Package: EEEA (via r-universe)

June 4, 2026

Type Package

Title Explicit Exploration Strategy for Evolutionary Algorithms

Description Implements an explicit exploration strategy for evolutionary algorithms in order to have a more effective search in solving optimization problems. Along with this exploration search strategy, a set of four different Estimation of Distribution Algorithms (EDAs) are also implemented for solving optimization problems in continuous domains. The implemented explicit exploration strategy in this package is described in Salinas-Gutiérrez and Muñoz Zavala (2023) <doi:10.1016/j.asoc.2023.110230>.

Version 1.0.1

License GPL-3

Encoding UTF-8

RoxygenNote 7.2.1

Imports mvtnorm

Author Rogelio Salinas Gutiérrez [aut, cre, cph] (ORCID: <https://orcid.org/0000-0002-1669-4460>), Pedro Abraham Montoya Calzada [aut, cph] (ORCID: <https://orcid.org/0009-0002-3497-210X>), Angel Eduardo Muñoz Zavala [aut, cph] (ORCID: <https://orcid.org/0000-0002-7484-2097>), Alejandro Fausto Cortés Salinas [aut, cph], Ilse Daniela Saldivar Olvera [aut, cph] (ORCID: <https://orcid.org/0009-0002-2406-2946>)

Maintainer Rogelio Salinas Gutiérrez <rsalinas@correo.uaa.mx>

NeedsCompilation no

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

Date/Publication 2025-10-31 18:10:19 UTC

RemoteUrl https://github.com/cran/EEEA

RemoteRef HEAD

RemoteSha c30657fce2feffbcbd7514eb53127547fdd11b0e

Contents

`ackley` *Ackley Function for Benchmarking Optimization Algorithms*

Description

The Ackley function is a standard benchmark function used for testing optimization algorithms. It is highly multimodal, with a nearly flat outer region and many local minima, making it challenging for global optimization algorithms. Although it can be defined in any number of dimensions, it is commonly evaluated in 4 dimensions in this documentation.

Usage

```
ackley(x)
```

Arguments

`x`	A numeric vector of parameters for which the Ackley function is evaluated.

Value

Returns a numeric value, which is the evaluation of the Ackley function at the input vector `x`.

References

Ackley, D. H. (1987). A Connectionist Machine for Genetic Hillclimbing. *Springer*.

Examples

```
# Evaluation 1: Global minimum point in a four-dimensional space
x <- rep(0, 4)
ackley(x)

# Evaluation 2: A point in a six-dimensional space
x <- c(0, 0.24, 11, -1, -0.7, pi)
ackley(x)

# Contour Plot: Visualizing the Ackley Function
x1 <- seq(-10, 10, length.out = 100)
x2 <- seq(-10, 10, length.out = 100)
z <- outer(x1, x2, FUN = Vectorize(function(x, y) ackley(c(x, y))))
contour(x1, x2, z, nlevels = 20, main = "Contour plot of the Ackley Function")

# EDA.mnorm() example
res = EDA.mnorm(fun = ackley, lower = c(-10,-10), upper = c(10,10), n = 30,
                k = 2, tolerance = 0.01, maxiter = 200)
res$sol

# Contour plot: Visualizing solution with EDA.mnorm()
x1 <- seq(-10, 10, length.out = 100)
x2 <- seq(-10, 10, length.out = 100)
z <- outer(x1, x2, FUN = Vectorize(function(x, y) ackley(c(x, y))))
contour(x1, x2, z, nlevels = 20, cex.axis = 0.8,
        main = "Contour plot of the Ackley Function with EDA.mnorm solution")
points(res$sol[1], res$sol[2], col = "red", pch = 19)
```

`cigar` *Cigar function for optimization problems*

Description

The Cigar function is a test function in optimization characterized by a narrow, elongated valley. This function is useful for testing the performance of optimization algorithms in situations where the minimum is located in a flat region with a high condition number, making it difficult for algorithms to converge efficiently.

Usage

```
cigar(x)
```

Arguments

`x`	A numeric vector of parameters for which the Cigar function is evaluated.

Value

Returns a numeric value, which is the evaluation of the Cigar function at the input vector `x`.

References

Hansen, N., Müller, S. D., & Koumoutsakos, P. (2003). *Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES)*. Evolutionary Computation, 11(1), 1-18.

Examples

```
# Evaluation 1: Global minimum point in a four-dimensional space
x <- rep(0, 4)
cigar(x)

# Evaluation 2: A point in a six-dimensional space
x <- c(0, 0.24, 11, -1, -0.7, pi)
cigar(x)

# Contour Plot: Visualizing the Cigar Function
x1 <- seq(-10, 10, length.out = 100)
x2 <- seq(-10, 10, length.out = 100)
z <- outer(x1, x2, FUN = Vectorize(function(x, y) cigar(c(x, y))))
contour(x1, x2, z, nlevels = 20, main = "Contour of the Cigar Function")

# EDA.mnorm() example
res = EDA.mnorm(fun = cigar, lower = c(-10,-10), upper = c(10,10), n = 30,
                k = 2, tolerance = 0.01, maxiter = 200)
res$sol

# Contour plot: Visualizing solution with EDA.mnorm()
x1 <- seq(-10, 10, length.out = 100)
x2 <- seq(-10, 10, length.out = 100)
z <- outer(x1, x2, FUN = Vectorize(function(x, y) cigar(c(x, y))))
contour(x1, x2, z, nlevels = 20, cex.axis = 0.8,
        main = "Contour plot of the Cigar Function with EDA.mnorm solution")
points(res$sol[1], res$sol[2], col = "red", pch = 19)
```

---

cigar_tablet *Cigar Tablet function for optimization problems*

Description

The Cigar Tablet function is a variant of the Cigar function, designed to present an additional challenge for optimization algorithms. This function features a narrow valley with varying scales in different directions, making it difficult for algorithms to converge due to the anisotropy of the problem.

Usage

```
cigar_tablet(x)
```

Arguments

x	A numeric vector of parameters for which the Cigar Tablet function is evaluated.

Value

Returns a numeric value, which is the evaluation of the Cigar Tablet function at the input vector x.

