

---

# **open-moldyn**

***Release 0.0.1***

**Jul 11, 2019**



---

## Contents:

---

<b>1</b>	<b>Presentation</b>	<b>1</b>
1.1	Simulation . . . . .	1
1.2	Processing . . . . .	8
1.3	Utils . . . . .	11
<b>2</b>	<b>Indices and tables</b>	<b>17</b>
<b>3</b>	<b>Credits</b>	<b>19</b>
	<b>Python Module Index</b>	<b>21</b>
	<b>Index</b>	<b>23</b>



Open-moldyn is the result of an internship project at the IPR (*Institut de Physique de Rennes*) aiming at creating an (hopefully) fast and easy to use 2D molecular dynamics simulation tool with python.

This tool features a GUI (graphical interface) and is meant to be installed with pip (as every respectable python packages should be).

**Some links bellow:**

- Github repository : <https://pypi.org/project/open-moldyn/>
- PyPI depot : <https://github.com/open-moldyn/moldyn>

## 1.1 Simulation

### 1.1.1 Model builder

Model builder. Stores and defines physical properties of a group of atoms.

The model handles two species of atom (one of them can be ignored by setting the correct mole fraction). The first species values are stored in the first part of arrays (lower indices), the second species in what lasts (higher indices). This is meant to facilitate computation of inter-atomic forces and potential energy.

```
class moldyn.simulation.builder.Model (pos=None, v=None, npart=0, x_a=1.0)
```

**Parameters**

- **pos** (*np.array*) – Atoms' position.

---

**Note:** If *v* is not set, it is initialized as an array full of zeros.

---

- **v** (*np.array*) – Atoms' speed.

---

**Note:** Taken in account only if *pos* is set.

---

- **npart** (*int*) – Total number of atoms.

---

**Note:** Setting *pos* overrides this.

---

- **x\_a** (*float*) – Mole fraction of species A.

**T**

Temperature. Is calculated from the average kinetic energy of the atoms. May be set to any positive value, in which case atoms' speed will be scaled to match the desired temperature.

**Type** float

**EC**

Microscopic kinetic energy.

---

**Note:** Cannot be changed as-is, but setting *T* is one way to do so.

---

**Type** float

**total\_EC**

Total kinetic energy.

---

**Note:** Cannot be changed as-is.

---

**Type** float

**kB**

Boltzmann constant. If changed, will affect the way the model behaves regarding temperature.

**Type** float

**pos**

**v**

List of atom positions and speeds. First axis represents the atom, second axis the dimension (x or y).

**Type** np.array

**dt**

Timestep used for simulation.

---

**Note:** An acceptable value is calculated when species are defined, but it may be set to anything else.

---

**Type** float

**npart**

Total number of atoms.

**Type** int

**x\_a**

Mole fraction of species A.

**Type** float**n\_a**Atom number for species A, calculated from *x\_a* and *npart*. If set, *x\_a* will be recalculated.**Type** int**epsilon\_a****epsilon\_b**

Epsilon value (J, in Lennard-Jones potential) for species a or b.

**Type** float**sigma\_a****sigma\_b**

Sigma value (m, in Lennard-Jones potential) for species a or b.

**Type** float**epsilon\_ab****sigma\_ab**

Inter-species epsilon and sigma values.

---

**Note:** Cannot be changed as-is. If you want to change these values, modify the corresponding items in the *params* dictionary.

---

**Type** float**re\_a****re\_b****re\_ab**

Estimated radius of minimum potential energy.

**Type** float**rcut\_fact**When the model is simulated, atoms further than *rcut\_fact\*re* do not interact. Defaults to 2.0.**Type** float**params**

Model parameters, needed for the simulation.

<p><b>Warning:</b> Changing directly these values may lead to unpredicted behaviour if not documented.</p>
--

**Type** dict**kong**

Kong rules to estimate inter-species sigma and epsilon parameters.

