Large-scale dynamics of self-propelled particles moving through obstacles: Analysis of Patterns in 2D

P. F. Valsecchi Oliva

September 20, 2020

Abstract

This paper is an attempt to build upon the work done by Manhart et al. [1] and extend the analysis and simulations to 2 dimensions. In 2 dimensions we see a variety of patterns emerging, including moving clusters, trails and travelling bands. This motivates an exploration into a partial differential equation based analysis, as the model itself — with the large number of coupled ordinary differential equations — does not yield any insight into the behaviours we would expect. The motivation of this paper is to attempt to confirm that in the simulation of the interaction of self-propelled particles (SPPs) and tethered obstacles, we can accurately predict the size of the patterns that may be exhibited. To do this we look at a macroscopic partial differential equation, and linear stability analysis reveals to us the conditions under which patterns emerge and what factors affect pattern size.

1 Introduction

1.1 Motivation and Aims

The purpose of this paper is to simulate the interaction of SPPs, which behave as they would in the Vicsek model, and tethered obstacles in 2 dimensions. The behaviour of this model (defined in Sec. 2.1) yields varying macroscopic behaviours depending on the parameters of the micro-scale interactions, motivating us to explore the possibility of predicting the behaviour, or pattern size more specifically, of the SPPs. Examples of the various behaviours of collective dynamics occur in school of fish (milling), swarming in some species of birds, aggregation and patterns in bacterial and single-celled organism movement, as well as, lane formation in people [2]. In all these cases we observe large scale patterns emerging from local, uncoordinated interactions that exhibit non-local effects. The inclusion of obstacles in this model is to allow for the emergence of these collective behaviours, as oftentimes they will only emerge when we take into account the resistance to movement by the environment (an example of this being the formation of paths in grasslands by large mammals, where they follow each others' paths for less resistance [2]).

The need for a prediction of these behaviours emerges from the difficulty of predicting or interpreting the macro-scale behaviour of SPPs and obstacles from the model, formed by thousands of coupled ordinary differential equations (ODEs). To this end, we investigate the possibility of using the partial differential equations (PDEs) that were derived by Manhart et al. [1], in 2 dimensions. Reframing the problem of predicting pattern size in this form allows us to perform analysis and approximate the behaviour of the model for sufficiently large time periods. The purpose of this paper is then to identify whether, independently of the stochastic processes built in the model and independently of the initial, uniformly and randomly distributed positions of SPPs and obstacles, we can approximate correctly the pattern size that should be observed. Thus, giving us the ability to analyse and determine the effects of changing variables on the macro effects that we observe, to allow us to better understand the processes that underly uncoordinated collective behaviour.

1.2 Summary

The paper introduces the model in Sec. 2.1 based on the Vicsek model and taken from [1], to include obstacles and stochastic processes. Then in Sec. 2.2 and 2.3 two different approaches to the numerical approach are introduced and then compared for efficiency in Sec. 2.4, where we determine that the convolution based

numerical implementation is superior to the other approach in computational efficiency and so we proceed to use this approach to model the behaviour of the model and compare it to the analytical insight we gain from Sec. 3. In this latter section we analyse the density-based PDE form [1], known as the macromodel, and look at the predictions it makes about the effects of the variables on the ability for the model to form patterns and the effect they have on pattern size. In Sec. 4 we compare the predictions of the macromodel to the model and reveal that it can accurately give an indication of the pattern size we expect to see. Due to this, the observations of the implications of the macromodel, made in Sec. 3, can be transferred to the model and its collective behaviour. Finally, in Sec. 5 we discuss the implications of Sec. 3 and 4 and show that we have gained some insight into how varying the variables of the local interactions affects the overall behaviour of the model on a macro scale.

2 The Individual-Based Model (IBM)

2.1 IBM Formulation

We start the investigation by looking at the individual motion of self-propelled swimmers (SPPs) coupled with tethered obstacles, as described in [1]. This model combines the eclectic Vicsek model for collective motion of SPPs [3] combined with the aforementioned obstacles. We look at this motion in 2D. The interactions are shown schematically in Fig. 1. The regime explored here is an over-damped one, therefore friction dominates.

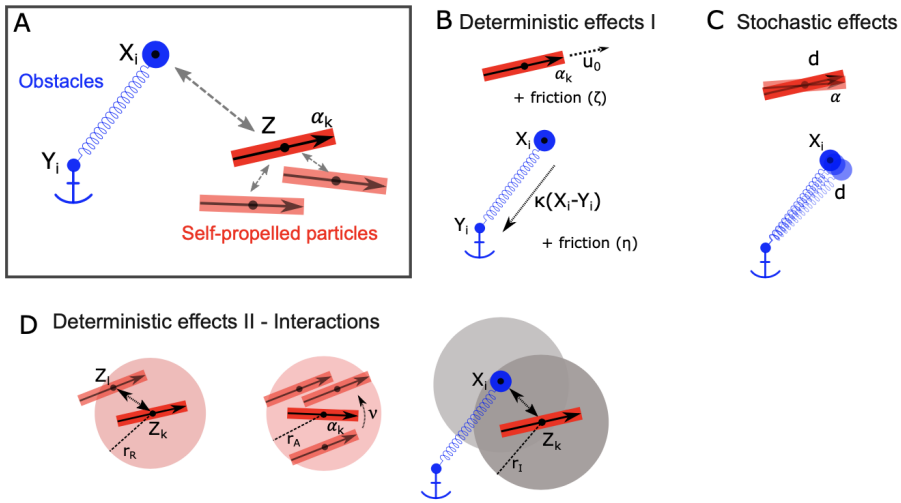


Figure 1: Elements of the IBM. A: Shown are the two agents modelled in the IBM, the SPPs in red, whilst the obstacles together with their tethers in blue. B: The deterministic effects that are modelled independently of the presence of other SPPs or obstacles. SPPs propelling themselves at constant velocity u_0 in direction α_k and experiencing friction ζ. The obstacle experiencing the Hookean spring that tethers them to their anchor point Y_i with stiffness constant κ and environmental friction η. C: The stochastic effects on the orientation of SPPs α and on the obstacle's position. D: Interactions of SPP-SPP repulsion in radius r_R, SPP alignment with neighbours in radius r_A with alignment frequency ν and SPP-obstacle repulsion in a neighbourhood r_I. (taken from [1])

Components of the Model. The IBM models the behaviour of the following agents:

Obstacles: We consider N moving obstacles defined by positions $X_i(t) \in \mathbb{R}^2$ for $i \in \{1, 2, ..., N\}$ and time $t \geq 0$. These obstacles are each tethered to a fixed anchor $Y_i \in \mathbb{R}^2$ through a Hookean

spring with stiffness constant $\kappa > 0$ and experience friction with friction constant $\eta > 0$.

SPPs: We consider M SPPs defined by positions $Z_k(t) \in \mathbb{R}^2$ for $k \in \{1, 2, ..., N\}$ and time $t \geq 0$. These SPPs have an orientation $\alpha_k(t) \in \mathbb{S}^1$ and a self-propulsion speed u_0. The SPPs experience friction with friction constant $\zeta > 0$.

Interactions. The following interactions are modelled:

SPP alignment: We have each SPP align its orientation α_k to the mean orientation $\bar{\alpha}_k$ of SPPs in a neighbourhood of radius r_A. This occurs with an alignment frequency ν as in the Vicsek model [3].

SPP repulsion: SPPs repel each other locally to prevent clustering beyond certain densities. We model this with an axisymmetric pushing potential $\psi : \mathbb{R}^2 \mapsto \mathbb{R}$ with radius of interaction $r_R > 0$. The force exerted on each other by two swimmers positioned at Z_i and Z_j is given by $\nabla\psi(Z_i - Z_j)$.

Obstacle-SPP interaction: SPPs and obstacles repel each other. We model this with an axisymmetric pushing potential $\phi : \mathbb{R}^2 \mapsto \mathbb{R}$ with radius of interaction $r_I > 0$. The force exerted on a swimmer and an obstacle positioned at X_i and Z_j respectively is given by $\nabla\phi(X_i - Z_j)$.

Stochasticity. In the model we include two different forms of uncertainty, modelled by Brownian motions: once in the obstacle positions (with intensity d_o) and in the SPP orientation (with intensity d_s). This is to prevent that the model entrenches itself in particular solutions and continues to experience fluctuations to yield broader behaviours on a macro scale.

Model Equations. The aforementioned effects can be modelled with the following, coupled, stochastic ODEs. Note we are using the same notation as above and that the positions of the obstacles and SPPs are still given in two spatial dimensions. We also introduce a constant velocity component u_0 that determines the velocity of the SPPs motion in the absence of forces. These equations are to be considered on a periodic domain of size L to allow for the approximation of the behaviour of these SPPs on an infinite domain with SPP density M/L^2 and obstacle density N/L^2.

$$\mathrm{d}X_i = -\frac{\kappa}{\eta}(X_i - Y_1)\mathrm{d}t - \frac{1}{\eta}\frac{1}{M}\sum_{k=1}^{M}\nabla\phi(X_i - Z_k)\mathrm{d}t + \sqrt{2d_o}\mathrm{d}B_t^i \tag{1}$$

$$\mathrm{d}Z_k = u_0\alpha_k\mathrm{d}t - \frac{1}{\zeta}\frac{1}{N}\sum_{i=1}^{N}\nabla\phi(Z_k - X_i)\mathrm{d}t - \frac{1}{\zeta}\frac{1}{M}\sum_{l\neq k}^{M}\nabla\psi(Z_k - Z_l)\mathrm{d}t \tag{2}$$

$$\mathrm{d}\alpha_k = P_{\alpha_k^\perp} \circ \left[\nu\bar{\alpha}_k\mathrm{d}t + \sqrt{2d_s}\mathrm{d}\tilde{B}_t^k\right] \tag{3}$$

where α_k is the mean direction defined by the mean flux J_k as follows

$$\bar{\alpha}_k = \frac{J_K}{|J_K|} \qquad \text{where } J_k = \sum_{\substack{j=1 \\ |Z_k - Z_j| \leq r_A}}^{M}\alpha_j \tag{4}$$

The original tether positions Y_i are uniformly distributed in a random fashion and the obstacles X_i begin in those positions. The operator $P_{\alpha_k^\perp}$ in (3) is an orthogonal projection onto $\alpha_k^\perp$ and ensures that $\forall t \geq 0 \quad \alpha_k(t) \in \mathbb{S}^1$.

Interaction Kernels. As discussed above the SPP-obstacle and SPP-SPP interactions are determined by the interaction kernels ϕ and ψ. For this model it was chosen that their shape be axisymmetric and have a repulsive force proportional to the proximity to the position of the agent exerting the force. The kernels

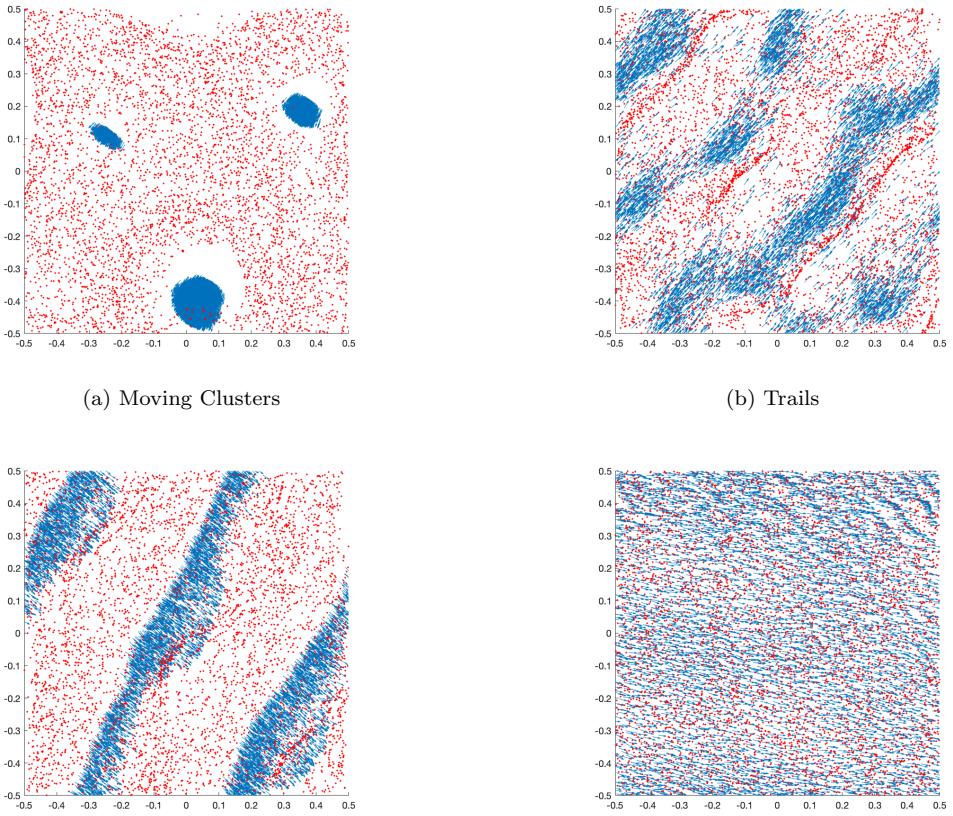
are compactly supported on balls with radii r_I and r_R respectively. They take the following shape

$$\phi(x) = \frac{3A_I}{2r_I^3\pi}(r_I - |x|)^2 H(r_I - |x|) \tag{5}$$

$$\psi(x) = \frac{3A_R}{2r_R^3\pi}(r_R - |x|)^2 H(r_R - |x|) \tag{6}$$

where $H(x)$ is the Heaviside function. The interaction kernels have been normalised such that the force masses are A_I and A_R respectively.

