Report for exercise 3 from group B

Tasks addressed:	4
Authors:	HORIA TURCUMAN (03752553)
	GEORGI HRUSANOV (03714895)
	UPPILI SRINIVASAN (03734253)
Last compiled:	2022–12–08
Source code:	https://gitlab.com/turcumanhoria/crowd-modeling/-/tree/main/Exercise%203

The work on tasks was divided in the following way:

HORIA TURCUMAN (03752553)	Task 1	33.3%
	Task 2	33.3%
	Task 3	33.3%
	Task 4	33.3%
GEORGI HRUSANOV (03714895)	Task 1	33.3%
	Task 2	33.3%
	Task 3	33.3%
	Task 4	33.3%
UPPILI SRINIVASAN (03734253)	Task 1	33.3%
	Task 2	33.3%
	Task 3	33.3%
	Task 4	33.3%

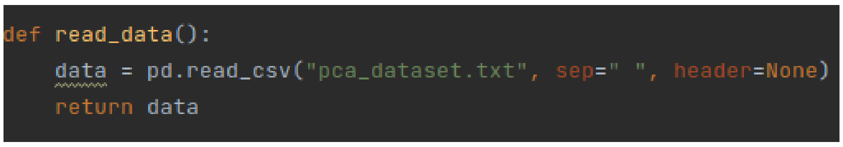
Report on task Principal Component Analysis

Principal component analysis, also known as PCA, is a method of dimensionality reduction that is frequently used to reduce the dimensionality of large data sets. This is accomplished by transforming a large set of variables into a smaller set of variables that still retains the majority of the information that was contained in the large set.

When you decrease the number of variables in a data collection, accuracy will inevitably suffer as a result. However, the key to successful dimensionality reduction is to sacrifice some accuracy in exchange for increased simplicity. Because smaller data sets are simpler to examine and visualize, and because they make it more simpler and quicker for machine learning algorithms to analyze data because they don't have to take into account as many unnecessary factors.

To summarize, the primary objective of principal component analysis (PCA) is to lessen the number of variables in a data collection while maintaining as much information as is practicable.

The very first thing we had to do was to create a method that has to read the two-dimensional data set `pca_dataset.txt` on Moodle that is optimal in the sense of variance reduction. This is done with the help of the `read_data` method as you can see on Figure 1. This method reads the data from the file and returns it. In order to be able to read the data the `Pandas` library of Python was used and when the data is read, it is stored in a new Pandas `DataFrame` and returned. Throughout the first exercise the data, we will be working with is a Pandas `DataFrame`.



```python
def read_data():
    data = pd.read_csv("pca_dataset.txt", sep=" ", header=None)
    return data
```

Figure 1: The read_data method for reading the Data

In Figure 2 you can see the two dimensional data of `pca_dataset.txt` visualized using a scatter plot.

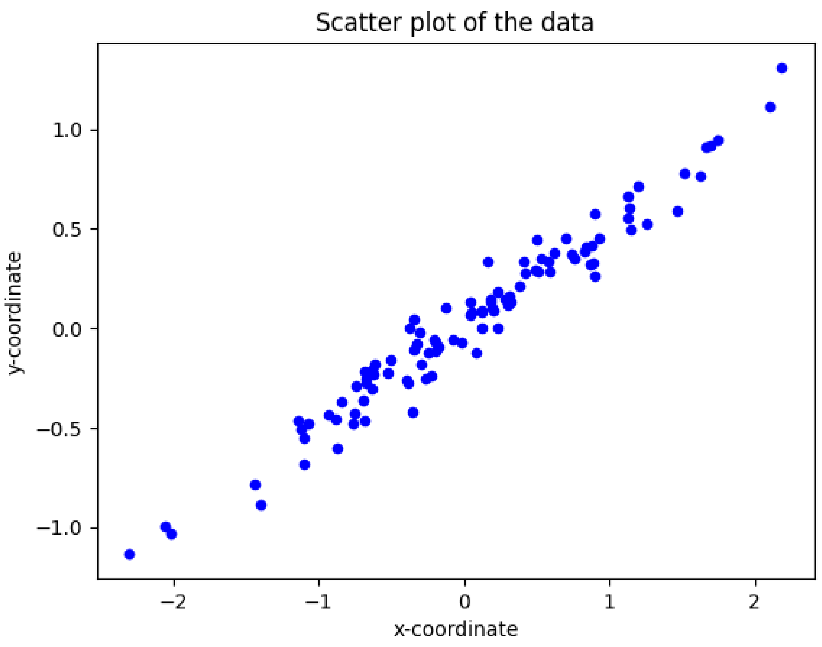


Figure 2: Visualization of the two dimensional Data using a Scatter Plot

We standardize/normalize data before doing PCA.

In most cases, the goal of normalization is to bring all of the features to the same scale. For instance, our dataset for predicting house prices contains a number of diverse variables. While the price is expressed in dollars, the area is expressed in square feet. Because it has a larger numerical value, the technique will now give more weight to the numerical value that is greater. As a result, we must keep all of the features within the same range in order to normalize them.

When doing mean centering, attention is made to ensure that the first Principal Component is oriented in the direction of the variable with the greatest degree of variation.

Mean centering will be performed by first eliminating the mean from all channels or features. On Figure 3 we could observe the difference between the orginal data and after applying mean centering to it.

