Notes on *Machine Learning for Crowd Modelling and Simulation*[1]

The dynamics of human crowds form complex systems, where many people interact and react, leading to diverse patterns of behavior. Studying crowds scientifically covers a broad range of areas, such as creating mathematical models, developing algorithms for machine learning, examining experimental and simulated data, implementing simulation software, and conducting research in psychology and sociology.

Modeling of Human Crowds

State Space

Cellular automata serve as computational tools for representing complex systems, consisting of multiple interacting components [Wol84, Wol83]. While individual components are typically simple, their interactions can lead to intricate behaviors. An illustrative example of a cellular automaton with elementary rules resulting in complex behavior is Conway's "Game of Life" [Gar70]. Similarly, a cellular automaton can be used to simulate a human crowd, where individuals occupy distinct cells and interact through simple rules with each other, obstacles, and objectives. Boccara's book provides a comprehensive overview of cellular automata [Boc04].

We start by setting up a basic modeling environment that serves as the foundation for more complex logic. In our crowd simulation model, we use a simple cellular automaton approach, with discrete-time updates and consistent time shifts. The simple pseudo-code for the update logic is given in algorithm 1:

Algorithm 1 Update scheme for the cellular automaton

Data: Initial automaton state $x^{(0)} \in X$ at time step $n = 0$
Result: Iterate over the automaton states.
while *true* **do**
  Calculate the next state: $x^{(n+1)} \leftarrow f(x^{(n)})$ Advance time step: $n \leftarrow n + 1$ **if** *termination condition is met* **then**
    break

To proceed, the following certain tasks are fulfilled:

1. Basic visualization.
2. Adding pedestrians in cells.
3. Adding targets in cells.
4. Adding obstacles by making certain cells inaccessible.
5. Simulation of the scenario (being able to move the pedestrians).

Figure 1 Illustrates the basic visualization of the automaton, where the red, blue, and pink cells represent pedestrians, targets, and obstacles, respectively. It has also been ensured that an external file can adjust the number of pedestrians and targets. Next, the scenario simulation was run, such that the pedestrians rush towards the targets. The light pink cells represent the visited cells, intending the visualize the pedestrians' path to the targets.

Setup

The detailed description of the setup to achieve the above mentioned tasks within the cellular automaton network is as follows:

[1] I wish to express my gratitude to Prof. Felix Dietrich at TUM, who served as the main lecturer for this course. Enclosed are my notes and contributions from the course, highlighting aspects that I find particularly noteworthy and relevant to the context of Machine Learning for Crowd Modelling and Simulation.

(a) **Grid Visualization**: The simulation begins with creating a grid, which serves as the spatial environment for the crowd scenario. This grid is visualized to provide a clear representation of the scenario. Each cell in the grid is associated with a specific state, such as empty, target, obstacle, or pedestrian. Visualizing the grid is an essential component for representing the spatial layout of the scenario.

(b) **Pedestrian Definition**: To model the behavior of individuals in a crowd, the code defines a Pedestrian class. This class allows for the creation of individual pedestrians, each with distinct attributes:

- Position: The initial position of the pedestrian within the grid.
- Desired Speed: The desired speed at which the pedestrian aims to move.

(c) **Target Placement**: Targets are crucial elements within the simulation. They represent the destinations or objectives that pedestrians aim to reach. The code allows for the placement of targets within the grid, which pedestrians will navigate towards. The positioning of targets plays a significant role in determining the flow of the crowd.

(d) **Obstacle Configuration**: Obstacles are introduced into the scenario to create barriers that pedestrians must navigate around. Obstacles are represented as inaccessible grid cells, and their placement within the grid impacts the paths pedestrians take. This feature enables the modeling of realistic scenarios where physical barriers influence pedestrian movement.

(e) **Simulation Process**: The simulation process involves the movement of pedestrians toward their target positions while considering collision avoidance and obstacle navigation. The simulation operates in discrete time steps, with pedestrians making decisions based on their desired speeds and the distance to their targets. The steps in the simulation process include:

- Calculating the distance to the target for each pedestrian.
- Determining the potential next cell for each pedestrian based on the target distance.
- Checking for obstacles and other nearby pedestrians to avoid collisions.
- Updating the positions of pedestrians based on their desired speeds and obstacle-free paths.
- Tracking the progress and movement of pedestrians throughout the simulation.

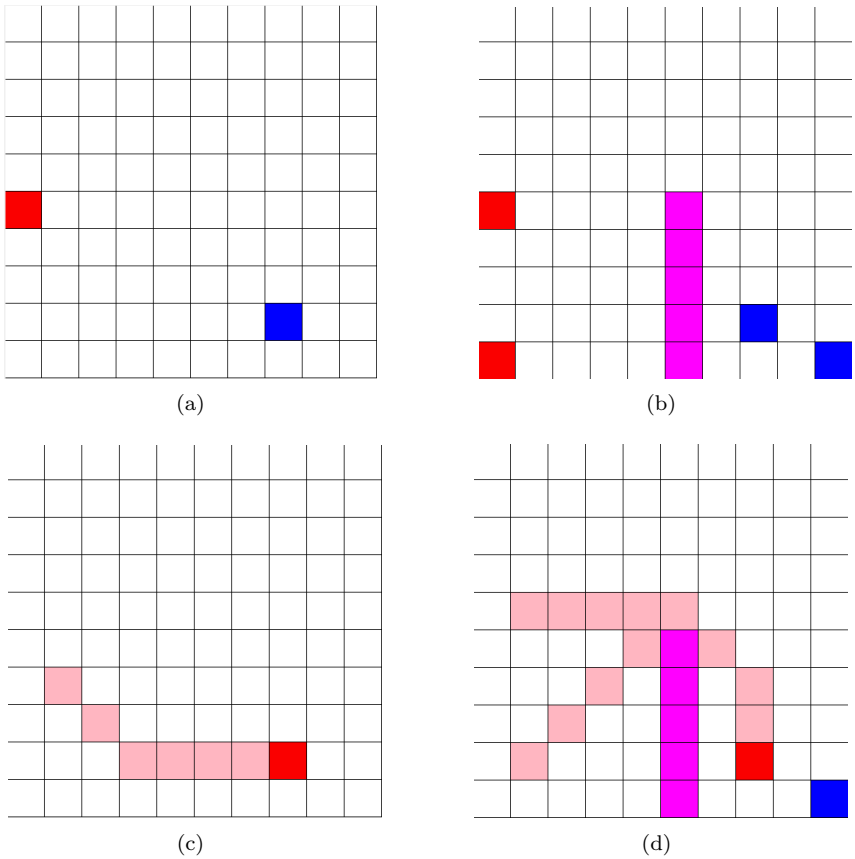


Figure 1: (a) Basic visualization of the automaton with 1 pedestrian and 1 target in cells. (b) Basic visualization of the automaton with multiple pedestrians and targets in cells. (c) The resulting "trail" of the path towards the target, for scenario (a). (d) The resulting shared "trail" of the paths toward the target, for scenario (b).

Scenario definition

As an example case, the following scenario has been simulated in 25 time steps:

- The environment is composed of 2500 cells (50x50 square)
- A single pedestrian is placed at the coordinate (5,25), which is 20 cells away from its target lies on (25,25) Figure 2 describes the scenario and the results of the simulation. The pink cells (visited cells) serve the role of the "trail" of the path, which is for demonstration purposes of the path and to avoid a batch of 20 images in the report.

