BERGISCHE
UNIVERSITÄT
WUPPERTAL

Lane Formation in Pedestrian Dynamics Simulations: A Stochastic Port-Hamiltonian Approach

Master's Thesis

by

Rafay Nawaid Alvi

Master's Program

Computer Simulation in Science

Faculty of Mathematics and Natural Science
Bergsiche Universität Wuppertal

Supervisors

Prof. Dr. Antoine Tordeux Prof. Dr. Barbara Rüdiger

March 24, 2025

Acknowledgement

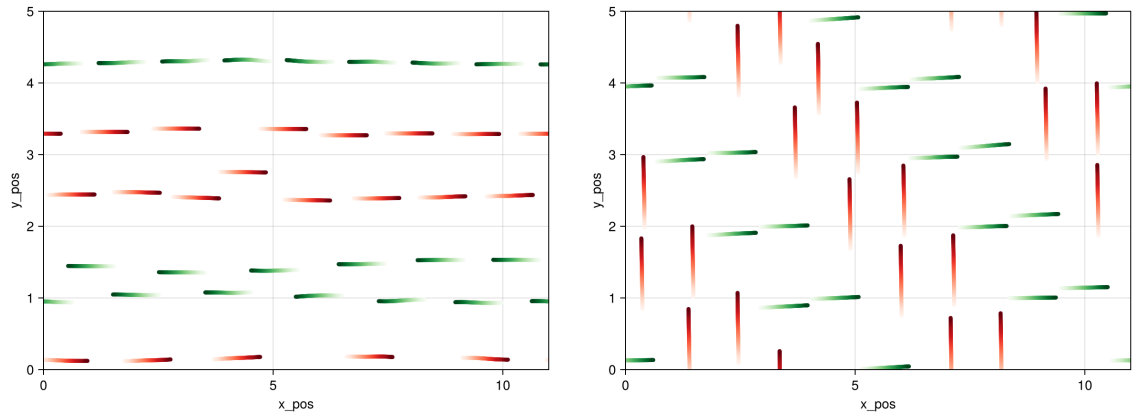
I am thankful to my supervisors Antoine Tordeux and Barbara Rüdiger, for their active support throughout the course of the thesis, and also for giving me an opportunity to embrace my curious endeavors by introducing this fascinating subject. I would also like to express my gratitude to the professors in computer simulation in science program for their lectures; in particular Lukas Arnold and Tristan Hehnen for their career advice and support throughout my university years.

A special thanks to my family, my beloved cats, and my small group of close friends, who've always been there with me through thick and thin, namely Muhammad Faisal Jamali, Haider Hussain, Qendresa Buzolli, Dhruv Patel, A. Raihan Bhuyian, and Zain Ul Haq Ansari. Their constant support and love were invaluable in helping me remain motivated.

Contents

1 Introduction

Collective dynamics is an interesting research area in many fields of study ranging from natural science, mathematics, to social science. The reasons for which are also far and wide, ranging from relevant practical applications to plain curiosity. Collective dynamics, as the name suggests, involve interactivity of a collection of objects [1] (e.g. crowds of pedestrians, flocks of birds, social systems etc.). These moving parts – under the right conditions – may exhibit a self-organizing characteristic that emerge out of the local interaction of the moving parts [2]. Examples of such collective behavior include oscillatory behavior of dense crowds [3], stop-and-go waves from traffic flows and crowd dynamics [4, 5], formation of opinions in a society [6] to name a few.



(a) Lane formation for counter flows

(b) Stripe formation for cross flows

Figure 1.1: Examples of collective phenomenon in pedestrian dynamics. The colors differentiate the pedestrians based on their desired directions.

Microscopic Model for Pedestrian Dynamics

In this study, we pay our attention to the collective behavior observed in pedestrian models using the idea from Helbing's social force model (SFM) [7]. The movement of pedestrians arises from a combination of *social forces* that affect them individually into taking certain directions as a result. Depending on the spacial boundaries, pedestrian interactivity, and their direction of movement, one can observe self-organizing patterns (see Fig. 1.1) such as lane formation for counter-flows i.e. when two groups of pedestrians move in opposite directions (e.g. left-to-right and right-to-left), and stripe/band formation for cross-flows i.e. when two groups of pedestrians move in perpendicular directions (e.g. left-to-right and up-to-down) [8, 9, 10, 11]. These macroscopic patterns emerge out of the microscopic interaction between individual pedestrians, serving as one of the examples of emergent phenomenon within complex systems [12].

Port-Hamiltonian Systems Approach

It is understandable that due to the involved interactivity and self-propelled behavior, the resulting nonlinear dynamics poses quite a challenge in modeling and controlling pedestrian systems [13]. In this thesis, we will attempt to describe the dynamics

through a modeling approach known as port-Hamiltonian systems (PHS), giving an energy-based perspective to describe the overall behavior of the model.

Port-Hamiltonian systems theory is a relatively new, yet promising, paradigm to model nonlinear physical systems [14, 15]. Extending the classical Hamiltonian mechanics to allow modeling of open systems, as it allows systems to interact with other systems through ports that serve as inputs and outputs for the flow of energy between them. The framework is used for control for nonlinear multiphysics systems as well [16, 17]. Microscopic models are described by finite-dimensional PHS, which has been used to model various multi-agent systems including consensus and opinion formation [18, 19], swarm behavior [20], multi-input multi-output agent systems [21]. Whereas macroscopic systems are modelled using infinite-dimensional PHS, these include fluid dynamic models [22], and macroscopic traffic flow models [23]. To account for uncertainties in the system, the theory of port-Hamiltonian systems also extends to stochastic cases [24], involving stochastic car-following models [25], and more generally self-driven agent models [26].

Overview and Objectives

The objective of this thesis is to represent the dynamics of the microscopic force-based pedestrian model through the port-Hamiltonian formulation. We establish that the port-Hamiltonian formulation is interchangeable with the agent-based formulation, signifying that the model possesses Hamiltonian structure. Once this is established, we devise that the Hamiltonian can be used as a measure to quantify macroscopic behavior of the overall model; and observe the relationship between the self-organized collective behavior and the overall system energy, i.e. the Hamiltonian. Next, we move on to the stochastic case, and subject the dynamics to additive noise. We investigate the dynamics of the stochastic port-Hamiltonian formulation of the model and observe its impact on the Hamiltonian measure as well as on the macroscopic collective behavior. We witness how minor addition of noise can also improve the self-organization in cross-flow compared to the case with no noise.

Furthermore, we also show the respective codes that were used in the model to create numerical solvers, and the functions needed to create the agent-based model. Next, a complete coding example is shown to demonstrate how the model is constructed and simulated with data collection, this step-by-step also includes code to plot the model attributes such as the Hamiltonian, and agent attributes such the pedestrian trajectories.

To simulate the models, we keep in mind that the numerical method should be considerate of the energy of the system, and thus we compare different solvers and establish that symplectic methods are best suited for our port-Hamiltonian model. The agent-based models are made using the Julia programming language with `Agents.jl` [27] package. To add more flexibility and to easily compare a variety of different numerical solvers, we also make use of the `DifferentialEquations.jl` [28] package to construct our solvers, this proved useful particularly for the stochastic port-Hamiltonian case.

2 Microscopic Pedestrian Model

We will first construct the pedestrian model focusing on the interaction of agents to each other i.e. in the microscopic scale. In contrast, the macroscopic scale would focus on the resulting dynamics of the agents' interactions as a collective, which will be presented in more detail in the next section.

2.1 Pedestrian Attributes

Let us first start by describing our model. We follow the approach presented in [11] which follows a general class of microscopic force-based models approach given in [7, 29]. The pedestrians exist on a toroidal surface, in other words, our model has periodic boundaries. For simplicity, the mass for all pedestrians is set to 1, hence their momentum p_i is equal to the velocity, i.e. $p_i = m_i v_i = v_i$. The pedestrians also posses a desired velocity, which is assigned to each of them as an input to the model, this directs the pedestrians to desire a certain direction to move towards during the simulation run.

Given a pedestrian i in $\mathbb{R}^2$ and time $0 \leq t \leq T$, it possesses the following attributes:

- Desired velocity: $u_i(t) : [0, T] \mapsto \mathbb{R}^2$
- Current velocity: $p_i(t) : [0, T] \mapsto \mathbb{R}^2$
- Current position: $q_i(t) : [0, T] \mapsto \mathbb{R}^2$

```julia
## Define Agent
using Agents
@agent struct Pedestrian(ContinuousAgent{2,Float64})
    u_i::Vector{Float64} # Desired Velocity
end
```

Code 2.1: Defining the pedestrian agent in Julia's Agents.jl package. It is to be noted that `ContinuousAgent` specifies that our Pedestrian is a continuous agent with predefined position and velocity attributes constructed within the `@agent` macro. We only need to declare additional attributes such as `u_i`

For a model with $N \geq 2$ pedestrians, the relative position $\dot{Q}_{ij}(t)$ of a pedestrian i to pedestrian j is denoted as follows

$$Q_{ij}(t) = q_i - q_j$$

The vector of relative positions $Q_i(t)$ of a pedestrian i to all pedestrians $j \neq i$ is denoted as follows

$$Q_i(t) = (Q_{ij}(t))_{i \neq j} : [0, T] \mapsto \mathbb{R}^{2 \cdot (N-1)} \text{ for } j = 1, \ldots, N, \quad j \neq i$$

2.2 Microscopic Model Dynamics

The dynamics of the pedestrians are based on (i) short-range repulsion U among pedestrians based on their distances from others and (ii) attraction λ toward a desired velocity u_i. The microscopic dynamics for the pedestrian i is given by the following equations:

$$\begin{aligned} \dot{Q}_{ij}(t) &= p_i(t) - p_j(t), & j = 1, \ldots, N \quad j \neq i \quad \dot{Q}_i(0) &= Q_i^0 \\ \dot{p}_i(t) &= \lambda(u_i(t) - p_i(t)) - \sum_{j \neq i} \nabla U(Q_{ij}(t)), & \dot{p}_i(0) &= p_i^0 \end{aligned} \tag{2.1}$$

Here,

- $\lambda \in \mathbb{R}_{\geq 0}$, is a parameter for the relaxation rate. In other words, it determines the intensity to reach the desired velocity.
- $U(Q_{ij})$, is a nonlinear repulsive interaction potential, such that it is always positive and convex. This determines the magnitude of repulsion of a pedestrian i to other pedestrians j as a function of relative position $Q_{ij} = q_i - q_j$. It is defined as follows

$$U(x) : \mathbb{R}^2 \mapsto \mathbb{R}, \qquad U(x) = ABe^{-|x|/B} \tag{2.2}$$

  Here, $|\cdot|$ is the minimum euclidean distance on the torus. A and B are scalar-valued parameters for the repulsion strength and the range of interaction respectively. Note that $U(x)$ is an even function, such that $\forall x \in \mathbb{R}^n, U(-x) = U(x)$.

Hence, $\nabla U(x) : \mathbb{R}^2 \mapsto \mathbb{R}^2$ is defined as

$$\nabla U(x) = -A\frac{x}{|x|}e^{-|x|/B} = -\nabla U(-x) \tag{2.3}$$

It can be seen that this parameter is responsible for avoiding collisions between pedestrians, ensuring that the repulsive forces increase in magnitude the closer a pedestrian is. The underlying assumption is that the repulsive forces are a function of distance only. This is adapted from the concept of social forces and how they are determined from the surroundings of pedestrians. Also, note that $\nabla U(x)$ is an odd function.

Using the product rule, the Hessian $\mathbf{H}_U = \nabla^2 U(x) : \mathbb{R}^2 \mapsto \mathbb{R}^{2\times 2}$ can be expressed as:

$$\begin{aligned} \mathbf{H}_U = \nabla^2 U(x) &= -A\left(\frac{x}{|x|}\frac{\mathrm{d}}{\mathrm{d}x}\left[e^{-|x|/B}\right] + e^{-|x|/B}\frac{\mathrm{d}}{\mathrm{d}x}\left[\frac{x}{|x|}\right]\right) \\ &= \nabla^2 U(x) = -A\left[\frac{1}{|x|}\left(I - \frac{xx^T}{|x|^2}\right) - \frac{1}{B}\frac{xx^T}{|x|^2}\right]e^{-|x|/B} \end{aligned} \tag{2.4}$$

Which will be used to evaluate the stochastic derivative in Section. 3.5

2.3 Stochastic Model Dynamics

To study the stochastic variant of our model, and how they effect the overall dynamics, we will introduce stochastic terms into our model. Generally a typical stochastic

differential equation (SDE) is of the form

$$\mathrm{d}X_t = \underbrace{f(X_t, t)\mathrm{d}t}_{\text{drift}} + \underbrace{g(X_t, t)\mathrm{d}W_t}_{\text{diffusion}}$$

Which is a combination of a deterministic component, denoted as the *drift* term; and a stochastic component, denoted as the *diffusion* term. In our case, we introduce an additive noise term $\sigma dW_i(t)$ to our deterministic model from before Eq. 2.1 resulting in Eq. 2.5. To be able to have a better physical interpretation of the model, the noise is introduced in the momentum term, rather than the position term. So one can interpret this as agents moving randomly while maintaining a continuous motion, as opposed to randomly teleporting in different locations at every time-step.

$$\begin{aligned} \mathrm{d}Q_{ij}(t) &= (p_i(t) - p_j(t))\mathrm{d}t \\ \mathrm{d}p_i(t) &= \lambda(u_i(t) - p_i(t))\mathrm{d}t - \sum_{j \neq i} \nabla U(Q_{ij}(t))\mathrm{d}t + \sigma \mathrm{d}W_i(t) \end{aligned} \tag{2.5}$$

The equation is reminiscent of a typical SDE with the first two terms being the deterministic part of the equation, i.e. drift and the last term being the stochastic part, i.e. the diffusion term. The randomness is induced via the Wiener process $(W_i(t))_{i=1}^N : [0, \infty) \times \Omega \to \mathbb{R}^N$, which is a mathematical description of the Brownian motion, that generates a sequence of random variables defined on a probability space (Ω, F, P) with independent increments [30]. Here, Ω is the sample space, F the event space, and P the probability measure. The stochastic term is amplified by the constant $\sigma \in \mathbb{R}$, which is often denoted as the volatility or diffusion coefficient.

2.4 Identifying Collective Phenomenon

With the dynamics presented so far, we can observe the individual pedestrian's attributes change over time. However, to identify whether a collective phenomenon is occurring we have also constructed a simple formula to measure the mean direction of all agents relative to their desired velocity direction, i.e. direction alignment ϕ.

To measure how much an agent i is moving in the direction of its particular desired velocity u_i, we can take the following dot product.

$$\phi_i = \frac{p_i}{|p_i|} \cdot \frac{u_i}{|u_i|}, \quad \phi \in \mathbb{R}, \quad \phi \in [-1, 1] \tag{2.6}$$

The range of ϕ is $[-1, 1]$, with 1 indicating that the pedestrian is moving exactly in its desired direction, -1 indicating that the pedestrian is moving exactly opposite to its desired direction, whereas 0 indicates that it is either not moving or it is moving perpendicular to its desired speed. We can use this measure to quantify the mean direction of all agents. In other words, how aligned the mean direction of all agents is to their desired direction.

$$\begin{aligned} \text{Alignment} &= \frac{1}{N} \sum_{i=1}^{N} \phi_i \\ &= \frac{1}{N} \hat{p}^T \hat{u} \end{aligned} \tag{2.7}$$

This will be useful, as we will later see, in identifying ordered and disordered collective dynamics. From a collective point of view, we can say that the value near 1 or -1 will indicate orderly behavior, signalling that all pedestrians are moving in their respective desired directions. Whereas values near 0 will indicate disorder.

Calculation of Alignment

In our simulations, we can take advantage of the fact that u_i is always going to be a unit vector, i.e. either $[1, 0]$, $[0, 1]$, or $[0, 0]$. Thus, we only use the unit vector of the velocity in the calculation of the alignment from Eq. 2.7, as shown in the code below.

```
function calc_alignment(model)
    return mean(
        [transpose(i.vel/norm(i.vel))*(i.u_i) for i in allagents(model)]
        )
end
```

Code 2.2: Calculation of Mean direction of all pedestrians

The benefit of this measure is made apparent in Fig. 2.1. Here, we see how the mean direction of all the pedestrians gradually aligns with their desired velocity direction as the value moves from 0 to 1, showing the presence of order in the system.

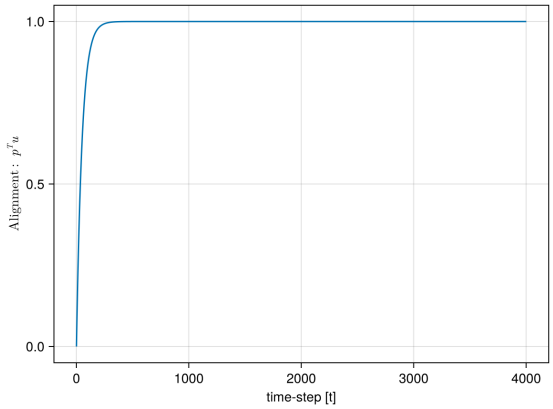


Figure 2.1: Alignment for unidirectional flow.

Through this measure, we can witness a shift from order to disorder by means of pedestrians aligning with their respective desired direction. Examples of this with various other cases will be shown in Section. 5.

3 Port-Hamiltonian Formulation of the Microscopic Model

This section serves to represent the microscopic dynamics of our pedestrian model using the framework of port-Hamiltonian systems, the generalized finite-dimensional PH formulation is given as:

$$\begin{aligned} \dot{z}(t) &= (J - R)\nabla H(z(t)) + Bu(t) \\ y(t) &= B^T \nabla H(z(t)) \end{aligned} \tag{3.1}$$

3.1 Preliminaries

Given a port-Hamiltonian System (PHS), such as the one defined in Eq. 3.1, it is able to gain energy from or dissipate energy to other interacting systems. Here, z is the state of the system, u is the input, and y is the output of the system. Furthermore, $J \in \mathbb{R}^{n\times n}$ is a skew-symmetric matrix, $R \in \mathbb{R}^{n\times n}$ is a positive semi-definite matrix that represents dissipation from the system, and $B \in \mathbb{R}^{n\times m}$ where $m \ll n$ being the coefficient for the input.

