MolOpt

estevaopbs

Nov 03, 2021

CONTENTS:

CHAPTER

ONE

MOLOPT DOCUMENTATION

1.1 MolOpt module

class MolOpt.**MolOpt**(*first_molecule:* molecular.Molecule, *fitness_param: str*, *strategies:* genetic.Strategies, *max_age: int*, *pool_size: int*, *mutate_after_crossover: bool*, *crossover_elitism: list[int]*, *elitism_rate: list[int]*, *freedom_rate: int*, *parallelism: bool*, *local_opt: bool*, *max_seconds: Optional[Union[int, float]]*, *time_toler: Optional[Union[int, float]]*, *gens_toler: Optional[int]*, *max_gens: Optional[int]*, *save_directory: str*, *threads_per_calc: int*)

Bases: *genetic.Genetic*

Molecular geometry optimization class

Parameters **Genetic** (*ABC*) – Genetic algorithm abstract class

Initializes the MolOpt object

Parameters

- **first_molecule** (Molecule) – Molecule which is wanted to be optimized
- **fitness_param** (*str*) – The string which precedes the energy value in Molpro's output
- **strategies** (Strategies) – Strategies object
- **max_age** (*int*) – The max amount of times a Chromosome can suffer chaging strategies without improve its fitness
- **pool_size** (*int*) – The amount of candidates being optimized simultaneously
- **mutate_after_crossover** (*bool*) – If it's True, than after each crossover operation, the resultant child Chromosome suffer a mutate operation before return to the genetic algorithm, but if it's false the mutation doesn't occur and the child Chromosome is returned immediately after the crossover
- **crossover_elitism** (*Union[list[numeric], None]*) – The rate each candidate tends to be selected to be the gene's donor in any crossover operation from the best to the worst. Its lenght must be equal pool_size value. If pool_size is 3 and crossover_elitism is [3, 2, 1] the best candidate has the triple of the chance to be selected than the worst, the medium candidate has double. It can also receive None, and it means all candidates are equally probable to be selected for being the genes' donor on a crossover
- **elitism_rate** (*Union[list[int], None]*) – list of reprodution rate of each candidate, from the best to the worst. the sum of its elements also must be less or equal than pool_size. If pool_size is 16 and elitism_rate is [4, 3, 2] it means the best candidate in the current generation's pool of candidates will provide 4 descendants for the next generation, the second best will provide 3 and the third best will provide two, then then remain 7 available spaces

in the next generation's pool will be filled with one descendant of each of the next seven candidates in this order

- **freedom_rate** (*int*) – The number of candidate generation strategies (Mutate, Crossover and Create) the candidate will suffer aways a new candidate is needed to be generated (if Create is selected it means the candidate is supposed to be substituted by a whole new one without any relation with the parent candidate)
- **parallelism** (*bool*) – If it's True than each fitness calculation will be done in a different process, what changes the whole dynamics of the genetic algorithm. With paraellism enabled, the concept of generations emerges as we can have different candidates being caculated at the same time. If it's False, there will be no generations and candidates' fitnesses will be calculated sequentially
- **local_opt** (*bool*) – If its True makes that every time the algorithm genetic gets a new best candidate it is sent to the local_optimize function (which in this case must be override by the subclass) that is supposed to perform a local optimization (in the solution-space of the specific problem) over the genes of the best candidate, and then return a new best candidate with the optimized genes
- **max_seconds** (*Union[numeric, None]*) – The max amount of seconds the genetic algorithm can run. Once exceeded this amount, the the running will be stoped and the best candidate will be returned. It can also receive None, and in this case this limit wouldn't exist
- **time_toler** (*Union[numeric, None]*) – The max amount of seconds the algorithm can still running without has any improvements on its best candidate's fitness. Once exceeded this amount, the running will be stoped and the best candidate will be returned. It can also receive None, and in this case this limit wouldn't exist
- **gens_toler** (*Union[numeric, None]*) – The maximum amount of generations the algorithm genetic can run in sequence without having any improvement on it's best parent fitness. It can also receive None and in this case this limit wouldn't exist. It only works when parallelism is True, otherwise it doesn't affect anything
- **max_gens** (*Union[numeric, None]*) – The max amount of generations the genetic algorithm can run. Once exceeded this amount, the the running will be stoped and the best candidate will be returned. It can also receive None, and in this case this limit wouldn't exist. It only works when parallelism is True, otherwise it doesn't affect anything
- **save_directory** (*str*) – The directory address relative to __main__ where the outputs will be saved. If its None than it will receive the instant of time the running started
- **threads_per_calc** (*int*) – Number of threads useds in each Molpro calculation

static catch(*candidate:* genetic.Chromosome) → None

Static method which will be executed if an error occurs during a candidate generation

Parameters **candidate** (Chromosome) – Candidate which during generation some exception occurred

static crossover_1(*parent:* genetic.Chromosome, *donor:* genetic.Chromosome) → *molecular.Molecule*

Produces a new molecule with the crossover of parent's and donor's molecules by cutting each one in one point and combining the resultant pieces.