**Type** dict

**inter\_species\_rule**

Rules to automatically estimate inter-species sigma and epsilon parameters. Defaults to *kong*.

**Type** dict

**x\_lim\_inf****y\_lim\_inf**

Lower x and y position of the box boundaries.

**Type** float

**x\_lim\_sup****y\_lim\_sup**

Upper x and y position of the box boundaries.

**Type** float

**length\_x****length\_y**

Size of the box along x and y axis. Those parameter are tied with *\*\_lim\_\** and any change on one of them is correctly taken in account.

**Type** float

**lim\_inf****lim\_sup****length**

2 elements wide array containing corresponding (*x\_\**, *y\_\**) values.

---

**Note:** Cannot be changed as-is.

---

**Type** np.array

**x\_periodic****y\_periodic**

Defines periodic conditions for x and y axis. Set to 1 to define periodic boundary condition or 0 to live in an infinite empty space. Defaults to 0.

**Type** int

**mass**

Total mass in the model.

---

**Note:** Cannot be changed as-is.

---

**Type** float

**m**

Mass of each atom. Shape is (*npart*, 2) in order to facilitate calculations of kinetic energy and Newton's second law.



**Warning:** You should not change those values unless you know what you are doing.

**Type** np.array

**`_m()`**

Constructs *m*.

**Returns** *m*

**Return type** np.array

**`atom_grid(n_x, n_y, d)`**

Creates a grid containing  $n_x \times n_y$  atoms.

Sets *npart*, *pos*, *dt*, *v* and *m*.

**Parameters**

- **`n_x(int)`** – number of columns
- **`n_y(int)`** – number of rows
- **`d(float)`** – inter-atomic distance

**`copy()`**

**Returns** Copy of the current model

**Return type** *builder.Model*

**`random_speed()`**

Gives a random speed to the atoms, following a normal law, in order to have a strictly positive temperature.

**`set_a(epsilon, sigma, m)`**

Sets species a parameters.

**Parameters**

- **`epsilon(float)`** – Epsilon in Lennard-Jones potential.
- **`sigma(float)`** – Sigma in Lennard-Jones potential.
- **`m(float)`** – Mass of the atom.

**`set_ab(a, b)`**

Sets species a and b parameters, and calculates inter-species parameters.

**Parameters**

- **`a(tuple)`** – First species parameters, under the form (*epsilon*, *sigma*, *mass*)
- **`b(tuple)`** – Second species parameters, under the same form.

**`set_b(epsilon, sigma, m)`**

Same as *set\_a*.

**`set_dt(dt=None)`**

Defines the timestep used for simulation.

**Parameters** **`dt(float)`** – Desired timestep. If not set, will be calculated from species a's properties.

**`set_periodic_boundary(x=1, y=1)`**

Set periodic boundaries on both axis.

**Parameters**

- **x**(*int*) – see *x\_periodic*
- **y**(*int*) – see *y\_periodic*

**shuffle\_atoms()**

Shuffle atoms' position in order to easily create a homogeneous repartition of the two species. Should be called just right after the positions are defined.

---

**Note:** Atoms' speed is not shuffled.

---

## 1.1.2 Simulator

Simulator. Simulates the dynamics of a model (on the CPU or GPU).

```
class moldyn.simulation.runner.Simulation (model=None, simulation=None, prefer_gpu=True)
```

Simulator for a model.

**Parameters**

- **model** (*builder.Model*) – Model to simulate. The original model object is copied, and thus preserved, which allows it to serve as a reference.
- **simulation** (*Simulation*) – Simulation to copy. The simulation is copied, and the computing module reinitialized.
- **prefer\_gpu** (*bool*) – Specifies if GPU should be used to compute inter-atomic forces. Defaults to *True*, as it generally results in a significant speed gain.

**model**

Model that is simulated.