One can achieve the classical Hamiltonian representation by assuming a system with no input, i.e. $B = 0$, and no dissipation $R = 0$ as shown below using the first equation from Eq. 3.1

$$\begin{aligned} \dot{z}(t) &= J \cdot \nabla H \\ \begin{bmatrix} \dot{q}(t) \\ \dot{p}(t) \end{bmatrix} &= \begin{bmatrix} 0 & I_n \\ -I_n & 0 \end{bmatrix} \begin{bmatrix} \partial H/\partial q \\ \partial H/\partial p \end{bmatrix} \end{aligned} \tag{3.2}$$

By setting the block matrices I to size $n = 1$, it can be demonstrated that the above equations yield the classical Hamiltonian equations, signifying that the skew-symmetry of J is responsible for the conservation of the system.

$$\dot{q} = \frac{\partial H}{\partial p}, \quad \dot{p} = -\frac{\partial H}{\partial q}$$

Having $R > 0$ allows the system to dissipate energy and the input u allows the system to gain energy. As the inputs and outputs allows the system to interact with other systems, these can be denoted as ports of the system.

3.2 Port-Hamiltonian Model

For a system representing the microscopic model of $N > 2$ pedestrians, the state $z(t)$ would contain the relative positions and velocities of all pedestrians at a given time: $z(t) = (Q, p)^T$, where $Q = (Q_1, \ldots, Q_N)$ and $p = (p_1, \ldots, p_N)$ are vectors of relative positions and velocities of all pedestrians respectively. Such a system, as shown in Eq. 2.1 can be described through port-Hamiltonian formulation presented in Eq. 3.1 as follows:

$$\begin{aligned} \dot{z}(t) &= (J - R)\nabla H(z(t)) + \lambda\tilde{u}(t), \qquad z(0) = (Q^0, p^0) \\ y(t) &= \lambda \nabla H(z(t)) \end{aligned} \tag{3.3}$$

Here,

$$\dot{z}(t) = \begin{bmatrix} Q_1 & \cdots & Q_N & p_1 & \cdots & p_N \end{bmatrix}^T \tag{3.4}$$

$$\tilde{u} = \begin{bmatrix} 0 & \cdots & 0 & u_1(t) & \cdots & u_N(t) \end{bmatrix}^T \tag{3.5}$$

$$J = \begin{bmatrix} 0 & M \\ -M^T & 0 \end{bmatrix}, \quad R = \begin{bmatrix} 0 & 0 \\ 0 & \lambda I \end{bmatrix}, \quad M = \begin{bmatrix} M_1 \\ \vdots \\ M_N \end{bmatrix} \tag{3.6}$$

The resulting system is a linear input-state-output port-Hamiltonian system with dissipation. The Hamiltonian can now be defined as the total energy of the system, i.e. the sum of kinetic and potential energies

$$H(z(t)) = \frac{1}{2}||p(t)||^2 + \frac{1}{2}\sum_{i=1}^{N}\sum_{j=i}^{N} U(Q_{ij}(t)) \tag{3.7}$$

The gradient of the Hamiltonian would then be

$$\nabla H = \begin{bmatrix} \partial H/\partial Q \\ \partial H/\partial p \end{bmatrix} = \begin{bmatrix} \frac{1}{2}\nabla U(Q) \\ p \end{bmatrix} \tag{3.8}$$

The matrix equation representation of Eq. 3.3 can be written as

$$\begin{bmatrix} \dot{Q} \\ \dot{p} \end{bmatrix} = \left(\begin{bmatrix} 0 & M \\ -M^T & 0 \end{bmatrix} - \begin{bmatrix} 0 & 0 \\ 0 & \lambda I \end{bmatrix} \right) \begin{bmatrix} \frac{1}{2}\nabla U(Q) \\ p \end{bmatrix} + \lambda \begin{bmatrix} 0 \\ u \end{bmatrix} \tag{3.9}$$

The block matrix M can be defined such that

$$\dot{Q} = Mp \tag{3.10}$$

with its elements

$$M_1 = \begin{bmatrix} 1 & -1 & 0 & 0 & \dots & 0 \\ 1 & 0 & -1 & 0 & \dots & 0 \\ \vdots & & & & & \vdots \\ 1 & & & & & -1 \end{bmatrix}, \quad M_2 = \begin{bmatrix} -1 & 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & -1 & 0 & \dots & 0 \\ \vdots & \vdots & & & & \vdots \\ 0 & 1 & & & & -1 \end{bmatrix}, \quad \dots \tag{3.11}$$

R is positive semi-definite

One can show that R is positive semi-definite by showing $x^T Rx \geq 0$, where $x \in \mathbb{R}^n$ is a real-valued vector $x = [x_1, \dots x_N]$:

$$\begin{aligned} x^T Rx &= x^T \begin{bmatrix} 0 & 0 \\ 0 & \lambda I \end{bmatrix} x \\ &= \begin{bmatrix} x_1 \dots x_N \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & \lambda I \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix} \\ &= \begin{bmatrix} 0 & \lambda x_2 & \dots & \lambda x_N \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix} \\ &= \lambda(x_2^2 + \dots + x_N^2) \geq 0 \end{aligned}$$

Since $\lambda \geq 0$ and the squared terms are also positive, we can conclude that last expression as a whole is also greater than or equal to zero, proving that $x^T R x \geq 0$

It can be observed that the parameter λ is present in both the dissipation term R and the coefficient term of the input $\tilde{u}$.

Example: $N = 3$

To better illustrate the implementation of the model, here we will concretely show the formulation of the model using $N = 3$ pedestrians. Here the matrix $M = (M_1, M_2, M_3)^T$, where each M_i shows the difference in quantities with respect to the pedestrian i as shown below.

$$M_1 = \begin{bmatrix} 1 & -1 & 0 \\ 1 & 0 & -1 \end{bmatrix},\ M_2 = \begin{bmatrix} -1 & 1 & 0 \\ 0 & 1 & -1 \end{bmatrix},\ M_3 = \begin{bmatrix} -1 & 0 & 1 \\ 0 & -1 & 1 \end{bmatrix}$$

It can be observed that M unfolds the relative velocities of pedestrians, resembling the dynamics presented in the first equation of Eq. 2.1

$$M = \begin{bmatrix} M_1 \\ M_2 \\ M_3 \end{bmatrix}, \qquad \dot{Q} = Mp = \begin{bmatrix} p_1 - p_2 \\ p_1 - p_3 \\ p_2 - p_1 \\ p_2 - p_3 \\ p_3 - p_1 \\ p_3 - p_2 \end{bmatrix}$$

Lastly, we represent the Hamiltonian

$$H = \frac{1}{2}(p_1^2 + p_2^2 + p_3^2) + \frac{1}{2}(U(Q_{12}) + U(Q_{13}) + U(Q_{21}) + U(Q_{23}) + U(Q_{31}) + U(Q_{32}))$$
$$H = \frac{1}{2}(p_1^2 + p_2^2 + p_3^2) + U(Q_{12}) + U(Q_{13}) + U(Q_{23})$$

To obtain the simplified expression of H, the relationship $Q_{21} = -Q_{12}$ is amended by the property of $U(x)$ being even, as defined in Eq. 2.2.

3.3 Deriving the Equations of Motion

We can show that the port-Hamiltonian formulation Eq. 3.3 is indeed the same model as Eq. 2.1 by deriving the latter from the former. Let us start by simplifying the matrix equation Eq. 3.9 for $\dot{Q}$ and $\dot{p}$

$$\dot{Q} = \begin{bmatrix} 0 & M \end{bmatrix} - \begin{bmatrix} 0 & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{2}\nabla U(Q) \\ p \end{bmatrix} + \lambda \begin{bmatrix} 0 \end{bmatrix}$$

$$\dot{p} = \begin{bmatrix} \begin{bmatrix} -M^T & 0 \end{bmatrix} - \begin{bmatrix} 0 & \lambda I \end{bmatrix} \end{bmatrix} \begin{bmatrix} \frac{1}{2}\nabla U(Q) \\ p \end{bmatrix} + \lambda \begin{bmatrix} u \end{bmatrix}$$

Resolving $\dot{Q}$ would reiterate the relationship described in Eq. 3.10

$$\dot{Q} = Mp$$

And, resolving $\dot{p}$ gives us the equation for the velocity in matrix form

$$\dot{p} = -M^T \frac{1}{2} \nabla U(Q) - \lambda p + \lambda u$$

Using the definition of M from Eq. 3.6 and Eq. 3.11, we can construct the equations for Q for every $i = 1, \ldots, N$.

$$\dot{Q}_{ij} = p_i - p_j$$

Similarly, using the property that $U(x)$ is an odd function Eq. 2.3, we can derive $\dot{p}$

$$\dot{p}_i = \lambda(u_i - p_i) - \frac{1}{2} \sum_{j \neq i} (\nabla U(Q_{ij}) - \nabla U(Q_{ji}))$$
$$\dot{p}_i = \lambda(u_i - p_i) - \sum_{j \neq i} \nabla U(Q_{ij})$$

By showing the derivation of the same microscopic dynamics from the port-Hamiltonian formulation, we can establish that the proposed microscopic model for the pedestrian dynamics has Hamiltonian structure, and that the system can be described using the Hamiltonian H as a quantitative measure for the collective dynamics of the pedestrians.

3.4 Stochastic Port Hamiltonian Formulation

Continuing from the stochastic induced dynamics of the microscopic model in Eq. 2.5, we can use the port-Hamiltonian formulation of the deterministic dynamics Eq. 3.3 and represent it with the noise term.

$$\begin{aligned} \mathrm{d}z(t) &= (J - R)\nabla H(z(t))\mathrm{d}t + \lambda \tilde{u}(t)\mathrm{d}t + \sigma \mathrm{d}\xi(t) \\ \mathrm{d}y(t) &= \lambda \nabla H(z(t))\mathrm{d}t \end{aligned} \tag{3.12}$$

Similar to Eq. 3.9, and following the formulation from [25], the matrix equation representation would become

$$\begin{bmatrix} \mathrm{d}Q \\ \mathrm{d}p \end{bmatrix} = \left(\begin{bmatrix} 0 & M \\ -M^T & 0 \end{bmatrix} - \begin{bmatrix} 0 & 0 \\ 0 & \lambda I \end{bmatrix} \right) \begin{bmatrix} \frac{1}{2}\nabla U(Q) \\ p \end{bmatrix} \mathrm{d}t + \lambda \begin{bmatrix} 0 \\ u \end{bmatrix} \mathrm{d}t + \sigma \begin{bmatrix} 0 \\ \mathrm{d}W(t) \end{bmatrix} \tag{3.13}$$

This will be used as the basis of our stochastic system. Following from the dynamics presented in Eq. 2.5, it can be noted that the stochastic term is only effecting the momentum equation.

It can also be noted that the expression for the Hamiltonian Eq. 3.7 will remain the same, since the stochastic terms are included within the momentum term $\dot{p}$.

3.5 Macroscopic Observations from Hamiltonian Behavior

Deterministic port-Hamiltonian Model Case:

The time derivative of the Hamiltonian is derived by taking the following chain rule:

$$\frac{\mathrm{d}}{\mathrm{d}t}H(z(t)) = \nabla^T H(z(t)) \cdot \dot{z}(t)$$

Substituting $\dot{z}(t)$ from Eq. 3.3 denoting that $y = \lambda \nabla H$ and thus $y^T = \nabla^T H \lambda$, we get

$$\frac{\mathrm{d}}{\mathrm{d}t}H(z(t)) = \nabla^T H(z(t))(J-R)\nabla H(z(t)) + \underbrace{\nabla^T H(z(t))\lambda}_{y^T}\tilde{u}$$

Using the skew-symmetric property of J that $x^T J x = 0$ we can simplify and obtain the expression for the time derivative of the Hamiltonian.

$$\frac{\mathrm{d}}{\mathrm{d}t}H(z(t)) = y^T(z(t))\tilde{u}(t) - \nabla^T H(z(t))R\nabla H(z(t)) \tag{3.14}$$

Expanding the terms further by substituting the gradient of the Hamiltonian Eq. 3.8 and the definition of R Eq. 3.6 can help us further simplify the expression giving us the following energy balance

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}t}H(z(t)) &= \lambda \nabla^T H(z(t))\tilde{u} - \nabla^T H(z(t))R\nabla H(z(t)) \\
&= \lambda \begin{bmatrix} \partial H/\partial Q & \partial H/\partial p \end{bmatrix} \begin{bmatrix} 0 \\ u \end{bmatrix} - \begin{bmatrix} \partial H/\partial Q & \partial H/\partial p \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & \lambda I \end{bmatrix} \begin{bmatrix} \partial H/\partial Q \\ \partial H/\partial p \end{bmatrix} \\
&= \lambda p^T u - \lambda p^T p \\
&= \lambda p^T (u - p) \\
\frac{\mathrm{d}}{\mathrm{d}t}H(z(t)) &= \lambda \langle p(t), u(t) - p(t) \rangle
\end{aligned} \tag{3.15}$$

It can be seen that the time derivative of the Hamiltonian $\frac{d}{dt}H(z(t))$ in the deterministic case depends only on the velocities of the pedestrians.

With this, we can confirm the following claims

- The Hamiltonian remains constant $\frac{d}{dt}H(z(t)) = 0$ for all times $t \geq 0$ if the system has no dissipation $\lambda = 0$.

  $$\forall t \geq 0, \quad \frac{\mathrm{d}}{\mathrm{d}t}H(z(t)) = 0 \qquad \text{if} \qquad \lambda = 0$$

  Reiterating that purely Hamiltonian behavior is indeed conservative.

- If the desired velocities are zero, i.e. $u = 0$, then the Hamiltonian decreases over time

  $$\forall t \geq 0, \quad \frac{\mathrm{d}}{\mathrm{d}t}H(z(t)) \leq 0 \qquad \text{if} \qquad \forall i,\ u_i = 0$$

  Since the system is allowed to dissipate without input feed, the asymptotic behavior would yield crystallization, i.e. the pedestrians would stop moving.

- If all the pedestrians move at their desired velocities, i.e. $p(t) = u(t)$, then the time derivative is zero

$$\frac{\mathrm{d}}{\mathrm{d}t}H(z(t)) = 0 \qquad \text{if} \qquad \forall i,\ p_i(t) = u_i(t)$$

Stochastic port-Hamiltonian Case:

The expression for the stochastic derivative of the Hamiltonian $dH(z)$ can be derived using Ito's Lemma on the Hamiltonian $H(z)$:

$$\mathrm{d}H(z(t)) = \nabla^T H(z(t))\mathrm{d}z(t) + \frac{\sigma^2}{2}\mathrm{Tr}\{\nabla^2 H(z(t))\}\mathrm{d}t$$

With the last term being the trace $\mathrm{Tr}\{\}$ of the Hessian ∇^2 of the Hamiltonian H. Substituting $\mathrm{d}z(t)$ from Eq. 3.12 and following the same steps as before will give us

$$\mathrm{d}H(z(t)) = \underbrace{\mathcal{L}H(z(t))\mathrm{d}t}_{\text{drift}} + \underbrace{\sigma\nabla^T H(z(t))\mathrm{d}\xi}_{\text{diffusion}}$$

Where $\mathcal{L}$ can be denoted as the infinitesimal generator of Eq. 3.12 with Hamiltonian as the potential function [25], containing all the drift terms:

$$\mathcal{L}H(z(t)) = y^T(z(t))\tilde{u}(t) - \nabla^T H(z(t))R\nabla H(z(t)) + \frac{\sigma^2}{2}\mathrm{Tr}\{\nabla^2 H(z(t))\}$$

We see that the first two terms of the drift part simplifies to the same expression as before Eq. 3.15 from the deterministic case: $\lambda p^T(u-p)$. The last term is summation of all diagonal entries in the Hessian of H, we can simplify this as follows:

$$\begin{aligned}
\mathrm{Tr}\{\nabla^2 H(z(t))\} &= \sum_{i=1}^{N}\sum_{j\neq i}^{N}\frac{\partial^2 H}{\partial Q_{ij}} + \sum_{i=1}^{N}\frac{\partial^2 H}{\partial p_i} \\
&= \frac{1}{2}\sum_{i=1}^{N}\sum_{j\neq i}^{N}\frac{\partial \nabla U(Q_{ij})}{\partial Q_{ij}} + \sum_{i=1}^{N}\frac{\partial p_i}{\partial p_i} \\
&= \frac{1}{2}\sum_{i=1}^{N}\sum_{j\neq i}^{N}\mathrm{Tr}\{\nabla^2 U(Q_{ij})\} + N
\end{aligned} \tag{3.16}$$

Here, $\nabla^2 U(x)$ is defined as Eq. 2.4, and N is the number of pedestrians. Next, we'll move on to the diffusion term, and simplify it

$$\begin{aligned}
\nabla^T H(z(t))\sigma\mathrm{d}\xi &= \sigma\begin{bmatrix}\partial H/\partial Q & \partial H/\partial p\end{bmatrix}\begin{bmatrix}0\\ \mathrm{d}W\end{bmatrix} \\
&= \sigma p^T \mathrm{d}W
\end{aligned}$$

Combining the simplified drift and diffusion terms leads to the stochastic derivative of the Hamiltonian.

$$\mathrm{d}H(z(t)) = \left(\lambda p^T(u-p) + \frac{\sigma^2}{4}\sum_{i=1}^{N}\sum_{j\neq i}^{N}\mathrm{Tr}\{\nabla^2 U(Q_{ij})\} + \frac{\sigma^2}{2}N\right)\mathrm{d}t + \sigma p^T \mathrm{d}W \tag{3.17}$$

4 Numerical Schemes and Simulation

After establishing that the dynamics of our microscopic model possesses Hamiltonian structure, we can now show these dynamics in action through simulations. In this section we will construct and compare schemes for both the stochastic and deterministic models. For the simulation, the following JuliaLang packages are extensively utilized, `Agents.jl` and `DifferentialEquations.jl`. All the code in this section can be found in the project directory `./PH_Project/src/module_ph_ped.jl`.

4.1 Model Setup

The simulation is run on a fixed model setup as shown in Code. 4.1. Here we initialize the values for the number of pedestrians, the extent of the domain in 2D, the desired velocities of the pedestrians, and the parameter values of λ, A, B, and σ.

