Modular Evolutionary Algorithm Framework Documentation

Release 0.3.0

Sander van Rijn

Feb 04, 2023

Contents

CHAPTER 1

A note from the author

This documentation serves as a reference for the structure of this framework.

Please note, this framework is still under development, and no guarantees will be given on it's correctness of execution. Any modules, classes or functions currently specified may be renamed, removed, reinvented, rescued, rolled over, and whatever else turns out to be most useful for the future of this framework as a whole.

Last updated: (Feb 04, 2023)

CHAPTER 2

Contents

2.1 modea

2.1.1 modea package

Submodules

modea.Algorithms module

Collection of some standard algorithms and the fully customizable CMA-ES as used in 'Evolving the Structure of Evolution Strategies', all based on the same `baseAlgorithm()` function.

class modea.Algorithms.**CMAESOptimizer**(*n*, *fitnessFunction*, *budget*, *mu=None*, *lambda_=None*, *elitist=False*)

Bases: *modea.Algorithms.EvolutionaryOptimizer*

Implementation of a default (mu +/, lambda)-CMA-ES Requires the length of the vector to be optimized, a fitness function to use, and the budget

Parameters

- **n** – Dimensionality of the problem to be solved
- **fitnessFunction** – Function to determine the fitness of an individual
- **budget** – Number of function evaluations allowed for this algorithm
- **mu** – Number of individuals that form the parents of each generation
- **lambda** – Number of individuals in the offspring of each generation
- **elitist** – Boolean switch on using a (mu, l) strategy rather than (mu + l). Default: False

class modea.Algorithms.**CustomizedES**(*n*, *fitnessFunction*, *budget*, *mu=None*, *lambda_=None*, *opts=None*, *values=None*)

Bases: *modea.Algorithms.EvolutionaryOptimizer*

This function accepts a dictionary of options 'opts' which selects from a large range of different functions and combinations of those. Instrumental in Evolving Evolution Strategies

Parameters

- **n** – Dimensionality of the problem to be solved
- **fitnessFunction** – Function to determine the fitness of an individual
- **budget** – Number of function evaluations allowed for this algorithm
- **mu** – Number of individuals that form the parents of each generation
- **lambda** – Number of individuals in the offspring of each generation
- **opts** – Dictionary containing the options (elitist, active, threshold, etc) to be used
- **values** – Dictionary containing initial values for initializing (some of) the parameters

addDefaults(*opts*)

bool_default_opts = ['active', 'elitist', 'mirrored', 'orthogonal', 'sequential', 'thr

calculateDependencies(*opts*, *lambda_*, *mu*)

string_default_opts = ['base-sampler', 'ipop', 'selection', 'weights_option']

class modea.Algorithms.**EvolutionaryOptimizer**(*population*, *fitnessFunction*, *budget*, *functions*, *parameters*, *parallel=False*)

Bases: object

Skeleton function for all ES algorithms Requires a population, fitness function handle, evaluation budget and the algorithm-specific functions

The algorithm-specific functions should (roughly) behave as follows: * recombine The current population (mu individuals) is passed to this function, and should return a new population (lambda individuals), generated by some form of recombination

- mutate An individual is passed to this function and should be mutated 'in-line', no return is expected
- select The original parents, new offspring and used budget are passed to this function, and should return a new population (mu individuals) after (mu+lambda) or (mu,lambda) selection
- mutateParameters Mutates and/or updates all parameters where required

Parameters

- **population** – Initial set of individuals that form the starting population of the algorithm
- **fitnessFunction** – Function to determine the fitness of an individual
- **budget** – Number of function evaluations allowed for this algorithm
- **functions** – Dictionary with functions 'recombine', 'mutate', 'select' and 'mutateParameters'
- **parameters** – Parameters object for storing relevant settings
- **parallel** – Set to True to enable parallel evaluation. Note: this disables sequential evaluation

Returns The statistics generated by running the algorithm

determineRegime()

evalPopulation()

evalPopulationSequentially()

initializePopulation()

instantiateParameters(*params*)

recordStatistics()

runLocalRestartOptimizer(*target=None*, *threshold=None*)
Run the baseAlgorithm with the given specifications using a local-restart strategy.

