Package ‘GA’

February 19, 2015

Type Package

Version 2.2

Date 2014-10-15

Title Genetic Algorithms

Description
An R package for optimization using genetic algorithms. The package provides a flexible general-purpose set of tools for implementing genetic algorithms search in both the continuous and discrete case, whether constrained or not. Users can easily define their own objective function depending on the problem at hand. Several genetic operators are available and can be combined to explore the best settings for the current task. Furthermore, users can define new genetic operators and easily evaluate their performances. GAs can be run sequentially or in parallel.

Author Luca Scrucca <luca@stat.unipg.it>

Maintainer Luca Scrucca <luca@stat.unipg.it>

Depends R (>= 2.15), methods, foreach, iterators

Suggests parallel, doParallel

License GPL (>= 2)

ByteCompile true

LazyLoad yes

NeedsCompilation no

Repository CRAN

Date/Publication 2014-10-15 13:20:01

R topics documented:

`GA-package` *Genetic Algorithms*

Description

An R package for optimization using genetic algorithms. The package provides a flexible general-purpose set of tools for implementing genetic algorithms search in both the continuous and discrete case, whether constrained or not. Users can easily define their own objective function depending on the problem at hand. Several genetic operators are available and can be combined to explore the best settings for the current task. Furthermore, users can define new genetic operators and easily evaluate their performances. GAs can be run sequentially or in parallel.

References

Scrucca L (2012). GA: A Package for Genetic Algorithms in R. *Journal of Statistical Software*, 53(4), 1-37, `http://www.jstatsoft.org/v53/i04/`.

Author(s)

Luca Scrucca `<luca@stat.unipg.it>`

See Also

The main function `ga`, and the documentation accompanying the package.

`binary2decimal` *Binary encoding of decimal numbers and viceversa.*

Description

Functions for computing binary to decimal conversion of numbers and viceversa.

Usage

```
decimal2binary(x, length)
binary2decimal(x)
```

Arguments

`x`	input value.
`length`	an optional value giving the length of binary string to return.

Details

`decimal2binary` converts a numerical value (which is forced to be an integer) to a binary representation, i.e. a vector of 0s and 1s. For real numerical values see the example below.

`binary2binary` converts a binary value, i.e. a vector of 0s and 1s, to a decimal representation.

Author(s)

Luca Scrucca

See Also

`binary2gray`

Examples

```
# for integer values
dval <- 12
(bval <- decimal2binary(dval))
binary2decimal(bval)

# for real values
dval <- 12.456
# use
(bval <- decimal2binary(dval*1000))
binary2decimal(bval)/1000
```

`binary2gray` *Gray encoding for binary strings*

Description

Functions for computing Gray encoding from/to binary strings.

Usage

```
binary2gray(x)
gray2binary(x)
```

Arguments

`x`	the string to be evaluated

Details

Gray encoding allows to obtain binary strings not affected by the well-known Hamming cliff problem. With Gray encoding the number of bit differences between any two consecutive values is one, whereas in binary strings this is not always true.

Author(s)

Luca Scrucca

See Also

`binary2decimal`

Examples

```
# Consider a five-bit encoding of values 15 and 16  using the standard
# binary coding
decimal2binary(15, 5)
decimal2binary(16, 5)
# Moving from 15 to 16 (or viceversa) all five bits need to be changed,
# but using Gray encoding the two binary strings differ by one bit.
binary2gray(decimal2binary(15, 5))
binary2gray(decimal2binary(16, 5))
```

ga *Genetic Algorithms*

Description

Maximization of a `fitness` function using genetic algorithms.

Usage

```
ga(type = c("binary", "real-valued", "permutation"),
   fitness, ...,
   min, max, nBits,
   population = gaControl(type)$population,
   selection = gaControl(type)$selection,
   crossover = gaControl(type)$crossover,
   mutation = gaControl(type)$mutation,
   popSize = 50,
   pcrossover = 0.8,
   pmutation = 0.1,
   elitism = base::max(1, round(popSize*0.05)),
   maxiter = 100,
   run = maxiter,
   maxfitness = Inf,
   names = NULL,
   suggestions = NULL,
   keepBest = FALSE,
   parallel = FALSE,
   monitor = gaMonitor,
   seed = NULL)
```

