Active Brownian Particles Lab

Daniel A Matoz-Fernandez

How can we characterize the phases?

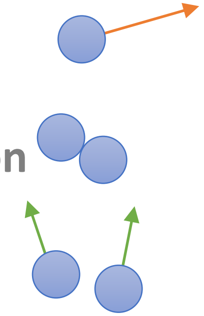
Self-Propulsion

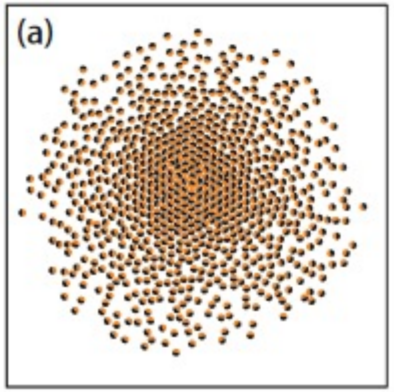
+

“Hard-core” Interaction

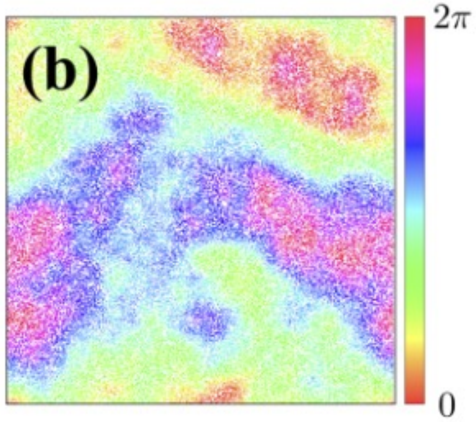
+

Alignment interaction

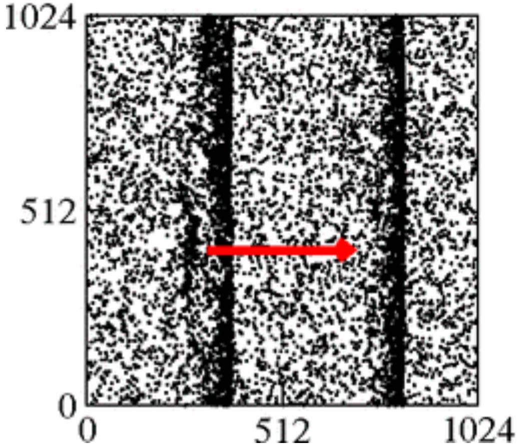




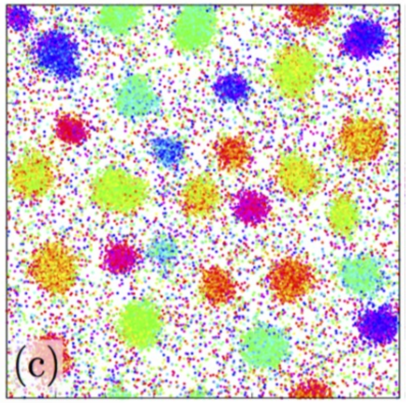
O. Pohl and H. Stark Eur. Phys. J. E 36, 1 (2015).



Levis, Demian et al PRR 1.2 (2019): 023026.