Parameters

- **parent** (Chromosome) – Candidate which Molecule will suffer crossover_1
- **donor** (Chromosome) – Candidate which will donate parameters for the crossover_1 operation

Returns Child molecule

Return type *Molecule*

static crossover_2(*parent:* genetic.Chromosome, *donor:* genetic.Chromosome) → *molecular.Molecule*
Produces a new molecule with the crossover of parent's and donor's molecules by cutting each one in two points and combining the resultant pieces

Parameters

- **parent** (Chromosome) – Candidate which Molecule will suffer crossover_1
- **donor** (Chromosome) – Candidate which will donate parameters for the crossover_1 operation

Returns Child molecule

Return type *Molecule*

static crossover_n(*parent:* genetic.Chromosome, *donor:* genetic.Chromosome) → *molecular.Molecule*
Returns a new molecule which randomly carries parameters from the parent's and donor's molecules

Parameters

- **parent** (Chromosome) – Candidate which Molecule will suffer crossover_1
- **donor** (Chromosome) – Candidate which will donate parameters for the crossover_1 operation

Returns Child molecule

Return type *Molecule*

get_fitness(*candidate:* genetic.Chromosome) → float
Receives a candidate's Chromosome and returns its fitness

Parameters **candidate** (Chromosome) – Candidate which fitness must be calculated

Returns Candidate's fitness

Return type float

local_optimize(*candidate:* genetic.Chromosome) → *molecular.Molecule*
Executes geometric optimization over the candidate's molecule using Molpro and returns a new molecule with the optimized geometry

Parameters **candidate** (Chromosome) – Candidate which Molecule will suffer local_optimize operation

Returns Optimized molecule

Return type *Molecule*

static mutate_angles(*parent:* genetic.Chromosome) → *molecular.Molecule*
Returns parent's molecule's copy with some random angle parameter randomized between 0 and 360 degrees

Parameters **parent** (Chromosome) – Candidate which Molecule will suffer mutate_angles

Returns New molecule

Return type *Molecule*

static mutate_distances(*parent:* genetic.Chromosome) → *molecular.Molecule*
Returns a parent's molecule's copy with some random distance parameter randomized in the range gave by parent.genes.rand_range

Parameters **parent** (Chromosome) – Candidate which Molecule will suffer mutate_distances

Returns New molecule

Return type *Molecule*

static randomize(*parent:* genetic.Chromosome) → *molecular.Molecule*
Returns a parent's molecule's copy with all distances and angles parameters randomized

Parameters **parent** (Chromosome) – Candidate which Molecule will suffer randomize operation

Returns New molecule

Return type *Molecule*

static save(*candidate:* genetic.Chromosome, *file_name: str*, *directory: str*) → None
Saves the candidate data in a .inp document

Parameters

- **candidate** (Chromosome) – Candidate which data will be saved
- **file_name** (*str*) – Document's name
- **directory** (*str*) – Directory where the document will be saved

static swap_mutate(*parent:* genetic.Chromosome) → *molecular.Molecule*
Returns a parent's molecule's copy with randomly swapped places parameters

Parameters **parent** (Chromosome) – Candidate which Molecule will suffer swap_mutate

Returns New molecule

Return type *Molecule*

1.2 genetic module

class genetic.**Chromosome**(*genes: Optional[Any] = None*, *fitness: Optional[Any] = None*, *strategy: list[collections.abc.Callable[[~ Chromosome], ~ Genes]] = []*, *age: int = 0*, *lineage: list = []*, *label: Optional[str] = None*)

Bases: object

Object that represents the candidates

Initializes the Chromosome object

Parameters

- **genes** (*Any, optional*) – What is wanted to optimize
- **fitness** (*Any, optional*) – Value which describes how much the genes fits what is wanted. It can be of any type since it can be compared with > and < and can be printed
- **strategy** (*list[Callable[[Chromosome], Genes]], optional*) – Container which stores the functions used to obtain the current Chromosome, defaults to []
- **age** (*int, optional*) – How many times the candidate was modificated without having any improvement, defaults to 0
- **lineage** (*list[Chromosome], optional*) – The historic of Chromosomes used to find the current Chromosome, defaults to []
- **label** (*str, optional*) – A tag which can be used to identify the Chromosome object, defaults to None, defaults to None

property strategy_str: str

Returns a string which represents the list of the functions used to obtain the current Chromosome object :return: String of a list of the names of the functions used to obtain the current Chromosome object :rtype: str

class genetic.**Create**(*methods: list[collections.abc.Callable[[genetic.Chromosome], ~ Genes]]*, *methods_rate: list[typing.Union[int, float]]*)

