COMET

Programming Environment for Numerical Computing

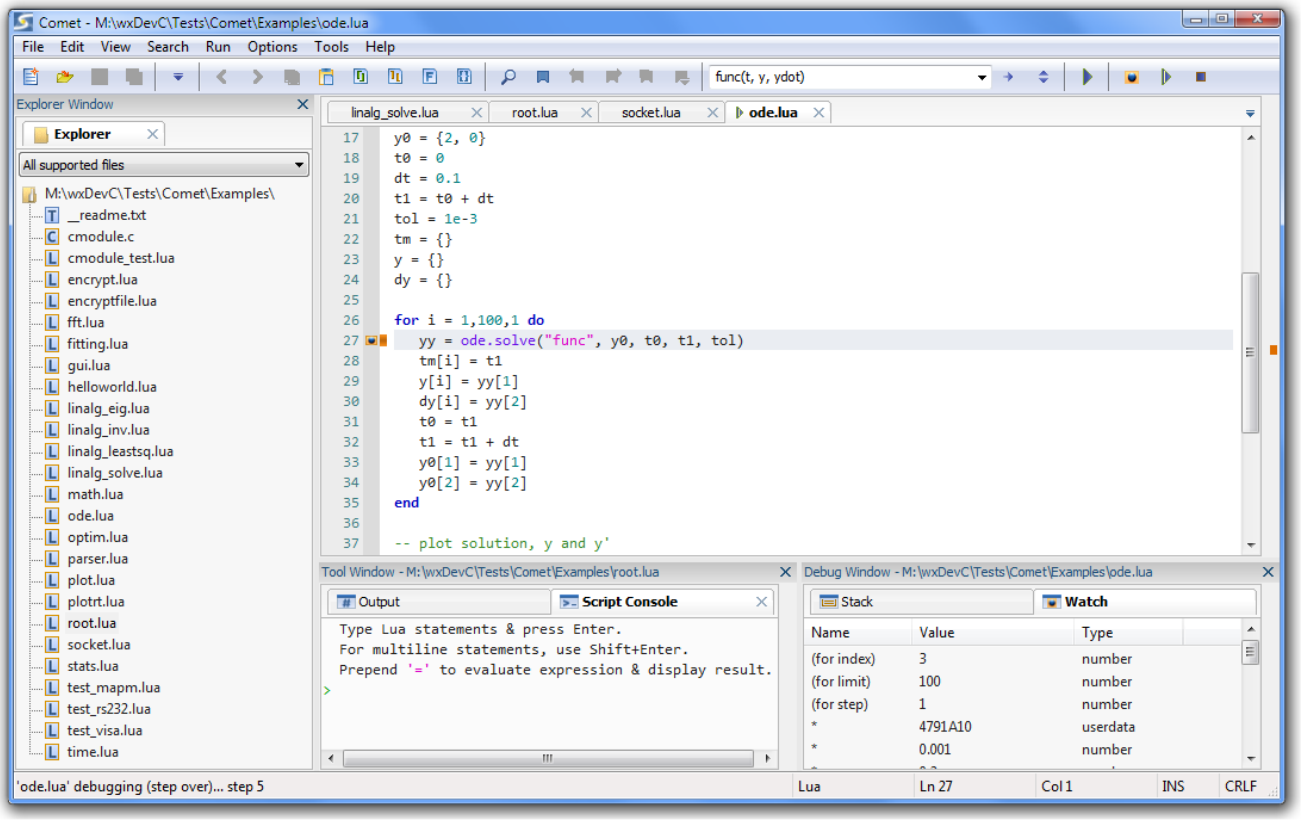
http://www.hamady.org

Version 1.8 Build 2104

Contents

1. INTRODUCTION

Comet is a programming environment for numerical computing and data analysis using the Lua scripting language. It is available for Android, Linux and Windows with built-in Lua scripting engine, integrated numerical, data plotting and analysis modules, full-featured editor with syntax highlighting, code completion, inline documentation, script console, Lua samples and code snippets, debugging capabilities, etc.



The main goal of Comet is to provide an **easy-to-use development environment for numerical computing using the Lua programming language on Android, Linux and Windows**. It integrates the Lua scripting engine with all Lua functionalities, and a lot of specific Comet functionalities including **numerical functions** (linear algebra, differential equations, mathematical optimization, etc.), **data plotting** and **analysis** modules and an **extended mathematical library**. Within the Comet environment, you can develop algorithms for science and engineering with one of the most elegant and fast scripting languages.

To learn the Lua programming language, you can visit the Lua official website: http://www.lua.org

Excellent tutorials, covering all Lua aspect from basics to advanced programming techniques, can be found here: http://lua-users.org/wiki/TutorialDirectory

Comet can also be used as a **general purpose full-featured editor supporting C/C++, Bash/Text, Python, Octave, Fortran, LaTeX and Makefile** with configurable tools (e.g. to run a compiler or a bash script).

2. INSTALL

- For **portable version** on **Windows** and **Linux**: Download ***comet_windows_64bit.zip*** (Windows 64bit) or ***comet_windows_32bit.zip*** (Windows 32bit) or ***comet_linux_64bit.tgz*** (Linux 64bit) or ***comet_linux_32bit.tgz*** (Linux 32bit) from http://www.hamady.org, unzip/untar in any location (USB key or memory stick for example) and run *comet.exe* (Windows) or *comet* (Linux) in the bin directory.
- For **setup version** (on **Windows 64bit**): Download ***comet_windows_64bit.msi*** from http://www.hamady.org, double-click on the setup file and it will install Comet.

If you need only the Lua editor, without the numerical and visualization modules, you can download Comet N, the Comet stripped version:

For portable version (64bit): Download:

cometn_windows_64bit.zip (Windows 64bit)

cometn_linux_64bit.tgz (Linux 64bit)

unzip/untar in any location and run comet.exe or comet in the bin directory.

For 32bit portable version: Download:

cometn_windows_32bit.zip (for Windows 32bit)

cometn_linux_32bit.tgz (for Linux 32bit)

Under Ubuntu/Debian 64bit, if you encounter error such as:

./comet: error while loading shared libraries: libpng12.so.0

Install libpng12 as follows:

cd /tmp/

wget http://mirrors.kernel.org/ubuntu/pool/main/libp/libpng/libpng12-0_1.2.54-1ubuntu1_amd64.deb

sudo dpkg -i . /libpng12.deb

sudo rm . /libpng12.deb

If you encounter (in Ubuntu/Debian 64bit) error such as:

Failed to load module "canberra-gtk-module"

Install libcanberra-gtk-module as follows:

sudo apt install libcanberra-gtk-module libcanberra-gtk3-module

- For **Android**: Install from Google Play.

To know if a new version is available, click *Menu/Help/Check for Update...* or visit my website: http://www.hamady.org

3. QUICK START GUIDE

With the Comet toolbar, you can open, save, navigate in the document with bookmarks, run/debug script.

