Diffusion-driven aggregation of particles in quasi-2D membranes

Oscar Rausis

June 2023

1 Abstract

Many biological membranes can be modeled as two-dimensional (2D) viscous fluid sheets surrounded by three-dimensional (3D) fluids of different viscosity. Such membranes are dubbed quasi-2D as they exhibit properties of both 2D and 3D fluids. The Saffman length is a parameter that describes the energy exchange between the membrane and bulk fluids and controls the cross-over from 2D to 3D hydrodynamics. We aim to model diffusion-driven aggregation of particles embedded in a quasi-2D membrane. It is known that hydrodynamic interactions between solute particles significantly reduce their aggregation rate in 3D fluids. It is expected that in quasi-2D membranes the reduction of the aggregation rate will be even more pronounced due to a slower spatial decay of hydrodynamic interactions in 2D. We perform computer simulations to study the aggregation rate of solute particles as a function of the Saffman length.

2 Introduction

Many biological processes occur in thin membranes. If a particle embedded in the membrane is significantly larger than the particles making up the membrane, we can model the membrane as a 2D viscous fluid sheet surrounded by a 3D “bulk” fluid of different viscosity. One example is a typical mammalian cell membrane, made up of lipids with a size less than half a nanometer, and surrounded by a bulk fluid. Proteins embedded in such membranes are much larger, on the order of a few nanometers. Another example of such a fluid is a sheet of freely suspended smectic crystals. The crystals form discrete layers, where each layer is a molecule thick, and these layers are surrounded by air. There is a much greater energy cost to move vertically between layers as opposed to within the plane of a single molecule-thick sheet, thus embedded particles are effectively confined to 2 dimensions. Such membranes are dubbed quasi-2D membranes as they exhibit properties of both 2D and 3D fluids. The Saffman length is given by the relative viscosities between the 2D fluid membrane and 3D bulk fluid:

\[ l_S = \frac{\eta_m}{2\eta_f} \]  

where \( \eta_m \) is the viscosity of the membrane and \( \eta_f \) is the viscosity of the bulk fluid. The Saffman length has units of meters, as \( \eta_m \) has units of \( Pa \cdot s \cdot m \), and \( \eta_f \) has units of \( Pa \cdot s \). As a particle moves through the fluid, the fluid will move with it in a radius roughly equal to the Saffman length. The Saffman length describes the energy exchange between the membrane and bulk fluid. Because
of its direct dependence on the relative viscosities of the membrane and bulk fluids, the Saffman length can be altered experimentally. When using films of smectic crystals, a discrete number of layers can be created, altering the viscosity and thus Saffman length within the quasi-2D fluid.

Hydrodynamic interactions are generated by the forces particles exert on the fluid. These forces generate flow fields in the fluid, and these flows affect other particles, allowing particles to interact indirectly. The fluid is the agent that transfers information, and the hydrodynamic forces are the only forces between particles. At large distances, we can approximate these particles as point-like, and find that the strength of these hydrodynamic interactions decays with an inverse squared relationship to distance in 3D, but logarithmically in 2D. The Saffman length dictates the range of hydrodynamic interactions, with the strength of interactions decaying as an inverse squared function beyond the Saffman length, but logarithmically on a scale shorter than the Saffman length. As such, the Saffman length characterizes the crossover between 2D and 3D hydrodynamics.

Hydrodynamic interactions allow particles to generate fluid flows as they move through the membrane. For example, as a particle moves it will push on the fluid, generating a fluid flow away from itself. As such, two particles moving towards one another will generate a fluid flow that will hinder their mutual approach. These hydrodynamic interactions are expected to reduce the rate of aggregation by roughly 46% in 3D [2] for a neutral species (U(r)=0), meaning there are no forces between particles other than hydrodynamic forces. Hydrodynamics are expected to have an even greater effect on reducing the aggregation rate in 2D due to the longer spatial memory of such interactions.

We simulate the aggregation of particles embedded in quasi-2D membranes to determine the effect of the Saffman length on aggregation. Particles are subject to thermal agitations, and do not interact with other particles other than by hydrodynamic interactions. Thus, we model diffusion-driven aggregation, where particles will stick together if the distance between their centers is shorter than the critical sticking distance $r_{eq}$. In our model aggregation is irreversible, meaning particles will not detach from one another after sticking.

3 Methods

We simulate diffusion-driven aggregation of particles embedded in quasi-2D fluids. We treat the fluid as continuous and particles as gaussian envelopes instead of having rigid edges. To reduce the computational power needed, we use an immersed boundary condition, where particles are treated as gaussian envelopes instead of solid particles as they would be in a no slip boundary condition. Either boundary condition is an assumption about the microscopic world, and at large
distances compared to particle size, both conditions predict the same fluid dynamics [1]. The immersed boundary condition reduces computational power as fluid particles are not allowed to “stick” to solute particles. The particles instead shadow the fluid flow beneath them, meaning the fluid velocity is weighted by the gaussian envelope of the particle over it, and averaged to give the velocity it imparts to the particle. We use a finite simulation cell with a periodic boundary condition.

However, our immersed boundary condition does not allow us to model hydrodynamic interactions between particles and fluid. Under immersed boundary conditions, the fluid can influence particles, but this does not allow particles to push back on the fluid. Instead, we use an approach borrowed from Camley and Brown [1], where hydrodynamic interactions are modeled using heptamers: a central particle surrounded by six evenly spaced particles each at equilibrium distance from another. Connections between particles are modeled as very stiff, overdamped springs, which allow the distance between connected particles to change slightly due to thermal forces. As thermal or hydrodynamic forces act on particles within a heptamer, the distance between connected particles decreases below or increases below or above the equilibrium distance, the springs compress or stretch. Now, the relaxation of these springs allow particles to exert a force on the fluid. The displacement of these spring-like connections is used to describe the elastic forces each heptamer exerts on the membrane as the springs return to equilibrium and generate forces in the fluid that can be felt by other particles.

The overall shape of the heptamer is disk-like, and does not deform due to the rigidity of the connections. Connections between particles forming an aggregate are also modeled as springs. Heptamers were chosen for their disk-like shape, and any additional particles would further slow down calculations. We assume that these radial forces combined with the forces generated from neighboring particles can accurately describe hydrodynamic forces.

The length of the timestep is determined by the relaxation time of the overdamped spring oscillators. In order to see the movement of springs, each timestep is short enough to view one tenth of the relaxation time of a spring. If a spring were to relax over a single timestep, we may miss the compressing or stretching of a spring, failing to model the hydrodynamic forces it produces.
If it is much shorter than a tenth of the relaxation time, we don’t gain much more information for the computational power needed for our simulation. At each timestep, particles are subject to hydrodynamic forces as well as thermal agitations.

3.1 Thermal agitations

Our simulation uses a stochastic model to thermally agitate particles. Stochastic forces follow the Gibbs-Boltzmann equilibrium distribution: \( P^{ss}(\{X\}) \sim \exp[-U(\{X\})/k_B T] \). We generate the thermal agitations in Fourier space, the magnitude follows a gaussian with variance chosen to satisfy the fluctuation-dissipation theorem\[1\]:

\[
\langle f_{i \text{therm}}(q,t) f_{j \text{therm}}(-k,t') \rangle = 2k_B T L^2(\eta_m q^2 + 2\eta_f q)\delta_{ij} \delta_{q,k} \delta(t-t').
\]

(2)

See appendix 6.1 for details. Before particles are agitated, we first check if any particles are within the critical sticking distance, or equilibrium distance.

3.2 Aggregation

The check to see if particles are within equilibrium distance, the adjacency check, was previously done by finding the shortest distances (over periodic boundaries) between all particles. This method is very costly computationally, and the neighbor list algorithm is a more efficient method. First, we construct a box extending the equilibrium distance on all sides around each particle, which we dub the neighbor box, and find the subset of particles contained within it. If this box extends past the simulation cell, we simply add another box over periodic boundary conditions. Now we only have to search the distances between particles that share a box, reducing computation time. However, we can decrease computation time further using a second box. Instead of recalculating which particles belong to the neighbor box at each timestep, we can calculate another box that extends a “skin distance” past the neighbor box every few timesteps, which we dub the skin box. This way, we only have to check if any particles within the skin box have entered the neighbor box each time step.