runOneGeneration()

runOptimizer(*target=None*, *threshold=1e-08*)

tpaUpdate()

class modea.Algorithms.**GAOptimizer**(*n*, *fitnessFunction*, *budget*, *mu*, *lambda_*, *population*, *parameters=None*)
Bases: *modea.Algorithms.EvolutionaryOptimizer*

Defines a Genetic Algorithm (GA) that evolves an Evolution Strategy (ES) for a given fitness function

Parameters

- **n** – Dimensionality of the search-space for the GA
- **fitnessFunction** – Fitness function the GA should use to evaluate candidate solutions
- **budget** – The budget for the GA
- **mu** – Population size of the GA
- **lambda** – Offpsring size of the GA
- **population** – Initial population of candidates to be used by the MIES
- **parameters** – Parameters object to be used by the GA

class modea.Algorithms.**MIESOptimizer**(*n*, *mu*, *lambda_*, *population*, *fitnessFunction*, *budget*, *parameters=None*)
Bases: *modea.Algorithms.EvolutionaryOptimizer*

Defines a Mixed-Integer Evolution Strategy (MIES) that evolves an Evolution Strategy (ES) for a given fitness function

Parameters

- **n** – Dimensionality of the search-space for the MIES
- **fitnessFunction** – Fitness function the MIES should use to evaluate candidate solutions
- **budget** – The budget for the MIES
- **mu** – Population size of the MIES
- **lambda** – Offpsring size of the MIES
- **population** – Initial population of candidates to be used by the MIES
- **parameters** – Parameters object to be used by the MIES

class modea.Algorithms.**OnePlusOneOptimizer**(*n*, *fitnessFunction*, *budget*)
Bases: *modea.Algorithms.EvolutionaryOptimizer*

Implementation of the default (1+1)-ES Requires the length of the vector to be optimized, the handle of a fitness function to use and the budget

Parameters

- **n** – Dimensionality of the problem to be solved
- **fitnessFunction** – Function to determine the fitness of an individual
- **budget** – Number of function evaluations allowed for this algorithm

modea.Individual module

This module contains definitions of Individual classes, that allow for a common interface with different underlying genotypes (float, int, mixed-integer).

class modea.Individual.**FloatIndividual**(*n*)
Bases: object

Data holder class for individuals using a vector of floating point values as genotype. This type of individual can therefore be used by an Evolution Strategy (ES) such as the CMA-ES. Stores the genotype column vector and all individual-specific parameters. Default genotype is np.ones((n,1))

Parameters **n** – dimensionality of the problem to be solved

class modea.Individual.**MixedIntIndividual**(*n*, *num_discrete*, *num_ints*, *num_floats=None*)
Bases: object

Data holder class for individuals using a vector containing both floating point and integer values as genotype. This type of individual can therefore be used by a GA with mixed-integer mutation operators. Stores the genotype column vector and all individual-specific parameters. Default genotype is np.ones((n,1))

Parameters

- **n** – Dimensionality of the problem to be solved, consisting of discrete, integers and floating point values
- **num_discrete** – Number of discrete values in the genotype.
- **num_ints** – Number of integer values in the genotype.
- **num_floats** – Number of floating point values in the genotype.

stepsizeMIES

exception modea.Individual.**MixedIntIndividualError**
Bases: Exception

modea.Mutation module

This Module contains a collection of Mutation operators to be used in the ES-Framework

A Mutation operator mutates an Individual's genotype inline, thus returning nothing.

modea.Mutation.**CMAMutation**(*individual*, *param*, *sampler*, *threshold_convergence=False*)
CMA mutation: x = x + (sigma * B*D*N(0,I))

Parameters

- **individual** – *FloatIndividual* to be mutated
- **param** – *Parameters* object to store settings
- **sampler** – *Sampling* module from which the random values should be drawn
- **threshold_convergence** – Boolean: Should threshold convergence be applied. Default: False

modea.Mutation.**MIES_Mutate**(*individual*, *param*, *options*, *num_options*)
Self-adaptive mixed-integer mutation of the structure of an ES

Parameters

- **individual** – MixedIntegerIndividual whose integer-part will be mutated
- **param** – *Parameters* object
- **options** – List of tuples options with the number of tunable parameters per module
- **num_options** – List num_options with the number of available modules per module position that are available to choose from

modea.Mutation.**MIES_MutateDiscrete**(*individual*, *begin*, *end*, *u*, *num_options*, *options*)
Mutate the discrete part of a Mixed-Integer representation

