly or indirectly via `LIMIT`) sets flag 24 enabling guess at $x = 0^+$.

- Sometimes, `Erable` can not find the antiderivative of $\sin(x)$. Why?
  You are in complex mode. In this mode, `RISCH` does not recognize trigonometric functions, you must convert them to complex exponentials (`EXPLN`) or go back to real mode (`13 CF`).

- I have `alg48` installed, I would like to use the factorization routines of `alg48` inside `Erable`. Is it possible?
  Yes, but since I experienced many crashes, this feature is not included in the released versions. You can compile you own version and try different configurations to get this working. To do this, install the HP `GNU` tools and `erablsrc.zip`, uncompress it and uncomment in `kernel.s` and `erable.s` the lines between
  `*teste la presence d'ALG48` and `*fin de test ....`
  Type `compile` to compile the whole package, download to your HP and test!

B 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 in `HOME`, `algb` or `algbg`:

Name	Function	Arguments	Returns
EPS	ε		%
ERABLMSG	Risch log		string
INVLAP	Last inverse Laplace	nothing	s
MATRIX	Last matrix	nothing	m
MODULO	Arithmetic in $\mathbb{Z}/n\mathbb{Z}$		n
ODETYPE	Ordinary diff. equ. type		string
PRIMIT	Last primitive	nothing	s
SYSTEM	Last system	nothing	$\{mv\}$
GX/SX/UKEYS	User keys string	nothing	string
VX	integration variable	Rien	v
fr	French short doc	nothing	string
us	English short doc	nothing	string

If you are short in memory, you can erase all variables in { `HOME` } and subdirectories except `EPS`, `VX` and `MODULO`.

Functions of the `Erable` library and of the `other`, `algb` or `algbg` directories:

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	$p_2 \circ p_1$
CURL	Rotationnal	2: { s_1 s_2 s_3 } 1: { v }	{ s'_1 s'_2 s'_3 }
DEGRE	Order	{ p }	n
DIV	Divergence	2: { s_1 ... s_k } 1: { v }	s
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 }
DIVPC	Division in ascending power	3: s, 2: s', 1: n	s
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 to exp, ln	s	s
EXPLIN	Linearization of exp	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	Factorization	o	3: { v } 2: f , 1: { $s_1 n_1 s_2 n_2$ ... }
FSIGN	Sign of a rational fraction	s	tagged list
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
HALFTAN	To half angle tangent		
HERMITE	Hermite polynomial	integer n	H_n
HESS	Hessian	2: s, 1: { v }	matrix
HILBERT	Hilbert matrix	integer n	$n \times n$ matrix
HORN	Horner scheme	2:p , 1: r	3: $p/(X - r)$, 2: r, 1: $P(r)$
ILAP	Inverse laplace transform	s	$L^{-1}(s)$
INIT	Initialization	nothing	nothing
INVL	Inversion	o	o^{-1}
IPP	Integration by part	$\int_a^b f(t)dt$, u	$[uv]_a^b - \int_a^b uv'(t)dt$ ($v = f/u'$)
JORDAN	Diagonalization	endomorphism	7 to 1: *cf.* section 12
KERN	Kernel of a lin. appl.	m	4 to 1: *cf.* section 12
LAP	Laplace transform	2:f, 1:g	$L(f)/g$
LAPL	Laplacian	2: f, 1: { v }	Δf
LATEX	LaTeX conversion	1: s	1: string
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	Lin Diff Equ Cst Coef	2: { v }, 1: { m }	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	Limit	3:s, 2:v, 1:n	s
LNCOLC	Collect log	s	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 → cycles	p	3: p, 2: cycles, 1: signature
PCAR	Characteristic polynomial	endomorphism	s
PF	Partial fraction	f	$\sum_i f_i$
PFEXEC	exec between + and −	2:$\sum_i f_i$ 1: prg	$\sum_i \mathrm{prg}(f_i)$
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	$P(X-o)$
PURG	Purge algb(g)	nothing	nothing
QXA	s quadratic form to array	2: s, 1: { v }	{ m }
		s	2:{ m }, 1: { v }
RANG	Réduction sous-diagonale	{ m }	2: spec. cases, 1:{ m }
RDET	Determinant (rref)	endomorphism { m }	2: { m }, 1: determinant
RISCH	Symbolic integration	s	s
S2L	Symbolic to list	2: o, 1: { v }	2: { v },1:{ p }
		2: o, 1: v	{ p }
SCROLL	Scrolls a grob	grob	
SERIES	Series	3: s, 2: v, 1: n	6: 6-1: s
SETFR	Set French Flags	nothing	nothing
SIMP2	Simplification	2: o_2, 1: o_1	2: o'_2, 1: o'_1
SINCOS	Exponential to sine/cosine	s	s
SOLGEN	Solves a linear system	{ eqns { v } }	*cf.* section 12
SOLV	Solve	2: s, 1: x	2: x, 1: solutions
SQRT	Square root	n or C% or s	n or C% or s
STUDMULT	"students" × of matrices	M, M'	"MM' "
SUBT	Subtraction	2: o_2, 1: o_1	$o_2 - o_1$
SXL	Conversion	Internal [user]	User [internal]
SYST	Solves a linear system	{ eqns { v } }	*cf.* section 12
TAN2SC	Tangent to sin/cos		
TAN2SC2	Tangent to sin/cos2θ		
TCHEB	Polynomials	integer r	list of $r+1$ polynomials
TEXPA	Expand transcendent functions	s	s
TNBA	Tangent, normal, ...	{v}	
TR	trace	[ ] or { m }= $(a_{ij})_{1\le i,j\le n}$	$\sum_{i=1}^{n} a_{ii}$
TRAN	transposed	[ ] or { m }	[ ] ou { m }
TRIG	Trigonometry: $\to \sin, \cos, \arctan$	s	s
TRIGCOS	Trigonometry: $\sin^2 \to 1-\cos^2$	s	s
TRIGLIN	Trig. linearization	C%, s ou { p }	s
TRIGSIN	Trigonometry: $\sin^2 \to 1-\cos^2$	s	s
TRUNC	Truncate an asymptotic expansion	2: s, 1: rest s'	s
TSIMP	Simplification (transcendental)	s	s
VAND	Vandermonde matrix	list of objects	matrix
VER	Version	rien	% 2.99
XFRC	To quadratic irrational	o	o
XNUM	→ Numeric	o	o
XQ	→ Rational	o	o
XY	Scalar product of 2 vectors	2: x 1: y	$x.y$
abs	Absolute value	s	s
add	Addition	2: o_2, 1: o_1	$o_2 + o_1$
arg	Argument	1: s	1: s
comb	Combinaisons	2: n, 1: n'	$C_{\overline{o}}^{n'}$
conj	Conjugate	o	
cross	Wedge product	2: x, 1: y	$x \wedge y$
der	derivative or gradient	2: s, 1: v	1: s
der1	derivative	s	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
tEVAL	Execution time	..., 1: o	EVAL(o), 1: time