**Warning:** Works as usual, but changing its properties will not affect the simulation as intended. You should construct another simulation from this model to correctly take in account the changes. This is due to the fact that some values (eg. Lennard-Jones parameters) are treated as constants during shader initialisation, in order to speed up the calculations.

**Type** *builder.Model*

**current\_iter**

Number of iterations already computed, since initialisation.

**Type** *int*

**context**

ModernGL context used to build and run compute shader.

**Type** *moderngl.Context*

**F**

Last computed forces applied to atoms. Initialized to zeros.

**Warning:** Changing the values will affect behavior of the model.

**Type** numpy.ndarray

**$\mathbf{F}_f(t)$**

**Parameters**  $t$  – Time in seconds.

**Returns** External forces applied at time  $t$  (2-component vector).

**Return type** numpy.ndarray

**`iter`** ( $n=1$ ,  $callback=None$ )

Iterates one or more simulation steps.

Basically, follows the Position-Verlet method to update positions and speeds, and computes inter-atomic forces derived from Lennard-Jones potential.

**Parameters**

- **`n`** (*int*) – Number of iterations to perform.
- **`callback`** (*callable*) – A callback function that must take the Simulation object as first argument. It is called at the end of each iteration.

**Note:** Setting `n` is significantly faster than calling `iter()` several times.

## Example

```
model.iter(5)
```

**`set_Fx_ramps`** ( $t, Fx$ )

Creates a function based on ramps and uses it for external forces control along axis x. See `set_T_ramps` for further details.

**Parameters**

- **`t`** (*array*) – Time.
- **`Fx`** (*array*) – Associated forces.

**`set_Fy_ramps`** ( $t, Fy$ )

Creates a function based on ramps and uses it for external forces control along axis y. See `set_T_ramps` for further details.

**Parameters**

- **`t`** (*array*) – Time.
- **`Fy`** (*array*) – Associated forces.

**`set_T_f`** ( $f$ )

Sets function that controls temperature.

**Parameters** **`f`** (*callable*) – Must take time (float) as an argument and return temperature (in K, float).

**set\_T\_ramps** (*t*, *T*)

Creates a function based on ramps and uses it for temperature control. Values of the function are interpolated between points given in *t* and *T*. Temperature is supposed constant before the first point and after the last one.

**Parameters**

- **t** (*array*) – Time.
- **T** (*array*) – Associated temperatures.

**Inter-atomic compute modules**

**class** moldyn.simulation.forces\_CPU.**ForcesComputeCPU** (*consts*, *compute\_npart=None*,  
*compute\_offset=0*)

Compute module. Runs on CPU. Uses *numba* for JIT compilation and *multiprocessing* for multicore. See *ForcesComputeGPU* for documentation.

**class** moldyn.simulation.forces\_GPU.**ForcesComputeGPU** (*consts*, *compute\_npart=None*)

Compute module. Runs on GPU.

**Parameters** **consts** (*dict*) – Dictionary containing constants used for calculations.

**npart**

Number of atoms.

**Type** int

**consts**

Dictionary containing constants used for calculations, and some parameters to run the compute shader.

**Type** dict

**get\_COUNT** ()

**Returns** Average near atoms (one could count this as bonds).

**Return type** np.ndarray

**get\_F** ()

**Returns** Computed inter-atomic forces.

**Return type** np.ndarray

**get\_PE** ()

**Returns** Computed potential energy.

**Return type** np.ndarray

**set\_pos** (*pos*)

Set position array and start computing forces.

**Parameters** **pos** (*np.ndarray*) – Array of positions.

## 1.2 Processing

### 1.2.1 Data processing

moldyn.processing.data\_proc.**PDF** (*pos*, *nb\_samples*, *rcut*, *bin\_count*)

Pair Distribution Function. Returns normalized histogram of distance between atoms.