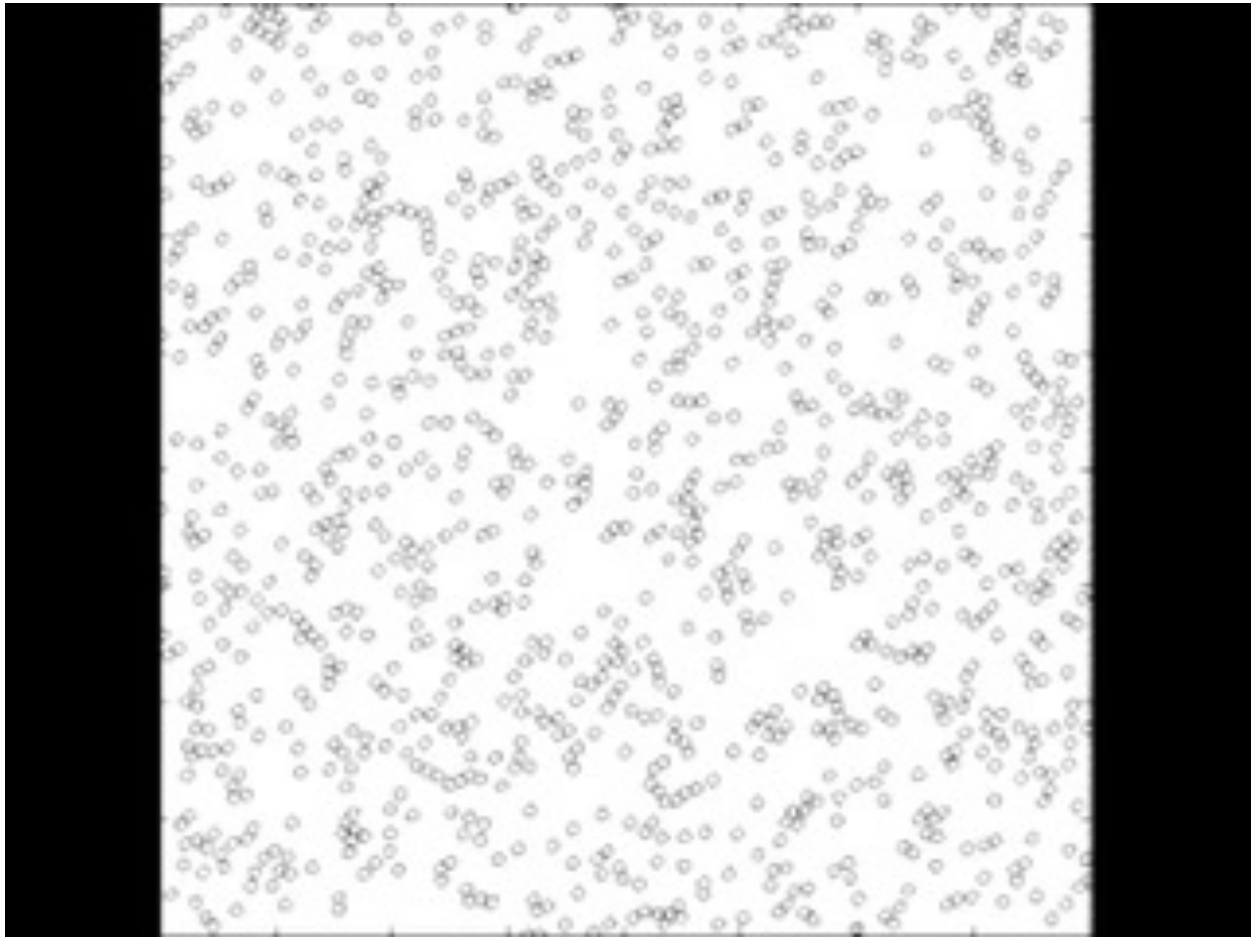
*Chaté, Hugues, et al PRE 77.4 (2008): 046113.
Vicsek, Tamás, et al PRL 75.6 (1995): 1226.



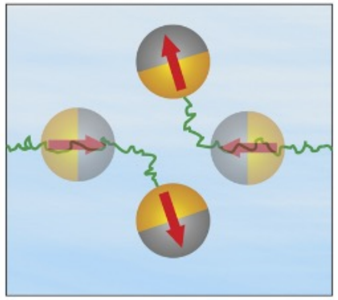
Liebchen, Benno, and Demian Levis PRL 119.5 (2017): 058002.

… and many others

Athermal phase separation

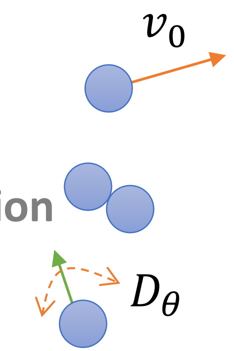


Mechanism



Löwen, Hartmut. *J. Chem Phys* 152.4 (2020): 040901.