References

Auger, A., & Hansen, N. (2005). *A restart CMA evolution strategy with increasing population size*. In The 2005 IEEE Congress on Evolutionary Computation, Vol. 2, pp. 1769-1776.

Examples

```
# Evaluation 1: Global minimum point in a four-dimensional space
x <- rep(0, 4)
cigar_tablet(x)

# Evaluation 2: A point in a six-dimensional space
x <- c(0, 0.24, 11, -1, -0.7, pi)
cigar_tablet(x)

# Contour Plot: Visualizing the Cigar Tablet Function
x1 <- seq(-10, 10, length.out = 100)
x2 <- seq(-10, 10, length.out = 100)
z <- outer(x1, x2, FUN = Vectorize(function(x, y) cigar_tablet(c(x, y))))
contour(x1, x2, z, nlevels = 20, main = "Contour of the Cigar Tablet Function")

# EDA.mnorm() example
res = EDA.mnorm(fun = cigar_tablet, lower = c(-10,-10), upper = c(10,10), n = 30,
                k = 2, tolerance = 0.01, maxiter = 200)
res$sol

# Contour plot: Visualizing solution with EDA.mnorm()
x1 <- seq(-10, 10, length.out = 100)
x2 <- seq(-10, 10, length.out = 100)
z <- outer(x1, x2, FUN = Vectorize(function(x, y) cigar_tablet(c(x, y))))
contour(x1, x2, z, nlevels = 20, cex.axis = 0.8,
        main = "Contour plot of the Cigar Tablet Function with EDA.mnorm solution")
points(res$sol[1], res$sol[2], col = "red", pch = 19)
```

`EDA.ecdf` *Distribution estimation algorithm with empirical cumulative distribution function (ECDF).*

Description

Implements an EDA with ECDF.

Usage

```
EDA.ecdf(fun,
  lower,
  upper,
  n = 30,
  maxiter,
  k = 2,
  k0 = 10,
  tolerance = 0.01,
  ...)
```

Arguments

`fun`	A function to be minimized, with first argument the vector of parameters over which minimization is to take place.
`lower`	Lower limits of the parameters.
`upper`	Upper bounds on the parameters.
`n`	Number of individuals per generation
`maxiter`	Maximum number of iterations.
`k`	The minimum number of consecutive generations using the same search distribution.
`k0`	The minimum number of consecutive generations using the uniform distribution.
`tolerance`	Criterion for determining whether the distribution has changed.
`...`	Additional arguments of the objective function.

Details

Implements an EDA with ecdf that uses the proposal presented in Salinas-Gutiérrez and Zavala, 2023 before changing the model.

Value

Returns a list with the following entries:

`sol`	The best solution found.
`par`	The n best individuals generated by the last model.
`value`	The best value of the objective function.
`historical`	The best fitness value in each generatio.

References

Salinas-Gutiérrez, R., & Zavala, A. E. M. (2023).An explicit exploration strategy for evolutionary algorithms. Applied Soft Computing, 140. https://doi.org/10.1016/j.asoc.2023.110230

See Also

EDA.hist, EDA.mnorm, EDA.norm, ExplicitExploration

Examples

```
#Example happy cat
fun <- function(X){
  D <- length(X)
  f <- abs(sum(X^2) - D)^(1/4) + (0.5 * sum(X^2) + sum(X))/D + 0.5
  return(f)
}
n <- 30
k <- 2
tolerance <- 0.01
lower <- rep(-5,2)
upper <- rep(5,2)
res <- EDA.ecdf(fun, lower = lower,
                upper = upper,n = n,
                k = k, tolerance = tolerance,
                maxiter = 200)
z <- outer(X = seq(-5, 5, 0.05), Y = seq(-5, 5, 0.05),
           FUN = Vectorize(function(X, Y) { fun(c(X, Y)) }))
contour(seq(-5, 5, 0.05),seq(-5, 5, 0.05),z,
        nlevels = 20, cex.axis = .8)
points(res$sol[1],res$sol[2],
       col = "blue", pch = 19)

#Multiple regression example
set.seed(pi)
x1 <- rnorm(n = 100, mean = 10, sd = 5)
x2 <- rnorm(n = 100, mean = 8, sd = 2)
y <- 7 + 4 * x1 - 6 * x2 + rnorm(n = 100, mean = 0, sd = 3)

ec <- function(par,y,x1,x2){
  b0 <- par[1]
  b1 <- par[2]
  b2 <- par[3]
  y_hat <- b0 + b1 * x1 + b2 * x2
  value <- sum((y - y_hat)^2)
  return(value)
}

opt <- EDA.ecdf(fun = ec, lower = c(-20,-20,-20),
                upper = c(20,20,20), maxiter = 150,
                y = y, x1 = x1, x2 = x2)
opt$sol
```

`EDA.hist` *Distribution estimation algorithm with histograms.*

Description

Implements an EDA with histograms.

Usage

```
EDA.hist(fun,
  lower,
  upper,
  n = 30,
  maxiter,
  k = 2,
  k0 = 10,
  tolerance = 0.01,
  ...)
```

Arguments

`fun`	A function to be minimized, with first argument the vector of parameters over which minimization is to take place.
`lower`	Lower limits of the parameters.
`upper`	Upper bounds on the parameters.
`n`	Number of individuals per generation
`maxiter`	Maximum number of iterations.
`k`	The minimum number of consecutive generations using the same search distribution.
`k0`	The minimum number of consecutive generations using the uniform distribution.
`tolerance`	Criterion for determining whether the distribution has changed.
`...`	Additional arguments of the objective function.

Details

Implements an EDA with histograms that uses the proposal presented in Salinas-Gutiérrez and Zavala, 2023 before changing the model.

Value

Returns a list with the following entries:

`sol`	The best solution found.
`par`	The n best individuals generated by the last model.
`value`	The best value of the objective function.
`historical`	The best fitness value in each generatio.

