Cells on a Sphere Simulation

Eduardo Pavinato Olimpio

August 13, 2015

1 Movement Simulations

In order to understand and get insights on the movement of the organoids, we performed some simulations of cells on a sphere. The cells are treated as self-propelled particles (SPP) in an onverdamped regime. The basis for the simulation is found in the reference [1]. All the code is found in

1.1 How to use compiled FORTRAN in Python

Before explaining the simulations, we have made use of some compiled FORTRAN code in Python. This has the advantage of (i) speed and (ii) a famous Delaunay triangulation for the sphere is written in FORTRAN. This is achieved using a numpy modeule called `f2py` (`http://docs.scipy.org/doc/numpy-dev/f2py/`). To make it work on Windows, you have to follow the instruction below.

1.1.1 Use Pure Python (no distribution)

Sorry, but I could only make it work on the pure Python installation (`https://www.python.org/downloads/`). If you have any distribution (e.g. Anaconda) you will need to uninstall it to avoid conflict. It can be possible to make it work with Anaconda with some tuning in the Windows register, but it would take too much time. Moreover, I found it much more easy if you use the 32-bit Python interpreter. In Linux things run more smoothly.

When you switch back to the normal Python, you will need to get the binaries of the packages in `http://www.lfd.uci.edu/~gohlke/pythonlibs/`. Then use the command line and go to the forder where the binaries are and type:

```
$PYTHON_PATH$\Scripts\pip.exe install <name_of_the_binary>
```

where we assume that the installation of Python is in the folder `$PYTHON_PATH$`. Even in a 64-bit system I recommend the use of the 32-bit Python interpreter in case of compiled code. In this case, you will need the win32 version of the wheel files in the website given above.

Required Packages for the simulation:

1. numpy
2. scipy
3. matplotlib

1.1.2 Install a FORTRAN compiler(I used G95)

You will need a Fortran compiler native for Windows. I used G95 which worked nicely. The installation procedure is quite simple and you can obtain the compiler at `http://www.fortran.com/the-fortran-company-homepage/whats-new/g95-windows-download/`.

1.1.3 Install MinGW

Now you can download the nice distribution of MinGW available in `https://github.com/develersrl/gccwinbinaries`. This installation already take care of all the environment variable that needs to be adjusted. MinGW essentially installs the GCC libraries for compiling the code using `f2py`. You can find more information about MinGW in `http://www.mingw.org/`.

1.1.4 Test if it works

Write the below fortran code in a file `foo.f95`.

```
subroutine foo(n, a)
  integer, intent(in)  :: n
  real,    intent(out) :: a(n,n)
  do i=1,n
    a(i,i) = 1.0
  end do
end subroutine foo
```

Now run the command:

```
f2py foo.f95 -m foo -h foo.pyf
```

if it gives no error[1] then run:

```
f2py -c --fcompiler=g95 --compiler=mingw32 foo.pyf foo.f95
```

If the compilation gives no error, you can use your library in Python. To do it, write the following code:

```
from numpy import *
import foo

print(foo.foo.__doc__)  # shows python interface generated by f2py
a = foo.foo(4)
print(a)
```

When you run this you will see how the foo function is called and the matrix a (an identity matrix).

1.1.5 Compiling the code

Before you run the code, if the compiled libraries are not working, you will have to compile it on your machine. This is done by running the batch file `compile.bat` and choosing the appropriate FORTRAN file.