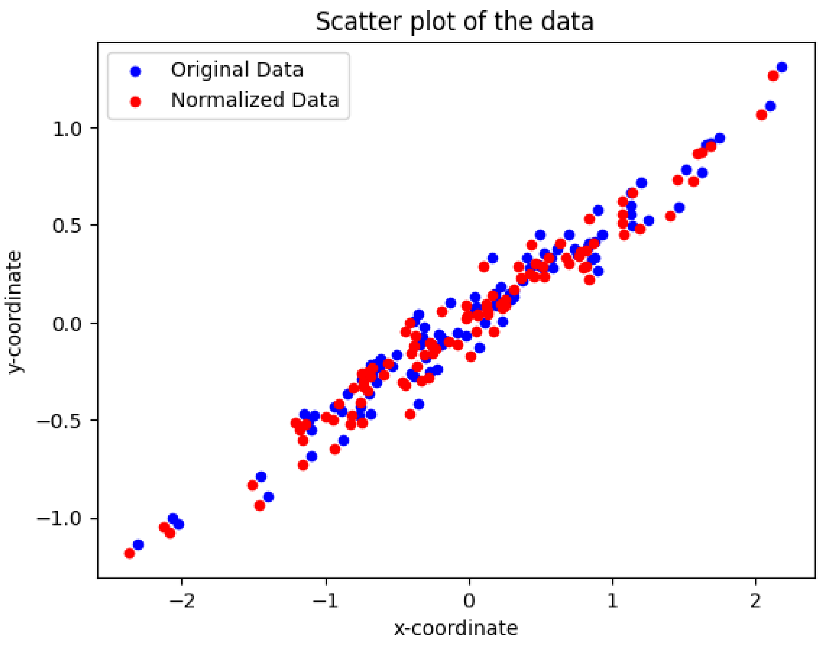
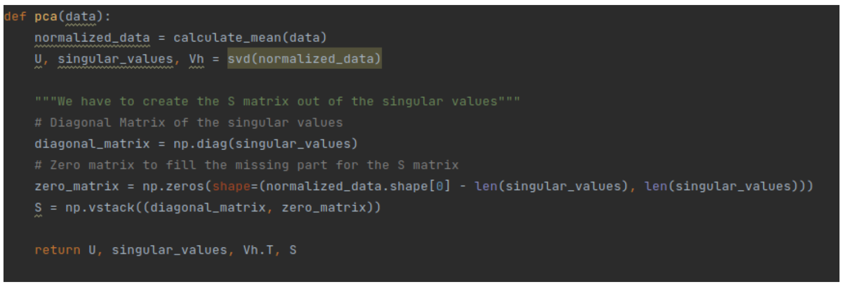


Figure 3: Comparison of the two dimensional Data before and after Mean Centering using a Scatter Plot

The next step would be to decompose the centered data matrix into singular vectors **U, V** and values **S**. This is done with the `svd` method from the `scipy.linalg` library which supports this functionality. This method takes data as a parameter and then this data is factorized into two unitary matrices **U** and **Vh**, as well as a one-dimensional array **s** of real and non-negative singular values, so that the input data would be equal to $U * S * Vh$, where S is an appropriately shaped matrix of zeros with a main diagonal s(the singular values). In order to construct the **S** matrix we create a diagonal matrix called `diagonal_matrix` which has all the singular values as its diagonal and also `zero_matrix` which is a matrix filled with zeros which is done in order to fill the missing part for the **S** matrix. In the end we return the results of the `svd` function as well as the newly created **S** matrix. This could be more specifically observed on Figure 4



```
def pca(data):
    normalized_data = calculate_mean(data)
    U, singular_values, Vh = svd(normalized_data)

    """We have to create the S matrix out of the singular values"""
    # Diagonal Matrix of the singular values
    diagonal_matrix = np.diag(singular_values)
    # Zero matrix to fill the missing part for the S matrix
    zero_matrix = np.zeros(shape=(normalized_data.shape[0] - len(singular_values), len(singular_values)))
    S = np.vstack((diagonal_matrix, zero_matrix))

    return U, singular_values, Vh.T, S
```

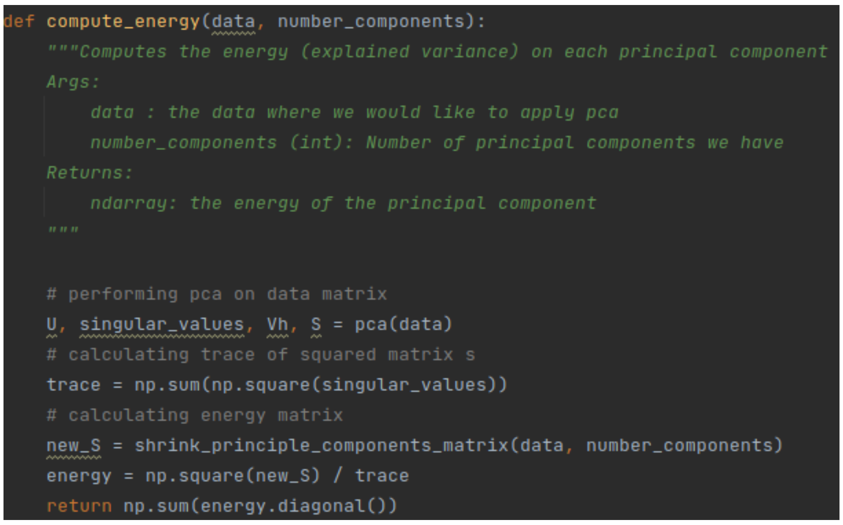
Figure 4: Implementation of the PCA, using Singular Value Decomposition

The next thing to do within the scope of the task is to be able to calculate the "Energy" of each Principal

Component. The "energy" of the ith main component, also known as the explained variance, may be found in the singular value σ_i located on the diagonal of the matrix **S**. Calculating the proportion of the total energy that can be described by using a certain number L of principle components in order to describe the data is possible thanks to the following equation:

$$\frac{1}{trace(S^2)} \sum_{i=1}^{L} \sigma_i^2$$

Trace(S^2) is equals the sum over all the squared singular values. In our implementation is done in the function `compute_energy` which takes the data we would like to reduce the dimension of as argument and also `number_components`, which stands for the number of principle components. First we call the pca method in order to perform the singular value decomposition, after that we calculate the trace, then we calculate the energy matrix and the last step would be to divide the energy matrix to the trace value. Thus we have calculated the energy. The concrete approach could be seen in Figure 5



```
def compute_energy(data, number_components):
    """Computes the energy (explained variance) on each principal component
    Args:
        data : the data where we would like to apply pca
        number_components (int): Number of principal components we have
    Returns:
        ndarray: the energy of the principal component
    """

    # performing pca on data matrix
    U, singular_values, Vh, S = pca(data)
    # calculating trace of squared matrix s
    trace = np.sum(np.square(singular_values))
    # calculating energy matrix
    new_S = shrink_principle_components_matrix(data, number_components)
    energy = np.square(new_S) / trace
    return np.sum(energy.diagonal())
```

Figure 5: Implementation of the `compute_energy` method for calculating the Energy(Explained Variance)

Now when we have all this information, we could apply the Principal Component Analysis to our first set of data. In order to be able to visualize the behaviour of the program we implemented the function `visualize_pc`. The main idea behind it is to draw the directions that represent the principal components. To do this we have to find the center of the plot and draw the corresponding line for each of the principle components. For the representation of the principle components we use `quiver` from the `matplotlib` library in Python. Quiver plots a 2D field of arrows. Both principle components could be seen in Figure 7. The colors are taken from a simple color cycler again from `matplotlib`.