Clustering in active systems: *Model*



Self-Propulsion

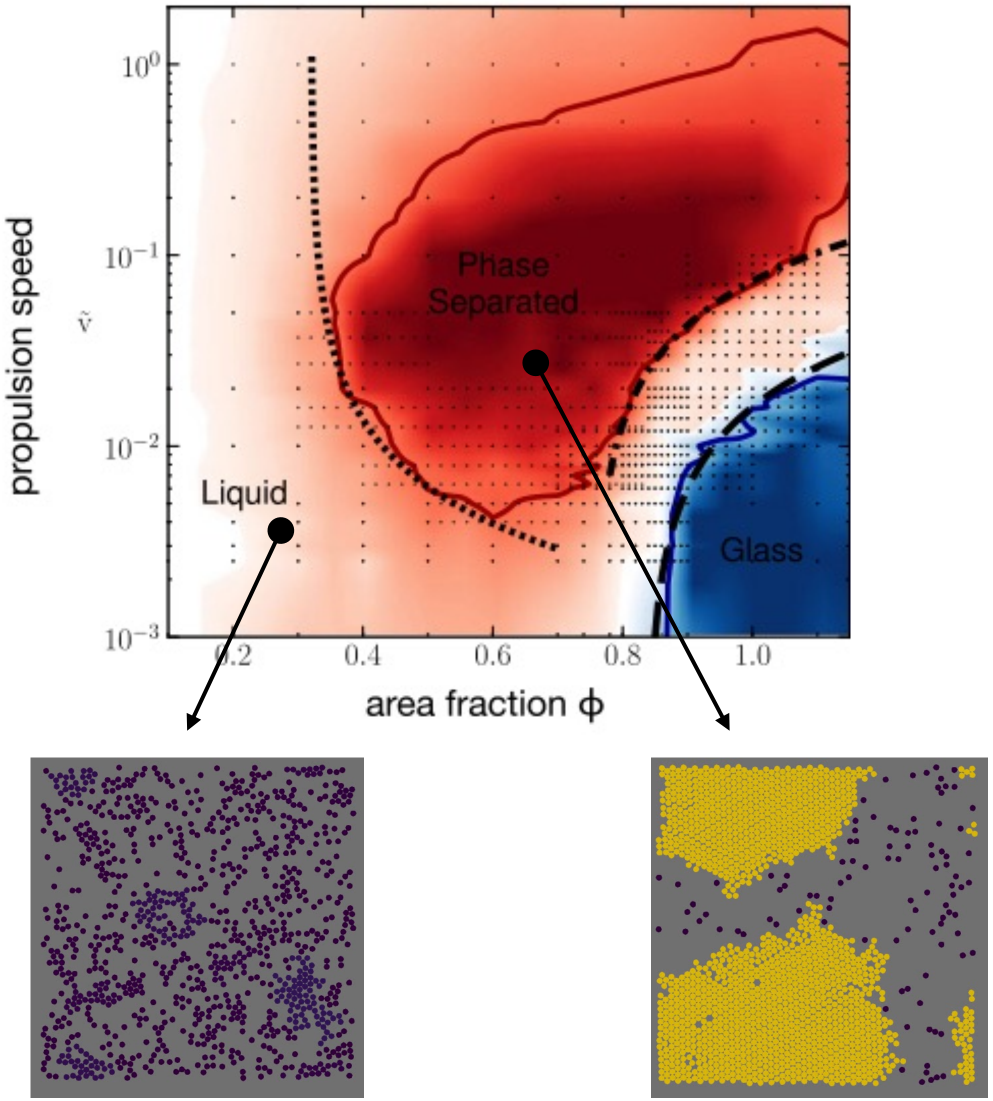
+

“Hard-core” Interaction

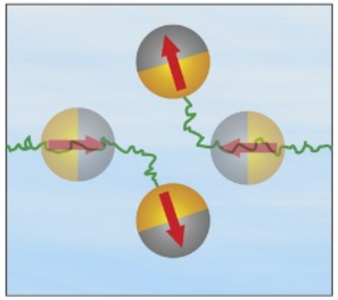
+

Rotational Noise

Fily, Yaouen, Silke Henkes, M. Cristina Marchetti *Soft Matt* 10.13 (2014): 2132-2140.



