Package ‘MaOEA’

October 12, 2022

Type Package

Title Many Objective Evolutionary Algorithm

Version 0.6.2

Maintainer Dani Irawan <irawan_dani@yahoo.com>

Description A set of evolutionary algorithms to solve many-objective optimization. Hybridization between the algorithms are also facilitated. Available algorithms are: 'SMS-EMOA’ <doi:10.1016/j.ejor.2006.08.008> 'NSGA-III' <doi:10.1109/TEVC.2013.2281535> 'MO-CMA-ES' <doi:10.1145/1830483.1830573>


License GPL (>= 3)

Encoding UTF-8

LazyData true

BugReports https://github.com/dots26/MaOEA/issues

URL https://github.com/dots26/MaOEA

RoxygenNote 7.1.1

Imports reticulate, nsga2R, lhs, nnet, stringr, randtoolbox, e1071, MASS, gtools, stats, utils, pracma

Suggests testthat

SystemRequirements Python 3.x with following modules: PyGMO, NumPy, and cloudpickle

NeedsCompilation no

Author Dani Irawan [aut, cre] (<https://orcid.org/0000-0002-4213-941X>)

Repository CRAN

Date/Publication 2020-08-31 09:40:07 UTC
R topics documented:

MaOEA-package ................................................................. 3
AdaptiveNormalization ...................................................... 4
cmaes_gen ................................................................. 4
compute_R2HV ............................................................. 6
compute_R2HVC ............................................................. 7
compute_R2mtch ............................................................. 8
createWeights .............................................................. 9
createWeightsSobol ...................................................... 10
DTLZ1 ................................................................. 10
DTLZ2 ................................................................. 11
DTLZ3 ................................................................. 12
DTLZ4 ................................................................. 12
EvaluateIndividual ......................................................... 13
EvaluatePopulation ......................................................... 14
GetHVContribution ........................................................ 14
GetHypervolume ............................................................ 15
GetIGD ................................................................. 16
GetLeastContribution ...................................................... 17
GetLeastContributor ....................................................... 18
InitializePopulationLHS ................................................... 19
install_python_dependencies ............................................ 20
load_python_dependencies ............................................... 20
MOCMAES ............................................................. 21
Normalize ............................................................... 22
NSGA3 ............................................................... 23
optimMaOEAm .......................................................... 24
SMOCMAES ............................................................ 26
SMSEMOA ............................................................. 27
WFG1 ............................................................... 29
WFG2 ............................................................... 29
WFG4 ............................................................... 30
WFG5 ............................................................... 31
WFG6 ............................................................... 32
WFG7 ............................................................... 32
WFG8 ............................................................... 33
WFG9 ............................................................... 34

Index .............................................................................. 35
MaOEA-package  Many-Objective Evolutionary Algorithm

Description
MaOEA contains several algorithms for solving many-objective optimization problems. The algorithms are provided as a sequence of operators used in a single iteration. For example, the SMSEMOA function calls the recombination (SBX) and mutation operator (polynomial mutation) to produce 1 offspring, and perform the S-metric selection. The function then returns a list containing the population and population objective after the procedure is conducted once. The purpose of only doing a single iteration is to support users if they wish to formulate hybrid algorithms.

Details
Alternatively, users can use the optimMaOEA function to solve an optimization problem with their chosen algorithm. This function is a simple wrapper to call the algorithms listed above for several iterations. Using this function, users can simply supply the initial population, objective function, the chosen algorithm, and the number of iterations. If number of iteration is not supplied, then only a single iteration is conducted.

Note: This package uses column-major ordering, i.e. an individual should be contained in a single column, each row represents different variable. All optimization variable should be scaled to 0-1.

Acknowledgments
This work is funded by the European Commission’s H2020 programme through the UTOPIAE Marie Curie Innovative Training Network, H2020-MSCA-ITN-2016, under Grant Agreement No. 722734 as well as through the Twinning project SYNERGY under Grant Agreement No. 692286.

Maintainer
Dani Irawan <irawan_dani@yahoo.com>

Author(s)
Dani Irawan <irawan_dani@yahoo.com>

See Also
Main interface function is optimMaOEA.
AdaptiveNormalization  *Objective space normalization.*

**Description**

Normalize the objectives to 0-1. The origin is the ideal point. (1,...,1) is not the nadir point. The normalization is done by using adaptive normalization used in NSGA-III.

**Usage**

`AdaptiveNormalization(objectiveValue)`

**Arguments**

- `objectiveValue`: Set of objective vectors to normalize

**Value**

A list containing the following: `normalizedObjective` The normalized values
- `idealPoint`: The ideal point corresponding to the origin
- `nadirPoint`: The location of nadir point in the normalized space

**Examples**

```r
nObj <- 5
nIndividual <- 100
nVar <- 10
population <- InitializePopulationLHS(nIndividual,nVar,FALSE)
objective <- matrix(,nrow=nObj,ncol=nIndividual)
for(individual in 1:nIndividual){
    objective[,individual] <- WFG4(population[,individual],nObj)
}
AdaptiveNormalization(objective)
```

cmaes_gen  *Generator for cmaes_gen class.*

**Description**

Create a list with `cmaes_gen` class. Basically, the function transforms the population into a class that is accepted by the MOCMAES and SMOCMAES function.
Usage

cmaes_gen(
  population,
  ps_target = (1/(5 + (1/2)^0.5)),
  stepSize = 0.5,
  evoPath = rep(0, nrow(population)),
  covarianceMatrix = diag(nrow(population))
)

Arguments

  population: The number of objective functions. A scalar value.
  ps_target: The target success rate. Used to initialize cmaes_gen$averageSuccessRate.
  stepSize: The initial step size.
  evoPath: A vector of numbers indicating evolution path of each variable.
  covarianceMatrix: Covariance matrix of the variables.

