

---

**MolOpt**

**Estevão Santos**

**Nov 06, 2021**



**CONTENTS:**

<b>1</b>	<b>MolOpt</b>	<b>1</b>
1.1	MolOpt package . . . . .	1
<b>2</b>	<b>Indices and tables</b>	<b>15</b>
	<b>Python Module Index</b>	<b>17</b>
	<b>Index</b>	<b>19</b>



## MOLOPT

## 1.1 MolOpt package

### 1.1.1 genetic subpackage

```
class 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.

**property strategy\_str:** str

Returns a string which represents the list of the functions used to obtain the current Chromosome object

**Returns** String of a list of the names of the functions used to obtain the current Chromosome object

**Return type** str

```
class Create(methods: list[collections.abc.Callable[[MolOpt.genetic.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 the 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*

```
class Crossover(methods: list[collections.abc.Callable[[MolOpt.genetic.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*

```
class Genetic(first_genes: MolOpt.genetic.genetic.Chromosome, strategies: MolOpt.genetic.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 to 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

```
__generate_parent(queue: Optional[multiprocessing.context.BaseContext.Queue] = None, label:
                  Optional[str] = None)
```

Tries to generate a new candidate using the Create object from strategies inside a ‘while True’ loop with a ‘try except’ statement which is broke when the new candidate is successfully generated. If an exception occurs during the candidate generation try, the function ‘catch’ will be called with such candidate as argument

**Parameters**

- **queue** (`mp.Queue`, *optional*) – Multiprocessing queue by which the candidates are returned in case of parallelism, defaults to None
- **label** (`str`, *optional*) – New parent’s Chromosome’s label, defaults to None

**Returns** New candidate

**Return type** *Chromosome*

```
__get_child(candidates: list[MolOpt.genetic.genetic.Chromosome], parent_index: int, queue:
             Optional[multiprocessing.context.BaseContext.Queue] = None, child_index: Optional[int] =
             None, label: Optional[str] = None)
```

Tries to generate a new candidate inside a ‘while True’ loop with a ‘try except’ statement which is broke when the new candidate is successfully generated. If an exception occurs during the candidate generation try, the function ‘catch’ will be called with such candidate as argument

**Parameters**

- **candidates** (`list[Chromosome]`) – Pool of candidates
- **parent\_index** (`int`) – Index of the current parent’s Chromosome in candidates
- **queue** (`mp.Queue`, *optional*) – Multiprocessing queue by which the candidates are returned in case of parallelism, defaults to None
- **child\_index** (`int`, *optional*) – Index of child in the next generation pool. It’s used to identify the returned chromosome in the main process in case of parallelism, defaults to None
- **label** (`str`, *optional*) – Child’s Chromosome’s label, defaults to None

**Returns** Child

**Return type** *Chromosome*

```
__get_improvement()
```

Generator of genetic improvements

**Yield** Best candidate achieved until the moment

**Return type** *Chromosome*

**\_\_get\_improvement\_mp()**

Genetic improvements generator using multiprocessing

**Raises Exception** – Raises an exception if elitism rate is incompatible with pool size

**Yield** Best candidate achieved until the moment

**Return type** *Chromosome*

**\_\_local\_optimization**(*candidate*: MolOpt.genetic.genetic.Chromosome) →  
*MolOpt.genetic.genetic.Chromosome*

Receives a candidate then performs a local optimization on its genes and so returns a new Chromosome with these new genes and its fitness

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

**Returns** Optimized candidate

**Return type** *Chromosome*

**static catch**(*candidate*: MolOpt.genetic.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*: MolOpt.genetic.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 found
- **timediff** (*float*) – Time difference between the candidate is found and the start of the execution

**abstract get\_fitness**(*candidate*: MolOpt.genetic.genetic.Chromosome) →  
*MolOpt.genetic.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()** → *MolOpt.genetic.genetic.Chromosome*

Returns a Chromosome with the genes and the fitness of the first\_parent

**Returns** Copy of first candidate

**Return type** *Chromosome*

**local\_optimize**(*candidate*: MolOpt.genetic.genetic.Chromosome) → MolOpt.genetic.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*: MolOpt.genetic.genetic.Chromosome) → MolOpt.genetic.genetic.Genes

The actual mutate\_best function. This is a 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*: MolOpt.genetic.genetic.Chromosome) → MolOpt.genetic.genetic.Genes

The actual mutate\_first function. This is a 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**() → MolOpt.genetic.genetic.Chromosome

Starts the genetic algorithm execution

**Returns** Best candidate

**Return type** Chromosome

**abstract save**(*candidate*: MolOpt.genetic.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 Mutate**(*methods*: list[collections.abc.Callable[[MolOpt.genetic.genetic.Chromosome], ~ Genes]],  
*methods\_rate*: list[typing.Union[int, float]])

Bases: object

A container for storing mutation functions and its rates. When it's 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

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

Bases: object

A container for storing 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



**mutate\_best**(*best\_candidate*: MolOpt.genetic.genetic.Chromosome) → MolOpt.genetic.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

**mutate\_first**(*first\_parent*: MolOpt.genetic.genetic.Chromosome) → MolOpt.genetic.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

### 1.1.2 molecular subpackage

**class 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

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.

