wasora's an advanced suite for optimization & reactor analysis

wasora is a free computational tool designed to aid a cognizant expert—i.e. you, whether an engineer, scientist, technician, geek, etc.—to analyze complex systems by solving mathematical problems by means of a high-level plain-text input file containing algebraic expressions, data for function interpolation, differential equations and output instructions amongst other facilities. At a first glance, it may look as another high-level interpreted programming language, but—hopefully—it is not: wasora should be seen as a syntactically-sweetened way to ask a computer to perform a certain mathematical calculation. For example, see below to find how the famous Lorenz system may be solved by writing the three differential equations into a plain-text input file as humanly-friendly as possible.

Although its ultimate subject is optimization, it may hopefully help you with the tough calculations that usually appear when working with problems that have some kind of complexity, allowing the user to focus on what humans perform best—expert judgment and reaching conclusions. Some of its main features include

- evaluation of algebraic expressions
- one and multi-dimensional function interpolation
- scalars, vectors and matrices operations
- numerical integration, differentiation and root finding of functions
- possibility to solve iterative and/or time-dependent problems
- adaptive integration of systems of differential-algebraic equations
- I/O from files and shared-memory objects (with optional synchronization using semaphores)
- execution of arbitrary code provided as shared object files
- parametric runs using quasi-random sequence numbers to efficiently sweep a sub-space of parameter space
- non-linear fit of scattered data to one or multi-dimensional functions
- non-linear multidimensional optimization

Almost any single feature included in the code was needed at least once by the author during his career in the nuclear industry. Nevertheless, wasora is aimed at solving general mathematical problems (see below for a description of the wasora Real Book). Should a particular calculation be needed, wasora's features may be extended by the implementation of dynamically-loaded plugins, for example:

besssugo builds scientific videos out of wasora computations

fino solves partial differential equations using the finite element method

milonga solves the multigroup neutron diffusion equation

waspy runs Python code within wasora sharing variables, vectors and matrices

xdfrrpf eXtracts Data From RELAP Restart-Plot Files

A template that can be used to write an *ad-hoc* plugin can be cloned with

```
$ hg https://bitbucket.org/gtheler/skel
```

The set of wasora plus one or more of its plugins is referred to as the *wasora suite*.

The code heavily relies on the numerical routines provided by the GNU Scientific Library, whose installation is mandatory. In fact, wasora can be seen as a high-level front-end to GSL's numerical procedures. The solution of differential-algebraic systems of equations is performed using the SUNDIALS IDA Library, although this feature usage is optional. See the file `INSTALL` for details about compilation and installation.

Running wasora

Following a design decision, wasora reads a plain-text file referred to as the *input file* that contains a set of alphanumeric keywords with their corresponding arguments that define a certain mathematical problem that is to be solved. See the file `examples/parser.was` that explains how wasora parses its input files.

If you obtained the source tree—either by downloading the tarball or by cloning the mercurial repository—wasora has to be compiled to obtain a binary executable (see the file `INSTALL` for details). If you downloaded a binary tarball for your architecture, the executable should be located in the root directory of the distribution. This executable can be either installed in a system-wide location (for example in `/usr/bin`), into a directory contained in the user's `$PATH` environment variable (for example in `$HOME/bin`) or even in present working directory (i.e. where the input file is). The appropriate decision is up to the user. In any case, wasora expects the name of the input file (or a path if it is not located in the current directory, although this situation may mangle the access to other needed files) as the first argument. Assuming wasora is installed in a directory listed in the `$PATH` variable and that the input file is named `input.was`, then the proper execution instruction is

```
$ wasora input.was
```

There exist some command line options—that may be consulted using the `--help` option—that are detailed discussed in the complete documentation. In particular, the `--version` option shows information about the wasora version and the libraries it was linked against:

```
$ wasora --version
wasora 0.3.2 default (2014-08-16 17:09 -0300)
wasora's an advanced suite for optimization & reactor analysis

 revision id 1a6320c8e3e2a3c8482c3ee17b8adfa18894f907
 last commit on 2014-08-16 17:09 -0300 (rev 2)

 compiled on 2014-08-16 23:19:52 by jeremy@tom (linux-gnu x86_64)
 with gcc (Debian 4.9.1-4) 4.9.1 using -O2 and linked against
  GNU Scientific Library version 1.16
  GNU Readline version 6.3
  SUNDIALs Library version 2.5.0


 wasora is copyright (C) 2009-2014 jeremy theler
 licensed under GNU GPL version 3 or later.
 wasora is free software: you are free to change and redistribute it.
 There is NO WARRANTY, to the extent permitted by law.
$
```

The input file may also ask for command line options—for example to pass run-time arguments so the same input file can be used to solve similar problems—by referring them as `$1`, `$2`, etc. These `$n` expressions are literally replaced by the command line arguments provided after the input file. So for example, the following single-line input file (which can be found in `examples/calc.was`):

```
PRINT %g $1
```

can be used as a command-line calculator:

```
$ wasora calc.was 1+1
2
$
```

See the `examples/parser.was` file, the Examples & test suite and The wasora Real Book sections below for examples of usage of arguments.