References

Salinas-Gutiérrez, R., & Zavala, A. E. M. (2023).An explicit exploration strategy for evolutionary algorithms. Applied Soft Computing, 140. https://doi.org/10.1016/j.asoc.2023.110230

See Also

EDA.ecdf, EDA.mnorm, EDA.norm, ExplicitExploration

Examples

```
#Example happy cat
fun <- function(X){
  D <- length(X)
  f <- abs(sum(X^2) - D)^(1/4) + (0.5 * sum(X^2) + sum(X))/D + 0.5
  return(f)
}
n <- 30
k <- 2
tolerance <- 0.01
lower <- rep(-5,2)
upper <- rep(5,2)
res <- EDA.hist(fun, lower = lower,
                upper = upper,n = n,
                k = k, tolerance = tolerance,
                maxiter = 200)
z <- outer(X = seq(-5, 5, 0.05), Y = seq(-5, 5, 0.05),
           FUN = Vectorize(function(X, Y) { fun(c(X, Y)) }))
contour(seq(-5, 5, 0.05),seq(-5, 5, 0.05),z,
        nlevels = 20, cex.axis = .8)
points(res$sol[1],res$sol[2],
       col = "blue", pch = 19)

#Multiple regression example
set.seed(pi)
x1 <- rnorm(n = 100, mean = 10, sd = 5)
x2 <- rnorm(n = 100, mean = 8, sd = 2)
y <- 7 + 4 * x1 - 6 * x2 + rnorm(n = 100, mean = 0, sd = 3)

ec <- function(par,y,x1,x2){
  b0 <- par[1]
  b1 <- par[2]
  b2 <- par[3]
  y_hat <- b0 + b1 * x1 + b2 * x2
  value <- sum((y - y_hat)^2)
  return(value)
}

opt <- EDA.hist(fun = ec, lower = c(-20,-20,-20),
                upper = c(20,20,20), maxiter = 150,
                y = y, x1 = x1, x2 = x2)
opt$sol
```

`EDA.mnorm` *Distribution estimation algorithm with multivariate normal distribution.*

Description

Implements an EDA with multivariate normal distribution.

Usage

```
EDA.mnorm(fun,
  lower,
  upper,
  n = 30,
  maxiter,
  k = 2,
  k0 = 10,
  tolerance = 0.01,
  ...)
```

Arguments

`fun`	A function to be minimized, with first argument the vector of parameters over which minimization is to take place.
`lower`	Lower limits of the parameters.
`upper`	Upper bounds on the parameters.
`n`	Number of individuals per generation
`maxiter`	Maximum number of iterations.
`k`	The minimum number of consecutive generations using the same search distribution.
`k0`	The minimum number of consecutive generations using the uniform distribution.
`tolerance`	Criterion for determining whether the distribution has changed.
`...`	Additional arguments of the objective function.

Details

Implements an EDA with multivariate normal distribution that uses the proposal presented in Salinas-Gutiérrez and Zavala, 2023 before changing the model.

Value

Returns a list with the following entries:

`sol`	The best solution found.
`par`	The n best individuals generated by the last model.
`value`	The best value of the objective function.
`historical`	The best fitness value in each generatio.

References

Salinas-Gutiérrez, R., & Zavala, A. E. M. (2023).An explicit exploration strategy for evolutionary algorithms. Applied Soft Computing, 140. https://doi.org/10.1016/j.asoc.2023.110230

See Also

EDA.ecdf, EDA.hist, EDA.norm, ExplicitExploration

Examples

```
#Example happy cat
fun <- function(X){
  D <- length(X)
  f <- abs(sum(X^2) - D)^(1/4) + (0.5 * sum(X^2) + sum(X))/D + 0.5
  return(f)
}
n <- 30
k <- 2
tolerance <- 0.01
lower <- rep(-5,2)
upper <- rep(5,2)
res <- EDA.mnorm(fun, lower = lower,
                 upper = upper,n = n,
                 k = k, tolerance = tolerance,
                 maxiter = 200)
z <- outer(X = seq(-5, 5, 0.05), Y = seq(-5, 5, 0.05),
           FUN = Vectorize(function(X, Y) { fun(c(X, Y)) }))
contour(seq(-5, 5, 0.05),seq(-5, 5, 0.05),z,
        nlevels = 20, cex.axis = .8)
points(res$sol[1],res$sol[2],
       col = "blue", pch = 19)

#Multiple regression example
set.seed(pi)
x1 <- rnorm(n = 100, mean = 10, sd = 5)
x2 <- rnorm(n = 100, mean = 8, sd = 2)
y <- 7 + 4 * x1 - 6 * x2 + rnorm(n = 100, mean = 0, sd = 3)

ec <- function(par,y,x1,x2){
  b0 <- par[1]
  b1 <- par[2]
  b2 <- par[3]
  y_hat <- b0 + b1 * x1 + b2 * x2
  value <- sum((y - y_hat)^2)
  return(value)
}

opt <- EDA.mnorm(fun = ec, lower = c(-20,-20,-20),
                 upper = c(20,20,20), maxiter = 150,
                 y = y, x1 = x1, x2 = x2)
opt$sol
```

EDA.norm *Distribution estimation algorithm with normal distribution.*

Description

Implements an EDA with normal distribution.

Usage

```
EDA.norm(fun,
  lower,
  upper,
  n = 30,
  maxiter,
  k = 2,
  k0= 10,
  tolerance = 0.01,
  ...)
```

Arguments

fun	A function to be minimized, with first argument the vector of parameters over which minimization is to take place.
lower	Lower limits of the parameters.
upper	Upper bounds on the parameters.
n	Number of individuals per generation
maxiter	Maximum number of iterations.
k	The minimum number of consecutive generations using the same search distribution.
k0	The minimum number of consecutive generations using the uniform distribution.
tolerance	Criterion for determining whether the distribution has changed.
...	Additional arguments of the objective function.

Details

Implements an EDA with normal distribution that uses the proposal presented in Salinas-Gutiérrez and Zavala, 2023 before changing the model.

Value

Returns a list with the following entries:

sol	The best solution found.
par	The n best individuals generated by the last model.
value	The best value of the objective function.
historical	The best fitness value in each generatio.