Bases: `object`

A container for storing genes creation functions and its rates. When called it receives the first_parent Chromosome and returns a new Chromosome This class is supposed to receive functions which receive a generic parent Chromosome and return a whole new genes without any bound to the parent's one. The exceptions are mutate_best and mutate_first, which respectively returns a result of passing the best and the first parent, respectively to the mutate object

Returns New Chromosome object with the created genes

Return type *Chromosome*

Initializes the Crossover object by receiving its parameters

Parameters

- **methods** (*list[Callable[[Chromosome], Genes]]*) – Functions which receives the first_parent Chromosome and returns a new genes
- **methods_rate** (*list[numeric]*) – The rate the functions tends be randomly chosen when the Crossover object is called. It must have the same lenght as methods. Suppose methods is [m1, m2, m3] and methods_rate is [1, 2, 3]. m2 tends to be chosen twice the m1 is and m3 thrice the m1 is.

class genetic.**Crossover**(*methods: list[collections.abc.Callable[[genetic.Chromosome], ~ Genes]]*, *methods_rate: list[typing.Union[int, float]]*)

Bases: `object`

A container for storing crossover functions and its rates. When called it receives two Chromosome objects and returns a new Chromosome This class is supposed to receive functions which receive two Chromosome objects and returns a random combination of their genes

Returns Child Chromosome

Return type *Chromosome*

Initializes the Crossover object by receiving its parameters

Parameters

- **methods** (*list[Callable[[Chromosome]*) – Functions which receives two Chromosome objects and returns a new genes
- **methods_rate** (*list[numeric]*) – The rate the functions tends be randomly chosen when the Crossover object is called. It must have the same lenght as methods. Suppose methods is [m1, m2, m3] and methods_rate is [1, 2, 3]. m2 tends to be chosen twice the m1 is and m3 thrice the m1 is.

class genetic.**Genetic**(*first_genes:* genetic.Chromosome, *strategies:* genetic.Strategies, *max_age: Optional[int]*, *pool_size: int*, *mutate_after_crossover: bool*, *crossover_elitism: Optional[list[typing.Union[int, float]]]*, *elitism_rate: Optional[list[int]]*, *freedom_rate: int*, *parallelism: bool*, *local_opt: bool*, *max_seconds: Optional[Union[int, float]]*, *time_toler: Optional[Union[int, float]]*, *gens_toler: Optional[Union[int, float]]*, *max_gens: Optional[Union[int, float]]*, *save_directory: str*)

Bases: `abc.ABC`

Genetic algorithm abstract class This abstract class provides a framework for creating problem-specific genetic algorithms. To use it you must create a class that inherits it. The class that inherits it must have at least two methods: get_fitness and save.

Parameters **ABC** (*class*) – Helper class that provides a standard way to create an abstract class using inheritance.

Initializes the Genetic object by receiving its parameters

Parameters

- **first_genes** (Chromosome) – The genes of the first candidate in the genetic algorithm
- **strategies** (Strategies) – Strategies object
- **max_age** (*int*) – The max amount of times a Chromosome can suffer chaging strategies without improve its fitness
- **pool_size** (*int*) – The amount of candidates being optimized simultaneously
- **mutate_after_crossover** (*bool*) – If it's True, than after each crossover operation, the resultant child Chromosome suffer a mutate operation before return to the genetic algorithm, but if it's false the mutation doesn't occur and the child Chromosome is returned immediately after the crossover
- **crossover_elitism** (*Union[list[numeric], None]*) – The rate each candidate tends to be selected to be the gene's donor in any crossover operation from the best to the worst. Its lenght must be equal pool_size value. If pool_size is 3 and crossover_elitism is [3, 2, 1] the best candidate has the triple of the chance to be selected than the worst, the medium candidate has double. It can also receive None, and it means all candidates are equally probable to be selected for being the genes' donor on a crossover
- **elitism_rate** (*Union[list[int], None]*) – List of reprodution rate of each candidate, from the best to the worst. the sum of its elements also must be less or equal than pool_size. If pool_size is 16 and elitism_rate is [4, 3, 2] it means the best candidate in the current generation's pool of candidates will provide 4 descendants for the next generation, the second best will provide 3 and the third best will provide two, then then remain 7 available spaces in the next generation's pool will be filled with one descendant of each of the next seven candidates in this order
- **freedom_rate** (*int*) – The number of candidate generation strategies (Mutate, Crossover and Create) the candidate will suffer aways a new candidate is needed to be generated (if Create is selected it means the candidate is supposed to be substituted by a whole new one without any relation with the parent candidate)
- **parallelism** (*bool*) – If it's True than each fitness calculation will be done in a different process, what changes the whole dynamics of the genetic algorithm. With paraellism enabled, the concept of generations emerges as we can have different candidates being caculated at the same time. If it's False, there will be no generations and candidates' fitnesses will be calculated sequentially
- **local_opt** (*bool*) – If its True makes that every time the algorithm genetic gets a new best candidate it is sent to the local_optimize function (which in this case must be override by the subclass) that is supposed to perform a local optimization (in the solution-space of the specific problem) over the genes of the best candidate, and then return a new best candidate with the optimized genes
- **max_seconds** (*Union[numeric, None]*) – The max amount of seconds the genetic algorithm can run. Once exceeded this amount, the the running will be stoped and the best candidate will be returned. It can also receive None, and in this case this limit wouldn't exist