Arguments

`type`	the type of genetic algorithm to be run depending on the nature of decision variables. Possible values are: `"binary"` for binary representations of decision variables; `"real-valued"` for optimization problems where the decision variables are floating-point representations of real numbers; `"permutation"` for problems that involves reordering of a list.
`fitness`	the fitness function, any allowable R function which takes as input an individual `string` representing a potential solution, and returns a numerical value describing its “fitness”.
`...`	additional arguments to be passed to the fitness function. This allows to write fitness functions that keep some variables fixed during the search.
`min`	a vector of length equal to the decision variables providing the minimum of the search space in case of real-valued or permutation encoded optimizations.
`max`	a vector of length equal to the decision variables providing the maximum of the search space in case of real-valued or permutation encoded optimizations.

`nBits`	a value specifying the number of bits to be used in binary encoded optimizations.
`population`	an R function for randomly generating an initial population. See `ga_Population` for available functions.
`selection`	an R function performing selection, i.e., a function which generates a new population of individuals from the current population probabilistically according to individual fitness. See `ga_Selection` for available functions.
`crossover`	an R function performing crossover, i.e., a function which forms offsprings by combining part of the genetic information from their parents. See `ga_Crossover` for available functions.
`mutation`	an R function performing mutation, i.e., a function which randomly alters the values of some genes in a parent chromosome. See `ga_Mutation` for available functions.
`popSize`	the population size.
`pcrossover`	the probability of crossover between pairs of chromosomes. Typically this is a large value and by default is set to 0.8.
`pmutation`	the probability of mutation in a parent chromosome. Usually mutation occurs with a small probability, and by default is set to 0.1.
`elitism`	the number of best fitness individuals to survive at each generation. By default the top 5% individuals will survive at each iteration.
`maxiter`	the maximum number of iterations to run before the GA search is halted.
`run`	the number of consecutive generations without any improvement in the best fitness value before the GA is stopped.
`maxfitness`	the upper bound on the fitness function after that the GA search is interrupted.
`names`	a vector of character strings providing the names of decision variables.
`suggestions`	a matrix of solutions strings to be included in the initial population. If provided the number of columns must match the number of decision variables.
`keepBest`	a logical argument specifying if best solutions at each iteration should be saved in a slot called `bestSol`. See `ga-class`.
`parallel`	a logical argument specifying if parallel computing should be used (`TRUE`) or not (`FALSE`, default) for evaluating the fitness function. This argument could also be used to specify the number of cores to employ; by default, this is taken from `detectCores`. Finally, the functionality of parallelization depends on system OS: on Windows only 'snow' type functionality is available, while on Unix/Linux/Mac OSX both 'snow' and 'multicore' (default) functionalities are available.
`monitor`	an R function which takes as input the current state of the `ga` object and show the evolution of the search. By default, the function `gaMonitor` prints the average and best fitness values at each iteration. If set to `plot` these information are plotted on a graphical device. Other functions can be written by the user and supplied as argument.
`seed`	an integer value containing the random number generator state. This argument can be used to replicate the results of a GA search.

Details

Genetic algorithms (GAs) are stochastic search algorithms inspired by the basic principles of biological evolution and natural selection. GAs simulate the evolution of living organisms, where the fittest individuals dominate over the weaker ones, by mimicking the biological mechanisms of evolution, such as selection, crossover and mutation.

The **GA** package is a collection of general purpose functions that provide a flexible set of tools for applying a wide range of genetic algorithm methods.

The `ga` function enables the application of GAs to problems where the decision variables are encoded as `"binary"`, `"real-valued"`, or `"permutation"` strings.

Default genetic operators are set via `gaControl`. To retrieve the currently set operators:

```
gaControl("binary")

gaControl("real-valued")

gaControl("permutation")
```

Value

Returns an object of class `ga-class`. See `ga-class` for a description of available slots information.

Author(s)

Luca Scrucca <`luca@stat.unipg.it`>

References

Back T, Fogel D, Michalewicz Z (2000). *Evolutionary Computation 1: Basic Algorithms and Operators*. IOP Publishing Ltd., Bristol and Philadelphia.

Back T, Fogel D, Michalewicz Z (2000b). *Evolutionary Computation 2: Advanced Algorithms and Operators*. IOP Publishing Ltd., Bristol and Philadelphia.