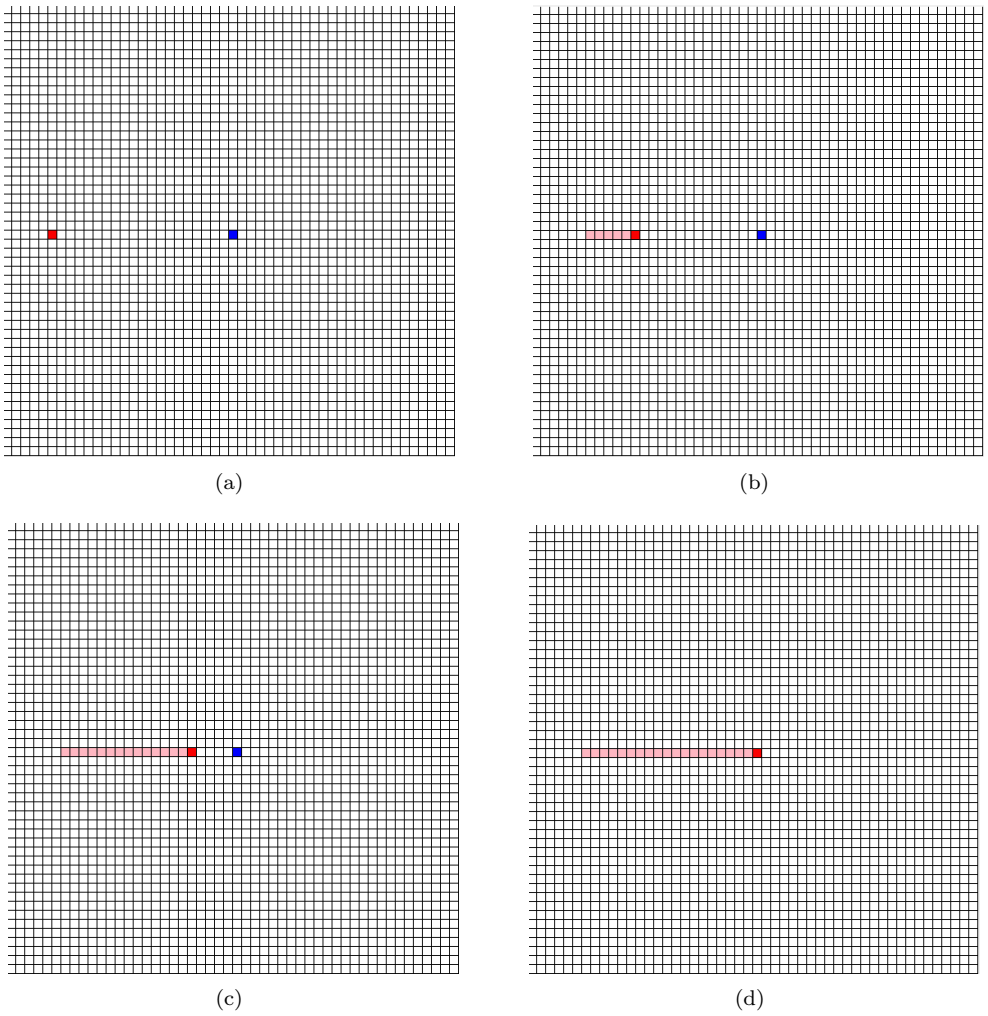


Figure 2: (a) Initial version of the scenario (b) Resulting state after 6 steps c) Resulting state after 15 steps d) Terminal state (resulting state after 20 steps)

Lastly, let us concisely discuss some details about obstacle avoidance.

Rudimentary obstacle avoidance

Obstacle avoidance for pedestrians has been achieved by adding a cost penalty for obstacles. We then test the effects of obstacle avoidance algorithm in a particular scenario called 'chicken test'. Dijkstra algorithm has been used as a path finding find algorithm. Cells were flooded with distance values, starting with zero distance value at the target, such that obstacle cells are not included in the set of possible cells and are set to very high values to prevent pedestrians from reaching obstacles. The cost function to update the state is gives as:

$$Cost = d(_{C_{N,i,j}, C_{T,k,l}}) = \sqrt{(i-k)^2 + (j-l)^2} \tag{1}$$

that is, the Euclidean distance between the cell indices (i,j) and (k,l), where i and k are the row indices, and j and l are the column indices of the cells. The neighboring cell is called C_n, and the target cell is called C_t.

Additionally, the concepts such as interaction of the pedestrians, obstacle avoidance, and test case evaluations of the RiMEA paper [BKL$^+$09], have been collaborated. Next independent topic is Simulation Software.

Simulation Software

Experiments were conducted to explore the functionalities of the Vadere software, a tool utilized for simulating and visualizing human crowds. The objective of this exercise was to familiarize oneself with the software's graphical user interface, as well as to gain proficiency in creating, executing, and modifying simulation scenarios. Additionally, the report aimed to demonstrate the process of integrating a new SIR model into the Vadere software. This knowledge would prove valuable not only for utilizing Vadere but also for working with other simulation tools or implementing custom ones. To commence the exercise, the Vadere software was obtained from the following link: http://www.vadere.org/releases/. It is important to note that the "master branch" should be used instead of the stable branch, as the SIR code may not be compatible with the latter. The source code for Vadere can be acquired from the LRZ gitlab project, accessible via this link: https://gitlab.lrz.de/vadere/vadere.

Setting up the Vadere environment

The objective of this task was to establish the experimental environment by installing the Vadere software via the master branch of the source code. This initial step was successfully executed by adhering to the comprehensive guidelines provided on Moodle for Vadere setup. Subsequently, the recreation of RiMEA scenarios 1 and 6, along with the "chicken test" scenario, was undertaken, employing the Optimal Steps Model as the chosen pathfinding algorithm.

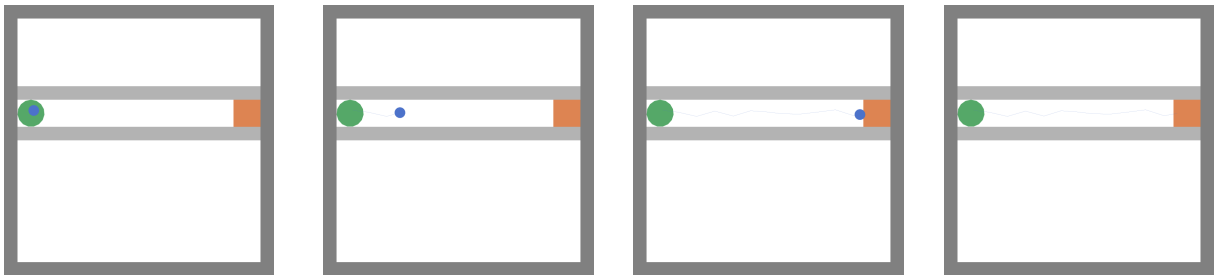
We begin with examining the findings from the RiMEA 1 scenario. The simulation results of the RiMEA 1 scenario, depicted in Figure 3 using the Optimal Steps Model, reveal an interesting observation. Even though the pedestrians are positioned directly in front of each other, the trajectory of their movement, as decided by the Optimal Steps model, deviates from a perfectly straight line. Instead, the model accurately emulates the natural, non-linear steps characteristic of human movement.

The implicit functionality of the Optimal Steps Model contributes to a realistic representation of pedestrian behaviour. Notably, pedestrians instinctively decelerate when navigating through dense crowds, adopting smaller steps. This observed pedestrian behaviour stands in stark contrast to the model developed in our previous exercise sheet, where pedestrians followed a perfectly straight line.

In fact, this type of behaviour is not exclusive to the RiMEA 1 scenario; we observe a similar pattern in both the RiMEA 6 and Chicken Test Scenarios (Figure 4 and Figure 5). The fundamental distinction in behavior between this model and ours stems from our use of the Dijkstra algorithm for determining the shortest path. However, it becomes evident that crowd simulation requires a more intricate approach than simply applying a shortest-path algorithm to reach the destination.

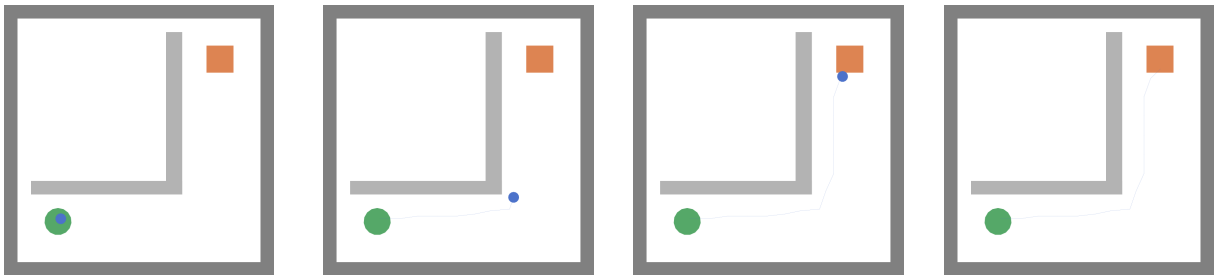
To better illustrate the complexity of the situation, we provide a simulation of a crowd—comprising multiple pedestrians headed towards the goal—in Figure 7. This simulation serves to highlight the different interactions and dynamic adjustments in movement that occur within a crowd setting, emphasizing the limitations of a simplistic shortest-path approach in capturing the intricacies of real-world pedestrian behaviour.

Lastly, regarding visualization differences, the Vadere software offers a more sophisticated setup, featuring numerous options to visualize dynamic instantaneous direction, path trails of all pedestrians, grouping pedestrians and other advantages that accommodate more behaviour options for pedestrians. This enhanced visualization capability provides a more detailed and different representation of pedestrian dynamics, allowing for a comprehensive analysis of their movement patterns and interactions within the environment. Yet, points like the grid structure and eventually reaching to the destination, not stepping on other pedestrians, are similar characteristics to our earlier model.



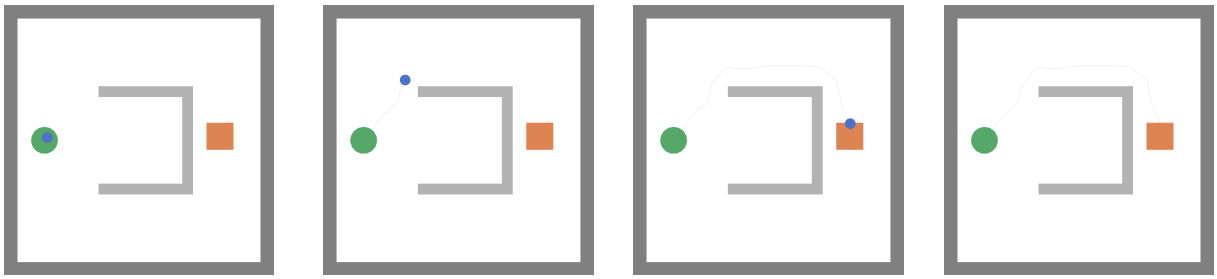
(a) (b) (c) (d)

Figure 3: RiMEA 1 Scenario. (a) Initial state (b) Pedestrian starts its path by 'non-straight' line towards the target c) Pedestrian has been drawing 'non-straight' path in the entire route d) Terminal state



(a) (b) (c) (d)