- **time_toler** (*Union[numeric, None]*) – The max amount of seconds the algorithm can still running without has any improvements on its best candidate's fitness. Once exceeded this amount, the running will be stoped and the best candidate will be returned. It can also receive None, and in this case this limit wouldn't exist
- **gens_toler** (*Union[numeric, None]*) – The maximum amount of generations the algorithm genetic can run in sequence without having any improvement on it's best parent fitness. It can also receive None and in this case this limit wouldn't exist. It only works when parallelism is True, otherwise it doesn't affect anything
- **max_gens** (*Union[numeric, None]*) – The max amount of generations the genetic algorithm can run. Once exceeded this amount, the the running will be stoped and the best candidate will be returned. It can also receive None, and in this case this limit wouldn't exist. It only works when parallelism is True, otherwise it doesn't affect anything
- **save_directory** (*str*) – The directory address relative to __main__ where the outputs will be saved. If its None than it will receive the instant of time the running started

static catch(*candidate:* genetic.Chromosome) → None

Static method which will be executed if an error occurs during a candidate generation It can be override if needed. By default it just raises an exception

Parameters **candidate** (Chromosome) – Candidate which was being generated while error occurs

Raises **Exception** – Raises exception if error occurs during candidate generation

static display(*candidate:* genetic.Chromosome, *timediff: float*) → None

Generate what is printed on console everytime a new best candidate is reached It can be override if needed

Parameters

- **candidate** (Chromosome) – Best candidate founded
- **timediff** (*float*) – Time difference between the candidate is founded and the start of the execution

abstract get_fitness(*candidate:* genetic.Chromosome) → genetic.Fitness

Abstract method which must be override by one which receives a candidate's Chromosome and returns it's fitness

Parameters **candidate** (*Fitness*) – Candidate wich fitness is wanted

Return type Fitness

load() → *genetic.Chromosome*

Returns a candidate with the first_genes and the fitness of the first_parent

Returns First candidate

Return type *Chromosome*

local_optimize(*candidate:* genetic.Chromosome) → genetic.Genes

This method must be override in case of local_opt is True for a method which receives a candidate's Chromosome and returns its genes locally optimized

Parameters **candidate** (Chromosome) – Candidate which genes are wanted to be optimized

Raises **Exception** – Raises exception if this method was not override by the subclass

Returns Optimized genes

Return type Genes

mutate_best(*best_candidate:* genetic.Chromosome) → genetic.Genes
The actual mutate_best function Creation function that receives the Chromosome of the best candidate and pass it to the Mutate object to return the result

Parameters **best_candidate** (Chromosome) – Best candidate

Returns Mutated genes of the best candidate

Return type genes

mutate_first(*first_parent:* genetic.Chromosome) → genetic.Genes
The actual mutate_first function Creation function that receives the Chromosome of the first candidate and pass it to the Mutate object to return the result

Parameters **first_parent** (Chromosome) – First created candidate

Returns Mutated genes of the first candidate

Return type genes

run() → *genetic.Chromosome*
Starts the genetic algorithm execution

Returns Best candidate

Return type *Chromosome*

abstract save(*candidate:* genetic.Chromosome, *file_name: str*, *directory: str*) → None
Abstract method which must be override by one which receives a candidate, a file_name and a directory This method which overrides it must receive a candidate's Chromosome, a string which is the name of the file the candidate will be saved and a string which is the directory where it will be saved than this function must save the candidate relevant informations in a document named as given in the directory given. The directory is given relative to __main__

Parameters

- **candidate** (Chromosome) – Candidate which is wanted to save
- **file_name** (*str*) – Name of the file in which the candidate's informations must be saved
- **directory** (*str*) – Directory where the file with the candidate's information must be saved

class genetic.**Mutate**(*methods: list[collections.abc.Callable[[genetic.Chromosome], ~ Genes]]*, *methods_rate: list[typing.Union[int, float]]*)