Functions of the `kernel` library. Don't forget to set an integer n in the variable `MODULO` (by default $n = 2$):

Name	Function	Arguments	Returns
`{KERNEL.LIB}`	(`0:788`)		
MODADD	Modular addition	2: n_1, 1:n_2	$(n_1 + n_2) \bmod n$
MODSUBT	Modular subtraction	2: n_1, 1:n_2	$(n_1 - n_2) \bmod n$
MODMULT	Modular multiplicatin	2: n_1, 1:n_2	$(n_1 * n_2) \bmod n$
MODDIV	Modular division	2: n_1, 1:n_2	$(n_1/n_2) \bmod n$
MODPOW	Modular power	2: n_1, 1:n_2	$n_1^{n_2} \bmod n$
MODINV	Modular inversion	1: n_1	$n_1^{-1} \bmod n$
PA2B2	Prime factorization	1: p ($p \equiv 1[4]$)	1: $a + ib/\ a^2 + b^2 = p$

C User Keys.

From top left corner to bottom right corner, α - Right Shift -ed keys:

Princ. Key	Reminder	Function
MTH		(normal MTH)
PRG		(normal PRG)
CST	MODES	(Erable config, for GX only)
EVAL		EXEC
SIN	∂	der1
COS	$\int$	RISCH
$\sqrt{}$	$\sqrt{}$	SQRT
y^x	y^x	POWER
$1/x$	$1/x$	INVL
$\pm$	pm	CHS
DEL		∞
7	(solve, factor)	
8	(exp and ln)	
9	(diff. calc.)	
$\div$	$\div$	DIV1
4	(matrices)	
5	(conversions)	
6	(arithmetic)	
$\times$	$\times$	MULT
1	(basic algebra)	
2	(complex)	
3	(trigonometry)	
-	-	SUBT
SPC		EXPA
+	+	add

Other redefined keys:

- G/GX only: `MTH` key (runs `Erable` main menu)

- G/GX only: `PRG` key (runs a HP48GX main menu)
- `CST` key: i
- `DEL` key: `X`
- right shift-`DEL` key: $-\infty$
- downarrow key (35.1): calls the Miniwriter. Type:
  `35.1 DELKEYS`
  if you do not have the `Miniwriter` installed. If you have `JAZZ` or `TEDVV` instead, you may redefine this keystroke by typing:
  `{ TED 35.1 } STOKEYS`
- left shift-downarrow (35.2): calls `AGROB` followed by `SCROLL`. Needs `eqstk` or `JAVA`. Type:
  `35.2 DELKEYS`
  if you do not have `eqstk` or `JAVA`
- S/SX only: 33.2 (`XQ`) and 33.3 (`XNUM`)
- G/GX only: 33.2 (`XNUM`) and 35.6 (`XQ`).