**Parameters**

- **pos** (*np.array*) – Array containing atoms position
- **nb\_samples** (*int*) – Number of atoms from which to generate the histogram
- **rcut** (*number*) – Maximum distance to consider
- **bin\_count** (*int*) – Number of bins of the histogram

**Returns** **bins, hist** – *bins* being the distances, *hist* the normalized (regarding radius) histogram

**Return type** tuple(np.array, np.array)

```
moldyn.processing.data_proc.cached (f, _cache={<function PDF>:
<sphinx.ext.autodoc.importer._MockObject
object>, <function density>:
<sphinx.ext.autodoc.importer._MockObject
object>, <function compute_strain>:
<sphinx.ext.autodoc.importer._MockObject object>})
```

**Parameters f –**

```
moldyn.processing.data_proc.compute_strain (model0, model1, rcut)
```

Compute the local deformation tensor for each atom.

It will try to use GPU but will fallback on CPU if not available

**Parameters**

- **model0** (*simulation.builder.Model*) – The model at time t
- **model1** (*simulation.builder.Model*) – The model at time t-dt
- **rcut** (*float*) –

**Returns**

- A vector containing the 2D deformation tensor of each atom
- (in the order of *model.pos*).

---

**Note:** Due to numerical calculation imprecision the deformation tensor may not be quantitatively accurate (or even symmetrical).

---

```
moldyn.processing.data_proc.density (model, refinement=0)
```

Create a Voronoi mesh and calculate the local particle density on its vertices.

The local density is calculated as follows: for each vertex, compute the density of each neighbour region as one over the area and assign the average of the neighbouring density to the vertex.

**Parameters**

- **model** (*simulation.builder.Model*) – the Model object containing
- **refinement** (*int (defaults : 0)*) – number of subdivision for refining the mesh (0 == None)

**Returns**

- **tri** (*matplotlib.tri.Triangulation*) – the triangulation mesh (refined if set as)
- **vert\_density** (*numpy.array*) – the array containing the local density associated with the tri mesh

### Example

To plot the result using matplotlib use :

```
import matplotlib.pyplot as plt
tri, density = data_proc.density(model)
plt.tricontour(tri, density) # to draw contours
plt.tricontourf(tri, density) # ot draw filled contours
plt.show()
```

---

**Note:** As of now, the numerical results may not be quantitatively accurate but should qualitatively represent the density.

---

## 1.2.2 Visualization

`moldyn.processing.visualisation.make_movie` (*simulation*, *dynstate*, *name*, *pfilm=5*, *fps=24*,  
*callback=None*)

Makes a .mp4 movie of simulation from the history saved in dynstate.

#### Parameters

- **simulation** (*Simulation*) – The simulation object
- **dynstate** (*DynState*) – The dynState object containing the position history file.
- **name** (*str*) – The path (and name) of the file to be written.
- **pfilm** (*int*) – To make the movie, takes every pfilm position.
- **fps** (*int*) – The number of frame per seconds of the film.
- **callback** (*function*) – An optional callback function that is called every time a frame is made and is passed the current iteration number.

`moldyn.processing.visualisation.plot_density` (*model*, *levels=None*, *refinement=0*)

Compute and plot the contours of the local density of model.

#### Parameters

- **model** (*Model*) – The model to plot
- **levels** (*int or array-like*) – the number of levels or a sorted array-like object of levels
- **refinement** (*int*) – the level of refinement

`moldyn.processing.visualisation.plot_density_surf` (*model*, *refinement=0*)

Compute and plot the 3D surface of the local density of model.

#### Parameters

- **model** (*Model*) – The model to plot.
- **refinement** (*int*) – the level of refinement

`moldyn.processing.visualisation.plot_densityf` (*model*, *levels=None*, *refinement=0*)

Compute and plot the filled contours of the local density of model.

#### Parameters

- **model** (*Model*) – The model to plot

- **levels** (*int* or *array-like*) – the number of levels or a sorted array-like object of levels
- **refinement** (*int*) – the level of refinement

`moldyn.processing.visualisation.plot_particles` (*model*)  
Plot the position stored in model.