Remark. *For simplicity in this paper we choose to set $r_R = r_A$ and we leave the following parameters constant throughout: $d_o = 0$, $\eta = 1$, $d_s = 0.1$, $u_0 = 1$, $L = 1$, $M = N = 5000$, $\zeta = 1$ and $A_I = \pi$. So that the following parameters are left to be set: κ, ν, r_I, r_R and A_R.*



(a) Moving Clusters

(b) Trails

(c) Travelling Bands

(d) Uniform Distribution

Figure 2: IBM simulations. Depicted are the final frames of the IBM simulation (at t=30) with the parameters mentioned in the remark, as well as, setting $\kappa = 100$, $\zeta = 1$, $\nu = 10$, $r_R = 0.05$ and varying A_R and r_I. The SPPs are depicted as blue arrows, pointing in the direction of the motion of the SPP α and the obstacles are depicted as red diamonds. We see the following macro-scale behaviours: moving clusters (a) ($r_I = 0.3$ and $A_R = 4/30$), trails (b) ($r_I = 0.2$ and $A_R = 0.6$), travelling bands (c) ($r_I = 0.3$ and $A_R = 0.4$), uniform distribution (d) ($r_I = 0.3$ and $A_R = 64/30$). The relative strength of the SPP-SPP interaction (determined by A_R) increases as we proceed from (a) through (d) and this gives rise to the varying patterns. As this interaction increases in strength locally, relative to the SPP-obstacle interaction, we get a decrease in the local density of the SPPs, as we would expect.

In Fig. 2 we see that small variations in the parameters yield drastically varying pattern shapes and pattern sizes. We see that varying the force mass A_R of the SPP-SPP interaction kernel ψ produces the

different behaviours discussed in the introduction, such as the clustering of single-celled organisms (Fig. 2 (a)), the creation of trails in grasslands (Fig. 2 (b)), wave-like behaviour in sperm (Fig. 2 (c)) or no pattern whatsoever (Fig. 2 (d)). These observations are the principal motivator for the analysis done in the rest of the paper, to allow us to predict, without the need to simulate, what parameters will yield which pattern sizes.

2.2 Neighbour Based Numerical Approach

For the implementation of the coupled ODEs (1), (2) and (3), we define the sequences $\langle X_i^t \rangle_{t\in T}$ and $\langle Z_k^t \rangle_{t\in T}$, where t is the time step we compute the position at, as we cannot solve the coupled ODEs explicitly. $X_i^{t_0}$ and $Z_k^{t_0}$ are the initial positions. This means that at each time step we compute the forces exerted on each of the agents and then have the agents move accordingly. The choice of T will be discussed below, but we will consider it as $T = \{t_n\}_{n=0}^{\infty}$.

The following equations define these sequences such that they satisfy the ODEs

$$X_i^{t_{n+1}} = X_i^{t_n} - \left(\frac{\kappa}{\eta}(X_i^{t_n} - Y_1) + \frac{1}{\eta}\frac{1}{M}\sum_{k=1}^{M} \nabla\phi(X_i^{t_n} - Z_k^{t_n}) \right) |t_{n+1} - t_n| + \sqrt{2d_o}\mathrm{d}B_{t_n}^i \tag{7}$$

$$Z_k^{t_{n+1}} = Z_k^{t_n} + \left(u_0\alpha_k^{t_n} - \frac{1}{\zeta}\frac{1}{N}\sum_{i=1}^{N} \nabla\phi(Z_k^{t_n} - X_i^{t_n}) - \frac{1}{\zeta}\frac{1}{M}\sum_{l\neq k}^{M} \nabla\psi(Z_k^{t_n} - Z_l^{t_n}) \right) |t_{n+1} - t_n| \tag{8}$$

$$\alpha_k^{t_{n+1}} = \alpha_k^{t_n} + P_{\alpha_k^{\perp}} \circ \left[\nu\bar{\alpha}_k^{t_n} |t_{n+1} - t_n| + \sqrt{2d_s}\mathrm{d}\tilde{B}_{t_n}^k \right] \tag{9}$$

where we maintain (4) but take into account the positions and angles at each time step. The stochastic process $B_{t_n}^i$ and $\tilde{B}_{t_n}^k$ are sampled from standard normal distributions for each agent at each timeframe. For the purpose of simplicity we will not use the superscript t_n in the rest of the paper, but it is assumed that when we refer to an agent, X_i of Z_k, we are evaluating this at a position $t \in T$.

Considering the importance of the proximity of different agents to each other, an intuitive approach to modelling the behaviour of the model is to use a neighbour based numerical approach. This is due to the alignment being dependent on local swimmer orientations and the interaction kernels having a compactly supported force on balls of radius $r_I, r_R < L$. To calculate the interactions we search for all the neighbours of each agent and then use this to determine the force each of these exerts on the initial agent, summing these to attain the overall force exerted. This is illustrated below.

Calculating the Forces. The forces exerted on each of the particles are determined by a combination of (1), (2), (3) and the interaction kernels to form the following compactly supported sums

$$\frac{1}{M}\sum_{k=1}^{M} \nabla\phi(X_i - Z_k) = \frac{1}{M}\sum_{k\in B_i^1} \left(\frac{3A_I}{r_I^3\pi} \left(\frac{r_I}{|X_i - Z_k|} - 1 \right) (X_i - Z_k) \right) \tag{10}$$

$$\frac{1}{N}\sum_{i=1}^{N} \nabla\phi(Z_k - X_i) = \frac{1}{M}\sum_{i\in B_k^2} \left(\frac{3A_I}{r_I^3\pi} \left(\frac{r_I}{|Z_k - X_i|} - 1 \right) (Z_k - X_i) \right) \tag{11}$$

$$\frac{1}{M}\sum_{l=1}^{M} \nabla\psi(Z_k - Z_l) = \frac{1}{N}\sum_{l\in B_k^3} \left(\frac{3A_R}{r_R^3\pi} \left(\frac{r_R}{|Z_k - Z_l|} - 1 \right) (Z_k - Z_l) \right) \tag{12}$$

where

$$B_i^1 \coloneqq \{j = 1, ..., M \text{ s.t. } |Z_j - X_i| < r_I\} \quad B_k^2 \coloneqq \{j = 1, ..., N \text{ s.t. } |X_j - Z_k| < r_I\}$$
$$B_k^3 \coloneqq \{j = 1, ..., M \text{ s.t. } |Z_j - Z_k| < r_R\}$$

Thus this numerical approach to the implementation of the IBM requires us to find the neighbours of each of the agents. This implies that we have to check the distance of 9999 agents to another agent for 10000

agents when we set $N = M = 5000$. Clearly this approach, though intuitive, is incredibly computationally intensive as the computational cost increases quadratically with the number of agents.

Calculating the SPP Direction α. We will also employ a neighbour based approach to calculate the mean direction $\bar{\alpha}_k$ of each SPP's neighbourhood. We use the functions as defined in (4) and sum over all j such that $Z_j \in B^3$.

Variable Time Step. It is clear that to ensure our sequence is sufficiently close to the theoretical result of the coupled ODEs, we need to use a small time step. This is because every time the agents move a certain distance, they "ignore" the interactions between their starting point and end point. So the smaller the step, the more accurate our solution. Our sequences of steps for each agent, $\langle X_i^t \rangle_{t\in T}$ and $\langle Z_k^t \rangle_{t\in T}$, would approach the correct solution as $\max |t_{n+1} - t_n| \to 0$. At the same time the smaller the time step, the more steps need to be computed for us to achieve some concrete patterning or even change to occur, so we use a variable time step to control the distance over which an agent can "ignore" new interactions.

We introduce a variable time step so that we can define what t_n is. In this paper we want to define the distance over which there are no new computations to half the radius of interaction between SPPs and obstacles r_I. This gives the following

$$t_{n+1} = t_n + \frac{r_I}{2 \max_l |F_l^{t_n}|} \tag{13}$$

where we define $F_l^{t_n}$ to be the total force contribution to any agent in the model (the index l is over both SPPs and obstacles). We also limit the size of $|t_{n+1} - t_n|$ to ensure that we do not see jumps beyond a certain size, to allow for low force regimes, as the SPPs will move with constant velocity u_0 and our time step needs to be small enough to capture this. For the rest of the paper the maximum value for the time step will be set at 0.075.

The implementation of this numerical approach in Matlab is given in the appendix in A.1.

2.3 Convolution Based Numerical Approach

This approach was employed to reduce computational time and to take advantage of the Fast Fourier-Transform (FFT) Algorithms, known to have low computational complexity. The motivation for this approach is to reduce the computational complexity of our numerical implementation of the IBM, when compared to the neighbour based numerical approach in Section 2.2. In this numerical implementation of the model we use a convolution based implementation. The variable time step, sequences and angle calculation remain the same (primarily due to their low computational cost), what is changed is the approach to the calculation of the forces as justified below in Prop. 2.1.

Proposition 2.1. *When considering the positions of swimmers and obstacles as densities (ρ_g and ρ_f respectively) we can calculate the forces (10),(11) and (12) with the following convolutions, evaluated at X_i or Z_k respectively:*

$$(\rho_g * \nabla\phi)(X_i) = \frac{1}{M} \sum_{k=1}^{M} \nabla\phi(X_i - Z_k) \tag{14}$$

$$(\rho_f * \nabla\phi)(Z_k) = \frac{1}{N} \sum_{i=1}^{N} \nabla\phi(Z_k - X_i) \tag{15}$$

$$(\rho_g * \nabla\psi)(Z_k) = \frac{1}{M} \sum_{l=1}^{M} \nabla\psi(Z_k - Z_l) \tag{16}$$

Proof. Let us define the positions of the swimmers and obstacles as densities as follows

$$\rho_g(x) = \frac{1}{M}\sum_{k=1}^{M}\delta(x - Z_k) \tag{17}$$

$$\rho_f(x) = \frac{1}{N}\sum_{i=1}^{N}\delta(x - X_i) \tag{18}$$

Where $\delta(x)$ is the Dirac-delta in 2 dimensions. Now

$$(\rho_g * \nabla\phi)(x) = \int_{y\in\mathbb{R}^2}\rho_g(y)\nabla\phi(x-y)\mathrm{d}y \tag{19}$$

$$= \frac{1}{M}\sum_{k=1}^{M}\int_{y\in\mathbb{R}^2}\delta(y - Z_k)\nabla\phi(x-y)\mathrm{d}y \tag{20}$$

using the fact that $(\delta * \rho) = \rho$ we get

$$(\rho_g * \nabla\phi)(x) = \frac{1}{M}\sum_{k=1}^{M}\nabla\phi(x - Z_k) \tag{21}$$

It follows that (15) and (16) can be as easily obtained using (17) and (18) in the same fashion as just shown. Further, it is assumed that all these positions are time-dependent and have been evaluated here at a given time. □

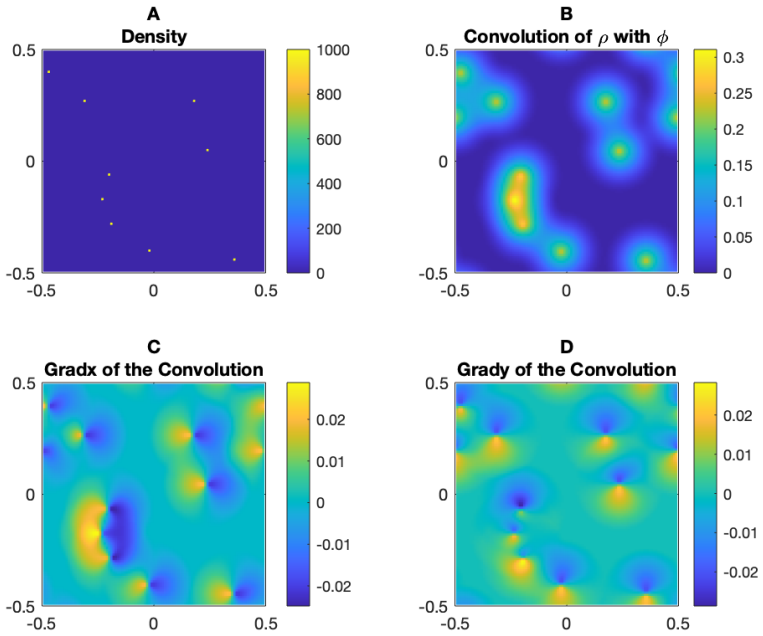


Figure 3: Implementation of convolutions. Shown is the interpretation of the convolutions and its implementation. The density of the SPPs or obstacles (A), density convoluted with ϕ (B), gradient of this convolution (C) and (D). (shown here is an example of 10 obstacles with an interaction kernel (5) with radius of interaction $r_I = 0.2$, force mass $A_I = \pi$ and grid size $\Delta x = 0.01$)

This approach, although faster due to the FFTs used in calculating the forces experienced by all the agents, involves an additional numerical approximation as both the convolution and the gradient of this

convolution, have to be evaluated on a grid to be evaluated numerically. If we define the grid size as Δx (this being the size of each of the bins, made to be squares), we note that the equations described above will be satisfied only as $\Delta x \to 0$. Obviously this choice is not possible, so our choice will introduce a numerical error proportional to the size of the grid size chosen. As will be shown later a choice of $\Delta x = 0.01$ is sufficiently small to yield accurate results.