[1] Sometimes it gives an error related to the `f2py` script path. This can be easily fixed by editing the first line of `$PYTHON_PATH$/Scripts/f2py.py` in order to comply with `$PYTHON_PATH$`

1.2 The model

We assume that the cells are self-propelled particles (SPP) moving in a overdamped regime and constrained to move on a sphere. Defining the projector operator of a vector in the plane tangent to the sphere in a point $\vec{r}_i$ as:

$$P_T\left(\hat{r}_i, \vec{a}\right) = \vec{a} - \left(\hat{r}_i \cdot \vec{a}\right)\hat{r}_i \tag{1}$$

we know that by the balance of forces in this plane we have:

$$\gamma\dot{\vec{r}}_i = P_T\left(\hat{r}_i, \vec{F}_{\text{act}} + \sum_j \vec{F}_{ij}\right) \tag{2}$$

where γ is the friction coefficient, $\vec{F}_{\text{act}}$ is the active force, pointed in a direction $\hat{n}_i$ and $\vec{F}_{ij}$ is the two-body forces of different pairs. As usual we define $\vec{F}_{\text{act}} = \gamma\nu_0\hat{n}_i$ such that the dynamics of the position is given by:

$$\dot{\vec{r}}_i = P_T\left(\hat{r}_i, \nu_0\hat{n}_i + \mu\sum_j \vec{F}_{ij}\right) \tag{3}$$

where $\mu = \gamma^{-1}$.

The change in the direction $\hat{n}_i$, which is in the tangent plane, is given by its tendency to align to a vector $\hat{k}_j$. This leads to an interaction that tends to rotate the direction by $J\hat{n}_i \times \hat{k}_j$, J denoting the coupling strength[2]. This vector must be projected in the direction normal to the plane, and the direction of the movement of the direction vector is perpendicular both to the tangente plane and the direction vector, leading to:

$$\dot{\hat{n}}_i = \left[\hat{r}_i \cdot \left(J\hat{n}_i \times \hat{k}_j\right) + \eta_n\xi_i\right]\left(\hat{r}_i \times \hat{n}_i\right) \tag{4}$$

where η_n is the noise strength and ξ_i is a normal Gaussian variable. This introduces some noise in the alignment of the particles.

The "alignment vector" $\hat{k}_j$ can be defined in several ways. In [1], for example, they define it as a kind of XY model where the particles align to the sum of their neighbors directions. In our model we use that the particles tend to align with the direction of the resultant force $\sum_j \vec{F}_{ij}$.

1.2.1 Numerical computation

For all the simulations we use $\tau = \frac{1}{\mu k}$ (k is the elatic constant of the forcing term) as the characteristic time and σ (the "size" of a cell) as the characteristic distance. The parameters of the system are then:

1. **N:** Number of particles in the system.
2. **Φ:** "Packing ratio" of the system, given by $\frac{N\pi\sigma^2}{4\pi\rho^2}$. This defines the radius ρ of the sphere.
3. **ν_0:** Active force strength with units of velocity ($\sigma\tau^{-1}$).
4. **η_n:** Noise strength with units of τ^{-1}.

[2] If in the plane, this is equivalent to $J\sin\theta$, with θ denoting the angle between the unitary vectors. This is XY-model type interaction, although it is not related to the "mean field", or mean alignment, of the neighbor particles.

5. J: Coupling strength with units of τ^{-1}.

In our computation we use $\Delta x = 0.005\sigma$ ($\Delta t = \Delta x/\nu_0$) and run the simulation for $t = 50\tau$ after a relaxation (to avoid the overlaps due to random initial conditions) that runs for the equivalent of 2σ or $2/\Delta x\tau$. The choice of Δx is done by the fact that we must have, in $\vec{r}$ integration, $\Delta\vec{r} << \sigma$. Care must be taken for extremely low values of ν_0.

The integration is done by using a simple Euler-Maruyama method, which requires a small time step to avoid bigger numerical errors. For each time step, after the update of $\vec{r}_i$, we correct the vector to constrain it to the sphere [1].

1.2.2 Spring constant

As can be seen, the spring constant is not a parameter of our system. This is due to the fact that this would be redundant, implying only a rescale of the parameters and of the characteristic time τ. As $\tau = \frac{1}{\mu k}$ and all the variables (except for the geometrical factor Φ) scale with that, the simulation can include changes in the spring constant by simply scaling the parameters J, ν_0 and η_n proportinaly. For example, if we want to make the spring twice as strong we need to multiply the parameters by half. We must be aware, however, that the value of ν_0 should not get too high (say in the order of 100) without changing Δt.