Figure 4: RiMEA 6 Scenario. (a) Initial state (b) Pedestrian starts its path by 'non-straight' line towards the target c) Pedestrian have been drawing 'non-straight' path in the entire route d) Terminal state



(a) (b) (c) (d)

Figure 5: Chicken Test Scenario. (a) Initial state (b) Pedestrian starts its path by 'non-straight' line towards the target c) Pedestrian have been drawing 'non-straight' path in the entire route d) Terminal state

Simulation with different models

In this section, we compare simulations using various path methods for the specified scenarios. In order to aid in the more precise monitoring of individual pedestrian actions, Figure 6 offers a brief outline of how one single pedestrian begins walking in the direction of the objective, followed by a complete summary of the entire path. The dual presentation aims to offer insights into both the initial movement decisions and the overall trajectories of pedestrians influenced by the diverse path algorithms across various scenarios. As mentioned before, Figure 7 portrays a better picture of the crowd dynamics for the given algorithms. Our attempt to highlight the differences between the Optimal Steps Model, Social Force Model and Gradient Navigation Model is as follows: The Optimal Steps Model closely mirrors real-world human walking patterns. However, in this model, pedestrians appear somewhat detached, as they don't make a cooperative effort to collaborate despite all heading towards the same target. This is in contrast to the crowd dynamics observed in the Social Force Model, where pedestrian movement is likened to being influenced by "social forces." These forces do not directly stem from the pedestrians' immediate surroundings but rather serve as a representation of individuals' internal motivations guiding their actions or movements. In real-life scenarios, the Social Force Model may accurately reflect the psychological implications on the overall pedestrian community, suggesting that individual members tend to stay close to a common path due to the aggregated flow of their actions. On the contrary, the Gradient Navigation Model closely resembles our previous implementation, seeking the shortest path to the target in a greedy manner. Interestingly, the 'social' aspect appears to have minimal impact in this model, as the overall paths individuals take toward the goal show almost no variations, regardless of whether they are walking alone or with other pedestrians.

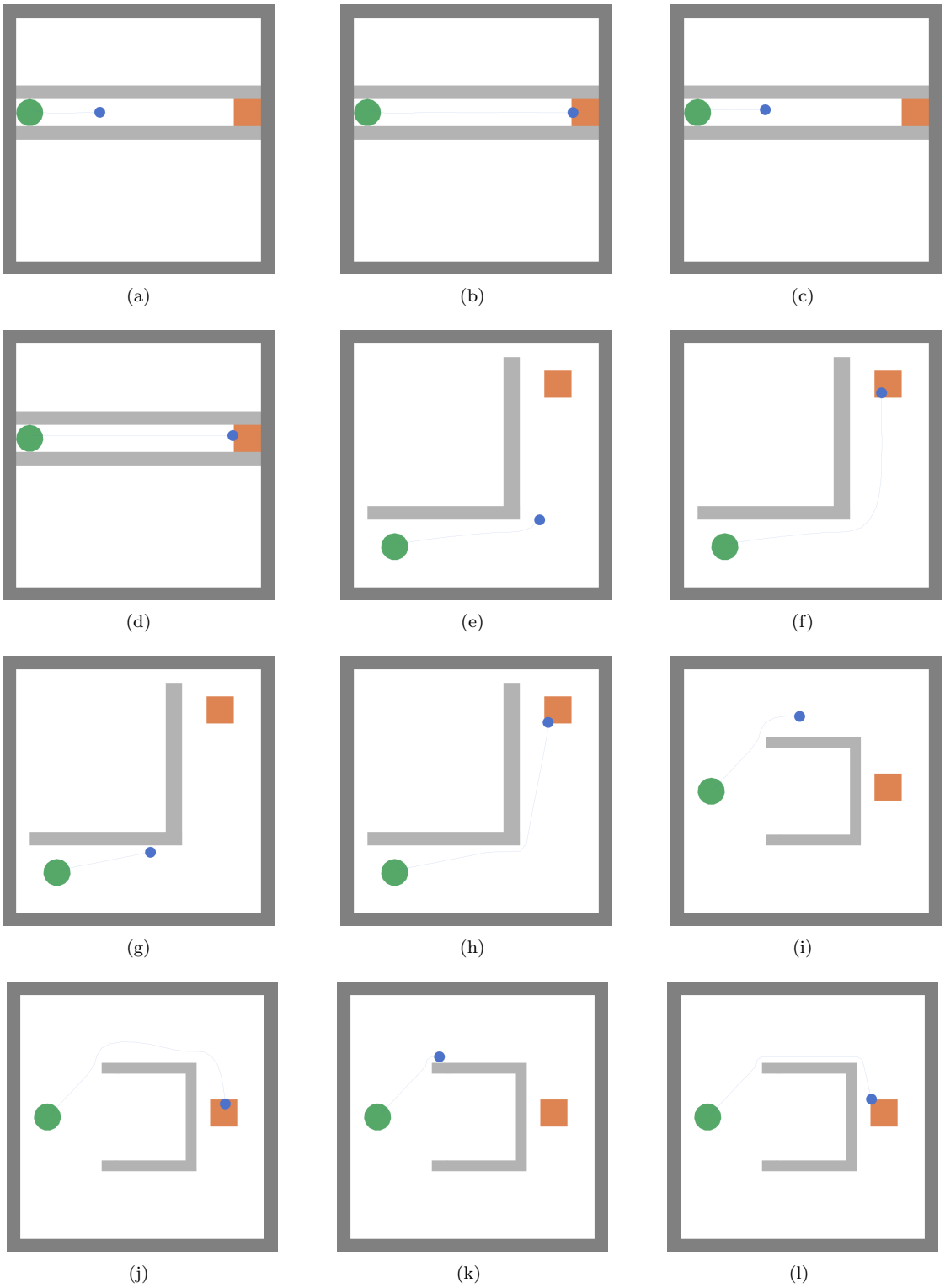


Figure 6: Single pedestrian simulations. (a) RiMEA 1 Social Force Model Pedestrian starting path b) RiMEA 1 Social Force Model Pedestrian entire path c) RiMEA 1 Gradient Navigation Model Pedestrian starting path d) RiMEA 1 Gradient Navigation Model Pedestrian entire path e) RiMEA 6 Social Force Model Pedestrian starting path f) RiMEA 6 Social Force Model Pedestrian entire path f) RiMEA 6 Gradient Navigation Model Pedestrian starting path g) RiMEA 6 Gradient Navigation Model Pedestrian entire path (a) Chicken Test Scenario Social Force Model starting path (b) Chicken Test Scenario Social Force Model entire path f) Chicken Test Scenario Gradient Navigation Model Pedestrian starting path g) Chicken Test Scenario Gradient Navigation Model Pedestrian entire path.