For this implementation we have to numerically approximate the density functions and Dirac-delta so that the density integrated over the domain is always 1. The density functions are defined as follows

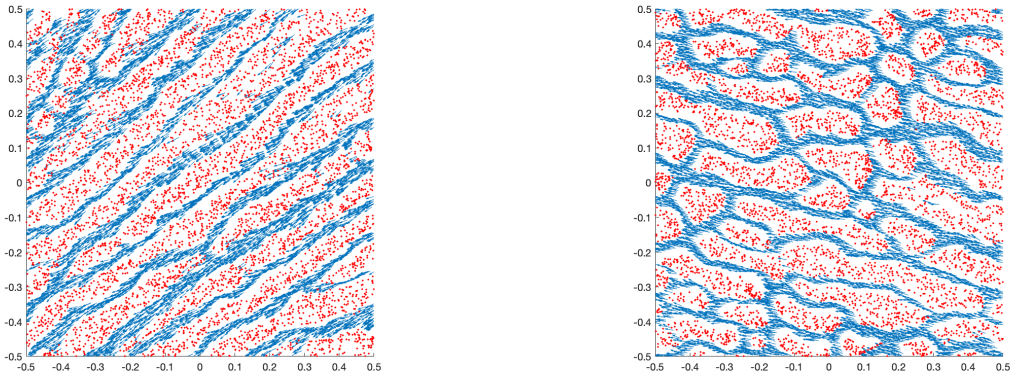
$$\rho_g(i,j) = \frac{n_{i,j}}{M\Delta x^2} \qquad \rho_f(i,j) = \frac{m_{i,j}}{N\Delta x^2}$$

where (i,j) is the coordinate of a bin of the grid and $n_{i,j}$ is the number of SPPs in that bin, whilst $m_{i,j}$ is the number of obstacles in that bin.

As shown in Fig. 3 the above definition is used in conjunction with the function ϕ or ψ evaluated on the grid to achieve a convolution on this same grid. The density of the SPPs or obstacles is calculated and assigned to bins on a grid (A), this is then convoluted with ϕ (or ψ)(B), and to determine the force in each of the vector components, we take the gradient of this convolution (C) and (D). So the force exerted by the obstacles on the SPPs or vice versa is then calculated by looking at the gradient of the convolution (C) and (D) at the position of the SPP or obstacle. To calculate the exact force exerted on an agent, we linearly interpolate the resulting grid to the position of the agent. The example in this figure only includes ten agents for simplicity of visualisation, but the concepts can be easily extended to the cases we actually simulate with thousands of agents.

The implementation of this numerical approach in Matlab is given in the appendix in A.2.

2.4 Comparison of Approaches

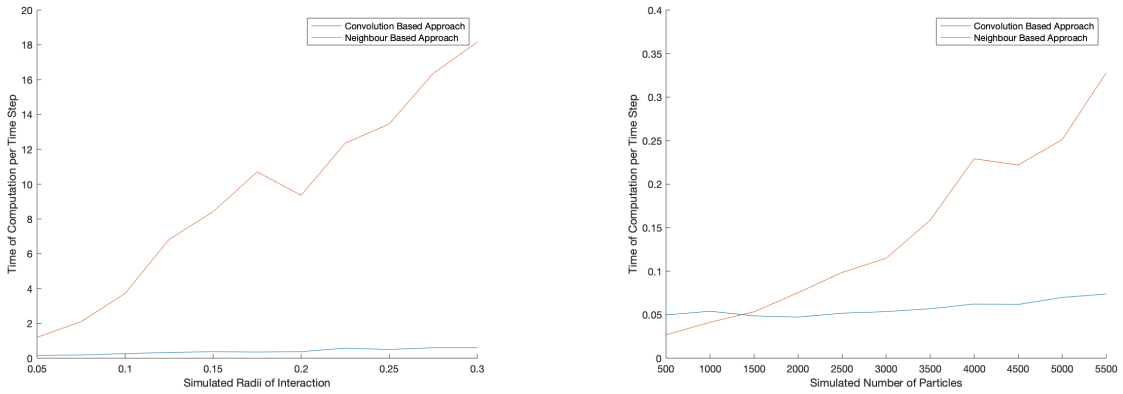


(a) Convolution Based Implementation

(b) Neighbour Based Implementation

Figure 4: Different Implementations. Here is shown the result of the implementation of the two different numerical approaches beginning from the same initial X_i and Z_k positions and under the same parameter regimes. (Here $r_I = r_R = 0.05$, $A_I = \pi$, $A_R = 1.6$ and $\Delta x = 0.01$)

We want to ensure that under a certain, positive, magnitude Δx we get that the convolution based numerical approach yields similar results to the neighbour based approach. Thus allowing us to demonstrate that the prior approach is simply a faster approach of the latter. As can be seen in the example shown in Fig. 4 there will be necessary differences in the end result, primarily due to the stochastic effects that are built in and will lead each numerical implementation to be unique. On the other hand, in the example we can see that the pattern's appearance, and therewith also its size, appears to be identical. This was tested under a variety of parameter regimes, using $\Delta x = 0.01$, and under all of them the results were consistent between the two implementations. With this observation it was considered that $\Delta x = 0.01$ was a sufficiently small grid size to allow for an accurate numerical implementation of the model.



(a) Comparison of $r_I = r_R$ and Time

(b) Comparison of $N = M$ and Time

Figure 5: Comparison of computation times. Shown are comparisons of the mean times taken to implement the two numerical approaches to the model at each time step. In (a) we vary the size of r_I, set equal to r_R, whilst in (b) we vary the number of SPPs M and obstacles N, again kept equal. (shown here is the computation time in seconds taken on a computer, this will obviously vary from machine to machine and is only meant to be an indication of proportion)(we take $A_R = 0.002$, $\kappa = 100$, $\zeta = 1$ and $\nu = 10$ in both simulations and $r_R = r_I = 0.05$ in (b))

It is quite intuitive that both numerical approaches will have pitfalls with regard to their numerical approaches and are not perfectly accurate. The model is based on continuous time which cannot be simulated on a computer, meaning that both approaches rely on discrete time steps. This means that every time a particle moves a certain distance it avoids the interactions between where it begins and where it ends up in a single time step; this is partially compensated by the variable time step we use in the simulations. The convolution based approach introduces a further element of approximation with the grid, which would only disappear as we let $\Delta x \to 0$, again this is not possible. So with the convolution based approach we introduce an additional numerical error, but, as we will discuss, we trade this for computational efficiency.

If we look at Fig. 5 we can see the efficiency of the convolution based numerical implementation, due to the efficiency of the FFT algorithms employed in this numerical implementation. For all shown interaction radii in Fig. 5 (a) the convolution based approach is significantly faster and for most tested SPP and obstacle numbers in Fig. 5 (b) we see that the convolution based implementation is more efficient. The neighbour based approach shows computational complexity $O(n^2)$ (as we would expect considering the size of the neighbourhood or the number of particles interacting) and using the computational complexity of the FFT we see that we have an overall computational complexity of $O(n \log n)$, where n is the number of agents.

The comparison of the model with the macromodel, that will be discussed in Sec. 3, requires us to use a large number of SPPs and obstacles as well as using radii of interaction $r > 0.1$. To this end it becomes clear that the best choice to optimise the time employed on the project, becomes the convolution based approach.

3 Analysis of the Macromodel

3.1 Scaling Assumptions

For the purpose of the derivation of the macromodel we introduce a scaling parameter ε to make some scaling assumptions for the the SPPs and the obstacles. For the macromodel we assume that the SPP alignment is purely local and as such $r_A = O(\varepsilon)$. Furthermore we assume that the alignment frequency ν and the orientational diffusion d_s to be of order $1/\varepsilon$. Furthermore, to relate the macromodel, based on SPP and obstacle densities, to the simulations we assume that $M, N \to \infty$.

Size and scaling of the SPP-SPP repulsion kernel. For the purpose of simplicity in the rest of the paper, though we do not assume the locality of the SPP-SPP interactions, we define the following constant

μ as a reference point for the overall magnitude of the interaction kernel ψ.

$$\mu \coloneqq \int_{x\in\mathbb{R}^2} \psi(x)\mathrm{d}x$$

Using the interaction kernel ψ as defined in (6) we can then obtain the following relationship with our original variables

$$\begin{aligned}\mu &= \int_{x\in\mathbb{R}^2} \frac{3A_R}{2r_R^3\pi}(r_R - |x|)^2 H(r_R - |x|)\mathrm{d}x \\ &= \int_{\theta=0}^{2\pi}\int_{r=0}^{r_R} \frac{3A_R}{2r_R^3\pi}(r_R - r)^2 r\mathrm{d}r\mathrm{d}\theta \\ &= \frac{A_R r_R}{4}\end{aligned}$$

Scaling for obstacles. Further, we make the assumption that the distribution of the anchors across the domain is constant and thus the the anchor density $\rho_A \equiv 1$. For simplicity we also introduce the quantities

$$\gamma = \frac{\eta}{\kappa} \qquad \delta = d_o\gamma$$

In the derivation we will assume both to be small. This implies that we have stiff tethers on our obstacles as the stiffness constant κ is large relative to the friction of the environment η. A small δ ensures that the relative noise of the obstacles is low and thus the effect of the spring is relatively high compared to the diffusion of the stochastic process.

Furthermore, for the rest of the paper we will define the following Gaussian with variance δ and centred around 0

$$M_\delta(z) = \frac{1}{(2\pi\delta)} e^{-\frac{|z|^2}{2\delta}}$$

3.2 Derivation of the Modified Macromodel

In this section we will discuss the differences between the macromodel derived in [1] and will quote multiple results from this paper and illustrate how they are modified to arrive to the macromodel used in this paper. These results will be provided without proof and limited to their application in 2 dimensions. The principle difference between this paper and [1] is that this paper does not assume the locality of the SPP-SPP repulsion and thus we will modify the results to take this into account.

Using the scaling notation from above and taking into account the scaling assumptions, as mentioned in Sec. 3.1, we get the following theorem.

Theorem 3.1 (SPP-Obstacle Macromodel (Theorem 1 in [1]))**.** *Let us consider the density of SPPs $\rho_g(x,t)$ with orientation $\Omega_g(x,t) \in \mathbb{S}^1$ and the density of the obstacles $\rho_f(x,t)$ with $x \in \mathbb{R}^2$ and $t \geq 0$. These then satisfy the following*

$$\partial_t\rho_g + \nabla_x \cdot (U\rho_g) = 0 \tag{22}$$

$$\rho_g\partial_t\Omega_g + \rho_g(V\cdot\nabla_x)\Omega_g + dP_{\Omega_g^\perp}\nabla_x\rho_g = 0 \tag{23}$$

$$U = c_1\Omega_g - \frac{1}{\zeta}\nabla_x\bar{\rho}_f - \frac{1}{\zeta}\nabla_x\tilde{\rho}_g \qquad V = c_2\Omega_g - \frac{1}{\zeta}\nabla_x\bar{\rho}_f - \frac{1}{\zeta}\nabla_x\tilde{\rho}_g \tag{24}$$

Where the constants $c_1 > 0$ and $c_2 > 0$ depend only on $d = d_s/\nu$ and are defined as in [1].

$$\rho_f(x,t) = 1 - \frac{\gamma}{\delta\eta}\left[\bar{\rho}_g(x) - [M_{2\delta} * \bar{\rho}_g](x)\right] + O(\gamma^2) \tag{25}$$

In the above equations we use the abbreviations

$$\bar{\rho}(x,t) \coloneqq (\phi * \rho)(x,t) \qquad \tilde{\rho}(x,t) \coloneqq (\psi * \rho)(x,t) \tag{26}$$

Interpretation. We note that in the absence of any convolutions the motion of SPPs and obstacles due to the interactions would be purely repulsive, leading to motion away from areas of high density and along the steepest gradients. The convolutions $\bar{\rho}$ and $\tilde{\rho}$ account for the potential for non-local effects for the SPP-obstacle and SPP-SPP interactions, respectively.