Pedestrian Sensitivity:	$\lambda = 2$
Desired Speed:	$\|u\| = 1$
Repulsion Strength:	$A = 5$
Interaction Range:	$B = 0.3$
Volatility:	$\sigma = 0.0$

These parameter values are chosen to replicate the results from [11], which follows the works of [7, 31]. For the stochastic case, we replicate the results from [10]. The volatility coefficient σ is only relevant for stochastic cases – as we will see in later sections, whereas in deterministic cases it remains zero.

```
properties = Dict(
    :λ => 2,
    :A => 5,
    :B => 0.3,
    :dt => 0.01,
    :σ => 0.0,
    :Hamiltonian => 0.0,
    :dH => 0.0
)
number_of_peds = 32 # number of pedestrians
x_len = 11 # domain length in x
y_len = 5 # domain length in y
seed = 42
model = initialize(number_of_peds, x_len, y_len, leapfrog_step, properties,
                   seed)
```

Code 4.1: Initialization of the model

The `initialize` function shown in Code. 4.1 initializes the model with the given parameters. It also utilizes the functionalities provided by the `Agents.jl` package shown in Code. 4.2 such as `ContinuousSpace` and `StandardABM`.

```
space = ContinuousSpace((x_len, y_len); periodic = true)
model = StandardABM(
    Pedestrian,
    space;
    model_step! = (model) -> model_step!(model, num_solver),
    properties,
    rng # random number generator
)
```

Code 4.2: Inside the `initialize` function from Code. 4.1, showing that `ContinuousSpace` method takes care of the space as well as the periodic boundaries

The argument `num_solver` is used to specify an ODE solver. As an example Code. 4.1 uses `leapfrog_step` from Code. 4.7 as a solver. The construction of which is discussed in Section. 4.2. Additionally, one can construct other methods to be used in place of `num_solver`, such methods can be constructed using the solvers provided by the `DifferentialEquations.jl` package. These powerful solvers are used for complex scenarios such as solving SDEs (Stochastic Differential Equations) as will be discussed later in this section.

Once the model is initialized, it places the specified number of pedestrians randomly on the grid Code. 4.3. Here `rng` makes use of a Random Number Generator with a specified `seed` that remains fixed for the model. This is done so that simulation results can be replicated.

```
for i in 1:number_of_peds
    add_agent!(model;
        pos = [rand(rng)*x_len, rand(rng)*y_len],  # Initial position
        vel = [0.0,0.0], # Initial velocity
        u_i = [1.0,0.0] # uni-directional flow
        # u_i = mod(i,2) == 0 ? [1,0] : [-1,0] # counter_flow
        # u_i = mod(i,2) == 0 ? [0,1] : [1,0] # cross_flow
        # u_i = [0.0, 0.0] # no desired velocity : crystallization
    )
end
```

Code 4.3: Code snippet inside `initialize`, here every pedestrian is initialized with a position in the space and a desired velocity (u_i). It can be seen that we can customize the desired velocities to initiate counter flow and cross flow.

We can also now define a `model_step!` function Code. 4.4 that can be used to run our model. At each time-step of the simulation this `model_step` takes in the results of the solver used for every agent and modifies the values of each agent. It also calculates the Hamiltonian and its time derivative at each step which are defined in Section. 4.3

```
function model_step!(model,num_solver)
    random_list = [] # for sdesolver
    for agent in allagents(model)
        v, p, rest... = num_solver(agent,model)
        if length(rest) > 0
            push!(random_list, rest[1])
        end
        p = normalize_position(p,model)
        agent.vel = v
        agent.pos = p
    end
    model.hamiltonian = calc_hamiltonian(model)
    model.dH = calc_dH(model)
    model.no_disp_H = calc_no_disp_hamiltonian(model)
    model.alignment = calc_alignment(model)
    model.stoch_dH = calc_stoch_dH(model, random_list)
end
```

Code 4.4: At each step simulation of the model, `model_step!` updates the model parameters as well as the agent parameters.

4.2 Construction of Numerical Schemes

It is well-established that the preferred numerical methods for Hamiltonian systems are the symplectic methods. These methods are designed such that they preserve the Hamiltonian structure, in other words, they take into account the conservation of energy and hence are better to suited to solve physical systems.

For every pedestrian i, the numerical schemes solve for two equations: the positions q^{k+1} and velocities p^{k+1} at time-step δt. We recall our dynamic equations from Eq. 2.1, however we write it a bit differently.

$$\begin{aligned} \dot{p}_i &= a(q_i, p_i) \\ \dot{q}_i &= p_i \end{aligned} \tag{4.1}$$

For clarity, we emphasize the acceleration function and denote it as $a_i(q, p)$ Eq. 4.2. We use this representation to illustrate the numerical methods used.

$$a_i(q, p) = \lambda(u_i - p_i) - \sum_{j \neq i} \nabla U(q_i - q_j) \tag{4.2}$$

In Code. 4.6 we define the acceleration function, however we first construct a function `dU` in Code. 4.5 to calculate an interaction potential given the positions `(pos1, pos2)` of a pair of pedestrians, this is needed in the second term of Eq. 4.2, which sums all interaction potentials given the pedestrian i.

```
function dU(pos1, pos2, model)
    A = abmproperties(model)[:A]
    B = abmproperties(model)[:B]
    dist = euclidean_distance(pos1, pos2, model)
    result = ((get_direction(pos1,pos2,model))/dist)*A*exp(-dist/B)
    return result
end
```

Code 4.5: Calculation of `dU` from Eq. 2.3. The `properties` defined in Code. 4.2 can be extracted using `abmproperties`. The methods `euclidean_distance`, and `get_direction` are also from `Agents.jl`

```
function acc(pos::SVector{2,Float64}, vel::SVector{2,Float64}, agent, model)
    λ = abmproperties(model)[:λ]
    val = λ.*(agent.u_i - vel) -
          sum(
            dU(pos,i.pos,model) for i in allagents(model)
            if i.id != agent.id
            )
    return val
end
```

Code 4.6: Implementation of Eq. 4.2. The `allagents` function returns an iterator over all pedestrians in the model

The numerical schemes can be a combination of different schemes for each of these equations. The methods used here are:

- Explicit/Explicit Euler scheme:

$$\begin{aligned} p^{k+1} &= p^k + \delta\mathrm{t}\ a(q^k, p^k) \\ q^{k+1} &= q^k + \delta\mathrm{t}\ p^k \end{aligned} \tag{4.3}$$

- Implicit/Implicit Euler scheme:

$$\begin{aligned} p^{k+1} &= p^k + \delta\mathrm{t}\ a(q^{k+1}, p^{k+1}) \\ q^{k+1} &= q^k + \delta\mathrm{t}\ p^{k+1} \end{aligned} \tag{4.4}$$

  Given that this is an implicit method, i.e. p^{k+1} exists on both the right and left side of the equation, $p^{k+1} = p^k + \delta\mathrm{t}\ a(q^{k+1}, p^{k+1})$ needs to be solved numerically for each time-step $\delta\mathrm{t}$.

- Leapfrog scheme:

$$\begin{aligned} p^{k+1} &= p^k + \frac{\delta\mathrm{t}}{2}\ \left(a(q^k, p^k) + a(q^{k+1}, p^{k+1})\right) \\ q^{k+1} &= q^k + \delta\mathrm{t}\ p^k + \frac{\delta\mathrm{t}^2}{2}\ a(q^k, p^k) \end{aligned}$$

It is to be noted that:

$$a(q^{k+1}, p^{k+1}) = a(q^{k+1}, p^k) + \lambda(p^k - p^{k+1}) \tag{4.5}$$

the leapfrog scheme can be simplified to:

$$\begin{aligned} p^{k+1} &= p^k + \frac{\delta \mathrm{t}}{2 + \delta \mathrm{t}} \left(a(q^k, p^k) + a(q^{k+1}, p^k) \right) \\ q^{k+1} &= q^k + \delta \mathrm{t}\, p^k + \frac{\delta \mathrm{t}^2}{2}\, a(q^k, p^k) \end{aligned} \tag{4.6}$$

To demonstrate how these numerical methods are defined in code, we can see the implementation of leapfrog scheme in Code. 4.7. It can be noted that the explicit Euler can be defined similarly because both of these methods (Eq. 4.3 and Eq. 4.6) are explicit.

```
function leapfrog_step(agent::Pedestrian, model)
    acc_old = acc(agent.pos, agent.vel, agent, model)
    p = agent.pos + model.dt.*agent.vel + ((model.dt^2)/2).*acc_old
    p = normalize_position(p, model)
    acc_new = acc(p, agent.vel, agent, model)
    v = agent.vel + ((model.dt)/(2 + model.λ*model.dt)).*(acc_old + acc_new)
    return v, p
end
```

Code 4.7: The leapfrog scheme Eq. 4.6 in code

Using ODE solvers provided by `DifferentialEquations.jl`

To define implicit Euler method Eq. 4.4, we make use of the package solvers. As per the directions of `DifferentialEquations.jl`, we first construct our `ODEProblem` based on the dynamics provided in Eq. 4.1

```
function ph_model!(du,u,p,t)
    agent, model = p
    vel, pos = u[1:2], u[3:4]
    du[1:2] = acc(agent.pos, agent.vel, agent, model)
    du[3:4] = vel
end
```

Code 4.8: Defining Eq. 4.1 as an `ODEProblem`

Next we define a solver function that integrates the ODE. Here, we provide the initial conditions `u0` as the pedestrians velocities and positions, and an integrator function that generates a solution for a single time-step `dt` using `step!`. This function can now be used as an argument for `initialize()` in place of `num_solver` in Code. 4.2.

```julia
function ode_step(agent, model)
    u0 = [agent.vel[1], agent.vel[2], agent.pos[1], agent.pos[2]]
    tspan = (0.0, Inf)
    p = (agent, model)
    prob = ODEProblem{true}(ph_model!, u0, tspan, p)

    # Use the model's integrator
    integrator = init(prob, ImplicitEuler(), dt=model.dt)

    # Solve and update agent state
    step!(integrator, model.dt)
    vel = integrator.u[1:2]
    pos = integrator.u[3:4]
    return vel, SVector{2}(pos)
end
```

Code 4.9: ODE solver function to solve the model defined in Code. 4.8 as `ODEProblem`. Here, we have chosen the solver as `ImplicitEuler`

Similar to `ODEProblem` in Code. 4.9, one can also go a step further and define and solve a `DynamicalODEProblem` or `SDEProblem` using `DifferentialEquations.jl`. Since we are also interested in symplectic methods such as the `leapfrog`, it would also be important to be able to construct a solver for our dynamic problem as well. Here, in contrast to our previous approach, the only difference is that we define the equations for $\dot{p}$ and $\dot{q}$ separately, as shown in Code. 4.10

```julia
function ph_p!(dv,v,u,p,t)
    agent, model = p
    vel = v[1:2]
    pos = u[1:2]
    dv[1:2] = acc(agent.pos, agent.vel, agent, model)
end
function ph_q!(du,v,u,p,t)
    agent, model = p
    vel = v[1:2]
    du[1:2] = vel
end
```

Code 4.10: Defining the `DynamicalODEProblem` using separate functions for $\dot{p}$ and $\dot{q}$

Furthermore, we explicitly separate the initial conditions for velocities and positions, and define the problem using `DynamicalODEProblem`. We use `VerletLeapfrog` as our solver as demonstrated in Code. 4.11.

```julia
function dynamic_ode_step(agent, model)
    v0 = [agent.vel[1], agent.vel[2]]
    u0 = [agent.pos[1], agent.pos[2]]
    tspan = (0.0, Inf)
    p = (agent, model)
    prob = DynamicalODEProblem(ph_p!,ph_q!,v0,u0, tspan, p)

    # Use the model's integrator
    integrator = init(prob, VerletLeapfrog(), dt=model.dt)

    # Solve and update agent state
    step!(integrator, model.dt)
    vel = integrator.u[1:2]
    pos = integrator.u[3:4]
    return vel, SVector{2}(pos)
end
```

Code 4.11: Solving the model using `VerletLeapfrog`

This approach of using `DifferentialEquations.jl` to define and solve our model allows more flexibility to choose from a wider range of sophisticated solvers provided by the package. However, one concerning issue is that this solver uses leapfrog discretization in its original form that will still discretize the model as an implicit scheme, whereas previously, we used the convenient relationship provided by the formulation of the acceleration Eq. 4.5 to manipulate the original discretization to form an explicit leapfrog Eq. 4.6, which proved to be more beneficial as it provides the same results with less computational load.

To solve for the stochastic port-Hamiltonian case, we can make use of an SDE solver. We will first have to define our model as an `SDEProblem`. To do this, we will define the drift term and diffusion term separately as shown in Code. 4.12. This reflects the standard SDE form as described in Eq. 2.5.

```julia
function ph_stoc_drift!(du,u,p,t)
    agent, model = p
    vel, pos = u[1:2], u[3:4]
    du[1:2] = acc(agent.pos, agent.vel, agent, model)
    du[3:4] = vel
end
function ph_stoc_diffu!(du,u,p,t)
    agent, model = p
    vel, pos = u[1:2], u[3:4]
    σ = model.sigma
    du[1] = σ
    du[2] = σ
    du[3:4] = [0.0,0.0]
end
```

Code 4.12: Defining the `SDEProblem`

And finally, we construct our solver to be used as another stepping function for our simulation. It can be seen how the stochastic component is added as a Wiener Process. The solver used for most simulations in the next section is the Euler-Heun, however other solvers are also provided the package such as the `SImplicitMidpoint` [28]. This provides room for reassessment on the choice for the solver to be used, specially in a case such as ours that demands symplectic solvers [32].

We induce ach element in the vector with a different random number. We also retrieve the randomly generated values (`wiener_increment`) to be used for analysis and calculation of the stochastic derivative $\mathrm{d}H$.

```
function stochastic_ode_step(agent, model)
    u0 = [agent.vel[1], agent.vel[2], agent.pos[1], agent.pos[2]]
    tspan = (0.0, Inf)
    p = (agent, model)
    W = WienerProcess(0.0, zeros(2), zeros(2))
    prob = SDEProblem(ph_stoc_drift!,ph_stoc_diffu!,u0, tspan, p,
                      noise=W, noise_rate_prototype=zeros(4, 2))

    # Use the model's integrator
    integrator = init(prob, EulerHeun(), dt=model.dt)

    # Solve and update agent state
    step!(integrator)
    vel = integrator.u[1:2]
    pos = integrator.u[3:4]
    wiener_increment = integrator.W.dW
    return vel, SVector{2}(pos), wiener_increment
end
```

Code 4.13: Solving the SDE using Euler-Heun

To summarize, in this section we've devised mainly four solvers: for the deterministic port-Hamiltonian model, we have explicit Euler, implicit Euler, and leapfrog; for the stochastic port-Hamiltonian model, we have implicit Euler-Heun method. We have allowed ourselves further flexibility by introducing packaged solvers. For the upcoming simulations in the remainder of this section, we will be using Code. 4.9 for the implicit Euler as well as explicit Euler by changing the solver to `Euler`; and Code. 4.7 for the leapfrog. Next, we will formulate functions to calculate the Hamiltonian in our model, and then compare the results obtained by each solver.

4.3 Calculation of Hamiltonian

To compare these numerical schemes, we can of course observe how the agents interact over time and whether they reach a lane or stripe formation based on the provided desired velocity $\mathtt{u_i}$. But to have a quantitative measure, we also would like to calculate the Hamiltonian Eq. 3.7 and observe the change in Hamiltonian Eq. 3.15 over time. For this reason we included `:Hamiltonian` and `:dH` as key-value pairs in the `properties`

dictionary from Code. 4.1 and define functions Code. 4.14 and Code. 4.15 to calculate them.

```
function calc_hamiltonian(model)
    kinetic_energy = 0.5.*sum(
        [transpose(a.vel)*a.vel for a in allagents(model)]
        )
    potential_energy = 0.5.*sum(
        [sum_potU(a,allagents(model),model) for a in allagents(model)]
        )
    return kinetic_energy + potential_energy
end
```

Code 4.14: Calculation of Hamiltonian $H(z(t))$ as defined in Eq. 3.7. For clarity, the kinetic and potential terms are calculated separately. One can easily observe that the function `sum_potU` evaluates the double summation.

```
function calc_dH(model)
    return sum(
        [model.λ*norm(i.vel)*norm(i.u_i - i.vel) for i in allagents(model)]
        )
end
```

Code 4.15: Calculation of $\frac{d}{dt}H(z(t))$ as defined in Eq. 3.15

Inner summation of the second term in Eq. 3.7 is defined in the function `sum_potU`. This simply adds all the individual potential energies between pairs of all combinations of the pedestrians in the model. The potential energy Code. 4.16 is calculated following the definition provided in Eq. 2.2

```
function potU(a1::Pedestrian,a2::Pedestrian, model)
    A = model.A
    B = model.B
    dist = euclidean_distance(a1, a2, model)
    return A*B*exp(-dist/B)
end

function sum_potU(a1::Pedestrian, agents, model)
    return sum([i.id != a1.id ? potU(a1, i, model) : 0 for i in agents])
end
```

Code 4.16: Evaluating the inner summation of potential energies from Eq. 3.7 using `sum_potU` in the calculation of the Hamiltonian in Code. 4.14

Stochastic Hamiltonian Calculation

Here, we calculate the stochastic derivative of the Hamiltonian, i.e. dH Eq. 3.17 for the parameter `:stoch_dH` in Code. 4.1. The drift part includes the deterministic derivative of H from `calc_dH` Code. 4.15 and calculates the trace of the Hessian of the Hamiltonian `calc_trace_ddH`. Whereas, the diffusion part is calculated separately. Finally, both values are summed to evaluate dH. It can be seen that the function also takes `random_list` as an argument, which is the list of randomly generated values retrieved from the SDE solver Code. 4.13.

```
function calc_stoch_dH(model, random_list)
    if length(random_list) == 0
        return 0
    else
        dW = random_list
    end
    σ = model.sigma
    drift_H = calc_dH(model) + calc_trace_ddH(model)
    diffu_H = σ * sum(transpose(i.vel) * dW[i.id] for i in allagents(model))
    return (drift_H + diffu_H)
end
```

Code 4.17: Stochastic derivative dH for the Hamiltonian as defined in Eq. 3.17

The trace of the Hamiltonian Eq. 3.16 from Code. 4.17, consists of the summation of the number of pedestrians N and the trace of Hessian of the potential function, i.e $\mathrm{Tr}\{\nabla^2 U(x)\}$ calculated for all pedestrian combination $i \neq j$.

```
function calc_trace_ddH(model)
    total_ddU = 0
    for i in allagents(model)
        for j in allagents(model)
            if i.id != j.id
                total_ddU += ddU(i,j,model)
            end
        end
    end
    σ = model.sigma
    return (σ^2/4)*total_ddU + (σ^2/2)*nagents(model)
end
```

Code 4.18: Trace of the Hessian of the Hamiltonian H, from Eq. 3.16

The Hessian of the potential function, i.e. $\nabla^2 U(x)$, as defined in Eq. 2.4 is first computed in Code. 4.19 and its trace is returned, which is then used in $\mathrm{Tr}\{\nabla^2 H\}$ to be evaluated for the stochastic derivative of the Hamiltonian dH.

```
function ddU(a1, a2, model)
    x = get_direction(a1.pos,a2.pos,model) # x
    dist = euclidean_distance(a1,a2,model)
    val_ddU = - model.A*
        (
            (1/dist)*(I - (x*transpose(x)/dist^2)) -
            (1/(model.B*dist^2))*(x*transpose(x)/dist^2)
        ) * exp(-dist/model.B)
    return tr(val_ddU)
end
```

Code 4.19: Hessian of U, Eq. 2.4, and then taking its trace as required in the stochastic derivative of the Hamiltonian

These functions are used to evaluate dH for the stochastic case, and study the plots that result from the simulation with differing values of the volatility σ. It can be observed that the equations of dH resolve to the deterministic derivative of the Hamiltonian when $\sigma = 0$. The inclusion of volatility σ not only introduces the diffusion part, but also includes extra terms in the drift part of the derivative of the Hamiltonian. In contrast to the deterministic Hamiltonian which was only a function of the velocities of the pedestrians, the stochastic derivative is a function of the velocities as well as the relative positions Q_{ij}.