Figure 3: Aggregation of heptamers, heptamers of the same color belong to the same aggregate
The skin distance is determined from the magnitude of thermal agitations and the number of timesteps between calculations. We plan for two particles that move towards each other with two standard deviations above the mean thermal shift each timestep. It is possible this condition fails, as hydrodynamic forces may push particles together, although we expect them to generally slow the mutual approach between particles. It is possible that multiple particles within an aggregate will each experience a net force greater than what we accounted for. We assume these scenarios are unlikely enough to not significantly impact our findings.

3.3 Placing particles

Diffusing particles are placed randomly without overlapping. First, our simulation cell is divided into spaces with length equal to the equilibrium distance between particles. These are all of the potential spaces in which a particle can be placed without coming into contact with another particle. Depending on the set packing fraction, particles will be randomly placed in these spaces without repetition to meet the specified packing fraction. If the specified packing fraction can not be met exactly, it will be rounded down to the nearest possible value that can be satisfied.

![Initial placement of particles modeled as heptamers. Blobs belonging to the same heptamer are the same color. The initial angle each heptamer is placed at is randomized.](image)

4 Conclusion

The future goal of this project is to determine the aggregation rate as a function of the Saffman length, and the behavior of aggregation at the crossover between 2D and 3D regimes. We wish to determine the aggregation rate with and without hydrodynamic interactions. The aggregation rate can be measured using the method of Stankiewicz [4]: determining the fractal dimensionality. The fractal dimension can be obtained from the relationship between the radius of gyration and cluster mass. We plan to compare our results to the theoretical results in 3D, expecting a much larger dependence of aggregation rate on hydrodynamic interactions due to the slower spatial decay of forces in 2D. We also plan on comparing our results to experimental data of aggregation in quasi-2D films. Research at University of Colorado, Boulder [3] studies buckyball nanoparticle aggregation in
smectic crystal films, finding the fractal dimensionality of aggregates indicative of diffusion limited 2D aggregation. Our simulation results could be compared against experimental results like these to determine the legitimacy of our current models.

5 References

6 Appendix
6.1 The Z function
Thermal agitations are found using the Z function:

$$Z_j(q, \Delta t) = \int_t^{t+\Delta t} dt' f_j^{\text{therm}}(q, t')$$

(3)

The Z function is a complex-valued Gaussian random variable with zero mean and variance

$$\langle Z_i(q, \Delta t) Z_j(-k, \Delta t) \rangle = 2k_B T \mathcal{L}^2 (\eta_m q^2 + 2\eta_f q) \delta_{ij} \delta_{q,k} \Delta t.$$  

(4)

The $Z_i (i = x, y)$ function in Eq. (3) is generated in Fourier space. In discrete Fourier space, the $q$ variable takes on values $q = (m, n)2\pi/\mathcal{L}$ with $-N/2 < m, n \leq N/2$ ($N$ is assumed to be even). Since $Z_i(r, t)$ is real-valued, its Fourier transform must satisfy the condition $Z_i(q, \Delta t)^* = Z_i(-q, \Delta t)$. Indeed, we can express the thermal force field in terms of its Fourier transform:

$$f^{\text{therm}}(r) = \frac{1}{\mathcal{L}} \sum_q f^{\text{therm}}(q)e^{iq\cdot r} = \frac{1}{\mathcal{L}} \sum_q f^{\text{therm}}(-q)e^{-iq\cdot r},$$

(5)

where in the last equality we replaced the summation variable $q \rightarrow -q$. Since the variable $q$ runs over positive and negative values, this change of the variable does not affect the result of the summation. The complex-conjugated thermal force field is then given by

$$(f^{\text{therm}}(r))^* = \frac{1}{\mathcal{L}} \sum_q (f^{\text{therm}}(q))^* e^{-iq\cdot r}.$$  

(6)

Since the force field $f^{\text{therm}}(r)$ is real-valued, we require that $f^{\text{therm}}(r) = (f^{\text{therm}}(r))^*$. Therefore, from Eqs. (5) and (6), we arrive at the constraint for the Fourier transform of $f^{\text{therm}}$:

$$(f^{\text{therm}}(q))^* = f^{\text{therm}}(-q).$$

(7)
According to Eq. (3), the same condition applies to the $Z$ function, $Z_i(q, \Delta t)^* = Z_i(-q, \Delta t)$. The Fourier transform of $Z$ in a finite domain takes the form

$$Z_j(r, \Delta t) = \frac{1}{L^2} \sum_q Z_j(q, \Delta t)e^{iqr} \Rightarrow (8)$$

$$Z_j(r_{kl}, \Delta t) = \frac{1}{L^2} \sum_{m=-N/2+1}^{N/2} \sum_{n=-N/2+1}^{N/2} Z_j(q_{mn}, \Delta t)e^{i\frac{2\pi}{N}(m-k+n-l)}, \quad (9)$$

with

$$Z_j(q, \Delta t)^* = Z_j(-q, \Delta t) \Rightarrow Z_j^*(q_{mn}, \Delta t) = Z_j(q_{-m,-n}, \Delta t). \quad (10)$$

Due to the constraint, Eq. (10), not all modes of $Z_j(q_{mn}, \Delta t)$ are independent. In general, $Z_j(q_{mn}, \Delta t)$ is a complex number and we can write it as $Z_j(q_{mn}, \Delta t) = f + ig$. For $q_{mn}$ with $(m, n) = (0, 0), (N/2, 0), (0, N/2), (N/2, N/2)$, $Z_j(q_{mn}, \Delta t)$ is purely real with $g = 0$ and $\langle |f|^2 \rangle = 2k_BTL^2(\eta_m q^2 + 2\eta_f q)\Delta t$. For complex modes, $f$ and $g$ are selected from a distribution with variance $\langle |f|^2 \rangle = \langle |g|^2 \rangle = k_BTL^2(\eta_m q^2 + 2\eta_f q)\Delta t$.

### 6.2 Main code

```plaintext
%%% Aggregate w/ Periodic Boundary Conditions

% set(0,'defaultfigurecolor',[1 1 1]) % White boarder
% set(0,'defaultAxesFontSize', 18) % Bigger fonts
% set(0,'defaultfigureposition',[0 0 700 350]);
% format compact; % Compact output
clear all; close all; clc; % clean up workspace

%%% Physical parameters relevant to freely-suspended Smectic A liquid crystal

%%% films
NLayers = 3; %%% number of Smectic layers
OnelayerThickness = 3.17e-9; %%% thickness of one Smectic A layer (in meters)
etam = 0.052 * NLayers * OnelayerThickness; %%% membrane viscosity
  *in Pa.s.m* (3d viscosity of 8CB times the number of layers times
  the thickness of one layer)
etaf = 1.827e-5; %%% bulk fluid viscosity in *Pa . s* (air)
LSdimensional = etam/(2*etaf); %%% The Saffman length in meters
kBT = (1.38e-23) * 294; %%% at room temp (294 K), in *Joules = kg *
  m^-2/s^-2*
  % adimensional = 0.5 * LSdimensional; %%% the radius of a single
  blob
  % ldimensional = adimensional; %%% grid size
%%% Set the size of simulation cell (ten times the largest dimension )
```
Ldimensional = 10 * LSdimensional; %%% the size of simulation domain, *in m* along each axis (x- and y-)

%%% number of grid points ATTN: must be even number; number of cells (grid spacings) along each dimension

N = 22; %%% ATTN: must be even number; number of cells (grid spacings) along each dimension

ldimensional = Ldimensional/(N-1); %%% grid size

adimensional = ldimensional;

%%% Find (dimensional) spring constant and time step

[ksdimensional, DeltDimensional] = SpringConstTimeStep(adimensional, kBT, etam, LSdimensional);

%%% In the paper the authors use the spring constant equal to %%% k_sp = 200 pN/nm = 0.2 N/m
% ksdimensional = 0.2; %%% spring constant in N/m

%%% Scaled parameters

L = Ldimensional/LSdimensional; %%% the size of simulation domain scaled by the Saffman length