Corollary 3.1.1 (2D equations)**.** *Taking the assumptions from Sec. 3.1 into account and using Thm. 3.1 we get, in 2 spatial dimensions, the following equations for the SPP density $\rho_g(x,t)$ with orientation $\Omega(x,t)$ and the obstacle density $\rho_f(x,t)$*

$$\partial_t \rho_g + c\nabla_x \cdot (\Omega_g \rho_g) = \frac{1}{\zeta} \nabla_x \cdot (\rho_g \nabla_x \bar{\rho}_f + \rho_g \nabla_x \tilde{\rho}_g) \tag{27}$$

The obstacle density up to order γ and with $\delta \to 0$ we simplify (25) to

$$\rho_f(x,t) = 1 + \frac{\gamma}{\eta} \Delta_x \bar{\rho}_g \tag{28}$$

3.3 Linear Stability Analysis

In this section we will look at the derived macromodel, Eq.s (27) and (28) and its implications for pattern onset and size in 2 spatial dimensions.

With the large number of agents we consider on our domain we can easily consider our initial SPP density to be roughly constant, defined as the density ρ_0, and experiencing a perturbation $\rho : \mathbb{R}^2 \mapsto \mathbb{R}$ of order ε. So we assume we have an initial SPP density in the form of $\rho_g = \rho_0 + \varepsilon\rho$. If we now combine Eq.s (27) and (28) we see that ρ satisfies the linearised equation

$$\partial_t \rho + c\nabla_x \cdot (\Omega_g \rho) = \frac{\rho_0}{\zeta} \nabla_x \cdot \left(\frac{\gamma}{\eta} \nabla_x \overline{\Delta_x \bar{\rho}} + \nabla_x \tilde{\rho} \right) \tag{29}$$

where terms are only up to order ε.

Proposition 3.1 (Linear Stability)**.** *Consider (29) on $x \in [-0.5, 0.5]^2$ with periodic boundary conditions ($t \geq 0$). (i) The system permits solutions of the form $\hat{\rho}(x,t) = Ce^{\alpha(k)t}$, $C \neq 0$ and $k \in \mathbb{R}^2$ where α satisfies the following*

$$\alpha(k) = |k|^2 \rho_0 \left(\frac{|k|^2 \gamma}{\zeta\eta} \hat{\phi}^2 - \frac{\hat{\psi}}{\zeta} \right) - cik \cdot \Omega_g \tag{30}$$

where we define

$$\hat{\phi}(k) = \int_{x \in \mathbb{R}^2} e^{-ikx} \phi(x)\, dx \qquad \hat{\psi}(k) = \int_{x \in \mathbb{R}^2} e^{-ikx} \psi(x)\, dx$$

(ii) The constant steady state $\rho_g(x,t) = \rho_0$ is linearly stable iff

$$\sup_{k \in \mathbb{R}^2} |k|^2 \frac{\hat{\phi}^2(k)}{\hat{\psi}(k)} < \frac{\eta}{\gamma} \tag{31}$$

Proof. (i) Let us apply the Fourier transform to (29) and use the following properties of the Fourier transform

$$\widehat{f * g} = \hat{f}\hat{g} \qquad \widehat{\partial_x f} = ik\hat{f}$$

This allows us to obtain

$$\partial_t \hat{\rho} = \left(\frac{|k|^4 \gamma}{\zeta\eta} \hat{\phi}^2 \rho_0 - \frac{|k|^2 \hat{\psi}}{\zeta} \rho_0 - cik \cdot \Omega_g \right) \hat{\rho}$$

Giving

$$\partial_t \hat{\rho}(k,t) = \alpha(k)\hat{\rho}(k,t)$$

Thus allowing us to achieve the desired result, as claimed.

(ii) We note that the magnitude of the perturbation ρ is dependent on the $\Re(\alpha(k))$ and we will only see the disappearance of the perturbation if this real part disappears for large t. It is sufficient to look at the component in the brackets of $\alpha(k)$ as it determines the overall sign, because ρ_0 must be positive. Thus our result. □

If we consider (31) we note that the LHS is equal to κ and that for small values of κ there will be an attenuation of the pattern formation. This is a strong indication of the importance of the interaction of the obstacles with the SPPs to allow for pattern formation and that as $\hat{\psi} > 0$, due to our use of a positive, repulsive interaction kernel ψ, we will not get any pattern formation in the absence of a spring resistance κ. Furthermore we note that the principle determinant for pattern formation is the interaction kernel $\hat{\phi}$. In the case where both $\hat{\phi}$ and $\hat{\psi}$ are purely local, i.e. they are both constant, we can see that there always be a pattern. This highlights their role in the formation of patterns. To look into the effect of the parameters on the overall pattern formation we introduce the next corollary to take into account the actual shape of the interaction kernels (5) and (6).

Corollary 3.1.2. *(i) With the interaction kernels (5) and (6) we have*

$$\hat{\phi} = \frac{3A_I}{|k|^2 r_I}(-2J_2(r_I|k|) + \pi J_1(r_I|k|)H_0(r_I|k|) - \pi J_0(r_I|k|)H_1(r_I|k|))$$
$$\hat{\psi} = \frac{3A_R}{|k|^2 r_R}(-2J_2(r_R|k|) + \pi J_1(r_R|k|)H_0(r_R|k|) - \pi J_0(r_R|k|)H_1(r_R|k|))$$

(ii) The system of the macromodel is linearly stable iff

$$\sup_{k\in\mathbb{R}^2} \frac{(-2J_2(r_I|k|) + \pi J_1(r_I|k|)H_0(r_I|k|) - \pi J_0(r_I|k|)H_1(r_I|k|))^2}{|k|^2(-2J_2(r_R|k|) + \pi J_1(r_R|k|)H_0(r_R|k|) - \pi J_0(r_R|k|)H_1(r_R|k|))} < \frac{\eta A_R r_I^2}{3\gamma A_I^2 r_R} \tag{32}$$

where:
$J_i(x)$ *is the* i^{th} *Bessel Function of the first kind*
$H_i(x)$ *is the* i^{th} *Struve Function of the first kind*
We denote the RHS of the inequality as $F(|k|)$.

Proof. (i) We will show the derivation of $\hat{\phi}$ as $\hat{\psi}$ is derived analogously.

$$\hat{\phi} = \frac{3A_I}{2\pi r_I^3}\int_S (r_I - x)^2 e^{-ik\cdot x}\mathrm{d}x \qquad \text{where: } S \coloneqq \{x \in \mathbb{R}^2 \text{ s.t. } |x| < r_I\}$$

If we change to polar coordinates and rewrite k in terms of its distance to the origin and its angle φ to it we get

$$\begin{aligned}\hat{\phi} &= \frac{3A_I}{2\pi r_I^3}\int_0^{r_I}\int_0^{2\pi} r(r_I - r)^2 e^{ir|k|cos(\theta-\varphi)}\mathrm{d}\theta\mathrm{d}r \\ &= \frac{3A_I}{|k|^2 r_I}(-2J_2(r_I|k|) + \pi J_1(r_I|k|)H_0(r_I|k|) - \pi J_0(r_I|k|)H_1(r_I|k|))\end{aligned}$$

Giving us the desired result.
(ii) The result is purely a rearrangement of (31) with the interaction kernels' Fourier transforms introduced. □

Now if we fix the interaction radii r_I and r_R it becomes clear that as A_R increases any patterns would disappear and the same is true as A_I decreases. The intuition behind this is that as we increase A_R the SPP self-repulsion increases and as such their distribution becomes increasingly uniform, whilst as A_I decreases

the interaction between obstacles and SPPs disappears and the pattern ceases to appear as the interactions with the obstacles allow for the formation of patterns in the first place. We can also note that

$$\lim_{|k|\to\infty} F(|k|) = 0 \qquad \lim_{|k|\to 0} F(|k|) = 0$$

As such it is intuitive that we will have a $|k| \in \mathbb{R}$ such that this $|k|$ would give us the supremum for $F(|k|)$. It is beyond the scope of this paper to check whether this supremum is finite, i.e. the denominator of $F(|k|)$ is non-zero for all $|k| > 0$, or to establish for what $|k|$ this supremum is achieved. Suffice it to say that it is intuitive that this $|k|$ can be established numerically.

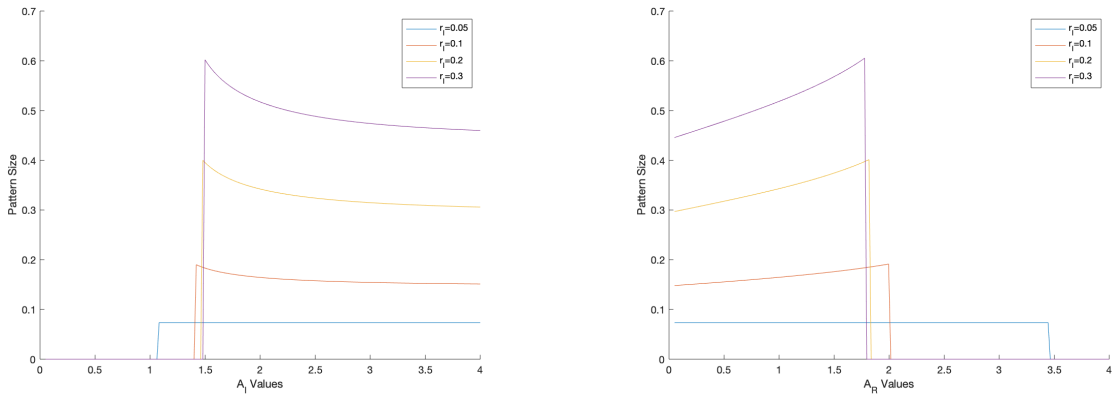
If we look at $\Re(\alpha(|k|))$ we can equally see that $\Re(\alpha(0))$ and that $\lim_{|k|\to\infty} \Re(\alpha(|k|)) = 0$. Furthermore it follows, due to the composition of the function, that $\Re(\alpha(|k|))$ does not have any singularities or discontinuities and as $\hat{\phi}$ is not constant we know that $\Re(\alpha(|k|)) \neq 0$ for some $|k|$. This allows us to say that if indeed there should be a pattern, i.e. $\exists |k| \in \mathbb{R}$ s.t. $\Re(\alpha(|k|)) > 0$, we will have a well defined maximum attained at some k_{max}. This allows us to say that $2\pi/k_{max}$ will be a good approximation for the pattern size we expect to see in simulations of the IBM. This can be derived from the fact that

$$\rho = \int_{k\in\mathbb{R}^2} e^{\alpha(k)t + ik\cdot x} \mathrm{d}k$$

so ρ will exhibit a spatial pattern of size $2\pi/k_{max}$ as it will be the pattern with the most dominant effect on the integral overall. We will numerically investigate the validity of this claim, with respect to the IBM in the following Sec. 4.

3.4 Analysis of the Predictions

As we will determine in Sec. 4.2, we can use the above discussed relations to approximate the pattern sizes we expect to appear for large t in the IBM. Thus, we have an easily computable tool that can be analysed and graphed to allow us to easily make predictions on the IBM and understand the relation the parameters have on the formation of large-scale patterns. To this end, we will discuss some relationships we see with the macromodel in this section and the implications they will have.



(a) Comparison of A_I and Pattern Size

(b) Comparison of A_R and Pattern Size

Figure 6: Shown are the pattern sizes plotted against a range of A_R (a) and against a range of A_I (b) for a variety of radii of interaction r_I. The other parameters have been kept constant throughout so that we may see how the interaction radius and the force mass interplay. We note that in areas where the plots overlap only the largest r_I plot will be visible.

