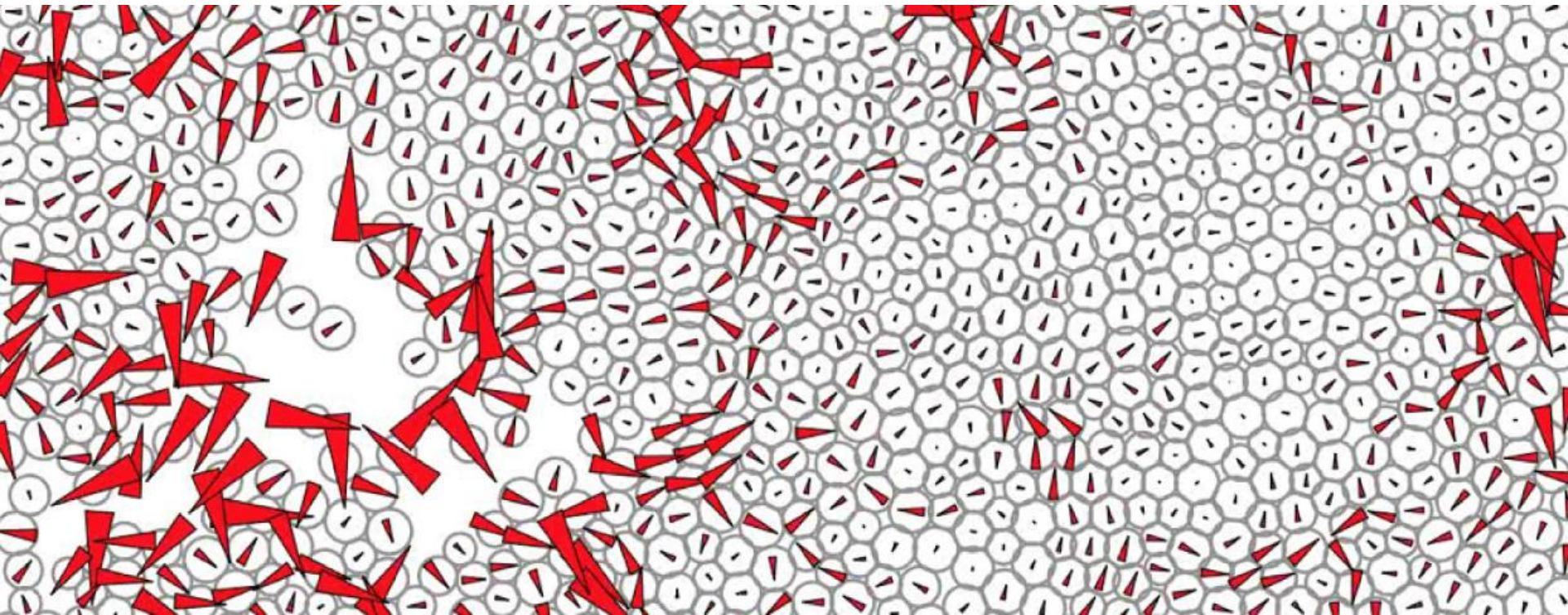


Simulating passive and active particles



Solving ordinary differential equations

$$\frac{dy(t)}{dt} = f(y, t) \quad \text{Generic first order ordinary differential equation (ODE)}$$