Bases: object

A container for storing mutation functions and its rates. When called it receives a Chromosome and returns a new Chromosome This is supposed to receive functions which receives a Chromosome object and returns its genes with some random modification

Returns Mutated Chromosome

Return type *Chromosome*

Initializes the Mutate object by receiving its parameters

Parameters

- **methods** (*list[Callable[[Chromosome], Genes]]*) – Functions which receives a Chromosome object and returns a new genes
- **methods_rate** (*list[numeric]*) – The rate the functions tends be randomly chosen when the Mutate object is called. It must have the same lenght as methods. Suppose methods is

[m1, m2, m3] and methods_rate is [1, 2, 3]. m2 tends to be chosen twice the m1 is and m3 thrice the m1 is

class genetic.**Strategies**(*strategies: Tuple[genetic.Mutate, genetic.Crossover, genetic.Create]*, *strategies_rate: list[typing.Union[int, float]]*)

Bases: object

A container for sotoring the candidate generation strategies (Create, Mutate and Crossover objects) and its rates It must receive only one of each generation strategy (Create, Mutate and Crossover) object.

Returns New Chromosome object with the genes generated by the strategy randomly selected

Return type *Chromosome*

Initializes the Strategies object by receiving its parameters

Parameters

- **strategies** (*Tuple[Create, Mutate, Crossover]*) – A list with Create, Mutate and Crossover objects
- **strategies_rate** (*list[numeric]*) – The rate the Mutate, Crossover and Create objects tends be randomly chosen when the Strategies object is called. It must have the same lenght as methods. Suppose strategies is [s1, s2, s3] and strategies_rate is [1, 2, 3]. s2 tends to be chosen twice the s1 is and s3 thrice the s1 is.

genetic.**mutate_best**(*best_candidate: genetic.Chromosome*) → genetic.Genes

Creation function that receives the Chromosome of the best candidate and pass it to the Mutate object to return the result This function is actually a void, it exists only to be called by import and to be override by the actual mutate_best function

Parameters **best_candidate** (Chromosome) – Best candidate

Returns Mutated genes of the best candidate

Return type genes

genetic.**mutate_first**(*first_parent: genetic.Chromosome*) → genetic.Genes

Creation function that receives the Chromosome of the first created parent and pass it to the Mutate object to return the result This function is actually a void, it exists only to be called by import and to be override by the actual mutate_first function

Parameters **first_parent** (Chromosome) – First created candidate

Returns Mutated genes of the first candidate

Return type genes

1.3 molecular module

class molecular.**Molecule**(*basis: str*, *geometry: list[list[typing.Union[str, int, float]]]*, *settings: list[str]*, *parameters: dict[str, typing.Union[int, float]] = {}*, *rand_range: Optional[Tuple[Union[int, float], Union[int, float]]] = None*, *label: Optional[str] = None*, *output: Optional[str] = None*, *output_values: dict[str, typing.Union[int, float]] = {}*, *was_optg: bool = False*)

Bases: object

Molecular geometry compatible with Molpro This class provides a framework for storing molecular geometry, generating Molpro inputs and store output information. It also brings functions to work with genetic algorithm.

Initializes the Molecule object by receiving its parameters

Parameters

- **basis** (*str*) – Hilbert space basis
- **geometry** (*list[list[str]]*) – Z-matrix input
- **settings** (*list[str]*) – Molpro calculation settings
- **parameters** (*dict[str, numeric], optional*) – The values of geometry variables, defaults to dict()
- **rand_range** (*Tuple[numeric, numeric], optional*) – Min and max values that can be generated when distances are mutated, defaults to None
- **label** (*str, optional*) – A tag which can be used to identify the Molecule object, defaults to None
- **output** (*str, optional*) – The address of the molecule's .out document generated by Molpro, defaults to None
- **output_values** (*dict[str, numeric], optional*) – Values extracted from the output document, defaults to dict()
- **was_optg** (*bool, optional*) – True if the geometry was already optmized with optg, false if it doesn't

copy() → *molecular.Molecule*

Returns a totally independent copy of itself

Returns Copy of self

Return type *Molecule*

property dist_unit: str

Gets the distance unit used to describe the molecule

Returns Distance unit

Return type str

get_value(*wanted: list[str]*, *document: Optional[str] = None*, *directory: str = 'data'*, *keep_output: bool = False*, *nthreads: int = 1*, *update_self: bool = True*) → dict[str, float]

Reads the Molpro's output file and return the wanted values. Reads the Molpro's output file, searchs for wanted strings and gets the numeric value that is in the same row then returns a dictionary where the keys are the wanted strings and the values are the numeric strings correspondents. If document.out already exists in /directory the document will be read and the values returned. If it doesn't and document.inp already exists in directory, it will execute Molpro over the input. If neither document.out nor document.inp exists, document.inp will be created and Molpro executed over it and after document.inp will be deleted

