Comet for Android
Environment for Numerical Computing

Sidi HAMADY
Full Professor, Dr. habil. Eng.
Université de Lorraine, France
LMOPS Lab., Université de Lorraine & CentraleSupélec, France
http://www.hamady.org

The latest version of this manual is available here:
http://www.hamady.org/download/comet.pdf
This manual is related to Comet for Android only. The manual of Comet for Linux and Windows is here: http://www.hamady.org/download/legacy/comet.pdf. Note that Comet for Linux and Windows is now open-source under the MIT license: https://github.com/sidihamady/Comet.

Comet for Android Lua engine uses:
the Lua programming language, (C) Lua.org, PUC-Rio.
the ODE solver developed by Scott D. Cohen and Alan C. Hindmarsh @ LLNL.

Contents

Listings

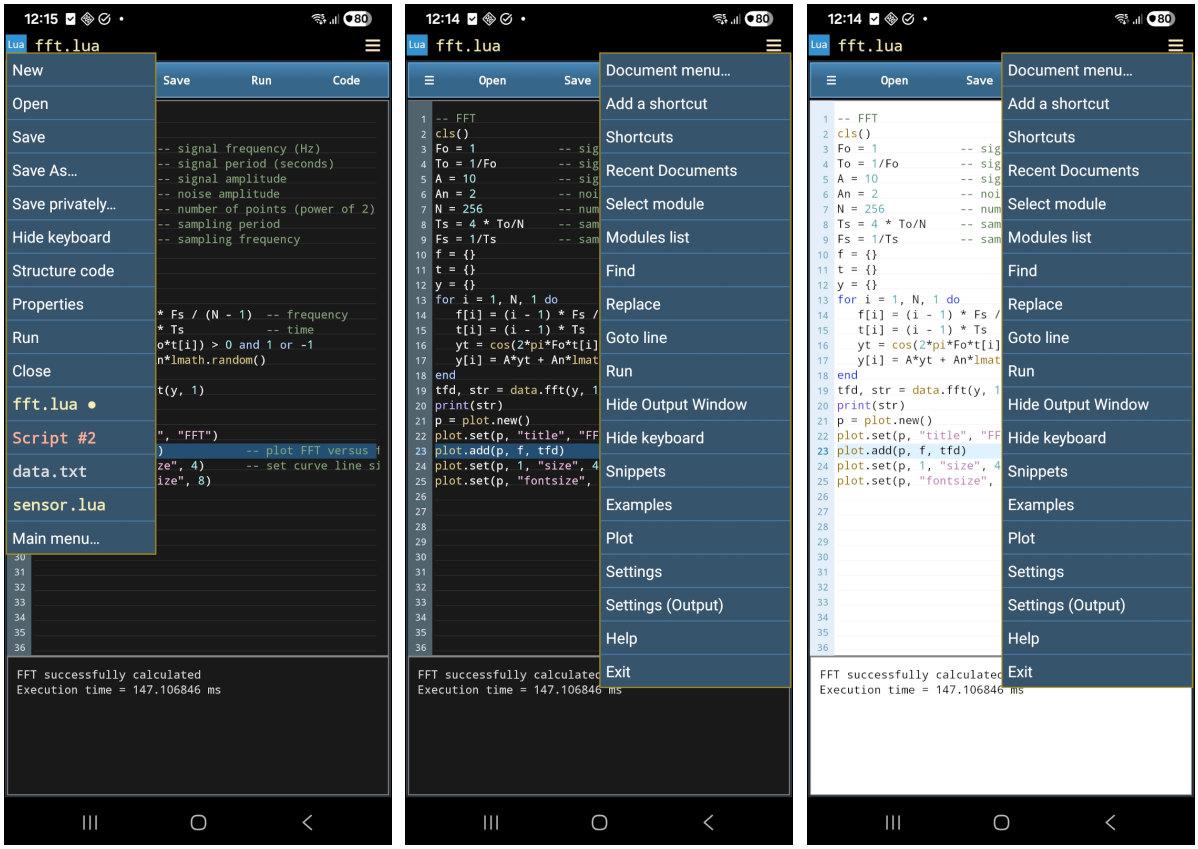

Figure 1: Comet for Android Screenshots.