You can perform the same actions, and more, with the menu.

Comet includes code examples and snippets (*Help/Examples* and *Edit/Insert Snippet* menu). With code snippets, you can easily insert some pieces of code in your script for better productivity.

For Android: If you want to change the font size, select menu/Settings. You can create a new script or save the current script to another location, by selecting menu. If you want to automatically run a script on startup (AutoRun), select menu/Settings/AutoRun and choose your script in the file explorer. If you long click the AutoRun button in menu/Settings, the AutoRun feature will be disabled. You can enable or disable the soft keyboard by selecting menu/Keyboard.

You can print text on the output area by using the Lua standard print or *io.write* function. To clear the output area, use the *cls()* function or select *Clear* in the context menu.

For Android: You can set the output area height by selecting S (small), M (median) or L (large) in Menu / Settings (Output).
As for Windows and Linux, Comet for Android has two graphical themes: light and dark, and syntax highlighting functionality. You can switch between themes, enable or disable syntax highlighting in menu/Settings.

- ✓ You can create a new script or save the current script to another location, by selecting ***File****/New* (*Menu/New* on Android) or *File/Save As* (*Menu/Save As* on Android) menu or the toolbar corresponding buttons.
- ✓ You can **print text** on the output area by using the Lua standard **print** function (or the **io.write** function).To clear the output area, use the **cls()** function.
- ✓ You can set **bookmark** in you code and then navigate easily by clicking click Add/Remove Bookmark, Next or Previous Bookmark (either Menu/Search or the correspond toolbar buttons). Bookmarks are saved with your document and are restored next time you open it. The list of all active bookmarks can be displayed by selecting *View/Bookmarks* in the menu. The bookmarks, modification and search markers can be saved (or not) alongside the document by checking (or unchecking) the corresponding option: *Options/Save Navigation Info* menu.
- ✓ The **editor options** (Font, Syntax highlighting, AutoCompletion, Call Tips, Colors, Indentation and long lines indicator, Folding …) can be easily modified in the *Options* menu.
- ✓ You can **view** end of lines, white spaces, file explorer, output window, script console, tool bar, status bar … in the *View* menu.
- ✓ The **editor toolbar** includes, in addition to the commonly used commands, an integrated edit zone to (i) navigate through the current document functions (or sections for LaTeX): just drop down the list and select a function; (ii) to search in the document: just type a text and click the toolbar button to the right (or press ENTER); (iii) go to a line: just type the line number and click the toolbar button to the right (or press ENTER). Note that the document functions list can be manually updated by

clicking the *'Update List'* toolbar button (this list is created when the document is loaded or executed). To reset the list, right click in it and choose 'Reset' in the popup menu.

- ✓ The **editor infobar** on the right border includes the main navigation markers (modification, find results, bookmarks …), useful for long documents. Click on a marker in the infobar to go to the corresponding line. The infobar can be enabled or disabled by clicking *Options/View Infobar Markers* menu.
- ✓ Comet's **execution mode** is designed to perform long calculations, without user interaction. This feature makes it suitable for math and algorithms development.

For Android: you can activate the long calculations mode, which set the device processor to stay awake during script execution, by selecting Menu/Settings.

Comet can be executed from the command line:

```
$ comet -run input [-out outfile] [-show]
```

input may be a filename or a double-quoted expression and **outfile** is the output filename:

```
$ comet -run "print(sin(math.pi/4))"
$ comet -run cometin.lua -out cometout.txt
```

Note that the plot functions and other GUI functions are only available in the Comet GUI.

- ✓ **Comet** integrates a **Lua script console** (Menu View / Script Console). You can use it as an interactive programming console to perform quick tests for example. To use the script console, simply write a Lua valid statements (example: `x = math.pi/3; y = sin(x); print(y);`), and press ENTER. You can enter multi-line statements: just press SHIFT+ENTER after each line and ENTER to run. You can use any standard Lua function or Comet module.

First example:

Type the following script in the editor...

```
-- Lua first script
cls()
a = math.cos(math.pi/4)
print("a = ", a)
```

... and click Run (or press F12 on Windows and Linux).

4. COMET AS A GENERAL-PURPOSE EDITOR

Comet can be used as a general purpose **full-featured editor supporting C/C++, Bash/Text, Python, Octave, Fortran, LaTeX and Makefile** with configurable tools (e.g. to run a compiler or a bash script). The document language is automatically selected using the file extension. For unknown extension, you can manually select the document language in *File/Current File/File Type* menu. To configure the tools used, for example, to compile and execute code, select *Options/External Tools* menu. Comet supports **configurable tools for C/C++, Python, LaTeX and Makefile**.

To configure a specific tool, select menu *Options/External Tools:*

(i) For Windows, you can create a batch file (named ***build.bat*** for example), put the command in this file and add the batch command ***call build.bat*** in the *External Tools* dialog after selecting the corresponding file type (example: ***LaTeX***). The following example gives a typical ***build.bat*** content for building LaTeX document under Windows using MiKTeX and the Sumatra PDF viewer:

```
@echo off
@del /f /q myreport.pdf >nul 2>&1
latexmk -pdf -pvc- -halt-on-error myreport.tex
if %errorlevel% equ 0 (start "" "SumatraPDF.exe" myreport.pdf) else (exit /b
1)
```

(ii) For Linux, similarly you can create a bash file (named ***build.sh*** for example), put the command in this file (example: `pdflatex -halt-on-error -file-line-error mydoc.tex`) and add ***./build.sh*** in the *External Tools* dialog after selecting the corresponding file type (example: ***LaTeX***).

(iii) For Python (for both Windows or Linux), just put a command like this (replace with your installed Python interpreter): ***C:\Python\python.exe –u %s.py*** (Windows) or ***python -u %s.py*** (Linux).

Comet includes a **user-defined syntax highlighting template** that can be customized for a specific language or file type: click *Options/User-defined Syntax* menu, enter the corresponding file extension (without dot) and the space-separated list of keywords to highlight. You can save the user-defined syntax highlighting template to a file (button *Save…*) or load the template from a file (button *Load…*).

5. GUI FUNCTIONS

To interact graphically with the user, one can use the GUI functions:

```
gui.alert(msg)              -- show an alert dialog box
answer = gui.ask(msg)       -- display Yes-No dialog.
                            -- Returns 1 if Yes selected, 0 otherwise
```

Example:

```
-- GUI
answer = gui.ask("Save calculation results?")
if answer == 1 then
    -- …
    gui.alert("Calculation results saved.")
end
```

6. TIME FUNCTIONS

```
time.tic()          -- sets the wall-clock timer
time.toc()          -- returns the number of milliseconds elapsed
                    -- since the last tic() call
time.sleep(n)       -- sleeps for n milliseconds
time.format(n)      -- formats a duration (integer in milliseconds)
                    -- into string
```

Example:

```
-- Time
time.tic()
time.sleep(200)
dt = time.toc()
print(time.format(dt))
```

7. MATH FUNCTIONS

With Comet, the math functions are mapped to global functions (e.g.: you can use **cos** or **math.cos**).