Parameters

- **wanted** (*list[str]*) – list of variables to be search in the output. They must be the string which precedes the wanted numeric value in molpro's output
- **document** (*str, optional*) – Documents' name, defaults to None
- **directory** (*str, optional*) – Directory adress where the input and the output will be relative to __main__, defaults to 'data'
- **keep_output** (*bool, optional*) – If it's True, the output will be kept and its name will be put in self.output, else it will be deleted after is read. defaults to False
- **nthreads** (*int, optional*) – Number of threads useds in Molpro calculation, defaults to 1

- **update_self** (*bool, optional*) – If it's True, self.output_values will be updated with each item of wanted. If it's False, self.output_values will not be updated, and the output values will can only be accessed by the returned dict

Returns Wanted strings and correspondent values

Return type dict[str, float]

static load(*file: str*, *rand_range: Optional[Tuple[Union[int, float], Union[int, float]]] = None*, *label: Optional[str] = None*, *output: Optional[str] = None*, *output_values: dict[str, typing.Union[int, float]] = {}*, *was_optg: bool = False*) → *molecular.Molecule*

Loads a molecule from a .inp document and returns its Molecule object

Parameters

- **file** (*str*) – File name, it must end with .inp but the .inp may not be included here
- **rand_range** (*Tuple[numeric, numeric], optional*) – Min and max values that can be generated when distances are mutated, defaults to None
- **label** (*str, optional*) – A tag which can be used to identify the object, defaults to None
- **output** (*str, optional*) – The address of the molecule's .out document generated by Molpro, defaults to None
- **output_values** (*dict[str, numeric], optional*) – Values extracted from the output document, defaults to dict()
- **was_optg** (*bool, optional*) – True if the geometry was already optmized with optg, false if it doesn't

Raises **Exception** – Invalid Z-matrix

Returns Loaded molecule

Return type *Molecule*

The file must have the one of the following structures:

***,

basis={ ! ! aluminium (6s,4p) -> [3s,2p] s, AL , 0.5605994123E+00, 0.1923360636E+00, 0.7304329554E+01, 0.1852570854E+01, 0.1343774607E+03, 0.2391912027E+02 c, 1.2, -0.2983986045E+00, 0.1227982887E+01 c, 3.4, 0.4947176920E-01, 0.9637824081E+00 c, 5.6, 0.4301284983E+00, 0.6789135305E+00 p, AL , 0.7304329554E+01, 0.1852570854E+01, 0.5605994123E+00, 0.1923360636E+00 c, 1.2, 0.5115407076E+00, 0.6128198961E+00 c, 3.4, 0.3480471912E+00, 0.7222523221E+00 }

r1=2.706 r2=2.414 r3=2.481 r4=2.817 r5=2.529 r6=2.547 r7=2.510 r8=2.401 r9=2.666 t1=66.357 t2=65.256 t3=115.675 t4=107.280 t5=105.113 t6=97.637 t7=104.449 t8=63.620 a1=104.0 a2=89.4 a3=345.8 a4=272.6 a5=43.5 a6=341.2 a7=238.2

geometry={ang Al Al, 1, r1 Al, 2, r2 , 1, t1 Al, 2, r3 , 1, t2, 3, a1 Al, 3, r4 , 2, t3, 1, a2 Al, 1, r5 , 2, t4, 3, a3 Al, 5, r6 , 3, t5, 2, a4 Al, 7, r7, 5, t6, 3, a5 Al, 5, r8, 3, t7, 2, a6 Al, 4, r9, 2, t8, 1, a7 }

SET,CHARGE=0 direct {ks, b3lyp,maxit=200}

—

or

***,

basis={ ! ! aluminium (6s,4p) -> [3s,2p] s, AL , 0.5605994123E+00, 0.1923360636E+00, 0.7304329554E+01, 0.1852570854E+01, 0.1343774607E+03, 0.2391912027E+02 c, 1.2, -0.2983986045E+00, 0.1227982887E+01 c, 3.4, 0.4947176920E-01, 0.9637824081E+00 c, 5.6, 0.4301284983E+00, 0.6789135305E+00 p, AL , 0.7304329554E+01, 0.1852570854E+01, 0.5605994123E+00, 0.1923360636E+00 c, 1.2, 0.5115407076E+00, 0.6128198961E+00 c, 3.4, 0.3480471912E+00, 0.7222523221E+00 }

geometry={ang Al Al, 1, 2.706 Al, 2, 2.414, 1, 66.357 Al, 2, 2.481, 1, 65.256, 3, 104.0 Al, 3, 2.817, 2, 115.675, 1, 89.4 Al, 1, 2.529, 2, 107.280, 3, 345.8 Al, 5, 2.547, 3, 105.113, 2, 272.6 Al, 7, 2.510, 5, 97.637, 3, 43.5 Al, 5, 2.401, 3, 104.449, 2, 341.2 Al, 4, 2.666, 2, 63.620, 1, 238.2 }