Presentation

Comet is a development environment for the Lua scripting language for Android, Windows and Linux with built-in Lua scripting engine. It is mainly dedicated to numerical computing and data analysis.

Features: built-in Lua scripting engine, mathematical, data analysis and plotting modules, syntax highlighting, included Lua samples and code snippets, output area, save/open to/from memory, ...

Comet is available in **English**, **French**, **Portuguese** and **Spanish**, depending on your device and user settings.

Note that Comet for Linux and Windows is now replaced by Solis that offers similar functionalities and some new additional ones. Note also that Solis is not fully compatible with Comet.

To learn the Lua programming language, you can read the reference book[1] and 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

HowTo

With the **toolbar**, you can **run**, **open** or **save** document from or to the memory.
You can perform the same actions with the menu.

Comet includes some code samples and snippets (menu/Examples ; menu/Snippets or the Snippets button in the toolbar). With code snippets, you can insert some pieces of code (for, switch, if , function, ...) in your script.

If you want to change the font size, select menu/Settings.
To select the app language, select menu/Settings and choose between **English** (en), **French** (fr), **Portuguese** (pt) and **Spanish** (es).

You can create a new document or save the current document to another location, by selecting menu. Scripts can be saved using the native Android file manager (**menu/Save as**), or in the Comet private directory (**menu/Save privately**). The files saved in the Comet private directory will be deleted after uninstalling the application.

You can add a shortcut from the menu, and by long-pressing the Comet icon on the home screen, you will see a list of shortcuts. You can tap any shortcut to open it in Comet, or drag and drop it to the home screen to launch it directly from there.
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 long press and select *Clear*.

You can set the output area height: Menu / Settings (Output).
Comet 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 **search** text in the document: menu/Find. To **replace** a text in the document: menu/Replace. The **recent open files list** can be shown: menu / Recent Documents.

Example of Lua script: Type the **Listing 1** in the editor and click **Run** or select **menu/Run**.

The run time (script execution time in millisecond) can be printed out. To enable or disable this feature, long press on the output area and select *+ (or -) Execution time*, or select menu/Settings, or long press the Run button.

```lua
-- Lua first script

cls()

a = math.cos(math.pi/4)
print("result = ", a)
```

Listing 1: Example of Lua Script

Function	Purpose
droid.log(tag, msg, type="d")	logs a message to the Android Logcat (*type* = "d", "i", "w" or "e")
droid.toast(msg)	show a quick pop up notification message
droid.alert(ttl, msg)	show an alert dialog box, with title and message
droid.disp(ttl, txt)	display text, with title, in a dialog, for relatively long text
droid.vibrate(ms)	vibrate during ms milliseconds. The vibration is simulated using a sound so that Comet requires no permissions.
droid.speak(msg)	speak message using Text-to-Speech
info = droid.cpuinfo()	retrieves information about the processor
info = droid.batteryinfo()	retrieves information about the battery (current level, charging status)
v = droid.version()	retrieves the Android version
droid.beep()	plays a short beep sound
data = droid.sensor(name)	returns a sensor reading, where *name* is "accelerometer", "gyroscope", "magnetometer", "light", "proximity", "pressure" or "temperature"
droid.nosleep()	keeps the device's screen on, which is useful during sensor measurements
droid.notify(message)	posts a notification

Table 1: Android specific functions.

Android specific functions

Some Android functions are added to Comet (**droid** namespace) as shown in **Table 1**. Example in **Listing 2**.

Execution mode: Comet's execution mode is designed to perform long calculations, without user interaction. This makes it suitable for math and algorithms development.