$y(t)$ solution: a function y of time

Euler's method: Expand to first order

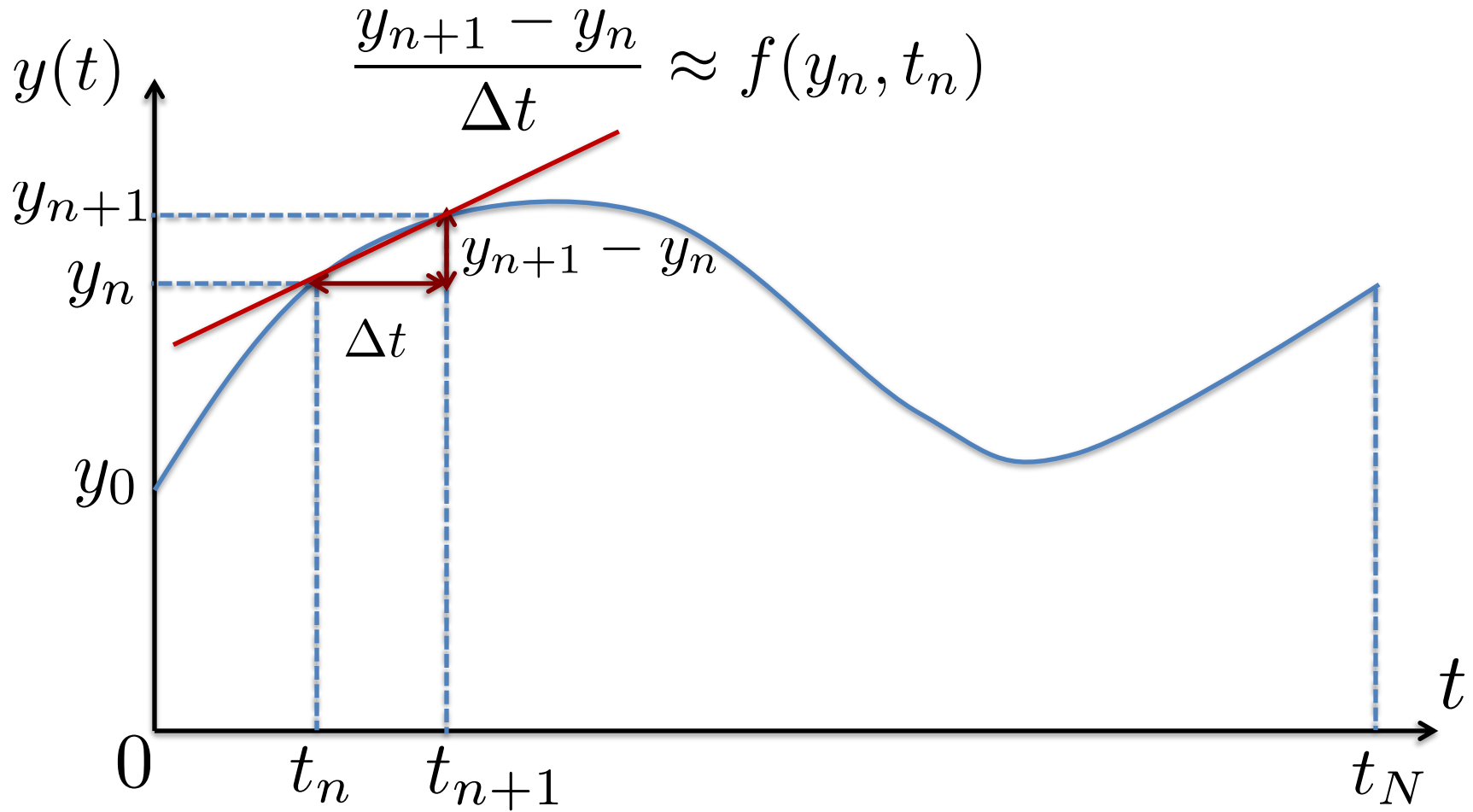
$$\frac{dy(t)}{dt} = \lim_{\Delta t \rightarrow 0} \frac{y(t + \Delta t) - y(t)}{\Delta t}$$

$$t_1 = t_0 + \Delta t, \dots t_{n+1} = t_n + \Delta t \quad \text{discretise time}$$

$$y_{n+1} \equiv y(t_{n+1}) = y(t_n + \Delta t) \quad \dots \text{and the solution}$$

Solving ordinary differential equations

Approximate $f(y,t)$, i.e. the local slope, by its discrete estimate from times t_n and t_{n+1}



Euler's method

We can now iteratively solve the differential equation if we choose an initial condition y_0 at time t_0 .

$$y_{n+1} \approx y_n + f(y_n, t_n) \Delta t \quad y_1 \approx y_0 + f(y_0, t_0) \Delta t$$

Example: Exponential decay

$$\frac{dy(t)}{dt} = -ky(t), \quad y(t) = y(0)e^{-kt}$$

This very simple differential equation has only one solution, that is an exponential decay with decay constant k , and initial condition $y(0)$.

Euler's method simply uses the form of $f(y,t)$ defined on the left hand side:

$$f(y, t) = -ky, \quad y_{n+1} \approx y_n - ky_n \Delta t$$

Exercise 1: Write a program `exponential_decay.py` that solves this equation. Plot the numerical and the analytical solution on the same graph

```
# Euler' method: Basic example: exponential decay
```