**Parameters** `model` (`Model`) – The model to plot

## 1.3 Utils

Utility module

### 1.3.1 Data management

**class** `moldyn.utils.data_mng.DynState` (*treant*, \*, *extraction\_path*='/home/docs/.local/share/open-moldyn/tmp')

A Treant specialized for handling .npz and .json files for moldyn

**POS**

standard name of the position file ("pos.npz")

**Type** str

**POS\_H**

standard name of the position history file ("pos\_history.npz")

**Type** str

**VEL**

standard name of the velocity file ("velocities.npz")

**Type** str

**STATE\_FCT**

standard name of the state function file ("state\_fct.json")

**Type** str

**PAR**

standard name of the parameter file ("parameters.json")

**Type** str

**open** (*file*, *mode*='r')

Open the file in this tree (ie. directory and subdir). Return the appropriate IO class depending of the type of the file.

If the type is not recognize, it opens the file and return the BytesIO or StringIO object.

**Parameters**

- **file** (*str*) – The name of the file. This file must be in this tree (ie. directory and subdir).
- **mode** (*str* (*default*='r')) –

**The mode with which to open the file :**

- 'r' to read
- 'w' to write

- 'a' to append
- 'b' to read or write bytes (eg. 'w+b' to write bytes)

**Returns**

- If file is a .npy file, return a NumpyIO object.
- If file is a .json file, return a ParamIO object.
- Else, return a StringIO or a BytesIO depending on mode.

---

**Note:** This method is designed to be used with a context manager like this

```
t = DynState(dirpath)
# here t.open returns a NumpyIO object as the file is .npy
with t.open("pos.npy", 'r') as IO:
    arr = IO.load() #load an array
with t.open("pos.npy", 'w') as IO:
    IO.save(arr) #save an array
```

---

**save\_model (model)**

Save the positions, the velocities and the parameters of the model.

The position and velocity arrays are saved as numpy files and the parameter dictionary as a .json file

**Parameters** **model** (`simulation.builder.Model`) – The model to be saved.

**to\_zip (path)**

Zip every leaf (aka. file) of the dynState treant into an archive at path.

**Parameters** **path** (`str`) – The path of the archive.

**class** moldyn.utils.data\_mng.**NumpyIO** (`dynState, file, mode`)

An interface to interact with numpy save files with a context manager.

**dynState**

The treant that support the leaf (file) associated with it.

**Type** `datreant.Treant`

**mode**

The access mode of the file (usually 'r' or 'w' or 'a')

**Type** `str`

**file\_name**

The file name of the .npy file associated with it.

**Type** `datreant.Leaf`

**file**

the file that is opened (contains None until entering a context manager)

**Type** `file object`

**Example**



```

t = DynState(dirpath)
# here t.open returns a NumpyIO object as the file is .npy
with t.open("pos.npy", 'r') as IO:
    arr = IO.load() #load an array
with t.open("pos.npy", 'w') as IO:
    IO.save(arr) #save an array

```

**close()**

Close the internal file.

**load()**

Load an array stored in the file.

**Returns** **arr** – the array loaded

**Return type** ndarray

**open()**

Open the internal file.

**save(arr)**

Save an array to the opened file.

**Parameters** **arr** (ndarray (numpy)) – The array to be stored

**class** moldyn.utils.data\_mng.**ParamIO** (dynState, file, \*\*kwargs)

An interface to interact with json files as a dictionary.

Designed to be used with context manager (the with statement) and datreant. Upon entering the with statement it will attempt to load the json file at self.file\_name into itself (as a dict). When leaving this context, it will store itself into the json file.

Inherits dict.

It will try to update the tags and categories of the treant dynState.

**dynState**

The treant that support the leaf (file) associated with it.

**Type** datreant.Treant

**file\_name**

The file name of the json file associated with it.