Editor options: The Comet editor can display lines number and highlight the current line and the line where a Lua error occurred, ... You can set the editor font size, who or hide the

```
-- Droid

ttl = "Droid Test"
msg = "Hello, world!"
droid.alert(ttl, msg)
droid.vibrate(1000)
```

Listing 2: Using Android Specific 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		

Table 2: Lua math functions. NB: Lua gives the Napierian logarithm the name *log* and the decimal logarithm is named *log10*, as in C language.

output area, ... To enable or disable editor features, select menu/Settings.

Comet data (current document, application state, ...) are permanently saved, and automatically restored when Comet restart.

Math functions

With Comet, the math functions are mapped to global functions (e.g. : user can use *sin* or *math.sin*). **Table 2** gives a summary of the Lua math functions. Special math functions are added to Comet (**lmath** namespace), including and extending the Lua math functions. A summary of lmath functions and constants is shown in **Table 3** and **Table 4**.

Time functions

tic() set the wall-clock timer
toc() return the number of milliseconds elapsed since the last tic() call
sleep(n) sleep for n milliseconds

Example in **Listing 3**.

Lua Function	Math Formula
lmath.exp2(x)	2^x
lmath.exp(x)	e^x
lmath.cbrt(x)	$\sqrt[3]{x}$
lmath.hypot(x,y)	$\sqrt{(x^2 + y^2)}$
lmath.erf(x)	error function
lmath.erfc(x)	complementary error function
lmath.lgamma(x)	$\ln(\Gamma(x))$
lmath.tgamma(x)	$\Gamma(x)$
lmath.trunc(x)	nearest integer
lmath.round(x)	nearest integer, rounding
lmath.isinf(x)	number is infinite?
lmath.isnan(x)	not a number?
lmath.isnormal(x)	number is normal?
lmath.asinh(x)	
lmath.acosh(x)	
lmath.atanh(x)	
lmath.gauss(x,b,c)	$G(x) = \exp(-\frac{(x-b)^2}{2c^2})$
lmath.lorentz(x,b,c)	$L(x) = \frac{c}{\pi((x-b)^2+c^2)}$
lmath.fact(n)	factorial of n
lmath.gcd(a,b)	greatest common divisor of a and b
lmath.lcm(a,b)	least common multiple of a and b
lmath.comb(n,k)	the number of k-combinations of n
lmath.perm(n,k)	the number of k-permutations of n

Table 3: Lua extended math functions.

Constant Symbol	Constant Name
lmath.q	Electron charge (in C)
lmath.me	Electron mass (kg)
lmath.kb	Boltzmann constant (J/K)
lmath.h	Planck constant (Js)
lmath.c	Speed of Light in vacuum (m/s)
lmath.na	Avogadro constant (1/mol)

Table 4: Universal constants defined in *lmath*, in international units (SI).

```
-- Timer

tic()

sleep(1000)

dt = toc()

print("Elapsed time = ", dt, " ms")
```

Listing 3: Time Functions.

Numerical Array Functions

Functions to handle numerical arrays that can be called through the linalg namespace. Below is a summary of the **linalg** functions:

x = linalg.array(n,a)
create array of size n and initialize its values to a

x = linalg.zeros(n)
create n-size array and initialize its values to zero

x = linalg.ones(n)
create n-size array and initialize its values to one

linalg.swap(x,y)
swap two arrays

linalg.copy(x,y)
copy array x to y

s = linalg.dot(x,y)
scalar product

z = linalg.add(x,y,a,b)
return a*x + b*y

z = linalg.mult(a,b,m,n)
multiply a matrix (with m columns) with a matrix (with n columns)

y = linalg.get(x,is,ie)

return x subarray from index is to ie

z = linalg.cat(x,y)
concatenate x and y arrays

B = linalg.transpose(A,m)
transpose a matrix with m columns

s = linalg.format(A,m)
format an array with m columns and return a string

s = linalg.print(A,m)
print an array with m columns

x = linalg.rand(n,rmin,rmax)
return an array of random values between rmin and rmax