Parameters

- **individual** – The individual to mutate
- **begin** – Start index of the discrete part of the individual's representation
- **end** – End index of the discrete part of the individual's representation
- **u** – A pre-determined random value from a Gaussian distribution
- **num_options** – List num_options with the number of available modules per module position that are available to choose from
- **options** – List of tuples options with the number of tunable parameters per module

Returns A boolean mask array to be used for further conditional mutations based on which modules are active

modea.Mutation.**MIES_MutateFloats**(*conditional_mask*, *individual*, *begin*, *end*, *u*, *param*)
Mutate the floating point part of a Mixed-Integer representation

Parameters

- **conditional_mask** – Conditional mask that determines which floating point values are allowed to mutate
- **individual** – The individual to mutate
- **begin** – Start index of the integer part of the individual's representation
- **end** – End index of the integer part of the individual's representation
- **u** – A pre-determined random value from a Gaussian distribution
- **param** – *Parameters* object

modea.Mutation.**MIES_MutateIntegers**(*individual*, *begin*, *end*, *u*, *param*)
Mutate the integer part of a Mixed-Integer representation

Parameters

- **individual** – The individual to mutate
- **begin** – Start index of the integer part of the individual's representation
- **end** – End index of the integer part of the individual's representation
- **u** – A pre-determined random value from a Gaussian distribution
- **param** – *Parameters* object

modea.Mutation.**adaptStepSize**(*individual*)
Given the current individual, randomly determine a new step size offset that can be no greater than maxStepSize - baseStepSize

Parameters **individual** – The *FloatIndividual* object whose step size should be adapted

modea.Mutation.**addRandomOffset**(*individual*, *param*, *sampler*)
Mutation 1: x = x + sigma*N(0,I)

Parameters

- **individual** – *FloatIndividual* to be mutated
- **param** – *Parameters* object to store settings
- **sampler** – *Sampling* module from which the random values should be drawn

modea.Mutation.**mutateBitstring**(*individual*)
Simple 1/n bit-flip mutation

Parameters **individual** – *Individual* with a bit-string as genotype to undergo p=1/n mutation

modea.Mutation.**mutateFloatList**(*individual*, *param*, *options*)
Self-adaptive, uniformly random floating point mutation on the tunable parameters of an ES

Parameters

- **individual** – MixedIntegerIndividual whose integer-part will be mutated
- **param** – *Parameters* object
- **options** – List of tuples options with the number of tunable parameters per module

modea.Mutation.**mutateIntList**(*individual*, *param*, *num_options_per_module*)
Self-adaptive random integer mutation to mutate the structure of an ES

Parameters

- **individual** – MixedIntegerIndividual whose integer-part will be mutated
- **param** – *Parameters* object
- **num_options_per_module** – List num_options with the number of available modules per module position that are available to choose from

modea.Mutation.**mutateMixedInteger**(*individual*, *param*, *options*, *num_options_per_module*)
Self-adaptive mixed-integer mutation of the structure of an ES

Parameters

- **individual** – MixedIntegerIndividual whose integer-part will be mutated
- **param** – *Parameters* object
- **options** – List of tuples options with the number of tunable parameters per module
- **num_options_per_module** – List num_options with the number of available modules per module position that are available to choose from

modea.Parameters module

This Parameters module is a container for all possible parameters and all ways in which they are adapted by various optimization methods.

class modea.Parameters.**BaseParameters**

Bases: object

Data holder class for all hardcoded values that are independent of problem dimensionality

alpha_mu = 2

c = 0.817

c_p = 0.08333333333333333

conditioncov = 100000000000000.0

p_target = 0.18181818181818182

p_thresh = 0.44

tolfun = 1e-12

tolupsigma = 1e+20

class modea.Parameters.**Parameters**(*n*, *budget*, *sigma=None*, *mu=None*, *lambda_=None*, *weights_option=None*, *l_bound=None*, *u_bound=None*, *seq_cutoff=1*, *wcm=None*, *active=False*, *elitist=False*, *local_restart=None*, *sequential=False*, *tpa=False*, *values=None*)

Bases: *modea.Parameters.BaseParameters*

Data holder class that initializes *all* possible parameters, regardless of what functions/algorithm are used If multiple functions/algorithms use the same parameter name, but differently, these will be split into separate parameters.