Example:

```
-- Math
cls()
a = cos(pi/4)
print("result = ", a)
```

Below is a summary of the Lua math functions:

math.abs	**math.acos**	**math.asin**
math.atan	**math.atan2**	**math.ceil**
math.cos	**math.cosh**	**math.deg**
math.exp	**math.floor**	**math.fmod**
math.frexp	**math.huge**	**math.ldexp**
math.log	**math.log10**	**math.max**
math.min	**math.modf**	**math.pi**
math.pow	**math.rad**	**math.random**
math.randomseed	**math.sin**	**math.sinh**
math.sqrt	**math.tanh**	**math.tan**

And a summary of the Comet global math functions:

abs	acos	asin
atan	atan2	ceil
cos	cosh	deg
exp	floor	fmod
frexp	huge	ldexp
log	log10	max
min	modf	pi
pow	rad	random
randomseed	sin	sinh
sqrt	tanh	tan

NB: Lua gives the neperian logarithm the name `log` and the decimal logarithm is named `log10`, as in C language.

8. EXTENDED MATH FUNCTIONS

Special math functions are added to Comet (**math** namespace), including and extending the Lua math functions.

Summary of **math** additional functions and constants:

Functions:

```
math.exp2(x)          -- 2^x
math.logb(x)          -- log base 2 of x
math.cbrt(x)          -- cubic root
math.hypot(x,y)       -- sqrt(x²+y²)
math.erf(x)           -- error function
math.erfc(x)          -- complementary error function
math.lgamma(x)        -- ln(gamma(x))
math.tgamma(x)        -- gamma(x)
math.trunc(x)         -- nearest integer
math.round(x)         -- nearest integer, rounding
math.isinf(x)         -- number is infinite?
math.isnan(x)         -- not a number?
math.isnormal(x)      -- number is normal?
math.asinh(x)
math.acosh(x)
math.atanh(x)
math.gauss(x,b,c)     -- G(x) = exp(-(x - b)² / 2c²)
math.lorentz(x,b,c)   -- L(x) = (c / ((x - b)² + c²))
```

Constants:

Universal constants in international units (SI)

```
math.q                -- Electron charge (in C)
math.me               -- Electron mass (kg)
math.kb               -- Boltzmann constant (J/K)
math.h                -- Planck constant (Js)
math.c                -- Speed of Light in vacuum (m/s)
math.na               -- Avogadro constant (1/mole)
```

9. LINEAR ALGEBRA FUNCTIONS

Comet uses the well-established Lapack library for linear algebra calculations. The functions can be called through the **linalg** namespace. Comet uses tables for matrix and vector with the **row-major order**: The 3x3 matrix {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}} is represented as the table **{1, 2, 3, 4, 5, 6, 7, 8, 9}**.

Note: the linear algebra module is not available for Android.

Below is a summary of the **linalg** functions:

linalg.add(x,y,a,b)	y = a*x + b*y where a, b are scalars and x,y tables. The input y is modified.
x = linalg.array(n,a)	Create n-size array and init its values to a
z = linalg.cat(x,y)	Concatenate x and y arrays
linalg.copy(x,y)	Copy x to y
d = linalg.det(A)	Matrix determinant
d = linalg.dot(x,y)	d = x' * y
wr, wi, vl = linalg.eig(A)	Eigenvalues and eigenvetors of matrix A wr: vector with real part of eigenvalues wi: vector with imaginary part of eigenvalues vl: matrix with the eigenvetors as columns
str = linalg.format(A,m)	Format matrix with m rows into string
y = linalg.get(x,is,ie)	Returns subarray of x from index is to ie
im = linalg.max(x)	Index of the max value in x
im = linalg.min(x)	Index of the min value in x
B = linalg.inv(A)	Returns the inverse matrix
x = linalg.leastsq(A,b,m)	Solves rectangular system Ax = b with m rows for b and therefore m columns for A.
z = linalg.mult(x,y,m)	z = x*y with m rows
r = linalg.norm(x)	Norm-2 of x
A = linalg.ones(n)	Returns an n-size array of 1
A = linalg.rand(n)	Returns an n-size array of random values
xr, yr = linalg.rot(x,y,c,s)	Plane rotation of x,y vectors with: given cosine c and sine s Returns the rotated vectors
x = linalg.solve(A,b)	Solve the square linear system Ax = b
x,iters = linalg.sparse(A,b,x0,tol, iters)	Solve iteratively the square system Ax = b Returns the solution and iterations #

s = linalg.sum(x)	Absolute sum of x
linalg.swap(x,y)	Swap x et y
t = linalg.trace(A)	Calculate matrix trace
x = linalg.tridiag(down,diag,up, b)	Solve tridiagonal system with: the main, upper and lower diagonals
B = linalg.trans(A,m)	Transpose matrix with m rows
A = linalg.zeros(n)	Return an n-size array of 0

Example 1:

```
cls()
print "\nlinalg.solve\n"
A = { 3, 2, -1,   2, -2, 4,   -1, 0.5, -1 }
b = { 1, -2, 0 }
x = linalg.solve(A,b)
print("x = " .. linalg.format(x,3))
```

Example 2:

```
cls()
print "\nlinalg.inv\n"
A = { 1, 2, 3,   0, 1, 4,   5, 6, 0 }
iA = linalg.inv(A)
print("iA = " .. linalg.format(iA,3))
```

Example 3:

```
cls()
print "\nlinalg.eig\n"
A = { 2, 0, 0,   0, 3, 4,   0, 4, 9 }
wr, wi, vl = linalg.eig(A)
print("wr = " .. linalg.format(wr,3))
print("vl = " .. linalg.format(vl,3))
```

Example 4:

```
cls()
print "\nlinalg.leastsq\n"

-- A with 6 columns
A = { 1.44, -7.84, -4.39,  4.53, -9.96, -0.28,
     -3.24,  3.83, -7.55,  3.24,  6.27, -6.64,
      8.34,  8.09,  5.28,  2.06,  7.08,  2.52,
      0.74, -2.47, -5.45, -5.70, -1.19,  4.70 }

-- b with 6 rows
b = { 8.58,
      8.26,
      8.48,
     -5.28,
      5.72,
      8.93 }

-- x will have 4 rows (as for matrix A)
-- x should be { -0.450637,  -0.84915,  0.706612,  0.128886 }
x = linalg.leastsq(A,b,6)

print("x = " .. linalg.format(x,4))
```

10. ORDINARY DIFFERENTIAL EQUATIONS

With the Comet ODE solver can integrate differential system given: the system functions, the initial values and the independent variable value (x, t, ...). It is not necessary to provide the Jacobian which is approximated by the solver.