Delt = DeltDimensional * kBT/(etam * LSdimensional^2); %%% dimensionless time step

a = adimensional/LSdimensional; %%% the blob radius scaled by the Saffman length

l = ldimensional/LSdimensional; %%% grid spacing scaled by the Saffman length

ks = ksdimensional * LSdimensional^2/ kBT; %%% scaled spring constant

req = 2 * a; %%% scaled %%% distance between blobs in equilibrium

reff = req + a;

ks = ksdimensional * LSdimensional^2/ kBT; %%% scaled spring constant

TSteps = 140; %%% number of time steps

%%% Precompute qx and qy coordinates in the reciprocal q-space

Delq = 2*pi/L; %%% cell size in the reciprocal q-space, scaled by 1/ Ls

%%% Tij Calculation

qx = Delq * linspace(-N/2 + 1, N/2, N); %%% form a row of qx values

[Txx, Txy, Tyy, Qx, Qy, q] = MembraneTensor(qx, N);

% qy = qx; %%% also a row

%%% information about the meshgrid function:

%%% [X,Y] = meshgrid(x,y) returns 2-D grid coordinates based on the coordinates

%%% contained in vectors x and y. X is a matrix where each row is a copy of x,
and $Y$ is a matrix where each column is a copy of $y$. The grid represented by the coordinates $X$ and $Y$ has length($y$) rows and length($x$) columns.

$$[Qx, Qy] = \text{meshgrid}(qx, qy);$$

$$q = \sqrt{Qx.^2 + Qy.^2};$$

This is a matrix

$$%Response tensor scaled by \frac{1}{ls/(2*etaf)} = \text{meters/(Pa \times \text{seconds})}$$

$$Txx = \frac{1}{(q.^2 + q)} \cdot (1 - Qx .* Qx ./q.^2);$$

$$Tyy = \frac{1}{(q.^2 + q)} \cdot (1 - Qy .* Qy ./q.^2);$$

$$Txy = -\frac{1}{(q.^2 + q)} \cdot Qx .* Qy ./q.^2;$$

So, here $Txx(i, j) =$

set values of the response function for $q=0$ values

$$Txx(N/2, N/2) = 0;$$

$$Tyy(N/2, N/2) = 0;$$

$$Txy(N/2, N/2) = 0;$$

$q(N/2, N/2) = 0;$$

Define the simulation grid in real space

$$x = l \times \text{linspace}(-N/2 + 1, N/2, N);$$

$$y = l \times \text{linspace}(-N/2 + 1, N/2, N);$$

Calculate standard deviations, will be used later

$$SDcomplex = L \times \text{sqrt}(Delt \times (q.^2 + q));$$

$$SDreal = L \times \text{sqrt}(2 \times Delt \times (q.^2 + q));$$

Begining of the translational and rotational displacement code.

The blob coordinates as functions of time

$BlobCount = 20;$$

This creates initial conditions

$[xval, yval, HeptamerCount, xbinedges, ybinedges] = \text{PopulateCell\_Agg2}(l, L, N, reff, req, Bins, PF);$
color = ['#00FF00'; '#00FFFF'; '#FF00FF'; '#D95319'; '#0072BD'; '#EDB120'; '#77AC30'; '#FF00FF'; '#D95319'; '#77AC30'; '#0072BD'; '#7E2F8E'; '#7E2F8E'; '#0072BD'; '#DAAC40'; '#EDB120'; '#FF00FF'; '#D95319'; '#77AC30'; '#0072BD'; '#EDB120'; '#77AC30'; '#7E2F8E'; '#7E2F8E'; '#0072BD'; '#DAAC40'; '#B00B5E'; '#EDB120'; '#7E2F8E'];

S = zeros([TSteps, 2, BlobCount]); %%% WHAT IS "S"?

start=0;
for bb=1:HeptamerCount
  Blobs(1,1,start+1:bb*7)=xval(:, :, bb);
  Blobs(1,2,start+1:bb*7)=yval(:, :, bb);
  start=bb*7;
end

BlobsNoPeriodic = Blobs(1, : , :); %%% keep a grid of coordinates without shifting over BCs

%%% define the simulation grid in real space
xauxiliary = l * linspace(-N + 1, N, 2*N); %%% a row
yauxiliary = l * linspace(-N + 1, N, 2*N); %%% a row

AuxiliaryTheta = 0 : 0.01 : 2* pi; %%% to draw (blue) circles of radius a around the blob centers
StartX = a * cos(AuxiliaryTheta) + Blobs(1, 1, : );
StartY = a * sin(AuxiliaryTheta) + Blobs(1, 2, : );

% theta = 0 : 0.01 : 2* pi; %%% to draw (blue) circles of radius a around the blob centers
% StartX = a * cos(theta) + BlobsAndPeriodicImages(1, 1, : );
% StartY = a * sin(theta) + BlobsAndPeriodicImages(1, 2, : );

figure(1) %%%checking the initial conditions
for blobnumber = 1:BlobCount
  plot(Blobs(1, 1, blobnumber), Blobs(1, 2, blobnumber), 'k.', 'MarkerSize', 30);
  hold on
  plot(StartX(:, :, blobnumber), StartY(:, :, blobnumber), '-','LineWidth', 2);
end
axis([x(1) x(end) y(1) y(end)]);
daspect([1, 1, 1])
rectangle('Position', [x(1) y(1) x(end) -x(1) y(end) -y(1)])
title('Initial Positions')
xlabel('[x]')
ylabel('[y]')
hold off

%% Create Envelope Function for each particle.
%% Two possibilities here, use original location of each particle to
%% calculate the envelope function or just estimate at zero.
%% Size can be more than 3x3 when using location of blobs but is
%% always
%% 3x3 when doing it at origin

%% harder but more accurate
% for d = 1:BlobCount
% Del{d} = Envelope_functionvs_2(x, y, a, real(Blobs(1,1,d)) - round
% (real(Blobs(1,1,d))), ...
%       real(Blobs(1,2,d)) - round(real(Blobs(1,2,d)))
%     );
% end

%% this one is easier
EnvFunZero = Envelope_functionvs_2(x, y, a, 0, 0);
EnvFun = EnvFunZero .* ones(size(EnvFunZero), BlobCount);

connected = false;
[X, Y] = meshgrid(x, y);
[X1, Y1] = meshgrid(xauxiliary, yauxiliary);
size(Blobs(1, :, :)); %%% one row, two columns (for x- and y-
% coordinates) and blobCount slices (number of blobs)
f = 1;

% AdjOld=zeros(length(BlobCount),length(BlobCount));
SkinBox=zeros(length(BlobCount),length(BlobCount));
AdjBox=zeros(length(BlobCount),length(BlobCount));
SameAgg=zeros(length(BlobCount),length(BlobCount));
Mass=[];
Rad=[];

while f <= TSteps
    fprintf('time step f: \n')
    f
    %%% In EACH time step the array "Blobs" contains blobs that are
    %%% in the main cell
%% We expand this array and add periodic images of the blobs in the
%% neighboring cells

%% plot all the blobs: in the main cell and their images in the
%% neighboring cells
figure(2)
for blobnumber = 1:BlobCount
    plot(Blobs(f, 1, blobnumber), Blobs(f, 2, blobnumber), 'k.', 'MarkerSize', 30);
    hold all
    plot(Blobs(f, 1, blobnumber).*ones(size(AuxiliaryTheta)) + a * cos(AuxiliaryTheta), ...
    Blobs(f, 2, blobnumber).*ones(size(AuxiliaryTheta)) + a * sin(AuxiliaryTheta), 'b-', 'LineWidth', 2);
end
hold all
daspect([1,1,1])
axis([x(1)*2 x(end)*2 y(1)*2 y(end)*2]; %%% zoom in into a smaller region around the main cell
rectangle('Position',[x(1) y(1) x(end)-x(1) y(end)-y(1)])

% hold off

% connectedBlobs_vs5(L, BlobPositions, threshold, BlobCount,
% Skin_dist, TimeStep, delay, AdjBox)
%.0355 is roughly 2 std above mean thermal displacement

[Aggrnumb, clumpCount, adjacency, AdjBox, xindex, yindex] =
connectedBlobs_vs6(L, Blobs(f,:,,:), req, BlobCount, (.2844+
req), f, 4, AdjBox, AdjOld);

% this for loop will give a matrix indicating which blobs are in the same aggregate
for jj=1:length(unique(Aggrnumb)) % runs over aggregates
    AggDiff=unique(Aggrnumb); % gives all unique aggregate numbers
    include=find(Aggrnumb==AggDiff(jj)); % gives indices of all blobs within given aggregate
    SameAgg(include,include)=1; % SameAgg matrix will now have a "1" for blobs m and n in the same aggregate
end
delay = 5;
LJ_const1 = 0.005;
LJ_const2 = 0.0005;
Force_dist = 2.*req;
Skin_dist = 3.*req;