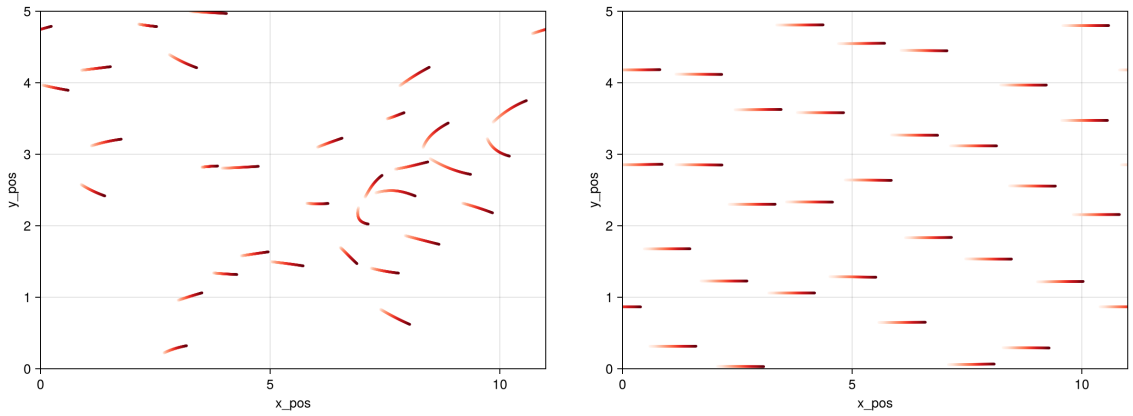
4.4 Comparison of Numerical Schemes

We can now compare the numerical schemes by determining how accurately they are describing the dynamics. This is done by evaluating the error, which is defined as the difference between the time derivative of the Hamiltonian at time-step k and the difference between the Hamiltonian at time step k and $k-1$, resulting in the following expression:

$$\begin{aligned} \text{Error} &= \frac{d}{dt}H(z^k) - \frac{1}{\delta \text{t}}\left(H(z^k)) - H(z^{k-1})\right) \\ \text{Error} &= \lambda\langle p^k, u - p^k\rangle - \frac{1}{\delta \text{t}}\left(H(Q^k, p^k)) - H(Q^{k-1}, p^{k-1})\right) \end{aligned} \tag{4.7}$$

The first term of Eq. 4.7 is the time derivative of the Hamiltonian $\frac{d}{dt}H(z(t))$, the expression of which was already obtained in Eq. 3.15. Since the second term is the approximation of the time derivative, it can be noted that the Error $\to 0$ as $\delta \text{t} \to 0$.

We concentrate once again to the deterministic port-Hamiltonian case, and run our simulations to compare the following numerical schemes: explicit Euler, implicit Euler, and Leapfrog. Hence, we initialize 3 models each using one of the mentioned solvers. The intent here is to replicate the results obtained from [11], hence we use the same model parameters as described in Code. 4.1 and Code. 4.3, namely $u_i = [1, 0]$ for all pedestrians i. The model is run for 20 seconds of simulation time, for each step-size δt ranging from 0.01 to 0.2.



(a) Unidirectional flow [start] (b) Unidirectional flow [steady-state]

Figure 4.1: The progress of the simulation at (a) the start of the simulation and (b) after some time, reaching a steady state

It can be observed from Fig. 4.1 how the pedestrians start from a random arrangement and gradually reach a steady state by achieving their desired unidirectional motion.

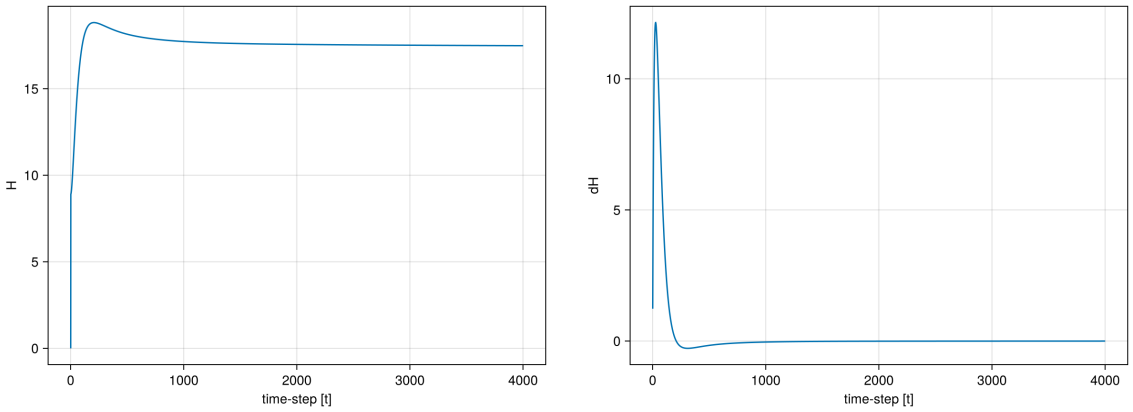
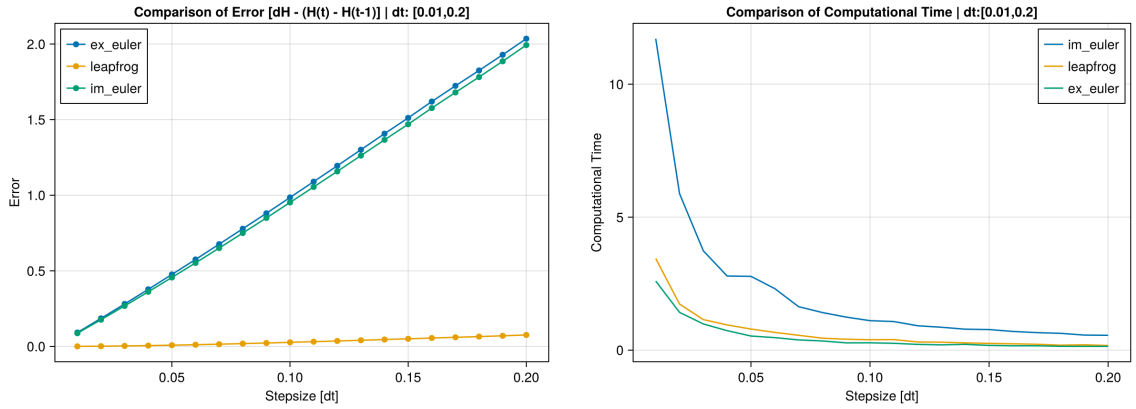


Figure 4.2: $H(z(t))$ and $\frac{d}{dt}H$ as the simulation progresses over time.

As demonstrated in Fig. 4.2 the change in Hamiltonian with respect to time, $\frac{d}{dt}H(z(t))$ tends to zero as the system reaches its steady state. Using this solution of the unidirectional flow, we have generated a plot for the errors obtained from each numerical scheme in Fig. 4.3. The plot Fig. 4.3a shows the mean errors obtained for solving the model using a numerical scheme with fixed time-steps δt, whereas Fig. 4.3b illustrates the computation time required to solve the model for fixed time-steps δt.



(a) Mean error of the schemes

(b) Computation time of the schemes

Figure 4.3: Comparison of numerical schemes in terms error and computation time

As one would expect the errors do increase when the δt increases, however the stark contrast between the minute numerical errors of leapfrog scheme to others clearly shows the benefit of using symplectic schemes for solving Hamiltonian systems. The computation time of leapfrog here is similar to that of explicit Euler. Implicit Euler, on the other hand, yields results that are only slightly better than its explicit counterpart regarding errors. Due to its implicit nature, the implicit Euler also requires extra steps to numerically solve equations resulting it to be the most computationally expensive compared to the other two.

With these results in mind, we conclude this section with leapfrog as our choice to be used for all deterministic dynamical models for our simulations in the coming section.

5 Simulation and Observations

In this section we make use of our model by simulating it for various cases. First we will demonstrate some basic dynamics based on the impact of dissipation λ, desired velocities u_i, and interaction A. Next we will focus on collective behavior, and present self-organization such as lane and stripe formations for counter and cross flows respectively. Later we will repeat the same experiments with addition of stochastic elements. Some insights are provided on the relationship between the parameters, Hamiltonian, and the collective behavior.

The model parameters defined for the simulation are the same as defined in Code. 4.1. Any changes made would be mentioned within their respective cases. For reproducibility and comparison, each simulation starts with the same arrangement of pedestrians on the xy-plane as shown below.

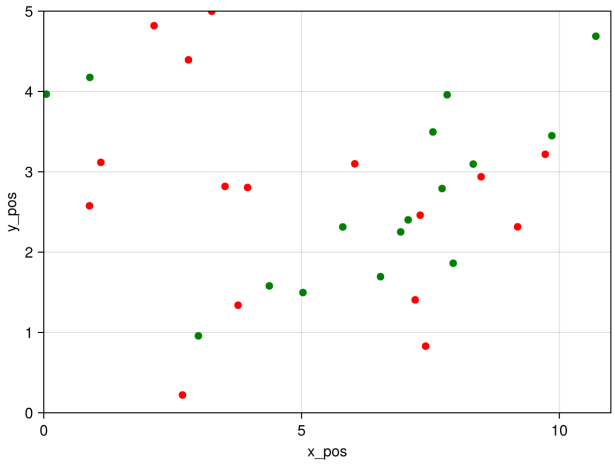


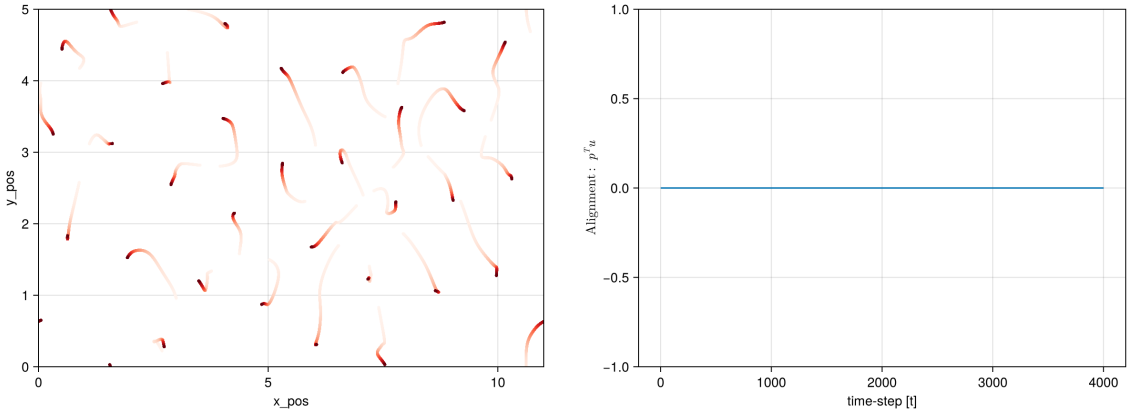
Figure 5.1: Starting positions of pedestrians.

The colors in Fig. 5.1 distinguishes groups of pedestrians each having same desired speeds. This is only relevant for flows that are multidirectional, as will be shown in Section. 5.2. In the cases of unidirectional flow or where the direction is irrelevant, the color for all is the same.

5.1 Basic Dynamics

This section serves to illustrate how the changes in the parameters affect the dynamics of the model, although none of the dynamics presented in the model are realistic, the diagrams provide a way to think about the dynamics of the model based on the changes in the parameter.

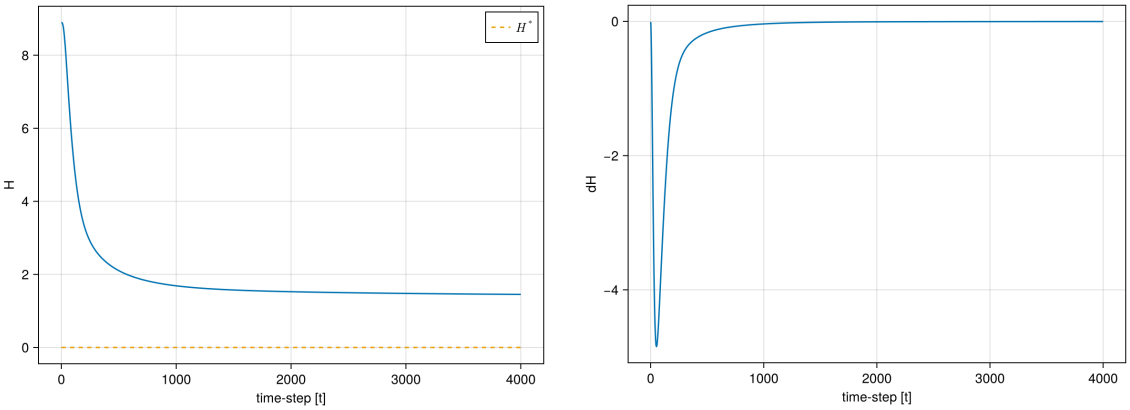
- **No desired velocity ($u_i = \begin{bmatrix} 0 & 0 \end{bmatrix}$) with dissipation ($\lambda > 0$):**



(a) Pedestrian trajectories (b) Alignment

Figure 5.2: Crystallization of pedestrians due to no input $u_i = 0$ and allowing the system to dissipate energy

Allowing the system to dissipate $\lambda > 0$ while simultaneously the pedestrians have no desired velocity Fig. 5.2, essentially creates a system with an active output port, but no input. The results are as expected, the system keeps losing energy leading the pedestrians to crystallize over time. The alignment here is zero, because the desired velocity $u_i = 0$

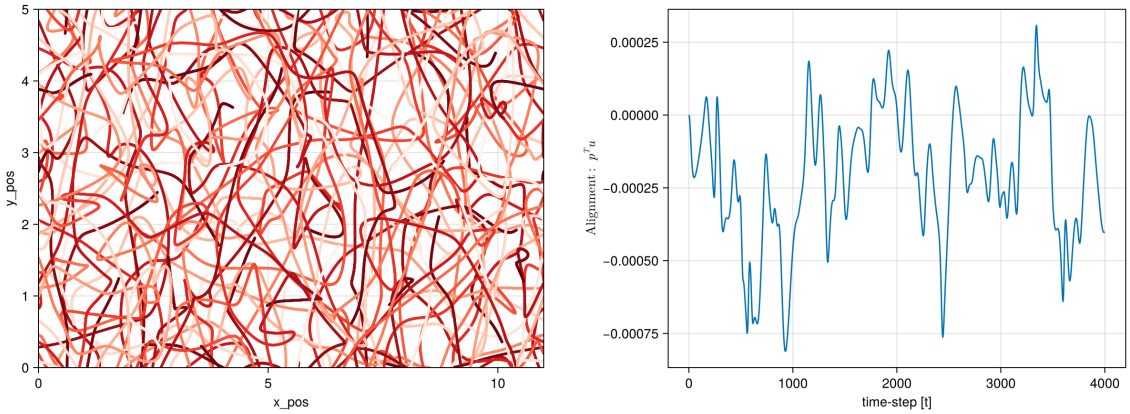


(a) Hamiltonian H over time (b) $\frac{\mathrm{d}}{\mathrm{d}t}H$ over time

Figure 5.3: Hamiltonian for crystallization with $u_i = 0$ with dissipation

The Hamiltonian, which is a representation of the total energy of the system relaxes towards lower energy levels, as the system only loses energy without any gain. Simultaneously, the derivative of the Hamiltonian $\frac{\mathrm{d}}{\mathrm{d}t}H$ reaches 0 as the Hamiltonian becomes constant.

