# Distance calculations in MDAnalysis

# Accessing positions

Atom coordinates are in the "**`.positions`**" attribute of an "**`AtomGroup`**"

The positions are returned as a NumPy array, which we can then readily manipulate.

built-in functions based on position data:

**`center_of_mass()`**

**`center_of_geometry()`**

```
pos = u.atoms.positions

print(pos)

[[ 11.736044    8.500797  -10.445281 ]
 [ 12.365119    7.839936  -10.834842 ]
 ...
 [  6.300186   19.363485   -7.935916 ]
 [  5.5854015  17.589624   -6.9656615]]
```

# The lib.distances module

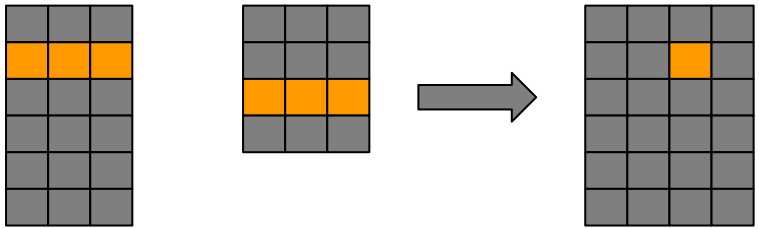
Particle positions are given as numpy arrays, so most work can be done using numpy (and numpy derived) libraries.

One import exception to this is distance calculations, where:

- periodic boundaries must be considered
- domain specific algorithms can be used

# distance_array

To calculate all pairwise distances between two arrays of coordinates.



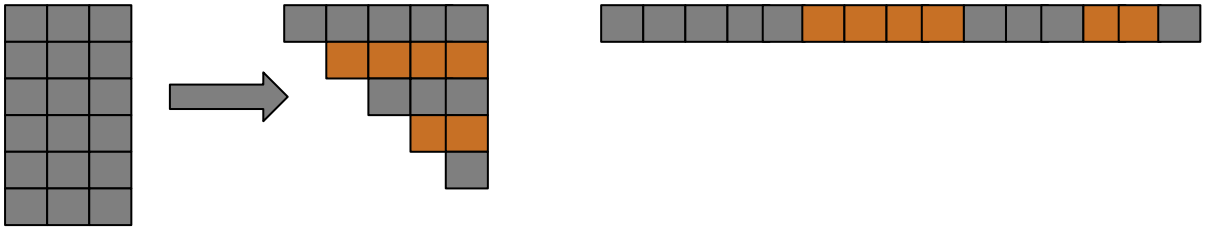
```
from MDAnalysis.lib import distances
import MDAnalysis as mda

u = mda.Universe(...)
ag1 = u.select_atoms(...)
ag2 = u.select_atoms(...)

reference = ag1.positions
configuration = ag2.positions

distances.distance_array(
      reference, configuration,
      box=None)
```

# self_distance_array

For calculating distances between all combinations of coordinates.

Takes a single array of coordinates and calculates all pairwise distances ( ½ n(n-1) results).

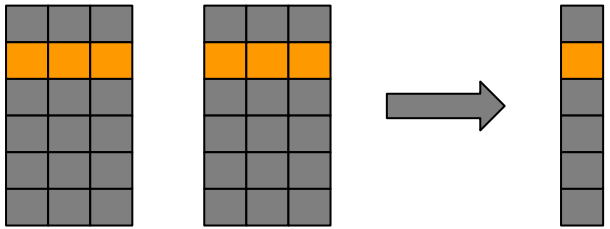
```
from MDAnalysis.lib import distances

distances.self_distance_array(
     reference,
     box=None, result=None)
```



# calc_bonds

For calculating a series of distances between points.

Takes two arrays of coordinates, of equal length, and returns the distances between coordinates in each row.

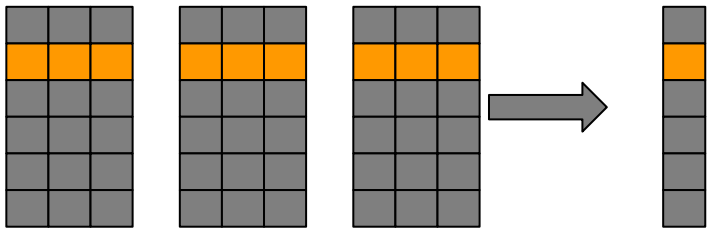


```
from MDAnalysis.lib import
distances

distances.calc_bonds(
    coords1, coords2,
    box=None)
```

# calc_angles & calc_dihedrals

For calculating either the angle or dihedral angle between 3 or 4 arrays of coordinates.

takes 3 or 4 arrays of identical length coordinates.



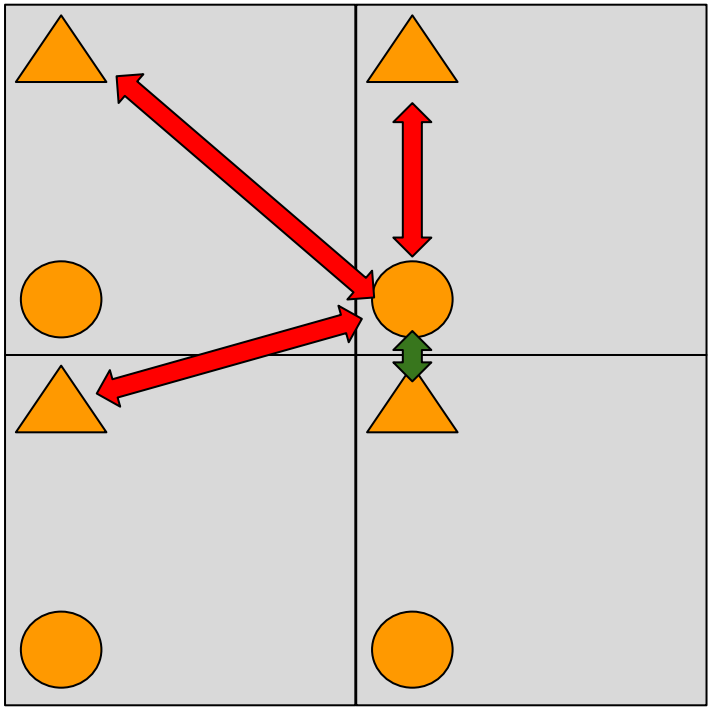
```
from MDAnalysis.lib import distances

distances.calc_angles(
    coords1, coords2, coords3,
    box=None, result=None)
```

```
distances.calc_dihedrals(
    coords1, coords2,
    coords3, coords4,
    box=None, result=None)
```