i = linalg.imin(x)
return index of the min value in x

i = linalg.imax(x)
return index of the max value in x

s = linalg.sum(x)
return the array sum

d = linalg.norm(x)
return the array norm-2

iA = linalg.inv(A)
calculate the inverse of a square matrix

d = linalg.det(A)
calculate the determinant of a square matrix

x = linalg.solve(A,b)
solve the linear system Ax = b using Gaussian elimination

x = linalg.tridiag(Al,Ad,Au,b)
solve the tridiagonal linear system Ax = b with: Al the lower diagonal; Ad the main diagonal and Au the upper diagonal

```
-- linalg.solve
-- numerical array functions

-- Solve a linear system: Ax = b
local A = {2, 1, -1, -3, -1, 2, -2, 1, 2}
local b = {8, -11, -3}
local x = linalg.solve(A,b)
-- should give {2, 3, -1}
io.write("x = ")
linalg.print(x)

-- Inverse and determinant of a matrix
local A = {2, 1, -1, -3, -1, 2, -2, 1, 2}
local iA = linalg.inv(A)
io.write("iA = ")
linalg.print(iA)
local M = linalg.mult(A,iA)
-- should give {1, 0, 0, 0, 1, 0, 0, 0, 1}
io.write("iA*A = ")
linalg.print(M)

-- Solve a tridiagonal system
local l = {-1, -1, -1} -- lower diagonal (length n-1)
local d = {2, 2, 2, 2} -- main diagonal (length n)
local u = {-1, -1, -1} -- upper diagonal (length n-1)
local b = {1, 0, 0, 1} -- rhs vector
local x = linalg.tridiag(l, d, u, b)
-- should give {1, 1, 1, 1}
io.write("x = ")
linalg.print(x,1) -- vector, then 1 column
```

Listing 4: Numerical Array Functions.

Example in **Listing 4**.

Ordinary Differential Equations (ODE)

Using the Comet ODE solver, which is based on the solver developed by Scott D. Cohen and Alan C. Hindmarsh at LLNL [2], one can integrate system of differential equations, 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
y0 table containing the initial values

t0 value for y0
t1 value to integrate for
tol the solver tolerance (optional)

Example in **Listing 5**.

Lua IO functions

The Lua IO functions are available in Comet (*open*, *read*, *write*, *close* and so on.).
Note that only the files located in the Comet private directory (given by *droid.appdir()*) can be accessed. Example in **Listing 6**.

Function io.read is completely implemented without limitations. Example in **Listing 7**.

Math Parser

parser namespace includes functions to evaluate mathematical expressions:
parser.init() initialize the parser p.
parser.set(name, value) set variable.
parser.get(name) return variable value.
parser.eval(expr) evaluate math expression.
parser.evalf(func, x) evaluate math function for x value.
parser.solve(eq, a, b) solve equation in [a,b] interval.
parser.destroy(p) destroy the parser.

The math parser supports the functions and constants shown in **Table 5** in **Table 6**.
Example of using the parser module is given in **Listing 8**.

XY Data Plotting

The **plot** namespace includes functions to plot data:
p = plot.new() create plots and returns an identifier p that will be used il all plot functions.
plot.add(p,x,y) adds to plot a curve with tables x and y as points coordinates.
plot.update(p,curve,x,y,autoscale) updates the x-y data of an existing curve (*curve* is 1, 2, etc.).
plot.activate(p,curve) to activate a curve; touching it then displays the x and y coordinates.
plot.set(p, curve, prop, val) sets the curve properties. prop can be:

```
-- Ordinary Differential Equation
-- Damped Oscillator: y'' + c y' + k y = 0

local a = -0.25
local b = 1
local c = -2 * a
local k = a*a + b*b

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()
plot.set(p, "title", "Ordinary Differential Equation")
plot.add(p, tm, y)
plot.add(p, tm, dy)
```