When we inspect further in order to be able to derive the energy of each component. The results we get are that the first principal component contains 92.33% of the energy in the data, whereas the seconds principal component contains 7.67% of the energy in the data.

Second Part: In this session, we had to do principal component analysis on a racoon image. The picture was first saved locally. A dedicated method called `racoon_processing` was created to handle the processing of the racoon image. It loads the image and after that converts it to grayscale with the help of *ImageOps* from the `PIL` library. Once the picture was converted to garyscale it was molded to the necessary dimensions, these being (249,185). We treated the photo columns as data points and then applied principal component analysis to the data. To be more specific, we attempted to reconstruct the image using all 120, 50, and 10 primary components, with the results displayed in Figure 4. In order to do that several helping functions were created: `recondstruction` and also `shrink_principle_components_matrix`. The main idea of `shrink_principle_components_matrix` being to shrink the diagonal **S** matrix. The other helping function just takes the data from the racoon image, performs Singular Value Decomposition on it, shrinks the **S** matrix accordingly to the new number of components and reconstructs the image. The reconstructured image is then

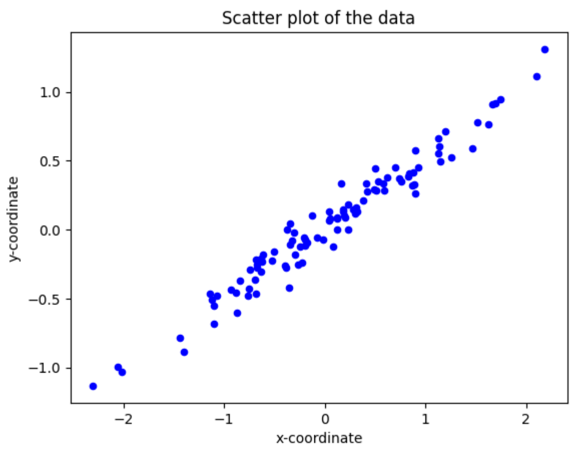


Figure 6: Plot of the data contained in `pca_dataset.txt`

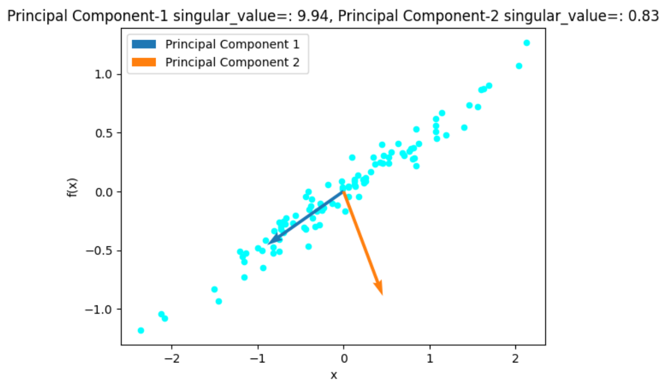


Figure 7: Plot of the centered data with the directions of the 2 principal components

converted to grayscale and displayed. Mathematically the reconstruction mechanism could be explained the following way: in order to obtain a reconstruction with a certain number L of principal components, what we have to do is just set the singular values to zero that are smaller than the *L-th singular value* and then reconstruct using

$$\hat{X} = U * \hat{S} * V^T$$

Even with only 50 primary components, there are some minor discrepancies between the original and reconstructed images. However, it is only until there are ten major components that these distinctions become clear. In reality, 120 principal components are adequate to catch 99.2% of the energy (the loss is less than 1%), 50 main components capture 93.10% of the energy, and 10 principal components collect just 75.76% of the energy. It's vital to remember that the formula in Eq was used to calculate the percentage of energy maintained by a certain number of major components.

All Figures:Fig9,Fig10,Fig11 show how does the reconstruction influence the original racoon image according to the different numbers of principle components

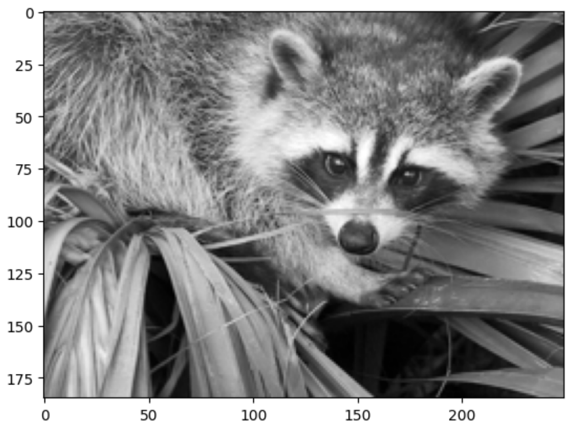


Figure 8: The Original Racoon Image

After observing the different images, we could clearly answer the question: "At what number is the information loss visible?" - this happens when the number of the principle components we would like to reduce the dimension of the image is 10.

The last question we had to answer in this subtask was "At what number is the energy lost through truncation smaller than 1%?". We tackled this task by creating the function called `find_pc`, which takes the

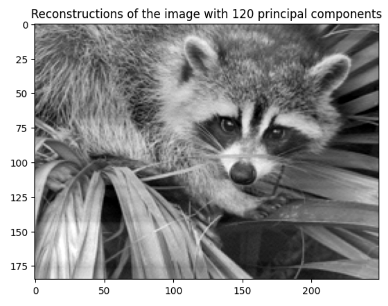


Figure 9: Racoon image reconstruction with 120 Components

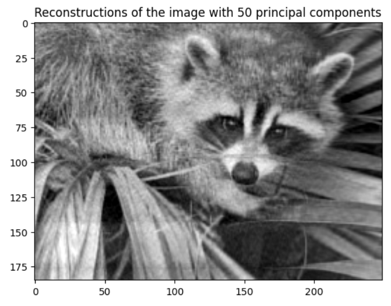


Figure 10: Racoon image reconstruction with 50 Components

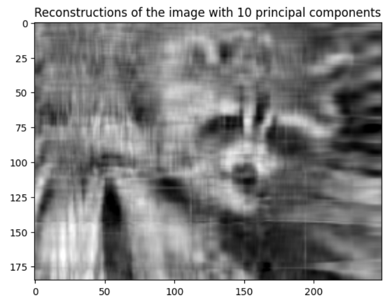


Figure 11: Racoon image reconstruction with 10 Components

data and performs reconstruction for every possible number of principle components. Here we utilize the earlier mentioned and described function `compute_energy` in order to compute the current energy and see the current energy loss. If the current energy loss is bigger than 1, this means that the last number of principle components gives us the last possible number of principle components to remain an energy loss less than 1%. In this case this is the number 86. The figure with the reconstruction using 86 principal components could be seen on Figure 12.