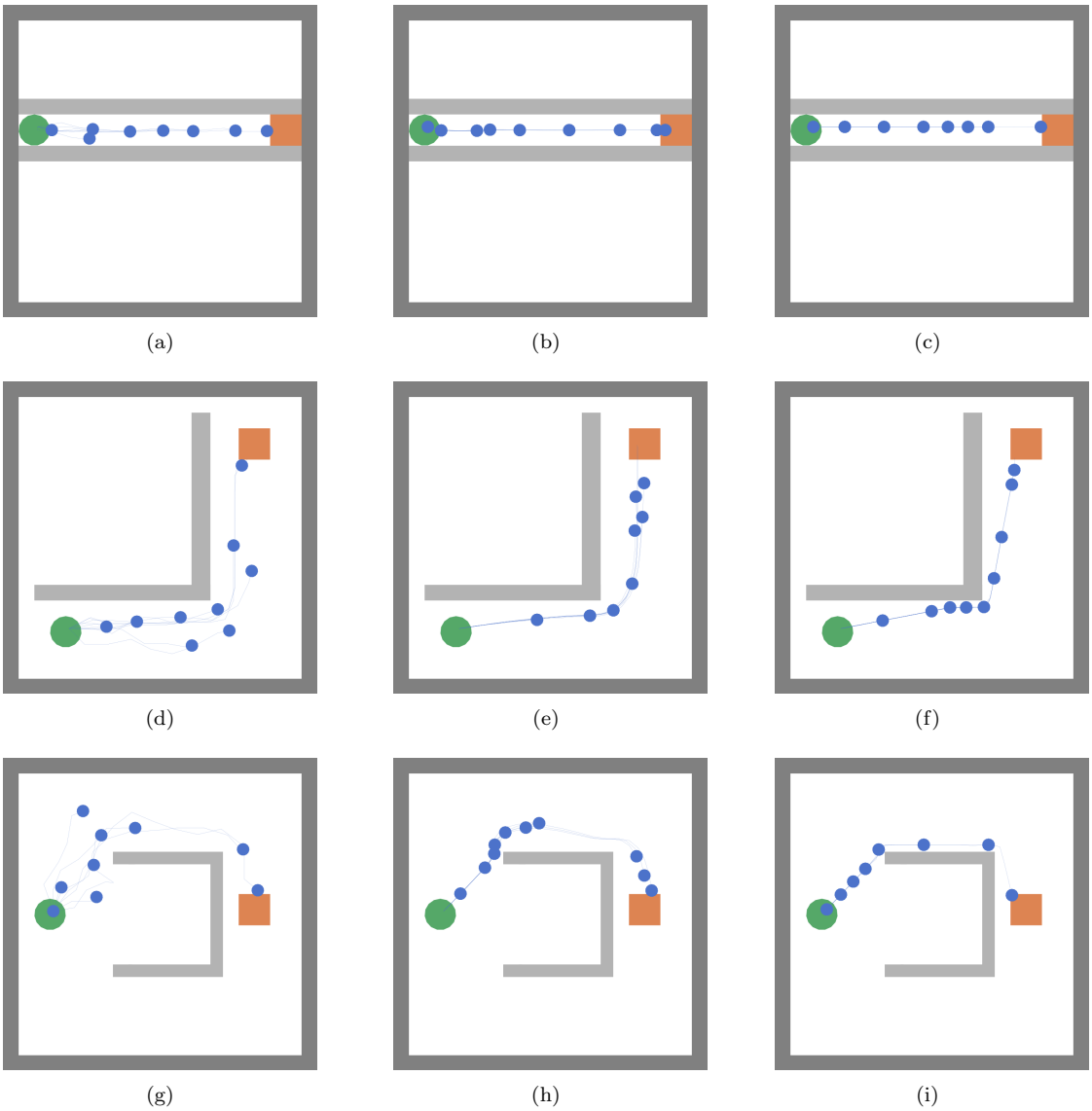


Figure 7: Multiple (10) pedestrian simulations. (a) RiMEA 1 Optimal Steps Model b) RiMEA 1 Social Force Model c) RiMEA 1 Gradient Navigation Model (d) RiMEA 6 Optimal Steps Model e) RiMEA 6 Social Force Model f) RiMEA 6 Gradient Navigation Model (g) Chicken Test Optimal Steps Model h) Chicken Test Social Force Model i) Chicken Test Gradient Navigation Model

Next independent topic is about data representations.

Representations of Data

PCA of a 2D Space onto a linear subspace

The objective of this task is to perform Principal Component Analysis (PCA) using Singular Value Decomposition (SVD) on a two-dimensional dataset (`pca_dataset.txt` on Github). The goal is to identify the one-dimensional, linear subspace that optimally reduces variance. This report outlines a step-by-step solution, including mathematical details, and concludes with the visualization of the dataset with its principal components. The given data can be visualized as follows,

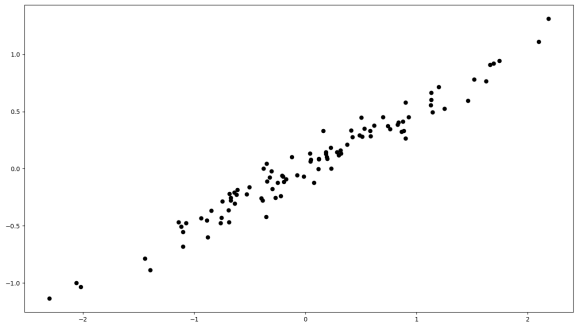


Figure 8: Data

The dataset is loaded, and each feature is centered by subtracting the mean. More specifically, given the original data X, with n data points, a centred data X_{centered} is obtained by:

$$X_{\text{centered}} = X - \bar{X} \quad ; \text{where } \bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i \tag{2}$$

The centered data matrix X_{centered} is decomposed into three matrices using the Singular Value Decomposition (SVD):

$$X_{\text{centered}} = U\Sigma V^T \tag{3}$$

where:

$$\begin{aligned} U &\text{ is an } m \times m \text{ orthogonal matrix (left singular vectors)},\\ \Sigma &\text{ is an } m \times n \text{ diagonal matrix with singular values},\\ V^T &\text{ is an } n \times n \text{ orthogonal matrix (right singular vectors)}. \end{aligned}$$

The detailed expression of the decomposition is given by:

$$X_{\text{centered}} = \sum_{i=1}^{r} \sigma_i u_i v_i^T \tag{4}$$

where:

$$\begin{aligned} r &\text{ is the rank of } X_{\text{centered}},\\ \sigma_i &\text{ are the singular values},\\ u_i &\text{ are the columns of } U,\\ v_i &\text{ are the columns of } V. \end{aligned}$$

The principal components of the centered data matrix X_{centered} can be obtained from the right singular vectors matrix V of its Singular Value Decomposition (SVD). In our implementation V is found as:

$$V = \begin{bmatrix} -0.88938337 & 0.45716213 \\ -0.45716213 & -0.88938337 \end{bmatrix} \tag{5}$$

The principal components matrix is given by the columns of V (or by the rows of V^T). Let k be the number of principal components we want to retain. The first k principal components are the first k columns of V, denoted as V_k. In our case $k = 2$, as we are in 2-dimensional space (i.e. X is a 100×2 matrix). Hence, we end up with two principle components: $V_1 = [-0.889, -0.457]$ (the first column of V) and $V_2 = [0.457, -0.889]$ (the second column of V). In a well-implemented PCA, the principal components should be orthogonal (form a right angle) because PCA aims to find the directions in which the data varies the most. Looking at the values of V_1 and

V_2, we verify that, they are, indeed, orthogonal (perpendicular) to each other, as the dot product of vectors V_1 and V_2 is given by:

$$V_1 \cdot V_2 = \sum_{i=1}^{2} V_{1i} V_{2i} = 0 \tag{6}$$

which is a characteristic of orthogonality.
Subsequently, we project the centered data onto the aforementioned principal components. This is done by multiplying the centered data matrix by the transposed matrix of the selected principal components:

$$X_{\text{projected}} = X_{\text{centered}} \cdot V^T \tag{7}$$

Note that V^T contains the first two principal components of the data, since V is a 2×2 matrix.
The resulting $X_{\text{projected}}$ data is a new representation of our data in terms of the principal components, and it captures the most important patterns or trends in the original data. Figure 9 visually presents the principal components for the corresponding dataset.
Finally we are going to report the amount of energy contained in each of the two components. We start with this formula:

$$\frac{1}{\text{tr}(\Sigma^2)} \sum_{i=1}^{k} \sigma_i^2 \tag{8}$$

where, from (2), $\Sigma = \begin{bmatrix} 9.94340494 & 0 \\ 0 & 0.82624201 \end{bmatrix}$ and σ_i is the i_{th} diagonal element of Σ. In (8), the variable k represents the number of principal components, and it corresponds to the components whose energy contribution is of interest.
Then, the amount of energy contained in the first component is given by:

$$\left(\frac{1}{\text{tr}(\Sigma^2)} \sigma_1^2 \right) \times 100 = 99.31\% \tag{9}$$

Similarly, for the second component, the energy contribution is:

$$\left(\frac{1}{\text{tr}(\Sigma^2)} \sigma_2^2 \right) \times 100 = 0.69\% \tag{10}$$

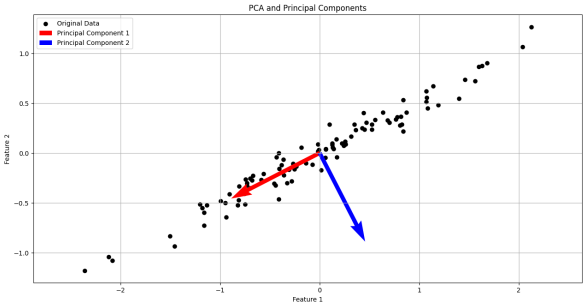


Figure 9: Data and its Principal Components

PCA of an image

In this task we analyse an image of a raccoon (Figure 10)from "datasets" from "scipy" library. The image is read in gray-scale and then resized to (249 x 185) pixels for further analysis. The underlying PCA algorithm implemented is identical to that described above, with some modifications.

The code is in provided Github repository. A function called **visualize_images()**, takes in the outputs from the SVD function and generates a composite plot that holds the representations of the original image with respect to the defined number of principal components. It uses the function **reconstruct_image()** to

Figure 10: Original racoon image

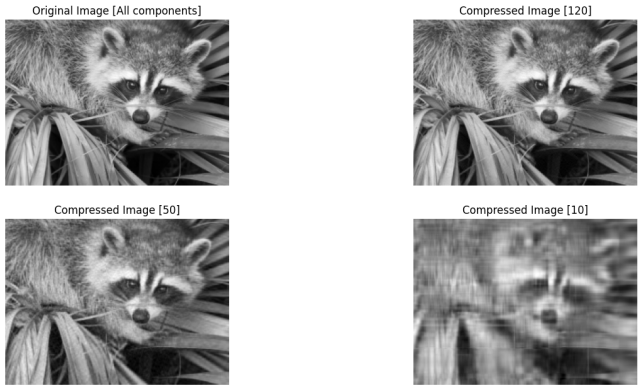


Figure 11: Representations for different principal components

regenerate each of the individual images based on the number of principal components under consideration. The output is as follows,

From the composite plot we can see that there is almost no difference for the first two plots (reconstructed with all and 120 components, respectively) meaning we can compress the image to 120 components without any visible loss of data. However, at 50 principal components the data compression becomes apparent. The image is still legible but it visibly grainy and there is somewhat of a loss of information with respect to the original full image. At 10 principal components, there is considerable loss of information with respect to the original image. Only some facial features can be made out, the rest of the image is very blurry. So in conclusion at 50 principal components there is slight loss of information but the image is still legible, but at **10** principal there is severe loss of information.

The function **calculate_energy_loss**, takes in the singular values as input and to calculate at what number of principal components is the loss in information less than 1% i.e., upto 99% information is conserved. We calculate the total energy and the cumulative energy and compare at what index does it go above our energy threshold of 99% or 0.99 for this subtask and return that indice. Adding 1 to this will give us the number of principal components required to conserve 99% of information. Finally, we print our results and plot12 a graph of cumulative energy v.s the number of components being used to calculate that cumulative energy as follows,

Hence, we can conclude that by considering **26** components, we conserve more than 99% (99.03%) of the information.