Coley D (1999). *An Introduction to Genetic Algorithms for Scientists and Engineers*. World Scientific Pub. Co. Inc., Singapore.

Eiben A, Smith J (2003). *Introduction to Evolutionary Computing*. Springer-Verlag, Berlin Heidelberg.

Goldberg D (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Professional, Boston, MA.

Haupt RL, Haupt SE (2004). *Practical Genetic Algorithms*. 2nd edition. John Wiley & Sons, New York.

Scrucca L (2012). GA: A Package for Genetic Algorithms in R. *Journal of Statistical Software*, 53(4), 1-37, `http://www.jstatsoft.org/v53/i04/`.

Sivanandam S, Deepa S (2007). *Introduction to Genetic Algorithms*. Springer-Verlag, Berlin Heidelberg.

Yu X, Gen M (2010). *Introduction to Evolutionary Algorithms*. Springer-Verlag, Berlin Heidelberg.

See Also

summary,ga-method, plot,ga-method, ga-class, ga_Population, ga_Selection, ga_Crossover, ga_Mutation, gaControl.

Examples

```
# 1) one-dimensional function
f <- function(x)  abs(x)+cos(x)
curve(f, -20, 20)

fitness <- function(x) -f(x)
GA <- ga(type = "real-valued", fitness = fitness, min = -20, max = 20)
summary(GA)
plot(GA)

curve(f, -20, 20)
abline(v = GA@solution, lty = 3)

# 2) one-dimensional function
f <- function(x)  (x^2+x)*cos(x) # -10 < x < 10
curve(f, -10, 10)

# write your own tracing function
monitor <- function(obj)
{
  curve(f, -10, 10, main = paste("iteration =", obj@iter))
  points(obj@population, obj@fitness, pch = 20, col = 2)
  rug(obj@population, col = 2)
  Sys.sleep(0.2)
}
## Not run:
GA <- ga(type = "real-valued", fitness = f, min = -10, max = 10, monitor = monitor)

## End(Not run)
# or if you want to suppress the tracing
GA <- ga(type = "real-valued", fitness = f, min = -10, max = 10, monitor = NULL)
summary(GA)

monitor(GA)
abline(v = GA@solution, lty = 3)

# 3) two-dimensional Rastrigin function

Rastrigin <- function(x1, x2)
{
  20 + x1^2 + x2^2 - 10*(cos(2*pi*x1) + cos(2*pi*x2))
}

x1 <- x2 <- seq(-5.12, 5.12, by = 0.1)
f <- outer(x1, x2, Rastrigin)
persp3D(x1, x2, f, theta = 50, phi = 20)
filled.contour(x1, x2, f, color.palette = jet.colors)
```

```
GA <- ga(type = "real-valued", fitness =  function(x) -Rastrigin(x[1], x[2]),
         min = c(-5.12, -5.12), max = c(5.12, 5.12),
         popSize = 50, maxiter = 100)
summary(GA)
plot(GA)

## Parallel GA #######
# Simple example of an expensive fitness function obtained artificially by
# introducing a pause statement.
## Not run:
Rastrigin <- function(x1, x2)
{
  Sys.sleep(0.1)
  20 + x1^2 + x2^2 - 10*(cos(2*pi*x1) + cos(2*pi*x2))
}

system.time(GA1 <- ga(type = "real-valued",
                     fitness =  function(x) -Rastrigin(x[1], x[2]),
                     min = c(-5.12, -5.12), max = c(5.12, 5.12),
                     popSize = 50, maxiter = 100, monitor = FALSE,
                     seed = 12345))

system.time(GA2 <- ga(type = "real-valued",
                     fitness =  function(x) -Rastrigin(x[1], x[2]),
                     min = c(-5.12, -5.12), max = c(5.12, 5.12),
                     popSize = 50, maxiter = 100, monitor = FALSE,
                     seed = 12345, parallel = TRUE))

## End(Not run)
```

ga-class *Class "ga"*

Description

An S4 class for genetic algorithms

Objects from the Class

Objects can be created by calls to the ga function.