Parameters

- **n** – Dimensionality of the problem to be solved
- **budget** – Number of fitness evaluations the algorithm may perform
- **mu** – Number of individuals that form the parents of each generation
- **lambda** – Number of individuals in the offspring of each generation
- **weights_option** – String to determine which weignts to use. Choose between `default` (CMA-ES) and `1/n`
- **l_bound** – Lower bound of the search space
- **u_bound** – Upper bound of the search space
- **seq_cutoff** – Minimal cut-off allowed in sequential selection
- **wcm** – Initial weighted center of mass
- **active** – Boolean switch on using an active update. Default: False
- **elitist** – Boolean switch on using a (mu, l) strategy rather than (mu + l). Default: False
- **sequential** – Boolean switch on using sequential evaluation. Default: False
- **tpa** – Boolean switch on using two-point step-size adaptation. Default: False
- **values** – Dictionary in the form of `{'name': value}` of initial values for allowed parameters. Any values for names not in `modea.Utils.initializable_parameters` are ignored.

adaptCovarianceMatrix(*evalcount*)

Adapt the covariance matrix according to the (Active-)CMA-ES.

Parameters **evalcount** – Number of evaluations used by the algorithm so far

addToFitnessHistory(*fitness*)
Record the latest `fitness` value (with a history of 5 generations)

Parameters **fitness** – Fitness value to be recorded

addToSuccessHistory(*t*, *success*)
Record the (boolean) `success` value at time `t`

Parameters

- **t** – Number of evaluations used by the algorithm so far
- **success** – Boolean that records whether the last update was a success

checkDegenerated()
Check if the parameters (C, s_mean, etc) have degenerated and need to be reset. Designed for use by a CMA ES

checkLocalRestartConditions(*evalcount*)
Check for local restart conditions according to (B)IPOP

Parameters **evalcount** – Counter for the current generation

Returns Boolean value `restart_required`, True if a restart should be performed

getParameterOpts()

getWeights(*weights_option=None*)

Defines a list of weights to be used in weighted recombination. Available options are:

- `1/n`: Each weight is set to 1/n
- `1/2^n`: Each weight is set to 1/2^i + (1/2^n)/mu
- `default`: Each weight is set to log((lambda-1)/2) - log(i)

param weights_option String to indicate which weights should be used.

returns Returns a np.array of weights, adding to 1

mu_int
Integer value of mu

oneFifthRule(*t*)
Adapts sigma based on the 1/5-th success rule

Parameters **t** – Number of evaluations used by the algorithm so far

recordRecentFitnessValues(*evalcount*, *fitnesses*)
Record recent fitness values at current budget

restart()
Very basic restart, done by resetting some of the variables for CMA-ES

updateThreshold(*t*)
Update the threshold that is used to maintain a minimum stepsize. Taken from: Evolution Strategies with Thresheld Convergence (CEC 2015)

Parameters **t** – Ammount of the evaluation budget spent

modea.Recombination module

This Module contains a collection of Recombination operators

A Recombination operator accepts (mu) individuals and returns (lambda) created individuals that are to form the new population

`modea.Recombination.`**`MIES_recombine`**(*pop*, *param*)
Returns a new set of individuals whose genotype is determined according to the Mixed-Integer ES by Rui Li.

Parameters

- **pop** – The population to be recombined
- **param** – *Parameters* object

Returns A list of lambda individuals, with as genotype the weighted average of the given population.

`modea.Recombination.`**`onePlusOne`**(*pop*, *param*)
Utility function for 1+1 ES strategies where the recombination is merely a copy

Parameters

- **pop** – The population to be recombined
- **param** – *Parameters* object

Returns A copy of the first individual in the given population

`modea.Recombination.`**`onePointCrossover`**(*ind_a*, *ind_b*)
Perform one-point crossover between two individuals.