References

Salinas-Gutiérrez, R., & Zavala, A. E. M. (2023).An explicit exploration strategy for evolutionary algorithms. Applied Soft Computing, 140. https://doi.org/10.1016/j.asoc.2023.110230

See Also

EDA.ecdf, EDA.hist, EDA.mnorm, ExplicitExploration

Examples

```
#Example happy cat
fun <- function(X){
  D <- length(X)
  f <- abs(sum(X^2) - D)^(1/4) + (0.5 * sum(X^2) + sum(X))/D + 0.5
  return(f)
}
n <- 30
k <- 2
tolerance <- 0.01
lower <- rep(-5,2)
upper <- rep(5,2)
res <- EDA.norm(fun, lower = lower,
                upper = upper,n = n,
                k = k, tolerance = tolerance,
                maxiter = 200)
z <- outer(X = seq(-5, 5, 0.05), Y = seq(-5, 5, 0.05),
           FUN = Vectorize(function(X, Y) { fun(c(X, Y)) }))
contour(seq(-5, 5, 0.05),seq(-5, 5, 0.05),z,
        nlevels = 20, cex.axis = .8)
points(res$sol[1],res$sol[2],
       col = "blue", pch = 19)

#Multiple regression example
set.seed(pi)
x1 <- rnorm(n = 100, mean = 10, sd = 5)
x2 <- rnorm(n = 100, mean = 8, sd = 2)
y <- 7 + 4 * x1 - 6 * x2 + rnorm(n = 100, mean = 0, sd = 3)

ec <- function(par,y,x1,x2){
  b0 <- par[1]
  b1 <- par[2]
  b2 <- par[3]
  y_hat <- b0 + b1 * x1 + b2 * x2
  value <- sum((y - y_hat)^2)
  return(value)
}

opt <- EDA.norm(fun = ec, lower = c(-20,-20,-20),
                upper = c(20,20,20), maxiter = 150,
                y = y, x1 = x1, x2 = x2)
opt$sol
```

`ellipsoid` *Ellipsoid function for optimization problems*

Description

The Ellipsoid function is a standard benchmark function used in optimization. It is convex and has a single global minimum at the origin. The function is a generalization of the quadratic function and is often used to evaluate the performance of optimization algorithms in high-dimensional spaces. Although it can be defined in any number of dimensions, it is commonly evaluated in 4 dimensions in this documentation.

Usage

```
ellipsoid(x)
```

Arguments

`x`	A numeric vector of parameters for which the Ellipsoid function is evaluated.

Value

Returns a numeric value, which is the evaluation of the Ellipsoid function at the input vector `x`.

References

De Jong, K. A. (1975). *An analysis of the behavior of a class of genetic adaptive systems*. Doctoral dissertation, University of Michigan.

Examples

```
# Evaluation 1: Global minimum point in a four-dimensional space
x <- rep(0, 4)
ellipsoid(x)

# Evaluation 2: A point in a six-dimensional space
x <- c(0, 0.24, 11, -1, -0.7, pi)
ellipsoid(x)

# Contour Plot: Visualizing the Ellipsoid Function
x1 <- seq(-10, 10, length.out = 100)
x2 <- seq(-10, 10, length.out = 100)
z <- outer(x1, x2, FUN = Vectorize(function(x, y) ellipsoid(c(x, y))))
contour(x1, x2, z, nlevels = 20, main = "Contour of the Ellipsoid Function")

# EDA.mnorm() example
res = EDA.mnorm(fun = ellipsoid, lower = c(-10,-10), upper = c(10,10), n = 30,
                k = 2, tolerance = 0.01, maxiter = 200)
res$sol
```

```
# Contour plot: Visualizing solution with EDA.mnorm()
x1 <- seq(-10, 10, length.out = 100)
x2 <- seq(-10, 10, length.out = 100)
z <- outer(x1, x2, FUN = Vectorize(function(x, y) ellipsoid(c(x, y))))
contour(x1, x2, z, nlevels = 20, cex.axis = 0.8,
        main = "Contour plot of the Ellipsoid Function with EDA.mnorm solution")
points(res$sol[1], res$sol[2], col = "red", pch = 19)
```

`expanded_schaffer` *Expanded Schaffer's F6 Function for Benchmarking Optimization Algorithms*

Description

Expanded Schaffer's F6 function is a standard benchmark function used for testing optimization algorithms. It is a highly multimodal function with a global minimum at `x = (0, 0, 0, 0)` for 4 dimensions, where the function value is `0.5`.

Usage

```
expanded_schaffer(x)
```

Arguments

`x`	A numeric vector representing the input variables for which the Expanded Schaffer's F6 function is evaluated.

Value

Returns a numeric value representing the evaluation of Expanded Schaffer's F6 function at the input vector `x`.

References

Schaffer, J. D. (1984). Multiple Objective Optimization with Vector Evaluated Genetic Algorithms. In *Proceedings of the First International Conference on Genetic Algorithms*.

Examples

```
# Evaluation 1: Global minimum point in a four-dimensional space
x <- rep(0, 4)
expanded_schaffer(x)

# Evaluation 2: A point in a six-dimensional space
x <- c(0, 0.24, 11, -1, -0.7, pi)
expanded_schaffer(x)

# Contour Plot: Visualizing Expanded Schaffer's F6 Function
x1 <- seq(-10, 10, length.out = 100)
```

```
x2 <- seq(-10, 10, length.out = 100)
z <- outer(x1, x2, FUN = Vectorize(function(x, y) expanded_schaffer(c(x, y))))
contour(x1, x2, z, nlevels = 20, main = "Contour of Expanded Schaffer's F6 Function")

# EDA.mnorm() example
res = EDA.mnorm(fun = expanded_schaffer, lower = c(-10,-10), upper = c(10,10), n = 30,
                k = 2, tolerance = 0.01, maxiter = 200)
res$sol

# Contour plot: Visualizing solution with EDA.mnorm()
x1 <- seq(-10, 10, length.out = 100)
x2 <- seq(-10, 10, length.out = 100)
z <- outer(x1, x2, FUN = Vectorize(function(x, y) expanded_schaffer(c(x, y))))
contour(x1, x2, z, nlevels = 20, cex.axis = 0.8,
      main = "Contour plot of the Expanded Schaffer's F6 Function with EDA.mnorm solution")
points(res$sol[1], res$sol[2], col = "red", pch = 19)
```

`ExplicitExploration` *Explicit exploration in evolutionary algorithms*

Description

It makes an explicit exploration as proposed in Salinas-Gutiérrez and Zavala, 2023.