Examples & test suite

After the compilation of the code (that follows the standard `./configure && make` procedure, see `INSTALL` for details), one recommended step is to run the test suite with

```
$ make check
```

It consists of ten examples of application that use different kind of the features provided by wasora. They work both as examples of usage and as a suite of tests that check that wasora implements correctly the functionalities that are expected. A more detailed set of examples that illustrate real applications of wasora in a wide variety of fields—ranging from classical mechanical systems up to analysis of blackjack strategies—can be found in The wasora Real Book. Some of the cases in the test suite generate graphical data which is shown on the screen using gnuplot, provided it is installed.

The `make check` command may not show the actual output of the examples but the overall result (i.e. whether the test passed, the test failed or the test was skipped). Each individual test may be repeated by executing the `test-*.sh` scripts located in the `examples` subdirectory. A brief description of the ten cases follows.

The Fibonacci sequence

Compute the first fifteen numbers of the Fibonacci sequence by building a vector and setting the individual elements as the sum of the previous two with `fibonacci.was`:

```
# compute the first 15 numbers of the fibonacci sequence
VECTOR f SIZE 15

f(i)<1:2> = 1
f(i)<3:vecsize(f)> = f(i-2) + f(i-1)

PRINT_VECTOR f FORMAT %g

# exercise: increase the size of vector f
```

```
$ wasora fibonacci.was
1
1
2
3
5
```

```
8
13
21
34
55
89
144
233
377
610
$
```

Estimating π

Compute an estimation of π using seven different ways and compare them to the exact value (up to double-precision floating-point binary representation):

```
# computing pi in eight different ways
VECTOR piapprox SIZE 8
VAR x y

# the double-precision internal representation of pi (M_PI)
piapprox(1) = pi

# four times the arc-tangent of the unity
piapprox(2) = 4*atan(1)

# the abscissae x where tan(x) = 0 in the range [3:3.5]
piapprox(3) = root(tan(x), x, 3, 3.5)

# the square of the numerical integral of the guassian curve
piapprox(4) = integral(exp(-x^2), x, -10, 10)^2

# the numerical integral of a circle inscribed in a unit square
piapprox(5) = integral(integral((x^2+y^2) < 1, y, -1, 1), x, -1, 1)

# the numerical integral of a circle parametrized with sqrt(1-x^2)
piapprox(6) = integral(integral(1, y, -sqrt(1-x^2), sqrt(1-x^2)), x, -1, 1)

# the gregory-leibniz sum (one hundred thousand terms)
piapprox(7) = 4*sum((-1)^(i+1)/(2*i-1), i, 1, 1e5)

# the abraham sharp sum (twenty-one terms)
piapprox(8) = sum(2*(-1)^i * 3^(1/2-i)/(2*i+1), i, 0, 20)


PRINT_VECTOR FORMAT "% .20f" piapprox piapprox(i)-pi
```

```
$ wasora pi.was
 3.14159265358979311600  0.00000000000000000000
 3.14159265358979311600  0.00000000000000000000
 3.14159265358979356009  0.00000000000000044409
 3.14159265358991079964  0.00000000000011768364
 3.14176053328579962809  0.00016787969600651209
 3.14159565486785119504  0.00000300127805807904
 3.14158265358971977577 -0.00001000000007334023
 3.14159265359563510955  0.00000000000584199356
$
```

One-dimensional interpolation

First generate some (x, y) pairs with a random component using `gendata1d.was`:

```
# generate pairs of (x,y) data to be used with interp1.was

CONST static_steps
static_steps = round(random_gauss(10,2))
x = step_static/static_steps

PRINT x+random(-0.04,0.04) 1-x+random(-0.2,0.2)+random_gauss(0,0.2)
```

```
$ wasora gendata1d.was > f.dat
```

Then read the generated data and define two functions, one using linear interpolation and another one using akima interpolation:

```
# read file f.dat and define two one-dimensional functions,
# one using linear interpolation (default) and another with akima
FUNCTION f(x) FILE_PATH f.dat INTERPOLATION linear
FUNCTION g(x) FILE_PATH f.dat INTERPOLATION akima

OUTPUT_FILE out f-interp.dat
PRINT_FUNCTION f g FILE out MIN f_a MAX f_b STEP 2e-3
```

```
$ wasora interp1d.was
$ gnuplot -p -e "plot 'f.dat' pt 2 ti 'data',\
                      'f-interp.dat' w l lt 3 ti 'linear',\
                      'f-interp.dat' u 1:3 w l lw 2 lt 7 ti 'akima'"
```

Building an histogram

Take a file with three hundred values of measurements taken with a digital chronometer of the period of a simple pendulum, and build an histogram to see what the distribution of the samples is:

```
# compute an histogram representation of individual samples
# call this input as
# wasora histogram.was histogramdata 300 15 2.25 2.55 | qdp --with histeps
# wasora histogram.was histogramdata 300 15 2.25 2.55 | gnuplot -p -e "plot '< cat' with histeps"
```