**\_receive**(*molecule*) → None

Receives all data from another molecule making the actual molecule its copy

**Parameters** **molecule** (Molecule) – The data donor molecule

**copy**() → MolOpt.molecular.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, searches 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*) → [MolOpt.molecular.molecular.Molecule](#)

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

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 beginning'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 '---'. The order of the sections also must respect the order of the chosen structure. The second structure, that is the structure with literal numbers inside the geometry brackets is not compatible with `optg` function, so the first structure, that is the one with declared variables before the geometry is always recommended

#### 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 optimized with `optg`, false if it doesn't

**Raises** **Exception** – Invalid Z-matrix

**Returns** Loaded molecule

**Return type** *Molecule*

**mutate\_angles**(*times: int = 1*) → *MolOpt.molecular.molecular.Molecule*

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*) → *MolOpt.molecular.molecular.Molecule*

Selects a random distance parameter from the geometry and assign it a random value in the range gave by *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*) → *MolOpt.molecular.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 optimized version

**Return type** *Molecule*

**save**(*document: Optional[str] = None, directory: str = 'data'*) → None

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**() → *MolOpt.molecular.molecular.Molecule*

Provokes a swap mutation in itsef

**Returns** Itself mutated

**Return type** *Molecule*

**\_get\_swap\_indexes**(*new\_molecule: MolOpt.molecular.molecular.Molecule*) → *Tuple[int, int, int, int]*

Randomly choices the indexes of the parameters that will be swapped

**Parameters** *new\_molecule (Molecule)* – Copy of the original moecule

**Returns** A tuple with the four indexes needed to perform the swap

**Return type** *Tuple[int, int, int, int]*

**crossover\_1**(*parent: MolOpt.molecular.molecular.Molecule, donor: MolOpt.molecular.molecular.Molecule, label: Optional[str] = None*) → *MolOpt.molecular.molecular.Molecule*

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*

**crossover\_2**(*parent*: *MolOpt.molecular.molecular.Molecule*, *donor*: *MolOpt.molecular.molecular.Molecule*, *label*: *Optional[str] = None*) → *MolOpt.molecular.molecular.Molecule*

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 least 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*

**crossover\_n**(*parent*: *MolOpt.molecular.molecular.Molecule*, *donor*: *MolOpt.molecular.molecular.Molecule*, *label*: *Optional[str] = None*) → *MolOpt.molecular.molecular.Molecule*

Creates a copy of the parent molecule, randomly chooses an amount of parameters the minimum being one and the maximum being the total amount of parameters minus one. Then randomly chooses this amount of parameters from the donor molecule to replace the respective 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*

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

Creates a molecule's copy and randomly chooses an angle parameter and set its value to a random value between 0 and 360. The amount of mutations performed is equal to 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*

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

Creates a molecule's copy and randomly chooses a distance parameter and set its value in the range given by *rand\_range*. The amount of mutations performed is equal to *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*

**optg**(*molecule*: MolOpt.molecular.molecular.Molecule, *wanted\_energy*: str, *directory*: str = 'data', *nthreads*: int = 1, *keep\_output*: bool = False) → MolOpt.molecular.molecular.Molecule

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 *optimized\_molecule.output\_values* and updates all angles and distances parameters in [*optimized\_molecule.parameters*]. To use this function there cannot be any literal value in the molecule's geometry, all values must be as variable names in geometry and have their 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 address where the input and the output will be relative to *\_\_main\_\_*, defaults to 'data'
- **nthreads** (int, optional) – Number of threads used 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 it is read. defaults to False

**Returns** Optimized molecule

**Return type** *Molecule*

**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') → MolOpt.molecular.molecular.Molecule

Creates a geometry with all distances parameters randomized in the range given 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 parenthesis 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*

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

Creates a molecule’s copy and replace all distance parameters with random values in the range gave by *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*

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

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*

### 1.1.3 MolOpt.MolOpt module

**class** MolOpt(*first\_molecule*: MolOpt.molecular.molecular.Molecule, *energy\_param*: *str*, *strategies*:  
MolOpt.genetic.genetic.Strategies, *max\_age*: Optional[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: MolOpt.genetic.genetic.Genetic