Value

An object of cmaes_gen class. It can be used as MO-CMA-ES parent. It is a 5 tuple: x (the
design point, length = number of variable),averageSuccessRate (scalar),stepSize (scalar), evoPath
(evolution path, vector, length = number of variable),covarianceMatrix (square matrix with ncol =
nrow = number of variable).

Examples

nVar <- 14
nObjective <- 5
nIndividual <- 100
crossoverProbability <- 1
ps_target <- 1 / (5 + (1 / 2 )^0.5
)  # create the population
a_list <- cmaes_gen(pop)
control <- list(successProbTarget=ps_target,crossoverProbability=crossoverProbability)

# run a generation of MO-CMA-ES with standard WFG8 test function.
numpyready <- reticulate::py_module_available('numpy')
pygmoready <- reticulate::py_module_available('pygmo')
py_module_ready <- numpyready & pygmoready
if(py_module_ready) # prevent error on testing the example
  newGeneration <- MOCMAES(a_list,nObjective,WFG8,control,nObjective)
compute_R2HV

*Modified powered tchebyscheff R2-indicator designed to approximate HV*

**Description**

Compute the R2-HV from Shang et al.

**Usage**

`compute_R2HV(dataPoints, reference, weights = NULL, nPoints = 100)`

**Arguments**

- `dataPoints` : The Points coordinate. Each column contains a single point (column major).
- `reference` : The reference point for computing R2-mtch (similar as reference for HV)
- `weights` : The weights/direction to be used to compute the achievement scalarization. Each column contains a single weight vector. If no weight is supplied, weights are generated using Sobol sequences.
- `nPoints` : Used only when no weights are supplied. An input for the weight generator (sobol sequences). This defines how many points are created.

**Value**

The function return the powered R2-indicator of the set.

**References**


**Examples**

```r
nPointToSample <- 100
nObjective <- 3
points <- matrix(runif(nPointToSample*nObjective), nrow = nObjective) # sample the points
ranks <- nsga2R::fastNonDominatedSorting(t(points)) # non-dominated sorting
points <- points[, ranks[[1]], drop=FALSE] # take only the non-dominated front
nPoints <- ncol(points) # check how many points are on the non-dominated front
reference <- rep(2, nObjective)

compute_R2HV(points, reference)
```
**compute_R2HVC**

*Modified tchebyscheff R2-indicator contribution designed to approximate HV*

**Description**

Compute the R2-HVC from Shang et al.

**Usage**

```r
compute_R2HVC(
    dataPoints,
    reference,
    weights = NULL,
    alpha = 1,
    nWeight = 300,
    indexOfInterest = 1:ncol(dataPoints)
)
```

**Arguments**

- `dataPoints`  
The Points coordinate. Each column contains a single point (column major).
- `reference`  
The reference point for computing R2-mtch (similar as reference for HV)
- `weights`  
The weights/direction to be used to compute the achievement scalarization. Each column contains a single weight vector. If no weight is supplied, weights are generated using Sobol sequences
- `alpha`  
Power factor on the gmtch and g*2tch utility functions.
- `nWeight`  
Used only when no weights are supplied. The number of weights generated by sobol sequence.
- `indexOfInterest`  
individuals to be evaluated. The R2 values will only be reported/returned for these individuals.

**Value**

The function return R2-indicator contribution of each point.

**References**

Examples

```r
nPointToSample <- 100
nObjective <- 3
points <- matrix(runif(nPointToSample*nObjective), nrow = nObjective) # sample the points
ranks <- nsga2R::fastNonDominatedSorting(t(points)) # non-dominated sorting
points <- points[, ranks[[1]], drop=FALSE] # take only the non-dominated front
nPoints <- ncol(points) # check how many points are on the non-dominated front
reference <- rep(2, nObjective)

compute_R2HVC(points, reference)
```

compute_R2mtch  
**Modified tchebyscheff R2-indicator**

Description

Compute the R2-mtch indicator from Shang et al.

Usage

```r
compute_R2mtch(dataPoints, reference, weights = NULL, nWeight = 100)
```

Arguments

- `dataPoints`: The Points coordinate. Each column contains a single point (column major).
- `reference`: The reference point for computing R2-mtch (similar as reference for HV)
- `weights`: The weights/direction to be used to compute the achievement scalarization. Each column contains a single weight vector. If no weight is supplied, weights are generated using Sobol sequences.
- `nWeight`: Used only when no weights are supplied. An input for the sobol weight generation. This defines how many points to be generated.

Value

The function return the R2-indicator of the set.