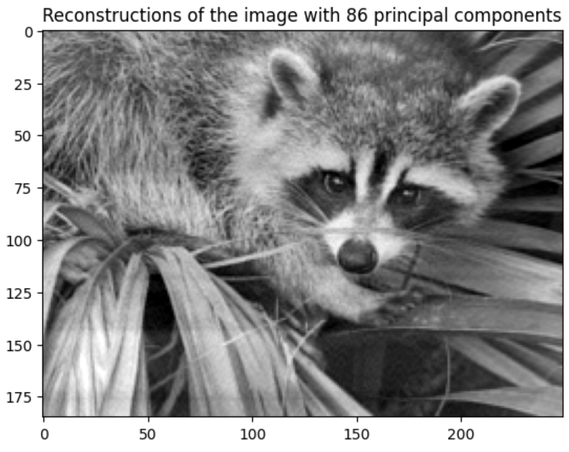


Figure 12: Racoon image reconstruction with 86 Components

As we can see when we use 86 principal components, there is basically no visible difference between the original image, the one with 120 components and the one with 86 components.

Third Part: Here we had to analyze another dataset: this time the data was containing trajectories of various pedestrians in the 2D. It is stored in the file `data_DMAP_PCA_vadere.txt`. Just like in the previous example, we created a method that will read the data and create a `DataFrame`. The first thing was to do some exploratory data analysis on the data to see how is it structured. The file contains position data of 15 pedestrians over 1000 time steps (in the form $x1, y1, x2, y2, ...$ for the x, y positions of all pedestrians). Since we had to analyze and work with the first two pedestrians, this means we had to extract from the dataset only the needed data, this being the first two columns for the first pedestrian and the next two columns for the second pedestrians. In order to make working with the data easier the column names of the newly created DataFrames was set to be ”x-coordinate” and ”y-coordinate”. Once we had done some work with the data, we visualize the data of the two pedestrians and their trajectory using a scatter plot that could be seen in Figure 13

When we carefully observe the scatter plot, we could see that some areas of the plot seem ”blurry” just like some type gaussian noise in sound for example.

The next thing was to reconstruct the data that we had using pca with two principle componnets and observe its behaviour. We again use the formula from Eq in order to reconstruct the data we will be working with. The following steps were the same: extracting the two pedestrians and reformatting the look of the DataFrame. We

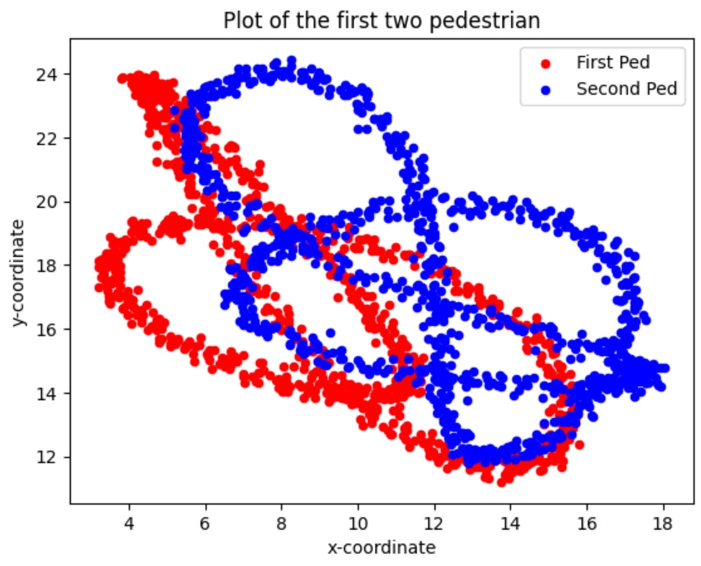


Figure 13: Trajectories of 2 pedestrians in the 2D space.

compute the energy using our `compute_energy` function and visualize the result we got. The scatter plot after the transformation and its enrgy could be seen in Figure 14

```
The energy captured by the first 2 principal components is 84.92%
```

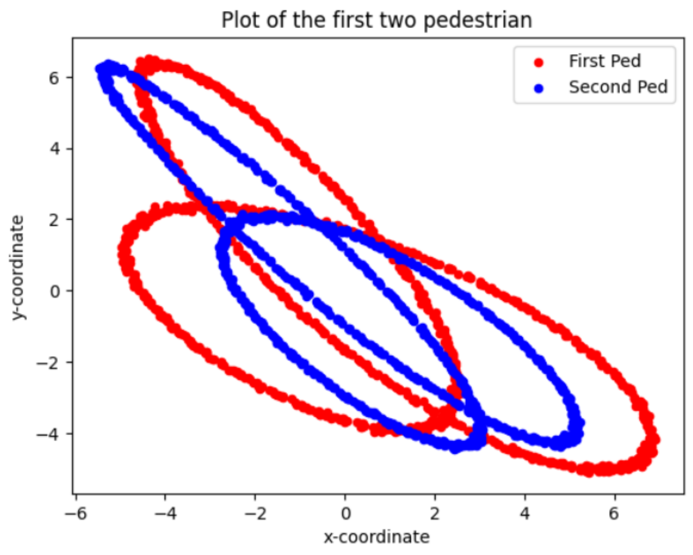


Figure 14: Trajectories of 2 pedestrians in the 2D space using 2 principal components

Using this graph we could see that the energy captured is just 84,92% percent. In order to be able to answer the question "How many do you need to capture most of the energy?". The thing we do was to check the energy for all possible number of components. The next possible option: 3 principle components delivered satisfactory results. The energy is at 99.71 % percent, which captures most of the energy and improves our previous energy by 14.79% percent. Figure 49 visualizes this observation.