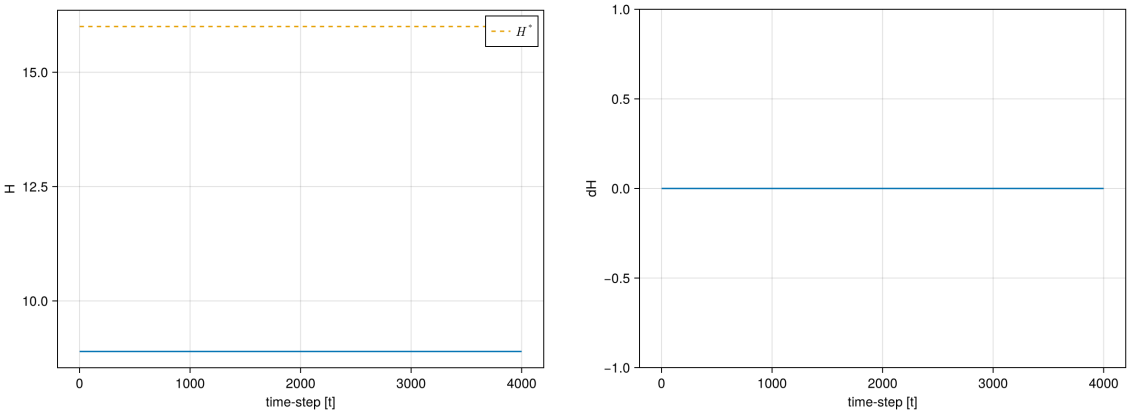
- **No dissipation** ($\lambda = 0$)



(a) Pedestrian trajectories (b) Alignment

Figure 5.4: Trajectories of pedestrians in a non-dissipative system ($\lambda = 0$)

Given the dynamics formulated in Eq. 3.3, it can be clearly seen that λ exists both as a coefficient of the input port $\tilde{u}$ as well as in the output port y. Thus setting $\lambda = 0$ clearly suggests that the resulting system has no dissipation as well as no energy feed. Consequently, the pedestrians move with no direction Fig. 5.4a regardless of the fact that the desired velocity is provided. The disordered directions of all agents is clearly quantified by the alignment measure as the values remain near 0, suggesting the mean direction of all pedestrians is not at all aligned to their desired velocity directions.

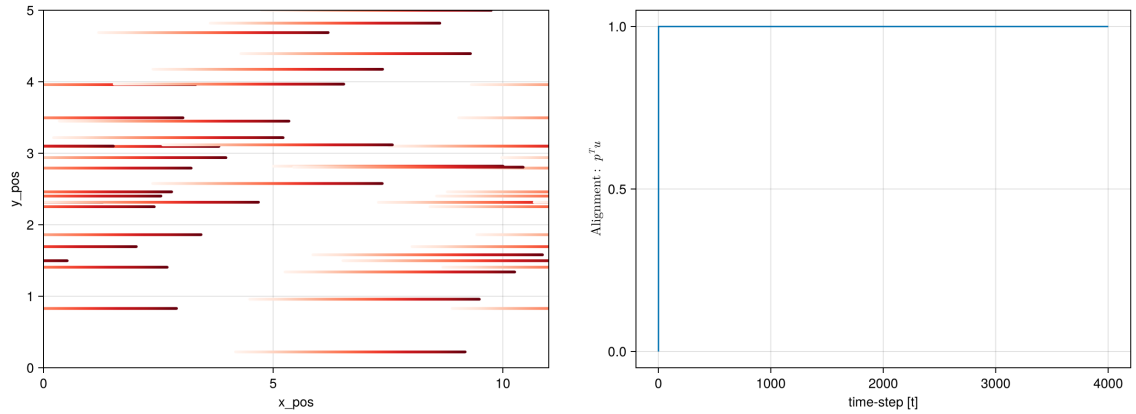


(a) Hamiltonian H over time (b) $\frac{\mathrm{d}}{\mathrm{d}t}H$ over time

Figure 5.5: Hamiltonian for crystallization with $u_i = 0$ with dissipation

The resulting system is a conservative Hamiltonian system Eq. 3.2, as by consequence the dissipative term R also vanishes. This can be clearly observed as the Hamiltonian remains constant over time.

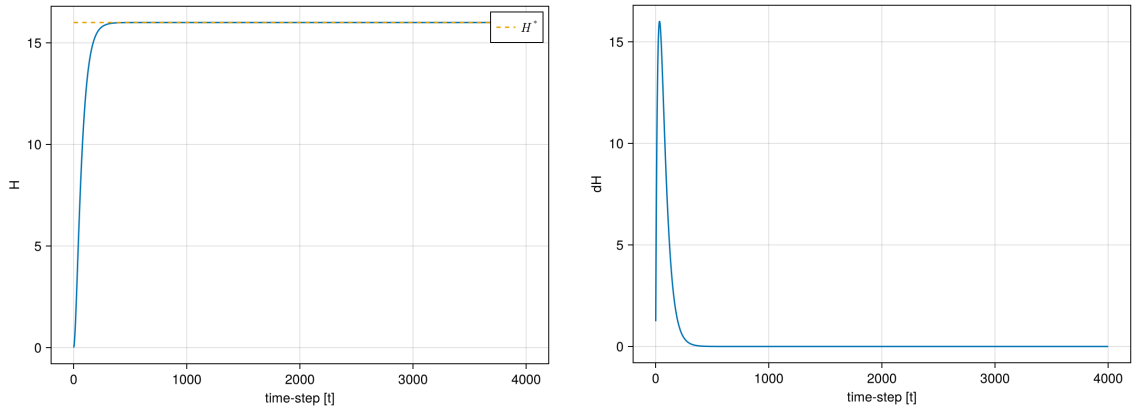
- **No pedestrian interaction ($A = 0$) with $u_i = \begin{bmatrix} 1 & 0 \end{bmatrix}$**



(a) Pedestrian trajectories (b) Alignment

Figure 5.6: Trajectories of pedestrians with no interaction with other pedestrian ($A = 0$) and desired velocities $u_i = \begin{bmatrix} 1 & 0 \end{bmatrix}$

By setting $A = 0$, the pedestrians in the system move with no interactivity with other pedestrians in the system. Essentially, every pedestrian simply moves with their desired velocities as soon as the simulation begins Fig. 5.6. Because this parameter directly effects the interaction potential $U(x)$ in Eq. 2.2, it can be clearly observed that the motion of the pedestrian is independent of their distance to each other, in other words they move with no concern to how close or far they are from other pedestrians. The alignment plot clearly quantifies this behavior as the plot linearly jumps from 0 to 1, indicating that that all pedestrians accelerate toward their desired directions without disruptions as soon as the simulation begins.



(a) Hamiltonian H over time (b) $\frac{\mathrm{d}}{\mathrm{d}t}H$ over time

Figure 5.7: Hamiltonian over time with no interaction ($A = 0$) and desired velocities $u_i = \begin{bmatrix} 1 & 0 \end{bmatrix}$

Given the lack of interaction, the pedestrians move with their desired velocities,

thus the Hamiltonian need not be dependent on time, but instead on the desired velocities of the pedestrians – mathematically $p_i = u_i$. One can also obtain this long term value of the Hamiltonian $H^*(u)$ by setting $A = 0$ (as it is for this case) in Eq. 3.7 leading to

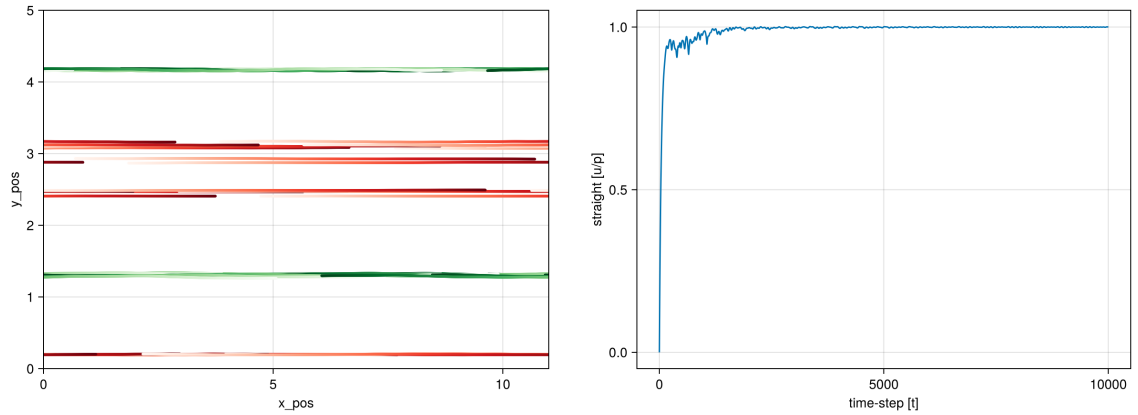
$$H(z(t)) = \frac{1}{2}||p(t)||^2 \rightarrow H^*(u) = \frac{1}{2}||u||^2 \tag{5.1}$$

This expression for the long term Hamiltonian $H^*(u)$ is a function of the desired velocities u_i. As u_i is set by the user, one easily evaluate this value. Even with potential interaction allowed in the system dynamics, the expression serves as a representation of what the system as a whole is trying to attain. Ideally, every pedestrian wants to achieve the desired velocity u_i, and the system as a whole wants to achieve the energy H^*.

5.2 Collective Dynamics

In this section, we will witness collective phenomenon based on the desired speeds and relaxation (or dissipation) parameter λ. The pedestrians will start from the same random positions generated in Fig. 5.1, but with different desired speeds u_i. This disordered arrangement goes through a phase transition and form ordered patterns, such as lane formation and stripe formation. We will notice how a sufficient λ is needed to achieve these phase transitions. Furthermore, we will see how $H^*(u)$ from Eq. 5.1 ties in as an order parameter to identify collective behaviors.

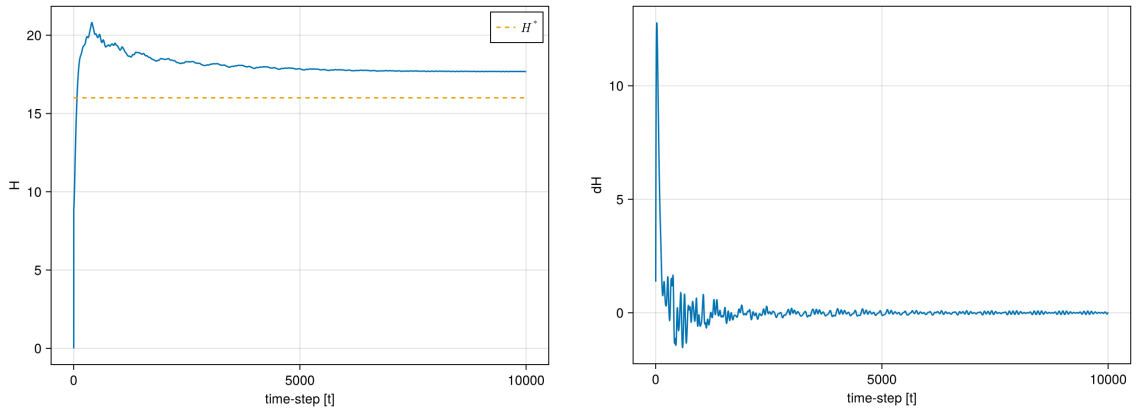
- **Lane Formation for counter flow with** $\lambda = 2$



(a) Pedestrian trajectories (b) Alignment

Figure 5.8: Lane formation for counter flow

Starting from a random arrangement of pedestrians as shown in Fig. 5.1, with predefined desired velocities $u_i = \begin{bmatrix} 1 & 0 \end{bmatrix}$ for half of the pedestrians (Red), and $u_i = \begin{bmatrix} -1 & 0 \end{bmatrix}$ for the other half (Green), as shown in Code. 4.3. As the time progresses, the pedestrians form lanes. The alignment clearly illustrates transition from disorder to order, as the pedestrians align themselves to their respective desired velocities over time.

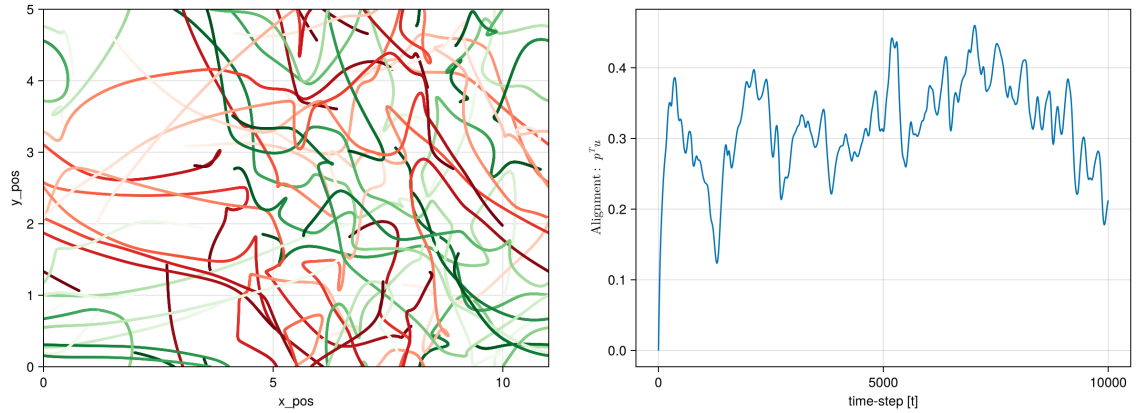


(a) Hamiltonian H over time (b) $\frac{d}{dt}H$ over time

Figure 5.9: Hamiltonian for counter flow with lane formation

From the point of view of energy in the system, the pedestrians self-organize themselves in a manner such that the Hamiltonian reaches a steady state and remains above H^*, while $\frac{d}{dt}H$ converges to zero.

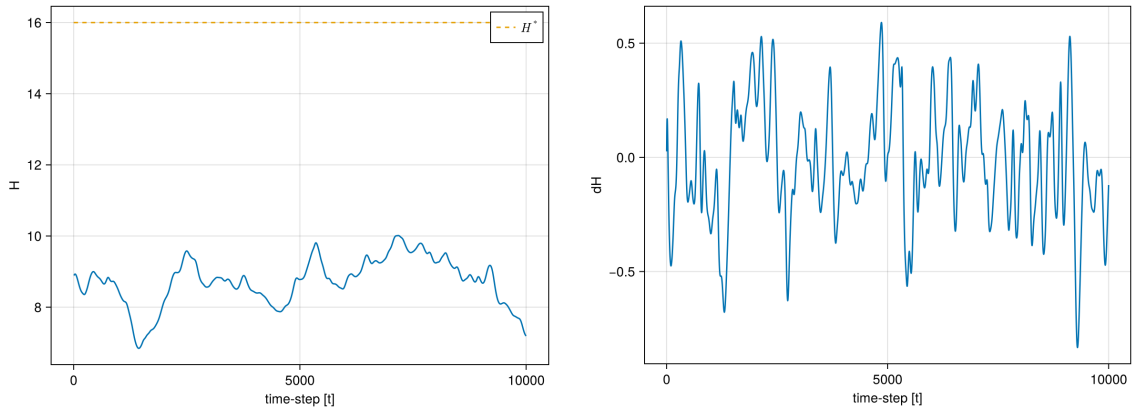
- **Gridlock for counter flow with** $\lambda = 0.2$



(a) Pedestrian trajectories (b) Alignment

Figure 5.10: Gridlock for counter flow

With the same parameters as the previous case, with the only difference being that λ is not sufficiently high, it can be observed that no lane formation occurs. The system neither gains nor dissipates enough energy, resulting in pedestrians behaving similarly to the trajectories presented in Fig. 5.4, leading to gridlocks. Since the pedestrians fail to reach their desired directions, the alignment stays near zero indicating disorder in the system, as it can be clearly witnessed from the trajectories.



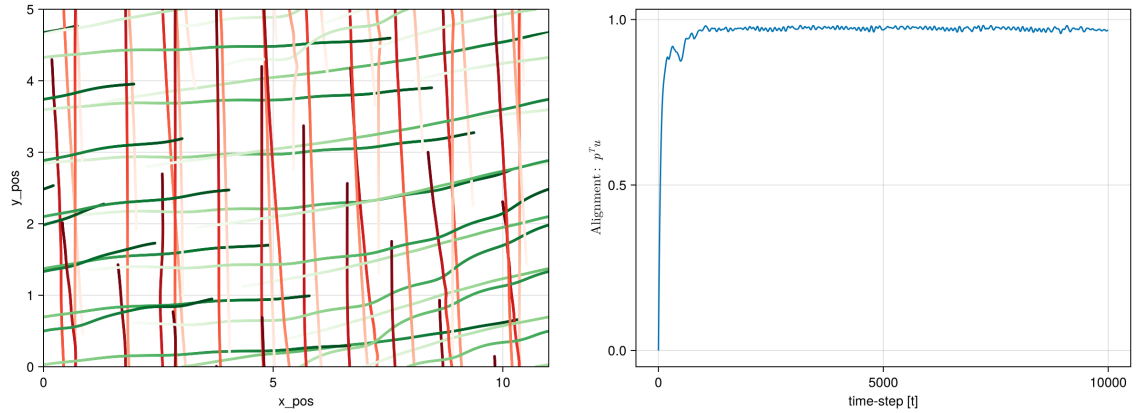
(a) Hamiltonian H over time

(b) $\frac{d}{dt}H$ over time

Figure 5.11: Hamiltonian for counter flow with gridlock

The Hamiltonian erratically fluctuates over time and remains below H^*. Here the significance of λ is apparent, as this parameter balances the conservation of energy enforced by the skew-symmetry with the flow of energy through dissipation and feed permitted by the ports.

- **Stripe Formation for cross flow with $\lambda = 2$**

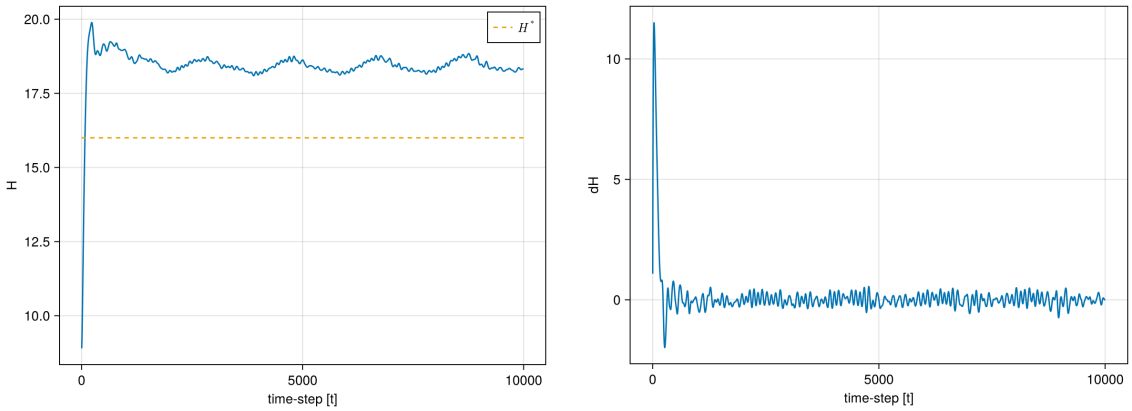


(a) Pedestrian trajectories

(b) Alignment

Figure 5.12: Stripe formation at a diagonal for cross flow with $\lambda = 2$

Starting from the same positions as Fig. 5.1 with the desired velocities set to $u_i = \begin{bmatrix} 1 & 0 \end{bmatrix}$ for half of the pedestrians (Green), and $u_i = \begin{bmatrix} 0 & 1 \end{bmatrix}$ for the other half (Red). We can observe that the pedestrians form a *diagonal* stripe formation; they are not able to reach the desired direction with much ease, forming angled stripes. Interestingly, alignment remains near but below 1, clearly indicating that the pedestrians are not quite able to reach their desired velocities even though the overall system is still ordered.