References

createWeights

Examples

nPointToSample <- 100
nObjective <- 3
points <- matrix(runif(nPointToSample*nObjective), nrow = nObjective) # sample the points
ranks <- nsga2R::fastNonDominatedSorting(t(points)) # non-dominated sorting
points <- points[,ranks[[1]],drop=FALSE] # take only the non-dominated front
nPoints <- ncol(points) # check how many points are on the non-dominated front
reference <- rep(2,nObjective)

compute_R2mtch(points,reference)

createWeights

Das and Dennis’s structured weight generation, normal boundary intersection (NBI).

Description

Generate a set of weights following Das and Dennis’s method. Each column returned is a weight vector.

Usage

createWeights(nDim, axisDivision = nDim + 2, noZero = FALSE)

Arguments

nDim The dimensionality of the problem. In EA, usually this is used in the objective space, hence nDim = nObjective
axisDivision Used only when no weights are supplied. An input for the structured weight distribution. This defines how many division are created in each axis.
noZero Default to false. If set to TRUE, reference vector containing zero, e.g. (1,0,0) will be removed. Used to generate weight in modified tch method.

Value

The function return a set of weight vectors.

References


Examples

nObjective <- 3
axisDiv <- 6
createWeights(nObjective,axisDiv)
createWeightsSobol  
*Sobol sequence weights*

**Description**
Generate a set of weights following Sobol sequence generator

**Usage**
```r
createWeightsSobol(nWeights, nDim, seed = 4177)
```

**Arguments**
- `nWeights`: Number of weights to generate.
- `nDim`: The dimensionality of the problem. In EA, usually this is used in the objective space, hence `nDim = nObjective`.
- `seed`: Seed for scrambling

**Value**
The function return a set of weight vectors.

**Examples**
```r
nObjective <- 3
nPoint <- 1000
createWeightsSobol(nPoint, nObjective)
```

**DTLZ1**
The *DTLZ1 test function*.

**Description**
The DTLZ1 test function.

**Usage**
```r
DTLZ1(individual, nObj)
```

**Arguments**
- `individual`: The vector of individual (or matrix of population) to be evaluated.
- `nObj`: The number of objective
DTLZ2

Value

A matrix of size nObjective x population size, containing the objective values for each individual.

References


Examples

```r
individual <- stats::runif(14)
nObj <- 4
DTLZ2(individual,nObj)
```

---

DTLZ2

*The DTLZ2 test function.*

Description

The DTLZ2 test function.

Usage

```r
DTLZ2(individual, nObj)
```

Arguments

- `individual`: The vector of individual (or matrix of population) to be evaluated.
- `nObj`: The number of objective

Value

A matrix of size nObjective x population size, containing the objective values for each individual.

References


Examples

```r
individual <- stats::runif(14)
nObj <- 4
DTLZ2(individual,nObj)
```
DTLZ3

The DTLZ3 test function.

Description

The DTLZ3 test function.

Usage

DTLZ3(individual, nObj)

Arguments

individual The vector of individual (or matrix of population) to be evaluated.
nObj The number of objective

Value

A matrix of size nObjective x population size, containing the objective values for each individual.

References


Examples

individual <- stats::runif(14)
nObj <- 4
DTLZ3(individual, nObj)

DTLZ4

The DTLZ4 test function.

Description

The DTLZ4 test function.

Usage

DTLZ4(individual, nObj, alpha = 100)
### EvaluateIndividual

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>individual</code></td>
<td>The vector of individual (or matrix of population) to be evaluated.</td>
</tr>
<tr>
<td><code>nObj</code></td>
<td>The number of objective</td>
</tr>
<tr>
<td><code>alpha</code></td>
<td>Alpha value of DTLZ4 function.</td>
</tr>
</tbody>
</table>

**Value**

A matrix of size nObjective x population size, containing the objective values for each individual.

**References**


**Examples**

```r
individual <- stats::runif(14)
nObj <- 4
DTLZ4(individual, nObj)
```

---

### EvaluateIndividual

*Evaluate objective values of a single individual*

**Description**

Evaluate individual with the specified test function. Non-feasible solution are given Inf as objective values.

**Usage**

```r
EvaluateIndividual(individual, fun, ...)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>individual</code></td>
<td>The individual to be evaluated</td>
</tr>
<tr>
<td><code>fun</code></td>
<td>A string containing which problem is being solved. Currently available DTLZ1-DTLZ4, WFG4-WFG9.</td>
</tr>
<tr>
<td><code>...</code></td>
<td>Further parameters used by <code>fun</code></td>
</tr>
</tbody>
</table>

**Value**

A matrix of size nObjective, containing the objective values.

**Examples**

```r
individual <- stats::runif(8)
EvaluateIndividual(individual, WFG4, 3) # the 3 is passed to WFG4 nObj
```
**EvaluatePopulation**  
*Evaluate objective value of a set of individuals*

**Description**

Evaluate a population with the specified test function. Non-feasible solutions are given Inf as objective values.

**Usage**

```
EvaluatePopulation(pop, fun, ...)
```

**Arguments**

- `pop`  
The population to be evaluated
- `fun`  
A string containing which problem is being solved. Currently available in the package: DTLZ1-DTLZ4, WFG4-WFG9.
- `...`  
Further parameters used by `fun`

**Value**

A matrix of size nObjective, containing the objective values.

**Examples**