Parameters

- **ind_a** – An individual
- **ind_b** – Another individual

Returns The original individuals, whose genotype has been modified inline

`modea.Recombination.`**`random`**(*pop*, *param*)
Create a new population by selecting random parents from the given population. To be used when no actual recombination occurs

Parameters

- **pop** – The population to be recombined
- **param** – *Parameters* object

Returns A list of lambda individuals, each a copy of a randomly chosen individual from the population

`modea.Recombination.`**`weighted`**(*pop*, *param*)
Returns a new set of individuals whose genotype is initialized to the weighted average of that of the given population, using the weights stored in the *Parameters* object. Set weights to 1/n to simply use the arithmetic mean

Parameters

- **pop** – The population to be recombined
- **param** – *Parameters* object, of which `param.weights` will be used to calculate the weighted average

Returns A list of lambda individuals, with as genotype the weighted average of the given population.

modea.Sampling module

This module contains several sampling options that can be used when drawing random values for mutations.

Some of the sampling options in this module can be considered *base-samplers*. This means that they produce a set of values without requiring any input. The remaining options will have a `base_sampler` optional argument, as they need input from some other sampler to produce values, as they perform operations on them such as mirroring.

Base samplers

- *GaussianSampling*
- *QuasiGaussianHaltonSampling*
- *QuasiGaussianSobolSampling*

Indirect samplers

- *MirroredSampling*
- *OrthogonalSampling*
- *MirroredOrthogonalSampling*

class modea.Sampling.**GaussianSampling**(*n*, *shape='col'*)
Bases: object

A sampler to create random vectors using a Gaussian distribution

Parameters

- **n** – Dimensionality of the vectors to be sampled
- **shape** – String to select between whether column (`'col'`) or row (`'row'`) vectors should be returned. Defaults to column vectors.

next()
Draw the next sample from the Sampler

Returns A new vector sampled from a Gaussian distribution with mean 0 and standard deviation 1

class modea.Sampling.**MirroredOrthogonalSampling**(*n*, *lambda_*, *shape='col'*, *base_sampler=None*)
Bases: object

Factory method returning a pre-defined mirrored orthogonal sampler in the right order: orthogonalize first, mirror second.

Parameters

- **n** – Dimensionality of the vectors to be sampled
- **shape** – String to select between whether column (`'col'`) or row (`'row'`) vectors should be returned. Defaults to column vectors
- **base_sampler** – A different Sampling object from which samples to be mirrored are drawn. If no base_sampler is given, a *GaussianSampling* object will be created and used.

Returns A `MirroredSampling` object with as base sampler an `OrthogonalSampling` object initialized with the given parameters.

next()
Draw the next sample from the Sampler

Returns A new vector, alternating between a new orthogonalized sample from the base_sampler and a mirror of the last.

reset()
Reset the internal state of this sampler, so the next sample is forced to be taken new.

class modea.Sampling.**MirroredSampling**(*n*, *shape='col'*, *base_sampler=None*)
Bases: `object`

A sampler to create mirrored samples using some base sampler (Gaussian by default) Returns a single vector each time, while remembering the internal state of whether the `next()` should return a new sample, or the mirror of the previous one.

Parameters

- **n** – Dimensionality of the vectors to be sampled
- **shape** – String to select between whether column (`'col'`) or row (`'row'`) vectors should be returned. Defaults to column vectors
- **base_sampler** – A different Sampling object from which samples to be mirrored are drawn. If no base_sampler is given, a *GaussianSampling* object will be created and used.

next()
Draw the next sample from the Sampler

Returns A new vector, alternating between a new sample from the base_sampler and a mirror of the last.

reset()
Reset the internal state of this sampler, so the next sample is forced to be taken new.

class modea.Sampling.**OrthogonalSampling**(*n*, *lambda_*, *shape='col'*, *base_sampler=None*)
Bases: `object`

A sampler to create orthogonal samples using some base sampler (Gaussian as default)

Parameters

- **n** – Dimensionality of the vectors to be sampled
- **lambda** – Number of samples to be drawn and orthonormalized per generation
- **shape** – String to select between whether column (`'col'`) or row (`'row'`) vectors should be returned. Defaults to column vectors
- **base_sampler** – A different Sampling object from which samples to be mirrored are drawn. If no base_sampler is given, a *GaussianSampling* object will be created and used.

next()
Draw the next sample from the Sampler

Returns A new vector sampled from a set of orthonormalized vectors, originally drawn from base_sampler

reset()
Reset the internal state of this sampler, so the next sample is forced to be taken new.