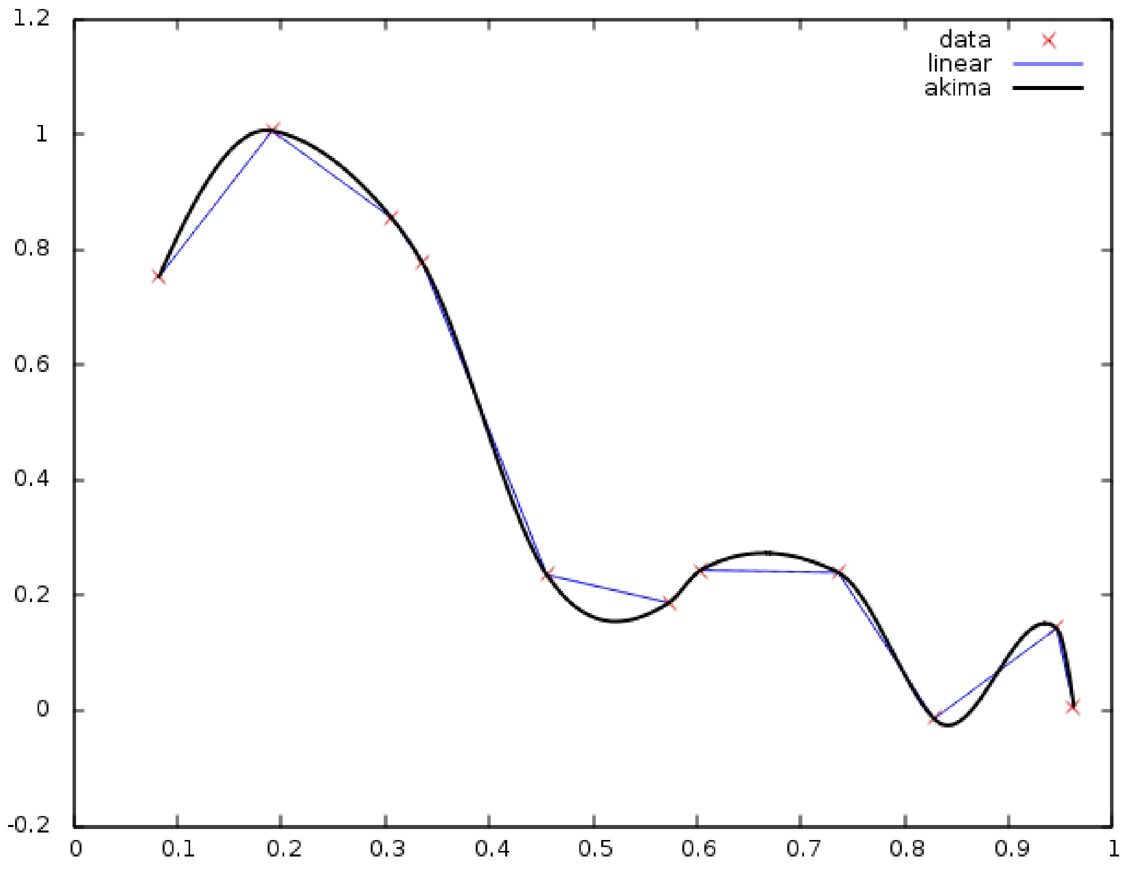

Figure 1: One-dimensional interpolation

```
NUMBER N $2    # number of samples
NUMBER n $3    # number of bins
NUMBER a $4    # lower end of range
NUMBER b $5    # upper end of range

VECTOR x SIZE N     # vector with the actual data
VECTOR hist SIZE n  # histogram bins

# either read data from a file
READ x ASCII_FILE_PATH $1
# or generate some random data
# x(i) = random_gauss(0.5,0.2)

s = (b-a)/n
hist(i) = sum(is_in_interval(x(j), a+(i-1)*s, a+i*s), j, 1, N)

PRINT_VECTOR FORMAT %g a+(i-1/2)*s hist
```

The input file expects five extra arguments that should be provided in the command line. The output wasora gives can be directly piped to gnuplot:

```
$ wasora histogram.was histogram-samples.dat 300 15 2.25 2.55 | gnuplot -p -e "plot '< cat' with hist
```

The logistic map

Build a classical chaotic attractor by parametrically and iteratively solving the logistic map for different values of the parameter r within a certain range:

```
# compute the logistic map for a range of the parameter r

DEFAULT_ARGUMENT_VALUE 1 2.6  # by default compute r in [2.6:4]
DEFAULT_ARGUMENT_VALUE 2 4

# sweep the parameter r between the arguments given in the commandline
# sample 1000 values from a halton quasi-random number sequence
PARAMETRIC r MIN $1 MAX $2 OUTER_STEPS 1000 TYPE halton

static_steps = 800     # for each r compute 800 steps
x_init = 0.5           # start at x = 0.5
x = r*x*(1-x)          # apply the map

# only print x for the last 50 steps to obtain the asymptotic behaviour
IF step_static>static_steps-50
 PRINT %g r x
ENDIF
```

Even though `logistic.was` expects the range of the parameter r to be given in the command line, it defaults to $[2.6 : 4]$ which shows the characteristic period-doubling route to chaos.

```
$ wasora logistic.was | gnuplot -p -e "plot '< cat' pt 0 lt 3 ti ''"
```