```
pop <- matrix(runif(8*50),nrow=8) # 8 variables, 50 individuals
EvaluatePopulation(pop,WFG4,3) # the 3 is passed to WFG4
```

**GetHVContribution**  
*Get HV contribution of all points.*

**Description**

Get the hypervolume (HV) contribution of the population. Dominated front will give 0 contribution.

**Usage**

```
GetHVContribution(
    populationObjective,
    reference = NULL,
    method = "exact",
    ref_multiplier = 1.1
)
```
**GetHypervolume**

**Description**

Compute the hypervolume formed by the points w.r.t. a reference point. If no reference is supplied, use the nadir point*(1.1,...,1.1).

**Usage**

```r
GetHypervolume(
    objective,
    reference = NULL,
    method = "exact",
    ref_multiplier = 1.1
)
```

**Arguments**

- `populationObjective`
  - The objective value of the corresponding individual
- `reference`
  - The reference point for computing HV
- `method`
  - The HV computation method. Currently ignored and uses the WFG exact method.
- `ref_multiplier`
  - Multiplier to the nadir point for dynamic reference point location

**Value**

A vector of length `ncol(populationObjective)`

**Examples**

```r
nObjective <- 5 # the number of objectives
nPoint <- 10 # the number of points that will form the hypervolume
objective <- matrix(stats::runif(nObjective*nPoint), nrow = nObjective, ncol = nPoint)

# Check if numpy and pygmo modules are available

if(py_module_ready) # prevent error on testing the example
    GetHypervolume(objective, "exact") # no reference supplied

reference <- rep(2,nObjective) # create a reference point at (2,2,2,2)

if(py_module_ready) # prevent error on testing the example
    GetHVContribution(objective, reference)
```
**Arguments**

**objective**  
The set of points in the objective space (The objective values). A single column should contain one point, so the size would be numberOfObjective x nPoint, e.g. in 5 objective problem, it is 5 x n.

**reference**  
The reference point for HV computation. A column vector.

**method**  
Exact using WFG method or approximate HV using the method by Bringmann and Friedrich. Default to "exact".

**ref_multiplier**  
Multiplier to the nadir point for dynamic reference point location

**Value**

Hypervolume size, a scalar value.

**Examples**

```r
nObjective <- 5  # the number of objectives
nPoint <- 10      # the number of points that will form the hypervolume
objective <- matrix(stats::runif(nObjective*nPoint), nrow = nObjective, ncol = nPoint)
numpyready <- reticulate::py_module_available('numpy')
pygmoready <- reticulate::py_module_available('pygmo')
py_module_ready <- numpyready && pygmoready
if(py_module_ready) # prevent error on testing the example
  GetHypervolume(objective, "exact") # no reference supplied
reference <- rep(2,nObjective)  # create a reference point at (2,2,2,2)
if(py_module_ready) # prevent error on testing the example
  GetHypervolume(objective, reference, "exact") # using reference point
```

---

**GetIGD**  
**Get IGD value**

**Description**

Get Inverted Generational Distance (IGD) value of the population objective w.r.t. a matrix of reference set (each row contain 1 point).

**Usage**