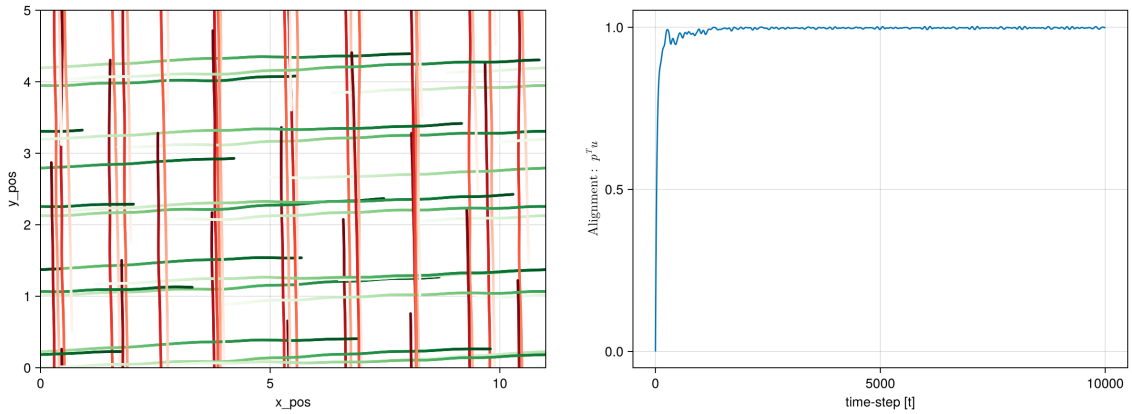
(a) Hamiltonian H over time

(b) $\frac{d}{dt}H$ over time

Figure 5.13: Hamiltonian for cross flow with stripe formation with $\lambda = 2$

This is also apparent when we look at the plots for the Hamiltonian, as both the Hamiltonian periodically fluctuates over time and $\frac{d}{dt}H$ fluctuates about zero. However, the long term dynamics of the pedestrians would result in an arrangement where the pedestrians don't deviate from their desired directions, as shown in the next case.

- **Stripe Formation for cross flow with** $\lambda = 3$

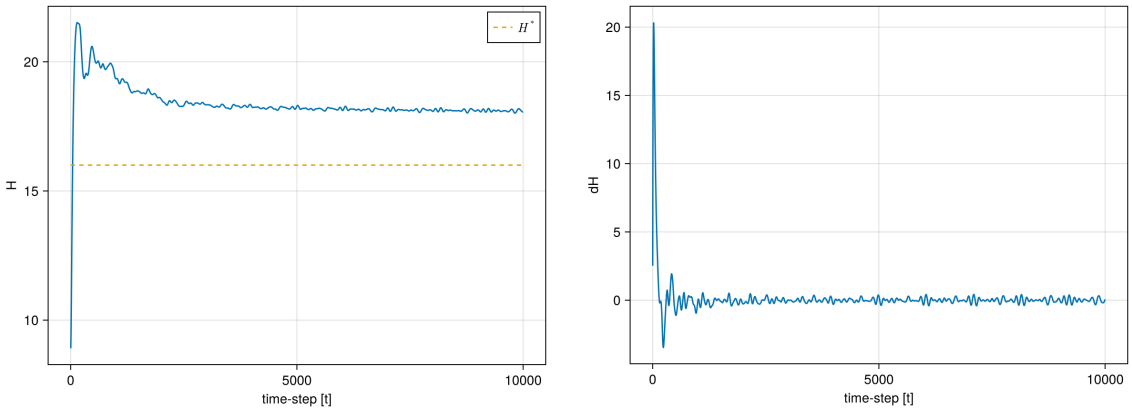


(a) Pedestrian trajectories

(b) Alignment

Figure 5.14: Improved stripe formation for cross flow with $\lambda = 3$

Here we replicate the model from before, though we set λ to a higher value, we can clearly observe a better stripe formation. The alignment also shows an improvement, as the value is closer to 1 than before, indicating that this stripe formation is an improvement from the previous case.



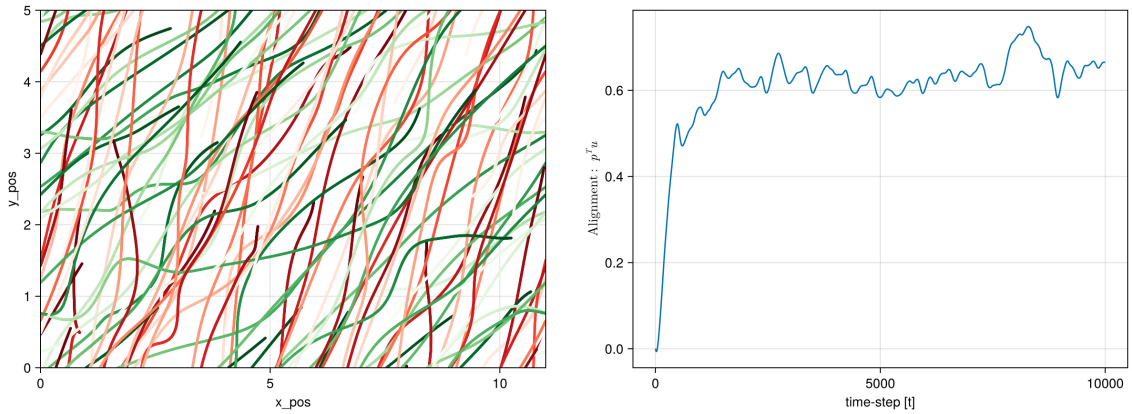
(a) Hamiltonian H over time

(b) $\frac{d}{dt}H$ over time

Figure 5.15: Hamiltonian for cross flow with lane formation with $\lambda = 3$

The Hamiltonian as well as its derivative also don't fluctuate too much as $\frac{d}{dt}H$ decreases more quickly. In both of these cases, we can observe that the stripe formation occurs, and the Hamiltonian remains above H^*.

- **Gridlock for cross flow with** $\lambda = 0.2$

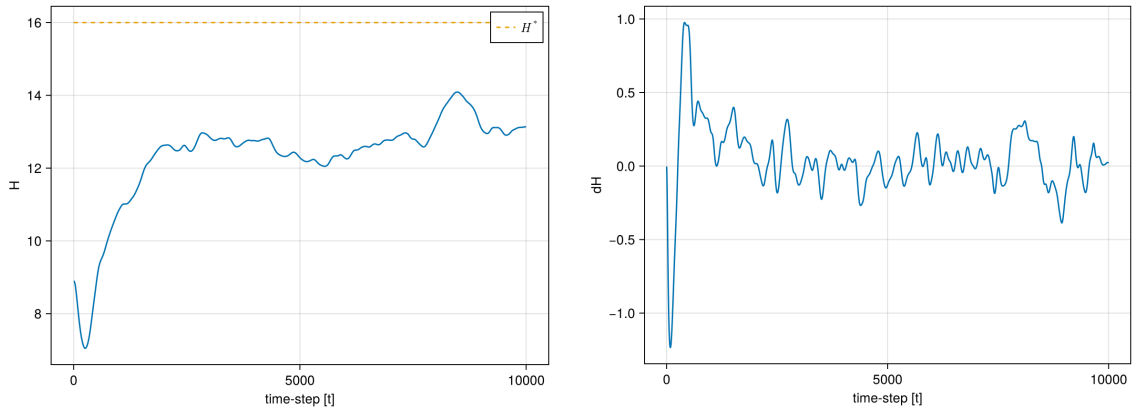


(a) Pedestrian trajectories

(b) Alignment

Figure 5.16: Gridlock for cross flow

Here we can see that no stripe formation occurs, as λ is not high enough for the pedestrians to be reactive. The alignment fluctuates and remains far below 1. The Hamiltonian also remain below H^*



(a) Hamiltonian H over time

(b) $\frac{d}{dt}H$ over time

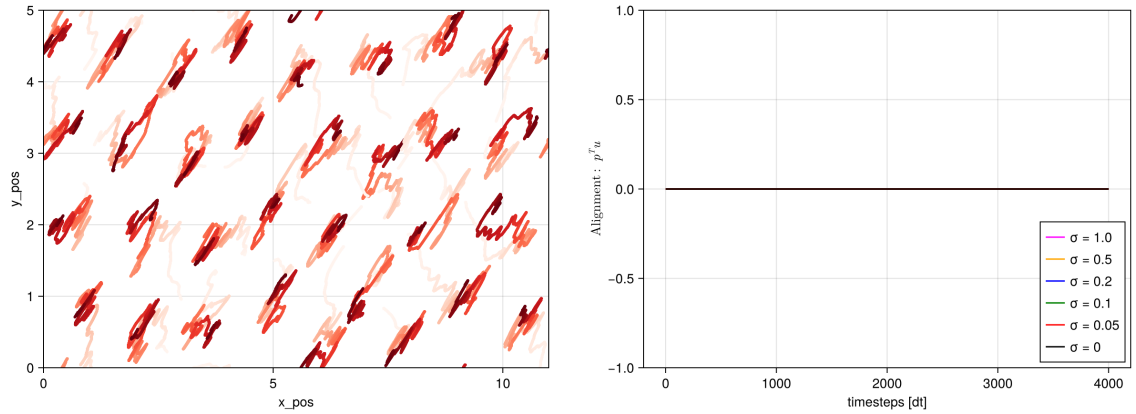
Figure 5.17: Hamiltonian for cross flow with gridlock

From the simulations above, we have observed that collective dynamics occur when the Hamiltonian remains stable and above H^*. We will test this idea further in the next section where we subject our model with stochastic elements.

5.3 Basic Dynamics with Stochastic Elements

In this section, we once again pay our attention to the stochastic model dynamics, and simulate the model with basic dynamics as before with addition of stochastic elements. From a physical sense, one can interpret the addition of noise as 'warming' the system, and hence we will notice, for most cases, that the overall energy of the systems increase with increase in noise. We observe the long term dynamics, the Hamiltonian, and also its stochastic derivative dH over time. We also compare the dynamics of the stochastic (non-deterministic) system $\sigma \geq 0$ with their respective deterministic case $\sigma = 0$.

- **No desired velocity** $u_i = \begin{bmatrix} 0 & 0 \end{bmatrix}$ **with** $\lambda > 0$ **and** $\sigma \geq 0$

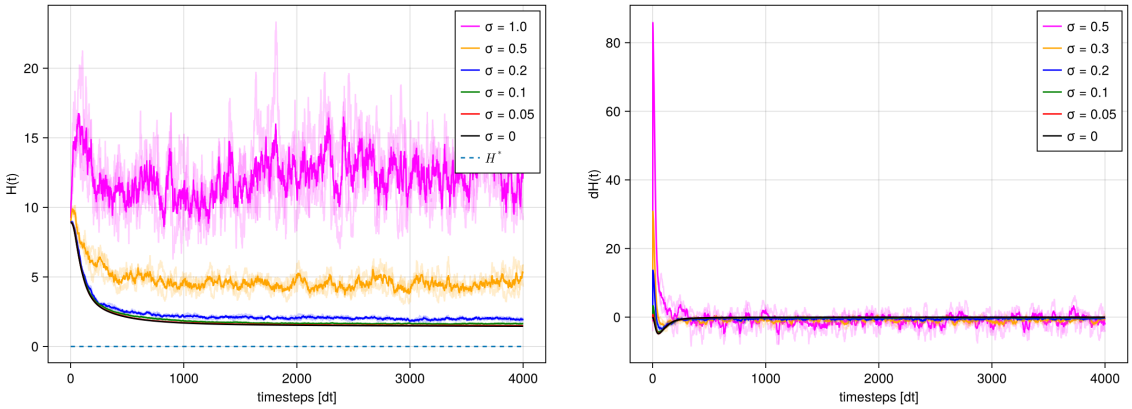


(a) Pedestrian trajectories for $\sigma = 0.2$

(b) Alignment

Figure 5.18: Crystallization with stochastic effects

Starting the same position as before, the pedestrians fluctuate about their positions with no clear direction as $u_i = 0$. Because of the dissipation parameter, the order of magnitude of the velocities with which the pedestrians fluctuate doesn't blow out of proportions, in contrast to the next case. The alignment, of course remain zero, as the pedestrians have no direction to align themselves to.

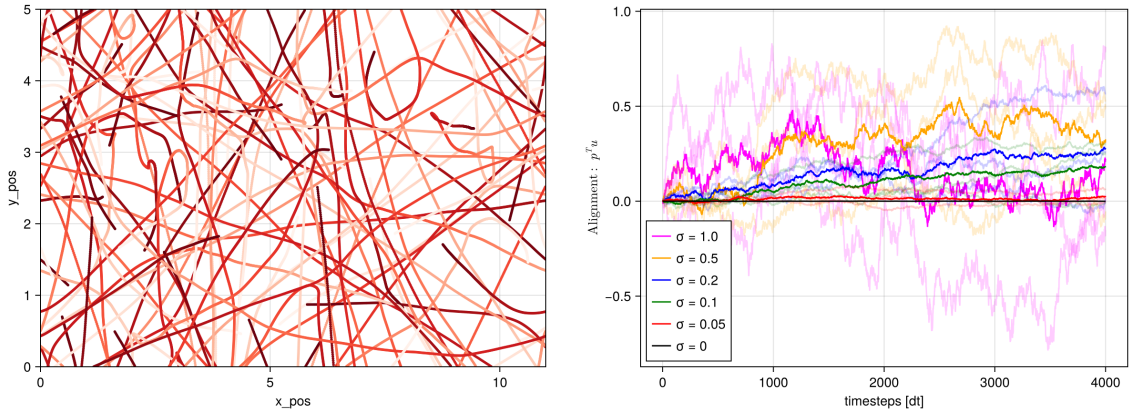


(a) Hamiltonian H over time (b) dH over time

Figure 5.19: Hamiltonian under $u_i = 0$ and $\lambda > 0$ with stochastic effects

It can be clearly seen that the Hamiltonian in all cases $\sigma \geq 0$ remains above H^* and fluctuates about a constant line, indicating that dissipation occurs, and that the system energy does not diverge to higher values.

- **No dissipation** ($\lambda = 0$) **with** $\sigma \geq 0$

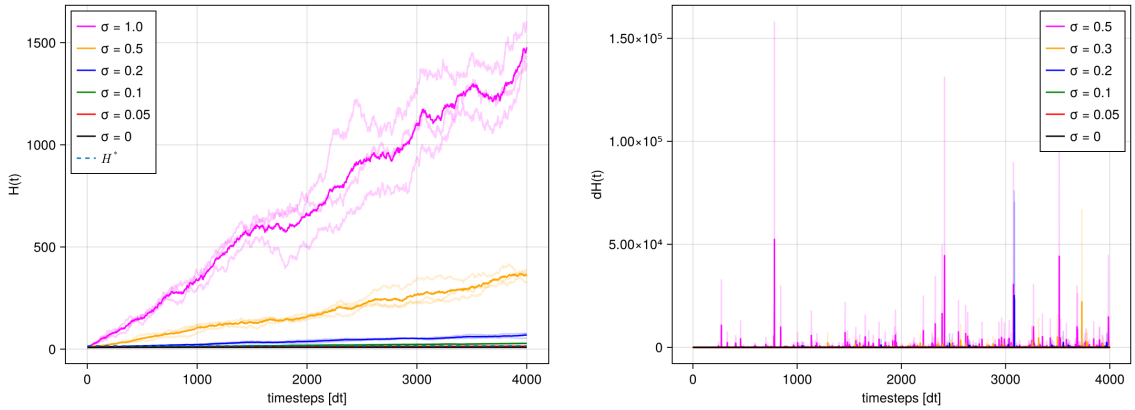


(a) Pedestrian trajectories for $\sigma = 0.2$ (b) Alignment

Figure 5.20: Trajectories of pedestrians in a non-dissipative system with stochastic effects

Here, with no dissipation, we witness the warming effects due to noise. The system gains energy and remains closed, the velocities of the pedestrians will

keep increasing. The system remains disordered as the pedestrians fail to align themselves with their desired velocities, as indicated by the alignment being near 0 and rapidly fluctuating.

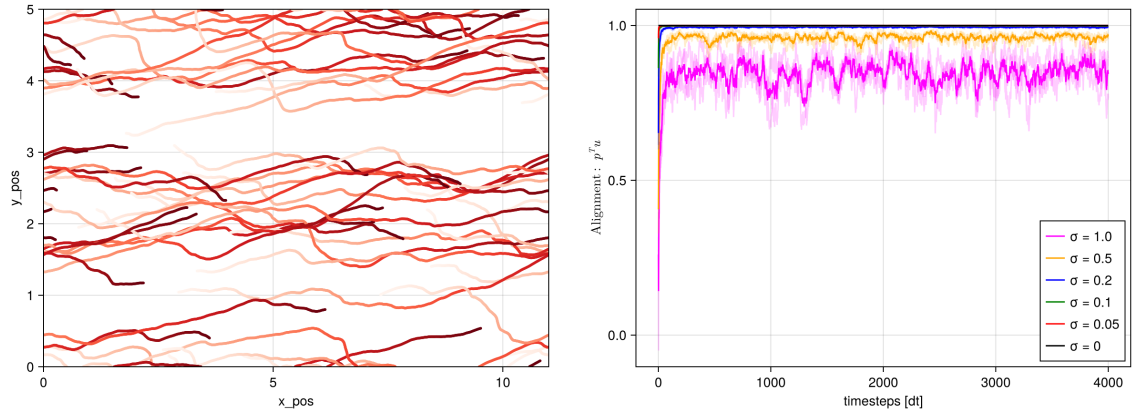


(a) Hamiltonian H over time (b) dH over time

Figure 5.21: Hamiltonian under non-dissipative system with stochastic effects

As the system gains energy without dissipation, the Hamiltonian diverges to higher values.