Slots

call an object of class "call" representing the matched call;

type a character string specifying the type of genetic algorithm used;

min a vector providing for each decision variable the minimum of the search space in case of real-valued or permutation encoded optimizations;

`max` a vector providing for each decision variable the maximum of the search space in case of real-valued or permutation encoded optimizations;

`nBits` a value specifying the number of bits to be used in binary encoded optimizations;

`names` a vector of character strings providing the names of decision variables (optional);

`popSize` the population size;

`iter` the actual (or final) iteration of GA search;

`run` the number of consecutive generations without any improvement in the best fitness value before the GA is stopped;

`maxiter` the maximum number of iterations to run before the GA search is halted;

`suggestions` a matrix of user provided solutions and included in the initial population;

`population` the current (or final) population;

`elitism` the number of best fitness individuals to survive at each generation;

`pcrossover` the crossover probability;

`pmutation` the mutation probability;

`fitness` the values of fitness function for the current (or final) population;

`summary` a matrix of summary statistics for fitness values at each iteration (along the rows);

`bestSol` if `keepBest = TRUE`, the best solutions at each iteration;

`fitnessValue` the best fitness value at the final iteration;

`solution` the value(s) of the decision variables giving the best fitness at the final iteration.

Author(s)

Luca Scrucca

See Also

For examples of usage see `ga`.

`gaControl` *A function for setting or retrieving defaults genetic operators*

Description

Default settings for genetic operators used in the GA package.

Usage

```
gaControl(...)
```

Arguments

`...`	no arguments, a single character vector, or a named list with components.

Details

If the function is called with no arguments returns the current default settings, i.e., a list with the following default components:

- `"binary"`
  - `population` = `"gabin_Population"`
  - `selection` = `"gabin_lrSelection"`
  - `crossover` = `"gabin_spCrossover"`
  - `mutation` = `"gabin_raMutation"`
- `"real-valued"`
  - `population` = `"gareal_Population"`
  - `selection` = `"gareal_lsSelection"`
  - `crossover` = `"gareal_laCrossover"`
  - `mutation` = `"gareal_raMutation"`
- `"permutation"`
  - `population` = `"gaperm_Population"`
  - `selection` = `"gaperm_lrSelection"`
  - `crossover` = `"gaperm_oxCrossover"`
  - `mutation` = `"gaperm_simMutation"`
- `"eps"` = the tolerance value used by the package functions. By default set at `sqrt(.Machine$double.eps)`.

The function may be called with a single string specifying the name of the component. In this case the function returns the current default settings.

To change the default values, a named component must be followed by a single value (in case of `"eps"`) or a list of component(s) specifying the name of the function for a genetic operator. See the Examples section.

Value

If the argument list is empty the function returns the current list of values. If the argument list is not empty, the returned list is invisible.

Note

The parameter values set via a call to this function will remain in effect for the rest of the session, affecting the subsequent behaviour of the functions for which the given parameters are relevant.

Author(s)

Luca Scrucca

See Also

`ga`

Examples

```
# get and save defaults
defaultControl <- gaControl()
print(defaultControl)
# get current defaults only for binary search
gaControl("binary")
# set defaults for selection operator of binary search
gaControl("binary" = list(selection = "gabin_tourSelection"))
gaControl("binary")
# set defaults for selection and crossover operators of binary search
gaControl("binary" = list(selection = "ga_rwSelection",
                          crossover = "gabin_uCrossover"))
gaControl("binary")
# restore defaults
gaControl(defaultControl)
gaControl()
```

---

gaMonitor *Monitor genetic algorithm evolution*

Description

A function which prints the average and best fitness values at each iteration of GA search.

Usage

```
gaMonitor(object, digits = getOption("digits"), ...)
```

Arguments

`object`	an object of class `"ga"`, usually resulting from a call to function `ga`.
`digits`	minimal number of significant digits.
`...`	further arguments passed to or from other methods.

Value

This function prints a summary of GA steps on the console.

Author(s)

Luca Scrucca

See Also

`ga`

`gaSummary` *Summarize genetic algorithm evolution*

Description

A function which returns fitness summary statistics at each iteration of GA search.

Usage

```
gaSummary(x, ...)
```

Arguments

`x`	a vector of fitness values for which summary statistics should be computed.
`...`	further arguments passed to or from other methods.

Details

This function computes summary statistics for a vector of fitness values at current iteration of GA search.