```r
GetIGD(populationObjective, referenceSet)
```

**Arguments**

**populationObjective**  
The objective value of the corresponding individual

**referenceSet**  
The reference points for computing IGD
GetLeastContribution

Value
The IGD metric. A Scalar value.

Description
Get the hypervolume (HV) contribution of the individual with least HV contribution.

Usage
GetLeastContribution(
    populationObjective,
    reference = NULL,
    method = "exact",
    ref_multiplier = 1.1
)

Arguments

populationObjective
    The objective value of the corresponding individual
reference
    The reference point for computing HV
method
    the HV computation method
ref_multiplier
    Multiplier to the nadir point for dynamic reference point location

Value
The HV contribution value of the least contributor.

Examples

nObjective <- 5  # the number of objectives
nPoint <- 10  # the number of points that will form the hypervolume
objective <- matrix(stats::runif(nObjective*nPoint), nrow = nObjective, ncol = nPoint)
numpyready <- reticulate::py_module_available("numpy")
pygmoready <- reticulate::py_module_available("pygmo")
py_module_ready <- numpyready & pygmoready
if(py_module_ready)  # prevent error on testing the example
    GetHypervolume(objective,"exact")  # no reference supplied

reference <- rep(2,nObjective)  # create a reference point at (2,2,2,2,2)
if(py_module_ready)  # prevent error on testing the example
    GetLeastContribution(objective,reference,"exact")
GetLeastContributor

Get least HV contributor

Description
Get index of the individual with least hypervolume (HV) contribution. For the contribution itself, use GetLeastContribution()

Usage

GetLeastContributor(
    populationObjective,
    reference = NULL,
    method = "exact",
    hypervolumeMethodParam = list(),
    ref_multiplier = 1.1
)

Arguments

populationObjective
The objective value of the corresponding individual

reference
The reference point for computing HV

method
the HV computation method

hypervolumeMethodParam
A list of parameters to be passed to the hypervolumeMethod

ref_multiplier
Multiplier to the nadir point for dynamic reference point location

Value
The index of the least contributor, an integer.

Examples

nObjective <- 5 # the number of objectives
nPoint <- 10 # the number of points that will form the hypervolume
objective <- matrix(stats::runif(nObjective*nPoint), nrow = nObjective, ncol = nPoint)
# run a generation of MO-CMA-ES with standard WFG8 test function.
numpyready <- reticulate::py_module_available('numpy')
pygmoready <- reticulate::py_module_available('pygmo')
py_module_ready <- numpyready && pygmoready
if(py_module_ready) # prevent error on testing the example
    GetHypervolume(objective,"exact") # no reference supplied

reference <- rep(2,nObjective) # create a reference point at (2,2,2,2)
if(py_module_ready) # prevent error on testing the example
    GetLeastContributor(objective,reference,"exact")
InitializePopulationLHS

Initialize population with Latin Hypercube Sampling

Description

Create initial sample using Latin Hypercube Sampling (LHS) method. The variables will be ranged between 0-1

Usage

InitializePopulationLHS(
    numberOfIndividuals,
    chromosomeLength,
    minVal = 0,
    maxVal = 1,
    samplingMethod = 0
)

Arguments

numberOfIndividuals
    The number of individual in the population (ncol). Integer > 0.

chromosomeLength
    The number of variables per individual (nrow)

minVal
    Minimum value of the resulting sample

maxVal
    Maximum value of the resulting sample

samplingMethod
    Not used

Value

A matrix of size chromosomeLength x nIndividual.

Examples

nVar <- 14
nIndividual <- 100
InitializePopulationLHS(nIndividual,nVar,FALSE)
install_python_dependencies

Install python modules required by MaOEA: numpy and PyGMO

Description
Install the required python package via conda.

Usage
install_python_dependencies(conda = "auto", envname = NULL, ...)

Arguments
conda Default: auto
envname Python virtual environment where the modules will be installed, default to 'r-reticulate'
... Further argument to pass to reticulate::py_install

Value
0 if dependencies installed and loaded successfully, 1 if fails.

load_python_dependencies

Install python modules required by MaOEA: numpy and PyGMO

Description
Import the required python package if it fails onLoad.

Usage
load_python_dependencies()

Value
0 if dependencies loaded successfully, 1 if fails.
**MOCMAES**

*Multi-Objective CMA-ES*

**Description**

Do an iteration of population based Multi-Objective Covariance Matrix Adaptation Evolution Strategy (MO-CMA-ES). The variation is using simulated binary crossover (SBX) and mutation following the CMA. The original MO-CMA-ES does not use crossover, to do this simply set crossover-Probability to zero.

**Usage**

```r
MOCMAES(parent, nObjective, fun, control = list(), ...)
```

**Arguments**

- `parent`: The parent generation, an object of class `cmaes_gen`. The MO-CMA-ES parent is a 5 tuple: `x` (the design point, length = number of variable), `averageSuccessRate` (scalar), `stepSize` (scalar), `evoPath` (evolution path, vector, length = number of variable), `covarianceMatrix` (square matrix with `ncol` = `nrow` = number of variable). The parent is then should be a vector of lists (see example).

- `nObjective`: The number of objective functions. A scalar value.

- `fun`: Objective function being solved.

- `control`: List of parameters for CMA-ES. Available control are as follows: `successProbTarget` Target success probability, `successProbThreshold` The threshold for success probability. If the average success probability is higher than this value, the success rate growth is slowed. `crossoverProbability` The probability of doing crossover. Should be between 0-1. Negative value will behave like a zero, and values larger than 1 will behave like 1. Default to 1. `crossoverDistribution` The distribution index for SBX. Larger index makes the distribution sharper around each parent.

- `...`: Further arguments to be passed to `fun`

**Value**

Returns a list for the next generation. It contains `list$new_generation` (class: `cmaes_gen`), `list$population` (basically a copy of `list$new_generation[[][]$x`), and `list$populationObjective`.

**References**

Examples

nVar <- 14
nObjective <- 5
nIndividual <- 100

crossoverProbability <- 1

ps_target <- 1 / ( 5 + ( 1 / 2 )^0.5 )

pop <- matrix(stats::runif(nIndividual*nVar), nrow = nVar) # create the population

a_list <- cmaes_gen(pop)

control <- list(successProbTarget=ps_target, crossoverProbability=crossoverProbability)

# run a generation of MO-CMA-ES with standard WFG8 test function.

numpyready <- reticulate::py_module_available(\'Var\numpy\Var\')

pygmoready <- reticulate::py_module_available(\'Var\pygmo\Var\')

py_module_ready <- numpyready && pygmoready

if(py_module_ready) # prevent error on testing the example

newGeneration <- MOCMAES(a_list,nObjective,WFG8,control,nObjective)


---

**Normalize**

**Objective space normalization.**

**Description**

Normalize the objectives AND reference (combined) to 0-1. The origin is the ideal point. (1,...,1) is the nadir.

**Usage**

`Normalize(objectiveValue, referencePoints = NULL)`

**Arguments**

- **objectiveValue**: Set of objective vectors to normalize
- **referencePoints**: Set of reference points to transform following the objective vector normalization

**Value**

A list containing the following: normalizedObjective The normalized values idealPoint The ideal point corresponding to the origin transformedReference The location of reference points in the normalized Space
NSGA3

Examples