Listing 5: Ordinary Differential Equations (ODE).

```
cls()

adir = droid.appdir()
print(adir)
fn = adir .. "/test.txt"
fp = io.open(fn, w+)
fp:write("Hello, world!")
fp:close()
fp = io.open(fn, r)
tt = fp:read(*all)
fp:close()
io.write(tt)
```

Listing 6: IO Functions.

```
-- io.read
cls()
io.write("\nEnter your name: ")
name = io.read("*a")
io.write("\nName: ", name)
```

Listing 7: io.read

Parser Function	Math Formula
Exp(x)	e^x
Ln(x)	natural (Napierian) logarithm
Log(x)	decimal logarithm
Log2(x)	base − 2 logarithm
Sin(x)	
Cos(x)	
Tan(x)	
Asin(x)	
Acos(x)	
Atan(x)	
Sinh(x)	
Cosh(x)	
Tanh	
Abs	absolute value
Sqrt(x)	$\sqrt{x}$
Cbrt(x)	$\sqrt[3]{x}$
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

Table 5: Math parser functions.

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

Listing 8: Math Parser.

Constant Symbol	Constant Name
π	
e	2.71828...
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)
NA	Avogadro constant (1/mol)
ε_0	Electric constant (F/m)
μ_0	Magnetic constant (F/m)
G	Constant of gravitation ($m^3/kg/s^2$)
Ri	Rydberg constant (1/m)
F	Faraday constant (C/m)
R	Molar gas constant (J/mol/K)

Table 6: Math parser constants.

```
-- Plot
cls()
x = {1,2,3,4,5}
y1 = {1,2,3,4,5}
y2 = {2,3,4,5,6}
p = plot.new()
plot.add(p, x, y1)
plot.add(p, x, y2)
plot.set(p, "title", "Plot")
plot.set(p, 1, "size", 2) -- Curve #1
plot.set(p, 1, "style", "-")
plot.set(p, 2, "size", 4)
plot.set(p, 2, "style", "-o")
```

Listing 9: XY Data Plotting.

"size" for curve line and symbol size. val is the size ("1", "2", ...).
"style" for curve. val is "o" for dot and "-" for line or both ("-o").
plot.set(p, prop, val) sets the plot properties. prop can be:
"fontsize" for the font size. val is the size ("10", "14", ...).
"title" for plot title. val is the title text.
plot.vline(p,x) adds a vertical line at x to the plot.
plot.hline(p,y) adds horizontal line at y to the plot.
plot.save(p,"myplot.pdf") exports the plot to PDF.
You can view the most recently created plot by selecting: menu/Plot.

Examples in **Listing 9** and **Listing 10**.

```
-- Plot reload
local x = {}
local y = {}
p = plot.new()
plot.set(p, 1, "style", "o-")

for i = 1, 10 do
    local s = droid.sensor("accelerometer")

    local a = {}
    for part in string.gmatch(s, "([^;]+)") do
        table.insert(a, part)
    end

    table.insert(x, i)
    table.insert(y, a[4])
    plot.update(p, 1, x, y)
    sleep(500)
end
```

Listing 10: XY Data Plotting - reload.

Mathematical Optimization

Comet integrates a mathematical optimization module including minimization of real function of n variables and root finding.

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 is the 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.

Root of real function:
xr = optim.root(func,a,b,iters,tol) Find the root of function in the interval a,b (optional) with the given tolerance (optional) and maximum number of iterations (optional).