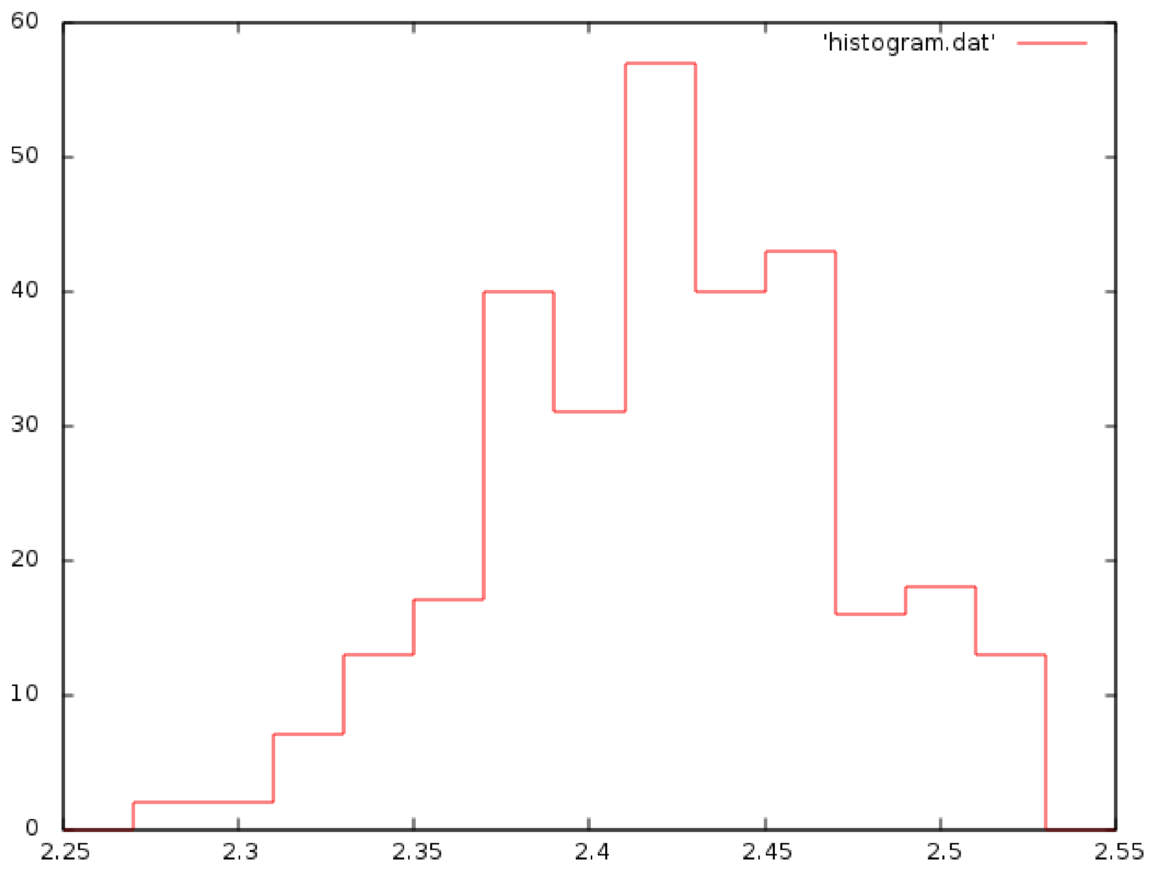

Figure 2: Histogram of experimental data

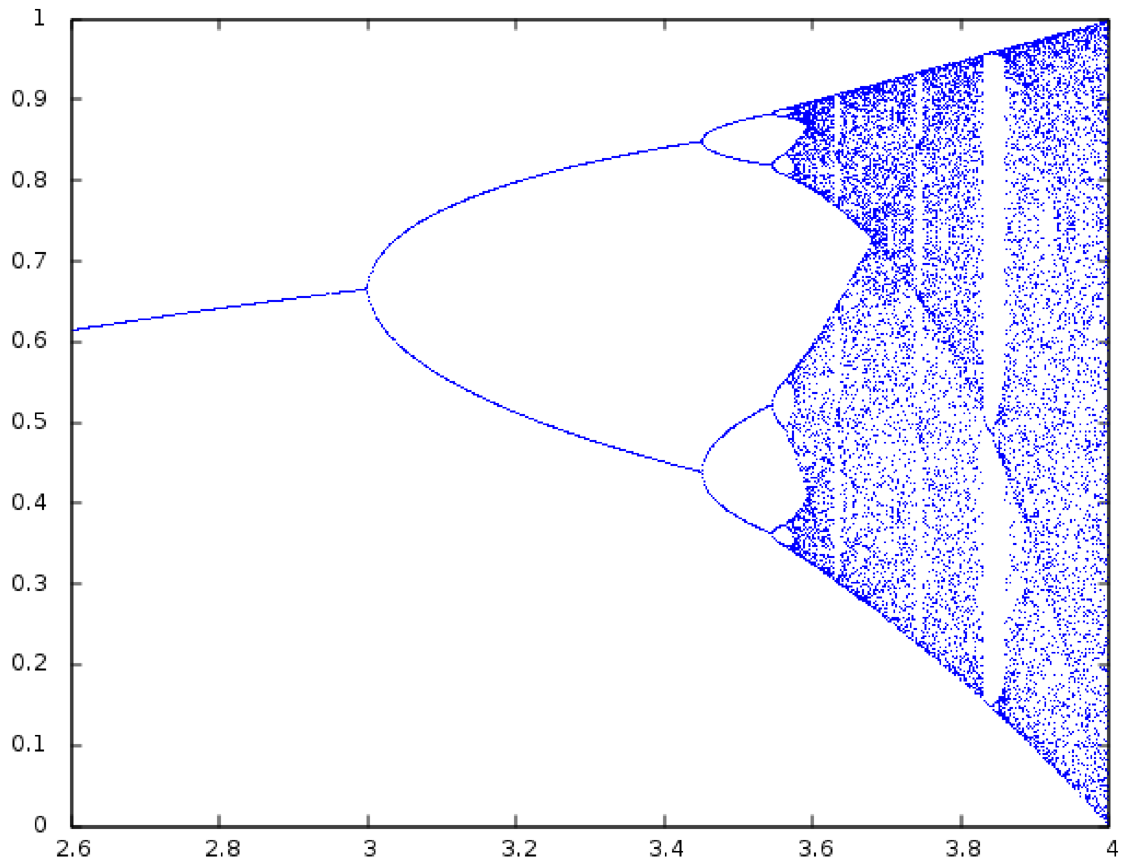

Figure 3: The logistic map

The Lorenz chaotic system

Build your own version of the famous Lorenz attractor that introduced for the first time the idea of what nowadays we call *chaotic attractor* in a humanly-friendly way. First, state which variables belong to the phase space and choose the final integration time. Then define the constant parameters and set the initial conditions. Finally write down the differential equations as naturally as it is possible to do so in a plain-text computer file:

```
# lorenz' seminal dynamical system
PHASE_SPACE x y z
end_time = 40

CONST sigma r b
sigma = 10            # parameters
r = 28
b = 8/3

x_0 = -11             # initial conditions
y_0 = -16
z_0 = 22.5

# the dynamical system
x_dot .= sigma*(y - x)
y_dot .= x*(r - z) - y
z_dot .= x*y - b*z

PRINT t x y z HEADER

# exercise: play with the system! change
# the parameters and plot, plot plot!
```

```
$ wasora lorenz.was | gnuplot -p -e "splot '< cat' u 2:3:4 w l lt 2 ti ''"