Mechanism



without inertia
release time 1/D

Löwen, Hartmut. *J. Chem Phys* 152.4 (2020): 040901.

Also see:

T. Speck et al PRL (2014)
Palacci et al Science (2013)
Redner et al PRL (2013)
Cates, Michael E., and Julien Tailleur. Annu. Rev. Condens. Matter Phys (2015)
Caporusso, Claudio et al PRL (2020)
Filly & Marchetti PRL (2013)

… and many others

Exercise 1

- Characterize* the Brownian dynamics for a system of active particles:

$$\vec{v}_i(t+\Delta t) = \sqrt{\alpha}\vec{\eta}_1$$

$$\vec{r}_i(t+\Delta t) = \vec{r}_i(t) + \frac{\Delta t}{\gamma} v_0 \vec{n}_i + \sqrt{\Delta t \alpha}\vec{\eta}_2$$

$$\theta_i(t+\Delta t) = \theta_i(t) + \sqrt{\Delta t \nu}\eta_3$$

With

$$\vec{r} = x\vec{\hat{e}}_x + y\vec{\hat{e}}_y$$

$$\vec{n} = \cos\theta\vec{\hat{e}}_x + \sin\theta\vec{\hat{e}}_y$$

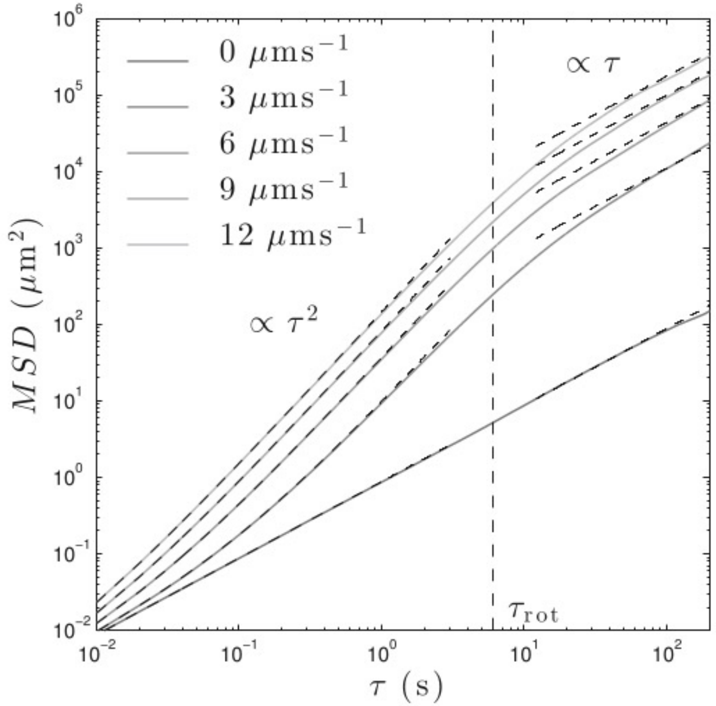
- $\nu = 10^{-1} - 10^{0}$
- $\alpha = 10^{-3} - 10^{0}$
- $v_0 = 0.0 - 10.0$
- $\gamma = 1.0$
- $N = 10^{3} - 10^{4}$
- $\Delta t = 10^{-2} - 10^{0}$

Exercise 1

- * Measure the mean square displacement given by:

$$MSD(t) = \left\langle (\mathbf{r}(t) - \mathbf{r}(t_0))^2 \right\rangle = \frac{1}{N} \sum_{i=1}^{N} (\mathbf{r}_i(t) - \mathbf{r}_i(t_0))^2$$