The ODE solver:

```
y = ode.solve(func, y0, t0, t1, tol)
```

Integrates ODE system, where:

func: name of the ODE system function.

func defined as func(t, y, ydot) where:

ydot vector updated with respect to y and t:

$$\begin{pmatrix} y_1^{(n)} \\ y_2^{(n)} \\ \dots \\ y_m^{(n)} \end{pmatrix} = \begin{pmatrix} F_1(t, y, y^{(1)}, y^{(2)}, \dots, y^{(n-1)}) \\ F_2(t, y, y^{(1)}, y^{(2)}, \dots, y^{(n-1)}) \\ \dots \\ F_m(t, y, y^{(1)}, y^{(2)}, \dots, y^{(n-1)}) \end{pmatrix}$$

y0 table containing the initial values

t0 value for y0

t1 value to integrate for

tol the solver tolerance (optional)

Example:

```
-- Damped Oscillator: y'' + c y' + k y = 0
local c = 0.5
local k = 1
function func(t, y, ydot)
    ydot[1] = y[2]
    ydot[2] = (-c * y[2]) + (-k * y[1])
end

-- Solve with 0.1 seconds as interval
y0 = {2, 0}
t0 = 0
dt = 0.1
t1 = t0 + dt
tol = 1e-3
tm = {}
y = {}
dy = {}
for i = 1,100,1 do
    yy = ode.solve("func", y0, t0, t1, tol)
    tm[i] = t1
    y[i] = yy[1]
    dy[i] = yy[2]
    t0 = t1
    t1 = t1 + dt
    y0[1] = yy[1]
    y0[2] = yy[2]
end

-- plot solution y and y'
p = plot.new(800, 600)
plot.set(p, "title", "Ordinary Differential Equation")
plot.add(p, tm, y)
plot.add(p, tm, dy)
plot.set(p, "xlabel", "time (s)")
plot.set(p, "ylabel", "amplitude")
plot.set(p, 1, "legend", "y(t)")
plot.set(p, 1, "style", "-")
plot.set(p, 1, "color", "red")
plot.set(p, 2, "legend", "y'(t)")
plot.set(p, 2, "style", "-")
plot.set(p, 2, "color", "blue")
plot.update(p)
```

11. MATHEMATICAL OPTIMIZATION

Comet integrates a mathematical optimization module including minimization of real function of n variables. The Comet minimization function uses the Hooke and Jeeves algorithm which do not require the jacobian to be evaluated.

iters = optim.minimize(func, pars, maxiters, tol, rho)

func: name of the function of n variables to be minimized.
func defined as func(x) where:
x vector containing the variables

pars table containing the initial values (will be updated to the calculated values)
maxiters maximum number of iterations (optional)
tol the algorithm tolerance (optional)
rho the algorithm parameter, between 0 and 1 (optional). Decrease rho to improve speed and increase it for better convergence.

xr = optim.root(func,a,b,iters,tol)

-- Finds the root of function in the [a,b] interval (optional), the given maximum number of iterations (optional) and.tolerance (optional).
Returns the root value.

Examples:

```
-- Optimization
function booth(x)
   return (math.pow(x[1] + 2*x[2] - 4, 2) + math.pow(2*x[1] + x[2] - 5, 2))
end

x = { -5, 5 }
iters = optim.minimize("booth", x, 100, 1e-6)

cls()
io.write(string.format("\n x = [%g %g]  iters = %d \n", x[1], x[2], iters))
```

```
function func(x)
   return x^2 - x - 1
end
x = optim.root("func",-5,5)     -- expected: x ~ -0.618034
print(x, func(x))
```

12. DATA ANALYSIS

12.1 DESCRIPTIVE STATISTICS

The **data** namespace includes functions to calculate the descriptive parameters of a list of values: Minimum (**data.min(t)**), Maximum (**data.max(t)**), Sum (**data.sum(t)**), Mean (**data.mean(t)**), Median (**data.median(t)**), Variance (**data.var(t)**), Standard Deviation (**data.dev(t)**), Coefficient of Variation (**data.coeff(t)**), Root Mean Square (**data.rms(t)**), Skewness (**data.skew(t)**) and Kurtosis excess (**data.kurt(t)**).

The formulas used are as below:

Mean	**Variance**
$\mu = \frac{1}{N}\sum_{i=0}^{N-1} x_i$	$\sigma^2 = \frac{1}{N-1}\sum_{i=0}^{N-1}(x_i-\mu)^2$
Skewness	**Kurtosis**
$Skew = \left(\frac{1}{N}\sum_{i=0}^{N-1}(x_i-\mu)^3\right) / \left(\frac{1}{N}\sum_{i=0}^{N-1}(x_i-\mu)^2\right)^{3/2}$	$Kurt = \left[\left(\frac{1}{N}\sum_{i=0}^{N-1}(x_i-\mu)^4\right) / \left(\frac{1}{N}\sum_{i=0}^{N-1}(x_i-\mu)^2\right)^2\right] - 3$

All the stats functions take a Lua table as argument, with 2048 maximum number of elements.

Example:

```
-- Stats
cls()
t = {1,1,2,3,4,4,5}
m = data.mean(t)        -- expected: m = 2.8571428571429
print(m)
```

12.2 DATA ANALYSIS

The data namespace includes also functions to perform data analysis including fitting using user-defined model, FFT and autocorrelation calculations:

```
pars, chi, iters, str = data.fit(func, tx, ty, fpar, ipar, tol, iters)
```

Runs the fitter algorithm with:

func the Lua model function name. The Lua function syntax is as following (replace with your own model):

```
function fitfun(fpar, dpar, x)
     dpar[1] = 1
     dpar[2] = x
     y = fpar[1] + fpar[2]*x
     return y
end
fpar        -- is the fitting parameters table.
dpar        -- is the table of partial derivatives.
x           -- independent variable.
```

```
tx          -- the table with X data.
ty          -- the table with Y data.
fpar        -- is the fitting parameters table.
ipar        -- table containing, for each parameter, value 1 if the
            -- parameter is varying or 0 if it is fixed.
            -- this parameter ipar is optional.
tol         -- the relative tolerance to be reached.
            -- this parameter tol is optional.
iters       -- the maximum number of iterations for the fitting algorithm.
            -- this parameter iters is optional.
```

The function **data.fit** returns four parameters: the obtained parameters table **pars** ; the **chi** number ; the number of performed iterations **iters** and a message **str** from the fitter engine.