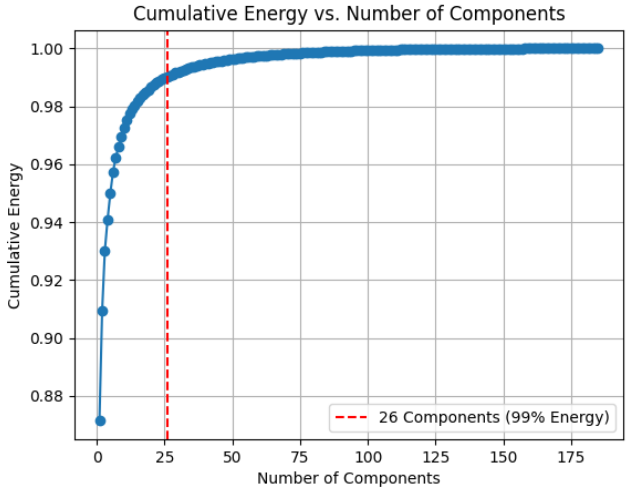


Figure 12: Energy v.s. Components

PCA of Pedestrian Trajectory

For this task, we analyse the trajectories of 15 pedestrians for 1000 time-steps. After reading the given file, we can plot the first 2 pedestrians as follows13, It can be inferred from the plot that the movement of the two

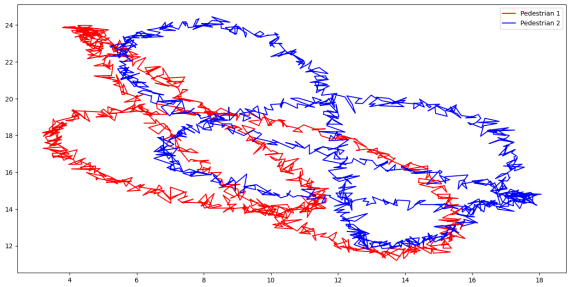


Figure 13: Trajectories of the first 2 pedestrians

pedestrians is intricate, and although their general trajectory is discernible, there is significant local variance in their path. To analyze the dataset, the algorithm employed in sub-task 2 was utilized, but with the energy threshold adjusted to 90% or 0.9. As the data is two-dimensional, the first two principal components and their respective energies were calculated, resulting in 47.33% and 37.59% conservation, respectively. Therefore, using only the first two principal components, approximately 84% of the information can be retained. This is due to the complexity of the relationship between the pedestrians and their coordinates, which cannot be fully captured by the two dimensions that account for the most variance.

To address this issue, a third principal component was included, which captures an additional 14.79% of the energy. This results in almost 100% conservation of the total energy, with the fourth and subsequent principal components each accounting for less than 0.5% of the energy. Thus, due to the multivariate nature of the dataset, three principal components are necessary to capture the data almost entirely.

Diffusion Maps

Next task is about diffusion maps. A detailed task is described in algorithm 2

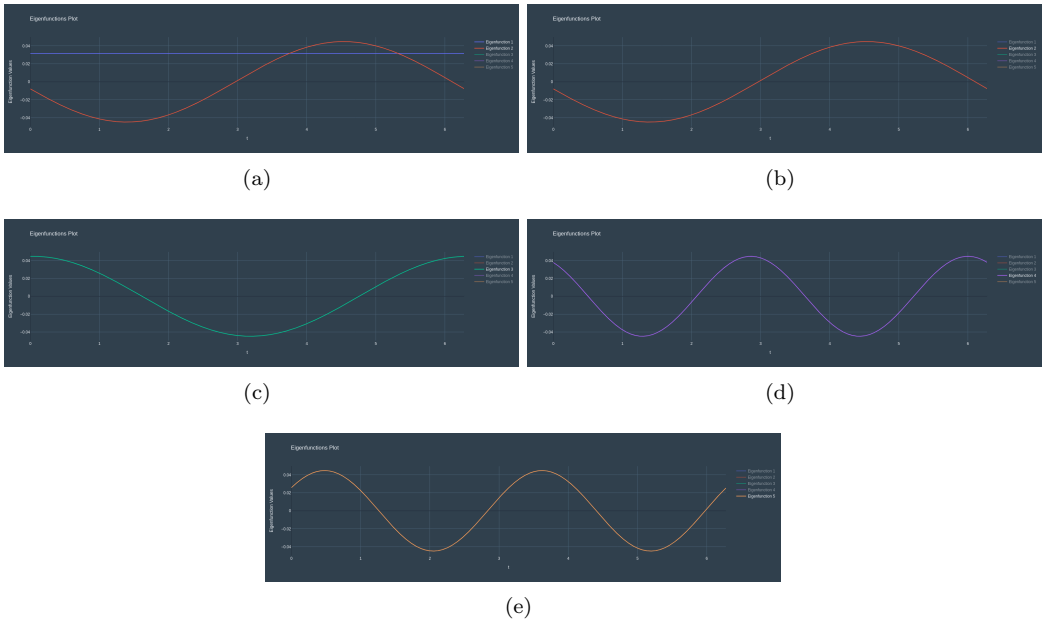
The Diffusion Map algorithm was customly implemented, in the language of choice, using only basic library support. Existing algorithms for Diffusion Maps were not used.

Algorithm 2 Demonstrating similarity of Diffusion Maps and Fourier analysis

1: Let $X = x_k \in \mathbb{R}^2 k = 1^N$ be a periodic data set with $N = 1000$ points given by $x_k = (\cos(t_k), \sin(t_k))$ where $t_k = \frac{2\pi k}{N+1}$.
2: Compute the diffusion matrix K using the Gaussian kernel $K(x_i, x_j) = \exp\left(-\frac{|x_i - x_j|^2}{\epsilon}\right)$, where ϵ is a suitable bandwidth parameter.
3: Compute the diagonal matrix D with entries $Dii = \sum_{j=1}^{N} K(x_i, x_j)$.
4: Compute the symmetric matrix $L = D^{-1/2} K D^{-1/2}$.
5: Compute the top m eigenvectors $\phi_1, \phi_2, \ldots, \phi_m$ of L associated with the largest eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_m$.
6: Plot the values of the eigenfunctions $\phi_l(x_k)$ against t_k for $l = 1, 2, \ldots, m$.

Diffusion Maps are a set of functions that project data points onto the eigenfunctions of the Laplace-Beltrami operator, which characterizes the manifold shaped by the data. The foundational principles of this algorithm and the Laplace-Beltrami operator are documented in various sources [CL06]. Conceptually, Diffusion Maps share commonalities with Principal Component Analysis, and this resemblance extends to kernel PCA as well. The core concept behind Diffusion Maps is to express each data point using the coordinates of the basis functions defined in the function space associated with the data, rather than its original coordinates. Similar to PCA, an eigendecomposition is performed within this function space to identify the most suitable basis. The basis functions identified by Diffusion Maps are the eigenfunctions ϕ_k, which map from the manifold M to the real numbers, satisfying the equation $\Delta\phi_k = \lambda_k\phi_k, \quad k \in \mathbb{N}$ for k being a natural number. An important aspect of the Diffusion Map approach is the algorithmic exclusion of the data points' sampling density from the determination of the optimal basis, a step not commonly taken in PCA, though certain adaptations like robust PCA and outlier detection methods attempt to address a related issue.

The dataset is composed of 1000 points. Five eigenvectors associated with the five largest eigenvalues are plotted in the **Figure 14**.



(a) (b) (c) (d) (e)

Figure 14: Plots of Different eigenvectors as a function of time

In the **Figure 14**, the first plot consists of eigenvectors 1 and 2 together and other plots represent different eigenfuntions. It can be observed from the figure that the first eigenfunction is constant and other perform a sinusoidal pattern with changing frequencies.

Relation to Fourier Analysis: The relationship between Diffusion Maps and Fourier analysis in this context is quite interesting. Fourier analysis decomposes a signal into its constituent frequencies, revealing periodicities and the fundamental frequency components of the signal. In our case, the eigenfunctions from the Diffusion Map, particularly those corresponding to the largest eigenvalues, show a pattern akin to Fourier modes. These eigenfunctions represent the principal modes of variation in the dataset, analogous to how Fourier modes capture fundamental frequencies. The periodicity in the dataset is effectively captured by these eigenfunctions, much like how a Fourier series would decompose a periodic signal into its constituent sine and cosine components. This demonstrates a conceptual similarity between the two methods in analyzing periodic structures within data.

The next topic is about Chaotic dynamics.

Learning dynamical systems

Chaotic dynamics in dynamical systems

In this part we study qualitative changes of dynamical systems over changes of their parameters. These changes in the qualitative behavior of the system are called bifurcations. We will explore the capacity of dynamical systems to display highly unpredictable behavior. Moreover, we will investigate how adjustments to their parameters can induce substantial changes in their overall dynamics.

We start by considering the following discrete map:

$$x_{n+1} = rx_n(1 - x_n), \quad n \in \mathbb{N} \tag{11}$$

Here r and x are parameters, where $r \in (0, 4]$ and$x \in [0, 1]$. We proceed with performing bifurcation analysis on the given discrete map 11.