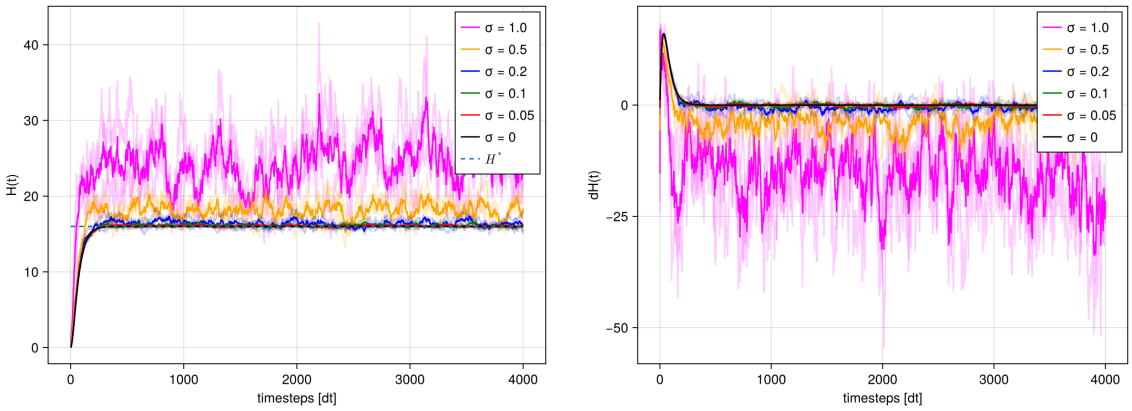
- **No interaction** $(A = 0)$ **with** $u_i = \begin{bmatrix} 1 & 0 \end{bmatrix}$ **and** $\sigma \geq 0$



(a) Pedestrian trajectories for $\sigma = 0.5$ (b) Alignment

Figure 5.22: Pedestrian trajectories with no interaction and stochastic effects

Recall the deterministic counterpart with no interaction with other the pedestrians Fig. 5.6, the pedestrians were able to reach their desired velocities as soon as the simulation starts. However, here, with the addition of noise, we witness a clear transition from ordered to disordered trajectories due to the volatility σ. This is also indicated in the alignment, as the system remains very near 1 for values below $\sigma \leq 0.2$, beyond this value the trajectories, however, are disrupted.



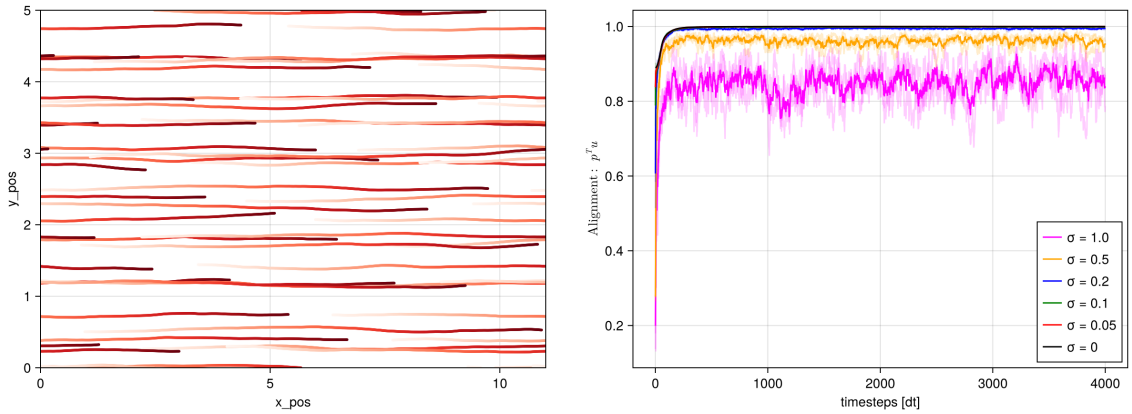
(a) Hamiltonian H over time

(b) dH over time

Figure 5.23: Hamiltonian under $A = 0$ and stochastic effects

The Hamiltonian also indicates orderly system behavior for $\sigma \leq 0.2$, as the plots remains at or above H^* and fluctuate only a little, however for values $\sigma \geq 0.5$, the system devolves into disorder as it can be witnessed by the rapid fluctuations in the Hamiltonian.

- **Unidirectional flow** ($\sigma \geq 0$)

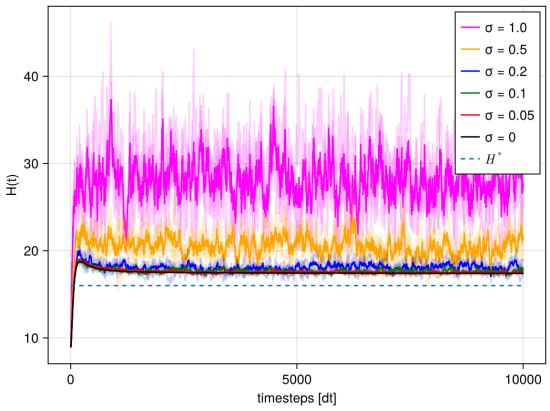


(a) Pedestrian trajectories for $\sigma = 0.05$

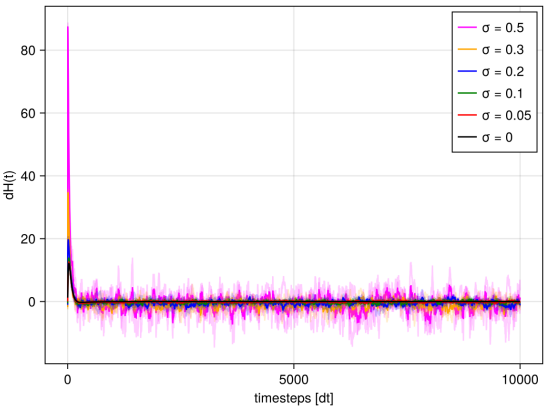
(b) Alignment

Figure 5.24: Unidirectional flow with stochastic effects

We can observe that increasing the volatility σ in the dynamics perturbs the velocity of the pedestrians in the unidirectional case. The effects of the randomness are as expected; the dynamics deviate more when the volatility is high. The alignment indicates that the system dynamics remain ordered for lower values of σ.



(a) Hamiltonian H over time



(b) dH over time

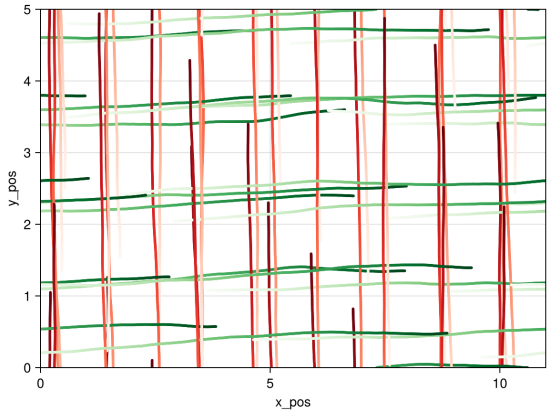
Figure 5.25: Hamiltonian for unidirectional flow with stochastic effects

The Hamiltonian is the least for cases where there is no noise, indicating that the pedestrians are able to reach their desired velocities without perturbations, however with higher volatility the system becomes disordered as indicated via the rapid fluctuations in the Hamiltonian.

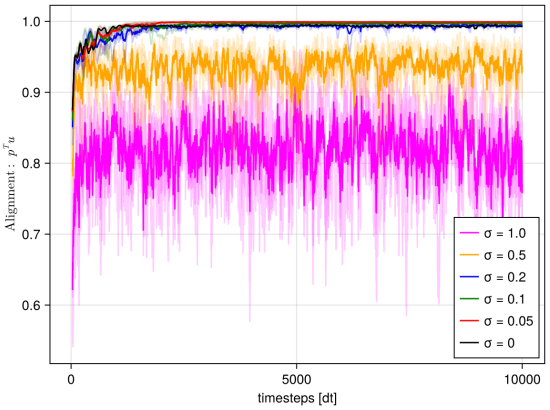
5.4 Collective Dynamics with Stochastic Effects

We will keep the same parameters as Code. 4.1, and focus mainly on counter and cross flows with addition of stochastic effects.

- **Cross flow** ($\sigma \geq 0$)



(a) Pedestrian trajectories for $\sigma = 0.05$

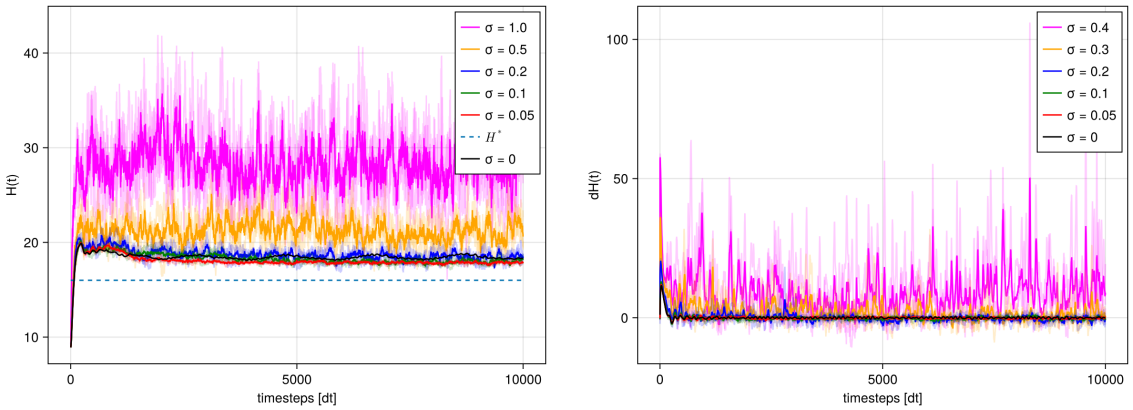


(b) Alignment

Figure 5.26: Cross flow with stochastic effects

For cross flows, the stochastic dynamics produce interesting results. It can be seen from the trajectories as well as the alignment that the stripe formation is improved in contrast to its deterministic counterpart from Fig. 5.12. Upon closer

inspection, we can see that the alignment for cases $\sigma \leq 0.2$ is much closer to 1 than that for $\sigma = 0$.

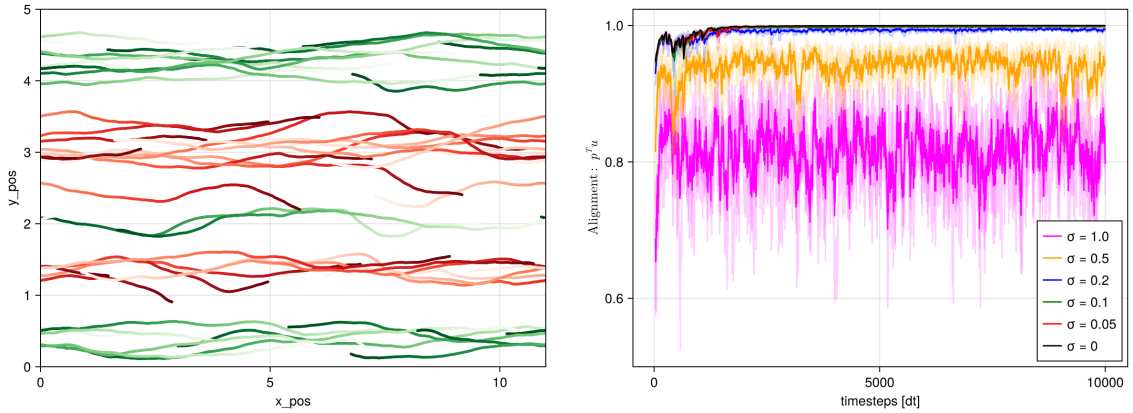


(a) Hamiltonian H over time (b) dH over time

Figure 5.27: Hamiltonian for cross flows with stochastic effects

This phenomenon is made even more apparent when look at the Hamiltonian, for small positive values of σ, the Hamiltonian is even less than the deterministic case. For values $\sigma \leq 0.2$, as indicated in [10], the trajectories of the pedestrians also show that the stripe formation is improved with the addition of noise in contrast to its deterministic counterpart Fig. 5.12. This counterintuitive phenomenon for the stochastic cross flow is known as 'noise-induced ordering' [33]. However, this stripe formation is disrupted with increase in noise, particularly for values $\sigma \geq 0.5$

- **Counter flow** ($\sigma \geq 0$)

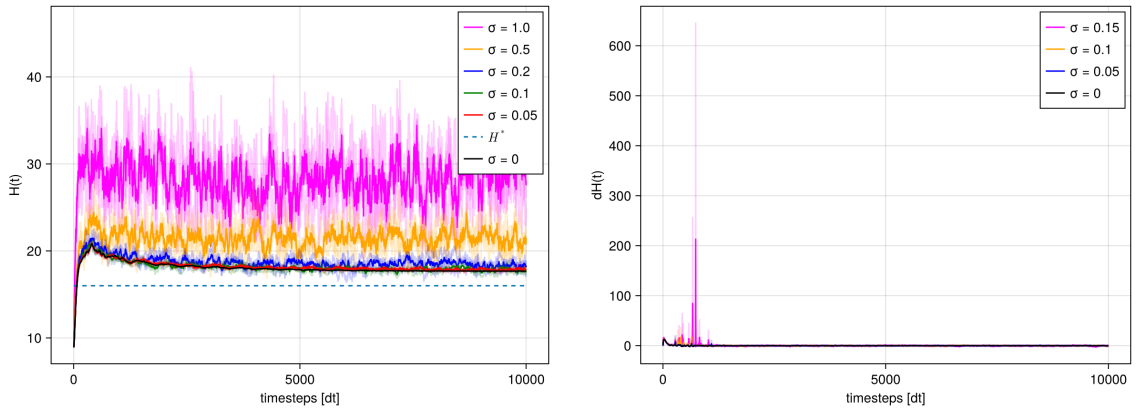


(a) Pedestrian trajectories for $\sigma = 0.2$ (b) Alignment

Figure 5.28: Counter flow with stochastic effects

In the case of the counter flow, however, we do not see an improvement in lane formation with addition of noise in the system. The lane formation is perturbed

with the increase in noise and becomes indistinguishable from disordered movement when $\sigma \geq 0.5$ as indicated both by the alignment and the Hamiltonian, signalling a phenomenon known as 'freezing by heating' [34].



(a) Hamiltonian H over time

(b) dH over time

Figure 5.29: Hamiltonian for counter flows with stochastic effects

All the dynamics shown in this section have the overall Hamiltonian greater than H^*, but for values of σ, the Hamiltonian rapidly fluctuates and the conditions for self-organization are not met. Indicating erratic movement of the overall dynamics of the agents, resulting in no phase changes from disordered to ordered dynamics. However, for low values of σ, the self-organization is improved for cross flows, resulting in stripe formations that are better than that of the deterministic case.

6 Complete Coding Examples

In this section we show coding examples to demonstrate to the reader how to set up a complete model. As we move on, we'll gradually add more customization options to our model setup to have more flexibility. We'll also show how to save and plot data obtained from our simulation, e.g. the pedestrians attributes such as positions and velocity, and also model attributes such as the Hamiltonian. The only files that we will be using from the root project directory are:

- `/activate_package.jl`
- `/PH_Project/src/module_ph_ped.jl`

As we will see in our example, the very first step to start a project would always be to activate the package by executing `activate_package.jl`. For any advanced customization, for e.g. changing the numerical solver for the `stochastic_ode_step`, we will have to manually modify the function Code. 4.13 within the `module_ph_ped.jl`.

6.1 Model Setup

To begin with our examples, let us create a new julia file `/main.jl` and activate our package.

```
include("./activate_package/activate_package.jl")
```

Example 1: Minimal model setup with default parameter values

```
model = initialize()
```

Code 6.1: Minimal code to construct a model with default parameter values

With a simple command shown above Code. 6.1, we can easily construct and initialize the model with default values. We've initialized the model with 32 pedestrians, in the space of 11 units and 5 units in the x and y directions respectively. To retrieve model properties, for e.g. `λ`, we can write `model.λ`. Similarly, for agent properties, we can write `model[1].pos` to retrieve the position of the agent with index 1.

```
StandardABM with 32 agents of type Pedestrian
agents container: Vector
space: periodic continuous space with [11.0, 5.0] extent and spacing=0.25
scheduler: fastest
properties: no_disp_H, A, λ, B, dH, dt, sigma, alignment, stoch_dH,
hamiltonian
```

Code 6.2: Output of Code. 6.1 in the terminal, showing model initialization with default values.

Example 2: Model setup with user-defined parameter values

```
seed = 42
properties = Dict(
    # constants
    :λ => 2, :A => 5, :B => 0.3, :dt => 0.01, :sigma => 0.1,
    # hamiltonian and alignment
    :no_disp_H => 0.0, :hamiltonian => 0.0, :dH => 0.0, :stoch_dH => 0.0,
    :alignment => 0.0
)
number_of_peds = 0
x_len, y_len = 11, 5
num_solver = leapfrog_step
```

Code 6.3: Initializing model parameters

If we wish to have flexibility in initializing the model parameters beforehand, we can do that by simply writing the arguments as shown in Code. 6.3 for the `initialize` function Code. 6.4.

```
model = initialize(
    number_of_peds, x_len, y_len, num_solver, properties; seed
    )
```

Code 6.4: Initialization of the model with modified parameters from Code. 6.3

As we have created the model with zero pedestrians, we can now manually add the pedestrians into our model space using `addagent!()` as shown below Code. 6.5, and initialize their attributes.

```
rng = Distributions.Xoshiro(seed)
number_of_peds = 40
for i in 1:number_of_peds
    Agents.add_agent!(model;
        pos = [rand(rng)*x_len, rand(rng)*y_len],  # Initial position
        vel = [0.0 ,0.0], # Initial velocity
        u_i = mod(i,2) == 0 ? [1,0] : [-1,0] # counter_flow
        # u_i = mod(i,2) == 0 ? [0,1] : [1,0] # cross
        # u_i = [1.0,0.0] # uni-directional flow
    )
end
```

Code 6.5: Initialization of the pedestrians

The `addagent!()` function adds a `Pedestrian` object as an agent into our model. The attributes of `Pedestrian` were defined in Code. 2.1, and hence `vel`, `pos`, and u_i are initialized in Code. 6.5.