Example of using the Fitter:

```
-- Linear Fit
cls()
function func(fpar, dpar, x)
    dpar[1] = 1
    dpar[2] = x
    y = fpar[1] + fpar[2] * x
    return y
end

tx = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
ty = {0.1, 0.8, 2.2, 3.1, 3.8, 5.1, 5.9, 7.1, 8.0, 9.2}
fpar = {3,3}
ipar = {1,1}
pars, chi, iters = data.fit("func",tx,ty,fpar,ipar,1e-3,100)
io.write(string.format("pars = [%g %g]\n", pars[1], pars[2]))

p = plot.new(800,600)
plot.set(p, "title", "Linear Fitting")
txf = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
tyf = {}
for i = 1, #txf, 1 do
   tyf[i] = pars[1] + pars[2]*txf[i]
end
plot.add(p, txf, tyf)      -- plot linear fit
plot.add(p, tx, ty)        -- plot data
plot.set(p, 1, "color", "red")
plot.set(p, 2, "style", "o")
plot.update(p)
```

ft = data.fft(data, idir)

Calculates the FFT with:

data -- the table with data

idir -- 1 for forward FFT and 0 for inverse

The function **data.fft** returns the obtained FFT table **ft**.

NB: the FFT amplitude is scaled (divided) by the number of points.

Example of using **data.fft**:

```
-- FFT
cls()

Fo = 50                    -- signal frequency (Hz)
To = 1/Fo                  -- signal period (seconds)
A = 5                      -- signal amplitude
An = 1                     -- noise amplitude
N = 256                    -- number of points (power of 2)
Ts = 4 * To/N              -- sampling period
Fs = 1/Ts                  -- sampling frequency
f = {}
t = {}
y = {}
for i = 1, N, 1 do
    f[i] = (i - 1) * Fs / (N - 1)     -- frequency
    t[i] = (i - 1) * Ts               -- time
    y[i] = A*cos(2*pi*Fo*t[i]) + An*lmath.random()
end

tfd = data.fft(y, 1)
p = plot.new(800,600)
plot.add(p, f, tfd)
plot.update(p)
```

ac = data.acorr(data)

Calculates the autocorrelation with **data** is the table with data.

The function **data.acorr** returns the obtained autocorrelation table **ac**.

yf = data.filter(x, y, forder)

Filters (smooths) x-y data using the **Savitzky-Golay** method, given the filter order. Returns the filtered data yf.

12.3 ASCII DATA FILES

`c1,c2,... = data.load(filename, sep, skip, colcount, rowcount)`

Loads ASCII data with:

`filename:` source file name.

`sep`: separator , usually tab or semicolon (optional).

`skip`: number of rows to be skipped (optional).

`colcount`: number of columns to load (optional).

`rowcount`: number of rows to load (optional).

The function **`data.load`** returns tables containing numeric data **`c1, c2, …`**

`rowcount = data.save(filename, sep, header, c1, c2, ...)`

Saves numeric data to ASCII file with:

`filename`: destination file name

`sep`: separator (usually tab or semicolon).

`header`: file header (comment, labels, ...)

`c1, c2, ...`: tables to save

The function **`data.save`** returns the number of rows actually saved **`rowcount`**.

Example of using **`data.load`** and **`data.save`**:

```
-- ASCII

cls()

fname = "C:\\Temp\\ascii.txt"
x = {1, 2, 3, 4, 5}
y = {1, 2, 3, 4, 5}
sep = "\t"
skip = 0
rc = data.save(fname, sep, "HEADER\n", x, y)
print(rc,"\n")

xt,yt = data.load(fname, sep, skip)
print(xt,"\n")
print(yt)
```

13. XY PLOTTING

The plot namespace includes functions to plot data:

p = plot.new(width, height, template)

-- creates plot.

plot.add(p, x, y)

-- adds a curve with tables x,y.

plot.add(p, x, y, ey)

-- adds a curve with tables x,y and error bar table ey.

plot.add(p, x, y, ex, ey)

-- adds a curve with tables x,y and error bar tables ex,ey.

plot.add(p, curve, x, y)

-- adds tables x,y to an existing curve.

plot.add(p, curve, x, y, ey)

-- adds tables x,y to an existing curve, with error bar table ey.

plot.add(p, curve, x, y, ex, ey)

-- adds tables x,y to an existing curve, with error bar tables ex,ey.

plot.add(p, expr, npoints, xstart, xend)

-- adds a curve with expression (like "sin(x)", "1 - exp(-x)" …).

plot.add(p, fname)

-- adds a curve from data file (columns 1 and 2, TAB separated).

plot.rem(p, curve)

-- removes curve from plot.

plot.set(p, curve, prop, val)

-- sets the curve properties. **prop** can be:

"**size**" for curve line and symbol size. **val** is the size ("1", "2")

"**style**" for curve. **val** is "o" for circle, "+" for plus sign, "s" for square, "d" for diamond, and "-" for line. Example: "-s" for line and squared markers.

"**legend**" for curve legend. **val** is the legend text.

"**color**" for curve color. **val** is color name like "red", "blue", ... or hex-value like "FF0000").

plot.set(p, prop, val)

-- sets the plot properties. **prop** can be:

"**title**" for the plot window title. **val** is the window title.

"**xlabel**" for the bottom axis label. **val** is the axis label.

"**ylabel**" for the left axis label. **val** is the axis label.

"**xscale**" for the bottom axis scale: **val** is "log" or "linear".

"**yscale**" for the left axis scale: **val** is "log" or "linear".

"**xlim**" for the bottom axis scale: **val** is "[min,max]" where min and max are the x-axis limits.

"**ylim**" for the left axis scale: **val** is "[min,max]" where min and max are the y-axis limits.

"**maxpoints**" set **val** as the maximum number of points per curve.

"**autolim**". **val** is "true": automatically set the axis limits.

plot.save(p, fname)

-- saves plot to file (SVG or PNG).

plot.update(p)

-- updates the plot window.

plot.close(p)

-- closes the plot window.

Example #1:

```
-- Plot
x = {0,1,2,3,4,5}
y = {0,1,2,3,4,5}
p = plot.new(800, 600)           --create a plot
plot.set(p, "title", "Plot example")
plot.add(p, x, y)                --add curve to the new plot
plot.set(p, 1, "size", 2)        --set curve #1 line size
plot.set(p, 1, "style", "-o")    --set curve #1 style(line and marker)
plot.set(p, 1, "color", "0000FF") --set curve #1 color
plot.update(p)
```

Example #2:

```
-- Real-time plot

x = {}
y = {}
x[1] = 1
-- random number between 0 and 1
y[1] = math.random()

p = plot.new()                    --create a plot
plot.add(p, x, y)                 --add curve to the new plot
plot.set(p, 1, "size", 2)         --set curve #1 line size
plot.set(p, 1, "style", "-o")     --set curve #1 style (line and marker)
plot.set(p, "maxpoints", 21)      --set the maximum number of points
plot.update(p)

-- the total number of points (only the last 21 points will be displayed)
n = 100
-- update the curve every 200 milliseconds
for i = 2, n, 1 do
    x[1] = i
    y[1] = math.random()           -- random number between 0 and 1
    plot.add(p, 1, x, y)           -- add new data to curve #1
    time.sleep(200)                -- sleep during 200 milliseconds
end
```

All the plot properties (curves options, scale, axis, colors …) can be easily modified in the plot window. All these properties can be saved in a style template file to be used later. One can create a style template for a particular plot type and use it in the Lua code (`plot.new` function). You can also add text, lines, rectangles, ellipses … to the plot and save it as PNG or SVG directly from within the plot window.