**Type** datreant.Leaf

## Example

```

t = DynState(dirpath)
# here t.open returns a ParamIO object as the file is .json
with t.open("params.json") as IO:
    IO["my_key"] = "my_value"
    random_var = IO[some_key]
# upon exiting the with block, the content of IO is stored
# in the .json file

```

### Parameters

- **dynState** (Treant) –
- **file** (Leaf) –

- **kwargs** –

**\_\_enter\_\_** ()

Try to load the content of the json file into itself

Try to open the json file from file\_name and load the informations it contains into itself. Will catch FileNotFoundError and JSONDecodeError

**Returns** self

**Return type** ParamIO

**\_\_exit\_\_** (exc\_type, exc\_val, exc\_tb)

Open the json file file\_name and dump itself into it.

Open the json file file\_name and dump itself into it + update the tags and categories of dynState according to the CATEGORY\_LIST of the module.

**Parameters**

- **exc\_type** –

- **exc\_val** –

- **exc\_tb** –

**from\_dict** (rdict)

Copy rdict into itself

**Parameters** **rdict** (dict) – The remote dictionary from which to copy

**to\_attr** (obj)

Dump the parameter dictionary in the object (obj) as attributes of said object.

**Warning:** Will change the value of each attribute of obj that have a name corresponding to a key of ParamIO.

**Parameters** **obj** (an object) – The object to which it will dump its content as attributes

**to\_dict** (rdict)

Copy itself into the remote dictionary :type rdict: dict :param rdict: The remot dictionary to which to copy :type rdict: dict

### 1.3.2 OpenGL utility tools

`moldyn.utils.gl_util.source (uri, consts={})`

Reads and replaces constants (in all caps) in a text file.

**Parameters**

- **uri** (str) – URI of the file to read.
- **consts** (dict) – Dictionary containing the values to replace.

**Returns** File contents with replacements.

**Return type** str

`moldyn.utils.gl_util.testGL ()`

Tries to initialize a compute shader.

**Returns** Initialisation success.

**Return type** bool



## CHAPTER 2

---

### Indices and tables

---

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



## CHAPTER 3

---

### Credits

---

Open-moldyn is made by Alexandre Faye-Bedrin and Arthur Luciani (at the time students of the *ENS Rennes* and interns of the *IPR*) with the supervision of Yann Gueguen.

This package also includes [datreant](#) and [appdirs](#) to help managing data and files.





### m

- `moldyn.processing.data_proc`, [8](#)
- `moldyn.processing.visualisation`, [10](#)
- `moldyn.simulation.builder`, [1](#)
- `moldyn.simulation.forces_CPU`, [8](#)
- `moldyn.simulation.forces_GPU`, [8](#)
- `moldyn.simulation.runner`, [6](#)
- `moldyn.utils.data_mng`, [11](#)
- `moldyn.utils.gl_util`, [14](#)



## Symbols

`__enter__()` (*moldyn.utils.data\_mng.ParamIO method*), 14  
`__exit__()` (*moldyn.utils.data\_mng.ParamIO method*), 14  
`_m()` (*moldyn.simulation.builder.Model method*), 5

## A

`atom_grid()` (*moldyn.simulation.builder.Model method*), 5

## C

`cached()` (*in module moldyn.processing.data\_proc*), 9  
`close()` (*moldyn.utils.data\_mng.NumpyIO method*), 13  
`compute_strain()` (*in module moldyn.processing.data\_proc*), 9  
`consts` (*moldyn.simulation.forces\_GPU.ForcesComputeGPU attribute*), 8  
`context` (*moldyn.simulation.runner.Simulation attribute*), 6  
`copy()` (*moldyn.simulation.builder.Model method*), 5  
`current_iter` (*moldyn.simulation.runner.Simulation attribute*), 6