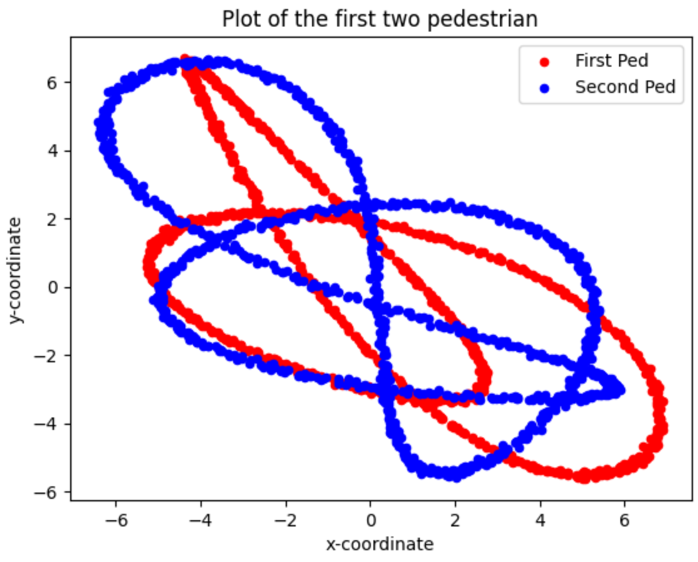


Figure 15: Trajectories of 2 pedestrians in the 2D space using 3 principal components

Now we can see that the "blurry" parts we mentioned at the begging look like sharp edges and doesn't look like some noisy part of the data.

Report on task TASK 2

Diffusion Maps are a family of functions that map data points into the set of eigenfunctions of the Laplace-Beltrami operator on a manifold describing the data. From an abstract point of view, Di usion Maps are not so different from Principal Component Analysis (that is also why they have similarity with "kernel-PCA"). The main idea in Diffusion Maps is to represent each data point not in its given coordinates, but `in coordinates of basis functions of the function space on the data.` A critical part of the Diffusion Map framework is to algorithmically remove the influence of the sampling density of the data points from the estimation of the optimal basis. This is typically not done in PCA, although some versions (robust PCA, outlier detection) try to accomplish a similar task.

The adaptation of diffusion maps is as follows:

- Form a distance matrix D with entries: This is done using the function `math.sqrt` as follows: `D[i][j] = math.sqrt(np.sum(distance ** 2)).`
- Set episolon to 5% of the diameter of the dataset: `epsilon = 0.05 * np.max(D)`
- Form the kernel matrix W : This is achieved by the function - `W=np.exp((-D**2)/epsilon)`
- Form the diagonal normalization matrix :

```
P_ii = \sum_{j=1}^N W_{ij}
    for i in range(N):
        P[i, i] = np.sum(W[i])
```

- Normalize W to form the kernel matrix: $K = P^{-1}WP^{-1}$

```
inv_P = LA.inv(P)
K = inv_P.dot(W.dot(inv_P))
```

- Form the diagonal normalization matrix $Q_{ii} = \sum_{j=1}^{N} K_{ij}$

```
for i in range(N):
   Q[i, i] = np.sum(K[i])
```

- Form the symmetric matrix $T = Q^{\frac{-1}{2}} K Q^{\frac{-1}{2}}$

```
inv_Q_sqrt = np.sqrt(LA.inv(Q))
    T = inv_Q_sqrt.dot(K.dot(inv_Q_sqrt))
```

- Find the L + 1 largest eigenvalues a_l and associated eigenvectors v_l of symmetric matrix of T – This can be done using the function `\eigsh"` from `\scipy.sparse.linalg"` library. `a, v = eigsh(T, k = L + 1)`

- Compute the eigenvalues of $T^{\frac{1}{\epsilon}}$ by $\lambda_l^2 = a_l^{\frac{1}{\epsilon}} lambda = np.sqrt(a**(1/epsilon))$

- Compute the eigenvectors ϕ_l of the matrix $T = Q^{-1}K$ by $\phi_l = Q^{\frac{-1}{2}} v_l$ ϕ_l `= inv_Q_sqrt.dot(v)`

The eigenvalues and the eigenvectors -l are the objects we are interested in.

Task 2 Part 1: Periodic dataset is as follows: $X = x_k \in R^2{}_{k=1}^N, x_k = (cos(t_k), sin(t_k)), t_k = \frac{2\pi k}{N+1}, N =$ 1000 We can visualize the dataset as follows:

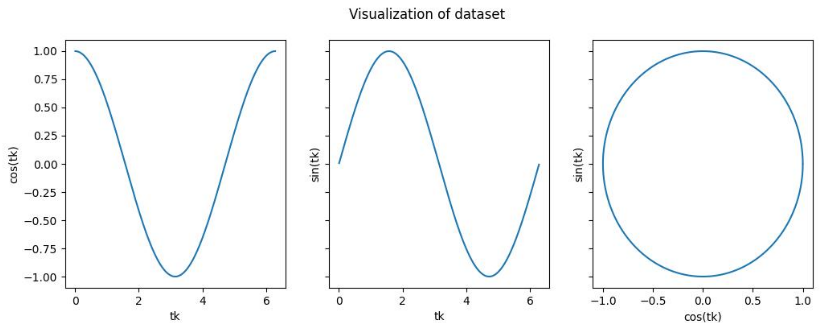


Figure 16: Visualization for task 2 part 1

Following are the graphs of eigenfunction versus t_k :