14. BSD SOCKET

The socket namespace includes BSD-like network functions:

```
s = socket.new(af,type,proto)    -- creates socket where:
                                 -- af: family (socket.AF_INET by default)
                                 -- type: type (socket.SOCK_STREAM by default)
                                 -- proto: protocol (def: socket.IPPROTO_TCP)
                                 -- returns the socket identifier

ok = socket.bind(s, addr, port)  -- binds socket s, where:
                                 -- s: socket identifier
                                 -- addr: address (ex: socket.INADDR_ANY)
                                 -- port: port to bind to
                                 -- returns true on success

ok = socket.listen(s, backlog)   -- listens on socket, where:
                                 -- s: socket identifier
                                 -- backlog: maximum queue length
                                 -- returns true on success

ok = socket.connect(s, addr, port)    -- connects socket, where:
                                      -- s: socket identifier
                                      -- addr: address
                                      -- port: port to connect to
                                      -- returns true on success

sa = socket.accept(s, addr, port)
      -- accepts connection and create new socket:
      -- s: socket identifier
      -- returns the new socket identifier sa

ok = socket.timeout(s, to)
      -- sets the recv and send timeout, where:
      -- s: socket identifier
      -- timeout in milliseconds
      -- returns true on success
```

```
ok = socket.setsockopt(s, opt, val)
      -- sets socket option, where:
      -- s: socket identifier
      -- option to set (ex: socket.SO_SNDTIMEO)
      -- val: option value
      -- returns true on success

ip = socket.getpeername(s)        -- gets the socket s peer ip address
                                  -- s: socket identifier
                                  -- returns peer ip address

hn = socket.gethostbyaddr(s, addr)
      -- gets the host name for ip address
      -- s: socket identifier
      -- addr: ip address
      -- returns host name

ha = socket.gethostbyname(s, name)
      -- gets the ip address for host name
      -- s: socket identifier
      -- name: host name
      -- returns ip address

ha = socket.getsockname(s)
      -- gets the socket name, where:
      -- s: socket identifier
      -- name: host name
      -- returns socket name

ok = socket.send(s, data, flags)
      -- sends data, where:
      -- s: socket identifier
      -- data: data to send
      -- flags: optional flags
      -- returns true on success
```

```
ok = socket.sendto(s, addr, port, data, flags)
      -- sends data, where:
      -- s: socket identifier
      -- addr: address
      -- port: port to send to
      -- data: data to send
      -- flags: optional flags
      -- returns true on success

data = socket.recv(s, datasize, flags)
      -- receives data, where:
      -- s: socket identifier
      -- datasize: data size to be received
      -- flags: optional flags
      -- returns received data

data = socket.recvfrom(s, addr, port, datasize, flags)
      -- receives data from, where:
      -- s: socket identifier
      -- addr: address
      -- port: port to receive from
      -- datasize: data size to be received
      -- flags: optional flags
      -- returns received data

errf = socket.iserr(s)          -- gets the error flag:
                                -- s: socket identifier
                                -- returns true if error flag set

errm = socket.geterr(s)         -- gets the error message:
                                -- s: socket identifier
                                -- Returns the error message, if any

socket.shutdown(s)              -- shutdowns socket, where:
                                -- s: socket identifier

socket.delete(s)                -- delete socket and free resources, where:
                                -- s: socket identifier
```

Example:

```
-- BSD Socket
cls()
s = socket.new(socket.AF_INET, socket.SOCK_STREAM, socket.IPPROTO_TCP)
ip = socket.gethostbyname(s, "www.debian.org")
socket.connect(s, ip, 80)
socket.send(s, "HEAD / HTTP/1.0\r\n\r\n")
data = socket.recv(s, 1024)
print(data)
socket.delete(s)
```

15. C MODULES

In addition of the built-in modules, **Comet includes five useful C modules**:

(i) **lvisa** module to control instruments through GPIB, USB, Serial, Ethernet .... This module is available only under Windows. It works with the interface VISA drivers (tested with NI and Agilent/Keysight GPIB cards. The NI drivers, for example, can be freely downloaded from the NI site: *http://www.ni.com/downloads/drivers/*). Using **lvisa** is straightforward: load module with `require` , initialize, open visa connection, communicate and finally close connection.

Example:

```
cls()
local vi = require("lvisa")       -- load the lvisa module
vi.load("visa32.dll")             -- load the driver DLL
v = vi.open("GPIB::8::INSTR")     -- open GPIB connection
vi.write(v, "*IDN?")              -- send command to device
time.sleep(50)                    -- sleep during 50 ms
r,n = vi.read(v,100)              -- read device IDN
print(r)                          -- print it
vi.close(v)                       -- close connection
print(vi.status())                -- print status
```

(ii) **lrs232** module to control instruments through RS232 serial port. This module is available under Windows and Linux.
Under Linux, add user to the dialout group: *sudo usermod -a -G dialout mylogin*
The following example shows how to use the **lrs232** module:

```
cls()

-- load the RS232 C module
local lrs232 = require("lrs232")

-- print out info about the RS232 C module release
print(lrs232.version())

-- open the serial port and configure it
-- port_handle = lrs232.open(port_num, settings, verbose)
--      port_num: usually 1 (COM1) or 2 (COM2)
--      settings has the same meaning than in the Windows BuildCommDCB
--      function except that BuildCommDCB has no 'timeout' parameter
--      that is specific to this Comet module
--      verbose (optional): true to show detailed messages
-- returns the port handle
p = lrs232.open(1, "baud=4800 parity=n data=8 stop=2 timeout=1000", true)

-- write to the serial port
-- lrs232.write(port_handle, command)
--      port_handle: the port handle, as returned by lrs232.open
--      command to send to the device
-- returns the number of bytes written
lrs232.write(p, "*IDN?\n")

-- read from the serial port
-- r,n = lrs232.read(port_handle, bytes)
--      port_handle: the port handle, as returned by lrs232.open
--      bytes: number of bytes to read
-- returns the string read and the number of bytes read
r,n = lrs232.read(p, 128)
io.write("\n recv = ", r, " ; bytes read = ", n)

-- close the serial port
-- lrs232.close(port_handle)
--      port_handle: the port handle, as returned by lrs232.open
lrs232.close(p)
```

