Package 'globpso'

September 23, 2025

Type Package

Title Swarm Intelligence Optimization

Version 1.3.1

Date 2025-09-23

Author Ping-Yang Chen [aut, cre],
Ray-Bing Chen [aut],
Russell Eberhart [ctb],
Kennedy Kennedy [ctb],
Jun Sun [ctb],
Bin Feng [ctb],
Wenbo Xu [ctb],
Mohammad Reza Bonyadi [ctb],
Zbigniew Michalewicz [ctb],
Ran Cheng [ctb],
Yaochu Jin [ctb],
Milan Stehlík [ctb],
Weng Kee Wong [ctb],
Jozef Kiseľák [ctb],
Rainer Storn [ctb],
Kenneth Price [ctb],
Okazaki Naoaki [cph]

Maintainer Ping-Yang Chen <pychen.ping@gmail.com>

Description A fast and flexible general-purpose implementation of Particle Swarm Optimization (PSO) and Differential Evolution (DE) for solving global minimization problems is provided. It is designed to handle complex optimization tasks with nonlinear, non-differentiable, and multi-modal objective functions defined by users. There are five types of PSO variants: Particle Swarm Optimization (PSO, Eberhart & Kennedy, 1995) <doi:10.1109/MHS.1995.494215>, Quantum-behaved particle Swarm Optimization (QPSO, Sun et al., 2004) <doi:10.1109/CEC.2004.1330875>, Locally convergent rotationally invariant particle swarm optimization (LcRiPSO, Bonyadi & Michalewicz, 2014) <doi:10.1007/s11721-014-0095-1>, Competitive Swarm Optimizer (CSO, Cheng & Jin, 2015) <doi:10.1109/TCYB.2014.2322602> and Double exponential particle swarm optimization (DExPSO, Stehlik et al., 2024) <doi:10.1016/j.asoc.2024.111913>. For the DE algo-

rithm, six types in Storn, R. & Price, K. (1997) <doi:10.1023/A:1008202821328> are included: DE/rand/1, DE/rand/2, DE/best/1, DE/best/2, DE/rand_to-best/1 and DE/rand_to-best/2.

License GPL-3

Encoding UTF-8

Depends R (>= 4.3.0)

Imports Rcpp (>= 1.0.12), RcppArmadillo (>= 0.12.6.6.1), methods, utils

LinkingTo Rcpp, RcppArmadillo,

RoxygenNote 7.3.2

Suggests knitr, rmarkdown

NeedsCompilation yes

Repository CRAN

Date/Publication 2025-09-23 12:20:08 UTC

Contents

`globpso-package` ***globpso**: Particle Swarm Optimization Algorithms and Differential Evolution for Minimization Problems*

Description

A fast and flexible general-purpose implementation of Particle Swarm Optimization (PSO) and Differential Evolution (DE) for solving global minimization problems is provided. It is designed to handle complex optimization tasks with nonlinear, non-differentiable, and multi-modal objective functions defined by users. There are five types of PSO variants: Particle Swarm Optimization (PSO, Eberhart & Kennedy, 1995) <doi:10.1109/MHS.1995.494215>, Quantum-behaved particle Swarm Optimization (QPSO, Sun et al., 2004) <doi:10.1109/CEC.2004.1330875>, Locally convergent rotationally invariant particle swarm optimization (LcRiPSO, Bonyadi & Michalewicz, 2014) <doi:10.1007/s11721-014-0095-1>, Competitive Swarm Optimizer (CSO, Cheng & Jin, 2015) <doi:10.1109/TCYB.2014.232 and Double exponential particle swarm optimization (DExPSO, Stehlik et al., 2024) <doi:10.1016/j.asoc.2024.111913>. For the DE algorithm, six types in Storn, R. & Price, K. (1997) <doi:10.1023/A:1008202821328> are included: DE/rand/1, DE/rand/2, DE/best/1, DE/best/2, DE/rand_to-best/1 and DE/rand_to-best/2.