The plots looks like:



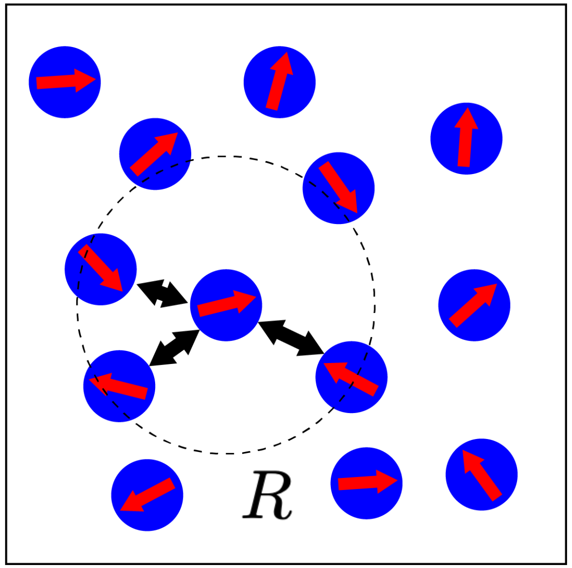
Exercise 2 (Extra)

Consider the case where particles interact with each other like in the xy-model:

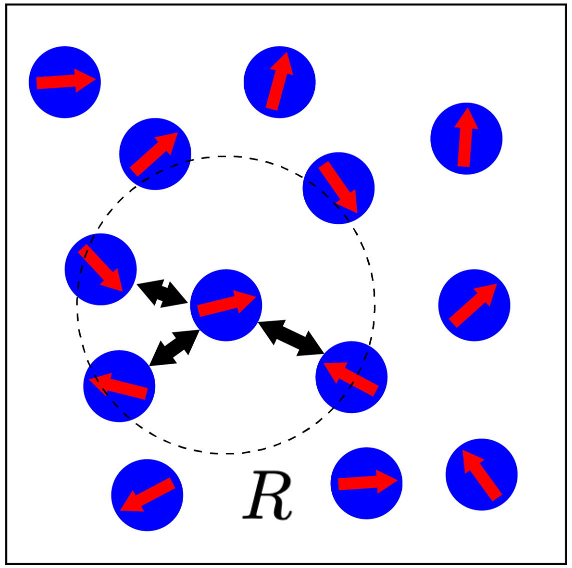
$$\vec{v}_i(t+\Delta t) = \sqrt{\alpha}\vec{\eta}_1$$

$$\vec{r}_i(t+\Delta t) = \vec{r}_i(t) + \frac{\Delta t}{\gamma} v_0 \vec{n}_i + \sqrt{\Delta t \alpha}\vec{\eta}_2$$

$$\theta_i(t+\Delta t) = \theta_i(t) + \Delta t \frac{K}{\pi R} \sum_{j\in R} \sin(\theta_j - \theta_i) + \sqrt{\Delta t \nu}\eta_3$$



Exercise 2 (Extra)