(iii) **lpeg** pattern matching module: documentation: http://www.inf.puc-rio.br/~roberto/lpeg/

```
-- http://www.inf.puc-rio.br/~roberto/lpeg/
local lpeg = require("lpeg")

-- matches a word followed by end-of-string
p = lpeg.R"az"^1 * -1

print(p:match("hello"))        --> 6
print(lpeg.match(p, "hello"))  --> 6
print(p:match("1 hello"))      --> nil
```

(iv) **lcrypt** AES encryption module:

```
lcrypt = require("lcrypt")

function printbytes(t)
  local str = '{ 0x'
  for _,v in pairs(t) do
        str = str .. string.format('%02X', v)
  end
  str = str .. ' }'
  print(str, "\n")
end

cls()

-- array of bytes
inp = { 0x6B, 0xC1, 0xBE, 0xE2, 0x2E, 0x40, 0x9F, 0x96, 0xE9, 0x3D, 0x7E,
0x11, 0x73, 0x93, 0x17, 0x2A }

-- AES key (16 bytes = 128 bits)
key = { 0x2B, 0x7E, 0x15, 0x16, 0x28, 0xAE, 0xD2, 0xA6, 0xAB, 0xF7, 0x15,
0x88, 0x09, 0xCF, 0x4F, 0x3C }

-- encrypt the array of bytes
outp = lcrypt.encrypt(inp, key)

-- decrypt the array of bytes to compare to the input data table
inp_decrypted = lcrypt.decrypt(outp, key)
printbytes(inp_decrypted)
```

(v) **lmapm** library for Arbitrary Precision Math (MAPM) by Michael C. Ring, modified Lua interface from lmapm by Luiz Henrique de Figueiredo

```
-- Library for Arbitrary Precision Math (MAPM) by Michael C. Ring
-- Lua interface by Luiz Henrique de Figueiredo

lmapm = require ("lmapm")

-- lmapm library functions:
--   __add(x,y)      cbrt(x)          mod(x,y)
--   __div(x,y)      ceil(x)          mul(x,y)
--   __eq(x,y)       compare(x,y)     neg(x)
--   __lt(x,y)       cos(x)           number(x)
--   __mod(x,y)      cosh(x)          pow(x,y)
--   __mul(x,y)      digits([n])      round(x)
--   __pow(x,y)      digitsin(x)      sign(x)
--   __sub(x,y)      div(x,y)         sin(x)
--   __tostring(x)   exp(x)           sincos(x)
--   __unm(x)        exponent(x)      sinh(x)
--   abs(x)          factorial(x)     sqrt(x)
--   acos(x)         floor(x)         sub(x,y)
--   acosh(x)        idiv(x,y)        tan(x)
--   add(x,y)        inv(x)           tanh(x)
--   asin(x)         iseven(x)        tonumber(x)
--   asinh(x)        isint(x)         tostring(x,[n,exp])
--   atan(x)         isodd(x)         version
--   atan2(y,x)      log(x)
--   atanh(x)        log10(x)

print("\nSquare root of 2")
print("math.sqrt(2)  ", math.sqrt(2))
print("lmapm.sqrt(2) ", lmapm.sqrt(2))
print(lmapm.version)
```

User C modules: you can write your own C modules to use in Comet.

First, write your C module, as in the following example:

```
/* put the Comet include dir in your include path */
#include <lua.h>
#include <lualib.h>
#include <lauxlib.h>

#ifdef WIN32
#define CMODULE_API __declspec(dllexport)
#else
#define CMODULE_API
#endif

/* compatibility with Lua 5.1 */
#if (LUA_VERSION_NUM == 501)
#define luaL_newlib(L, f) luaL_register(L, "Cmodule", f)
#endif

static int Cmodule_version(lua_State *pLua)
{
    lua_pushstring(pLua, "Cmodule v0.1");
    return 1;
}
static const luaL_Reg Cmodule[] = {
    { "version", Cmodule_version },
    { NULL, NULL }
};
CMODULE_API int luaopen_Cmodule(lua_State *pLua)
{
    luaL_newlib(pLua, Cmodule);
    return 1;
}
```

Second, build your C module with your favorite C compiler, Visual C++ on Windows or gcc or Clang on Linux, for example. Before compiling, add the Comet include directory (Comet Dir/include) in the compiler include path and set the linker to link against the Comet Lua Core library (luacore.dll under Windows and luacore.so under Linux). Under Visual C++, you can generate, for your specific VC version, the lib from luacore.dll and luacore.def by using the dumpbin and lib tools included in Visual C++ as follows:

```
cd C:\Comet\bin\
dumpbin /exports luacore.dll > ..\lib\luacore.txt
cd ..\lib\
echo LIBRARY luacore > luacore.def
echo EXPORTS >> luacore.def
for /f "skip=19 tokens=4" %A in (luacore.txt) do @echo %A >> luacore.def
lib /def:"luacore.def" /out:"luacore.lib" /machine:x64
```

A Visual C++ project example is given in Comet/examples/Cmodule. Under Linux, you can use a Makefile such as (file located in Comet/examples/Cmodule):

```
WORKDIR = `pwd`
CC = gcc
INC =
CFLAGS = -Wall -funwind-tables

INC_RELEASE = -I../../include $(INC)
CFLAGS_RELEASE =  -O2 -fPIC -DUSE_LUAJIT -std=c99 $(CFLAGS)
LDFLAGS_RELEASE = -s -L../../include -lluacore $(LDFLAGS)
OBJDIR_RELEASE = .
OUT_RELEASE = ./Cmodule.so
OBJ_RELEASE = ./Cmodule.o

all: release

clean: clean_release

before_release:
  test -d . || mkdir -p .

after_release:

release: before_release out_release after_release

out_release: before_release $(OBJ_RELEASE)
  $(LD) -shared $(OBJ_RELEASE) -o $(OUT_RELEASE) $(LDFLAGS_RELEASE)

$(OBJDIR_RELEASE)/Cmodule.o: Cmodule.c
  $(CC) $(CFLAGS_RELEASE) $(INC_RELEASE) -c Cmodule.c -o
$(OBJDIR_RELEASE)/Cmodule.o

clean_release:
  rm -f $(OBJ_RELEASE) $(OUT_RELEASE)

.PHONY: before_release after_release clean_release
```

and build with:

```
LD_RUN_PATH='$ORIGIN' make all
```

Of course, if your module is already built or purchased you can use it as usual with the Lua ad hoc functions.