Author(s)

Ping-Yang Chen <pychen.ping@gmail.com>

References

1. Bonyadi, M. R., & Michalewicz, Z. (2014). A locally convergent rotationally invariant particle swarm optimization algorithm. Swarm intelligence, 8(3), 159-198.
2. Cheng, R., & Jin, Y. (2014). A competitive swarm optimizer for large scale optimization. IEEE transactions on cybernetics, 45(2), 191-204.
3. Shi, Y., & Eberhart, R. (1998, May). A modified particle swarm optimizer. In Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on (pp. 69-73). IEEE.
4. Stehlík, M., Chen, P. Y., Wong, W. K., and Kiseľák, J. (2024). A double exponential particle swarm optimization with non-uniform variates as stochastic tuning and guaranteed convergence to a global optimum with sample applications to finding optimal exact designs in biostatistics. Applied Soft Computing, 163, 111913.
5. Storn, R., & Price, K. (1997). Differential evolution-a simple and efficient heuristic for global optimization over continuous spaces. Journal of global optimization, 11, 341-359.
6. Sun, J., Feng, B., and Xu, W. (2004a). Particle swarm optimization with particles having quantum behavior. In Evolutionary Computation, 2004. CEC2004. Congress on, volume 1, pages 325-331. IEEE.

`diffevo` *Differential Evolution Algorithms for Minimization Problems*

Description

The general-purpose implementation of differential evolution algorithms for minimizing an user-defined objective function.

Usage

```
diffevo(
  objFunc,
  lower,
  upper,
  init = NULL,
  fixed = NULL,
  DE_INFO = NULL,
  seed = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

objFunc	The R or Rcpp compiled objective function. See the example.
lower	The vector of finite lower bounds of the search domain. Its length should be equal to the dimension of the domain space.
upper	The vector of finite upper bounds of the search domain. Its length should be equal to the dimension of the domain space.
init	The vector of initial population. Its length should be equal to the dimension of the domain space. When there are more than one initial vectors, specify init as a matrix. Each row vector represents one initial point. The default for init is NULL.
fixed	The vector of real values and NA values that controls DE to search only for the NA-valued components.
DE_INFO	The list of DE parameters generated by getDEInfo().
seed	The random seed that controls initial population of DE. The default is NULL.
verbose	The logical value controls if DE would reports the updating progress. The default is TRUE.
...	Further arguments to be passed to objFunc.

Value

An List.

par the global best particle.

val the objective function value of the global best particle.

history a vector of objective function values of the global best particle in DE search history.

cputime the computational time in seconds.

References

Storn, R., & Price, K. (1997). Differential evolution-a simple and efficient heuristic for global optimization over continuous spaces. Journal of global optimization, 11, 341-359.

Examples

```
# three-dimensional function
objf <- function(x, loc) {
  val <- 0
  for (i in 1:length(x)) {
    val <- val + (x[i] - loc)^2
  }
  return(val)
}

upp_bound <- rep(5, 3)
low_bound <- rep(-5, 3)
loc_shift <- 1
```

```
alg_setting <- getDEInfo(nPop = 32, maxIter = 100, deType = "rand-1",
                         sf = 0.5, cr = 0.1)
res <- diffevo(objFunc = objf, lower = low_bound, upper = upp_bound,
               DE_INFO = alg_setting, loc = loc_shift)
res$par
res$val

# C++ function example

library(Rcpp)
library(RcppArmadillo)
objf_c <- cppFunction('double objf_c(SEXP x, SEXP loc) {
    double val = 0;
    double loc_c = (double)Rcpp::as<double>(loc);
   arma::rowvec x_c = (arma::rowvec)Rcpp::as<arma::rowvec>(x);
   for (arma::uword i = 0; i < x_c.n_elem; i++) {
     val += (x_c(i) - loc_c)*(x_c(i) - loc_c);
   }
    return val;
 }', depends = "RcppArmadillo")

alg_setting <- getDEInfo(nPop = 32, maxIter = 100, deType = "rand-1",
                         sf = 0.5, cr = 0.1)
res_c <- diffevo(objFunc = objf_c, lower = low_bound, upper = upp_bound,
                       DE_INFO = alg_setting, loc = 1)
res_c$par
res_c$val
```

`getDEInfo` *Generation function of DE parameter settings*

Description

Create a list with DE parameters for Minimization.

Usage