- Varying r from 0 to 2:
  For r values close to 0, the system quickly converges to 0, which is a stable fixed point. As r increases, a non-zero stable fixed point emerges and moves towards 1. This corresponds to the system settling into a steady state different from zero. As r approaches 2, the steady state value increases, reaching a maximum near $r = 2$. Throughout this range of r, the system does not display chaotic behavior or complex bifurcations like period doubling. Instead, it shows a transition from a zero steady state to a non-zero steady state as r increases. The exact numerical values of the steady states for different r values can be observed in the Figure 15, where the vertical axis represents the steady state values for each corresponding r value on the horizontal axis. We see that for $r \in [0, 1], x_n$ converges to 0. We also notice that for $r \in [1, 2], x_n$ converges to a stable state $x^* = \frac{r}{r-1}$.

- Varying r from 2 to 4:
  As we shift r from 2 to 4,the behavior of the system becomes increasingly complex, as seen in the bifurcation diagram in the Figure 16: Until $r < 3$, x_n converges to a stable state $x^* = \frac{r}{r-1}$. At $r = 3$, however, the first period-doubling bifurcation occurs. The stable fixed point becomes unstable, and the system starts to oscillate between two values. This marks the beginning of a period-doubling route to chaos. More specifically, for $3 \leq r1 + \sqrt{6} \approx 3.44949$, x_n converges to a permanent oscillation between two values $x^{(1)}$ and $x^{(2)}$. As per [**?**], the way $x^{(1)}$ and $x^{(2)}$ are depending on r is given as:

$$x^{(1)} = \frac{1}{2r}(r + 1 + \sqrt{(r - 3)(r + 1)}) \quad \text{and} \quad x^{(2)} = \frac{1}{2r}(r + 1 - \sqrt{(r - 3)(r + 1)}) \tag{12}$$

  As r increases, oscillations among sets of 4 values, then 8, 16, 32, and so on, become evident. For $3.44949 \lessapprox r \lessapprox 3.54409$, nearly all initial conditions will lead the x_n to converge to permanent oscillations between 4 distinct values (for our terminology: $x^{(1)}, x^{(3)}, x^{(3)}, x^{(4)}$). Similarly, for $3.54409 \lessapprox r \lessapprox 3.56995$, for almost all possible initial conditions, the x_n converges to a permanent oscillation between 8 values, 16 values, 32 values, and so forth (until $r \approx 3.56995$). The parameter intervals associated with oscillations of a specific length decrease rapidly. The ratio between the lengths of two consecutive bifurcation intervals converges towards the Feigenbaum constant $\sigma \approx 4.66920$.

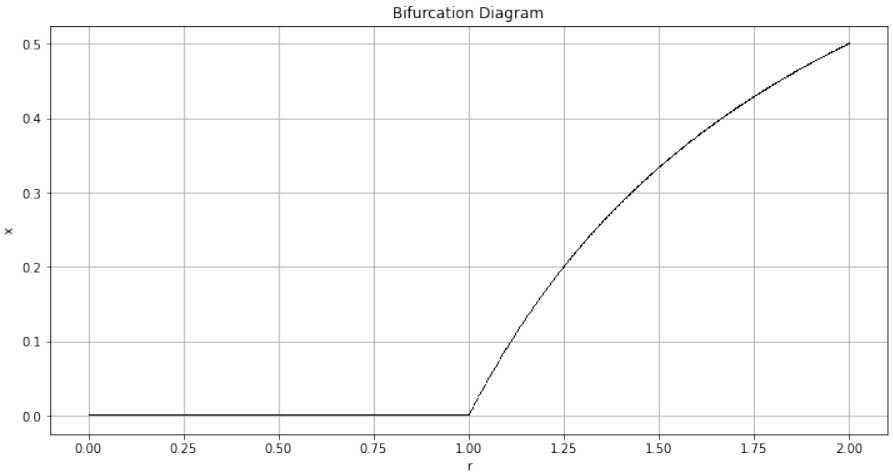


Figure 15: Bifurcation analysis of the discrete map 11, varying r from 0 to 2

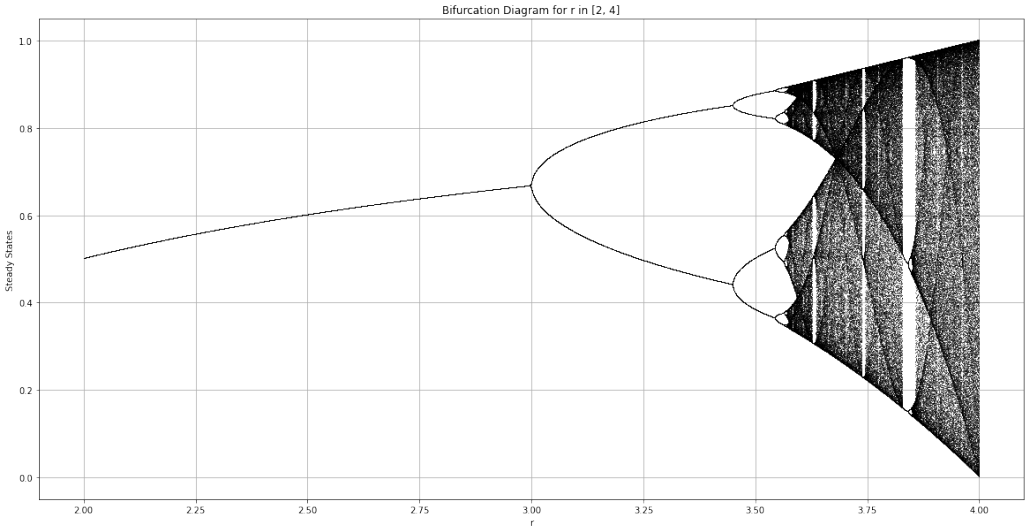


Figure 16: Bifurcation analysis of the discrete map, varying r from 2 to 4

At approximately $r \approx 3.56995$, we encounter the start of chaos, concluding the sequence of period-doubling bifurcations. From this point forward, for nearly all initial conditions, the system ceases to exhibit oscillations with a finite period. Instead, minor differences in starting values can lead to vastly divergent behaviors as time progresses, which is a defining feature of chaotic systems. Beyond this value of r, while chaotic behavior is predominant, there are still specific narrow ranges of r where the system reverts to regular, non-chaotic oscillations, known as 'islands of stability.' For example, starting at approximately $r = 1 + \sqrt{8} \approx 3/82843$ [**?**], there is a parameter range that displays oscillations among three distinct values. As r increases incrementally from this value, the oscillations double to six, then twelve, and so forth. Specifically, at $r = 1 + \sqrt{8}$, a stable period-3 cycle appears [**?**]. The transition from chaotic to periodic behavior as r varies within the range of approximately 3.56995 to 3.82843 is sometimes referred to as the Pomeau–Manneville scenario, marked by phases of regular, predictable behavior punctuated by sudden bursts of irregular, aperiodic activity.
We aimed to analyze the overall trends in the bifurcation diagram as the parameter r varied from 0 to 4. For a more detailed visual representation, refer to Figure 17.

As our next objective, we will explore and examine the limitations of dynamical systems in continuous time to exhibit chaotic dynamics when the dimension of the state space is less than three. A well-known illustration of such a system is the Lorenz attractor, situated in a three-dimensional space, giving rise to a strange attractor—a