SET,CHARGE=0 direct {ks, b3lyp,maxit=200}

—

Respecting empty lines between different sections of the file. That is, it must have one empty line between each section of the file. Here we can see we can have until six sections in the file: the beggining's section or just '***,', the basis' section, the parameters' section (that can exists or don't), the geometry's section, the settings' section and the end section or just '—'

mutate_angles(*times: int = 1*) → *molecular.Molecule*

Provokes a mutation in some random angle parameter Selects a random angle parameter from the geometry and assign it a random value between 0 and 360 degrees. This process is repeated an amount of times equal to times

Parameters **times** (*int, optional*) – Number of mutations, defaults to 1

Returns Itself mutated

Return type *Molecule*

mutate_distances(*times: int = 1*) → *molecular.Molecule*

Provokes a mutation in some random distance parameter Selects a random distance parameter from the geometry and assign it a random value in the range gave by self.rand_range. This process is repeated a number of times equal to times

Parameters **times** (*int*) – Number of mutations, defaults to 1

Returns Itself mutated

Return type *Molecule*

optg(*wanted: list[str]*, *directory: str = 'data'*, *nthreads: int = 1*, *keep_output=False*) → *molecular.Molecule*

Turns the molecule in its own geometric optimized version

Parameters

- **wanted** (*list[str]*) – list of variables to be search in the output
- **directory** (*str, optional*) – Directory adress where the input and the output will be relative to __main__, defaults to 'data'
- **nthreads** (*int, optional*) – Number of threads useds in Molpro calculation, defaults to 1
- **keep_output** (*bool, optional*) – If it's True, the output will be kept, else it will be deleted after be read. defaults to False

Returns Itself optmized version

Return type *Molecule*

save(*document: Optional[str] = None*, *directory: str = 'data'*) → None
Saves the object data in a .inp document Saves the object data in document.inp. If document receives None it'll be the molecule's label, if it still None it will be str(self.__hash__()). If the directory didn't exist it wil be created

Parameters

- **document** (*str, optional*) – Document's name, defaults to None
- **directory** (*str, optional*) – Directory where the document will be saved, defaults to 'data'

swap_mutate() → *molecular.Molecule*
Provokes a swap mutation in itsef

Returns Itself mutated

Return type *Molecule*

molecular.**crossover_1**(*parent:* molecular.Molecule, *donor:* molecular.Molecule, *label: Optional[str] = None*) → *molecular.Molecule*
Produces a new molecule with the crossover of parent and donor molecules by cutting each one in one point and combining the resultant pieces Creates a copy of the parent molecule then randomly choices a row and a 'column' from the geometry and there divides the geometry in two pieces. Then randomly pick one of these pieces and attach with the complementar part provided by the donor molecule generating the geometry of the child molecule

Parameters

- **parent** (Molecule) – Parent molecule
- **donor** (Molecule) – Donor molecule
- **label** (*str, optional*) – A tag which can be used to identify the child molecule, defaults to None

Returns Child molecule

Return type *Molecule*

molecular.**crossover_2**(*parent:* molecular.Molecule, *donor:* molecular.Molecule, *label: Optional[str] = None*) → *molecular.Molecule*
Produces a new molecule with the crossover of parent and donor molecules by cutting each one in two points and combining the resultant pieces Randomly cuts the parent molecule's geometry in two points. The sequence between these two points will either replace its correspondent in the donor molecule's geometry or be replaced by it (randomly) generating the geometry of the child molecule that will be returned. At leas four atoms are needed to realize this process. Trying it with molecules smaller than it will raise an exception

Parameters

- **parent** (Molecule) – Parent molecule
- **donor** (Molecule) – Donor molecule
- **label** (*str, optional*) – A tag which can be used to identify the child molecule, defaults to None

Raises **Exception** – At least 4 atoms are needed to perform crossover_2

Returns child Molecule

Return type *Molecule*

molecular.**crossover_n**(*parent:* molecular.Molecule, *donor:* molecular.Molecule, *label: Optional[str] = None*) → *molecular.Molecule*

Returns a new molecule which randomly carries parameters from the parent and donor molecules Creates a copy of the parent molecule, randomly choices an amount of parameters the minimun being one and the maximun being the total amount of parameters minus one. Then randomly choices this amount of parameters from the donor molecule to replace the respectives parameters the child molecule inherited from parent molecule

Parameters

- **parent** (Molecule) – Parent molecule
- **donor** (Molecule) – Donor molecule
- **label** (*str, optional*) – A tag which can be used to identify the child molecule, defaults to None