Usage

```
ExplicitExploration(
  fun,
  lower,
  upper,
  n = 30,
  maxiter,
  k = 5,
 tolerance = 0.01,
 ...
)
```

Arguments

`fun`	A function to be minimized, with first argument the vector of parameters over which minimization is to take place.
`lower`	Lower bounds on the parameters.
`upper`	Upper bounds on the parameters.
`n`	Number of individuals per generation
`maxiter`	Maximum number of iterations.
`k`	Number of consecutive generations without change in distribution before stopping.
`tolerance`	Criterion for determining whether the distribution has changed.
`...`	Additional arguments of the objective function.

Value

Returns a list with the following entries:

`par`	The top n individuals from the entire search.
`Y`	The value of the objective function for each of the best individuals.
`n_gen`	Number of generations required for the search.
`par_historical`	All individuals generated during the search.
`historical`	The value of the objective function for all generated individuals.

References

Salinas-Gutiérrez, R., & Zavala, A. E. M. (2023).An explicit exploration strategy for evolutionary algorithms. Applied Soft Computing, 140. https://doi.org/10.1016/j.asoc.2023.110230

Examples

```
fun <- function(X){
  D <- length(X)
  f <- abs(sum(X^2) - D)^(1/4) + (0.5 * sum(X^2) + sum(X))/D + 0.5
  return(f)
}

n <- 30
k <- 2
tolerance <- 0.01
lower <- c(-5,-5)
upper <- c(5,5)
res <- ExplicitExploration(fun, lower = lower,
                           upper = upper,n = n,
                           maxiter = 20,
                           k = k)

z <- outer(X = seq(-5, 5, 0.05), Y = seq(-5, 5, 0.05),
           FUN = Vectorize(function(X, Y) { fun(c(X, Y)) }))

contour(seq(-5, 5, 0.05),seq(-5, 5, 0.05),z,
        nlevels = 20, cex.axis = .8)
points(res$par_historical[,1],res$par_historical[,2],
       col = "blue")
points(res$par[,1],res$par[,2], col = "red",
       pch = 19)
```

griewank *Griewank Function for Benchmarking Optimization Algorithms*

Description

The Griewank function is a common test function for optimization algorithms. It combines a sum-of-squares term and a cosine-based product term, making it a useful benchmark for exploring search space properties and assessing the performance of optimization methods on functions with numerous local minima.

Usage

```
griewank(x)
```

Arguments

x	A numeric vector representing the input variables. The length of x determines the dimensionality of the problem.

Value

Returns a numeric value representing the evaluation of the Griewank function at the given input vector x.

References

Griewank, A. O. (1981). Generalized Descent for Global Optimization. *Journal of Optimization Theory and Applications*, 34(1), 11–39.

Examples

```
# Evaluation 1: Global minimum point in a four-dimensional space
x <- rep(0, 4)
griewank(x)

# Evaluation 2: A point in a six-dimensional space
x <- c(0, 0.24, 11, -1, -0.7, pi)
griewank(x)

# Contour Plot: Visualizing the Griewank Function
x1 <- seq(-10, 10, length.out = 100)
x2 <- seq(-10, 10, length.out = 100)
z <- outer(x1, x2, FUN = Vectorize(function(x, y) griewank(c(x, y))))
contour(x1, x2, z, nlevels = 20, main = "Contour of the Griewank Function")

# EDA.mnorm() example
res = EDA.mnorm(fun = griewank, lower = c(-10,-10), upper = c(10,10), n = 30,
                k = 2, tolerance = 0.01, maxiter = 200)
res$sol
```

```
# Contour plot: Visualizing solution with EDA.mnorm()
x1 <- seq(-10, 10, length.out = 100)
x2 <- seq(-10, 10, length.out = 100)
z <- outer(x1, x2, FUN = Vectorize(function(x, y) griewank(c(x, y))))
contour(x1, x2, z, nlevels = 20, cex.axis = 0.8,
        main = "Contour plot of the Griewank Function with EDA.mnorm solution")
points(res$sol[1], res$sol[2], col = "red", pch = 19)
```

`happy_cat` *Happy Cat Function for Benchmarking Optimization Algorithms*

Description

The Happy Cat function is a multimodal test function commonly used to assess the performance of optimization algorithms. It features a complex landscape that combines quadratic terms and interactions between variables, making it suitable for testing optimization methods in high-dimensional spaces.

Usage

```
happy_cat(x)
```

Arguments

`x`	A numeric vector representing the input variables. The length of `x` determines the dimensionality of the problem.

Value

Returns a numeric value representing the evaluation of the Happy Cat function at the input vector `x`.

References

Beyer, H.-G., & Finck, S. (2012). HappyCat – A Simple Function Class Where Well-Known Direct Search Algorithms Do Fail. In Parallel Problem Solving from Nature (pp. 367–376). Springer.

Examples

```
# Evaluation 1: Global minimum point in a four-dimensional space
x <- rep(0, 4)
happy_cat(x)

# Evaluation 2: A point in a six-dimensional space
x <- c(0, 0.24, 11, -1, -0.7, pi)
happy_cat(x)

# Contour Plot: Visualizing the Happy Cat Function
```

```
x1 <- seq(-10, 10, length.out = 100)
x2 <- seq(-10, 10, length.out = 100)
z <- outer(x1, x2, FUN = Vectorize(function(x, y) happy_cat(c(x, y))))
contour(x1, x2, z, nlevels = 20, main = "Contour plot of Happy Cat Function")

# EDA.mnorm() example
res = EDA.mnorm(fun = happy_cat, lower = c(-10,-10), upper = c(10,10), n = 30,
                k = 2, tolerance = 0.01, maxiter = 200)
res$sol

# Contour plot: Visualizing solution with EDA.mnorm()
x1 <- seq(-10, 10, length.out = 100)
x2 <- seq(-10, 10, length.out = 100)
z <- outer(x1, x2, FUN = Vectorize(function(x, y) happy_cat(c(x, y))))
contour(x1, x2, z, nlevels = 20, cex.axis = 0.8,
        main = "Contour plot of the Happy Cat Function with EDA.mnorm solution")
points(res$sol[1], res$sol[2], col = "red", pch = 19)
```

`modified_schwefel` *Modified Schwefel Function for Benchmarking Optimization Algorithms*

Description

The modified Schwefel function is a challenging test function for optimization algorithms due to its complex landscape with numerous local minima. It is a modified version of the original Schwefel function, further increasing its difficulty.