[ForceX, ForceY, SkinBox, nearest] = LJ_Force(L, Blobs(f,:,,:), f, delay, BlobCount, SkinBox, SameAgg, LJ_const1, LJ_const2, Force_dist, Skin_dist, req);

AdjNew = (adjacency - AdjOld);

SpringForceX, SpringForceY = ElasticForces_vs3(L, req, ks, Blobs(f,1,:), Blobs(f,2,:));

ElasticForces_Agg(L, req, ks, Blobs(f,1,:), Blobs(f,2,:), adjacency, BlobCount);

ReshapedSpringForceX = ones(size(EnvFun)) .* reshape(SpringForceX, [1,1, 7*HeptamerCount]);
ReshapedSpringForceY = ones(size(EnvFun)) .* reshape(SpringForceY, [1,1, 7*HeptamerCount]);

fpartX = EnvFun .* ReshapedSpringForceX;
fpartY = EnvFun .* ReshapedSpringForceY;

[IndX, IndY] = indexAgg(N, l, BlobCount, Blobs(f,:,:));
find the 3x3 area around each blob that the envelope function will multiply

premake NxN matrices that we will add all the forces to individually

to make the force field.

fbigX = zeros(N, N);
fbigY = zeros(N, N);

fbigLJX = zeros(N, N);
fbigLJY = zeros(N, N);

sum up all of the force fields of the blobs and add it to the N x N

matrix in the proper location

for g = 1:length(EnvFun)
    fbigX(IndX(:,g),IndY(:,g)) = fbigX(IndX(:,g),IndY(:,g)) + fpartX(:,g);
    fbigY(IndX(:,g),IndY(:,g)) = fbigY(IndX(:,g),IndY(:,g)) + fpartY(:,g);
% fbigLJX(IndX(:, :, g), IndY(:, :, g)) = fbigLJX(IndX(:, :, g), IndY(:, :, g)) + fLJpartX(:, :, g);
% fbigLJY(IndX(:, :, g), IndY(:, :, g)) = fbigLJY(IndX(:, :, g), IndY(:, :, g)) + fLJpartY(:, :, g);
end

%%% transform the force field to the Fourier space
Fourier_fpartX = l^2 * circshift(ifftshift(fft2(fftshift(circshift(fbigX, [1 1]))))), [-1, -1]); %%% N x N array
Fourier_fpartY = l^2 * circshift(ifftshift(fft2(fftshift(circshift(fbigY, [1 1]))))), [-1, -1]); %%% N x N array

%%% Calculate the fluid velocity field associated with stretching/compression
%%% of the springs
%%% the fluid velocity field in the Fourier space
Fourier_vX = Txx .* Fourier_fpartX + Txy .* Fourier_fpartY; %%% N x N array
%%% the fluid velocity field in real space
Real_vX = N^2/L^2 .* circshift(ifftshift(ifft2(fftshift(circshift(Fourier_vX, [1 1]))))), [-1, -1]); %%% N x N array

%%% the fluid velocity field in the Fourier space
Fourier_vY = Txy .* Fourier_fpartX + Tyy .* Fourier_fpartY; %%% N x N array
%%% the fluid velocity field in real space
Real_vY = N^2/L^2 .* circshift(ifftshift(ifft2(fftshift(circshift(Fourier_vY, [1 1]))))), [-1, -1]); %%% N x N array

%%% transform the force field to the Fourier space
Fourier_fLennardX = l^2 * circshift(ifftshift(fft2(fftshift(circshift(fbigLJX, [1 1]))))), [-1, -1]);
Fourier_fLennardY = l^2 * circshift(ifftshift(fft2(fftshift(circshift(fbigLJY, [1 1]))))), [-1, -1]);

%%% Calculate the fluid velocity field associated with Lennard Jones Interactions
%%% the fluid velocity field in the Fourier space
Fourier_vLennardX = Txx .* Fourier_fLennardX + Txy .* Fourier_fLennardY;
%%% the fluid velocity field in real space
Real_vLennardX = N^2/L^2 .* circshift(ifftshift(ifft2(fftshift(circshift(Fourier_vLennardX, [1 1]))))), [-1, -1]);
%%% the fluid velocity field in the Fourier space
Fourier_vLennardY = Txy .* Fourier_fLennardX + Tyy .* Fourier_fLennardY;
%%% the fluid velocity field in real space
Real_vLennardY = N^2/L^2 .* circshift(ifftshift(ifft2(fftshift(circshift(Fourier_vLennardX, [1 1]))))), [-1, -1]);
Fourier_fLennardY;
% %%% the fluid velocity field in real space
% Real_vLennardY = N^-2/L^-2 .* circshift(ifftshift(ifft2(fftshift(circshift(Fourier_vLennardY, [1 1])))),[-1, -1]);
%

%%% Stochastic Thermal Field, calculate once for a given time step
Zx = CalculateZ(N, SDcomplex, SDreal);
Zy = CalculateZ(N, SDcomplex, SDreal);

Gammax = circshift(Txx .* Zx + Txy .* Zy, [1 1]);
Gammay = circshift(Txy .* Zx + Tyy .* Zy, [1 1]);
Gamma_x_real = N^-2/L^-2 * circshift(ifftshift(ifft2(fftshift(Gammax))), [-1, -1]);
Gamma_y_real = N^-2/L^-2 * circshift(ifftshift(ifft2(fftshift(Gammay))), [-1, -1]);

%%% multiply the envelope function by the flow fields at the specific blobs location and sum it up. Will give a 1x7*HeptamerCount Matrix
for h = 1:length(EnvFun)
    SpringVelX(h) = sum(EnvFun(:, :, h) .* Real_vX(IndX(:, :, h), IndY(:, :, h)), [1, 2]);
    SpringVelY(h) = sum(EnvFun(:, :, h) .* Real_vY(IndX(:, :, h), IndY(:, :, h)), [1, 2]);

    LJVelX(h) = sum(EnvFun(:, :, h) .* Real_vLennardX(IndX(:, :, h), IndY(:, :, h)), [1, 2]);
    LJVelY(h) = sum(EnvFun(:, :, h) .* Real_vLennardY(IndX(:, :, h), IndY(:, :, h)), [1, 2]);

    ThermalVelX(h) = sum(EnvFun(:, :, h) .* Gamma_x_real(IndX(:, :, h), IndY(:, :, h)), [1, 2]);
    ThermalVelY(h) = sum(EnvFun(:, :, h) .* Gamma_y_real(IndX(:, :, h), IndY(:, :, h)), [1, 2]);
end

%%% compute blobs' displacements due to flow field generated by spring-like forces between the blobs in each heptamer
SpringShiftX(1, 1, :) = Delt * l^-2 * reshape(SpringVelX, [1, 1, 15]);
BlobCount]); %%% an array [1, 1, BlobCount] should maybe be [1,BlobCount]. changed old code
SpringShiftY(1, 1, :) = Delt * l^2 * reshape(SpringVelY, [1, 1, BlobCount]); %%% an array [7, 1, HeptamerCount]
%%% Replacing the integral from equation 10 in Camley and Brown with the summation given in equation 13
ThermalShiftX(1, 1, :) = (l^2) * reshape(ThermalVelX, [1, 1, BlobCount]); %%% an array [7, 1, HeptamerCount]
ThermalShiftY(1, 1, :) = (l^2) * reshape(ThermalVelY, [1, 1, BlobCount]); %%% an array [7, 1, HeptamerCount]

%%% new blobs positions at the end of the time step
Blobs(f, 1, :) = Blobs(f, 1, :) + SpringShiftX(:, 1, :) + ThermalShiftX(:, 1, :);% + ShiftX(:, 1, :);%
Blobs(f, 2, :) = Blobs(f, 2, :) + SpringShiftY(:, 1, :) + ThermalShiftY(:, 1, :);% + ShiftY(:, 1, :);%

%%% If, as a result of the thermal shift, the blob (in the main cell at the end of time step f) went outside
%%% the main cell, replace it with its periodic image in the main cell
HalfCellShift = (N/2-1)*l;
Blobs(f, 1, :) = mod(Blobs(f, 1, :) + HalfCellShift, L) - HalfCellShift;
Blobs(f, 2, :) = mod(Blobs(f, 2, :) + HalfCellShift, L) - HalfCellShift;