```
getDEInfo(
  nPop = 32,
  maxIter = 100,
  deType = "rand-1",
  freeRun = 1,
  tol = 1e-06,
  sf = 0.5,
  cr = 0.1
)
```

Arguments

nPop	A integer number of population size in DE algorithm.
maxIter	A integer number of maximal DE iterations.
deType	string. The type of DE. This package current supports the following types: **"rand-1"** Mutation operation on the current position with one random direction **"rand-2"** Mutation operation on the current position with two random directions **"best-1"** Mutation operation on the best position with one random direction **"best-2"** Mutation operation on the best position with two random directions **"rand_to-best-1"** Mutation operation on the current position with direction to the best and one random direction **"rand-to-best-2"** Mutation operation on the current position with direction to the best and two random directions The default type is 'rand-1'.
freeRun	A number between $[0, 1]$ that controls the percentage of DE iterations which are free from examining the stopping criterion, $\|f' - f\| < \varepsilon$ where f' and f are the objective function values in the previous and current iterations, respectively. The default is 1.0 implying the DE will completely ignore the stopping criterion. Otherwise, the DE checks the stopping criterion after free iterations.
tol	A small value for the tolerance, ε, in the stopping criterion. For freeRun smaller than 1.0, the default is 1e-6. Otherwise, this value would not affect the algorithm.
sf	The value of scaling factor in DE updating procedure. The default is 0.5.
cr	The value of crossover rate in DE updating procedure. The default is 0.1.

Value

A list of DE parameter settings.

Examples

```
DE_INFO <- getDEInfo(nPop = 32, maxIter = 100)
```

---

getPSOInfo *Generation function of PSO parameter settings*

Description

Create a list with PSO parameters for Minimization.

Usage

```
getPSOInfo(
  nSwarm = 32,
  maxIter = 100,
  psoType = "basic",
  freeRun = 1,
  tol = 1e-06,
  c1 = 2.05,
  c2 = 2.05,
  w0 = 1.2,
  w1 = 0.2,
  w_var = 0.8,
  vk = 4,
  Q_cen_type = 1,
  Q_a0 = 1.7,
  Q_a1 = 0.7,
  Q_a_var = 0.8,
  LcRi_L = 0.01,
  CSO_phi = 0.1,
  TE_b = 2
)
```

Arguments

`nSwarm`	A integer number of swarm size in PSO algorithm.
`maxIter`	A integer number of maximal PSO iterations.
`psoType`	string. The type of PSO. This package current supports the following types: **"basic"** Linearly Decreasing Weight PSO (Eberhart & Kennedy, 1995) **"quantum"** Quantum PSO (Sun et al., 2004) **"lcri"** LcRiPSO (Bonyadi & Michalewicz, 2014) **"comp"** Competitive Swarm Optimization (Cheng & Jin, 2014) **"dexp"** DExPSO (Stehlík et al., 2024)
`freeRun`	A number between $[0, 1]$ that controls the percentage of PSO iterations which are free from examining the stopping criterion, $\|f' - f\| < \varepsilon$ where f' and f are the objective function values in the previous and current iterations, respectively. The default is 1.0 implying the PSO will completely ignore the stopping criterion. Otherwise, the PSO checks the stopping criterion after free iterations.
`tol`	A small value for the tolerance, ε, in the stopping criterion. For `freeRun` smaller than 1.0, the default is `1e-6`. Otherwise, this value would not affect the algorithm.
`c1`	The value of cognitive parameter in PSO updating procedure. The default is 2.05.
`c2`	The value of social parameter in PSO updating procedure. The default is 2.05.
`w0`	The value of starting inertia weight in PSO updating procedure. The default is 1.2.

w1	The value of ending inertia weight in PSO updating procedure. The default is 0.2.
w_var	A number between $[0, 1]$ that controls the percentage of iterations that PSO linearly decreases the inertia weight from w0 to w1. The default is 0.8.
vk	The value of velocity clamping parameter. The default is 4.
Q_cen_type	The type of the center position in QPSO updating procedure (0: local attractor, default; 1: mean best).
Q_a0	The value of starting contraction-expansion (CE) coefficient in QPSO updating procedure. The default is 1.7.
Q_a1	The value of ending contraction-expansion (CE) coefficient in QPSO updating procedure. The default is 0.7.
Q_a_var	A number between $[0, 1]$ that controls the percentage of iterations that QPSO linearly decreases the CE coefficient from Q_a0 to Q_a1. The default is 0.8.
LcRi_L	The value of random number generator based on normal density social parameter in LcRiPSO updating procedure. The default is 0.01. (for psoType = c("lcri") only)
CSO_phi	The value of social parameter in CSO updating procedure. The default is 0.1. (for psoType = c("comp", "cdexp") only)
TE_b	The value of random number generator based on double-exponential density. The default is 2.0. (for psoType = c("dexp", "qdexp", "cdexp") only).

Value

A list of PSO parameter settings.

Examples