In Fig. 6 we can see that the variation of the interaction radius r_I between obstacles and SPPs varies strongly the predicted pattern size, nearly independently of A_I or A_R, showing the importance of the interaction radius in determining the patterns we expect to see. This is quite intuitive as we would expect the distance over which the SPPs and obstacles interact, and therewith the gradient of this interaction, to

play a role in determining how the SPPs will cluster in the long run. This means that for large interaction radii r_I we would expect to see a slow trend toward broad clusters as the force density, as shown in 3 (B), would be more diffused and we would see larger areas of roughly equal force, from which obstacles or SPPs would be repulsed. Interestingly, we also see that under the condition of $r_I = 0.05$ and so $r_I = r_R$, there is no variation of pattern size with A_I and A_R when a pattern does emerge. This once more illustrates the important role of the interaction radius as it will determine the shapes of the forces repelling the SPPs from themselves and the SPPs from the obstacles.

We note that Fig. 6 (a) and (b) appear to be mirrored versions of each other. The reason for this is that a strong SPP self-repulsion, a large A_R, would lead to a breakdown in patterning as the forces causing a clustering of the SPPs are weaker than those preventing this clustering (dependent on force masses A_I and A_R respectively). The converse is true for A_I, as a weak interaction between obstacles and SPPs would lead to the disappearance of the effect of the obstacles on the SPPs, which we know to be the cause for the emergence of patterns in the first place. The curve-like shape of the relations can be explained in a similar manner. In Fig. 6 (a) the larger the force mass A_I value, the more the SPP clusters are compressed by the presence of obstacles around them, making the overall pattern size smaller. In Fig. 6 (b) the larger the force mass A_R value, the more the SPPs repel each other the more the clusters try to spread their own distribution, leading to larger pattern sizes overall.

Interestingly the point for which the pattern appears and disappears seems to converge with an increase of r_I, but the identification of this limit and its causes remains something for further investigation and is beyond the scope of this paper. We use the above interpretations of the results as a partial justification for the validity of the pattern, as they do not defy basic intuitions about what we should expect, as well as, providing interesting insight into the overall relations between variables and pattern size.

4 Comparison with the Model

4.1 Conditions for Comparison

As discussed earlier in our scaling assumptions, Sec. 3.1, we require an alignment radius r_A of order ε and as we have decided to set $r_R = r_A$ we will have to limit both to a small order. For the purpose of all our future simulations and their comparison to the predictions of pattern analysis we will set $r_R = r_A = 0.05$, thus we consider $\varepsilon \approx 0.1$. This leads us to define $\nu = 10$ such that it satisfies its scaling assumption of being $1/\varepsilon$. As we have set $d_o = 0$ we satisfy our condition of smallness for δ and we choose $\kappa = 100$ as $\eta = 1$ to ensure that γ is indeed small.

We ensure that N and M are sufficiently large by having set them to 5000. Due to this assumption of largeness we also treat the random, uniform distribution of the anchors to be equivalent to saying that $\rho_A \equiv 1$. Under these conditions the assumptions that are employed to derive the macromodel have been achieved and we have set our choice of parameters.

A particularity of the macromodel, specifically the equation describing the density of the obstacles Eq. (25), is that it can yield negative densities when $-\eta/\gamma > \Delta_x \bar{\rho}_g$. To prevent this we need to ensure that $r_I \in [\,0.05, 0.3]$ and we want to ensure that $\mu \in [\,0.005, 0.03]$. So we see that the macromodel we use requires us to look at certain IBM regimes for us to even claim there is a possibility of a pattern comparison between the IBM and our pattern predictions.

4.2 Comparison

Pattern Analysis. As we work on periodic domain we can reasonably argue that the perturbation, or pattern, ρ can be expressed as a Fourier Series. This allows us to say that if we take the Fourier transform of ρ we will find peaks indicating the most prominent spatial patterns that we expect to see, giving us the number of times that the peaks of the pattern occur across the domain. As we are working on a 2 dimensional domain we want to ensure that we get the number the number of repetitions of the pattern in any direction

so we choose to take the Fourier transform of the perturbation $\hat{\rho}$ in polar coordinates

$$\hat{\rho}(r,\theta) = \frac{1}{M}\sum_{k=1}^{M} e^{-2\pi i c_k(r,\theta)}$$
$$c_k(r,\theta) = \begin{pmatrix} r\cos\theta \\ r\sin\theta \end{pmatrix} \cdot X_k$$

So the peaks we find will have the coordinates (r,θ), where we get the number of repetitions of the pattern in the domain r, and the direction in which the pattern is travelling θ. To establish the pattern size that we use to compare to our prediction, we identify the mean direction of the pattern θ and therewith the length of space over which the pattern is repeated and then we can identify the pattern size from this and the pattern number, or the number of times the pattern is repeated over the domain. We choose the highest peak of the Fourier transform as the magnitude of the peak is an indication of the prominence of the frequency the peak is located at in the overall formation of the pattern. We will always take the pattern at $t = 30$, as this appears to be a time at which any pattern that would appear has appeared, though the patterns should stabilise for $t \to \infty$ so the larger the t the more accurate our pattern analysis.

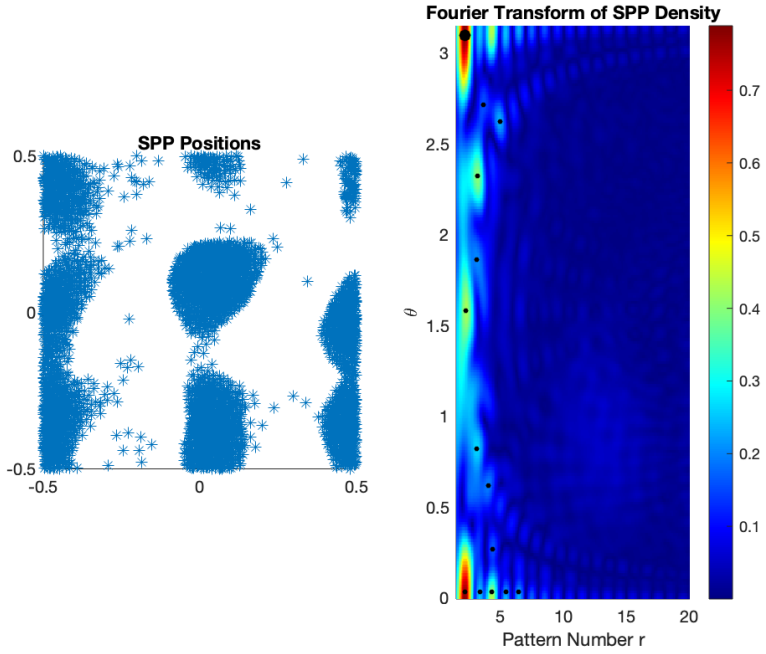


Figure 7: Shown is an example for which it is not clear what the exact pattern size given by the Fourier transform is. In this case this is due to the proximity of two of the peaks, both spatially and in terms of magnitude. In the Fourier transform we have highlighted all the local peaks with black dots, the largest one of which represents the global maximum on the domain. (the code used to generate this interpretation and used to analyse the pattern is given in A.4)

It is not always possible to accurately use this approach for a variety of reasons. Primarily the most prominent peak may not be the best peak to choose for the overall pattern, so that we have to moderate the choice of the peak. When this is the case the different peaks we can choose from are relatively close, as in Fig. 7. Sometimes there is a glaring discrepancy between the most prominent peaks and the pattern that we can quite clearly note in the IBM simulation. This occurs when the pattern takes the form of clusters, possibly moving in different directions, so the Fourier Transform is unable to take into account these time-based alterations and any result from the Fourier transform is inaccurate. In this case the pattern size is determined manually, by visual approximation. Furthermore, this approach does not deal with uniform ρ,

where no pattern occurs. In these cases we can see that the peaks are of low magnitude and the overall surface of the Fourier transform is more uniform, making it clear that there is no pattern. This case is manually set to give the pattern number 0.

As such we do have to vet and moderate the results that we get from this system of pattern analysis. In most cases the choices made, as discussed above, are quite uncontroversial, but there are undoubtedly situations in which a subjective judgement has had to be made, within this framework.

Comparison. We make a comparison between our predictions and the simulated results for two different r_I and we vary the value of μ between 0.01 and 0.035. The μ range has been chosen as it is within the domain of the μ values for which the model is valid and for which we know there will be μ values for which there will not be a pattern. This is so that we can test the ability of the model to predict pattern size and to test whether it accurately predicts the conditions under which no pattern emerges. We have chosen two different values for r_I, namely 0.2 and 0.3 (both within the range for which the macromodel and our comparison is supported), for illustrative purposes to demonstrate the consistency of the pattern predictions and what we can see in the IBM simulation. The trend shown extends to more values of r_I, but these would reduce the overall clarity of the diagram.

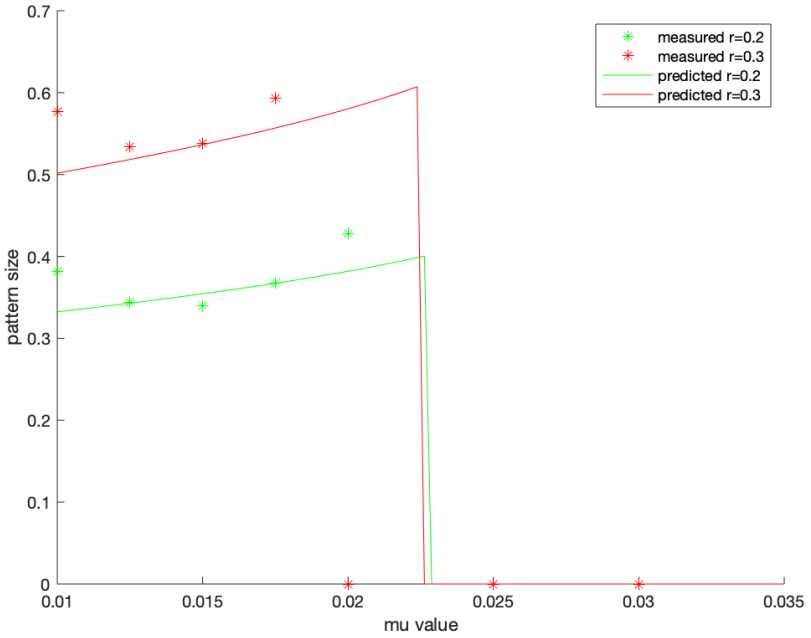


Figure 8: We depict the predicted and measured pattern size for comparison. Note that for the μ values 0.025 and 0.03 the measured values overlap so that it appears that there is no measurement for $r_I = 0.2$. (in the diagram r_I is referred to r and μ is referred to as mu)

As we can see in Fig. 8, the predicted pattern matches the measured patterns quite well. We need to take into account that the IBM simulations include a component of uncertainty with the stochastic processes, so they will always vary from the prediction to some extent. Furthermore, we only have linearised the PDEs of the macromodel, so we also introduce an uncertainty of magnitude ε^2. These elements introduce uncertainty so that even when equipped with a perfectly accurate prediction mechanism, we would expect to see some divergence from the prediction when using a simulation over a finite t. On the other hand, we see that the results that we find, when vetted for the aforementioned anomalies, will yield relatively consistent results, fluctuating around the predicted values (displayed is only one of these values to illustrate a representative series of models and their pattern sizes).

Interestingly we can see that the measured values seem to diverge from the prediction at the lowest μ

value, with consistency among various simulations and different choices of r_I. We can also see that the model is fairly accurate in predicting the μ values for which we would not expect to see a pattern, though there is some uncertainty as can be noted this the IBM measured pattern for $r_R = 0.3$ and $\mu = 0.02$, but for larger μ values we can consistently see the expected lack of patterning.

All this is to say that indeed the macromodel and the predictions it can make about the pattern sizes, or lack thereof, are fairly accurate, when we compare them to what we are able to see in the IBM simulations.

5 Conclusion

In this work we introduce an IBM for the interaction of SPPs and obstacles and, though the interactions have fairly simple form, we are able to see the emergence of patterns. This illustrates the ability of the system to self-organise and the ability of coupled ODEs to create structure on a macro scale, only employing an awareness of local scale interactions and relations. This insight allows us to look at the effects of the IBM's thousands of coupled ODEs from the perspective of the agents' densities, which we can describe in terms of two coupled, non-linear PDEs. The derivation of these equations, called the macro-model, assumed fast obstacle spring relaxation (small γ). We then limited ourselves to observations of the model in 2 dimensions.

We introduced two different approaches to the simulation of the IBM and came to the conclusion that the convolution based method, as described in 2.3, would be accurate and efficient in computing the IBM. Thus we were equipped with a relatively accurate approach of implementing the model to be able to identify what different parameters would yield.

At this point we introduce the macromodel and perform linear stability analysis on the PDEs, which revealed the conditions under which we could expect pattern formation and gave us the relations to estimate the pattern size. We discuss some of the intuition and behaviour behind the predictions made by the relations we have derived on pattern size. Here we see how the local effects of the interaction kernels show themselves in the broader macro effects of self-organisation into patterns. Surprisingly, we note that with $r_I = r_R$, the pattern size is unaffected by changes in force masses A_R and A_I, beyond determining the occurrence of patterns or lack thereof.