Molecular geometry optimization class

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

**static catch**(*candidate*: MolOpt.genetic.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: MolOpt.genetic.genetic.Chromosome, donor: MolOpt.genetic.genetic.Chromosome) → MolOpt.molecular.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: MolOpt.genetic.genetic.Chromosome, donor: MolOpt.genetic.genetic.Chromosome) → MolOpt.molecular.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: MolOpt.genetic.genetic.Chromosome, donor: MolOpt.genetic.genetic.Chromosome) → MolOpt.molecular.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: MolOpt.genetic.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: MolOpt.genetic.genetic.Chromosome) → MolOpt.molecular.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: MolOpt.genetic.genetic.Chromosome) →  
*MolOpt.molecular.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: MolOpt.genetic.genetic.Chromosome) →  
*MolOpt.molecular.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: MolOpt.genetic.genetic.Chromosome) →  
*MolOpt.molecular.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: MolOpt.genetic.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: MolOpt.genetic.genetic.Chromosome) →  
*MolOpt.molecular.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*

## INDICES AND TABLES

- `genindex`
- `modindex`
- `search`



## PYTHON MODULE INDEX

### m

`MolOpt.genetic.genetic`, [1](#)

`MolOpt.molecular.molecular`, [5](#)

`MolOpt.MolOpt`, [12](#)



## Symbols

\_\_generate\_parent() (*Genetic method*), 2  
 \_\_get\_child() (*Genetic method*), 2  
 \_\_get\_improvement() (*Genetic method*), 2  
 \_\_get\_improvement\_mp() (*Genetic method*), 3  
 \_\_local\_optimization() (*Genetic method*), 3  
 \_get\_swap\_indexes() (in module  
     MolOpt.molecular.molecular), 9  
 \_receive() (*Molecule method*), 5

## C

catch() (*Genetic static method*), 3  
 catch() (*MolOpt static method*), 12  
 Chromosome (*class in MolOpt.genetic.genetic*), 1  
 copy() (*Molecule method*), 5  
 Create (*class in MolOpt.genetic.genetic*), 1  
 Crossover (*class in MolOpt.genetic.genetic*), 1  
 crossover\_1() (in module  
     MolOpt.molecular.molecular), 9  
 crossover\_1() (*MolOpt static method*), 12  
 crossover\_2() (in module  
     MolOpt.molecular.molecular), 10  
 crossover\_2() (*MolOpt static method*), 13  
 crossover\_n() (in module  
     MolOpt.molecular.molecular), 10  
 crossover\_n() (*MolOpt static method*), 13

## D

display() (*Genetic static method*), 3  
 dist\_unit (*Molecule property*), 5

## G

Genetic (*class in MolOpt.genetic.genetic*), 1  
 get\_fitness() (*Genetic method*), 3  
 get\_fitness() (*MolOpt method*), 13  
 get\_value() (*Molecule method*), 5

## L

load() (*Genetic method*), 3  
 load() (*Molecule static method*), 6  
 local\_optimize() (*Genetic method*), 3

local\_optimize() (*MolOpt method*), 13

## M

module  
     MolOpt.genetic.genetic, 1  
     MolOpt.molecular.molecular, 5  
     MolOpt.MolOpt, 12  
 Molecule (*class in MolOpt.molecular.molecular*), 5  
 MolOpt (*class in MolOpt.MolOpt*), 12  
 MolOpt.genetic.genetic  
     module, 1  
 MolOpt.molecular.molecular  
     module, 5  
 MolOpt.MolOpt  
     module, 12  
 Mutate (*class in MolOpt.genetic.genetic*), 4  
 mutate\_angles() (in module  
     MolOpt.molecular.molecular), 10  
 mutate\_angles() (*Molecule method*), 8  
 mutate\_angles() (*MolOpt static method*), 14  
 mutate\_best() (*Genetic method*), 4  
 mutate\_best() (in module MolOpt.genetic.genetic), 4  
 mutate\_distances() (in module  
     MolOpt.molecular.molecular), 11  
 mutate\_distances() (*Molecule method*), 8  
 mutate\_distances() (*MolOpt static method*), 14  
 mutate\_first() (*Genetic method*), 4  
 mutate\_first() (in module MolOpt.genetic.genetic), 5

## O

optg() (in module MolOpt.molecular.molecular), 11  
 optg() (*Molecule method*), 9

## R

random\_molecule() (in module  
     MolOpt.molecular.molecular), 11  
 randomize() (in module MolOpt.molecular.molecular),  
     12  
 randomize() (*MolOpt static method*), 14  
 run() (*Genetic method*), 4

**S**

`save()` (*Genetic method*), [4](#)  
`save()` (*Molecule method*), [9](#)  
`save()` (*MolOpt static method*), [14](#)  
`Strategies` (*class in MolOpt.genetic.genetic*), [4](#)  
`strategy_str` (*Chromosome property*), [1](#)  
`swap_mutate()` (in *MolOpt.molecular.molecular*), [12](#) *module*  
`swap_mutate()` (*Molecule method*), [9](#)  
`swap_mutate()` (*MolOpt static method*), [14](#)