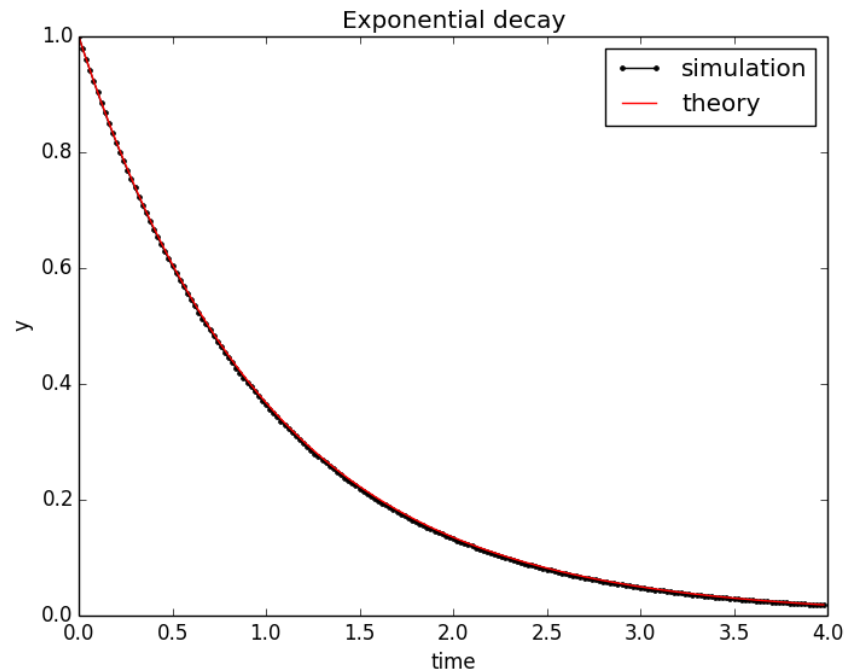
```
#  $dy(t)/dt = -ky$   
#  $y_{n+1} = y_n - ky_n dt$ 
```

```
k=1;  
dt=0.02;  
nmax=200;
```

```
y=np.zeros((nmax,))  
t=np.zeros((nmax,))  
# initial condition  
y[0]=1;  
for l in range(nmax-1):  
    t[l+1]=t[l]+dt  
    y[l+1]=y[l]-k*y[l]*dt
```

```
# Plotting position and the theoretical result
```

```
plt.figure()  
plt.plot(t,y,'.-k',label='simulation')  
plt.plot(t,y[0]*np.exp(-k*t),'-r',label='theory')  
plt.legend()  
plt.xlabel('time')  
plt.ylabel('y')  
plt.title('Exponential decay')
```



Integrating stochastic ODEs: Euler Maruyama

Stochastic ODE for $X(t)$: \longrightarrow In probability theory language:

$$\frac{dX}{dt} = a(X, t) + b(X, t)\eta$$

deterministic

$$dX_t = a(X_t, t)dt + b(X_t, t)dW_t$$

Wiener process

white Gaussian
(normally distributed)
noise $N(0,1)$

$$\langle \eta(t) \rangle = 0$$

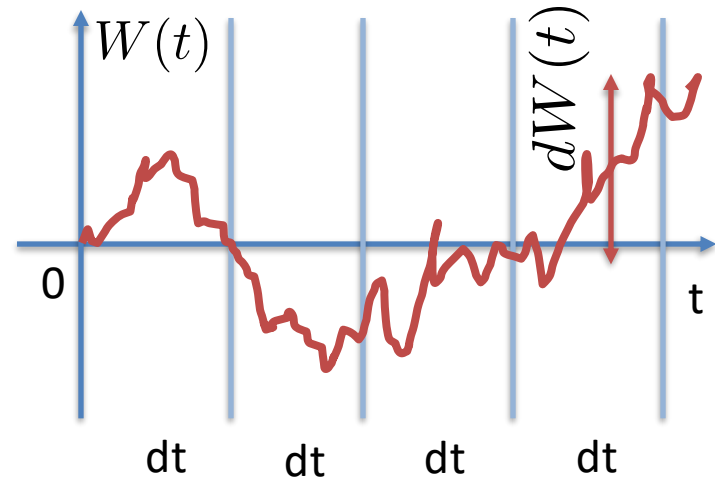
$$\langle \eta(t)\eta(t') \rangle = \delta(t - t')$$

Wiener process: A random walk, i.e. the integrated white noise

Compute statistics of it over a time step dt :

$$\langle dW(t) \rangle = 0$$

$$\langle (dW(t))^2 \rangle = dt$$



Euler-Maruyama algorithm:

Note the square root of time step in the stochastic part!

$$X(t_{n+1}) \approx X(t_n) + a(X(t_n), t)dt + b(X(t_n), t)\eta\sqrt{dt}$$

Numerical integration schemes

Convergence: Does the numerical scheme converge to the true solution?

$$\lim_{\Delta t \rightarrow 0} \max_k (||y_k - y(t_k)||) = 0$$

Truncation error: What order of Δt are the errors to the solution? This is closely associated with the order of the algorithm. For example, Euler is a 1st order algorithm, and has errors of the order Δt^2 . Some Runge-Kutta methods are fourth order, i.e they have errors of order Δt^4 .

Stability: In problems with several different time scales (think e.g. a rocket rising compared to the motion of the earth around the sun), the solution will either be inaccurate if the time step is too large, or much too slow for a time step which is sufficiently small.

Conservation laws: Does the numerical scheme conserve the same quantities as the underlying equations? For example, the Euler method does not conserve energy. Therefore, for problems where that is important (like planetary motion), we need to use a different algorithm.