```r
nObj <- 5
nVar <- 10
nIndividual <- 100
population <- InitializePopulationLHS(nIndividual,nVar,FALSE)
objective <- matrix(,nrow=nObj,ncol=nIndividual)
for(individual in 1:nIndividual){
  objective[,individual] <- WFG4(population[,individual],nObj)
}
Normalize(objective)
```

---

**NSGA3**  
**Elitist Non-dominated Sorting Genetic Algorithm version III**

Description

Do an iteration of Elitist Non-dominated Sorting Genetic Algorithm version III (NSGA-III). The variation is using SBX and polynomial mutation.

Usage

```r
NSGA3(population, fun, nObjective, control = list(), ...)
```

Arguments

- `population`  
  The parent generation. One individual per column. nrow = number of variable, ncol = number of individuals in the population.

- `fun`  
  Objective function being solved. Currently available in the package DTLZ1-DTLZ4, WFG4-WFG9.

- `nObjective`  
  The number of objective functions. A scalar value. Needed to generate weight vectors.

- `control`  
  A list, containing the following:  
  - `weightVector`  
    NSGA-III require a set of reference points defined a priori. The reference can be any point. If not supplied, 5*nObjective points are generated from a sobol sequence. Column major: nrow = nObjective, ncol = number of reference points.
  - `crossoverProbability`  
    The probability of doing crossover. Should be between 0-1. Negative value will behave like a zero, and values larger than 1 will behave like 1. Default to 1.
  - `mutationProbability`  
    The probability of doing mutation. Should be between 0-1. Negative value will behave like a zero, and values larger than 1 will behave like 1. Default to 1.
  - `mutationDistribution`  
    The distribution index for polynomial mutation. Larger index makes the distribution sharper around the parent.
  - `crossoverDistribution`  
    The distribution index for SBX. Larger index makes the distribution sharper around each parent.

...  
  Further arguments to be passed to `fun`
Value

#' @return Returns a list for the next generation population The new generation design points. Column major. populationObjective The new generation’s objective values. Column major.

References


Examples

nVar <- 14
nObjective <- 5
nIndividual <- 100
#control for NSGA3
ctrl <- list(crossoverProbability = 1,
mutationProbability = 1/nVar)
#Initial population
population <- matrix(runif(nIndividual*nVar), nrow = nVar)

# run a generation of NSGA-III with standard WFG8 test function.
NSGA3(population, WFG8,nObjective,ctrl,nObjective)

optimMaOEA

Elitist Non-dominated Sorting Genetic Algorithm version III

Description

Main interface for the many-objective optimization evolutionary algorithm (MaOEA) package.

Usage

optimMaOEA(
  x = NULL,
  fun,
  solver = NSGA3,
  nObjective,
  nGeneration = 1,
  nVar = nrow(x),
  populationSize = ncol(x),
  seed = 2000,
  control = list(),
  ...
)
Arguments

x The initial population. If not supplied, will be generated using LHS. Column major, each column contain one entry.

fun Objective function being solved.

solver Function name of the solver. Currently available: SMSEMOA, MOCMAES, SMOCMAES, and NSGA3.

nObjective The number of objective functions. A scalar value.

nGeneration Optional, the number of generation the solver should run.

nVar Number of variables, will be used if x is not given.

populationSize Number of individuals in the population, will be used if x is not given.

seed random number seed for reproduction of code

control A list, containing the following: weightVectorSet A set of weight vector for the optimizer. The weight vector can be any point in the objective space. If not supplied, 5*nObjective points are generated from a sobol sequence. Size: nrow = nObjective, ncol = number of weight vectors crossoverProbability The probability of doing crossover. Should be between 0-1. Negative value will behave like a zero, and values larger than 1 will behave like 1. Default to 1. mutationProbability The probability of doing mutation. Should be between 0-1. Negative value will behave like a zero, and values larger than 1 will behave like 1. Default to 1. WFGScaling The use of scaling factor in WFG. Will be ignored in DTLZ problems. Without the scaling, the Pareto front would be on the all-positive portion of hypersphere with radius 1. mutationDistribution The distribution index for polynomial mutation. Larger index makes the distribution sharper around the parent. crossoverDistribution The distribution index for SBX. Larger index makes the distribution sharper around each parent.

... Further arguments to be passed to fun

Value

Returns a list for the next generation population The new generation design points. populationObjective The new generation’s objective values.

Examples

nVar <- 14
nObjective <- 5
nIndividual <- 100
# control for NSGA3
ctrl <- list(crossoverProbability = 1,
mutationProbability = 1/nVar)
# Initial population can be supplied, like below but for this example, we skip it
# population <- matrix(runif(nIndividual*nVar), nrow = nVar)

numpyready <- reticulate::py_module_available('numpy')
pygmoready <- reticulate::py_module_available('pygmo')
py_module_ready <- numpyready && pygmoready
if(py_module_ready){ # prevent error on testing the example
# Hybrid NSGA-III and SMSEMOA example
# 2 calls for nObject. 1 for optimMaOEA, 1 for WFG8
# generate initial population and run 10 gen. NSGA-III with standard WFG8 test function.
newPop <- optimMaOEA( , WFG8,NSGA3,nObjective,10,nVar,nIndividual,,ctrl,nObjective)$x

# run 5 generations of SMSEMOA with standard WFG8 test function starting with newPop.
result <- optimMaOEA( newPop, WFG8,SMSEMOA,nObjective,5,,,1000,ctrl,nObjective)
finalPop <- result$x
finalObjective <- result$y
}

SMOCMAES

Steady-state Multi-Objective CMA-ES

Description

Do an iteration of population based steady state Multi-Objective Covariance Matrix Adaptation Evolution Strategy (MO-CMA-ES). The variation is using simulated binary crossover (SBX) and mutation following the CMA. The original MO-CMA-ES does not use crossover, to do this simply set crossoverProbability to zero.

Usage

SMOCMAES(parent, nObjective, fun, control = list(), ...)

Arguments

parent The parent generation, an object of class cmaes_gen. The MO-CMA-ES parent is a 5 tuple: x (the design point, length = number of variable),averageSuccessRate (scalar),stepSize (scalar), evoPath (evolution path, vector, length = number of variable ),covarianceMatrix (square matrix with ncol = nrow = number of variable). The parent is then should be a vector of lists (see example).
nObjective The number of objective functions. A scalar value.
fun Objective function being solved.
control List of parameters for CMA-ES. Available control are as follows: successProbTarget Target success probability successProbThreshold The threshold for success probability. If the average success probability is higher than this value, the success rate growth is slowed. crossoverProbability The probability of doing crossover. Should be between 0-1. Negative value will behave like a zero, and values larger than 1 will behave like 1. Default to 1. crossoverDistribution The distribution index for SBX. Larger index makes the distribution sharper around each parent.
...

Further arguments to be passed to fun
Value

Returns a list for the next generation. It contains list$new_generation (class: cmaes_gen), list$population (basically a copy of list$new_generation[[[]]]$x), and list$populationObjective

References


Examples