%%% calculate thermal translation and rotation of the clusters
%BlobsNoPeriodic=Blobs(f,:,:);

for c = 1:clumpCount %Loop over the aggregates

%%% "blobClump" is a subset of the blobs' coordinates for blobs that belong to the same aggregate with label c
ClumpSize = Blobs(f, :, Aggrnumb==c); %gives the x and y position of the clump
%size(blobClump)

%%% If some blobs in the aggregate are "hanging" outside the main
%%% simulation cell, we'll find periodic images of those blobs in the
%%% main cell since we will be using the envelope functions of only
the blobs in the main cell to calculate the thermal motion of
the entire aggregate; if all the blobs of the aggregate are
entirely in the main simulation cell, the mod operator below
will not change their coordinates

\[
\text{if size(ClumpSize, 3) == 1 \%%\ If the aggregate consists of a single blob.}
\]

\[
\text{Finding the closest grid point to the center of the blob}
\]
\[
\text{AlongX = round(real(Blobs(f,1,c))/l); \%%\ how many array elements along the x-axis}
\]
\[
\text{AlongY = round(real(Blobs(f,2,c))/l); \%%\ how many array elements along the y-axis}
\]
\[
\text{calculating the envelope function in the coordinate system where the origin is placed at a grid point closest to the center of the blob}
\]
\[
[Del] = \text{Envelope_function}(x, y, a, \text{real(Blobs(f,1,c)) - round(real(Blobs(f,1,c))}, ... \text{real(Blobs(f,2,c)) - round(real(Blobs(f,2,c)))});
\]
\[
[Del] = \text{Envelope_function}(x, y, a, \text{real(AggregateInMainCell(:, 1)) - round(real(AggregateInMainCell(:, 1))}, ... \text{real(AggregateInMainCell(:,2)) - round(real(AggregateInMainCell(:,2)))});
\]
\[
\text{rearrange the rows and columns of the envelope function calculated above to return to the original coordinate system in the main simulation cell}
\]
\[
\text{Del = circshift(Del, [AlongY AlongX])};
\]
\[
\text{Replacing the integral from equation 10 with the summation given in equation 13 don't need two sums here}
\]
\[ S_x = \text{real} \left( (l^2) \times \left( \sum_{i} \left( \text{EnvFun}(i,j,k) \times \Gamma_x_{\text{real}}(i,j,k) \right) \right) \right) \]

\[ S_y = \text{real} \left( (l^2) \times \left( \sum_{i} \left( \text{EnvFun}(i,j,k) \times \Gamma_y_{\text{real}}(i,j,k) \right) \right) \right) \]

%%% Updating Blob Positions

Blobs\( (f+1, :) \), c==\text{Aggrnumb} = \text{ClumpSize} + [S_x, S_y];

\%

\% Blobs\text{NoPeriodic}(1, :) = Blobs\text{NoPeriodic}(1, :) + [S_x, S_y];

\%

figure(2)

\%

plot(Blobs\( (f+1, 1) \), Blobs\( (f+1, 2) \), 'm.', 'MarkerSize', 30)

\%

%%% If, as a result of the thermal shift, the blob (in the main cell at the end of time step f) went outside the main cell, replace it with its periodic image in the main cell

HalfCellShift = (N/2-1)*l;

Blobs\( (f+1, :) \), Aggrnumb==c) = \text{mod}(Blobs\( (f+1, :) \), Aggrnumb ==c) + HalfCellShift, L) - HalfCellShift;

else % THIS PART IS For clusters that consist of more than one blobs

%%% Calculate the envelope functions centered at the blobs instantaneous positions

%%% Finding the closest grid point to the center of the blob

%%% TRANSLATIONAL DISPLACEMENT %%%

%%% Test -> form a loop over the blobs that belong to the same aggregate

NumberOfBlobsInAggr = size(ClumpSize, 3);

WhichBlobsInAggregate = find(Aggrnumb == c);

\%

AgriDel = zeros([N, N, NumberOfBlobsInAggr]);

AgriTherm = zeros([1,2,NumberOfBlobsInAggr]);

for i=1:NumberOfBlobsInAggr

\%

AgriAlongX = round(real(ClumpSize(1,1,i))/l); %%%

how many array elements along the x-axis

18
AgriAlongY = round(real(ClumpSize(1,2,i))/1);

how many array elements along the y-axis

[EnvFun] = Envelope_function(...
x, y, a, real(ClumpSize(1,1,i)) - round(real(
ClumpSize(1,1,i))), ...
real(ClumpSize(1,2,i)) - round(real(
ClumpSize(1,2,i)))...)

EnvFun = circshift(EnvFun, [AgriAlongY AgriAlongX ]);

AgriTherm(:, :, i) = [real(l^2 * (sum(sum(EnvFun(:, :,
WhichBlobsInAggregate(i)) .* Gamma_x_real (IndX
(:, :,WhichBlobsInAggregate(i)),IndY(:, :,
WhichBlobsInAggregate(i)))))),...
real(l^2 * (sum(sum(EnvFun(:, :,WhichBlobsInAggregate
(i)) .* Gamma_y_real (IndX(:, :,
WhichBlobsInAggregate(i)),IndY(:, :,
WhichBlobsInAggregate(i)))))),%%
translational displacement of ONE blob in the
aggregate along x and y directions

end

MeanTransDisplacement = mean(TranslDisplacement);

AgriTherm(:, :, i) = [real(l^2 * (sum(sum(EnvFun(:, :,
WhichBlobsInAggregate(i)) .* Gamma_x_real (IndX
(:, :,WhichBlobsInAggregate(i)),IndY(:, :,
WhichBlobsInAggregate(i)))))),...
real(l^2 * (sum(sum(EnvFun(:, :,WhichBlobsInAggregate
(i)) .* Gamma_y_real (IndX(:, :,
WhichBlobsInAggregate(i)),IndY(:, :,
WhichBlobsInAggregate(i)))))),%%
translational displacement of ONE blob in the
aggregate along x and y directions

AgriDel(:, :, i) = EnvFun;

MeanTransDisplacement = mean(TranslDisplacement);

AgriTherm(:, :, i) = [real(l^2 * (sum(sum(EnvFun(:, :,
WhichBlobsInAggregate(i)) .* Gamma_x_real (IndX
(:, :,WhichBlobsInAggregate(i)),IndY(:, :,
WhichBlobsInAggregate(i)))))),...
real(l^2 * (sum(sum(EnvFun(:, :,WhichBlobsInAggregate
(i)) .* Gamma_y_real (IndX(:, :,
WhichBlobsInAggregate(i)),IndY(:, :,
WhichBlobsInAggregate(i)))))),%%
translational displacement of ONE blob in the
aggregate along x and y directions

end

AgriTherm = [real(l^2 * (sum(sum(EnvFun(:, :,
WhichBlobsInAggregate)) .* Gamma_x_real (IndX(:, :,
WhichBlobsInAggregate),IndY(:, :,WhichBlobsInAggregate)))))),...]

real(l^2 * (sum(sum(EnvFun(:, :,
WhichBlobsInAggregate)) .* Gamma_y_real (IndX(:, :,
WhichBlobsInAggregate),IndY(:, :,WhichBlobsInAggregate)))))),%%
translational displacement of ONE blob in the
aggregate along x and y directions

end

AgriThermavg = mean(AgriTherm, 3);
AgriVtherm = AgriTherm ./ Delt; %average translational velocity

%AgriVthermMag = sqrt(dot(AgriVtherm, AgriVtherm, 2));
%%% speed
%
if size(ClumpSize, 3)==3
  abort
%
end
%%% ANGULAR DISPLACEMENT %%%
%%% Note: it is possible that some clusters extend beyond
%%% the main
%%% simulation cell. If this is the case, we need to find
%%% coordinates of the blobs (or possibly their periodic
%%% imagers)
%%% so that all blobs are "nearby" and form a cluster.