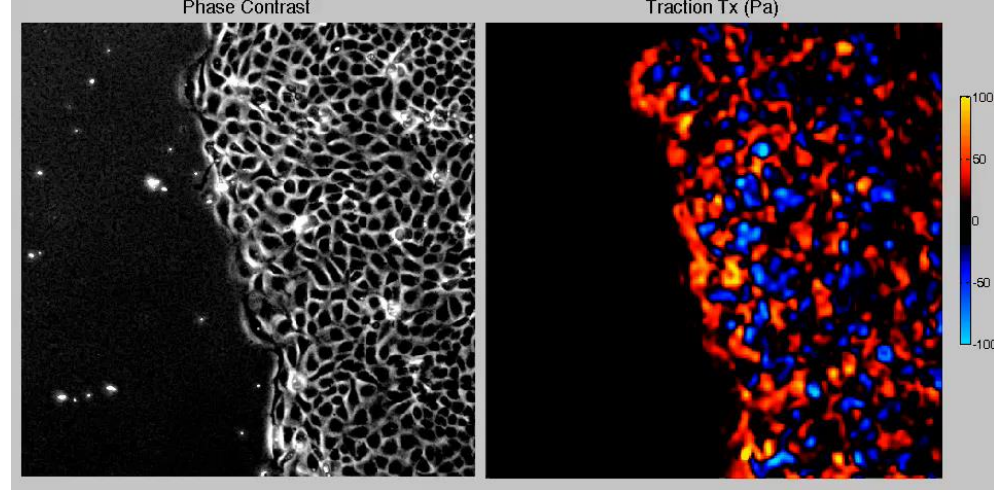
(don't tell anyone:

http://en.wikipedia.org/wiki/Numerical_methods_for_ordinary_differential_equations

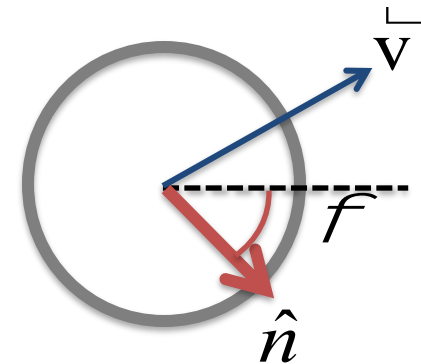
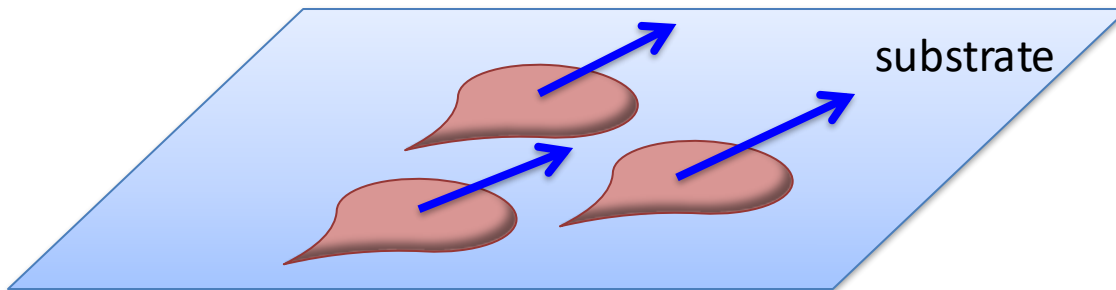
has a pretty good summary)

Active Brownian particles

e.g. cells, people, birds, colloids, little robots ...



- Particles move over a substrate or through a fluid with friction. Self-propulsion is an internally generated force in a direction \mathbf{n}

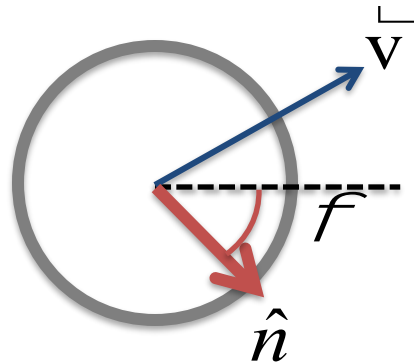


- Write down general equations of motion for each particle i ; Newton says:

$$m\vec{a}_i = \vec{F}_i$$

Includes active internal force, friction with the substrate, and repelling other particles

Active Brownian particles:

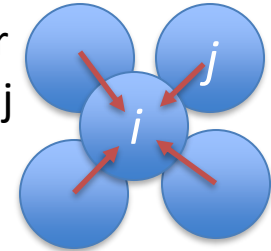


$$m\vec{a}_i = F_{\text{act}}\hat{n}_i - \zeta\vec{v}_i + \sum_j \vec{F}_{ij} + \vec{\eta}_i^T$$

active
force

friction
force

repelling all
the other
particles j



thermal
fluctuations

If we are studying something which is really small ...