```
PSO_INFO <- getPSOInfo(nSwarm = 32, maxIter = 100)
```

---

globpso *Particle Swarm Optimization Algorithms for Minimization Problems*

Description

The general-purpose implementation of particle swarm Optimization algorithms for minimizing an user-defined objective function.

Usage

```
globpso(
  objFunc,
  lower,
  upper,
  init = NULL,
  fixed = NULL,
```

```
  PSO_INFO = NULL,
  seed = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

objFunc	The R or Rcpp compiled objective function. See the example.
lower	The vector of finite lower bounds of the search domain. Its length should be equal to the dimension of the domain space.
upper	The vector of finite upper bounds of the search domain. Its length should be equal to the dimension of the domain space.
init	The vector of initial swarm. Its length should be equal to the dimension of the domain space. When there are more than one initial vectors, specify init as a matrix. Each row vector represents one initial point. The default for init is NULL.
fixed	The vector of real values and NA values that controls PSO to search only for the NA-valued components.
PSO_INFO	The list of PSO parameters generated by getPSOInfo().
seed	The random seed that controls initial swarm of PSO. The default is NULL.
verbose	The logical value controls if PSO would reports the updating progress. The default is TRUE.
...	Further arguments to be passed to objFunc.

Value

An List.

par the global best particle.

val the objective function value of the global best particle.

history a vector of objective function values of the global best particle in PSO search history.

cputime the computational time in seconds.

References

1. Bonyadi, M. R., & Michalewicz, Z. (2014). A locally convergent rotationally invariant particle swarm optimization algorithm. Swarm intelligence, 8(3), 159-198.
2. Cheng, R., & Jin, Y. (2014). A competitive swarm optimizer for large scale optimization. IEEE transactions on cybernetics, 45(2), 191-204.
3. Shi, Y., & Eberhart, R. (1998, May). A modified particle swarm optimizer. In Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on (pp. 69-73). IEEE.

4. Stehlík, M., Chen, P. Y., Wong, W. K., and Kiseľák, J. (2024). A double exponential particle swarm optimization with non-uniform variates as stochastic tuning and guaranteed convergence to a global optimum with sample applications to finding optimal exact designs in biostatistics. Applied Soft Computing, 163, 111913.
5. Sun, J., Feng, B., and Xu, W. (2004a). Particle swarm optimization with particles having quantum behavior. In Evolutionary Computation, 2004. CEC2004. Congress on, volume 1, pages 325-331. IEEE.

Examples

```
# three-dimensional function
objf <- function(x, loc) {
  val <- 0
  for (i in 1:length(x)) {
    val <- val + (x[i] - loc)^2
  }
  return(val)
}

upp_bound <- rep(5, 3)
low_bound <- rep(-5, 3)
loc_shift <- 1

alg_setting <- getPSOInfo(nSwarm = 32, maxIter = 100, psoType = "basic")
res <- globpso(objFunc = objf, lower = low_bound, upper = upp_bound,
               PSO_INFO = alg_setting, loc = loc_shift)
res$par
res$val

# C++ function example

library(Rcpp)
library(RcppArmadillo)
objf_c <- cppFunction('double objf_c(SEXP x, SEXP loc) {
    double val = 0;
    double loc_c = (double)Rcpp::as<double>(loc);
    arma::rowvec x_c = (arma::rowvec)Rcpp::as<arma::rowvec>(x);
    for (arma::uword i = 0; i < x_c.n_elem; i++) {
      val += (x_c(i) - loc_c)*(x_c(i) - loc_c);
    }
    return val;
  }', depends = "RcppArmadillo")

alg_setting <- getPSOInfo(nSwarm = 32, maxIter = 100, psoType = "quantum")
res_c <- globpso(objFunc = objf_c, lower = low_bound, upper = upp_bound,
                 PSO_INFO = alg_setting, loc = 1)
res_c$par
res_c$val
```

Index