```r
nVar <- 14
nObjective <- 5
nIndividual <- 100
crossoverProbability <- 1
ps_target <- 1 / (5 + (1 / 2))
pop <- matrix(stats::runif(nIndividual*nVar), nrow = nVar) # create the population
a_list <- cmaes_gen(pop)
control <- list(successProbTarget=ps_target,crossoverProbability=crossoverProbability)
# run a generation of SMO-CMA-ES with standard WFG8 test function.
newGeneration <- SMOCMAES(a_list,nObjective,WFG8,control,nObjective)
```

S-Metric Selection EMOA

Description

Do an iteration of S-Metric Selection (SMS)-EMOA. The variation used is simulated binary crossover (SBX) and polynomial mutation.

Usage

```r
SMSEMOA(population, fun, nObjective, control = list(), ...)
```

Arguments

- `population`: The parent generation. One individual per column.
- `fun`: Objective function being solved. Currently available in the package DTLZ1-DTLZ4, WFG4-WFG9.
- `nObjective`: Number of objective. Ignored as of version 0.6.1; number of row from fun is used instead.
control (list) Options to control the SMS-EMOA: 
mutationProbability The probability of doing mutation. Should be between 0-1. Negative value will behave like a zero, and values larger than 1 will behave like 1. Default to 1 
mutationDistribution The distribution index for polynomial mutation. Larger index makes the distribution sharper around the parent. 
crossoverDistribution The distribution index for SBX. Larger index makes the distribution sharper around each parent. 
referencePoint The reference point for HV computation on normalized objective space, i.e. (1,...,1) is the nadir point. If not supplied, the ref_multiplier is used instead. 
ref_multiplier In case that a reference point is not supplied, the reference is set as a multiply of the current nadir. Default to 1.1. 
lbound A vector containing the lower bound for each gene 
ubound A vector containing the upper bound for each gene 
scaleinput Whether the input should be scaled to 0-1. 

Further arguments to be passed to fun

Value

Returns a list for the next generation population The new generation. Column major, each row contain 1 set of objectives. successfulOffspring Binary, 1 if the offspring is kept in the new generation. Used in some adaptive schemes. populationObjective The new generation's objective values.

References


Examples

nVar <- 14
nObjective <- 5
nIndividual <- 100
crossoverProbability <- 1
mutationProbability <- 1/nVar
population <- matrix(runif(nIndividual*nVar), nrow = nVar)

# run a generation of SMS-EMOA with standard WFG6 test function.
numpyready <- reticulate::py_module_available('numpy')
pygmoready <- reticulate::py_module_available('pygmo')
py_module_ready <- numpyready && pygmoready
if(py_module_ready) # prevent error on testing the example
    SMSEMOA(population,WFG6,nObjective,list(crossoverProbability = crossoverProbability,
    mutationProbability = mutationProbability),nObjective)
WFG1

The WFG1 test function.

Description
The WFG1 test function.

Usage
WFG1(individual, nObj, k = nObj - 1)

Arguments
individual  The individual to be evaluated, the search space should be in [0-2^i] for variable number i. Can accept multiple individuals each in different column.
nObj        The number of objective
k           Number of distance related parameters. The reference suggests a positive integer multiplied by (nObj-1). Default to nObj-1

Value
A matrix of size nObjective, containing the objective values.

References

Examples
individual <- runif(14)
nObj <- 4
WFG1(individual,nObj)

WFG2

The WFG2 test function.

Description
The WFG2 test function.

Usage
WFG2(individual, nObj, k = nObj - 1)
Arguments

individual The individual to be evaluated, the search space should be in [0-2i] for variable number i. Can accept multiple individuals in each column.
nObj The number of objective
k Number of distance related parameters. The reference suggests a positive integer multiplied by (nObj-1). Default to nObj-1

Value

A matrix of size nObjective, containing the objective values.

References


Examples

individual <- runif(14)
nObj <- 4
WFG2(individual, nObj)

WFG4

The WFG4 test function.

Description

The WFG4 test function.

Usage

WFG4(individual, nObj, k = nObj - 1)

Arguments

individual The individual to be evaluated, the search space should be in [0-2i] for variable number i. Can accept multiple individuals in each column.
nObj The number of objective
k Number of distance related parameters. The reference suggests a positive integer multiplied by (nObj-1). Default to nObj-1

Value

A matrix of size nObjective, containing the objective values.
References

Examples
```
individual <- runif(14)
nObj <- 4
WFG4(individual,nObj)
```

---

**WFG5**

*The WFG5 test function.*

---

**Description**
The WFG5 test function.

**Usage**