$$m \sim R^3 \quad (\text{volume})$$

$$\zeta \sim R \quad (\text{Stokes drag})$$

therefore: $\frac{m}{\zeta} \sim R^2 \rightarrow 0 \quad \text{when} \quad R \rightarrow 0$

... we can neglect the mass of the particle! We have overdamped equations of motion:

$$\vec{v}_i = \frac{\vec{F}_{\text{act}}}{\zeta} + \frac{1}{\zeta} \sum_j \vec{F}_{ij} + \frac{\vec{\eta}_i^T}{\zeta}$$

Introduce: active velocity v_0 , and mobility μ

$$\vec{v}_i = v_0\hat{n}_i + \mu \sum_j \vec{F}_{ij} + \vec{\eta}_i^T$$

This is the equation we want to simulate

Angular dynamics can take many different symmetries:

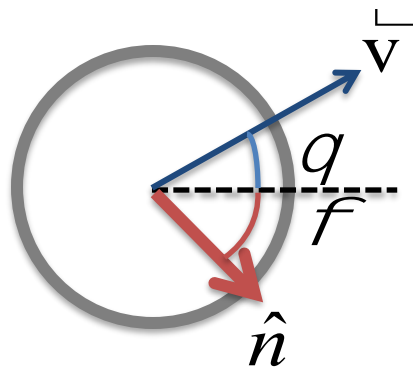
1. Apolar **diffusion** of the polarization vector: classic ABP dynamics

$$\dot{\phi}_i = \eta_i, \quad \langle \eta_i(t) \eta_j(t') \rangle = 2D_r \delta_{ij} \delta(t - t')$$

2. **Alignment** with neighbouring polarization vectors: **Vicsek-like**

$$\dot{\phi}_i = \frac{1}{\tau_z} \sum_j \sin(\phi_j - \phi_i) + \eta \quad \text{polar} \qquad \dot{\phi}_i = \frac{1}{\tau_z} \sum_j \sin(2(\phi_j - \phi_i)) + \eta \quad \text{nematic}$$

3. **Alignment**, with **coupling** to the own **velocity** or **forces** of the particle



For example:

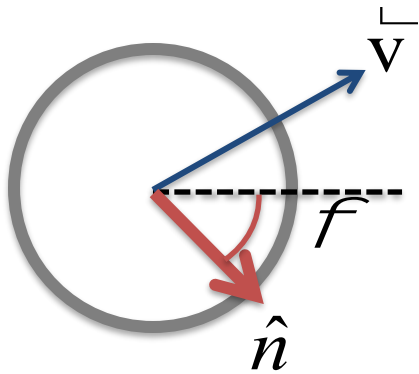
$$\dot{\phi} = \frac{1}{\tau} (\theta - \phi) + \eta$$

Simulating active & passive particles

$$\vec{v}_i = v_0 \hat{n}_i + \mu \sum_j \vec{F}_{ij} + \vec{\eta}_i^T \quad \text{1st order ODE for the x and the y component of each particle's motion}$$

But wait – what about the stochastic component and the direction of active motion?

$$\langle \vec{\eta}_i(t) \cdot \vec{\eta}_j(t') \rangle = 4D\delta_{ij}\delta(t - t') \quad \text{Thermal diffusion with diffusion coefficient D}$$

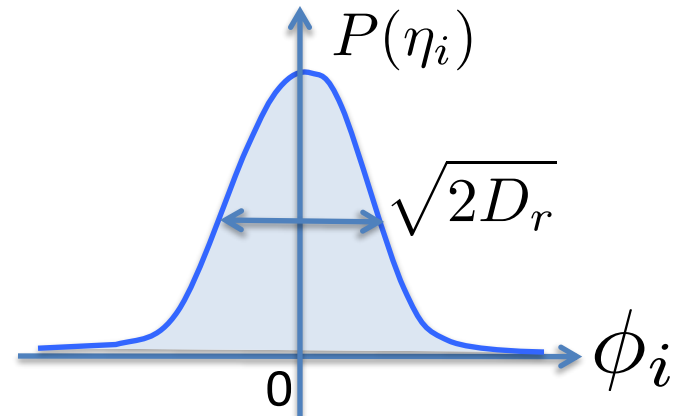
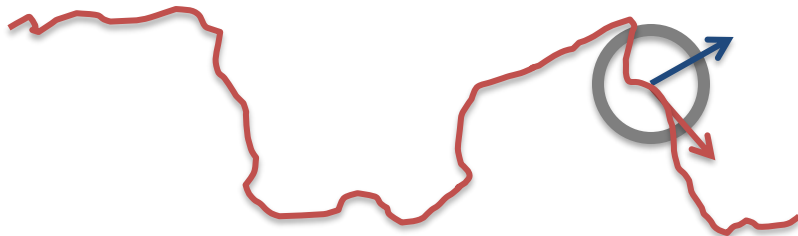