```

Math-Ace: the plot thickens

Donald Knuth introduced this mystery equation in one of his books. What does it represent? Find out by solving it parametrically sweeping the $x - y$ plane with a quasi-random number sequence.

```
# Knuth's mystery equation
f(x,y) := {
 ( abs(abs(3-abs(x)) - 3 + abs(x)) + abs(abs(sqrt(abs(9-x^2)) - abs(y-2/3*abs(x))) - sqrt(abs(9-x^2))
 ( abs(abs(x*y) + x*y) +
   ( abs(abs(2 - abs(33-3*abs(x))) - 2 + abs(33-3*abs(x))) + abs(14 - abs(y)) ) *
   ( abs(16 - abs(y) - 3*abs(11 - abs(x))) *
      abs(abs(sqrt(abs(1 - (11 - abs(x))^2)) - abs(11 - abs(y) + 2/3*abs(11 - abs(x)))) - sqrt(abs(1
      + abs(abs(1 - abs(11 - abs(x))) - 1 + abs(11 - abs(x))) )
 )
}

DEFAULT_ARGUMENT_VALUE 1 sobol
```

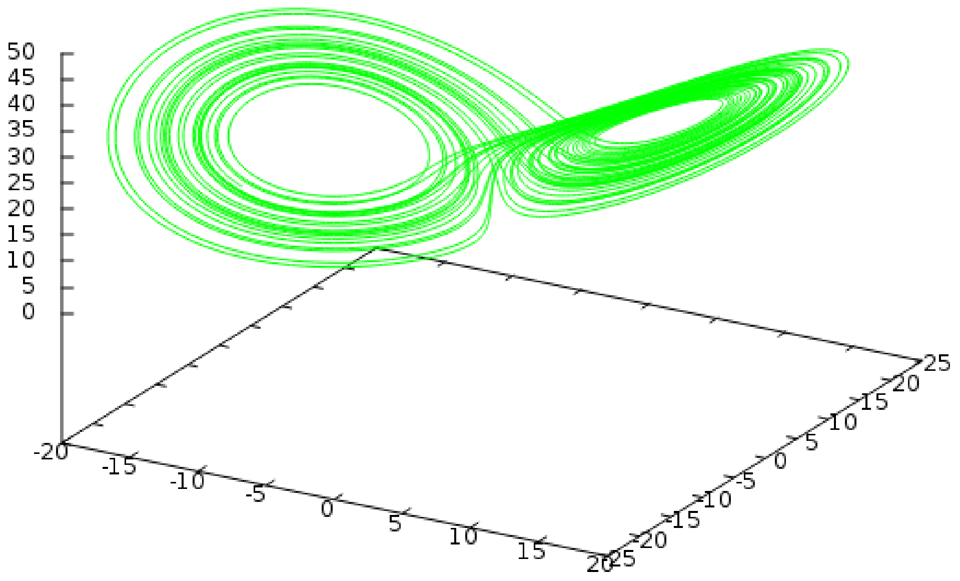

Figure 4: Lorenz' chaotic attractor

```
DEFAULT_ARGUMENT_VALUE 2 2e5

# the range to sweep
PARAMETRIC x y MIN -13 -18 MAX 13 18 TYPE $1 OUTER_STEPS $2

IF abs(f(x,y))<1
 PRINT %f x y
ENDIF
```

```
$ wasora mathace.was  | gnuplot -p -e "plot '< cat' pt 0 ti ''"
```