WhichBlobsInAggregate = find(Aggrnumb == c);
%%% mode will give the most common number in the matrix.
If the
%%% frequency of occurrence is the same it gives the
%%% smaller number
%%% Find the largest aggregate and which blobs were in
%%% it.
WhichBlobsInAggregatePrev = find(AggrnumbCheck == mode(
  AggrnumbCheck(WhichBlobsInAggregate)));
%%% aggregates in blob from previous time step
NumberOfBlobsInAggrPrev = length(
  WhichBlobsInAggregatePrev); %%% number of blobs in
aggregate previous tstep
NewBlobsInAggregate = WhichBlobsInAggregate(~ismember(
  WhichBlobsInAggregate,WhichBlobsInAggregatePrev));
%%% which blobs attached in current time step
WhichAggregatePrev = unique(AggrnumbCheck(
  NewBlobsInAggregate)); %%% number of different
aggregates that connected this time step

if NumberOfBlobsInAggr>NumberOfBlobsInAggrPrev %%% if
there are new blobs in aggregate

  %%%Removing this for Heptamers, no CM needed
  % for nn=1:length(WhichAggregatePrev) %loops over all new
aggregate(s)
  % NewBs= find(WhichAggregatePrev(nn)==AggrnumbCheck(
NewBlobsInAggregate)); % defines new blobs in aggregate
    NewBs=NewBlobsInAggregate(NewBs);
    [xb,yb]=find(AdjNew(NewBs,:)==1); % gives two columns of new connections within aggregate, each xb corresponds to a yb
    xb=NewBs(xb);
    
    for bb=1:length(xb) % loops over new connections in aggregate
        NewB=xb(bb); % defines single new blob
        Connects=find(AdjNew(NewB,:)==1); % gives index of new connection(s)
        
        for cc=1:length(Connects) % loops over all new connections of the new blob
            BlobC=Connects(cc); % index of connection to NewB
            AggC=AggrnumbCheck(BlobC); % gives aggregate number of the connecting blob
            OldAgg=AggrnumbCheck(NewB); % gives old agg num of the new blob
            
            xdist=BlobsNoPeriodic(1,1,BlobC)-BlobsNoPeriodic(1,1,NewB); % calculate distances between NewB and BlobC in nonperiodic coords
            ydist=BlobsNoPeriodic(1,2,BlobC)-BlobsNoPeriodic(1,2,NewB);
            
            xshift=sign(xdist).*gt(abs(xdist),(L/2)).*L; % determine if x dist is greater than L/2, will be shifted if so
            yshift=sign(ydist).*gt(abs(ydist),(L/2)).*L;
            
            TagAlong=find(AggrnumbCheck==OldAgg); % gives indexes of all blobs NewB was attached to, so they can be shifted as well
            
            BlobsNoPeriodic(1,1,TagAlong)=BlobsNoPeriodic(1,1,TagAlong)+xshift; % shifts coordinates of NewB and the rest of its agg
            BlobsNoPeriodic(1,2,TagAlong)=BlobsNoPeriodic(1,2,TagAlong)+yshift;
            
            AggrnumbCheck(TagAlong)=AggC; % redefine the aggregate that holds NewB, now if it is shifted again, this aggregate must shift as well
        end
    end
PeriodicCellNumber = zeros([NumberOfBlobsInAggr, 1]); %
  \%\% in which of the neighboring periodic cells the
blob is
CoordsShift = zeros([1, 2]);
end

PeriodicCellNumber(1) = 1; %\%\% in the main cell
Aggricm = mean(BlobsNoPeriodicCM, 1); %\%\% position of
the cluster's center of mass
HalfCellShift = (N/2-1)*l;
AggricmShift = mod(Aggricm + HalfCellShift, L) -
HalfCellShift;

figure(2)
for i=1:NumberOfBlobsInAggr
  plot(Blobs(f, 1, WhichBlobsInAggregate(i)), Blobs(f, 2, WhichBlobsInAggregate(i)), '.','Color', color(c,:), 'MarkerSize', 30)
  %plot(Blobs(f, 1, WhichBlobsInAggregate(i)), Blobs(f, 2, WhichBlobsInAggregate(i)), '.','Color',[c/BlobCount sin(2*pi*c/BlobCount).^10 1-c/BlobCount],
  'MarkerSize', 30)
end

plot(AggricmShift(1), AggricmShift(2), 'r.','MarkerSize', 30)

\%\%\% Positions of blobs w.r.t. the center of mass
blobsFromCM = BlobsNoPeriodicCM - Aggricm; \%\%\% "old"
blob coordinates with respect to the center of mass, before
rotation
distFromCM = sqrt(dot(blobsFromCM, blobsFromCM, 2));
\%\%\% Find angular velocity of each blob with respect to the
\%
\hspace{1cm} \text{cluster's center of mass due to random thermal forces}
\%
\hspace{1cm} \text{Information about the "cross" command}
\%
\hspace{1cm} \text{\texttt{C = cross(A,B,dim)} evaluates the cross product of arrays}
\%
\hspace{1cm} \text{\texttt{A} and \texttt{B} along dimension, \texttt{dim}. \texttt{A} and \texttt{B} must have the same size,}
\%
\hspace{1cm} \text{and both size(A,dim) and size(B,dim) must be 3.}
\%
\hspace{1cm} \text{\texttt{The dim input is a positive integer scalar.}}
\%
\hspace{1cm} \text{Before we apply the cross command, we need to add one more}
\%
\hspace{1cm} \text{column to the position and velocity arrays (the z-components)}
\%

\%
\hspace{1cm} \text{Blobs(f+1, :, Aggrnumb(1:BlobCount)==c) = ClumpSize +}
\%
\hspace{1cm} \text{permute(AgriThermavg, [3 2 1]) + permute(AgriRotShiftAvg, [3 2 1]);}
\%
\hspace{1cm} \text{Blobs(f+1, :, Aggrnumb(1:BlobCount)==c) = ClumpSize +}
\%
\hspace{1cm} \text{permute(AgriThermavg, [3 2 1]);}
\%
\%
\%
\%
\%

\%
\hspace{1cm} \text{BlobsNoPeriodic(1, :, WhichBlobsInAggregate) =}
\%
\hspace{1cm} \text{BlobsNoPeriodic(1, :, WhichBlobsInAggregate) + permute(}
\%
\hspace{1cm} \text{AgriThermavg, [3 2 1]) + permute(AgriRotShiftAvg, [3 2 1]);}
\%
\hspace{1cm} \text{If, as a result of the thermal shift, the blob (in the}
\%
\hspace{1cm} \text{main cell at the end of time step f) went outside}
\%
\hspace{1cm} \text{the main cell, replace it with its periodic image in the}
\%
\hspace{1cm} \text{main cell}
\%
\hspace{1cm} \text{HalfCellShift = (N/2-1)*1;}
\%
\hspace{1cm} \text{Blobs(f+1, :, Aggrnumb==c) = mod(Blobs(f+1, :, Aggrnumb}
\%
\hspace{1cm} \text{==c) + HalfCellShift, L) - HalfCellShift; \text{ \%\%ask}
\%
\hspace{1cm} kuriabova}
\%

\%
\hspace{1cm} \text{Compute fractal dimensionality}
\%
\hspace{1cm} \text{xrot=BlobsNoPeriodic(:,1,WhichBlobsInAggregate)-}
\%
\hspace{1cm} Aggricm(1);
\%
\hspace{1cm} \text{yrot=BlobsNoPeriodic(:,2,WhichBlobsInAggregate)-}
\%
\hspace{1cm} Aggricm(2);
\subsection{Placing particles}
\begin{lstlisting}[style=Matlab-editor]
function [BlobsX, BlobsY, HeptamerCount, xbinedges, ybinedges]... = PopulateCell(l, L, N, reff, req, Bins, PF)
  \%% This code will determine where the heptamers are placed and which
  \%% heptamers are bleached and not bleached

  \%% Input Parameters
  \%% l = grid spacing scaled by the Saffman length
  \%% L = the size of simulation domain scaled by the Saffman length
  \%% N = number of grid points along each dimension; must be even number
  \%% reff = scaled effective radius of entire heptamer
  \%% req = scaled distance between blobs in equilibrium
  \%% Bins = number of bins in each dimension
  \%% HeptamerCountinit = Initial number of heptamers
  \%% Dist = string of which distribution to use. Can either be 'Sine' or 'Gaussian'
\end{lstlisting}
Output Parameters