Value

A vector with the following values: `(max, mean, median, min)`

Author(s)

Luca Scrucca

See Also

`ga`

`ga_Crossover` *Crossover operators in genetic algorithms*

Description

Functions implementing crossover genetic operator.

Usage

```
ga_spCrossover(object, parents, ...)

gabin_spCrossover(object, parents, ...)
gabin_uCrossover(object, parents, ...)

gareal_spCrossover(object, parents, ...)
gareal_waCrossover(object, parents, ...)
gareal_laCrossover(object, parents, ...)
gareal_blxCrossover(object, parents, ...)

gaperm_cxCrossover(object, parents, ...)
gaperm_pmxCrossover(object, parents, ...)
gaperm_oxCrossover(object, parents, ...)
gaperm_pbxCrossover(object, parents, ...)
```

Arguments

`object`	An object of class `"ga"`, usually resulting from a call to function `ga`.
`parents`	A two-rows matrix of values indexing the parents from the current population.
`...`	Further arguments passed to or from other methods.

Value

Return a list with two elements:

`children`	a matrix of dimension 2 times the number of decision variables containing the generated offsprings;
`fitness`	a vector of length 2 containing the fitness values for the offsprings. A value `NA` is returned if an offspring is different (which is usually the case) from the two parents.

Author(s)

Luca Scrucca

See Also

`ga`

ga_Mutation *Mutation operators in genetic algorithms*

Description

Functions implementing mutation genetic operator.

Usage

```
gabin_raMutation(object, parent, ...)

gareal_raMutation(object, parent, ...)
gareal_nraMutation(object, parent, ...)
gareal_rsMutation(object, parent, ...)

gaperm_simMutation(object, parent, ...)
gaperm_ismMutation(object, parent, ...)
gaperm_swMutation(object, parent, ...)
gaperm_dmMutation(object, parent, ...)
gaperm_scrMutation(object, parent, ...)
```

Arguments

`object`	An object of class `"ga"`, usually resulting from a call to function `ga`.
`parent`	A vector of values for the parent from the current population where mutation should occur.
`...`	Further arguments passed to or from other methods.

Value

Return a vector of values containing the mutated string.

Author(s)

Luca Scrucca

See Also

`ga`

ga_pmutation *Variable mutation probability in genetic algorithms*

Description

A function which calculates the mutation probability for the current iteration. This enables to use GAs with variable mutation rate (see examples).

Usage

```
ga_pmutation(object, p0 = 0.5, p = 0.01, T = round(object@maxiter/2), ...)
```

Arguments

object	An object of class "ga", usually resulting from a call to function ga.
p0	initial probability of mutation.
p	limiting probability of mutation.
T	maximum iteration after which it should converges to p.
...	Further arguments passed to or from other methods.

Value

Return a numeric value in the range (0,1).

Author(s)

Luca Scrucca

See Also

ga, ga_Mutation

Examples

```
## Not run:
Rastrigin <- function(x1, x2)
{
  20 + x1^2 + x2^2 - 10*(cos(2*pi*x1) + cos(2*pi*x2))
}

GA <- ga(type = "real-valued",
         fitness =  function(x) -Rastrigin(x[1], x[2]),
         min = c(-5.12, -5.12), max = c(5.12, 5.12),
         popSize = 50, maxiter = 500, run = 100,
         pmutation = ga_pmutation)
plot(GA)

GA <- ga(type = "real-valued",
         fitness =  function(x) -Rastrigin(x[1], x[2]),
         min = c(-5.12, -5.12), max = c(5.12, 5.12),
         popSize = 50, maxiter = 500, run = 100,
         pmutation = function(...) ga_pmutation(..., p0 = 0.1))
plot(GA)

## End(Not run)
```

`ga_Population` *Population initialization in genetic algorithms*

Description

Functions for creating a random initial population to be used in genetic algorithms.

Usage

```
gabin_Population(object, ...)

gareal_Population(object, ...)

gaperm_Population(object, ...)
```

Arguments

`object`	An object of class `"ga"`, usually resulting from a call to function `ga`.
`...`	Further arguments passed to or from other methods.

Details

`gabin_Population` generates a random population of `object@nBits` binary values;

`gareal_Population` generates a random (uniform) population of real values in the range [`object@min`, `object@max`];

`gaperm_Population` generates a random (uniform) population of integer values in the range [`object@min`, `object@max`].