Control Parameters

$$\Delta t = 10^{-2}$$

$$N = 10^3 - 10^4$$

$$R = 1.0$$

$$\nu = 1.0$$

$$\alpha = 0$$

$$\gamma = 1.0$$

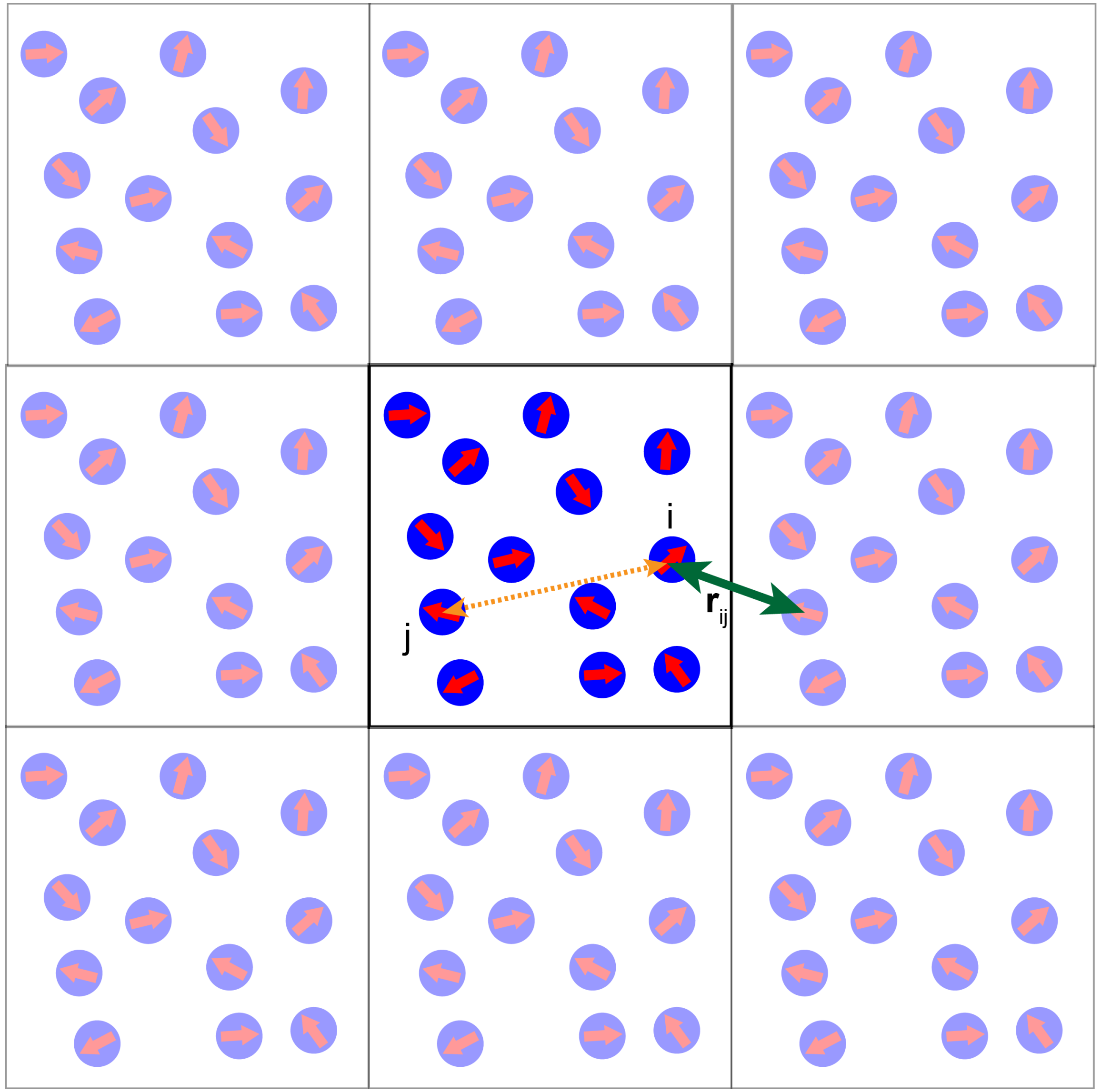
$$\rho_0 = \frac{NR^2}{L^2} = 10$$

$$Pe = \frac{2v_0}{\nu} = 2$$

$$g = \frac{2K}{\pi R^2 \nu} = 0 - 5 \times 10^{-1}$$

Periodic boundary conditions

```
def apply_periodic(x,y,L):
  if x < 0:
      x += L
  elif x > L:
      x -= L
  if y < 0:
      y += L
  elif y > L:
      y -= L
```



