Package: particle.swarm.optimisation (via r-universe)

August 25, 2024

Title Optimisation with Particle Swarm Optimisation

Version 1.0

Description A toolbox to create a particle swarm optimisation (PSO), the package contains two classes: the Particle and the Particle Swarm, this two class is used to run the PSO with methods to easily print, plot and save the result.

License GPL-3

Encoding UTF-8

LazyData false

RoxygenNote 7.1.1

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Imports R6, rgl

Config/testthat/edition 3

NeedsCompilation no

Author Theo Brunet [aut, cre]

Maintainer Theo Brunet <brunet.theo83140@gmail.com>

Date/Publication 2021-05-21 08:00:02 UTC

Repository https://brunettheo.r-universe.dev

RemoteUrl https://github.com/cran/particle.swarm.optimisation

RemoteRef HEAD

RemoteSha e8d7e4b31817a8be4c2e2a52437397c445ee01a8

Contents

Particle 2
ParticleSwarm 4

Index **10**

Particle *Particle*

Description

Class for the Particles used in the Particle Swarm Optimisation, It is call by the Particle Swarm object to make the population.

Active bindings

`values_ranges` (list) max and min for each value of the particle

`values` (numeric) values of the particle (his position in space)

`fitness` (numeric) fitness of the particle (his score)

`fitness_function` (function) function used to find the fitness

`personal_best_values` (numeric) Best values of the particle

`personal_best_fitness` (numeric) Fitness of the best values

`velocity` (numeric) Velocity of the particle (one velocity for each values)

`acceleration_coefficient` (numeric) coefficient c1 and c2 (for personal and global best)

`inertia` (numeric) inertia of the particle

Methods

Public methods:

- `Particle$new()`
- `Particle$get_fitness()`
- `Particle$update()`
- `Particle$update_personal_best_fitness()`
- `Particle$print()`
- `Particle$clone()`

Method `new()`**:** Create a new Particle object.

Usage:

```
Particle$new(
  values_ranges,
  values,
  fitness_function,
  acceleration_coefficient,
  inertia
)
```

Arguments:

`values_ranges` range for each value of the particle (min and max), his size need to be the same as values. (List)

`values`, values of the particles. (numeric)

`fitness_function` function used to test the Particle and find his fitness. (function)

`acceleration_coefficient` a vector of two values, one for c1 (the personal coefficient), and one for c2 (the global coefficient). (numeric)

`inertia` The inertia of the particle (the influence of the previous velocity on the next velocity). (numeric)

Returns: A new `Particle` object.

Method `get_fitness()`**:** Calculate the fitness of the particle with the fitness function and save it in self$fitness

Usage:

```
Particle$get_fitness()
```

Returns: self

Method `update()`**:** Update Particle's position and velocity.

Usage:

```
Particle$update(swarm_best)
```

Arguments:

`swarm_best` the best values of the swarm used to update the velocity

Returns: self

Method `update_personal_best_fitness()`**:** Update the Particle's best values and fitness.

Usage:

```
Particle$update_personal_best_fitness()
```

Returns: self

Method `print()`**:** print the current values of the particle and his fitness

Usage:

```
Particle$print()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Particle$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# If you use the Particle Swarm Object there is no need to manually create the Particle
# But if you want to use the Particle for another project:

# In this example we use the PSO to solve the following equation:
# a * 5 + b * 25 + 10 = 15

fitness_function <- function(values){
```

```
  a <- values[1]
  b <- values[2]
  particule_result <- a*5 + b*25 + 10
  difference <- 15 - particule_result
  fitness <- 1 - abs(difference)
  return(fitness)
}

values_ranges <- list(c(-10^3,10^3),c(-10^3,10^3))

particle_example <- Particle$new(values_ranges = values_ranges,
                                 values = c(0,0),
                                 fitness_function = fitness_function,
                                 acceleration_coefficient = c(0.5,0.5),
                                 inertia = 0.4)
print(particle_example)
particle_example$get_fitness()
print(particle_example)
particle_example$update(c(10,25))
print(particle_example)
```

`ParticleSwarm` *Swarm*

Description

Particle Swarm, used to launch the Particle Swarm Optimisation, The PSO is used to maximise the fitness.

Active bindings

`pop_size` (numeric) number of particles in the swarm

`ranges_of_values` (list) range for each value for the particle

`values_names` (list) list of names for each value (optionnal)

`pop` (list) list of particle in the swarm

`fitness_function` (function) fitness function used to find the fitness of the particle

`list_fitness` (list) list of fitness of the particles

`max_it` (numeric) maximum number of iteration

`acceleration_coefficient_range` (list) coefficient c1 and c2 for the particles

`swarm_best_fitness` (numeric) best fitness of the swarm

`swarm_best_values` (numeric) values of the particle with the best fitness

`inertia` (numeric) inertia of the particles

Methods

Public methods:

- ParticleSwarm$new()
- ParticleSwarm$run()
- ParticleSwarm$generate_pop()
- ParticleSwarm$move_the_swarm()
- ParticleSwarm$save_pop()
- ParticleSwarm$plot_the_swarm_2D()
- ParticleSwarm$plot_the_swarm_3D()
- ParticleSwarm$print()
- ParticleSwarm$clone()

Method `new()`**:** Create a new ParticleSwarm object.

Usage:

```
ParticleSwarm$new(
  pop_size,
  values_names,
  fitness_function,
  max_it,
  acceleration_coefficient_range,
  inertia,
  ranges_of_values
)
```

Arguments:

`pop_size` number of individu in the swarm. (numeric)

`values_names` list of names for each value (character)

`fitness_function` function used to test the Particle and find his fitness. (function)

`max_it` Maximum number of iteration for the PSO. (numeric)

`acceleration_coefficient_range` a vector of four values (min and max for c1 and c2) (numeric)

`inertia` The inertia for the particle (the influence of the previous velocity on the next velocity). (numeric)

`ranges_of_values` range for each value of the particle (min and max). (List)

Returns: A new `ParticleSwarm` object.