Lastly, when your module was built as dll (under Windows) or so (under Linux), you can load it and use it under Solis by using the standard Lua functions:

```
local Cmodule = require("Cmodule")
print(Cmodule.version())
```

16. MATH PARSER

The parser namespace includes functions to evaluate mathematical expressions:

```
p = parser.new()                -- creates a new parser.
parser.set(p, name, value)      -- sets variable.
val = parser.get(p, name)       -- returns variable value.
val = parser.eval(p, expr)      -- evaluates math expression.
val = parser.evalf(p, func, x)  -- evaluates math function for x value.
val = parser.solve(p, eq, a, b) -- solves equation in [a,b] interval.
parser.delete(p)                -- deletes a parser.
```

The math parser supports the following functions:

```
Exp(x)      -- exponential
Ln(x)       -- natural (neperian) logarithm
Log(x)      -- decimal logarithm
Log2(x)     -- base-2 logarithm
Sin(x)      -- sine
Cos(x)      -- cosine
Tan(x)      -- tangent
Asin(x)     -- arc sine
Acos(x)     -- arc cosine
Atan(x)     -- arc tangent
Sinh(x)     -- hyperbolic sine
Cosh(x)     -- hyperbolic cosine
Tanh(x)     -- hyperbolic tangent
Abs(x)      -- absolute value
Sqrt(x)     -- square root
Cbrt(x)     -- cubic root
Ceil(x)     -- ceiling, the smallest integer not less than x
Floor(x)    -- integer part of x
Rand()      -- random number between 0 and 1
Sign(x)     -- sign of x (-1 if x < 0, +1 if x > 0 and 0 if x = 0)
Erf(x)      -- error function
Fact(x)     -- factorial of x
```

The math parser supports also the following constants:

```
Pi          -- π
e           -- Natural (neperian) logarithm base (2.71828...)
```

Universal constants in international units (SI)

```
q           -- Electron charge (in C)
me          -- Electron mass (kg)
mp          -- Proton mass (kg)
kB          -- Boltzmann constant (J/K)
h           -- Planck constant (Js)
c           -- Speed of Light in vacuum (m/s)
eps0        -- Electric constant (F/m)
mu0         -- Magnetic constant (N/A²)
NA          -- Avogadro constant (1/mole)
G           -- Constant of gravitation (m3/kg/s²)
Ri          -- Rydberg constant (1/m)
F           -- Faraday constant (C/m)
R           -- Molar gas constant (J/mole/K)
```

Example of using the parser module:

```
-- Parser
cls()
p = parser.new()
parser.set(p, "x", 1)
parser.set(p, "a", 2)
y = parser.eval(p, "a*x + sin(x/a) + 2")
print("y = " .. y)
```

17. LUA IO FILE FUNCTIONS

The Lua **io** file functions are available in Comet (open, read, write, close and so on.).

Example:

```
-- IO script
cls()
local f = io.open("/mnt/sdcard/Comet/test.txt", "r")
local t = f:read("*all")
f:close()
```

18. LUA IO FUNCTIONS

The input function **io.read** is completely implemented for Windows, Linux and Android.

Example:

```
-- IO read
cls()
io.write("Enter your name: ")
name = io.read("*a")
io.write("Name: ", name)
```

19. DEBUGGING

Comet provides debugging capabilities on Windows and Linux: you can set breakpoints in the code (Menu *Run/Add Breakpoint* or the toolbar corresponding button), view the stack status and current variables set in the Watch window. To start debugging, click Debug (or Menu *Run/Debug* or Ctrl+F12). Debugger can 'Step Into' or 'Step Over' on a call depending on the option set by user in Menu *Run/Step Into* (unchecked for 'Step Over').

20. CHANGELOG

Version 1.8 Build 2104

- Improvements and bugfixes:

  Infobar added for long documents navigation.

  External tools module under Linux improved.

  Linear algebra Lapack-based module improved for Linux 64bit.

  Performance tuned and user interface minor changes.

Version 1.4 Build 1701

- Improvements and bugfixes:

  plot module updated.

  Performance improved.

  Serial (RS232) module added for instrumentation.

  Added library for Arbitrary Precision Math (MAPM) by Michael C. Ring.

  Lua *print* function implementation updated.

Version 1.2 Build 1601

- Minor improvements and bugfixes:

  New tool added: SigmaCalculator, an advanced mathematical expression-based calculator.

  New 'Tools' menu, including the external tools options and SigmaCalculator.

  LPeg pattern matching module added.

  AES encryption module added.

  'Find/Replace in files' functions improved under Linux.

  Explorer minor changes in the move/replace functions.

  Drag/Drop files fully supported under Linux.

Version 1.0 Build 1512

- Unified Android/Linux/Windows version (the same code runs on all supported platforms, except for the linear algebra and network functions not available on Android).
- Network module (**socket** namespace) for Linux and Windows (BSD socket style).
- Linear Algebra module based on the well-established Lapack routines.
- Ordinary differential equation solver.
- Mathematical optimization module.
- PDF User's Guide.

21. SPECIFICATIONS

SYSTEM REQUIREMENTS

Comet runs on:

(i) PC with Windows™ XP, Vista, Windows 7/8/10 32bit or 64bit installed.

(ii) PC with Linux 32bit or 64bit installed.

(iii) Smartphone or Tablet with Android 2.2 or later installed.

INSTALL

- For **portable version** on **Windows** and **Linux**: Download ***comet_windows_64bit.zip*** (Windows 64bit) or ***comet_windows_32bit.zip*** (Windows 32bit) or ***comet_linux_64bit.tgz*** (Linux 64bit) or ***comet_linux_32bit.tgz*** (Linux 32bit) from http://www.hamady.org, unzip/untar in any location (USB key or memory stick for example) and run *comet.exe* (Windows) or *comet* (Linux) in the bin directory.
- For **setup version** (on **Windows 64bit**): Download ***comet_windows_64bit.msi*** from http://www.hamady.org, double-click on the setup file and it will install Comet.
- For **Android**: Install from Google Play.

22. LINKS

To view the Comet online Help and read the latest news, visit my website: **http://www.hamady.org/**

For the language documentation, you can visit the Lua official website:

http://www.lua.org/ and wiki: http://lua-users.org/wiki/

23. SUPPORT

For assistance request, bug report, suggestion, question, you can either:

(i) send an e-mail to sidi@hamady.org.

(ii) visit my website (http://www.hamady.org) and submit a support request.

You will have an answer as soon as possible.

24. COPYRIGHT

Comet:

Comet uses:

The Lua programming language, (C) 1994–2013 Lua.org, PUC-Rio.

The Lua Just-In-Time Compiler, (C) 2005-2015 Mike Pall.

The f2c'ed version of Lapack, (C) 1992-2008 The University of Tennessee.

The LIS linear solvers, (C) The SSI Project, Kyushu University, Japan.

The ODE solver developed by Scott D. Cohen and Alan C. Hindmarsh @ LLNL.

The Scintilla Component, (C) 1998-2004 Neil Hodgson.

The wxWidgets GUI toolkit, (C) 1992-2013 the wxWidgets team.