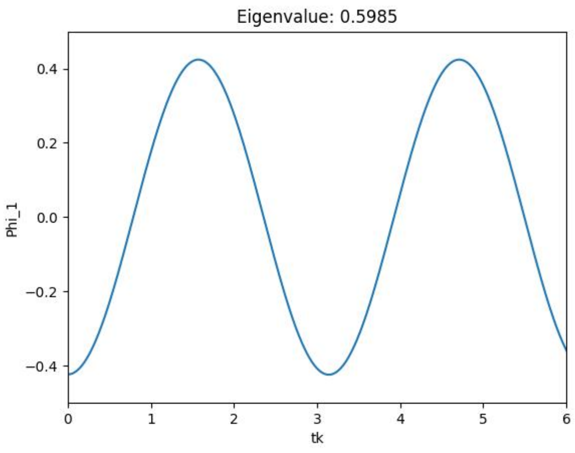


Figure 17: phi-1 vs tk

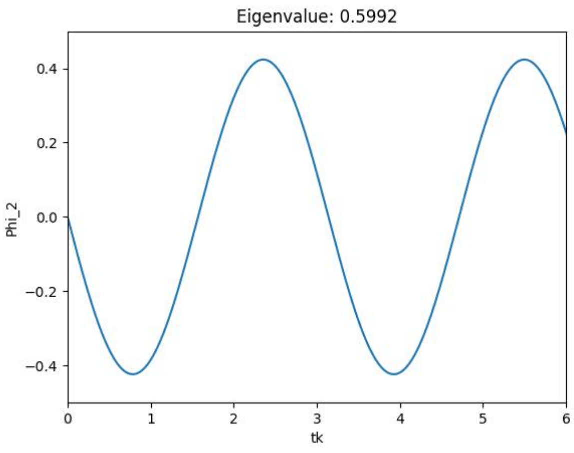


Figure 18: phi-2 vs tk

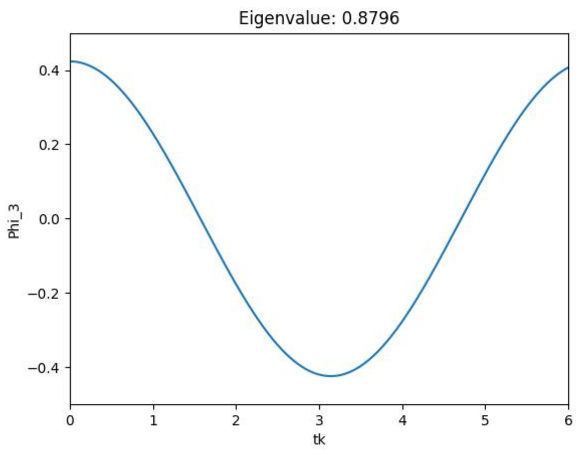


Figure 19: phi-3 vs tk

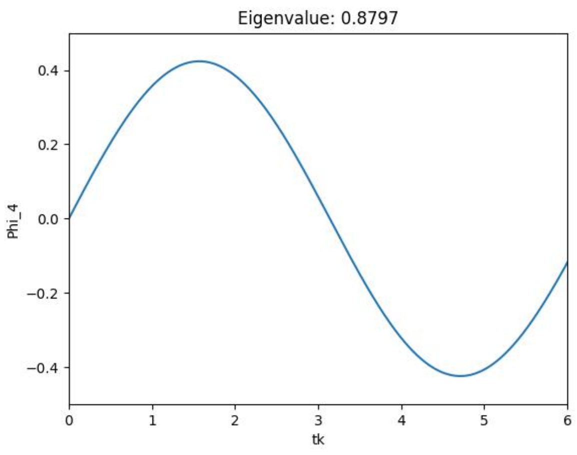


Figure 20: phi-4 vs tk

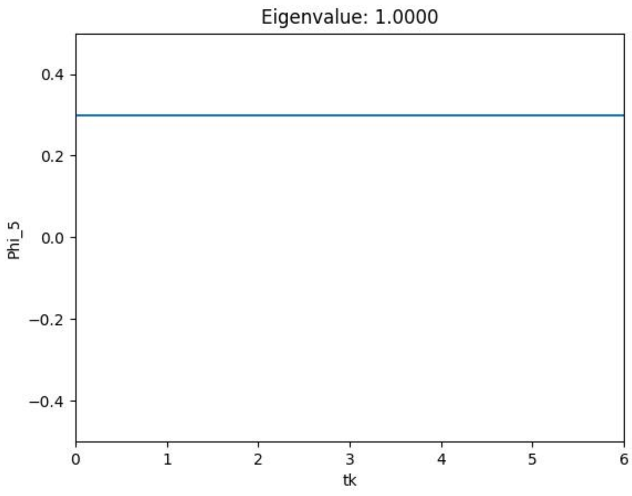


Figure 21: phi-5 vs tk

Task 2 part 2: Use the diffusion algorithm for swiss-roll manifold. Following is the function of swiss roll manifold: $X = x_k \in {R^3}_{k=1}^{N}, x_k = (ucos(u), v, usin(u)), N = 5000$ Where $(u, v) \in [0, 10]^2$ are chosen uniformly at random. Following image provides the visualization for swiss roll manifold.

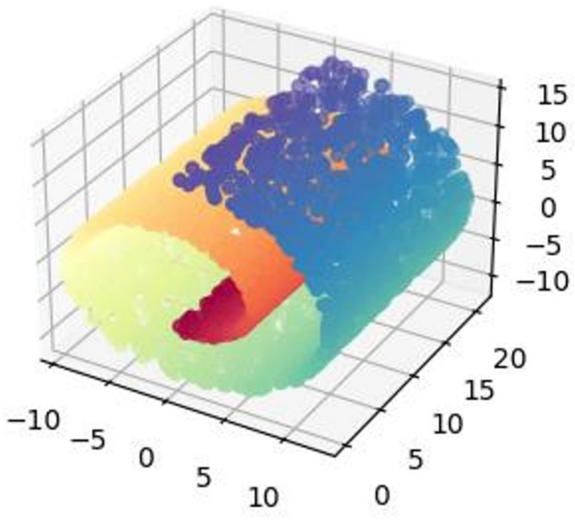


Figure 22: Visualization for swiss roll manifold

Following are the 10 eigenfunctions vs t plots:

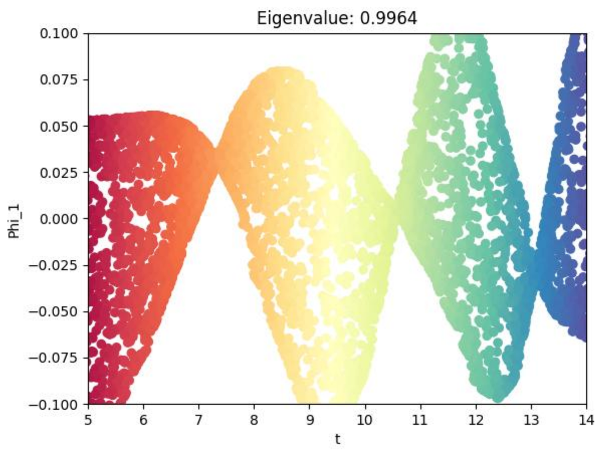


Figure 23: phi-1 vs t

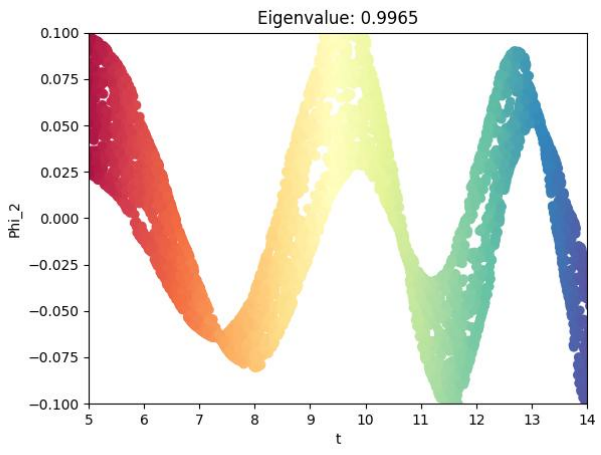


Figure 24: phi-2 vs t

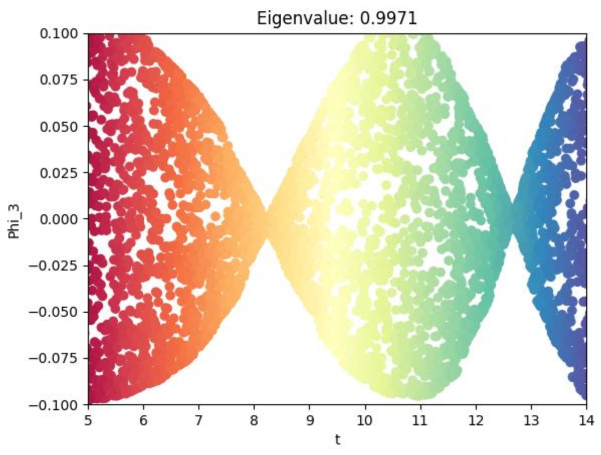


Figure 25: phi-3 vs t

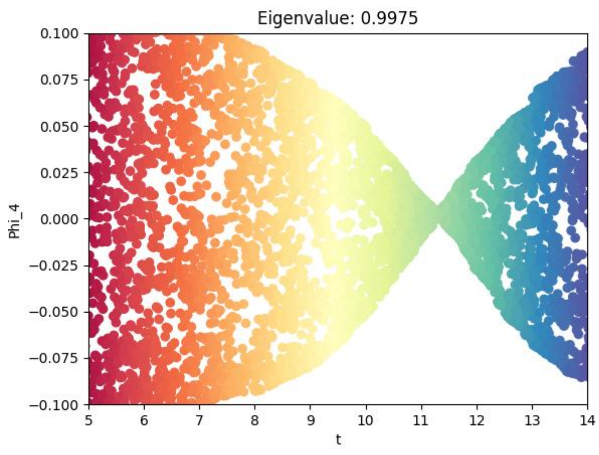


Figure 26: phi-4 vs t

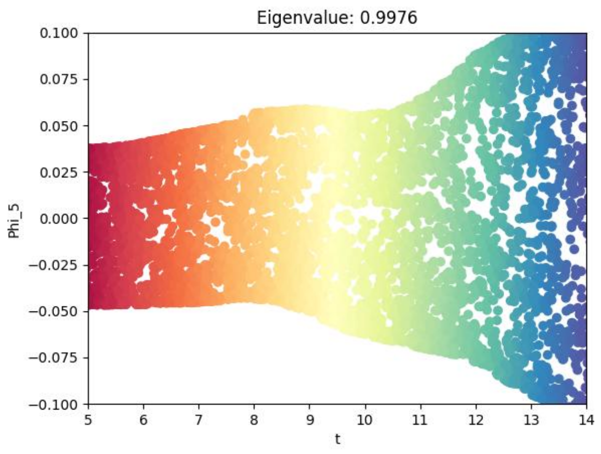


Figure 27: phi-5 vs t

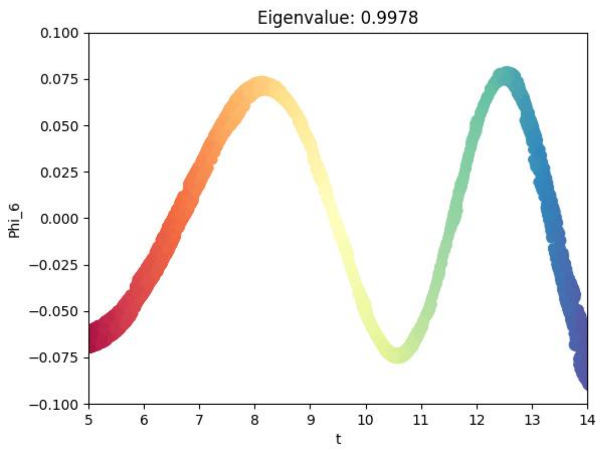


Figure 28: phi-6 vs t

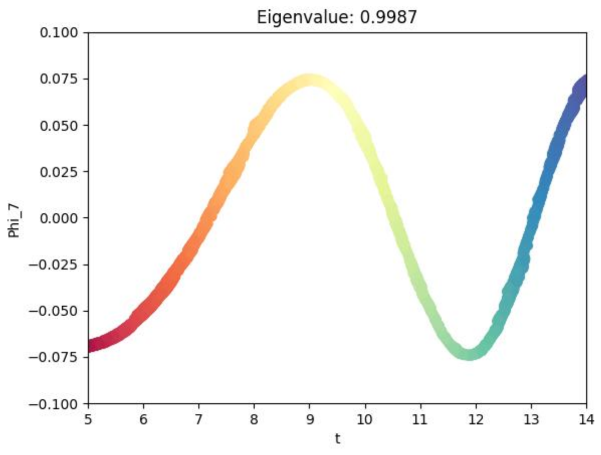


Figure 29: phi-7 vs t

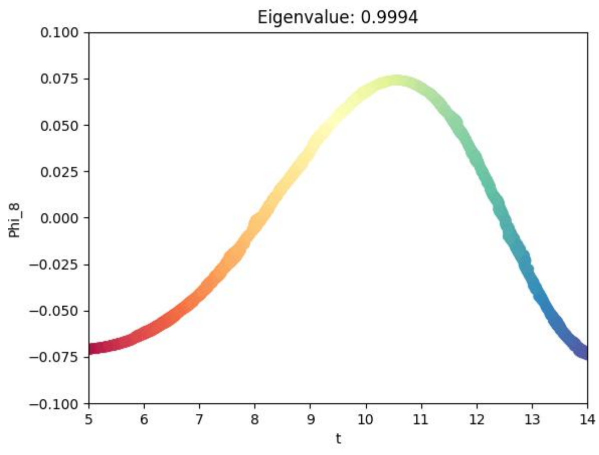


Figure 30: phi-8 vs t

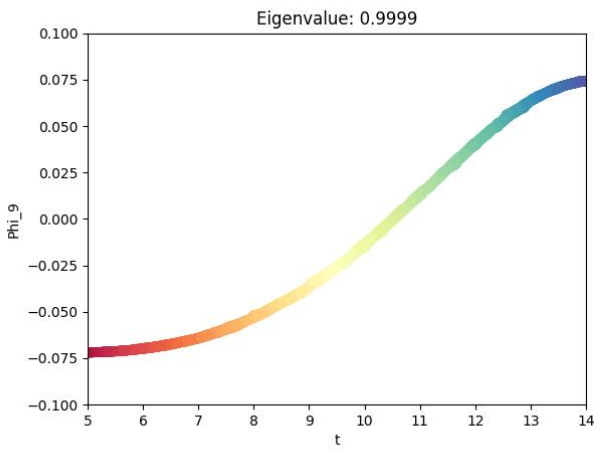


Figure 31: phi-9 vs t

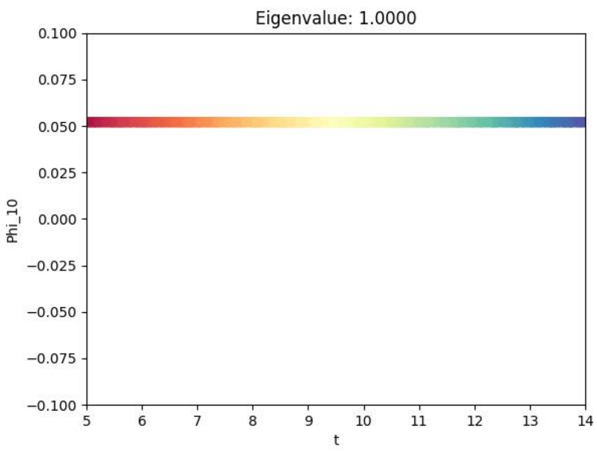


Figure 32: phi-10 vs t