Examples:

```
# Create a ParticleSwarm object
swarm <- ParticleSwarm$new(pop_size=20,
                           values_names=c('a','b'),
                           max_it=20,
                  fitness_function = function(values){return(values[1]+values[2])},
                           acceleration_coefficient=list(c(0.5,1),c(0.5,1)),
                           inertia=0.5,
                           ranges_of_values=list(c(-100,100),c(-100,100)))
```

Method `run()`**:** Make the Particle Swarm Optimisation

Usage:

```
ParticleSwarm$run(
  verbose = TRUE,
  plot = TRUE,
  save_file = FALSE,
  dir_name = "PSO_pop"
)
```

Arguments:

`verbose` print the different step (iteration and individu)

`plot` plot the result of each iteration (only for 2D or 3D problem)

`save_file` save the population of each Iteration in a file and save the plot if plot=TRUE

`dir_name` name of the directory, default value is PSO_pop

Returns: self

Examples:

```
# Create a ParticleSwarm object
swarm <- ParticleSwarm$new(pop_size=20,
                           values_names=c('a','b'),
                           max_it=20,
                  fitness_function = function(values){return(values[1]+values[2])},
                           acceleration_coefficient=list(c(0.5,1),c(0.5,1)),
                           inertia=0.5,
                           ranges_of_values=list(c(-100,100),c(-100,100)))
# run the PSO
swarm$run(verbose = FALSE,
          plot = FALSE,
          save_file = FALSE)
# return the best result:
print(swarm$swarm_best_values)
```

Method `generate_pop()`**:** create the population of the swarm (this method is automatically called by the run method)

Usage:

```
ParticleSwarm$generate_pop(verbose = TRUE)
```

Arguments:

`verbose` print the advancement or not

Returns: self

Method `move_the_swarm()`**:** The method used to change the location of each particle (this method is automatically called by the run method)

Usage:

```
ParticleSwarm$move_the_swarm(verbose)
```

Arguments:

`verbose` print or not the advancement

Returns: self

Method `save_pop()`**:** The method used to save the values and fitness of the population in a CSV file (this method is automatically called by the run method if you have chosen to save the result)

Usage:

```
ParticleSwarm$save_pop(nb_it, dir_name)
```

Arguments:

`nb_it` number of the iteration, used to create the name of the csv file

`dir_name` Name of the directory

Returns: self

Method `plot_the_swarm_2D()`**:** method used to plot a 2D plot (this method is automatically called by the run method if you have chosen to plot the swarm)

Usage:

```
ParticleSwarm$plot_the_swarm_2D(nb_it, save_file)
```

Arguments:

`nb_it` number of the iteration used to save the plot as a png

`save_file` save the plot as a file

Returns: self

Method `plot_the_swarm_3D()`**:** method used to plot a 3D plot

Usage:

```
ParticleSwarm$plot_the_swarm_3D(nb_it, save_file)
```

Arguments:

`nb_it` number of the iteration used to save the plot as a png (this method is automatically called by the run method if you have chosen to plot the swarm)

`save_file` save the plot as a file

Returns: self

Method `print()`**:** Print the current result of the population

Usage:

```
ParticleSwarm$print()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
ParticleSwarm$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# In this example we use the PSO to solve the following equation:
# a * 5 + b * 25 + 10 = 15

fitness_function <- function(values){
  a <- values[1]
  b <- values[2]
  particule_result <- a*5 + b*25 + 10
  difference <- 15 - particule_result
  fitness <- 1 - abs(difference)
  return(fitness)
}

values_ranges <- list(c(-10^3,10^3),c(-10^3,10^3))

swarm <- ParticleSwarm$new(pop_size = 200,
                           values_names = list("a","b"),
                           fitness_function = fitness_function,
                           max_it = 75,
                           acceleration_coefficient_range = list(c(0,1),c(0,1)),
                           inertia = 0.5,
                           ranges_of_values = values_ranges)
swarm$run(plot = FALSE,verbose = FALSE,save_file = FALSE)
# the solution is :
swarm$swarm_best_values
swarm$swarm_best_values[[1]]*5 + swarm$swarm_best_values[[2]] *25 + 10

## ------------------------------------------------
## Method `ParticleSwarm$new`
## ------------------------------------------------

# Create a ParticleSwarm object
swarm <- ParticleSwarm$new(pop_size=20,
                           values_names=c('a','b'),
                           max_it=20,
                       fitness_function = function(values){return(values[1]+values[2])},
                           acceleration_coefficient=list(c(0.5,1),c(0.5,1)),
                           inertia=0.5,
                           ranges_of_values=list(c(-100,100),c(-100,100)))

## ------------------------------------------------
## Method `ParticleSwarm$run`
## ------------------------------------------------

# Create a ParticleSwarm object
swarm <- ParticleSwarm$new(pop_size=20,
                           values_names=c('a','b'),
                           max_it=20,
                       fitness_function = function(values){return(values[1]+values[2])},
                           acceleration_coefficient=list(c(0.5,1),c(0.5,1)),
                           inertia=0.5,
                           ranges_of_values=list(c(-100,100),c(-100,100)))
```

```
# run the PSO
swarm$run(verbose = FALSE,
          plot = FALSE,
          save_file = FALSE)
# return the best result:
print(swarm$swarm_best_values)
```

Index

Particle, 2
ParticleSwarm, 4