Examples in **Listing 11** and **Listing 12**.

```
-- Minimization of a function
function booth(x)
   y = math.pow(x[1] + 2*x[2] - 4, 2)
   y = y + math.pow(2*x[1] + x[2] - 5, 2)
   return y
end

x = { -5, 5 }
iters = optim.minimize("booth", x, 100, 1e-7)
-- expected: x = {2,  1}
res = string.format("\n x = [%g %g]  iters = %d\n", x[1], x[2], iters)

cls()
io.write(res)
```

Listing 11: Mathematical Optimization: optim.minimize

```
-- Zero of a function

function func(x)
    return x^2 - x - 1
end

-- expected: x ~ -0.618034
x = optim.root("func", -2, 2)
res = string.format("x = %g   f(x) = %g\n", x, func(x))
cls()
io.write(res)
```

Listing 12: Mathematical Optimization: optim.root

Function	Purpose
data.min(t)	Minimum
data.max(t)	Maximum
data.sum(t)	Sum
data.mean(t)	Mean
data.median(t)	Median
data.var(t)	Variance
data.dev(t)	Standard Deviation
data.coeff(t)	Coefficient of Variation
data.rms(t)	Root Mean Square
data.skew(t)	Skewness
data.kurt(t)	Kurtosis Excess

Table 7: Descriptive statistics functions.

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

Listing 13: Descriptive Statistics.

Data Analysis

The **data** namespace includes functions to calculate the descriptive parameters of data, as shown in **Table 7**.

The formulas used can be found in my website: http://www.hamady.org/data.html
All the stats functions take a Lua table as argument, with 1048576 maximum number of elements.
Example in **Listing 13**.

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

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

Listing 14: data.fit: func

```
-- Data Fitting
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)
str = string.format("pars = [%g %g] \n", pars[1], pars[2] )

io.write(str)
p = plot.new()
plot.set(p, "title", "Linear Fitting")
plot.add(p, tx, ty) -- plot data
tx2 = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
ty2 = {}
for i = 1, #tx2, 1 do
   ty2[i] = pars[1] + pars[2]*tx2[i]
end
plot.add(p, tx2, ty2) -- plot linear fit
plot.set(p, 1, "style", "o")
```

Listing 15: Data Fitting.

pars, chi, iters, str = data.fit(func, tx, ty, fpar, ipar, tol, iters) run the fitter algorithm with:

func the Lua model function name. The Lua function syntax is given in **Listing 14** (replace with your own model).

tx is the table with X data.

ty is the table with Y data.

fpar is the fitting parameters table.

ipar is a table containing, for each parameter, value 1 if the parameter is varying or 0 if it is fixed. This parameter *ipar* is optional.

tol is the relative tolerance to be reached. This parameter *tol* is optional.

iters is the maximum number of iterations for the fitting algorithm. This parameter **iters** is optional.

The function **data.fit** return 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 is given in **Listing 15**.

fft, str = data.fft(data, idir) calculate the Fast Fourier Transform (FFT) with:

```
-- Fast Fourier Transform (FFT)
cls()

Fo = 1            -- signal frequency (Hz)
To = 1/Fo         -- signal period (seconds)
A = 10            -- 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
   yt = cos(2*pi*Fo*t[i]) > 0 and 1 or -1
   y[i] = A*yt + An*lmath.random()
end

tfd, str = data.fft(y, 1)
print(str)

p = plot.new()
plot.set(p, "title", "FFT")
plot.add(p, f, tfd)             -- plot FFT versus frequency
plot.set(p, 1, "size", 4)       -- set curve line size to 4
```

Listing 16: Fast Fourier Transform (FFT).

data the table with data
idir 1 for forward FFT and 0 for inverse
data.fft returns two parameters: the obtained FFT table **fft** and a message **str** from the calculator.
NB: the FFT amplitude is scaled (divided) by the number of points.
Example of using FFT in **Listing 16**.

ac, str = data.acorr(data) calculate the autocorrelation with:
data the table with data
The function **data.acorr** return two parameters: the obtained autocorrelation table **ac** and a message **str** from the calculator.