In the end we reveal that indeed the macromodel is able to relatively accurately predict the pattern sizes we expect to see in the long term, by comparing the predictions inferred from the macromodel to the actual numerical implementations of the IBM. This only works under a set of limiting conditions for which the macromodel itself is a valid model for the IBM and does not yield impossible results (i.e. negative densities). This allows us to: simplify the unsolvable behaviour of thousands of coupled ODEs into a few PDEs and use these to predict the behaviour of the IBM. Under this confirmation we are then able to analyse the effects that the parameters and components of the PDE have on each other and better understand the interactions that cause the patterning.

A Appendix

A.1 Neighbour Based Implementation of the IBM

```
function data=Model_mod4_tvar(p)

%This function takes in the parameters of the class p and then runs
%throught the following implementation, returning a data with structure
%elements data.x and data.y for the x and y coordinates of the SPPs,
%data.thetas to store the SPPs' orientation, data.X1 and data.X2 for the x
%and y coordinates of the obstacles as well as their fixed anchor points
%data.Y1 and data.Y2. This is for each data entry which corresponds to each
%frame of the simulation.

%% Parameters

% if no parameters are provided, use these
if nargin<1
    p.dt = 0.1; %time step
    p.N = 5000; % number of particles
    p.M = 5000; % number of tethers
    p.k = 1; %stiffness constant
    p.L = 10; % domain size
    p.tmax = 10; % number of timesteps
    p.noise = 0.3;
    p.noiseOb = 0.2;
    p.r = 0.5; % interaction radius of SPPs to themselves
    p.r1 = 0.2; % interaction radius of objects to SPPs
    p.vl = 1; %constant velocity
    p.eta = 1;   %friction coefficients
    p.zeta = 1;
    p.A = 1; %force mass
    p.AR = 0.02;
    p.nu = 1;
end

%% Initialize

% initialize results structure
data(p.tmax+1).x=[];
data(p.tmax+1).y=[];
data(p.tmax+1).thetas=[];
data(p.tmax+1).X1=[];   %Object positions
data(p.tmax+1).X2=[];
data(p.tmax+1).Y1=[];   %Tether positions
data(p.tmax+1).Y2=[];

% random initial positions
x=p.L*rand(p.N,1)-p.L/2;
y=p.L*rand(p.N,1)-p.L/2;
X1=p.L*rand(p.M,1)-p.L/2;
X2=p.L*rand(p.M,1)-p.L/2;
Y1=X1;
Y2=X2;

% random initial orientations
thetas=2*pi*rand(p.N,1);

% save initial conditions
data(1).x=x;
data(1).y=y;
data(1).thetas=thetas;
data(1).X1=X1;
data(1).X2=X2;
data(1).Y1=X1;
data(1).Y2=X2;

```

```
%% Run

i = 1;
%variable to keep track of timestep
totaldt = 0;

% look through timesteps
while i-1<p.tmax/0.05

    tic

    % update
    [x,y,thetas,X1,X2,dt]=update(x,y,thetas,X1,X2,Y1,Y2,p);

    totaldt = totaldt+dt;

    % store
    if totaldt > 0.05
        data(i).x=x;
        data(i).y=y;
        data(i).thetas=thetas;
        data(i).X1=X1;
        data(i).X2=X2;
        data(i).Y1=Y1;
        data(i).Y2=Y2;
        data(i).dt=dt;
        i = i+1;
        totaldt = 0;
    end

    % some reporting
    fprintf('Time step %s took %s sec\n', num2str(i), num2str(toc))

end

end

%% Ancillary function

function [x,y,thetas,X1,X2,dt]=update(x,y,thetas,X1,X2,Y1,Y2,p)
%{
update the positions and directions
%}
    X1m=mod(X1+p.L/2,p.L)-p.L/2;
    X2m=mod(X2+p.L/2,p.L)-p.L/2;

    % find neighbors of all particles
    [nb, indexExt]=neighbours(x,y,x,y,p.r,p);

    % find objects that are neighbours
    [NB1, index1] = neighbours(X1m,X2m,x,y,p.r1,p);

    % determine average angle
    avrg_angle=avrg_anglev(thetas,indexExt,nb,p);

    %aux vectors
    B_X=cos(thetas)+p.nu*p.dt/2*(cos(avrg_angle)-cos(thetas));
    B_Y=sin(thetas)+p.nu*p.dt/2*(sin(avrg_angle)-sin(thetas));

    %the signed angle between B and omega
    angle = -atan2d(B_X.*sin(thetas)-B_Y.*cos(thetas),B_X.*cos(thetas)+B_Y.*sin(thetas))*pi
    /180;

    % add noise
    avrg_angle_noise=thetas+2*angle+sqrt(2*p.noise*p.dt).*randn(p.N,1);

    %evaluate all of the sums in the ODEs
    [sumsw1, sumob1] = sumF(x,X1m,y,X2m,index1,NB1,p);
```

```
    [sumsw2, sumob2] = sumF(y,X2m,x,X1m,index1,NB1,p);
    [sum1, ~] = sumF(x,x,y,y,indexExt,nb,p);
    [sum2, ~] = sumF(y,y,x,x,indexExt,nb,p);

    %determine the overall forces acting on the SPPs and obstacles
    fx = 1/p.zeta*(p.A*sumsw1+p.AR*p.N/p.M*sum1);
    fy = 1/p.zeta*(p.A*sumsw2+p.AR*p.N/p.M*sum2);
    Fx = 1/p.eta*(p.k*(X1-Y1)+p.A*sumob1);
    Fy = 1/p.eta*(p.k*(X2-Y2)+p.A*sumob2);

    %determine the timestep to be taken
    m = max(abs([fx fy Fx Fy]),[],'all');

    dt = p.r/(2*m);

    %we limit the highest possible timestep
    if dt > 0.075
        dt = 0.075;
    end

    % update positions & direction
    x=x+dt*p.vl*cos(avrg_angle_noise)-dt*fx;
    y=y+dt*p.vl*sin(avrg_angle_noise)-dt*fy;
    thetas = avrg_angle_noise;
    X1=X1-dt*Fx+sqrt(2*dt*p.noiseOb)*randn(p.M,1);
    X2=X2-dt*Fy+sqrt(2*dt*p.noiseOb)*randn(p.M,1);

    % account for periodicity
    x=mod(x+p.L/2,p.L)-p.L/2;
    y=mod(y+p.L/2,p.L)-p.L/2;

    %fprintf('diff %s\n', sum(sumsw1)+sum(sumob1))

end

function [xExt, yExt, indexExt]=do_periodic_ext(x, y,p)
%{
make copies of particles to account for periodocity
%}
    border=p.r;

    % who is near a boundary
    isNearBoundary_L = x<=-p.L/2+border; isNearBoundary_R = x>=p.L/2-border;
    isNearBoundary_D = y<=-p.L/2+border; isNearBoundary_U = y>=p.L/2-border;

    % who is near an edge
    isNearEdge_LU=isNearBoundary_L & isNearBoundary_U; isNearEdge_LD=isNearBoundary_L &
    isNearBoundary_D;
    isNearEdge_RU=isNearBoundary_R & isNearBoundary_U; isNearEdge_RD=isNearBoundary_R &
    isNearBoundary_D;

    % extend
    xExt=[x; x(isNearBoundary_L)+p.L; x(isNearBoundary_R)-p.L; x(isNearBoundary_D); x(
    isNearBoundary_U);...
            x(isNearEdge_LU)+p.L; x(isNearEdge_LD)+p.L; x(isNearEdge_RU)-p.L; x(
    isNearEdge_RD)-p.L];
    yExt=[y; y(isNearBoundary_L); y(isNearBoundary_R); y(isNearBoundary_D)+p.L; y(
    isNearBoundary_U)-p.L;...
            y(isNearEdge_LU)-p.L; y(isNearEdge_LD)+p.L; y(isNearEdge_RU)-p.L; y(
    isNearEdge_RD)+p.L];

    % copy also index infos to identify correct neighbors later
    index=(1:length(x))';
    indexExt=[index; index(isNearBoundary_L); index(isNearBoundary_R); index(
    isNearBoundary_D); index(isNearBoundary_U);...
            index(isNearEdge_LU); index(isNearEdge_LD); index(isNearEdge_RU); index(
    isNearEdge_RD)];

```

```
end

function [nb, indexExt] = neighbours(x,y,X1,X2,r,p)
%{
Find neighbours within radius p.r
%}

    % extend to account for periodocity
    [xExt, yExt, indexExt]=do_periodic_ext(x, y,p);

    % find neighbors
    nb = rangesearch([xExt yExt],[X1 X2],r);

end

function [sumsw, sumob] = sumF(x,X1,y,X2,index,NB,p)

N=length(x);
M=length(X1);
sumsw=zeros(N,1);
sumob=zeros(M,1);

    for k=1:N
        iNB=index(NB{k});
        for i = 1:length(iNB)
            if X1(iNB(i)) == x(k) && X2(iNB(i)) == y(k) % to avoid singularities in dphi
            else
                P = dphi(x(k),X1(iNB(i)),y(k),X2(iNB(i)),p);
                sumsw(k) = sumsw(k)+P;
                sumob(iNB(i)) = sumob(iNB(i))-P;
            end
        end
    end

    sumsw=1/N*sumsw;
    sumob=1/N*sumob;

end

function r = dphi(x,X1,y,X2,p)

if abs(x-X1) > p.r1
    X1 = X1-sign(X1)*p.L;
elseif abs(y-X2) > p.r1
    X2 = X2-sign(X2)*p.L;
end

absv=sqrt((x-X1)^2+(y-X2)^2);
con=3/((p.r1^3)*pi);

r=con*(p.r1-absv)*(X1-x)/absv;

end

function avrg_angle = avrg_anglev(thetas,indexExt,nb, p)
%{
Find mean direction
%}

% intialize
avrg_angle=zeros(p.N,1);

    % loop through particles
    for k=1:p.N

        % neighbor indices
        iNb=indexExt(nb{k});

```

```
        % find mean direction by adding vectors
        xs=sum(cos(thetas(iNb)));
        ys=sum(sin(thetas(iNb)));

        % determine corresponding direction
        theta = atan2(ys,xs);

        % store
        avrg_angle(k) = theta;

    end
end
```