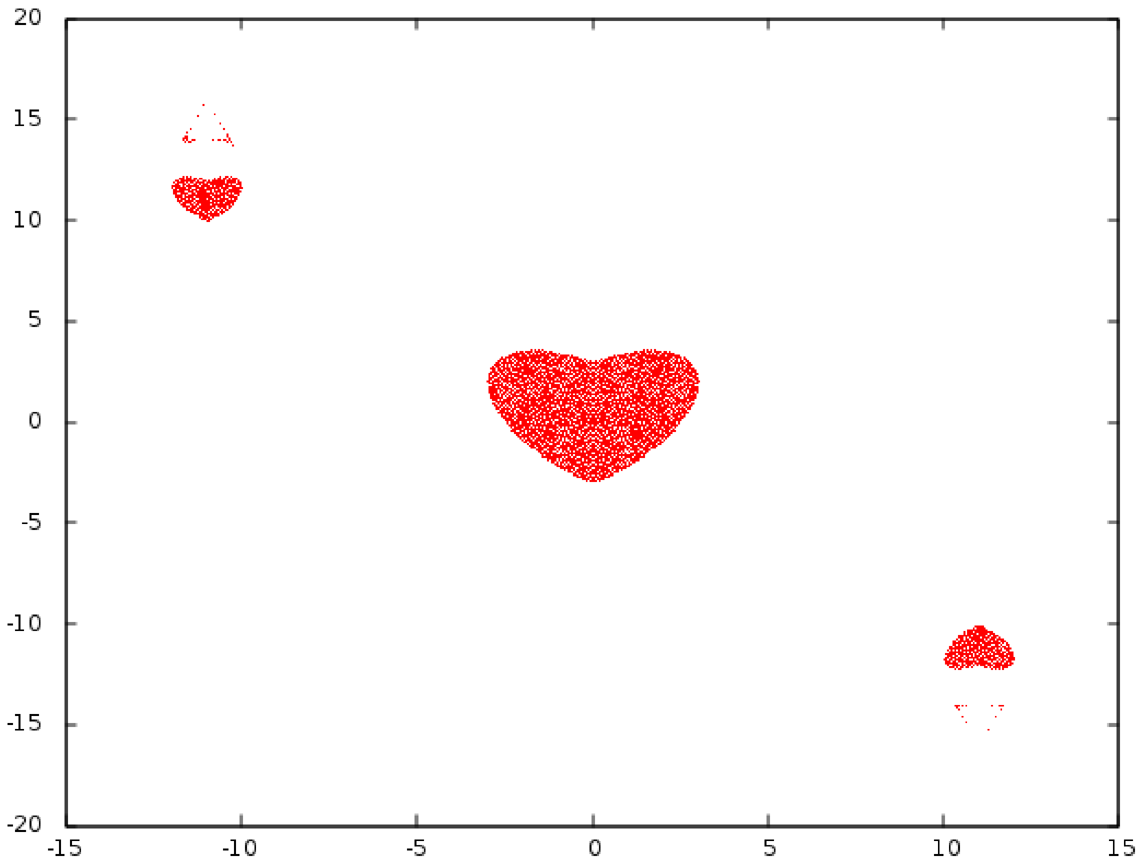

Figure 5: Knuth's mystery equation

Fitting the semi-empirical mass formula

See how wasora can be used to fit multidimensional data sets by finding the the empirical coefficients of Weiszäcker's formula for the binding energy of atomic nuclei:

```
# fit the six parameters of the semi-empirical mass formula to
# experimental binding energy per nucleon data
# initial guess
a1 = 1
a2 = 1
a3 = 1
a4 = 1
a5 = 1
gamma = 1.5

# the functional form of weiszacker's formula
delta(A,Z) := if(is_odd(A), 0, if(is_even(Z), +1, -1))
W(A,Z) := a1 - a2*A^(-1/3) - a3*Z^2*A^(-4/3) - a4*(A-2*Z)^2*A^(-2) + delta(A,Z) * a5*A^(-gamma)

# the experimental data
FUNCTION D(A,Z) FILE_PATH binding_per_A.dat

# fit W to D using the six parameters
FIT W TO D VIA a1 a2 a3 a4 a5 gamma

IF done_outer
 PRINT_FUNCTION D W D(A,Z)-W(A,Z)
ENDIF

$ wasora fsm.was | gnuplot -p -e "set cbrange [0:9]; set view map; \
                                  set xlabel 'A'; set ylabel 'Z'; \
                                  splot '< cat' u 1:2:4 w p pt 57 palette ti ''"
```

Checking wasora's coupling mechanisms

This test checks that wasora can read data from ASCII files, and that the coupling mechanism through semaphore-synchronized shared-memory objects (whose actual type is be OS-dependent) is able to correctly exchange information amongst two instances of wasora. First, an ASCII file with numerical data is created:

```
$ echo 1e-1 1.23456 0.999999999999999 -9.876543210987654321e2 > data.dat
```

One input file, `io-readfile-writeshm.was`, reads the file and writes the data into a shared-memory segment:

```
# read a scalar and a vector of size three from a file and
# send them  to another wasora instance through shared memory
VAR a
VECTOR b SIZE 3

FILE data data.dat MODE r   # define a file id
READ ASCII_FILE data a b    # read data from the file

WRITE SHM_OBJECT data a b   # output a & b to a shm-object called "data"
SEM data-written POST       # post a shared semaphore called "data-written"
SEM data-read    WAIT       # wait until the other end reads the data

PRINT %g a b SEP " "
```