Usage

```
modified_schwefel(x)
```

Arguments

`x`	A numeric vector of parameters for which the modified Schwefel function is evaluated.

Value

Returns a numeric value representing the evaluation of the modified Schwefel function at the input vector `x`.

References

Schwefel, H.-P. (1981). Numerical Optimization of Computer Models. *John Wiley & Sons*.

Examples

```
# Evaluation 1: Global minimum point in a four-dimensional space
x <- rep(-420.9687462275036, 4)
modified_schwefel(x)

# Evaluation 2: A point in a six-dimensional space
x <- c(0, 0.24, 11, -1, -0.7, pi)
modified_schwefel(x)

# Contour Plot: Visualizing the Modified Schwefel Function
x1 <- seq(-10, 10, length.out = 100)
x2 <- seq(-10, 10, length.out = 100)
z <- outer(x1, x2, FUN = Vectorize(function(x, y) modified_schwefel(c(x, y))))
contour(x1, x2, z, nlevels = 20, main = "Contour of the Modified Schwefel Function")

# EDA.mnorm() example
res = EDA.mnorm(fun = modified_schwefel, lower = c(-10,-10), upper = c(10,10), n = 30,
                k = 2, tolerance = 0.01, maxiter = 200)
res$sol

# Contour plot: Visualizing solution with EDA.mnorm()
x1 <- seq(-10, 10, length.out = 100)
x2 <- seq(-10, 10, length.out = 100)
z <- outer(x1, x2, FUN = Vectorize(function(x, y) modified_schwefel(c(x, y))))
contour(x1, x2, z, nlevels = 20, cex.axis = 0.8,
        main = "Contour plot of the Modified Schwefel Function with EDA.mnorm solution")
points(res$sol[1], res$sol[2], col = "red", pch = 19)
```

`rastrigin` *Rastrigin function for optimization problems*

Description

The Rastrigin function is a standard benchmark function used for testing optimization algorithms. It is a non-convex function with a global minimum at the origin. Although it can be defined in any number of dimensions, it is commonly evaluated in 4 dimensions in this documentation. Note that the typical search domain for the Rastrigin function is `[-5.12, 5.12]`, which is used for visualization purposes.

Usage

```
rastrigin(x)
```

Arguments

`x`	A numeric vector of parameters for which the Rastrigin function is evaluated.

Value

Returns a numeric value, which is the evaluation of the Rastrigin function at the input vector `x`.

References

Rastrigin, L. A. (1974). Systems of extremal control. Mir.

Examples

```
# Evaluation 1: Global minimum point in a four-dimensional space
x <- rep(0, 4)
rastrigin(x)

# Evaluation 2: A point in a six-dimensional space
x <- c(0, 0.24, 11, -1, -0.7, pi)
rastrigin(x)

# Contour Plot: Visualizing the Rastrigin Function
x1 <- seq(-10, 10, length.out = 100)
x2 <- seq(-10, 10, length.out = 100)
z <- outer(x1, x2, FUN = Vectorize(function(x, y) rastrigin(c(x, y))))
contour(x1, x2, z, nlevels = 20, main = "Contour of the Rastrigin Function")

# EDA.mnorm() example
res = EDA.mnorm(fun = rastrigin, lower = c(-10,-10), upper = c(10,10), n = 30,
                k = 2, tolerance = 0.01, maxiter = 200)
res$sol

# Contour plot: Visualizing solution with EDA.mnorm()
x1 <- seq(-10, 10, length.out = 100)
x2 <- seq(-10, 10, length.out = 100)
z <- outer(x1, x2, FUN = Vectorize(function(x, y) rastrigin(c(x, y))))
contour(x1, x2, z, nlevels = 20, cex.axis = 0.8,
        main = "Contour plot of the Rastrigin Function with EDA.mnorm solution")
points(res$sol[1], res$sol[2], col = "red", pch = 19)
```

---

rosenbrock *Rosenbrock function for optimization problems*

Description

The Rosenbrock function, also known as the "Rosenbrock's valley" or "banana function", is a non-convex function commonly used as a performance test for optimization algorithms. It features a narrow, curved valley that contains the global minimum, which makes it particularly challenging for optimization methods. The global minimum is located at the point where all variables are equal to 1.

Usage

```
rosenbrock(x)
```

Arguments

x	A numeric vector of parameters for which the Rosenbrock function is evaluated.

Value

Returns a numeric value, which is the evaluation of the Rosenbrock function at the input vector x.

References

Rosenbrock, H. H. (1960). *An Automatic Method for Finding the Greatest or Least Value of a Function*. The Computer Journal, 3(3), 175–184. doi:10.1093/comjnl/3.3.175

Examples

```
# Evaluation 1: Global minimum point in a four-dimensional space
x <- rep(1, 4)
rosenbrock(x)

# Evaluation 2: A point in a six-dimensional space
x <- c(0, 0.24, 11, -1, -0.7, pi)
rosenbrock(x)

# Contour Plot: Visualizing the Rosenbrock Function
x1 <- seq(-10, 10, length.out = 100)
x2 <- seq(-10, 10, length.out = 100)
z <- outer(x1, x2, FUN = Vectorize(function(x, y) rosenbrock(c(x, y))))
contour(x1, x2, z, nlevels = 20, main = "Contour of the Rosenbrock Function")

# EDA.mnorm() example
res = EDA.mnorm(fun = rosenbrock, lower = c(-10,-10), upper = c(10,10), n = 30,
                k = 2, tolerance = 0.01, maxiter = 200)
res$sol

# Contour plot: Visualizing solution with EDA.mnorm()
x1 <- seq(-10, 10, length.out = 100)
x2 <- seq(-10, 10, length.out = 100)
z <- outer(x1, x2, FUN = Vectorize(function(x, y) rosenbrock(c(x, y))))
contour(x1, x2, z, nlevels = 20, cex.axis = 0.8,
        main = "Contour plot of the Rosenbrock Function with EDA.mnorm solution")
points(res$sol[1], res$sol[2], col = "red", pch = 19)
```

---

schwefel_12 *Schwefel 1.2 function for optimization problems*

Description

The Schwefel 1.2 function is a commonly used benchmark function in optimization problems. It is non-convex and has a global minimum at the origin. The function is characterized by a cumulative sum of squared terms, making it challenging for optimization algorithms.