class modea.Sampling.**QuasiGaussianHaltonSampling**(*n*, *shape='col'*)
Bases: object

A quasi-Gaussian sampler based on a Halton sequence

Parameters

- **n** – Dimensionality of the vectors to be sampled
- **shape** – String to select between whether column ('col') or row ('row') vectors should be returned. Defaults to column vectors

next()
Draw the next sample from the Sampler

Returns A new vector sampled from a Halton sequence with mean 0 and standard deviation 1

class modea.Sampling.**QuasiGaussianSobolSampling**(*n*, *shape='col'*, *seed=None*)
Bases: object

A quasi-Gaussian sampler based on a Sobol sequence

Parameters

- **n** – Dimensionality of the vectors to be sampled
- **shape** – String to select between whether column ('col') or row ('row') vectors should be returned. Defaults to column vectors

next()
Draw the next sample from the Sampler

Returns A new vector sampled from a Sobol sequence with mean 0 and standard deviation 1

modea.Selection module

This module contains a collection of Selection operators.

Selection accepts (mu + lambda) individuals and returns (mu) individuals that are chosen to be the best of this generation according to which selection module is chosen.

modea.Selection.**best**(*population*, *new_population*, *param*)
Given the population, return the (mu) best. Also performs some 'housekeeping' for the CMA-ES by collecting all genotypes and most recent mutation vectors and storing them in the param object.

Parameters

- **population** – List of *FloatIndividual* objects containing the previous generation
- **new_population** – List of *FloatIndividual* objects containing the new generation
- **param** – *Parameters* object for storing all parameters, options, etc.

Returns A slice of the sorted new_population list.

modea.Selection.**bestGA**(*population*, *new_population*, *param*)
Given the population, return the (mu) best

Parameters

- **population** – List of *MixedIntIndividual* objects containing the previous generation
- **new_population** – List of *MixedIntIndividual* objects containing the new generation

- **param** – *Parameters* object for storing all parameters, options, etc.

Returns A slice of the sorted new_population list.

modea.Selection.**onePlusOneSelection**(*population*, *new_population*, *t*, *param*)
(1+1)-selection (with success history)

Parameters

- **population** – List of *FloatIndividual* objects containing the previous generation
- **new_population** – List of *FloatIndividual* objects containing the new generation
- **t** – Timestamp of the current generation being evaluated
- **param** – *Parameters* object for storing all parameters, options, etc.

Returns A slice of the sorted new_population list.

modea.Selection.**pairwise**(*population*, *new_population*, *param*)
Perform a selection on individuals in a population per pair, before letting *best()* make the final selection. Intended for use with a *MirroredSampling* sampler to prevent step-size bias.

Assumes that new_population contains pairs as [P1_a, P1_b, P2_a, P2_b, etc . . .]

Parameters

- **population** – List of *FloatIndividual* objects containing the previous generation
- **new_population** – List of *FloatIndividual* objects containing the new generation
- **param** – *Parameters* object for storing all parameters, options, etc.

Returns A slice of the sorted new_population list.

modea.Selection.**roulette**(*population*, *new_population*, *param*, *force_unique=False*)
Given the population, return mu individuals, selected by roulette, using 1/fitness as probability

Parameters

- **population** – List of *FloatIndividual* objects containing the previous generation
- **new_population** – List of *FloatIndividual* objects containing the new generation
- **param** – *Parameters* object for storing all parameters, options, etc.
- **force_unique** – Determine if an individual from the original population may be selected multiple times

Returns A slice of the sorted new_population list.

modea.Utils module

A collection of utilities for internal use by this package. Besides some trivial functions, this mainly includes the *ESFitness* class definition.

class modea.Utils.**ESFitness**(*fitnesses=None*, *target=1e-08*, *min_fitnesses=None*, *min_indices=None*, *num_successful=None*, *ERT=None*, *FCE=inf*, *std_dev_ERT=None*, *std_dev_FCE=None*)
Bases: object

Object to calculate and store the fitness measure for an ES and allow easy comparison. This measure consists of both the always available Fixed Cost Error (FCE) and the less available but more rigorous Expected Running Time (ERT).