## D

`density()` (*in module moldyn.processing.data\_proc*), 9  
`dt` (*moldyn.simulation.builder.Model attribute*), 2  
`DynState` (*class in moldyn.utils.data\_mng*), 11  
`dynState` (*moldyn.utils.data\_mng.NumpyIO attribute*), 12  
`dynState` (*moldyn.utils.data\_mng.ParamIO attribute*), 13

## E

`EC` (*moldyn.simulation.builder.Model attribute*), 2  
`epsilon_a` (*moldyn.simulation.builder.Model attribute*), 3

`epsilon_ab` (*moldyn.simulation.builder.Model attribute*), 3  
`epsilon_b` (*moldyn.simulation.builder.Model attribute*), 3

## F

`F` (*moldyn.simulation.runner.Simulation attribute*), 6  
`F_f()` (*moldyn.simulation.runner.Simulation method*), 7  
`file` (*moldyn.utils.data\_mng.NumpyIO attribute*), 12  
`file_name` (*moldyn.utils.data\_mng.NumpyIO attribute*), 12  
`file_name` (*moldyn.utils.data\_mng.ParamIO attribute*), 13  
`ForcesComputeCPU` (*class in moldyn.simulation.forces\_CPU*), 8  
`ForcesComputeGPU` (*class in moldyn.simulation.forces\_GPU*), 8  
`from_dict()` (*moldyn.utils.data\_mng.ParamIO method*), 14

## G

`get_COUNT()` (*moldyn.simulation.forces\_GPU.ForcesComputeGPU method*), 8  
`get_F()` (*moldyn.simulation.forces\_GPU.ForcesComputeGPU method*), 8  
`get_PE()` (*moldyn.simulation.forces\_GPU.ForcesComputeGPU method*), 8

## I

`inter_species_rule` (*moldyn.simulation.builder.Model attribute*), 3  
`iter()` (*moldyn.simulation.runner.Simulation method*), 7

## K

`kB` (*moldyn.simulation.builder.Model attribute*), 2  
`kong` (*moldyn.simulation.builder.Model attribute*), 3

## L

length (*moldyn.simulation.builder.Model* attribute), 4  
 length\_x (*moldyn.simulation.builder.Model* attribute), 4  
 length\_y (*moldyn.simulation.builder.Model* attribute), 4  
 lim\_inf (*moldyn.simulation.builder.Model* attribute), 4  
 lim\_sup (*moldyn.simulation.builder.Model* attribute), 4  
 load () (*moldyn.utils.data\_mng.NumpyIO* method), 13

## M

m (*moldyn.simulation.builder.Model* attribute), 4  
 make\_movie () (in module *moldyn.processing.visualisation*), 10  
 mass (*moldyn.simulation.builder.Model* attribute), 4  
 mode (*moldyn.utils.data\_mng.NumpyIO* attribute), 12  
 Model (class in *moldyn.simulation.builder*), 1  
 model (*moldyn.simulation.runner.Simulation* attribute), 6  
 moldyn.processing.data\_proc (module), 8  
 moldyn.processing.visualisation (module), 10  
 moldyn.simulation.builder (module), 1  
 moldyn.simulation.forces\_CPU (module), 8  
 moldyn.simulation.forces\_GPU (module), 8  
 moldyn.simulation.runner (module), 6  
 moldyn.utils.data\_mng (module), 11  
 moldyn.utils.gl\_util (module), 14

## N

n\_a (*moldyn.simulation.builder.Model* attribute), 3  
 npart (*moldyn.simulation.builder.Model* attribute), 2  
 npart (*moldyn.simulation.forces\_GPU.ForcesComputeGPU* attribute), 8  
 NumpyIO (class in *moldyn.utils.data\_mng*), 12

## O

open () (*moldyn.utils.data\_mng.DynState* method), 11  
 open () (*moldyn.utils.data\_mng.NumpyIO* method), 13