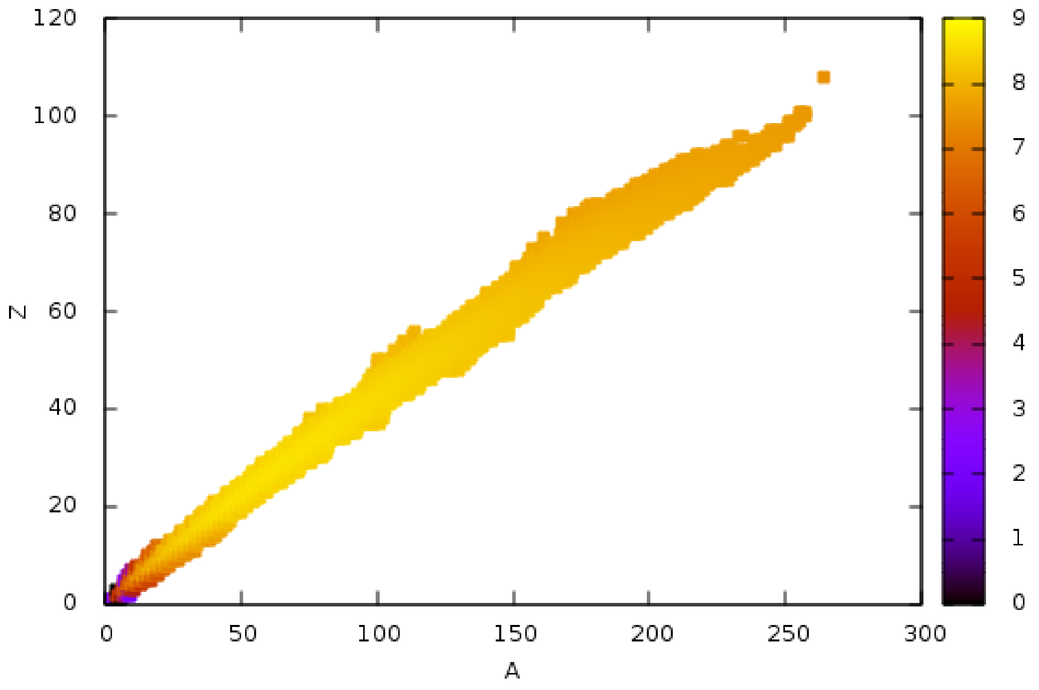

Figure 6: Weizsäcker's semi-empirical mass formula

Another input, `io-readshm.was`, waits until the data is ready to be read from the shared-memory object and writes the data to the standard output:

```
# read a scalar and a vector from a shared-memory object
VAR a
VECTOR b SIZE 3

SEM data-written WAIT       # wait for a semaphore called "data-written" to be posted
READ SHM_OBJECT data a b    # read from a shm-object called "data" a & b
SEM data-read    POST       # post a semaphore called "data-read" to inform the other end

PRINT %g a b SEP " "
```

These two inputs may be executed concurrently in separate terminals, or the first may be executed in background using Bash's ampersand `&` control operator:

```
$ wasora io-readfile-writeshm.was &
[4] 16201
$ wasora io-readshm.was
0.1 1.23456 1 -987.654
0.1 1.23456 1 -987.654
[4]+  Done                    wasora io-readfile-writeshm.was
$
```

Sending the Rössler attractor through shared memory

A further illustration of the coupling scheme wasora proposes—which may be used to couple wasora to other arbitrary codes—is obtained by solving a problem in one input and generating the output in another one. The first one solves the Rössler system:

```
# solve roessler attractor and write the instantenoeus state
# to a shared memory object
end_time = 100

VECTOR x SIZE 3
PHASE_SPACE x

CONST a b c            # system parameters
a = 0.2
b = 0.2
c = 5.7

x_0(i) = i+0.123456    # initial conditions as a function of i

# system of equations written in implicit form
0 = -x_dot(1) - x(2) - x(3)
0 = -x_dot(2) + x(1) + a*x(2)
0 = -x_dot(3) + b + x(3)*(x(1) - c)

# write t, dt, done and vector x to shared object "state"
```

```
WRITE SHM_OBJECT state t dt done x

# tell the receiver the data is ready through a "sender_ready" semaphore
SEM sender_ready   POST

# wait until the other read the data end before advancing another step
# if we are on the first step, we write a message to remind the user
# that the other instance of wasora is to be executed at the same time
IF in_static
 PRINT "\# waiting for other end to read my data..." NONEWLINE
ENDIF

SEM receiver_ready WAIT

IF in_static
 PRINT "ok!"
ENDIF

# note that this input does not produce any output
```

The second input just reads the data generated by the first and writes the status to the standard output at each time step:

```
# receive a 3d phase-space trajectory data from shared memory
# and write the ascii data into the standard output

# we do not know where the data comes, but it should
# be a three-dimensional phase-space trajectory
VAR x y z
# we do not know the end time either, so we start assuming
# we run through infinite but actually end when done is true
end_time = infinite

# print a message to remind the user that another wasora
# sending the data is to be executed
IF in_static
 PRINT "\# waiting for data..." NONEWLINE
ENDIF

SEM sender_ready WAIT

IF in_static
 PRINT "ok!"
ENDIF

# read t, dt, done, x, y and z from shm-object "state"
# note that roessler-sender defines x as a vector of size 3
# we defined x as a scalar, and read x, y and z
# this is a perfectly valid coupling schme!
READ SHM_OBJECT state t dt done x y z
SEM receiver_ready POST
```

```
# print the data so it can be finally plotted
PRINT %f t x y z
```