Returns Child molecule

Return type *Molecule*

molecular.**mutate_angles**(*molecule:* molecular.Molecule, *times: int = 1*, *label: Optional[str] = None*) → *molecular.Molecule*

Returns a molecule's copy with some random angle parameter randomized between 0 and 360 degrees Creates a moecule's copy and randomly choices an angle parameter and set it value to a random value between 0 and 360. The amount of mutations performed is equal times

Parameters

- **molecule** (Molecule) – Original molecule
- **times** (*int, optional*) – Amount of angle mutations, defaults to 1
- **label** (*str, optional*) – Label of the new molecule, defaults to None

Returns New molecule

Return type *Molecule*

molecular.**mutate_distances**(*molecule:* molecular.Molecule, *times: int = 1*, *label: Optional[str] = None*) → *molecular.Molecule*

Returns a molecule's copy with some random distance parameter randomized in the range gave by rand_range Creates a moecule's copy and randomly choices a distance parameter and set it value to a random value between 0 and self.rand_range. The amount of mutations performed is equal times

Parameters

- **molecule** (Molecule) – Original molecule
- **times** (*int, optional*) – Amount of angle mutations, defaults to 1
- **label** (*str, optional*) – Label of the new molecule, defaults to None

Returns New molecule

Return type *Molecule*

molecular.**optg**(*molecule:* molecular.Molecule, *wanted_energy: str*, *directory: str = 'data'*, *nthreads: int = 1*, *keep_output: bool = False*) → *molecular.Molecule*

Executes geometric optimization over the molecule using Molpro and returns a new molecule with the optimized geometry Creates a copy of the original molecule, appends 'optg' in the settings if it's not already there and calls the get_value function with it. Then updates the wanted_energy in optmized_molecule.output_values and updates all angles and distances parameters in [optmized_molecule.parameters]. To use this function there cannont be any literal value in the molecule's geometry, all values must be as variables names in geometry and has its literal values declared in molecule.parameters

Parameters

- **molecule** (Molecule) – Original molecule
- **wanted_energy** (*str*) – The key wanted for the energy in optimized_molecule.output_values
- **directory** (*str, optional*) – Directory adress where the input and the output will be relative to __main__, defaults to 'data'
- **nthreads** (*int, optional*) – Number of threads useds in Molpro calculation, defaults to 1
- **keep_output** (*bool, optional*) – If it's True, the output will be kept and its name will be put in self.output, else it will be deleted after is read. defaults to False

Returns Optmized molecule

Return type *Molecule*

molecular.**random_molecule**(*molecular_formula: str*, *basis: str*, *settings: list[str]*, *rand_range: Tuple[Union[int, float], Union[int, float]]*, *label: Optional[str] = None*, *dist_unit: str = 'ang'*) → *molecular.Molecule*

Creates molecule with a entirely random geometry given the molecular formula Creates a geometry with all distances parameters randomized in the range gave by rand_range and all angles randomized between 0 and 360 given the molecular formula which is a string containing the elements followed by its amount like'H2O', 'C6H12O6', 'Al10'… It doesn't support parenthesys and is invariant to the order of elements. What matters in molecular_formula is which number follows which element

Parameters

- **molecular_formula** (*str*) – The molecular formula of the wanted molecule
- **basis** (*str*) – Hilbert space basis
- **settings** (*list[str]*) – Molpro calculation settings
- **rand_range** (*Tuple[numeric, numeric]*) – Min and max values that can be generated when distances are mutated
- **label** (*str, optional*) – A tag which can be used to identify the molecule, defaults to None
- **dist_unit** (*str, optional*) – Distance unit, defaults to 'ang'

Returns Random molecule

Return type *Molecule*

molecular.**randomize**(*molecule:* molecular.Molecule, *label: Optional[str] = None*) → *molecular.Molecule*

Returns a molecule's copy with all distances and angles parameters randomized Creates a molecule's copy and replace all distance parameters with random values in the range gave by molecule.rand_range and replace all angle parameters with random values between 0 and 360

Parameters

- **molecule** (Molecule) – Original molecule
- **label** (*str, optional*) – A tag which can be used to identify the new molecule, defaults to None

Returns New molecule

Return type *Molecule*

molecular.**swap_mutate**(*molecule:* molecular.Molecule, *times: int = 1*, *label: Optional[str] = None*) → *molecular.Molecule*

Returns a molecule's copy with randomly swapped places parameters Creates a molecule's copy and randomly swap parameters of its places. It makes a amount of random swaps equal times.

Parameters

- **molecule** (Molecule) – Original molecule
- **times** (*int, optional*) – Number of random swaps, defaults to 1
- **label** (*str, optional*) – Label of the new molecule, defaults to None

Returns New molecule

Return type *Molecule*

PYTHON MODULE INDEX

INDEX