D User flags

List of the flags used by `Erable` (the sign * after the flag number means that the flag is cleared if `VER` is called, # means that the flag is set if `VER`is called):

- 01: if set then verbose mode (details of some algorithms are shown) else quiet mode.
- 10: 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 (hence algebraics are not simplified and multiple roots of polynomials are not detected)
- # 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 matrix 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 $x \to +\infty$.
- * 25: if set then the rule $\sqrt{x}^2 = x$ is not applied
- * 26: if set then `TRIG` tries to return only sines, otherwise it returns cosines

- * 28: determines whether embedded quotients are immediatly simplified or not (for example $2/4 + 3/6$ may be simplified first to $1/2 + 1/2$ and then to 1 or directly to $24/24 = 1$).
- * 29: if set then $|x|$ is simplified to x for every "variable" (as returned by `LVAR`)

You should only modify user flags 1,10,13,15 and 21 to 29.

Remark 9 *System flag 27 (`-27`) affects the way symbolic complex numbers are displayed. System flag 2 affects the way symbolic constants are evaluated (for example π is returned as a symbolic constant or as a numeric approximation). This flag is set or cleared by* `Erable` *according to the current mode of* `Erable` *(numeric or symbolic).*

E Error codes for the SERIES command.

- 1: can not determine series expansion for $\arctan(x)$ function with current argument x.
- 2: $\arcsin(x)$ not defined for infinite argument.
- 3: no series expansion for $\exp(x)$ at $x = \infty$ if sign is unknown.
- 4, 5, 6, 7: failed to compare 2 variables.
- 8: can not determine order for the current rest.
- 9: negative argument for logarithm function.
- 10 (`<Ah>`): insufficent order. (You can try again with a larger order)
- 11 (`<Bh>`): can not find sign of argument of `ABS` function
- 16 (`<10h>`): numeric input are not allowed

F Thanks to ...

Many people helped me during the creation and distribution 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 Maurice Al-Khaliedy, Christophe Burdin, Craig Clifford, Jerome Coss, David Czinczenheim, Ludovic Dumaine, Eduardo (`maciasval@mx2.redestb.es`), Frederic Hermann, Eric Gorka, 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.

- Some math teachers, particularly Renée de Graeve and Scott Guth who made tests, suggestions and bug reports.
- 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`, `JAVA` stack displays ([7], [17]), the `TED` and `Miniwriter` editors ([13], [1]), the `JAZZ` debugger ([12]), the `Metakernel` ([14]), various compilers (`JAZZ`, the `HP` tools ([2]), the `RPL` based tools ([16]) and eventually the GNU `tools` ([15]).
- I looked at the following book and softwares: [8], [3], [6], [4], [5], [10], [11], [9] . One of the best reference is certainly [4] and references therein. M. Heiskanen's WWW-homespage has a lot of interesting math links.

References

[1] J.-Y. Avenard. `miniwriter`. `http://www.epita.fr/~avenar_j`, 1997.

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

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

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

[5] J. Ferrard. *"Mathez" la HP 48 G/GX*. D31 Diffusion (HP48), 1993.

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

[7] C. N. Fiechter and M. Heiskanen. `EQSTK92.ZIP`. `http://www.hut.fi/~mheiskan`, 1997.

[8] C. N. Fiechter and M. Heiskanen. `ALG48V42.ZIP`. `http://www.cs.pitt.edu/~fiechter/hp48` `http://www.hut.fi/~mheiskan`, 1998.

[9] B. Fuchssteiner. `MuPAD`. `ftp://ftp.inria.fr/lang/MuPAD` `http://www.mupad.de`, 1998.

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

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

[12] M. Heiskanen. `JAZZV65.ZIP`. `http://www.hut.fi/~mheiskan`, 1996.

[13] M. Heiskanen. `TED31.ZIP`. `http://www.hut.fi/~mheiskan`, 1997.

[14] Maubert Development Group. `Metakernel, 48+`. `http://www.epita.fr/~avenar_j`, 1998.

[15] M. Mikocevic. `GNUTOOLS`. `ftp://srcm1.zems.fer.hr:/pub/hp48/tools2.1.9.zip`, 1995.

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

[17] R. Steventon, A. Schoorl, and W. Laughlin. `JAVA34.ZIP`. `ftp://ftp.cis.com/pub/hp48g/uploads/java34.zip` `http://www.engr.uvic.ca/~aschoorl`, 1998.