A.2 Convolution Based Implementation of the IBM

```
function data=Model_mod_numerical_tvar(p)

%This function takes in the parameters of the class p and then runs
%throught the following implementation, returning a data with structure
%elements data.x and data.y for the x and y coordinates of the SPPs,
%data.thetas to store the SPPs' orientation, data.X1 and data.X2 for the x
%and y coordinates of the obstacles as well as their fixed anchor points
%data.Y1 and data.Y2. This is for each data entry which corresponds to each
%frame of the simulation.

%% Parameters

% if no parameters are provided, use these
if nargin<1
    p.dt = 1; %time step
    p.N = 10; % number of particles
    p.M = 10; % number of tethers
    p.k = 1; %stiffness constant
    p.L = 1; % domain size
    p.tmax = 1; % number of timesteps
    p.noise = 0.3;
    p.noiseOb = 0.2;
    p.r = 0.2; % interaction radius of SPPs to themselves
    p.r1 = 0.2; % interaction radius of objects to SPPs
    p.vl = 1; %constant velocity
    p.eta = 1;   %friction coefficients
    p.zeta = 1;
    p.A = pi; %force mass
    p.AR = 0.02;
    p.nu = 1;
    p.res = 100;
    p.dx = p.L/p.res;
    p.xVals = linspace(-p.L/2+p.dx/2,p.L/2-p.dx/2,p.res);
    p.yVals = linspace(-p.L/2+p.dx/2,p.L/2-p.dx/2,p.res);
end

%% Initialize
%
% initialize results structure
data(p.tmax+1).x=[];
data(p.tmax+1).y=[];
data(p.tmax+1).thetas=[];
data(p.tmax+1).X1=[];   %Object positions
data(p.tmax+1).X2=[];
data(p.tmax+1).Y1=[];   %Tether positions
data(p.tmax+1).Y2=[];
data(p.tmax+1).dt=[];

% random initial positions
x=1*(p.L*rand(p.M,1)-p.L/2);
y=1*(p.L*rand(p.M,1)-p.L/2);
X1=1*(p.L*rand(p.M,1)-p.L/2);
X2=1*(p.L*rand(p.M,1)-p.L/2);
```

```
Y1=X1;
Y2=X2;
%}
% random initial orientations
thetas=2*pi*rand(p.N,1);

%
% save initial conditions
data(1).x=x;
data(1).y=y;
data(1).thetas=thetas;
data(1).X1=X1;
data(1).X2=X2;
data(1).Y1=X1;
data(1).Y2=X2;

i = 1;
totaldt = 0;

%% Run

% look through timesteps
while i-1<p.tmax/0.05

    tic

    % update
    [x,y,thetas,X1,X2,dt]=update(x,y,thetas,X1,X2,Y1,Y2,p);

    totaldt = totaldt+dt;

    % store
    if totaldt > 0.05
        data(i).x=x;
        data(i).y=y;
        data(i).thetas=thetas;
        data(i).X1=X1;
        data(i).X2=X2;
        data(i).Y1=Y1;
        data(i).Y2=Y2;
        data(i).dt=dt;
        i = i+1;
        totaldt = 0;
    end

    % some reporting
    fprintf('Time step %s took %s sec\n', num2str(i), num2str(toc))

end

end

%% Ancillary function

function [x,y,thetas,X1,X2,dt]=update(x,y,thetas,X1,X2,Y1,Y2,p)
%{
update the positions and directions
%}
    X1m=mod(X1+p.L/2,p.L)-p.L/2;
    X2m=mod(X2+p.L/2,p.L)-p.L/2;

    denss = density(x,y,p);    %swimmer density
    denso = density(X1m,X2m,p);  %object density
    [NPh,X_,Y_] = NPhi(p.r1,p);    %Phi evaluated on a p.dx by p.dx grid
    [NPs,X__,Y__] = NPhi(p.r,p);   %Psi evaluated on a p.dx by p.dx grid
    [anx,any] = convrp(X1m,X2m,denss,NPh,X_,Y_,p);    %object-swimmer interaction for object
    [anx1,any1] = convrp(x,y,denso,NPh,X_,Y_,p);    %object-swimmer interaction for swimmer
    [anx2,any2] = convrp(x,y,denss,NPs,X__,Y__,p);  %swimmer-swimmer interaction
```

```
122
123     %calculate all the forces
124     fx = -1/p.zeta*(p.A*anx1+p.AR*anx2);
125     fy = -1/p.zeta*(p.A*any1+p.AR*any2);
126     Fx = 1/p.eta*(p.k*(X1-Y1)+p.A*anx);
127     Fy = 1/p.eta*(p.k*(X2-Y2)+p.A*any);
128
129     %determine the timestep to be taken
130     m = max(abs([fx fy Fx Fy]),[],'all');
131
132     dt = p.r/(2*m);
133
134     %we limit the highest possible timestep
135     if dt > 0.075
136         dt = 0.075;
137     end
138
139     % find neighbors of all particles
140     [nb, indexExt]=neighbours(x,y,x,y,p.r,p);
141
142     % determine average angle
143     avrg_angle=avrg_anglev(thetas,indexExt,nb,p);
144
145     %aux vectors
146     B_X=cos(thetas)+p.nu*dt/2*(cos(avrg_angle)-cos(thetas));
147     B_Y=sin(thetas)+p.nu*dt/2*(sin(avrg_angle)-sin(thetas));
148
149     %the signed angle between B and omega
150     angle = -atan2d(B_X.*sin(thetas)-B_Y.*cos(thetas),B_X.*cos(thetas)+B_Y.*sin(thetas))*pi
        /180;
151
152     % add noise
153     avrg_angle_noise=thetas+2*angle+sqrt(2*p.noise*dt).*randn(p.N,1);
154
155     % update positions & direction
156     x=x+dt*(p.vl*cos(avrg_angle_noise)+fx);
157     y=y+dt*(p.vl*sin(avrg_angle_noise)+fy);
158     thetas = avrg_angle_noise;
159     X1=X1-dt*Fx+sqrt(2*dt*p.noiseOb)*randn(p.M,1);
160     X2=X2-dt*Fy+sqrt(2*dt*p.noiseOb)*randn(p.M,1);
161
162     % account for periodicity
163     x=mod(x+p.L/2,p.L)-p.L/2;
164     y=mod(y+p.L/2,p.L)-p.L/2;
165
166 end
167
168 function [xExt, yExt, indexExt]=do_periodic_ext(x, y,p)
169 %{
170 make copies of particles to account for periodocity
171 %}
172     border=p.r;
173
174     % who is near a boundary
175     isNearBoundary_L = x<=-p.L/2+border; isNearBoundary_R = x>=p.L/2-border;
176     isNearBoundary_D = y<=-p.L/2+border; isNearBoundary_U = y>=p.L/2-border;
177
178     % who is near an edge
179     isNearEdge_LU=isNearBoundary_L & isNearBoundary_U; isNearEdge_LD=isNearBoundary_L &
        isNearBoundary_D;
180     isNearEdge_RU=isNearBoundary_R & isNearBoundary_U; isNearEdge_RD=isNearBoundary_R &
        isNearBoundary_D;
181
182     % extend
183     xExt=[x; x(isNearBoundary_L)+p.L; x(isNearBoundary_R)-p.L; x(isNearBoundary_D); x(
        isNearBoundary_U);...
184             x(isNearEdge_LU)+p.L; x(isNearEdge_LD)+p.L; x(isNearEdge_RU)-p.L; x(
        isNearEdge_RD)-p.L];
```

```
    yExt=[y; y(isNearBoundary_L); y(isNearBoundary_R); y(isNearBoundary_D)+p.L; y(isNearBoundary_U)-p.L;...
            y(isNearEdge_LU)-p.L; y(isNearEdge_LD)+p.L; y(isNearEdge_RU)-p.L; y(isNearEdge_RD)+p.L];

    % copy also index infos to identify correct neighbors later
    index=(1:length(x))';
    indexExt=[index; index(isNearBoundary_L); index(isNearBoundary_R); index(isNearBoundary_D); index(isNearBoundary_U);...
            index(isNearEdge_LU); index(isNearEdge_LD); index(isNearEdge_RU); index(isNearEdge_RD)];

end

function [nb, indexExt] = neighbours(x,y,X1,X2,r,p)
%{
Find neighbours within radius p.r
%}

    % extend to account for periodocity
    [xExt, yExt, indexExt]=do_periodic_ext(x, y,p);

    % find neighbors
    nb = rangesearch([xExt yExt],[X1 X2],r);

end

%the function phi evaluated on a grid
function [R,X,Y] = NPhi(r,p)
    [X,Y] = meshgrid(p.xVals);
    R = phi(X,Y,r,p);
end

function R = phi(x,y,r,p)
    absv = sqrt(x.^2+y.^2);
    axis equal tight
    xlim([-1 1]*p.L/2); ylim([-1 1]*p.L/2)
    con = 3/(2*r^3*pi);
    I = absv < r;
    R = con*(r-absv).^2.*I;
end

%the convolution of the functions
function [anx,any] = convrp(x,y,dens,NPh,X0,Y0,p)
    [convx,convy] = gradient(convol(NPh,dens,p),p.dx);
    %We extend the grid with the following values
    lhx = interp2(X0,Y0,convx,-p.L/2+p.dx,p.yVals,'linear',0);
    rhx = interp2(X0,Y0,convx,p.L/2-p.dx,p.yVals,'linear',0);
    uhx = interp2(X0,Y0,convx,p.xVals,p.L/2-p.dx,'linear',0);
    bhx = interp2(X0,Y0,convx,p.xVals,-p.L/2+p.dx,'linear',0);
    lhy = interp2(X0,Y0,convy,-p.L/2+p.dx,p.yVals,'linear',0);
    rhy = interp2(X0,Y0,convy,p.L/2-p.dx,p.yVals,'linear',0);
    uhy = interp2(X0,Y0,convy,p.xVals,p.L/2-p.dx,'linear',0);
    bhy = interp2(X0,Y0,convy,p.xVals,-p.L/2+p.dx,'linear',0);
    [X,Y] = meshgrid(linspace(-p.L/2,p.L/2,p.res+2));
    convx = [(bhx(1)+rhx(1))/2 bhx (bhx(end)+lhx(1))/2;rhx convx lhx;(uhx(1)+rhx(1))/2 uhx (uhx(end)+lhx(1))/2];
    convy = [(bhy(1)+rhy(1))/2 bhy (bhy(end)+lhy(1))/2;rhy convy lhy;(uhy(1)+rhy(1))/2 uhy (uhy(end)+lhy(1))/2];
    anx = interp2(X,Y,convx,x,y,'linear',0);
    any = interp2(X,Y,convy,x,y,'linear',0);
end

%function to determine the weighted density
function rho = density(x,y,p)
    Xedges=[p.xVals p.L/2+p.dx/2]-p.dx/2;
    Yedges=[p.yVals p.L/2+p.dx/2]-p.dx/2;
    N = length(x);
```

```
    [r,~,~] = histcounts2(x,y,Xedges,Yedges);
    rho = r'/(N*p.dx^2);
end

function Y = convol(z,v,p)
    a = fft2(ifftshift(z));
    b = fft2(ifftshift(v));
    Y = p.dx^2*fftshift(ifft2(a.*b, 'symmetric'));
end

function avrg_angle = avrg_anglev(thetas,indexExt,nb, p)
%{
Find mean direction
%}

% intialize
avrg_angle=zeros(p.N,1);

    % loop through particles
    for k=1:p.N

        % neighbor indices
        iNb=indexExt(nb{k});

        % find mean direction by adding vectors
        xs=sum(cos(thetas(iNb)));
        ys=sum(sin(thetas(iNb)));

        % determine corresponding direction
        theta = atan2(ys,xs);

        % store
        avrg_angle(k) = theta;

    end
end
```