Usage

```
schwefel_12(x)
```

Arguments

x	A numeric vector of parameters for which the Schwefel 1.2 function is evaluated.

Value

Returns a numeric value representing the evaluation of the Schwefel 1.2 function at the input vector x.

References

Schwefel, H.-P. (1981). *Numerical Optimization of Computer Models*. Wiley.

Examples

```
# Evaluation 1: Global minimum point in a four-dimensional space
x <- rep(0, 4)
schwefel_12(x)

# Evaluation 2: A point in a six-dimensional space
x <- c(0, 0.24, 11, -1, -0.7, pi)
schwefel_12(x)

# Contour Plot: Visualizing the Schwefel 1.2 Function
x1 <- seq(-10, 10, length.out = 100)
x2 <- seq(-10, 10, length.out = 100)
z <- outer(x1, x2, FUN = Vectorize(function(x, y) schwefel_12(c(x, y))))
contour(x1, x2, z, nlevels = 20, main = "Contour of the Schwefel 1.2 Function")

# EDA.mnorm() example
res = EDA.mnorm(fun = schwefel_12, lower = c(-10,-10), upper = c(10,10), n = 30,
                k = 2, tolerance = 0.01, maxiter = 200)
res$sol

# Contour plot: Visualizing solution with EDA.mnorm()
x1 <- seq(-10, 10, length.out = 100)
x2 <- seq(-10, 10, length.out = 100)
z <- outer(x1, x2, FUN = Vectorize(function(x, y) schwefel_12(c(x, y))))
contour(x1, x2, z, nlevels = 20, cex.axis = 0.8,
        main = "Contour plot of the Schwefel 1.2 Function with EDA.mnorm solution")
points(res$sol[1], res$sol[2], col = "red", pch = 19)
```

---

sphere *Sphere Function for Benchmarking Optimization Algorithms*

Description

The Sphere function is one of the simplest and most commonly used benchmark functions in optimization. It is a unimodal, convex function with a single global minimum, making it useful for testing the performance of optimization algorithms, especially in smooth, continuous search spaces.

Usage

```
sphere(x)
```

Arguments

x	A numeric vector representing the input variables for which the Sphere function is evaluated.

Value

Returns a numeric value representing the evaluation of the Sphere function at the given input vector x.

References

De Jong, K. A. (1975). *Analysis of the Behavior of a Class of Genetic Adaptive Systems*. Ph.D. dissertation, University of Michigan.

Examples

```
# Evaluation 1: Global minimum point in a four-dimensional space
x <- rep(0, 4)
sphere(x)

# Evaluation 2: A point in a six-dimensional space
x <- c(0, 0.24, 11, -1, -0.7, pi)
sphere(x)

# Contour Plot: Visualizing the Sphere Function
x1 <- seq(-10, 10, length.out = 100)
x2 <- seq(-10, 10, length.out = 100)
z <- outer(x1, x2, FUN = Vectorize(function(x, y) sphere(c(x, y))))
contour(x1, x2, z, nlevels = 20, main = "Contour of the Sphere Function")

# EDA.mnorm() example
res = EDA.mnorm(fun = sphere, lower = c(-10,-10), upper = c(10,10), n = 30,
                k = 2, tolerance = 0.01, maxiter = 200)
res$sol

# Contour plot: Visualizing solution with EDA.mnorm()
x1 <- seq(-10, 10, length.out = 100)
x2 <- seq(-10, 10, length.out = 100)
z <- outer(x1, x2, FUN = Vectorize(function(x, y) sphere(c(x, y))))
contour(x1, x2, z, nlevels = 20, cex.axis = 0.8,
        main = "Contour plot of the Sphere Function with EDA.mnorm solution")
points(res$sol[1], res$sol[2], col = "red", pch = 19)
```

trid *Trid function for optimization problems*

Description

The Trid function is a benchmark function used in optimization. It is a non-convex function with a global minimum at a specific point within its domain. The function has a parabolic shape with cross terms that introduce dependencies between variables, making it a challenging test case for optimization algorithms.

Usage

```
trid(x)
```

Arguments

x	A numeric vector of parameters for which the Trid function is evaluated.

Value

Returns a numeric value representing the evaluation of the Trid function at the input vector x.

References

Aluffi-Pentini, F., Parisi, V., & Zirilli, F. (1985). Global optimization and stochastic differential equations. *Journal of Optimization Theory and Applications*, 47(1), 1-16.

Examples

```
# Evaluation 1: Global minimum point in a four-dimensional space
x <- c(-1, -0.333333, 0.333333, 1)
trid(x)

# Evaluation 2: A point in a six-dimensional space
x <- c(0, 0.24, 11, -1, -0.7, pi)
trid(x)

# Contour Plot: Visualizing the Trid Function
x1 <- seq(-10, 10, length.out = 100)
x2 <- seq(-10, 10, length.out = 100)
z <- outer(x1, x2, FUN = Vectorize(function(x, y) trid(c(x, y))))
contour(x1, x2, z, nlevels = 20, main = "Contour of the Trid Function")

# EDA.mnorm() example
res = EDA.mnorm(fun = trid, lower = c(-10,-10), upper = c(10,10), n = 30,
                k = 2, tolerance = 0.01, maxiter = 200)
res$sol

# Contour plot: Visualizing solution with EDA.mnorm()
```

```
x1 <- seq(-10, 10, length.out = 100)
x2 <- seq(-10, 10, length.out = 100)
z <- outer(x1, x2, FUN = Vectorize(function(x, y) trid(c(x, y))))
contour(x1, x2, z, nlevels = 20, cex.axis = 0.8,
        main = "Contour plot of the Trid Function with EDA.mnorm solution")
points(res$sol[1], res$sol[2], col = "red", pch = 19)
```

---

two_axes *Two Axes function for optimization problems*

Description

The Two Axes function is a benchmark function used in optimization. It is characterized by two components with different scaling factors, making the optimization problem anisotropic. The function presents a challenge due to its structure, with two regions requiring different search strategies for optimal convergence.