Again, both inputs may be run in different terminals or in the same one using the `&` operator. Only the second should be used to plot the result:

```
$ wasora roessler-sender.was &
[4] 16672
$ wasora roessler-receiver.was | gnuplot -p -e "splot '< cat' u 2:3:4 w l lt 1 ti ''"
# waiting for other end to read my data...      ok!
[4]+  Done                    wasora roessler-sender.was
$
```

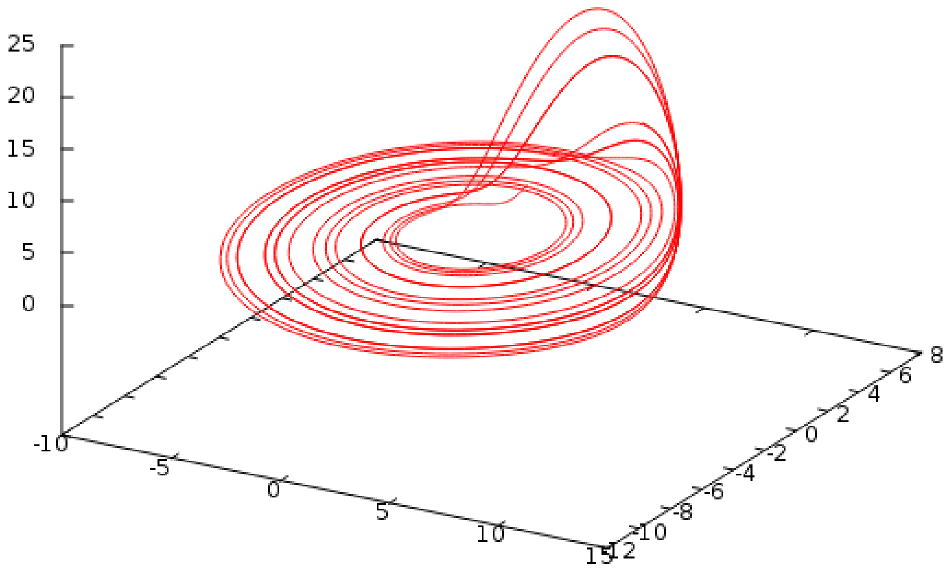

Figure 7: Rössler attractor

The wasora Real Book

As with jazz, wasora is best mastered when played. The wasora Realbook, as the original, introduces fully-usable examples of increasing complexity and difficulty. The examples come with introductions, wasora inputs, terminal mimics, figures and discussions. They range from simple mechanical systems, chaotic attractors and even blackjack strategies. Take a look at the index to see how a certain keyword is used in a real-world application.

- The wasora Realbook
  - About wasora
  - About the examples
- Hello World
  - hello.was
- The Peano axiom
  - peano.was
- Algebraic expressions
  - expr.was
  - hoc.was
- First-order lags
  - lag.was
  - compact.was
  - quasi-sine.was
- A simple differential equation
  - exp.was
  - exp-dt.was
- The Lorenz chaotic system
  - lorenz.was
- The Fibonacci sequence
  - fibo_vector.was
  - fibo_iterative.was
  - fibo_formula.was
- One-dimensional functions
  - algebraic.was
  - inline.was
  - file.was
  - vectors.was
  - satwater.was
- Having fun with integrals
  - sigma.was
  - resonance.was
  - lag-comparison.was
- Two-dimensional functions
  - paraboloid.was
  - nearest.was
  - rectangle.was
  - scattered.was
  - compwater.was
- Newton, Lagrange & Hamilton

- pendulum.was
  - period-vs-amplitude.was
  - pendulum-video.was
- Finding prime numbers
  - checkprimes.was
  - isprime.was
  - nprimes.was
- The double pendulum
  - double.was
  - trajectories.was
- The chaotic Lorenzian waterwheel
  - wheel.was
  - online.was
  - offline.was
  - video.was
- GNU Scientific Library examples rewritten
  - integration.was
  - diff.was
  - roots.was
  - min.was
  - rngunif.was
  - qrng.was
  - stat.was
  - ode-initval.was
  - interp.was
  - interpp.was
  - multimin.was
  - nmsimplex.was
  - nlfit.was
  - siman.was
- Math Ace: the plot thickens
  - mathace.was
- Semi-empirical mass formula fit
  - fit.was
  - compare.was
- Buffon's needle
  - buffon.was
- How lousy is a sixteen?
  - enhc.was
- The vertical boiling channel
  - boiling-2010-eta.was

  - boiling-2010.was
  - boiling-2010-parametric-coarse.was
  - boiling-2010-parametric-fine.was
  - boiling-2012-steady.was
  - boiling-2012-transient.was
  - channel.was

Further information

See the file `INSTALL` for compilation and installation instructions.
See the directory `examples` for the test suite and other examples.
See the contents of directory `doc` for full documentation.

Home page: http://www.talador.com.ar/jeremy/wasora
Mailing list and bug reports: wasora@talador.com.ar