A.3 Pattern Size Prediction

```

function data=Pattern_prediction2(p)
%This function takes in the parameters of the IBM and returns the predicted
%pattern size for that parameter regime

%If no parameters are inputted the following are taken
if nargin<1
    p.k = 100; %stiffness constant
    p.L = 1; % domain size
    p.noise = 0.1;
    p.noiseOb = 0;
    p.r = 0.05; % interaction radius of SPPs to themselves (rR)
    p.r1 = 0.1; % interaction radius of objects to SPPs (rI)
    p.vl = 1; %constant velocity
    p.eta = 1;   %friction coefficients
    p.zeta = 1;
    p.A = pi; %force mass
    p.AR = 0.00;
    p.nu = 10;
    p.mu = 0.01;
end

[~,dim,~]=patest(p);

data = dim;


function fun=StruveH0(z)
    %
```

```
    % StruveH0 calculates the function StruveH0 for complex argument z
    %
    % Author : T.P. Theodoulidis
    % Date   : 11 June 2012
    %
    % Arguments
    % z : can be scalar, vector, matrix
    %
    % External routines called         : cheval, StruveH0Y0
    % Matlab intrinsic routines called : bessely, besselh
    %
    bn=[...
     7.741208505217204e-002 -1.489049907224780e-001  1.365908688119806e-001...
    -1.244408977167160e-001  1.213210541746413e-001 -9.902285991131098e-002...
     9.285469594924457e-002 -8.889396057027860e-002  6.318108596562945e-002...
    -3.164140591816664e-002  1.160282350916835e-002 -3.255171535152174e-003...
     7.240935462046915e-004 -1.313122517655078e-004  1.984234861137538e-005...
    -2.542464986238115e-006  2.802187746192928e-007 -2.688335553599889e-008...
     2.267665290294826e-009 -1.696387437431367e-010  1.133879020090459e-011...
    -6.816279772928244e-013  3.706645666482425e-014 -1.832769429508806e-015...
     8.278232328724399e-017 -3.429926823742390e-018  1.308568653462734e-019...
    -4.612908468868213e-021  1.507287730295555e-022 -4.578534841587558e-024...
     1.296387199302720e-025 -3.430073819717729e-027  8.500386329076394e-029...
    -1.977322978489543e-030  4.326093533645215e-032];
    %
    x=z(:);
    %
    % |x|>16
    i2=abs(x)>16;
    x2=x(i2);
    i1=~i2;
    x1=x(i1);
    if isempty(x2)==0
        fun2=StruveH0Y0(x2)+bessely(0,x2);
    else
        fun2=[];
    end
    % |x|<=16
    if isempty(x1)==0
        z1=x1.^2/400;
        fun1=cheval('shifted',bn,z1)*2.*x1/pi;
    else
        fun1=[];
    end
    %

    %
    fun=x*0;
    fun(i1)=fun1;
    fun(i2)=fun2;
    %
    fun=reshape(fun,size(z));
    %
end

function fun=StruveH0Y0(z)
    %
    % StruveH0Y0 calculates the function StruveH0-BesselY0 for complex argument z
    %
    % Author : T.P. Theodoulidis
    % Date   : 11 June 2012
    %
    % Arguments
    % z : can be scalar, vector, matrix
    %
    % External routines called         : cheval, StruveH0
    % Matlab intrinsic routines called : bessely, besselh
    %
```

```
    nom=[4,0,8816              ,0,6778206           ,...
           0,2317961250        ,0,374638650750      ,...
           0,28306147182390    ,0,937687098467700   ,...
           0,11970864124436700,0,46174193179143750,...
           0,32071055725170000,0,840808761085125];
    %
    den=[4,0,8820              ,0,6786990           ,...
           0,2324669760        ,0,376904178000      ,...
           0,28663562736900    ,0,963414191990250   ,...
           0,12740661151218000,0,54025303535453250,...
           0,50023429199493750,0,4092826025413125];
    %
    x=z(:);
    %
    % |x|<=16
    i1=abs(x)<=16;
    x1=x(i1);
    if isempty(x1)==0
        fun1=StruveH0(x1)-bessely(0,x1);
    else
        fun1=[];
    end
    %
    % |x|>16 and real(x)<0 and imag(x)<0
    i2=(abs(x)>16 & real(x)<0 & imag(x)<0);
    x2=x(i2);
    if isempty(x2)==0
        x2=-x2;
        fun2=-(2/pi./x2.*polyval(nom,x2)./polyval(den,x2))+2i*besselh(0,1,x2);
    else
        fun2=[];
    end
    % |x|>16 and real(x)<0 and imag(x)>=0
    i3=(abs(x)>16 & real(x)<0 & imag(x)>=0);
    x3=x(i3);
    if isempty(x3)==0
        x3=-x3;
        fun3=-(2/pi./x3.*polyval(nom,x3)./polyval(den,x3))-2i*besselh(0,2,x3);
    else
        fun3=[];
    end
    % |x|>16 and real(x)>=0
    i4=(abs(x)>16 & real(x)>=0);
    x4=x(i4);
    if isempty(x4)==0
        fun4=2/pi./x4.*polyval(nom,x4)./polyval(den,x4);
    else
        fun4=[];
    end
    fun=x*0;
    fun(i1)=fun1;
    fun(i2)=fun2;
    fun(i3)=fun3;
    fun(i4)=fun4;
    %
    fun=reshape(fun,size(z));
    %
end

function eval = cheval(Ctype,An,xv)
    %
    % cheval evaluates any one of the four types of Chebyshev series.
    % It is a Matlab translation of the Fortran function EVAL
    % found in page 20 of:
    % Y.L. Luke, Algorithms for the computation of mathematical functions
    % Academic Press, 1977, p.20
    %
    % Author : T.P. Theodoulidis
```

```
    % Date   : 11 June 2012
    %
    % Ctype (string): type of Chebyshev polynomial
    %                 'regular' T_n(x)
    %                 'shifted' T*_n(x)
    %                 'even'    T_2n(x)
    %                 'odd'     T_2n+1(x)
    % An            : vector of Chebyshev coefficients
    % z             : argument, can be scalar, vector, matrix
    %
    switch Ctype
        case {'regular'}
            log1=1; log2=1;
        case {'shifted'}
            log1=1; log2=0;
        case {'even'}
            log1=0; log2=1;
        case {'odd'}
            log1=0; log2=0;
        otherwise
            return;
    end
    %
    x=xv(:);
    %
    xfac=2*(2*x-1);
    if log1 && log2, xfac=2*x; end
    if ~log1, xfac=2*(2*x.*x-1); end
    n=length(An);
    n1=n+1;
    Bn=zeros(length(x),n1+1);
    %
    for j=1:n
        Bn(:,n1-j)=xfac.*Bn(:,n1+1-j)-Bn(:,n1+2-j)+An(n1-j);
    end
    eval=Bn(:,1)-xfac.*Bn(:,2)/2;
    if ~log1 && ~log2, eval=x.*(Bn(:,1)-Bn(:,2)); end
    eval=reshape(eval,size(xv));
    %
end

function fun=StruveH1(z)
    %
    % StruveH1 calculates the function StruveH1 for complex argument z
    %
    % Author : T.P. Theodoulidis
    % Date   : 11 June 2012
    % Revised: 28 June 2012
    %
    % Arguments
    % z : can be scalar, vector, matrix
    %
    % External routines called         : cheval, StruveH1Y1
    % Matlab intrinsic routines called : bessely, besselh
    %
    bn=[1.174772580755468e-001 -2.063239340271849e-001  1.751320915325495e-001...
       -1.476097803805857e-001  1.182404335502399e-001 -9.137328954211181e-002...
        6.802445516286525e-002 -4.319280526221906e-002  2.138865768076921e-002...
       -8.127801352215093e-003  2.408890594971285e-003 -5.700262395462067e-004...
        1.101362259325982e-004 -1.771568288128481e-005  2.411640097064378e-006...
       -2.817186005983407e-007  2.857457024734533e-008 -2.542050586813256e-009...
        2.000851282790685e-010 -1.404022573627935e-011  8.842338744683481e-013...
       -5.027697609094073e-014  2.594649322424009e-015 -1.221125551378858e-016...
        5.263554297072107e-018 -2.086067833557006e-019  7.628743889512747e-021...
       -2.582665191720707e-022  8.118488058768003e-024 -2.376158518887718e-025...
        6.492040011606459e-027 -1.659684657836811e-028  3.978970933012760e-030...
       -8.964275720784261e-032  1.901515474817625e-033];
    %
```

```
    x=z(:);
    %
    % |x|<=16
    i1=abs(x)<=16;
    x1=x(i1);
    i2=~i1;
    x2=x(i2);
    if isempty(x1)==0
        z1=x1.^2/400;
        fun1=cheval('shifted',bn,z1).*x1.^2*2/3/pi;
    else
        fun1=[];
    end
    %
    % |x|>16
    if isempty(x2)==0
        fun2=StruveH1Y1(x2)+bessely(1,x2);
    else
        fun2=[];
    end
    %
    fun=x*0;
    fun(i1)=fun1;
    fun(i2)=fun2;
    %
    fun=reshape(fun,size(z));
    %
end

function fun=StruveH1Y1(z)
    %
    % StruveH1Y1 calculates the function StruveH1-BesselY1 for complex argument z
    %
    % Author : T.P. Theodoulidis
    % Date   : 11 June 2012
    %
    % Arguments
    % z : can be scalar, vector, matrix
    %
    % External routines called         : cheval, StruveH1
    % Matlab intrinsic routines called : bessely, besselh
    %
    nom=[4,0,9648              ,0,8187030            ,...
           0,3120922350        ,0,568839210030       ,...
           0,49108208584050    ,0,1884052853216100   ,...
           0,28131914180758500,0,126232526316723750,...
           0,97007862050064000,0,2246438344775625];
    %
    den=[4,0,9660               ,0,8215830            ,...
           0,3145141440         ,0,577919739600       ,...
           0,50712457149900     ,0,2014411492343250   ,...
           0,32559467386446000  ,0,177511711616489250,...
           0,230107774317671250,0,31378332861500625];
    %
    x=z(:);
    %
    % |x|<=16
    i1=abs(x)<=16;
    x1=x(i1);
    if isempty(x1)==0
        fun1=StruveH1(x1)-bessely(1,x1);
    else
        fun1=[];
    end
    %
    % |x|>16 and real(x)<0 and imag(x)<0
    i2=(abs(x)>16 & real(x)<0 & imag(x)<0);
    x2=x(i2);
```

```
    if isempty(x2)==0
        x2=-x2;
        fun2=2/pi+2/pi./x2.^2.*polyval(nom,x2)./polyval(den,x2)-2i*besselh(1,1,x2);
    else
        fun2=[];
    end
    % |x|>16 and real(x)<0 and imag(x)>=0
    i3=(abs(x)>16 & real(x)<0 & imag(x)>=0);
    x3=x(i3);
    if isempty(x3)==0
        x3=-x3;
        fun3=2/pi+2/pi./x3.^2.*polyval(nom,x3)./polyval(den,x3)+2i*besselh(1,2,x3);
    else
        fun3=[];
    end
    % |x|>16 and real(x)>=0
    i4=(abs(x)>16 & real(x)>=0);
    x4=x(i4);
    if isempty(x4)==0
        fun4=2/pi+2/pi./x4.^2.*polyval(nom,x4)./polyval(den,x4);
    else
        fun4=[];
    end
    fun=x*0;
    fun(i1)=fun1;
    fun(i2)=fun2;
    fun(i3)=fun3;
    fun(i4)=fun4;
    %
    fun=reshape(fun,size(z));
    %
end

%Here we define the function for Real(\alpha(k,p)), k being the coordinate
%|k| and p containing the information on the parameter regime
function y=Rea(k,p)
    gamma = p.eta/p.k;
    phi = 3*p.A/p.r1./k.^2.*(-2*besselj(2,p.r1.*k)+pi*besselj(1,p.r1.*k).*StruveH0(p.r1.*k)-
    pi*besselj(0,p.r1.*k).*StruveH1(p.r1.*k));
    psi = 3*p.AR/p.r./k.^2.*(-2*besselj(2,p.r.*k)+pi*besselj(1,p.r.*k).*StruveH0(p.r.*k)-pi*
    besselj(0,p.r.*k).*StruveH1(p.r.*k));
    y = k.^2/p.zeta.*(k.^2*gamma/p.eta.*phi.^2-psi);
end

%This function identifies the maximum point on the input function, with
%accuracy s^2
function [x,y] = high(fun,s,p)
    X = linspace(0.0001,1000,s);
    [~,x0] = max(fun(X,p));
    x0 = 0.0001+999.9999/(s-1)*x0;
    %once an approximate point is found we try to find a more accurate
    %point in the neighbourhood
    X = linspace(x0-1,x0+1,s);
    [y,x1]=max(fun(X,p));
    x = x0-1+2/(s-1)*x1;
end

%This function produces the pattern size 'dim' and returns a statistic on
%whether a pattern is expected 'stat', as well as, the actual size of the
%maximum point 'max'
function [stat,dim,max]=patest(p)
    alpha = @Rea;
    [x0,y] = high(alpha,10000,p);
    max = y;
    if y <= 0
        stat = 'No Pattern';
        dim = 0;        %when there is no pattern we say that the pattern size is 0
    else
```

```
        stat = 'Pattern';
        dim = 2*pi/x0;
    end
end
end
```

A.4 Pattern Size Measure

```
function [pattNr, pattDir, score]=get_fourier_results(xPos, yPos,nr)

%This function takes in the x-coordinates of the SPPs 'xPos', the
%y-coordinates 'yPos' and the number of the peak we want 'nr' and returns
%the pattern number 'pattNr', its direction 'pattDir' and the score it
%assigns to this results' accuracy 'score'.

isPlot=1;

N=length(xPos);

%% Define radius and direction values

nn=200;
rad=linspace(1.5,20,nn);
% phi=linspace(0,pi/2,nn);
phi=linspace(0,pi,nn);

[R,P]=meshgrid(rad,phi);

%% Calculate Fourier coefficient
fourierCoeff=0;

for i_part=1:N
    fourierCoeff=fourierCoeff+exp(-2*pi*1i*(R.*cos(P)*xPos(i_part)+R.*sin(P)*yPos(i_part)));
end

fourierCoeff=fourierCoeff/N;

% determine absolute value
absFourier=abs(fourierCoeff);

%% Find Peaks

peakIndices=FastPeakFind(absFourier);
xi=peakIndices(1:2:end);
yi=peakIndices(2:2:end);


% determine linear indices
ind=sub2ind(size(absFourier),yi,xi);
[vals, IX]=sort(absFourier(ind), 'desc');

pattNr=rad(xi(IX(nr)));
pattDir=phi(yi(IX(nr)));

% determine score of highest peak
% score=(vals(1)-vals(2))/vals(1);
score=vals(1);

if isPlot==1

    clf;

    subplot(1,2,1)
    cla; hold on
    title('SPP Positions')
    scatter(xPos, yPos,'*')
    axis equal
    xlim([-.5 .5])
```

```
    ylim([-.5 .5])

    subplot(1,2,2)
    cla; hold on
    title('Fourier Transform of SPP Density')
    imagesc(rad,phi,absFourier)
    scatter(rad(xi), phi(yi), 10, 'k', 'filled')
    scatter(pattNr, pattDir, 50, 'k', 'filled')


    xlabel('Pattern Number r')
    ylabel('\theta')
    colorbar
    colormap jet
    axis tight
end
```

References

[1] Pedro Aceves-Sanchez, Pierre Degond, Eric E. Keaveny, Angelika Manhart, Sara Merino-Aceituno, and Diane Peurichard. Large-scale dynamics of self-propelled particles moving through obstacles: model derivation and pattern formation. *arXiv e-prints*, page arXiv:2004.12638, April 2020.

[2] Francesco Ginelli. The Physics of the Vicsek model. *European Physical Journal Special Topics*, 225(11), November 2016.

[3] Tamás Vicsek, András Czirók, Eshel Ben-Jacob, Inon Cohen, and Ofer Shochet. Novel type of phase transition in a system of self-driven particles. *Physical Review Letters*, 75(6):1226–1229, Aug 1995.