# Minimum image convention

For molecular simulation coordinates it is important that we consider periodic boundary conditions and that the smallest (correct) distance is considered.

This can be achieved by supplying the box information as “**`box=ag.dimensions`**” to any distance function.

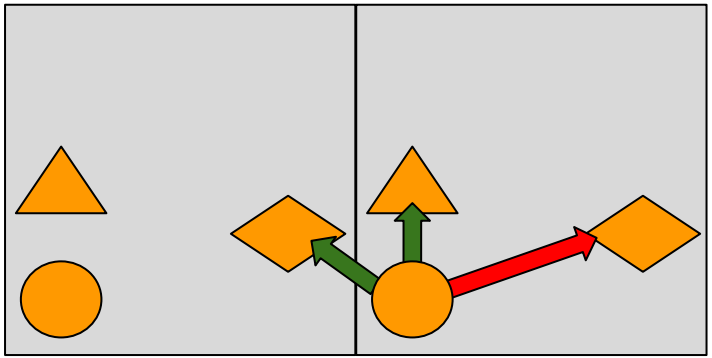


# Minimum image convention for angles

For angle calculations, minimum image convention is still important!

The vectors between coordinates must obey minimum image convention.

Again, pass "**`box=ag.dimensions`**" to the function and this will be taken care of for you.



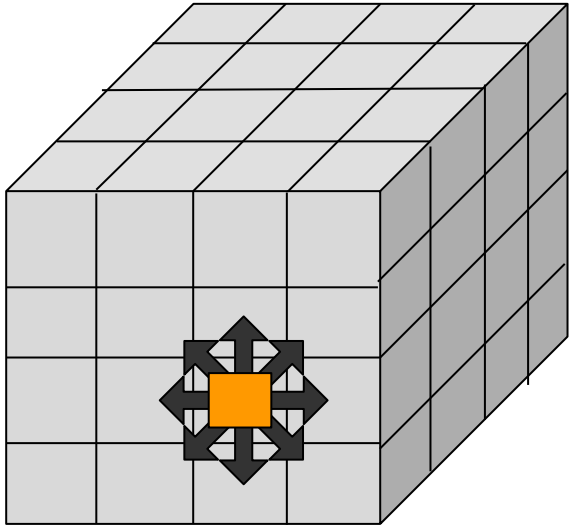
# Capped distances

# The problem: Quadratic scaling!

Calculating all pairwise distances between two sets of coordinates will scale badly for large systems.

Interactions between particles are quite short ranged and we're often only interested in relatively short distances.

We can therefore perform a search for all pairwise distances which are below a given limit - capped distance search.



# capped_distance & capped_self_distance

Only find distances up to a limit.

it returns:

- an array of indices
- an array of distances.

```
from MDAnalysis.lib import distances

distances.capped_distance(
    reference, configuration,
max_cutoff,
    box=None)

distances.self_capped_distance(
    reference, max_cutoff,
    box=None)
```

# Capped distance example

```
idx, dists = capped_distance(group1.positions,
                             group2.positions,
                             max_cutoff=3.0,
                             box=u.dimensions)

idx[:5], dists[:5]
(array([[    0, 13456],
        [    0, 19217],
        [    0, 19216],
        [    0,  8949],
        [    0, 14295]]),
 array([2.94881834, 2.91110877, 1.58488013, 2.24118461,
1.79654237]))
```

# Distance calculation recap

Calculating pairwise distances:

- `calc_bonds`
- `distance_array`
- `self_distance_array`

Faster, sparse pairwise distances:

- `capped_distance`
- `self_capped_distance`

Calculating angles:

- `calc_angles`
- `calc_dihedrals`

# But what about frames?

Trajectory files are read by MDAnalysis, but only one frame of a trajectory is ever loaded at a given time.

The currently loaded frame can be controlled by indexing the `.trajectory` attribute of a **Universe**.

```
print(u.atoms.positions[0])
[ 11.736044,   8.500797, -10.445281]

u.trajectory[1]
print(u.atoms.positions[0])
[ 11.505546,   8.062977, -10.38611 ]

u.trajectory[0]
print(u.atoms.positions[0])
[ 11.736044,   8.500797, -10.445281]
```

# All the frames!

To iterate over an entire trajectory, simply loop over the "**`.trajectory`**" attribute inside a for loop.

Warning!

Loading a new frame will discard any changes made to the trajectory.

```
for ts in u.trajectory[:5]:
    print(u.atoms[0].position)
[ 11.736044   8.500797 -10.445281]
[ 11.505546   8.062977 -10.38611 ]
[ 11.694641   8.390831 -10.681395]
[ 11.616836   8.354407 -10.223578]
[12.507129  8.750157 -8.96329 ]
```

# Now on to the notebook!

Remember:

- Go at your own pace
- Ask questions!
- Take breaks!