```
WFG5(individual, nObj, k = nObj - 1)
```

**Arguments**

- **individual**
  The individual to be evaluated, the search space should be in [0-2^i] for variable number i. Can accept multiple individualm each in different column.

- **nObj**
  The number of objective

- **k**
  Number of distance related parameters. The reference suggests a positive integer multiplied by (nObj-1). Default to nObj-1

**Value**

A matrix of size nObjective, containing the objective values.

**References**

**Examples**
```
individual <- runif(14)
nObj <- 4
WFG5(individual,nObj)
```
WFG6

*The WFG6 test function.*

### Description

The WFG6 test function.

### Usage

\[
\text{WFG6}(\text{individual}, \text{nObj}, k = \text{nObj} - 1)
\]

### Arguments

- **individual**: The individual to be evaluated, the search space should be in \([0-2^i]\) for variable number \(i\). Can accept multiple individuals each in different column.
- **nObj**: The number of objective
- **k**: Number of distance related parameters. The reference suggests a positive integer multiplied by \((\text{nObj}-1)\). Default to \(\text{nObj}-1\)

### Value

A matrix of size \(\text{nObjective}\), containing the objective values.

### References


### Examples

\[
\begin{align*}
\text{individual} & \leftarrow \text{runif}(14) \\
\text{nObj} & \leftarrow 4 \\
\text{WFG6}(\text{individual}, \text{nObj})
\end{align*}
\]

WFG7

*The WFG7 test function.*

### Description

The WFG7 test function.

### Usage

\[
\text{WFG7}(\text{individual}, \text{nObj}, k = \text{nObj} - 1)
\]
The WFG8 test function.

**Arguments**

- `individual` : The individual to be evaluated, the search space should be in \([0-2^i]\) for variable number i. Can accept multiple individuals each in different column.
- `nObj` : The number of objective
- `k` : Number of distance related parameters. The reference suggests a positive integer multiplied by \((nObj-1)\). Default to \(nObj-1\)

**Value**

A matrix of size \(nObj\), containing the objective values.

**References**


**Examples**

```r
individual <- runif(14)
nObj <- 4
WFG7(individual, nObj)
```

---

**Description**

The WFG8 test function.

**Usage**

`WFG8(individual, nObj, k = nObj - 1)`
References


Examples

```r
individual <- runif(14)
nObj <- 4
WFG8(individual,nObj)
```

---

**WFG9**  
*The WFG9 test function.*

Description

The WFG9 test function.

Usage

```r
WFG9(individual, nObj, k = nObj - 1)
```

Arguments

- `individual`: The individual to be evaluated, the search space should be in \([0-2^i]\) for variable number \(i\). Can accept multiple individualm each in different column.
- `nObj`: The number of objective
- `k`: Number of distance related parameters. The reference suggests a positive integer multiplied by \(nObj-1\). Default to \(nObj-1\)

Value

A matrix of size \(nObjective\), containing the objective values.

References


Examples

```r
individual <- runif(14)
nObj <- 4
WFG9(individual,nObj)
```
# Index

* **package**
  - MaOEA-package, 3
  - AdaptiveNormalization, 4
  - cmaes_gen, 4
  - compute_R2HV, 6
  - compute_R2HVC, 7
  - compute_R2mtch, 8
  - createWeights, 9
  - createWeightsSobol, 10
  - DTLZ1, 10
  - DTLZ2, 11
  - DTLZ3, 12
  - DTLZ4, 12
  - EvaluateIndividual, 13
  - EvaluatePopulation, 14
  - GetHVContribution, 14
  - GetHypervolume, 15
  - GetIGD, 16
  - GetLeastContribution, 17
  - GetLeastContributor, 18
  - InitializePopulationLHS, 19
  - install_python_dependencies, 20
  - load_python_dependencies, 20
  - MaOEA (MaOEA-package), 3
  - MaOEA-package, 3
  - MOCMAES, 21
  - Normalize, 22
  - NSGA3, 23
  - optimMaOEA, 3, 24
  - SMOCMAES, 26
  - SMSEMOA, 27
  - WFG1, 29
  - WFG2, 29
  - WFG4, 30
  - WFG5, 31
  - WFG6, 32
  - WFG7, 32
  - WFG8, 33
  - WFG9, 34