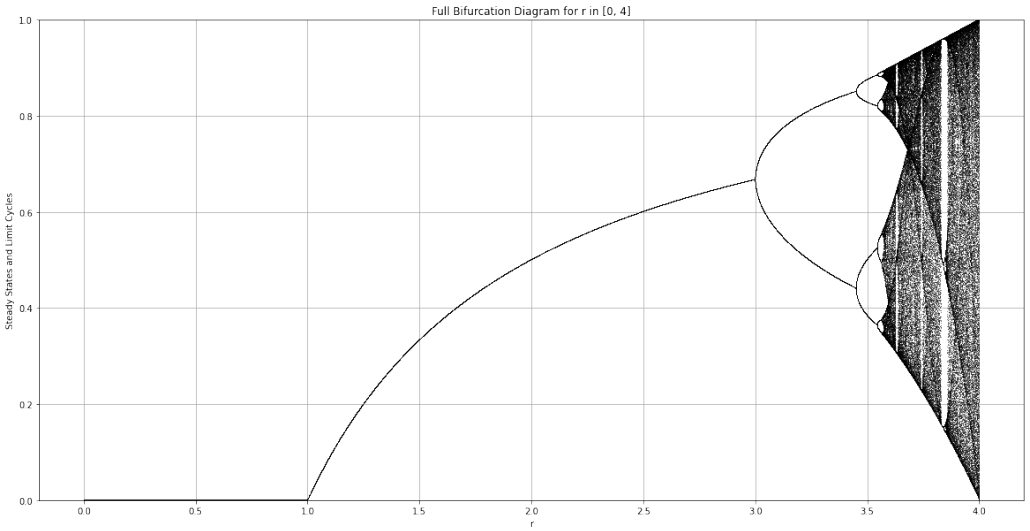


Figure 17: Bifurcation analysis of the discrete map 4, varying r from 0 to 4

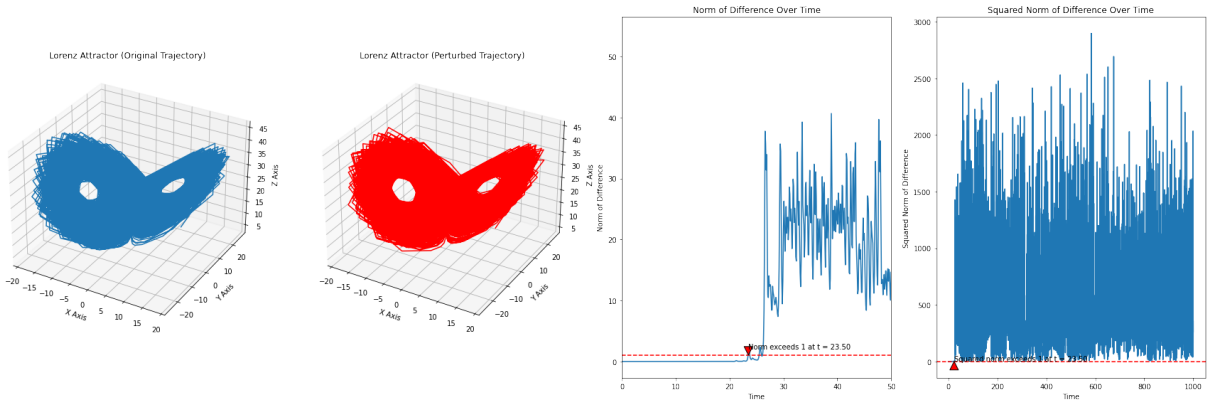


Figure 18: From left to right: Lorenz Attractor (original trajectory). Lorenz Attractor (perturbed trajectory). Zoomed in norm of difference over time. Squared norm of difference over time.

fractal set characterized by chaotic dynamics. For our exploration, we consider visualizing a single trajectory of the Lorenz system, taking initial point at $x_0 = (10, 10, 10)$, and extending until the end time $T_{\text{end}} = 1000$ with parameter values $\sigma = 10$, $\beta = \frac{8}{3}$, and $\rho = 28$. In order to capture the phenomenon of chaos by showing how the effect of the small perturbations in the initial condition will grow larger at an exponential rate, we consider another 'perturbed' trajectory with a 'perturbed' initial point $\hat{x_0} = (10 + 10^{-8}, 10, 10)$. To quantitatively assess the divergence between trajectories, we plot the norms of the differences over time. Specifically, we showcase both $\|x(t) - \hat{x}(t)\|$ and $\|x(t) - \hat{x}(t)\|^2$. While the squared norm is presented to meet task requirements, we also display the norm itself on a magnified scale. This enables a clearer observation of instances where the disparity between points on the trajectory exceeds 1 and, more broadly, undergoes exponential escalation. Figure 18 encapsulates these nuances. We observe that the (squared) norm of the difference between the trajectories exceeds 1 at t = 23.50.

As a last step, we change the parameter ρ to the value 0.5 and again compute and plot the two trajectories to observe any potential difference in terms of the sensitivity to the initial conditions. Figure 19 demonstrates the contrast between the resulting trajectories. Namely, when the parameter ρ is changed to 0.5, the dynamics of the Lorenz system and its sensitivity to initial conditions change significantly compared to when ρ is 28. The trajectories for the original and perturbed initial conditions with $\rho = 0.5$ are much closer together compared to those with $\rho = 28$. This is evident in the plots where the two trajectories (original and perturbed) for $\rho = 0.5$ nearly overlap, indicating less sensitivity to initial conditions. As ρ increases from 0.5 to 28, the system transitions from a regime with low sensitivity to initial conditions to a chaotic regime, indicating a series of

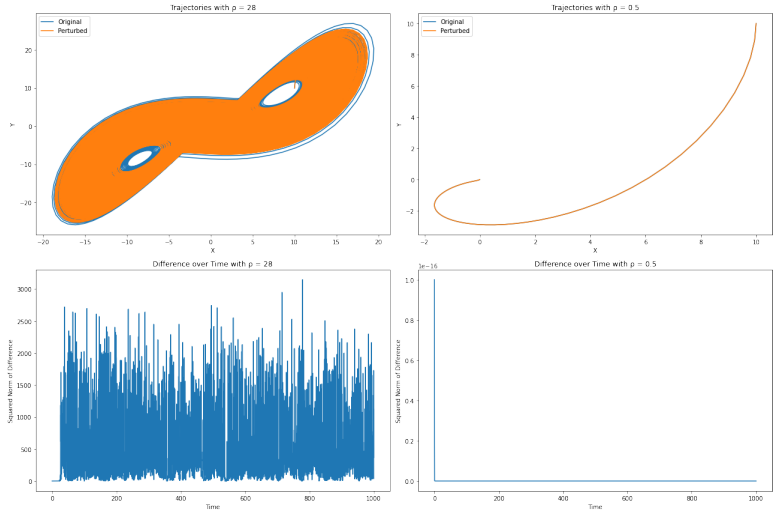


Figure 19: In reading order: Trajectories with $\rho = 28$.Trajectories with $\rho = 0.5$. Squared Norm of trajectory differences over time with $\rho = 28$. Squared Norm of trajectory differences over time with $\rho = 0.5$

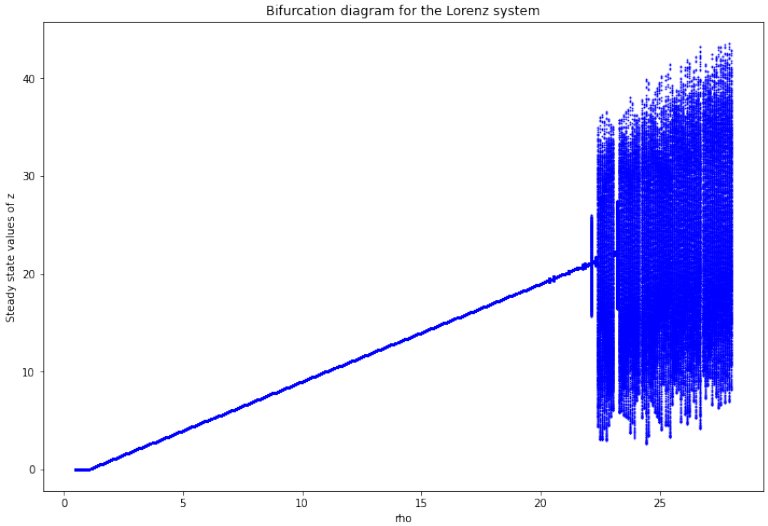


Figure 20: Bifurcation diagram of the Lorenz system

bifurcations. This is a characteristic feature of nonlinear dynamical systems like the Lorenz system, where varying a parameter can lead to significant changes in the system's dynamics, including the onset of chaos. To answer the question " Is there a bifurcation (or multiple ones) between the value 0.5 and 28?", we can create a bifurcation diagram by varying the parameter ρ in the interval of 0.5 and 28 and observe the resulting behaviour of the system. The bifurcation diagram for the Lorenz system is given in Figure 20. We can see that there are, indeed, bifurcations between ρ values of 0.5 and 28.

Function approximations

In this assignment, our objective is to demonstrate the concept of function approximation using the given examples: (A)linear function dataset and (B) nonlinear function function dataset.

We start with approximating the dataset (A) with a linear function. Let us visualize first the dataset (A) - see Figure 21. Before providing the results, let us concisely describe how are achieving a linear approximation. We frame the problem as a supervised problem with $X = \{x^{(k)}\}_{k=1}^{N} \subset \mathbb{R}^n$, and function values $F = \{f_{x^{(k)}}\}_{k=1}^{N} \subset \mathbb{R}^d$. Our approximated function then would be $\hat{f} : \mathbb{R}^n \to \mathbb{R}^d$ where $\min_{\hat{f}} e(\hat{f}) = \min_{\hat{f}} \|\mathbf{F} - \hat{f}(\mathbf{X})\|_2^2 = \min_A \|\mathbf{F} -$

$\mathbf{X}A^T\|_2^2$ has to be minimalised. That means we want to find the matrix A such that the sum of the square of the individual errors is minimal. Fortunately, the optimisation problem has a closed-form solution minimizing the least-squares error: $\hat{A}^T = (X^TX)^{-1}X^TF$. Note that we define the f as a linear function between two Euclidean spaces $\mathbb{R}^n, \mathbb{R}^d$ with $n, d \in \mathbb{N}$. f is a linear mapping function. Concretely, $f_{\text{linear}} : \mathbb{R}^n \to \mathbb{R}^d$, such that for $x \in \mathbb{R}^n$, $f_{\text{linear}}(x) = Ax \in \mathbb{R}^d$ for some matrix $A \in \mathbb{R}^{d \times n}$, whereas $\hat{A}^T = (X^TX)^{-1}X^TF$ yields the A we are looking for (the optimizer). We are using **scipy.linalg.lstsq** library for our purposes. See Figure 22 for the approximated function.