Value

Return a matrix of dimension `object@popSize` times the number of decision variables.

Author(s)

Luca Scrucca

See Also

`ga`

ga_Selection *Selection operators in genetic algorithms*

Description

Functions implementing selection genetic operator.

Usage

```
ga_lrSelection(object, r = 2/(object@popSize * (object@popSize - 1)),
                       q = 2/object@popSize, ...)
ga_nlrSelection(object, q = 0.25, ...)
ga_rwSelection(object, ...)
ga_tourSelection(object, k = 3, ...)

gabin_lrSelection(object, r = 2/(object@popSize * (object@popSize - 1)),
                          q = 2/object@popSize, ...)
gabin_nlrSelection(object, q = 0.25, ...)
gabin_rwSelection(object, ...)
gabin_tourSelection(object, k = 3, ...)

gareal_lrSelection(object, r = 2/(object@popSize * (object@popSize - 1)),
                           q = 2/object@popSize, ...)
gareal_nlrSelection(object, q = 0.25, ...)
gareal_rwSelection(object, ...)
gareal_tourSelection(object, k = 3, ...)
gareal_lsSelection(object, ...)
gareal_sigmaSelection(object, ...)

gaperm_lrSelection(object, r = 2/(object@popSize * (object@popSize - 1)),
                           q = 2/object@popSize, ...)
gaperm_nlrSelection(object, q = 0.25, ...)
gaperm_rwSelection(object, ...)
gaperm_tourSelection(object, k = 3, ...)
```

Arguments

object	An object of class "ga", usually resulting from a call to function ga.
r	A tuning parameter for the specific selection operator.
q	A tuning parameter for the specific selection operator.
k	A tuning parameter for the specific selection operator.
...	Further arguments passed to or from other methods.

Value

Return a list with two elements:

`population`	a matrix of dimension `object@popSize` times the number of decision variables containing the selected individuals or strings;
`fitness`	a vector of length `object@popSize` containing the fitness values for the selected individuals.

Author(s)

Luca Scrucca

See Also

`ga`

`jet.colors` *Jet Colors Palette*

Description

Create a vector of `n` colors beginning with dark blue, ranging through shades of blue, cyan, green, yellow and red, and ending with dark red.

Usage

```
jet.colors(n)
```

Arguments

`n`	a numerical value specifying the number of colors in the palette.

Details

This function creates a palette of colors beginning with dark blue, ranging through shades of blue, cyan, green, yellow and red, and ending with dark red.

Value

Returns vector of `n` color names.

See Also

`colors`.

Examples

```
jet.colors(5)

palette(jet.colors(21))
pie(rep(1,21), col = 1:21)
```

---

numericOrNA-class *Virtual Class "numericOrNA" - Simple Class for subassignment Values*

Description

The class "numericOrNA" is a simple class union (setClassUnion) of "numeric" and "logical".

Objects from the Class

Since it is a virtual Class, no objects may be created from it.

Examples

```
showClass("numericOrNA")
```

---

parNames-methods *Parameters or decision variables names from an object of class ga-class.*

Description

A method for obtaining the names of parameters or decision variables from an object of class ga-class.

Usage

```
parNames(object, ...)
## S4 method for signature 'ga'
parNames(object, ...)
```

Arguments

object	An object of class "ga", usually resulting from a call to function ga.
...	Further arguments, currently not used.

Value

A list of character values providing the names of parameters or decision variables.

Author(s)

Luca Scrucca

See Also

ga

---

persp3D *Perspective plot with colour levels*

Description

This function draws a perspective plot of a surface with different levels in different colors.