BlobsX = x-location of all heptamers and their blobs
BlobsY = y-location of all heptamers and their blobs
BleachInd = indices of which heptamers are bleached
NotBleachedInd = indices of which heptamers are not bleached
HeptamerCount = New heptamer count after distributing
xbinedges = x-location of the edges of the bins
ybinedges = y-location of the edges of the bins

BinCount = Bins^2; % total number of bins
BinLength = L/Bins; % Length of one bin

The code "UniformDistribute" will uniformly assigns points in bins that
the heptamers will be placed on.
[xval, yval, HeptamerCount, UniformCount, MaxHepRow] =
    UniformDistribute(BinCount, BinLength, reff, PF);

split the simulation into small bins
HalfBinShift = L/Bins/2; % half the length of a bin
xbins = HalfBinShift + l * linspace(-N/2 + 1, N/2 - N/Bins, Bins); %
        center of the bins
ybins = HalfBinShift + l * linspace(-N/2 + 1, N/2 - N/Bins, Bins); %
        center of the bins
xbinedges = l * linspace(-N/2 + 1, N/2, Bins+1);    % edges of the bins
ybinedges = l * linspace(-N/2 + 1, N/2, Bins+1);    % edges of the bins

make a matrix with the center of the bins
[xmat, ymat] = meshgrid(xbins, ybins);

depending on the "Dist" input either use a gaussian distribution
or
sinusoidal distribution.
if contains(Dist, 'Gaussian')
    A = 1; % amplitude
    B = UniformCount/HeptamerCount*pi/BinLength^2;
    sigma = ceil(sqrt(1/(2*B))); % standard deviation
    Z = A*exp(-((xmat-0).^2+(ymat-0).^2)/(2*sigma^2)); % gaussian distribution
else
    if contains(Dist, 'Sine')
        Z = .5*sin(3*pi/L*xmat)+.5; % sinusoidal distribution
    end
end
\[ Z = \text{UniformCount}; \] multiply by heptamer count in the bins so maximum is UniformCount
\[ Z = \text{round}(Z); \] round to the lower number
\[ \text{HeptamerCountBleach} = \text{sum}(Z, 'all'); \] count total number of bleached heptamers
\[ \text{Col} = Z(:); \] make Heptamer Count in each bin a column for easier use

\[ \text{randomly generate integer values without repeating numbers to place points} \]
\[ \text{in bins later and choose which heptamers to bleach. Store the indices} \]
\[ \text{as a cell array since the matrix changes size between bins.} \]
\[ \text{for } h = 1: \text{BinCount} \]
\[ \text{randint} = \text{randperm}(\text{MaxHepRow}^2, \text{UniformCount}); \] choose random integers without repeating to place in uniformly
\[ \text{BleachInd} = \text{randperm}(\text{length}(	ext{randint}), \text{Col}(h)); \] choose random integers without repeating to determine which are bleached
\[ \text{PosIndCell} \{h\} = \text{randint}; \] cell array with random positions to choose from
\[ \text{BleachIndCell} \{h\} = \text{BleachInd}; \] cell array with indices of heptamers to bleach
\[ \text{end} \]

\[ \text{will start in the top left bin and determine the bins vertically down} \]
\[ \text{then move repeat for each column. Takes the smaller edge values and} \]
\[ \text{adds on uniform distribution values. Determines location of the center} \]
\[ \text{blob of the heptamer} \]
\[ \text{count} = 1; \] initial heptamer number
\[ \text{HepInBin} = 0; \] total heptamer count in previous bins
\[ \text{for } d = 0: \text{Bins}-1 \] run over x-values
\[ \text{for } e = 1: \text{Bins} \] run over y-values

\[ \text{if } \text{cell2mat}(	ext{PosIndCell} \{\text{Bins}*d+e\}) > 0 \] make sure matrix is not empty
\[ \text{PosIndCellClean} = \text{cell2mat}(	ext{PosIndCell} \{\text{Bins}*d+e\}); \] make the cell into a matrix
\[ \text{for } f = 1: \text{length}(	ext{PosIndCellClean}) \]
\[ \text{BlobsX} \{1, 1, \text{count}\} = \text{xbinedges}(1+d) + \text{xval}(	ext{PosIndCellClean} \{f\}); \]
\[ \text{BlobsY} \{1, 1, \text{count}\} = \text{ybinedges}(\text{end}-e) + \text{yval}(	ext{PosIndCellClean} \{f\}) \]
count = count + 1;  \texttt{%%% next heptamer number}
end
end

\texttt{HepInBin = HepInBin + length(PosIndCellClean); \texttt{%%% add amount of heptamers in current bin}}
end
end

\texttt{NotBleachedInd = 1:HeptamerCount; \texttt{%%% make index for each heptamer}}
\texttt{%%% add 6 heptamers along the ring}

\texttt{ang=2*pi*rand(1,HeptamerCount*7);
for \texttt{ff} = 1:HeptamerCount
    \texttt{HeptamerAngles(ff,:)=linspace(ang(ff), ang(ff)+2*pi-pi/3, 6)';
    BlobsX(2:7, 1, ff) = BlobsX(1, 1, ff) + .99999.*req .* cos(HeptamerAngles(ff,:));
    BlobsY(2:7, 1, ff) = BlobsY(1, 1, ff) + .99999.*req .* sin(HeptamerAngles(ff,:));
end}
end

\texttt{\subsection{Elastic Forces}\begin{lstlisting}[style=Matlab-editor]
%%% Edited on 08/08/2022
%%% This function calculates spring forces acting on each blob in each heptamer
%%% Input parameters: req = scaled spring length in equilibrium
%%% ks = scaled spring constant
%%% BlobsX is an array of dimensions [7, HeptamerCount]
%%% each column corresponds to a separate heptamer
%%% rows -- x-coordinates of 7 blobs in each heptamer
%%% Similarly, BlobsY = y-coordinates of the blobs
\end{lstlisting}
Output parameters: arrays of size [7, HeptamerCount] with X- and Y-
components of the forces on each blob

Note: Blob 1 is one in the center of the heptamer

Determine stretching /compression of springs along the perimeter

X- and Y- components of springs' displacements; the array length is

circshift(Xpositions, -1) moves the first element to the last position

in the column

BlobsX = repmat(BlobsX(:,)', [BlobCount, 1]);
BlobsY = repmat(BlobsY(:,)', [BlobCount, 1]);
BlobsX=BlobsX.*adjacency; % now we have all connections
BlobsY=BlobsY.*adjacency;

DeltaX = BlobsX' - BlobsX; % Now all positions outside the skin box
and within the same aggregate are ignored
DeltaY = BlobsY' - BlobsY;

Xindex = -sign(DeltaX).*floor(abs(DeltaX)/(L/2)); % Distance may be
larger than L/2, would need to be shifted if so
Yindex = -sign(DeltaY) .*floor(abs(DeltaY)/(L/2));

DeltaX = Xindex .* L + DeltaX; % select the shortest distance
DeltaY = Yindex .* L + DeltaY;

Calculate spring displacements

SpringDisplacement = sqrt(DeltaX.^2 + DeltaY.^2) - req; %%

Check: should be a small number

Percentage1 = SpringDisplacement/req * 100

angle = atan2(real(DeltaY), real(DeltaX)); % output is the angle
in the range between -pi and + pi.

FSpringXPart1 = ks * SpringDisplacement.* cos(angle); %% the x-
component of the force on the ith blob due to the (i-1)th blob (3
on 2, 4 on 3, ..., 7 on 6, 2 on 7)
FSpringYPart1 = ks * SpringDisplacement.* sin(angle); %% the y-
component of the force on the ith blob due to the (i-1)th blob
NetForceOnBlobY = [sum(FSpringYToCenter); NetForceOnBlobY - FSpringYToCenter];

NetForceOnBlobX = sum(FSpringXPart1.*adjacency);
NetForceOnBlobY = sum(FSpringYPart1.*adjacency);
end

\subsection{Thermal agitations}
\begin{lstlisting}[style=Matlab-editor]
function Z = CalculateZ(N, SDcomplex, SDreal)

    % Z = zeros([N, N]);
    freal = normrnd(0, SDcomplex);
    gimag = normrnd(0, SDcomplex);

    Z = freal + 1i * gimag;

    %%% Now we need to correct some values of Z since Z has to satisfy a condition that its Fourier transform is a purely real number.
    %%% Therefore, Z^*(\vec{q}) = Z(-\vec{q}). See notes or Camley and Brown paper
    %%% for details.
    