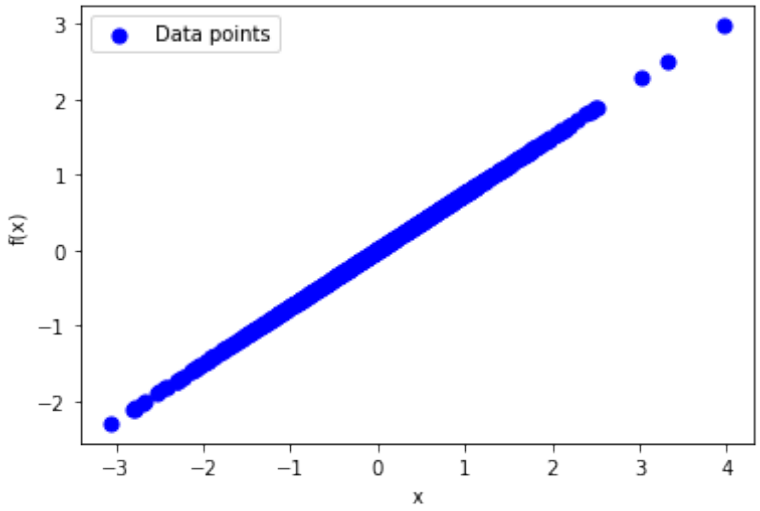


Figure 21: Linear dataset (A)

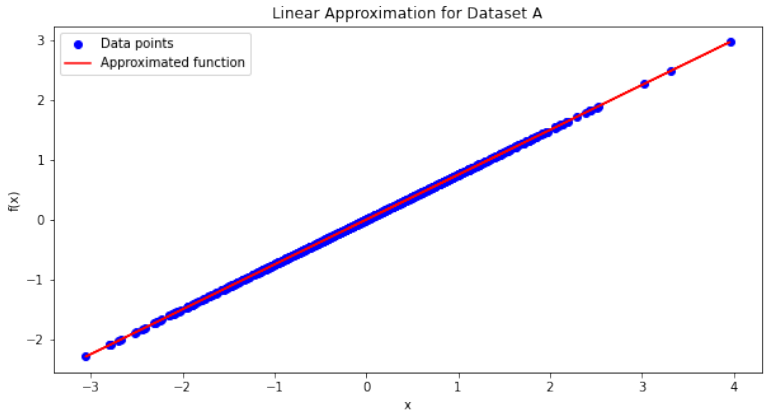


Figure 22: Linear approximation of dataset (A)

This completes the first subtask. We are also provided a non-linear dataset, which is demonstrated in Figure 23.

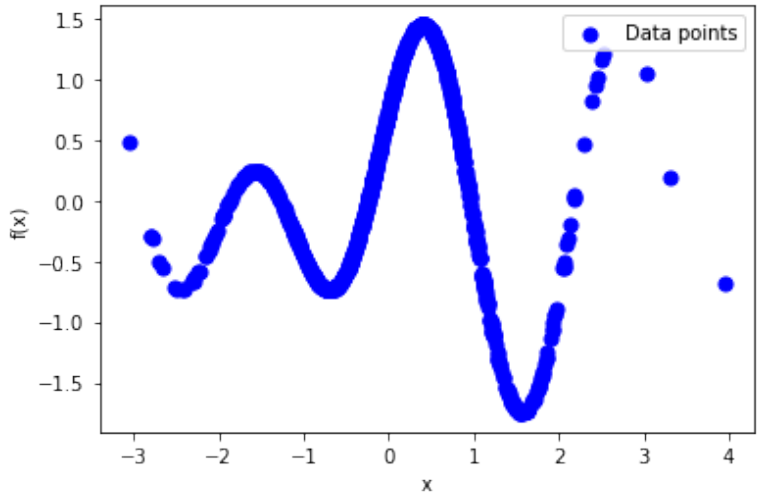


Figure 23: Non-linear dataset (B)

The second subtask requires us to use the same approach of linear approximation for this nonlinear data. As expected, and as seen in Figure 24, the linear approximation method for such highly nonlinear data is not suitable, and does not explain (approximate) the nature of the function.

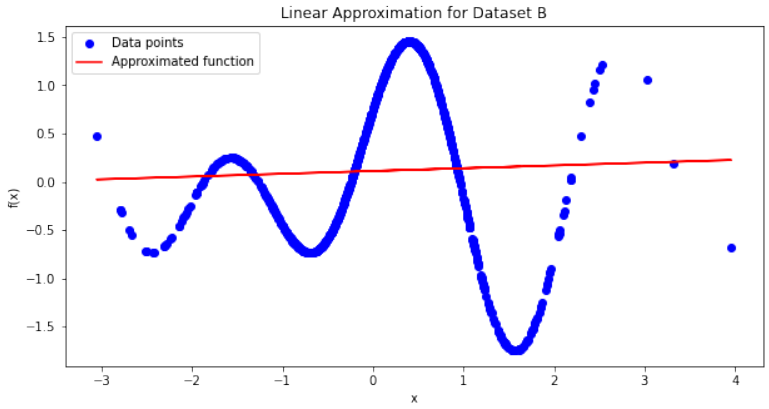


Figure 24: Non-linear dataset (B) approximated with a linear function

Hence, at this point, we need a non-linear approximation method. The core concept involves representing the unknown function f as a sum of familiar functions ϕ using the formula:

$$f(x) = \sum_{l=1}^{L} c_l \phi_l(x), \quad c_l \in \mathbb{R}^d.$$

If L is finite, this representation is limited to a finite-dimensional function space.

In this context, our focus is on specific functions ϕ, specifically radial basis functions defined as:

$$\phi_l(x) = \exp\left(-\frac{\|x_l - x\|^2}{\epsilon^2}\right).$$

We are using ϵ^2 in the denominator of the radial functions. Discussion of the correct choice of L and ϵ is worthwhile. Therefore, an essential aspect of the assignment involves explaining the procedure for selecting an appropriate combination of L and ϵ values. Before going into feature selection direction, let us quickly discuss why would it matter to find a good pair of L and ϵ. Choosing the value of the spread parameter ϵ in radial basis functions is a critical step because it controls the width of the Gaussian functions used in the approximation.

If ϵ is too small, the basis functions will be very narrow, leading to an RBF model that can potentially overfit the data by capturing noise rather than the underlying trend. Conversely, if ϵ is too large, the basis functions will be too wide, and the RBF model may underfit the data, failing to capture important variations.The choice of ϵ often involves a trade-off between bias and variance, and it is typically selected through experimentation, cross-validation, or domain knowledge. Furthermore, the choice of L (the number of radial basis functions or centers) can significantly affect the model's performance. Too few centers might not capture the complexity of the data, while too many can lead to overfitting. A common approach is to start with a number of centers that is less than the number of data points and increase it based on the model's performance.

Strategies to select an appropriate L and ϵ pair would include:

- Visually inspecting the data to get a ballpark estimate of the spread. This can give us an initial guess for L and ϵ values. However, this sounds like a very qualitative approach, although easy to do and practical.
- Performing a grid search over a range of L and ϵ values, and evaluate the performance of the RBF model on a validation set or using cross-validation. This is a more robust option.

We implement a simple grid search approach to find appropriate L and ϵ values. The grid intervals for L and ϵ are, $[0, 100]$ and $[0.01, 10]$, respectively. The log of the grid search results is a verbose file and we are not including the details to this report. However, all relevant (and reproducible) content exists in the code. As an output of the grid search, the best L is 60 and the best ϵ is 0.46415888336127775. Using these values in our non-linear function approximation implementation, we get a nice non-linear approximation result as in Figure 25.

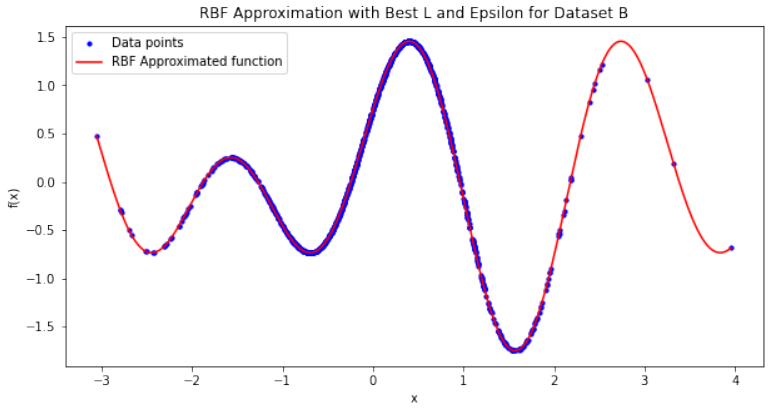


Figure 25: Non-linear dataset (B) approximated with a non-linear function

References

[BKL+09] U. Brunner, H. Kirchberger, C. Lebeda, M. Oswald, R. Könnecke, M. Kraft, A. Thoss, L. Mülli, A. Seyfried, C. Hartnack, S. Wader, G. Spennes, T. Kretz, M. Schwendimann, N. Waldau, P. Gattermann, C. Moroge, T. Meyer-König, and M. Schreckenberg. *RiMEA–Richtlinie für Mikroskopische Entfluchtungs-Analysen*. 2.2.0 edition, 2009. eprint from `http://www.rimea.de/`.

[Boc04] Nino. Boccara. *Modeling Complex Systems*. Graduate Texts in Contemporary Physics,. Springer New York, New York, NY, 2004.

[CL06] Ronald R. Coifman and Stéphane Lafon. Diffusion maps. *Applied and Computational Harmonic Analysis*, 21(1):5–30, 2006. Special Issue: Diffusion Maps and Wavelets.

[Gar70] Martin Gardner. Mathematical games. *Scientific American*, 223(4):120–123, 1970.

[Wol83] Stephen Wolfram. Statistical mechanics of cellular automata. *Rev. Mod. Phys.*, 55:601–644, Jul 1983.

[Wol84] Stephen Wolfram. Cellular automata as models of complexity. *Nature*, 311(5985):419–424, Oct 1984.