All parameters are listed as optional, but at least one of the following combinations have to be given to obtain FCE/ERT values.

```
>>> ESFitness(fitnesses=fitnesses)
>>> ESFitness(min_fitnesses=min_fitnesses, min_indices=min_indices, num_
↪successful=num_successful)
>>> ESFitness(ERT=ERT, FCE=FCE)
```

If `fitnesses` is specified, all other parameters other than `target` are ignored and everything is calculated from that. Otherwise, ERT and FCE are calculated from `min_fitnesses`, `min_indices` and `num_successful`. Only if none of these are specified, the direct `ERT` and `FCE` values are stored (together with their corresponding `std_dev_` values if specified)

Parameters

- **fitnesses** – Nested lists: A list of the fitness progression for each run
- **target** – What value to use as target for calculating ERT. Default set in `Config`
- **min_fitnesses** – Single list containing the minimum value of the `fitnesses` list (if given instead)
- **min_indices** – Single list containing the index in the `fitnesses` list where the minimum was found
- **num_successful** – Integer to simply track how many of the runs reached the target
- **ERT** – *Estimated Running Time*
- **FCE** – *Fixed Cost Error*
- **std_dev_ERT** – Standard deviation corresponding to the ERT value
- **std_dev_FCE** – Standard deviation corresponding to the FCE value

`modea.Utils.`**`create_bounds`**(*values*, *percentage*)

For a given set of floating point values, create an upper and lower bound. Bound values are defined as a percentage above/below the given values.

Parameters

- **values** – List of floating point input values.
- **percentage** – The percentage value to use, expected as a fraction in the range (0, 1).

Returns Tuple (u_bound, l_bound), each a regular list.

`modea.Utils.`**`getBitString`**(*opts*)

Reverse of getOpts, transforms options dictionary to integer 'bitstring'

Parameters **opts** – Dictionary with option names and the chosen option

Returns A list of integers that serve as index for the options tuple

`modea.Utils.`**`getFitness`**(*individual*)

Function that can be used as key when sorting

Parameters **individual** – Some object of one of the classes from the *Individual* module

Returns Fitness attribute of the given individual object

`modea.Utils.`**`getFullOpts`**(*opts*)

Ensures that an options dictionary actually contains all options that have been defined. Any missing options are given default values inline.

Parameters **opts** – Dictionary to be checked for option names and the chosen option

modea.Utils.**getOpts**(*bitstring*)
Transformation from integer 'bitstring' to options dictionary

Parameters **bitstring** – List/array of integers that serve as index for the options tuple

Returns Dictionary with all option names and the chosen option

modea.Utils.**getPrintName**(*opts*)
Create a human-readable name from an options dictionary

Parameters **opts** – Dictionary to be checked for option names and the chosen option

Returns Human-readable string listing all active CMA-ES options for the given dictionary

modea.Utils.**getVals**(*init_values*)
Transformation from real numbered vector to values dictionary

Parameters **init_values** – List/array of real values that serve as initial values for parameters

Returns Dictionary containing name-indexed initial parameter values

modea.Utils.**guaranteeFolderExists**(*path_name*)
Make sure the given path exists after this call

modea.Utils.**intToRepr**(*integer*)
Dencode the ES-structure from a single integer back to the mixed base-2 and 3 representation. Reverse of *reprToInt()*

```
>>> intToRepr(93)
>>> [0,0,0,0,0,1,0,1,0,1,0]
```

Parameters **integer** – Integer (e.g. outoutput from reprToInt())

Returns String consisting of all structure choices concatenated,

modea.Utils.**reprToInt**(*representation*)
Encode the ES-structure representation to a single integer by converting it to base-10 as if it is a mixed base-2 or 3 number. Reverse of *intToRepr()*

```
>>> reprToInt([0,0,0,0,0,1,0,1,0,1,0])
>>> 93
```

Parameters **representation** – Iterable; the genotype of the ES-structure

Returns String consisting of all structure choices concatenated,

modea.Utils.**reprToString**(*representation*)
Function that converts the structure parameters of a given ES-structure representation to a string

```
>>> reprToInt([0,0,0,0,0,1,0,1,0,1,0])
>>> '00000101010'
```

Parameters **representation** – Iterable; the genotype of the ES-structure

Returns String consisting of all structure choices concatenated, e.g.: `00000101010`

Module contents

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

m

Index

I

M

N

O

P

Q

R

S

T

U

W