Usage

```
persp3D(x, y, z, theta = 30, phi = 20, d = 5, expand = 2/3,
        xlim = range(x, finite = TRUE), ylim = range(y, finite = TRUE),
        zlim = range(z, finite = TRUE), levels = pretty(zlim, nlevels),
        nlevels = 20, color.palette = jet.colors, border = NA,
        ticktype = "detailed", xlab = NULL, ylab = NULL, zlab = NULL,
        ...)
```

Arguments

`x, y`	locations of grid lines at which the values in `z` are measured. These must be in ascending order. By default, equally spaced values from 0 to 1 are used. If `x` is a list, its components `x$x` and `x$y` are used for `x` and `y`, respectively.
`z`	a matrix containing the values to be plotted (NAs are allowed).
`theta, phi`	angles defining the viewing direction. `theta` gives the azimuthal direction and `phi` the colatitude.
`d`	a value which can be used to vary the strength of the perspective transformation.
`expand`	a expansion factor applied to the `z` coordinates.
`xlim, ylim, zlim`	x-, y- and z-limits for the axes.
`levels`	a vector of values specifying the levels to be used for plotting the surface with different colors.
`nlevels`	a value specifying the numbe of levels to be used for plotting. This value is used if `levels` argument is not specified.
`color.palette`	the color palette used for plotting.
`border`	the color of the line drawn around the surface facets. By default is set to `NA` so no borders are drawn.
`ticktype`	a character specifying the type of axes tickmarks. By default `"detailed"` ticks are drawn.

`xlab, ylab, zlab`	character strings specifying the titles for the axes.
`...`	Further arguments passed to the function `persp`.

Details

This function enhances the default perspective plot for drawing 3-dimensional surfaces.

Value

Return a list with the following elements:

`persp`	the viewing transformation matrix (see `link{persp}`);
`levels`	a vector of values giving the levels used for plotting the surface;
`colors`	a vector of strings giving the color used for plotting the surface.

Author(s)

Luca Scrucca

See Also

`link{persp}`

Examples

```
y <- x <- seq(-10, 10, length=60)
f <- function(x,y) { r <- sqrt(x^2+y^2); 10 * sin(r)/r }
z <- outer(x, y, f)
persp3D(x, y, z, theta = 30, phi = 30, expand = 0.5)
persp3D(x, y, z, color.palette = heat.colors, phi = 30, theta = 225,
        box = TRUE, border = NA, shade = .4)

x1 = seq(-3,3,length=50)
x2 = seq(-3,3,length=50)
y = function(x1, x2) sin(x1)+cos(x2)
persp3D(x1, x2, outer(x1,x2,y), zlab="y", theta = 150, phi = 20, expand = 0.6)
```

`plot.ga-method` *Plot of Genetic Algorithm search path*

Description

The `plot` method for `ga-class` objects gives a plot of best and average fitness values found during the iterations of the GA search.

Usage

```
## S4 method for signature 'ga'
plot(x, y, ylim, cex.points = 0.7,
     col = c("green3", "dodgerblue3",  adjustcolor("green3", alpha.f = 0.1)),
     pch = c(16, 1), lty = c(1,2), grid = graphics:::grid, ...)
```

Arguments

`x`	An object of class `"ga"`.
`y`	Not used.
`ylim`	A vector of two values specifying the limits on the y-axis.
`cex.points`	The magnification to be used for points.
`col`	The colors to be used for best and average fitness values.
`pch`	The type of points to be used for best and average fitness values.
`lty`	The type of lines to be used for best and average fitness values.
`grid`	A function for grid drawing of NULL to avoid drawing one.
`...`	Further arguments, currently not used.

Details

Plot best and average fitness values at each iteration of GA search.

Value

The method invisibly return a list with the following components:

`iter`	a vector of values for each interation.
`fitnessBest`	the best value of fitness function at each iteration.
`fitnessMean`	the mean value of fitness function at each iteration.

Author(s)

Luca Scrucca

See Also

`ga`, `ga-class`.

summary.ga-method *Summary for Genetic Algorithms*

Description

Summary method for class "GA".

Usage

```
## S4 method for signature 'ga'
summary(object, ...)

## S3 method for class 'summary.ga'
print(x, digits = getOption("digits"), ...)
```

Arguments

object	an object of class ga-class.
x	an object of class summary.ga.
digits	number of significant digits.
...	further arguments passed to or from other methods.

Value

The summary function returns an object of class summary.ga which can be printed by the corresponding print method. The function also returns invisibly a list with the information from the genetic algorithm search.

Author(s)

Luca Scrucca

See Also

ga

Examples

```
f <- function(x)  abs(x)+cos(x)
GA <- ga(type = "real-valued", fitness = function(x) -f(x), min = -20, max = 20)
out <- summary(GA)
print(out)
str(out)
```

Index