## P

PAR (*moldyn.utils.data\_mng.DynState* attribute), 11  
 ParamIO (class in *moldyn.utils.data\_mng*), 13  
 params (*moldyn.simulation.builder.Model* attribute), 3  
 PDF () (in module *moldyn.processing.data\_proc*), 8  
 plot\_density () (in module *moldyn.processing.visualisation*), 10  
 plot\_density\_surf () (in module *moldyn.processing.visualisation*), 10  
 plot\_densityf () (in module *moldyn.processing.visualisation*), 10  
 plot\_particles () (in module *moldyn.processing.visualisation*), 11

pos (*moldyn.simulation.builder.Model* attribute), 2  
 POS (*moldyn.utils.data\_mng.DynState* attribute), 11  
 POS\_H (*moldyn.utils.data\_mng.DynState* attribute), 11

## R

random\_speed () (*moldyn.simulation.builder.Model* method), 5  
 rcut\_fact (*moldyn.simulation.builder.Model* attribute), 3  
 re\_a (*moldyn.simulation.builder.Model* attribute), 3  
 re\_ab (*moldyn.simulation.builder.Model* attribute), 3  
 re\_b (*moldyn.simulation.builder.Model* attribute), 3

## S

save () (*moldyn.utils.data\_mng.NumpyIO* method), 13  
 save\_model () (*moldyn.utils.data\_mng.DynState* method), 12  
 set\_a () (*moldyn.simulation.builder.Model* method), 5  
 set\_ab () (*moldyn.simulation.builder.Model* method), 5  
 set\_b () (*moldyn.simulation.builder.Model* method), 5  
 set\_dt () (*moldyn.simulation.builder.Model* method), 5  
 set\_Fx\_ramps () (*moldyn.simulation.runner.Simulation* method), 7  
 set\_Fy\_ramps () (*moldyn.simulation.runner.Simulation* method), 7  
 set\_periodic\_boundary () (*moldyn.simulation.builder.Model* method), 5  
 set\_pos () (*moldyn.simulation.forces\_GPU.ForcesComputeGPU* method), 8  
 set\_T\_f () (*moldyn.simulation.runner.Simulation* method), 7  
 set\_T\_ramps () (*moldyn.simulation.runner.Simulation* method), 7  
 shuffle\_atoms () (*moldyn.simulation.builder.Model* method), 6  
 sigma\_a (*moldyn.simulation.builder.Model* attribute), 3  
 sigma\_ab (*moldyn.simulation.builder.Model* attribute), 3  
 sigma\_b (*moldyn.simulation.builder.Model* attribute), 3  
 Simulation (class in *moldyn.simulation.runner*), 6  
 source () (in module *moldyn.utils.gl\_util*), 14  
 STATE\_FCT (*moldyn.utils.data\_mng.DynState* attribute), 11

T

T (*moldyn.simulation.builder.Model* attribute), 2  
 testGL () (in module *moldyn.utils.gl\_util*), 14  
 to\_attr () (*moldyn.utils.data\_mng.ParamIO* method), 14

to\_dict() (*moldyn.utils.data\_mng.ParamIO method*),  
 14  
 to\_zip() (*moldyn.utils.data\_mng.DynState method*),  
 12  
 total\_EC (*moldyn.simulation.builder.Model attribute*),  
 2

## V

v (*moldyn.simulation.builder.Model attribute*), 2  
 VEL (*moldyn.utils.data\_mng.DynState attribute*), 11

## X

x\_a (*moldyn.simulation.builder.Model attribute*), 2  
 x\_lim\_inf (*moldyn.simulation.builder.Model attribute*), 4  
 x\_lim\_sup (*moldyn.simulation.builder.Model attribute*), 4  
 x\_periodic (*moldyn.simulation.builder.Model attribute*), 4

## Y

y\_lim\_inf (*moldyn.simulation.builder.Model attribute*), 4  
 y\_lim\_sup (*moldyn.simulation.builder.Model attribute*), 4  
 y\_periodic (*moldyn.simulation.builder.Model attribute*), 4