Simplest behaviour: The direction is not pre-determined, but simply fluctuates over time

$$\hat{n}_i = (\cos \phi_i, \sin \phi_i)$$

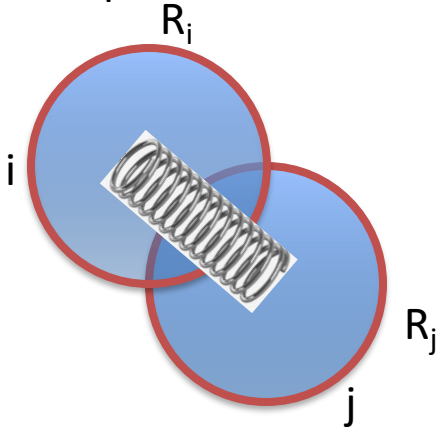
$$\frac{d\phi_i}{dt} = \eta_i \quad \leftarrow \text{random variable with mean 0, following e.g. a normal distribution:}$$

Path of a single active particle:



Interaction forces

- Interaction potentials set the force between particles. We generally want forces that are:
 - Repulsive if particles try to overlap
 - Short range, i.e. zero if we go to large distances
 - Pairwise additive
- Many options, including e.g. the Lennard-Jones and Weeks-Chandler-Anderson potentials used in molecular dynamics simulations.
- Simplest useful model: **One-sided harmonic springs**



$$V_{ij} = \frac{k}{2} (|\vec{r}_j - \vec{r}_i| - (R_i + R_j))^2 \quad \text{for} \quad |\vec{r}_j - \vec{r}_i| < (R_i + R_j)$$

$$= 0 \quad \text{for} \quad |\vec{r}_j - \vec{r}_i| > (R_i + R_j)$$

$$\vec{F}_{ij} = -\vec{F}_{ji} = -\vec{\nabla}_{\vec{r}_i} V_{ij}$$

Repel each other proportional to compression when in contact, no force otherwise

$$= -k (|\vec{r}_j - \vec{r}_i| - (R_i + R_j)) \frac{\vec{r}_j - \vec{r}_i}{|\vec{r}_j - \vec{r}_i|}$$

Euler-Maruyama algorithm

$$\frac{dx_i}{dt} = v_0 \cos(\phi_i) + \mu \sum_j F_{ij}^x(x_j - x_i, y_j - y_i) + \bar{\eta}_x$$

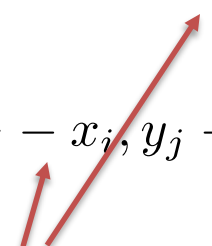
We have three ODEs to solve for each particle, and N particles.

$$\frac{dy_i}{dt} = v_0 \sin(\phi_i) + \mu \sum_j F_{ij}^y(x_j - x_i, y_j - y_i) + \bar{\eta}_y$$

There will be a total of 3N equations!

$$\frac{d\phi_i}{dt} = \eta_i$$

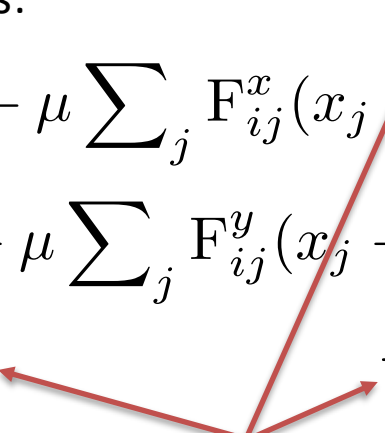
Forces couple the ODEs of different particles to each other



Use the Euler algorithm to solve these equations:

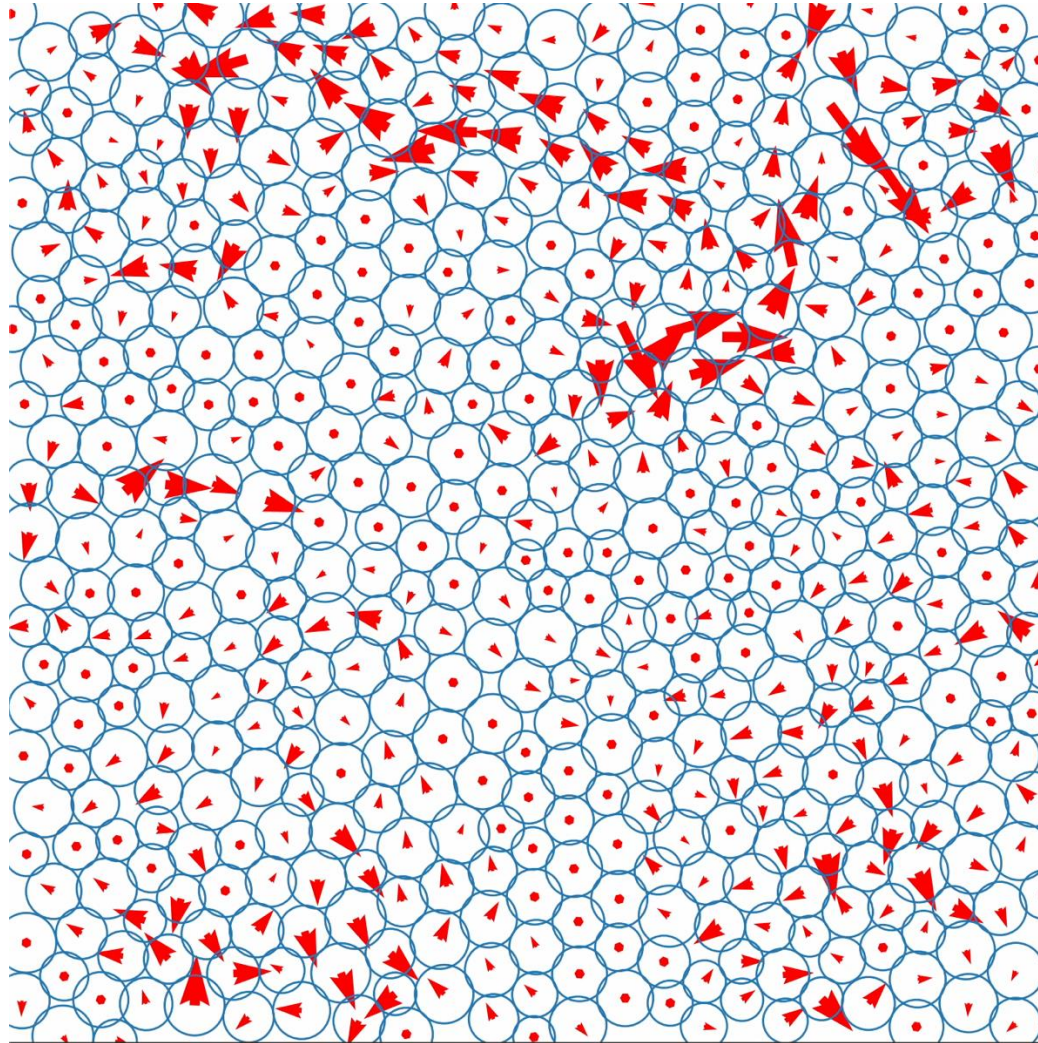
$$x_i(t_{n+1}) = x_i(t_n) + \Delta t \left[v_0 \cos(\phi_i) + \mu \sum_j F_{ij}^x(x_j - x_i, y_j - y_i) \right]$$

$$y_i(t_{n+1}) = y_i(t_n) + \Delta t \left[v_0 \sin(\phi_i) + \mu \sum_j F_{ij}^y(x_j - x_i, y_j - y_i) \right]$$

$$\phi_i(t_{n+1}) = \phi_i(t_n) + \sqrt{2D_r\Delta t} N(0, 1) + \sqrt{2D\Delta t} N(0, 1)$$


... except for these bits. $N(0,1)$ is a standard normally distributed random number. The $\sqrt{\Delta t}$ is because we have a stochastic ODE

Simulation of active particles in the dense liquid phase



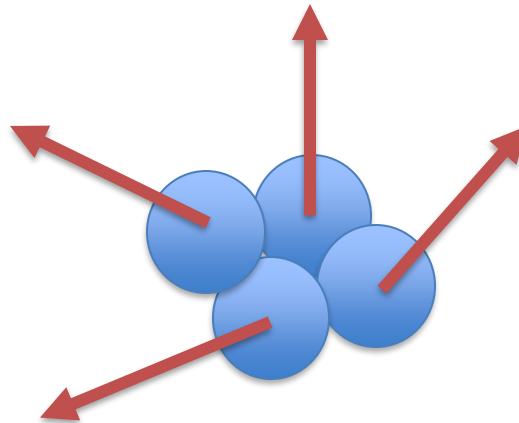
Arrows represent the velocity

Variations on ABPs: model with alignment

$$\frac{d\vec{r}_i}{dt} = v_0 \hat{n}_i + \mu \sum_j \vec{F}_{ij} + \vec{\eta}_i$$

$$\frac{d\theta_i}{dt} = -J \sum_{\langle ij \rangle} \sin(\theta_j - \theta_i) + \eta_i^r$$