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).

```
-- Save and load ASCII data

cls()

fname = "data.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)
```

Listing 17: Loadind and Saving ASCII Data.

rowcount number of rows to load (optional).
The function data.load returns tables containing numeric data c1, c2, ...
rowcount = data.save(filename, format, header, c1, c2, ...)
Saves numeric data to ASCII file with:
filename: destination file name
format: line format (example: "%f\t%f") or separator (usually TAB (\t) or semicolon (;)).
header: file header (comment, labels, ...)
c1, c2, ...: tables to save
The function data.save returns the number of rows actually saved rowcount.

An example of using data.load and data.save is given in **Listing 17**.

datayc, status = data.baseline(datay, alambda, itermax, reltol, verbose)
Baseline correction using the algorithm developed by Zhang et al. [3].
datay the table with data
alambda correction parameter (default value: 100)
itermax maximum number of iterations (default value: 10)
reltol relative tolerance to achieve (default value: 0.001)
verbose true to print messages
The function data.baseline returns the corrected data table **datayc** and a **status** (true if succeeded).
An example of using data.baseline is given in **Listing 18**.

datayn = data.normalize(datay, anorm) Normalize data to [0, anorm]:
datay the table with data
anorm maximal value to normalize to
The function data.normalize returns the normalized data table **datayn**.

```
-- Baseline correction

cls()

fname = "data.txt"
datax, datay = data.load(fname, "\t")
datayc, status = data.baseline(datay, 50, 100, 1e-3)
fname = "data_corrected.txt"
data.save(fname, "\t", "# corrected data", datax, datayc)

p = plot.new()
plot.add(p, datax, datay)
plot.add(p, datax, datayc)
plot.refresh(p)
```

Listing 18: Baseline correction.

```
-- 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
```

Listing 19: Pattern Matching Module.

C Modules

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

lpeg: Pattern matching

lpeg pattern matching module by R. Ierusalimschy [4] with an example given in **Listing 19**.

lcrypt: AES encryption

lcrypt AES encryption module with an example given in **Listing 20**.

```
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, 0
    x2A }

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

-- 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)
```

Listing 20: AES Encryption Module.

```
-- 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)
```

Listing 21: Library for Arbitrary Precision Math (MAPM).

lmapm: library for Arbitrary Precision Math

lmapm library for Arbitrary Precision Math (MAPM) by Michael C. Ring [5], modified Lua interface from lmapm by Luiz Henrique de Figueiredo. An example is given in **Listing 21**.

```
-- Library for SQL databases using the SQLite library

lsql = require ("lsql")

dbfilename = "database.sqlite"
db = lsql.create(dbfilename)            -- create the database
sql = "DROP TABLE IF EXISTS TABLE_HEADER; CREATE TABLE TABLE_HEADER(Id INTEGER, Name TEXT,
    Description TEXT);"
qy = lsql.exec(db, sql)                 -- exec a statement
sql = "INSERT INTO TABLE_HEADER VALUES(1, 'first', 'first description')"
qy = lsql.exec(db, sql)
sql = "INSERT INTO TABLE_HEADER VALUES(2, 'second', 'second description')"
qy = lsql.exec(db, sql)

qy = lsql.query(db, "SELECT * FROM 'TABLE_HEADER' LIMIT 0,2")
lsql.next(db, qy)                       -- go to the first row
id = lsql.read(db, qy, 1)               -- read column #1
name = lsql.read(db, qy, 2)             -- read column #2
description = lsql.read(db, qy, 3)      -- read column #3
print("\n", id, name, description)
lsql.next(db, qy)                       -- go to the next row
id = lsql.read(db, qy, 1)               -- read column #1
name = lsql.read(db, qy, 2)             -- read column #2
description = lsql.read(db, qy, 3)      -- read column #3
print("\n", id, name, description)
lsql.close(db)                          -- close the database

print(lsql.version())
```

Listing 22: Library for SQL databases.

lsql: library for SQL databases

lsql library for SQL databases using the SQLite library [6]. An example is given in **Listing 22**.