1.2.3 Determining the bin size for polar histogram

For the polar histogram to be built (after rotation when the system reaches stationaty state), we need to choose the appropriate bin size in order to have particles and good statistics in all of them. We define the bin size by considering that for $\Phi = 1$ the mean distance between particles is 2σ and then we must have $\Delta\theta \approx \frac{2\sigma}{R}$. As the mean distances scale approximately with $\Phi^{-1/2}$ then we have $\Delta\theta \approx \frac{4}{\sqrt{N}}$ and the number of bins is then $n_{bins} = \pi\frac{\sqrt{N}}{4}$

1.3 Using the code

There are essentially two forms to run the code, one that runs for a fixed set of parameters `sim_sphere_fortran.py` and other that calls this same file changing the set of parameters `loop_sim_sphere_fortran.py`. The parameters are set in the file `parameters.ini`.

1.3.1 Configuration file

You should set all the parameters in the file `parameters.ini`, which is contained below. The lines starting with `'#'` are comments, and the file is meant to be self explanatory. This file is read according to the function `readConfigFile` in the file `myutils.py`.

```
# number of particles
N = 100

# Force Parameters
# active force constant
nu_0 = 0.1
# spring 'coupling' term
J = 0.1
# noise strength
eta_n = 0.1
```

```
# Packing fraction (geometrical parameter)
phi_pack = 1

# Update Neighbors every time step? (0: no, 1: yes)
update_nn = 0

# Force type
# 0: repulsive only
# 1: Hooke's Law
# 2: Hooke's Law with anisotrpy and bond breaking
ftype = 1

# If using ftype=2 set the parameters below
# Anisotropy of the force (k_pull/k_push)
fanisotropy = 1
# maximum distance between centers for the breaking of the bond (in sigma, must be above
max_dist = 0

# Number of points to fix (randomly chosen)
N_fix = 3

# Include chemoattractant? (0: no, 1: yes)
chemoatt = 0
# If yes, choose the point(s).
# Ex.: chemo_points = [[0, 0, rho+2]] -> one point: x, y = 0, z = sphere radius(rho) + 2
# Ex.: chemo_points = [[0, 0, rho],[-1, 2*rho, 3]] -> two points
chemo_points = [[0,0,rho+0.1]]
# chemotaxis direction 'coupling' term
J_chemo = 1

# steps before starting saving the distributions
n_before = 1000
# Rotate coordinates according to the axis of rotation? (0: no, 1: yes)
# If so, they will be rotating at n_save_dist
rotate_coord = 0

# Simulation Parameters
# Number of steps
n_steps = 5000
# timestep max displacement (in sigma, dt = dx/nu_0)
dx = 0.005
# save data every n_save steps for the video output
n_save = 50

# outfile for data (if changed you have to change the reading scripts also)
outfile_video = ./data/sphere_data_video_%Y-%m-%d_%H-%M-%S.p
outfile_analysis = ./data/sphere_data_analysis_%Y-%m-%d_%H-%M-%S.p
outfile_postrotation = ./data/sphere_data_rotated_%Y-%m-%d_%H-%M-%S.p
```

1.3.2 Running the code

1.3.3 Navigating through the data

Each time the simulation at `sim_sphere_fortran.py` runs, it generates a new file at the folder data and update (or generate) a file called `matadata.dat`. This file is used to navigate through the parameters of the runs without opening the files. It allows both the graphing scripts to get all files with a determined set of parameters and also allows the user to open this file in Excel (tab delimited file) and navigate manually through the parameters and run the appropriate analysis by using its `date_string` reference. The code referent to the metadata is contained in the file `metadata_lib.py`.

References

[1] Rastko Sknepnek and Silke Henkes. Active swarms on a sphere. *Physical Review E*, 91, 2014.