A note on how to speed up simulation in python

- For this class we will model a system of active colloids or living matter that is capable to self-propel. The “particles” can interact with other particles and therefore pair forces need to be calculated. The simplest way to do this is by using a double loop as follows:

```
def get_force(x,y,…):

    force_x = np.zeros(N)
    force_y = np.zeros(N)

    for i in range(0, N):
        for j in range(0, N):
            if i != j:
                """
                calculate the force over i
                """
                fij = ...
                force_x[i]= fij...
                force_y[i]= fij...

    return force_x, force_y
```

But this has serious problems with complexity the loop is $\sim (N^2)$

A note on how to speed up simulation in python

$\sim (N^2)$

```
def get_force(x,y,…):
    force_x = np.zeros(N)
    force_y = np.zeros(N)

    for i in range(0, N):
        for j in range(0, N):
            if i != j:
                """
                calculate the force over i
                """
                fij = ...
                force_x[i]= fij...
                force_y[i]= fij...

    return force_x, force_y
```

$\sim (N^2)/2$

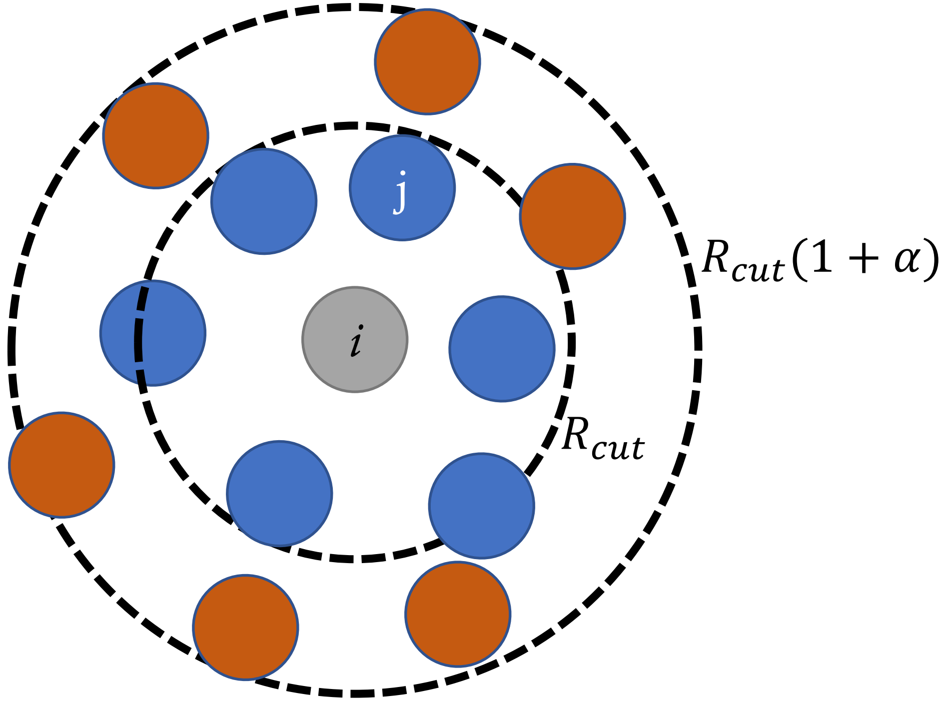
```
def get_force(x,y,…):

    force_x = np.zeros(N)
    force_y = np.zeros(N)

    for i in range(0, N):
        for j in range(i+1, N):
            """
            calculate the force over i
            """
            fij = ...
            force_x[i]= fij...
            force_y[i]= fij...

    return force_x, force_y
```

Using Verlet and Neighbor lists



The idea is to increase the "radius" of neighbor search by a factor of α so we don't need to update it at each time step.

```
def get_neighbour_brute(x, y , rcut):

    neighbour = [ [i] for i in range(0,len(x))]
    print(len(x))
    for i in range(0,len(x)):
        for j in range(0, len(x)):
            if i!=j:
                rij_x = x[i] - x[j]
                rij_y = y[i] - y[j]
                rij2 = rij_x**2 +rij_y**2
                if rij2 <= rcut**2:
                    neighbour[i].append(j)

    return neighbour
```