6.2 Running the Model

Once we are satisfied with our model, the next step is to run it! Naturally, we would also want to keep track of how the model and agent attributes change as the simulation runs. Continuing with the model defined in Example 2, we'll run this model for 100 seconds of simulation time, and save the values of each attribute in the `adf` and `mdf` dataframes.

```
T = 100
total_steps = T/model.dt # or T/properties[:dt]
mdata = [:hamiltonian, :dH, :no_disp_H, :alignment, :stoch_dH]
adata = [:pos, :vel]
adf, mdf = Agents.run!(model, T; mdata, adata)
```

Code 6.6: Running the model and tracking the attributes

Here, `mdf` is a dataframe containing a record of the changes over time of all model attributes declared in `mdata`, similarly the agents attributes declared in `adata` are saved in the `adf` dataframe.

Visualizing Model Data

Once the simulation is finished and the dataframes are obtained, we can easily analyze them. As an example, we will illustrate how we can plot the data, and finally show how we can construct a simple interactive application to simulate our model.

Continuing with the dataframes obtained in Code. 6.6, we will plot them using the `CairoMakie` backend.

```
t = model.dt:model.dt:T
h_data, h_star_data = mdf[2:end,:hamiltonian] , mdf[2:end,:no_disp_H]

CairoMakie.activate!()
fig, ax = lines(t, h_data)
lines!(ax, t, h_star_data, linestyle=:dash, label=L"H^*")

ax.ylabel = "H"
ax.xlabel = "time [s]"
ax.title = "Hamiltonian for counter flow | n = $number_of_peds"
axislegend()
fig
```

Code 6.7: Example code to plot model attributes

Here, we have ignored the data from the first step as it only contains the initialized values of the model. By executing Code. 6.7, we'll obtain the following plot Fig. 6.1

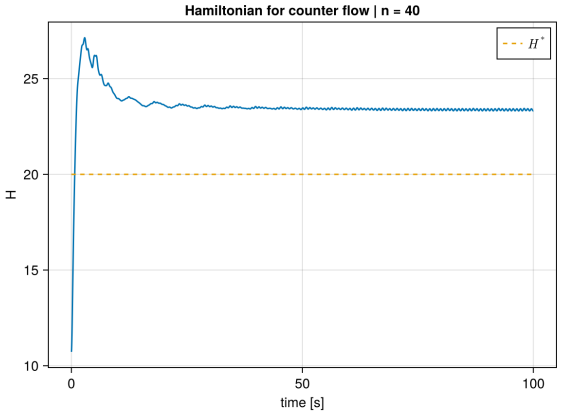


Figure 6.1: Model attribute plot from executing Code. 6.7

Visualizing Agent Trajectories

To plot the trajectories of every agent, we use the agent dataframe `adf` obtained from Code. 6.6. We loop through each `agent.id` and take its positions within a certain time. Since we have simulated a counter flow Code. 6.5, it would be nice to have a clear distinction between the two groups of agents based on their desired velocities, hence we color them individually.

```
path_length = 100
start_time = Int(T/model.dt) - path_length
end_time = start_time + path_length
fig = Figure()
ax = Axis(fig[1, 1],
          limits = (0,x_len,0,y_len),
          xlabel = "x_pos", ylabel = "y_pos")
for i in 1:number_of_peds
    if mod(i,2) == 0
        scatter!(ax, adf[adf.id .== i, :].pos[start_time:end_time],
                color=0:path_length, colormap=:Reds, markersize=5)
    else
        scatter!(ax, adf[adf.id .== i, :].pos[start_time:end_time],
                color=0:path_length, colormap=:Greens, markersize=5)
    end
end
fig
```

Code 6.8: Example code to plot agent trajectories

We want to see the trajectories of all agents through the 100 timesteps of the simulation. We first set the figure and axis to a defined extent based on the simulation domain that we've defined in Code. 6.3. Then loop through `adf`, to get the agent trajectories of each agent. Executing the code results in the plot below

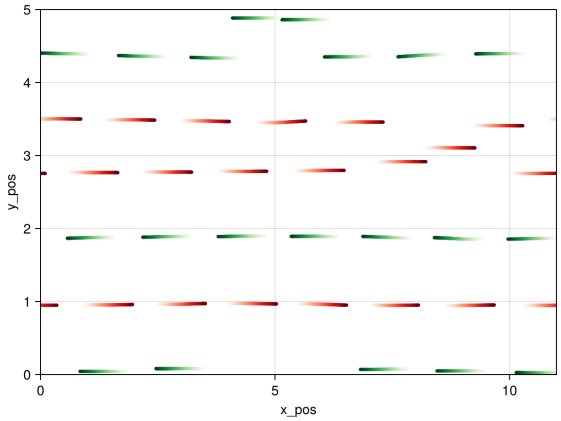


Figure 6.2: Agent trajectories from executing Code. 6.8

6.3 Interactive Visualization

To construct an interactive application, we'll use the `GLMakie` backend. Since we want to restart the model every time we open the app, we will initialize the model beforehand, thus we will combine all the codes from example 2 into one function `model_init` which will return the initialized model.

To ignore the first initialized values when plotting the data, we run the model for a single time-step using `Agents.step!(model)`.

```
GLMakie.activate!()
model = model_init()
Agents.step!(model)
params = Dict(
    :λ => 0:0.1:5,
    :A => 0:0.1:10,
    :B => 0:0.1:2,
    :sigma => 0:0.05:1
)
mdata = [:hamiltonian, :dH, :alignment]
fig, abmobs = Agents.abmexploration(model; dt=1:0.2:5, params, mdata)
fig
```

Code 6.9: Example code to generate the interactive app

Sliders can also be added by creating a dictionary with key value pairs of the model attributes with step ranges shown in Code. 6.9. Finally, to generate the plots of the attributes, we can include `mdata` with a list of attributes to track, similar to how it was done when running the model in Code. 6.6. With these arguments in `Agents.abmexploration`, one can create a simple interactive GUI as shown below in Fig. 6.3.

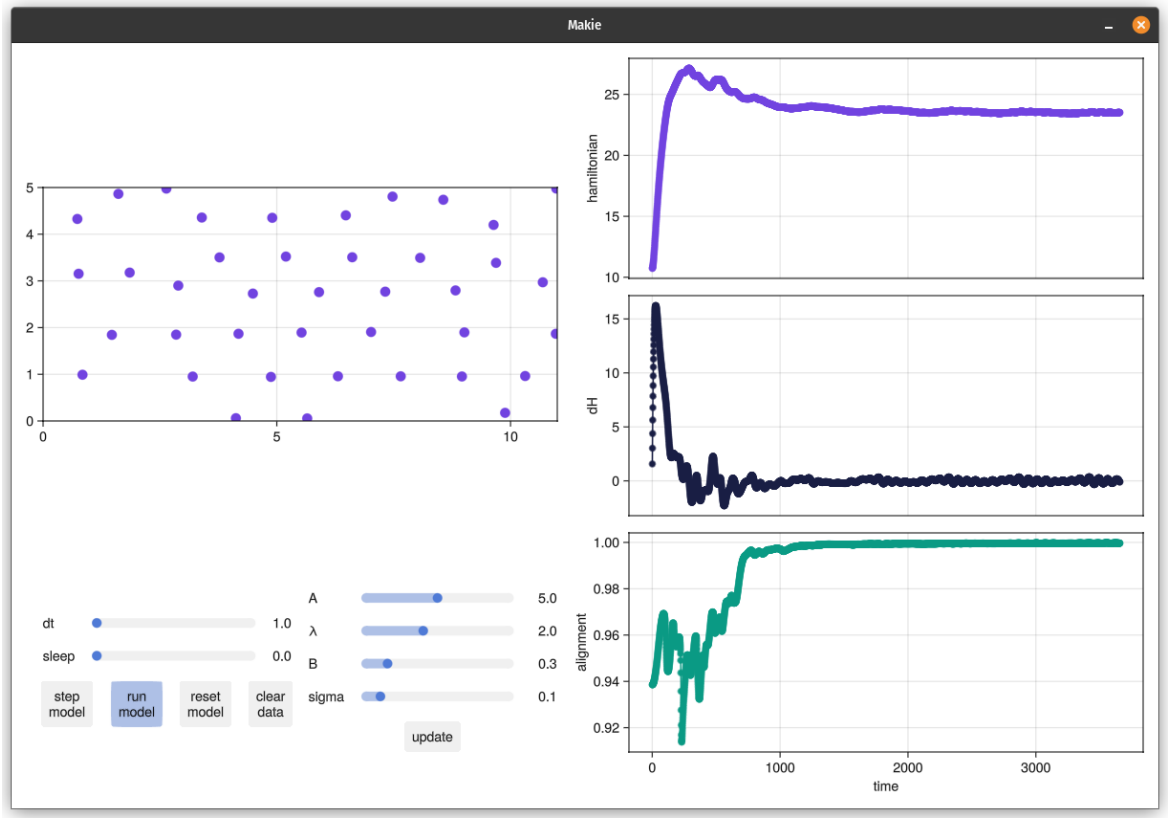


Figure 6.3: Interactive application to simulate the model by executing Code. 6.9

Of course, with the ease and flexibility provided by the `Agents.jl` package, one can easily construct more customizable agent based models apps without hassle.

In this section we demonstrated how to easily construct and run our model, we saved and plotted the model and agent data, and finally we also saw how to create an interactive environment to simulate our model with just a few lines of code.

7 Conclusion

We were successfully able to show the microscopic force-based pedestrian dynamics model through port-Hamiltonian formulation. We witnessed a shift in perspective from microscopic models, where we inspect agent interactivity on an individual level; to a holistic description of the system-wide interactivity through measuring the system energy. This energy-based perspective, enabled by port-Hamiltonian systems, allowed us to not only observe collective behavior, but also to identify it through quantitative means using the Hamiltonian and H^*. With the port-Hamiltonian formulation, we were able to replicate the findings from [11] of well-known collective phenomenon such as lane and stripe formation.

We can also observe that the interaction between the pedestrians is isotropic, this is due to the skew-symmetric nature of Hamiltonian systems. A physical interpretation of this, in the context of our model, assumes that the pedestrians interact equally with all of its surroundings, hence there are no vision cones effects that may produce bias in interaction in the direction of motion [7]. An attempt to incorporate such anisotropic effects within the port-Hamiltonian approach includes the use of state-dependent input terms [11] though it remains an active part of research.

Furthermore, we were also able to show the impact of induced noise on these collective phenomenon, in particular, "noise-induced ordering" for stripe formation [10, 33]. In the case of lane formation, the collective phenomenon was disrupted with increase in volatility, which is in accordance with the "freezing by heating" phenomenon [34], however formation of the clusters of agents could not be reproduced.

With this thesis, we've been able to show that port-Hamiltonian systems, although a relatively young area of research, promises a modeling approach to quantify collective dynamics. Because it models systems from the perspective of energy-based interactions, it is more abstract, and thus useful to understand the underlying dynamics of systems from to various disciplines.

Working on this project has been a great learning experience for me. The study of complex systems, collective behavior, emergent phenomenon, and nonlinear dynamics have always been alluring to me, yet also enigmatic. I am glad to have been introduced to a topic such as stochastic port-Hamiltonian systems as they provide a good tool for studying these topics. With this I've been introduced to a plethora of different paths to take, each a world of its own, such as stochastic differential equations, port-Hamiltonian systems on infinite-dimensional spaces, geometric integration, and differential forms, to name a few. These are a few topics that I would like to explore more as I dive deeper into the study of collective phenomenon through the lens of port-Hamiltonian systems.

Thank you.

List of Figures

List of Codes

References

[1] Frank Schweitzer. *Self-organization of complex structures: From individual to collective dynamics*. CRC Press, 1997.

[2] F Eugene Yates. *Self-organizing systems: The emergence of order*. Springer Science & Business Media, 2012.

[3] François Gu et al. "Emergence of collective oscillations in massive human crowds". In: *Nature* 638.8049 (2025), pp. 112–119.

[4] Hwasoo Yeo and Alexander Skabardonis. "Understanding stop-and-go traffic in view of asymmetric traffic theory". In: *Transportation and Traffic Theory 2009: Golden Jubilee: Papers selected for presentation at ISTTT18, a peer reviewed series since 1959*. Springer, 2009, pp. 99–115.

[5] Mohcine Chraibi, Antoine Tordeux, and Andreas Schadschneider. "A force-based model to reproduce stop-and-go waves in pedestrian dynamics". In: *Traffic and Granular Flow'15*. Springer. 2016, pp. 169–175.

[6] Mehdi Moussaïd. "Opinion formation and the collective dynamics of risk perception". In: *PLoS One* 8.12 (2013), e84592.

[7] Dirk Helbing and Peter Molnar. "Social force model for pedestrian dynamics". In: *Physical review E* 51.5 (1995), p. 4282.

[8] Pratik Mullick et al. "Analysis of emergent patterns in crossing flows of pedestrians reveals an invariant of 'stripe' formation in human data". In: *PLoS Computational Biology* 18.6 (June 2022). Ed. by Feng Fu, e1010210. ISSN: 1553-7358. DOI: 10.1371/journal.pcbi.1010210.

[9] Anna Sieben, Jette Schumann, and Armin Seyfried. "Collective phenomena in crowds — Where pedestrian dynamics need social psychology". In: *PLoS one* 12.6 (2017), e0177328.

[10] Basma Khelfa et al. "Initiating lane and band formation in heterogeneous pedestrian dynamics". In: *Collective Dynamics* 6 (2021), pp. 1–13.

[11] Antoine Tordeux and Claudia Totzeck. *Multi-scale description of pedestrian collective dynamics with port-Hamiltonian systems*. 2023. arXiv: 2211.06503 [physics.soc-ph]. URL: https://arxiv.org/abs/2211.06503.

[12] Jochen Fromm. *The emergence of complexity*. Kassel university press Kassel, 2004.

[13] Cécile Appert-Rolland et al. *Modelling vehicle and pedestrian collective dynamics: Challenges and advancements*. 2024. arXiv: 2410.22896 [physics.soc-ph]. URL: https://arxiv.org/abs/2410.22896.

[14] Arjan Van Der Schaft, Dimitri Jeltsema, et al. "Port-Hamiltonian systems theory: An introductory overview". In: *Foundations and Trends® in Systems and Control* 1.2-3 (2014), pp. 173–378.

[15] Vincent Duindam et al. *Modeling and control of complex physical systems: the port-Hamiltonian approach*. Springer Science & Business Media, 2009.

[16] Arjan van der Schaft. "Port-Hamiltonian nonlinear systems". In: *arXiv preprint arXiv:2412.19673* (2024).

[17] Arjan J van der Schaft. "Port-Hamiltonian systems: network modeling and control of nonlinear physical systems". In: *Advanced dynamics and control of structures and machines*. Springer, 2004, pp. 127–167.

[18] Arjan J Van der Schaft and Bernhard M Maschke. "Port-Hamiltonian systems on graphs". In: *SIAM Journal on Control and Optimization* 51.2 (2013), pp. 906–937.

[19] Dong Xue, Sandra Hirche, and Ming Cao. "Opinion behavior analysis in social networks under the influence of coopetitive media". In: *IEEE Transactions on Network Science and Engineering* 7.3 (2019), pp. 961–974.

[20] Ion Matei et al. "Inferring particle interaction physical models and their dynamical properties". In: *2019 IEEE 58th Conference on Decision and Control (CDC)*. IEEE. 2019, pp. 4615–4621.

[21] Miel Sharf and Daniel Zelazo. "Analysis and synthesis of MIMO multi-agent systems using network optimization". In: *IEEE Transactions on Automatic Control* 64.11 (2019), pp. 4512–4524.

[22] Ramy Rashad et al. "Port-Hamiltonian modeling of ideal fluid flow: Part I. Foundations and kinetic energy". In: *Journal of Geometry and Physics* 164 (2021), p. 104201.

[23] Harshit Bansal et al. "Port-Hamiltonian formulation of two-phase flow models". In: *Systems & Control Letters* 149 (2021), p. 104881.

[24] Francesco Cordoni, Luca Di Persio, and Riccardo Muradore. "Stochastic port-Hamiltonian systems". In: *Journal of Nonlinear Science* 32.6 (2022), p. 91.

[25] Barbara Rüdiger, Antoine Tordeux, and Baris E Ugurcan. "Stability analysis of a stochastic port-Hamiltonian car-following model". In: *Journal of Physics A: Mathematical and Theoretical* 57.29 (2024), p. 295203.

[26] Matthias Ehrhardt, Thomas Kruse, and Antoine Tordeux. "The collective dynamics of a stochastic port-Hamiltonian self-driven agent model in one dimension". In: *ESAIM: Mathematical Modelling and Numerical Analysis* 58.2 (2024), pp. 515–544.

[27] George Datseris, Ali R. Vahdati, and Timothy C. DuBois. "Agents.jl: a performant and feature-full agent-based modeling software of minimal code complexity". In: *SIMULATION* 0.0 (Jan. 2022), p. 003754972110688. DOI: 10.1177/00375497211068820.

[28] Christopher Rackauckas and Qing Nie. "DifferentialEquations.jl–a performant and feature-rich ecosystem for solving differential equations in Julia". In: *Journal of Open Research Software* 5.1 (2017).

[29] Mohcine Chraibi et al. “Force-based models of pedestrian dynamics”. In: *Networks and Heterogeneous Media* 6.3 (2011), pp. 425–442.

[30] Bernt Øksendal. *Stochastic differential equations*. Springer, 2003.

[31] Antoine Tordeux, Mohcine Chraibi, and Armin Seyfried. “Collision-free speed model for pedestrian dynamics”. In: *Traffic and Granular Flow'15*. Springer. 2016, pp. 225–232.

[32] Jialin Hong and Liying Sun. *Symplectic integration of stochastic Hamiltonian systems*. Springer, 2022.

[33] Otti D'Huys et al. “Canard resonance: on noise-induced ordering of trajectories in heterogeneous networks of slow-fast systems”. In: *Journal of Physics: Photonics* 3.2 (2021), p. 024010.

[34] Dirk Helbing, Illés J Farkas, and Tamás Vicsek. “Freezing by heating in a driven mesoscopic system”. In: *Physical review letters* 84.6 (2000), p. 1240.