Usage

```
two_axes(x)
```

Arguments

x	A numeric vector of parameters for which the Two Axes function is evaluated.

Value

Returns a numeric value, which is the evaluation of the Two Axes function at the input vector x.

References

Hansen, N., & Ostermeier, A. (2001). *Completely derandomized self-adaptation in evolution strategies*. Evolutionary Computation, 9(2), 159-195.

Examples

```
# Evaluation 1: Global minimum point in a four-dimensional space
x <- rep(0, 4)
two_axes(x)

# Evaluation 2: A point in a six-dimensional space
x <- c(0, 0.24, 11, -1, -0.7, pi)
two_axes(x)

# Contour Plot: Visualizing the Two Axes Function
x1 <- seq(-10, 10, length.out = 100)
x2 <- seq(-10, 10, length.out = 100)
z <- outer(x1, x2, FUN = Vectorize(function(x, y) two_axes(c(x, y))))
contour(x1, x2, z, nlevels = 20, main = "Contour of the Two Axes Function")
```

```
# EDA.mnorm() example
res = EDA.mnorm(fun = two_axes, lower = c(-10,-10), upper = c(10,10), n = 30,
                k = 2, tolerance = 0.01, maxiter = 200)
res$sol

# Contour plot: Visualizing solution with EDA.mnorm()
x1 <- seq(-10, 10, length.out = 100)
x2 <- seq(-10, 10, length.out = 100)
z <- outer(x1, x2, FUN = Vectorize(function(x, y) two_axes(c(x, y))))
contour(x1, x2, z, nlevels = 20, cex.axis = 0.8,
        main = "Contour plot of the Two Axes Function with EDA.mnorm solution")
points(res$sol[1], res$sol[2], col = "red", pch = 19)
```

`weierstrass` *Weierstrass Function for Benchmarking Optimization Algorithms*

Description

The Weierstrass function is a continuous but non-differentiable function commonly used for testing optimization algorithms. It is highly multimodal, with numerous local minima, making it challenging for optimization algorithms to locate the global minimum.

Usage

```
weierstrass(x,
            a = 0.5,
            b = 3,
            kmax = 20)
```

Arguments

`x`	A numeric vector representing the input variables. The length of `x` determines the dimensionality of the problem.
`a`	A numeric value representing the scaling factor. Typically, `a = 0.5`.
`b`	A numeric value representing the frequency factor. Typically, `b = 3`.
`kmax`	An integer specifying the maximum number of terms in the summation. Typically, `kmax = 20`.

Value

Returns a numeric value representing the evaluation of the Weierstrass function at the given input vector `x`.

References

Weierstrass, K. (1872). On Continuous Non-Differentiable Functions. In *Mathematische Werke*, 2(1), 71–74.

Examples

```
# Evaluation 1: Global minimum point in a four-dimensional space
x <- rep(0, 4)
weierstrass(x)

# Evaluation 2: A point in a six-dimensional space
x <- c(0, 0.24, 11, -1, -0.7, pi)
weierstrass(x)

# Contour Plot: Visualizing the Weierstrass Function
x1 <- seq(-10, 10, length.out = 100)
x2 <- seq(-10, 10, length.out = 100)
z <- outer(x1, x2, FUN = Vectorize(function(x, y) weierstrass(c(x, y))))
contour(x1, x2, z, nlevels = 20, main = "Contour of the Weierstrass Function")

# EDA.mnorm() example
res = EDA.mnorm(fun = weierstrass, lower = c(-10,-10), upper = c(10,10), n = 30,
                k = 2, tolerance = 0.01, maxiter = 200)
res$sol

# Contour plot: Visualizing solution with EDA.mnorm()
x1 <- seq(-10, 10, length.out = 100)
x2 <- seq(-10, 10, length.out = 100)
z <- outer(x1, x2, FUN = Vectorize(function(x, y) weierstrass(c(x, y))))
contour(x1, x2, z, nlevels = 20, cex.axis = 0.8,
        main = "Contour plot of the Weierstrass Function with EDA.mnorm solution")
points(res$sol[1], res$sol[2], col = "red", pch = 19)
```

---

zakharov *Zakharov function for optimization problems*

Description

The Zakharov function is a benchmark function used in optimization problems, known for its smoothness and separability. The function consists of a sum of squared terms and additional polynomial terms, making it useful for testing algorithms designed to handle non-separable problems with smooth landscapes.

Usage

```
zakharov(x)
```

Arguments

x	A numeric vector of parameters for which the Zakharov function is evaluated.

Value

Returns a numeric value, which is the evaluation of the Zakharov function at the input vector x.

References

Zakharov, A. (1973). *A benchmark test for optimization.* Russian Journal of Numerical Mathematics, 18(3), 206-211.

Examples

```
# Evaluation 1: Global minimum point in a four-dimensional space
x <- rep(0, 4)
zakharov(x)

# Evaluation 2: A point in a six-dimensional space
x <- c(0, 0.24, 11, -1, -0.7, pi)
zakharov(x)

# Contour Plot: Visualizing the Zakharov Function
x1 <- seq(-10, 10, length.out = 100)
x2 <- seq(-10, 10, length.out = 100)
z <- outer(x1, x2, FUN = Vectorize(function(x, y) zakharov(c(x, y))))
contour(x1, x2, z, nlevels = 20, main = "Contour of the Zakharov Function")

# EDA.mnorm() example
res = EDA.mnorm(fun = zakharov, lower = c(-10,-10), upper = c(10,10), n = 30,
                k = 2, tolerance = 0.01, maxiter = 200)
res$sol

# Contour plot: Visualizing solution with EDA.mnorm()
x1 <- seq(-10, 10, length.out = 100)
x2 <- seq(-10, 10, length.out = 100)
z <- outer(x1, x2, FUN = Vectorize(function(x, y) zakharov(c(x, y))))
contour(x1, x2, z, nlevels = 20, cex.axis = 0.8,
        main = "Contour plot of the Zakharov Function with EDA.mnorm solution")
points(res$sol[1], res$sol[2], col = "red", pch = 19)
```

Index