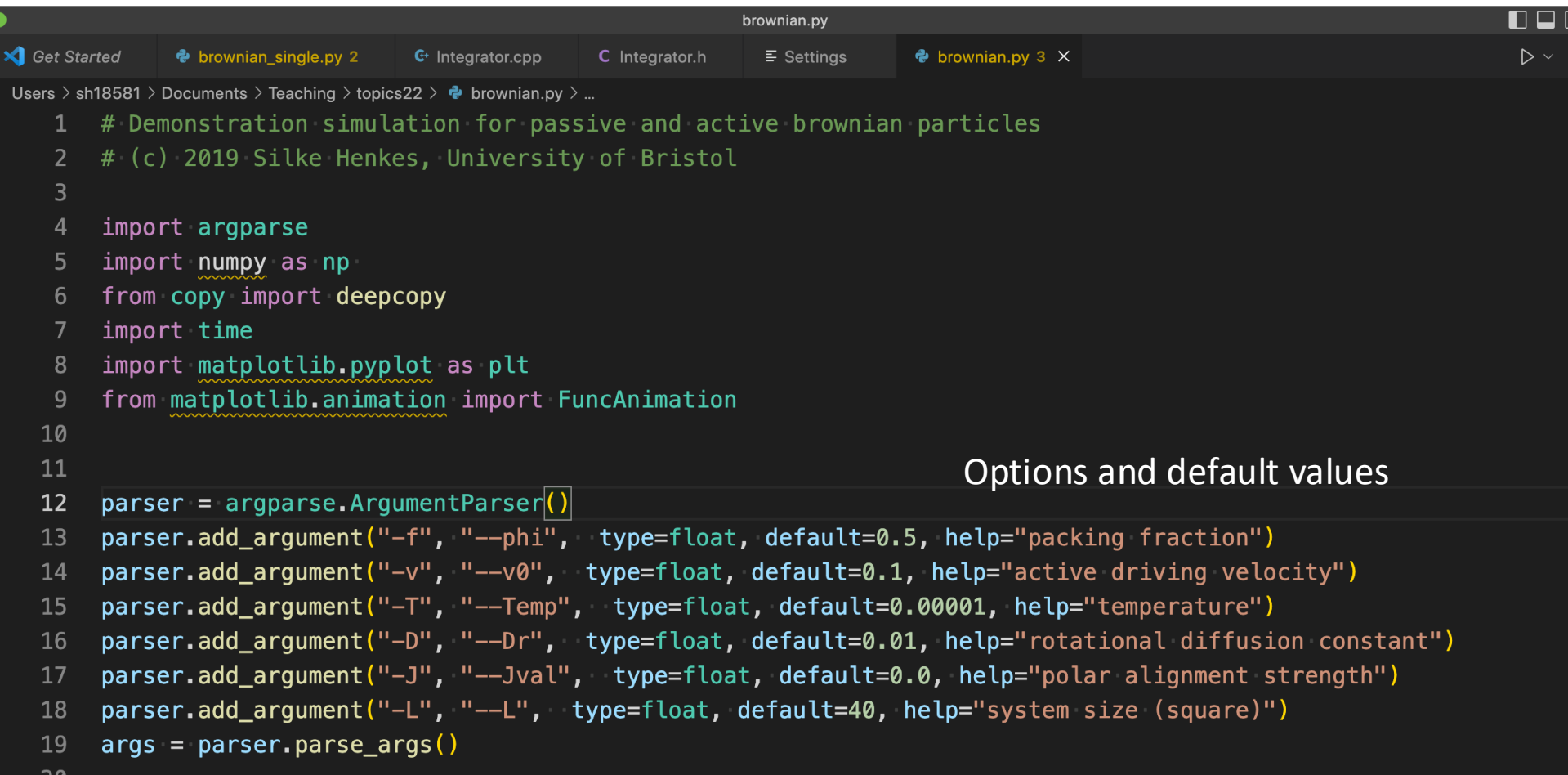
wants to align arrows of
particles that are touching
each other



Variations on ABPs: running Brownian.py

Syntax: `python brownian.py -f 0.1 -v 0.3 -T 0.0 -D 0.01 -J 1.0 -L 40`

each `-f` and so on is an option to change the default behaviour, and is **optional**.



```
1  # Demonstration simulation for passive and active brownian particles
2  # (c) 2019 Silke Henkes, University of Bristol
3
4  import argparse
5  import numpy as np
6  from copy import deepcopy
7  import time
8  import matplotlib.pyplot as plt
9  from matplotlib.animation import FuncAnimation
10
11
12  parser = argparse.ArgumentParser()
13  parser.add_argument("-f", "--phi", type=float, default=0.5, help="packing fraction")
14  parser.add_argument("-v", "--v0", type=float, default=0.1, help="active driving velocity")
15  parser.add_argument("-T", "--Temp", type=float, default=0.00001, help="temperature")
16  parser.add_argument("-D", "--Dr", type=float, default=0.01, help="rotational diffusion constant")
17  parser.add_argument("-J", "--Jval", type=float, default=0.0, help="polar alignment strength")
18  parser.add_argument("-L", "--L", type=float, default=40, help="system size (square)")
19  args = parser.parse_args()
```

Options and default values