    %%% A q-vector is determined by a pair of integers (m,n) (times the constant Delq)
    %%% the first index in Zx corresponds to integer m (qx) (rows), the second
    %%% index corresponds to integer n (qy) (columns)
    %%% purely real values of Z in the "corners" with (m, n) = (0,0), (N/2, 0), (0, N/2), (N/2, N/2)  
    %%% (m,n) = (0,0)
    %%% note: the relation between the index number for the array Z and the
    %%% actual values of the integers (m, n) is index_m = m + N/2 and
    %%% index_n = n + N/2
    Z(N/2, N/2) = normrnd(0, SDreal(N/2, N/2));  %%% (m,n) = (0,0)
    Z(N/2, N) = normrnd(0, SDreal(N/2, N));    %%% (m,n) = (0,N/2)
    Z(N, N/2) = normrnd(0, SDreal(N, N/2));    %%% (m,n) = (N/2,0)
    Z(N, N) = normrnd(0, SDreal(N, N));        %%% (m,n) = (N/2,N)
\end{lstlisting}
%%% Point on the horizontal (m) axis
Z(1:N/2-1, N/2) = conj(Z([N-1:-1:N/2+1], N/2));

%%% Periodic boundary conditions, special cases:
Z(1:N/2-1, N) = conj(Z([N-1:-1:N/2+1], N));
Z(N, 1:N/2-1) = conj(Z(N, [N-1:-1:N/2+1]));

%%% All other dependent points
Z(1:N-1, (N/2-1:-1:1)) = conj(Z([N-1:-1:1], N/2 + 1: N-1));

end

\subsection{Adjacency code}
\begin{lstlisting}[style=Matlab-editor]
function [comps, compCount, adjacency, AdjBox, Xindex, Yindex] =
    connectedBlobs_vs5(L, BlobPositions, threshold, BlobCount, Skin_dist, TimeStep, delay, AdjBox, adjacency)

    \% this if statement will compute the "skin box" every few time steps to
    \% indicate which blobs may form aggregates
    AdjOld=adjacency;

    if ~mod(TimeStep, delay)==1 | TimeStep==1
        \% runs only for the first time step or once every in every "delay" timesteps
        BlobPositionsX = BlobPositions(:,1,:);
        BlobPositionsY = BlobPositions(:,2,:);

        SkinL=BlobPositionsX(:,end)-Skin_dist; \% defines bounds of outermost
            skin box for each blob
        SkinR=BlobPositionsX(:,1)+Skin_dist;
        SkinT=BlobPositionsY(:,end)+Skin_dist;
        SkinB=BlobPositionsY(:,1)-Skin_dist;

        BlobPositionsX = repmat(BlobPositionsX(:,end), [BlobCount, 1]); \%
            blobcount by blobcount matrix of x positions
        BlobPositionsY = repmat(BlobPositionsY(:,end), [BlobCount, 1]);

end
\end{lstlisting}
Xindex = -sign(SkinL).*floor(abs(SkinL)/(L/2))-sign(SkinR).*floor(abs(SkinR)/(L/2)); % determines if box bounds exceed cell, and in which direction
Yindex = -sign(SkinT).*floor(abs(SkinT)/(L/2))-sign(SkinB).*floor(abs(SkinB)/(L/2)); % will give row of -1 0 or 1 depending on shift direction

Xshift = Xindex.*L; % again, a row of shifts for each blob's skin box
Yshift = Yindex.*L;

AdjBox = (BlobPositionsX <= SkinR & BlobPositionsX >= SkinL
 & BlobPositionsY <= SkinT & BlobPositionsY >= SkinB); % [blobcount, blobcount] array gives 1 if blob m is in the box of blob n, 0 otherwise

AdjBoxPeriodic = (BlobPositionsX <= SkinR + Xshift & BlobPositionsX >= SkinL + Xshift & BlobPositionsY <= SkinT + Yshift & BlobPositionsY >= SkinB + Yshift); % same as last, but with the potentially shifted box

AdjBox = or(AdjBox, AdjBoxPeriodic); % combines periodic and non periodic boxes to include all neighboring blobs

end

% begin here if using old SkinBox

BlobPositionsX = BlobPositions(:,1,:);
BlobPositionsY = BlobPositions(:,2,:);

BlobPositionsX = repmat(BlobPositionsX(:,1), [BlobCount, 1])
BlobPositionsY = repmat(BlobPositionsY(:,2), [BlobCount, 1])

BlobPositionsX = BlobPositionsX .* AdjBox .* ~ AdjOld; % ignores all blobs not within skin box

BlobPositionsY = BlobPositionsY .* AdjBox .* ~ AdjOld;

%% This is excluded for heptamers, as they can bend
% BlobPositionsX = BlobPositionsX .* ~(SameAgg - adjacency); % this excludes positions for blobs within the same aggregate to speed up calculations
% BlobPositionsY = BlobPositionsY .* ~(SameAgg - adjacency); % we exclude the adjacency matrix, otherwise previously connected blobs would be disconnected

DistX = BlobPositionsX' - BlobPositionsX; % Now all positions outside the skin box and within the same aggregate are ignored

DistY = BlobPositionsY' - BlobPositionsY;
Xindex = -sign(DistX).*floor(abs(DistX)/(L/2)); \%Distance may be larger than L/2, would need to be shifted if so
Yindex = -sign(DistY).*floor(abs(DistY)/(L/2));

DistX = Xindex.*L + DistX; \% select the shortest distance
DistY = Yindex.*L + DistY;

nearest=(abs(DistX)<=threshold) & (abs(DistY)<=threshold) & (abs(DistX)|abs(DistY))>0; \%returns only blobs within a box of req and larger than 0 (not the same blob)

DistX=DistX.*nearest; \%ignore all distances outside req
DistY=DistY.*nearest;

distances = sqrt(DistX.^2 + DistY.^2); \%find total distance for blobs within req

\% check to see if there are negative distances
\% adjacency is a logical NxN symmetric matrix. For example, if the element with indices (10,3) is "True", it means that blobs 10 and 3 are "connected".
adjacency = (distances <= threshold + 1e-6) & (distances > 0); \%TK the second condition eliminates the possibility of a blob to be connected to itself.
adjacency = (distances <= threshold) & (distances > 0); \%TK the second condition eliminates the possibility of a blob to be connected to itself.
adjacency = round((adjacency+AdjOld)./2); \% this makes sure there can't be any previously connected blobs that disconnect

\%triu(adjacency); \%keep only the upper right corner, to avoid identical pairs of connected blobs (i.e, (2,4) and (4,2))

\%[row, column] = find(triu(adjacency)) \%the output is two columns with indices of, respectively, rows and columns of blobs that are connected by "edges"

\% G = graph(A) creates a graph using a square, symmetric adjacency matrix, A.
\% For logical adjacency matrices, the graph has no edge weights.
\% For nonlogical adjacency matrices, the graph has edge weights.
The location of each nonzero entry in A specifies an edge for the graph, and the weight of the edge is equal to the value of the entry. For example, if A(2,1) = 10, then G contains an edge between node 2 and node 1 with a weight of 10.

Adjacency matrix, specified as a full or sparse, numeric matrix. The entries in A specify the network of connections (edges) between the nodes of the graph. The location of each nonzero entry in A specifies an edge between two nodes. The value of that entry provides the edge weight. A logical adjacency matrix results in an unweighted graph. Nonzero entries on the main diagonal of A specify self-loops, or nodes that are connected to themselves with an edge. Use the 'omitselfloops' input option to ignore diagonal entries. A must be symmetric unless the type input is specified. Use issymmetric to confirm matrix symmetry. For triangular adjacency matrices, specify type to use only the upper or lower triangle. Example: A = [0 1 5; 1 0 0; 5 0 0] describes a graph with three nodes and two edges. The edge between node 1 and node 2 has a weight of 1, and the edge between node 1 and node 3 has a weight of 5.

coms = conncomp(graph(adjacency));

bins = conncomp(G) returns the connected components of graph G as bins. The bin numbers indicate which component each node in the graph belongs to. If G is an undirected graph, then two nodes belong to the same component if there is a path connecting them.

compCount = max(comps);

this might be slow for larger systems
I=1;
end