Comet includes a **module manager** that allows modules to be grouped in Comet's private directory to allow C and Lua functions to access files, following the new Android API.
To use a module (through *require*): Menu/Select module. It will thus be copied into the Comet directory. The list of *.so* and *.lua* modules is accessible via Menu/Modules list.
Additionally, text files with the extension *.txt*, and SQL database files with the extension *.sqlite*, can be selected in the same way. Including data files and databases in the application's private directory allows them to be processed with Lua code without the need for permissions.

Learn Lua

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: https://www.lua.org/docs.html

Changelog

Comet Version 3.8.0 (#49)

- Minor UI changes.
- Enhanced UI performance.

Comet Version 3.6.0 (#48)

- New feature: AutoSave with user-defined timing.

Comet Version 3.4.0 (#46)

- Updated the Lua engine to version 5.5.0.
- Improved the help search functionality.

Comet Version 3.2.0 (#45)

- Syntax highlighter implementation refined.
- New feature: Word wrap.
- Improved the PDF exporter.
- Enhanced drawing performance.

Comet Version 3.0.0 (#44)

- New feature: Comet now supports multiple simultaneously open documents, available via the new Document menu.
- New feature: Comet now supports opening and saving text files (.txt).
- New languages: Comet is now available in Portuguese and Spanish, in addition to English and French.
- New features: Display document and plot properties; Code structuring for scripts.
- New Android functions:
  - *droid.speak(msg)* using Text-to-Speech.
  - *droid.log(tag,msg,type="d")* logs a message to the Android Logcat.
  - *droid.cpuinfo()* retrieves information about the processor.
  - *droid.batteryinfo()* retrieves information about the battery (current level, charging status).
  - *droid.version()* retrieves the Android version.
  - *droid.beep()* plays a short beep sound.

  - *droid.sensor(name)* returns a sensor reading.
  - *droid.nosleep()* keeps the device's screen on, which is useful during sensor measurements.
  - *droid.notify(message)* posts a notification.

- New plot function: *plot.update(p,curve,x,y,autoscale)* to update the x-y data of an existing curve.

- New plot functions:

  - *plot.update(p,curve,x,y,autoscale)* to update the x-y data of an existing curve.
  - *plot.activate(p,curve)* to activate a curve; touching it then displays the x and y coordinates.
  - *plot.vline(p,x)* to add a vertical line at x to the plot.
  - *plot.hline(p,y)* to add a horizontal line at x to the plot.

- Use of Android foreground service for long script runs, in line with the Google policy, with a notification when active.

- Updated examples.

- UI adjustments.

- Removed the vibration-related permission: Comet now uses no features that require any user permissions.

Bibliography

[1] R Ierusalimschy, L H Figueiredo, and Celes W. *Lua Reference Manual*. Lua.org, 2020 (cited p. 1).

[2] S D Cohen and A C Hindmarsh. *CVODE user guide*. Tech. rep. Technical Report UCRL-MA-118618, LLNL, 1994 (cited p. 8).

[3] Z M Zhang, S Chen, and Y Z Liang. "Baseline correction using adaptive iteratively reweighted penalized least squares". In: *Analyst* 135.5 (2010), pp. 1138–1146 (cited p. 18).

[4] R Ierusalimschy. *LPeg: pattern-matching library for Lua*. `https://www.inf.puc-rio.br/~roberto/lpeg/`. Lua repository. 2023 (cited p. 19).

[5] C R Michael. *MAPM: Arbitrary Precision Math Library*. `https://github.com/LuaDist/mapm`. GitHub repository. 2014 (cited p. 21).

[6] K P Gaffney, M Prammer, L Brasfield, D R Hipp, D Kennedy, and J M Patel. "SQLite: past, present, and future". In: *Proceedings of the VLDB Endowment* 15.12 (2022) (cited p. 22).