Following are the images of first non-constant eigenfunction ϕ_1 versus other eigen functions (ϕ_1 is on the horizontal axis):

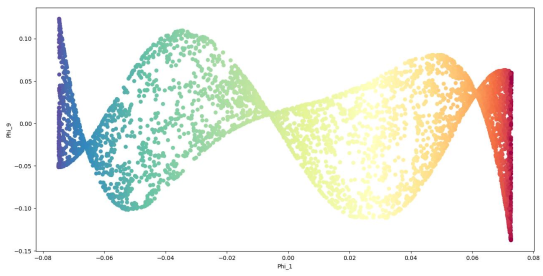


Figure 33: phi-9 vs phi-1

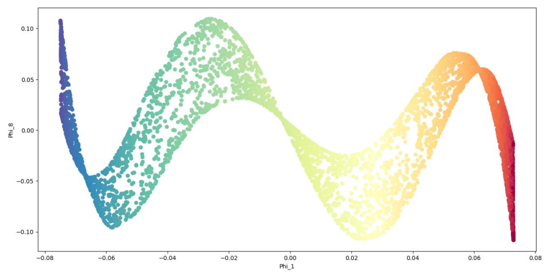


Figure 34: phi-8 vs phi-1

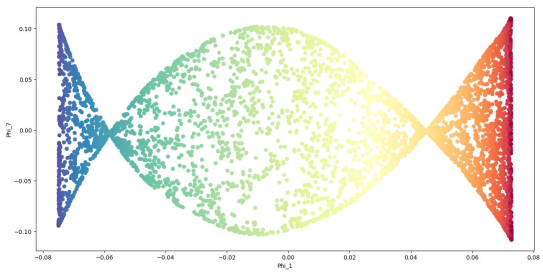


Figure 35: phi-7 vs phi-1

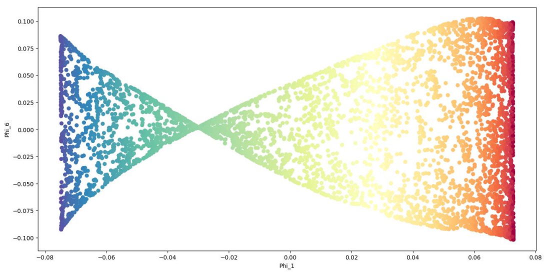


Figure 36: phi-6 vs phi-1

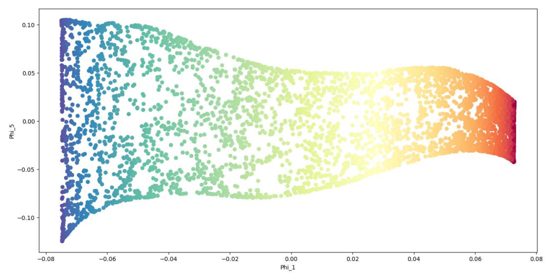


Figure 37: phi-5 vs phi-1

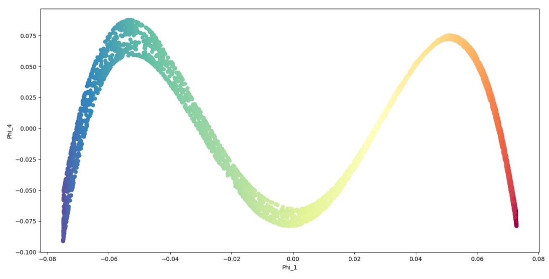


Figure 38: phi-4 vs phi-1

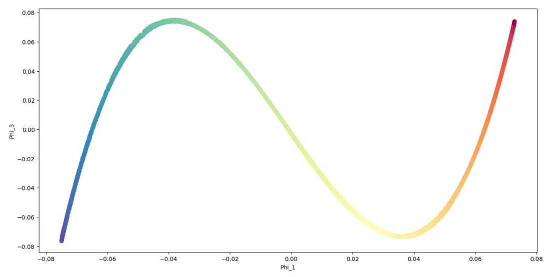


Figure 39: phi-3 vs phi-1

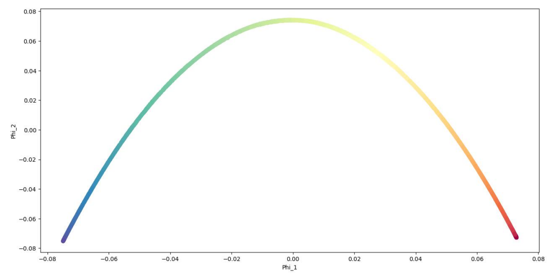


Figure 40: phi-2 vs phi-1

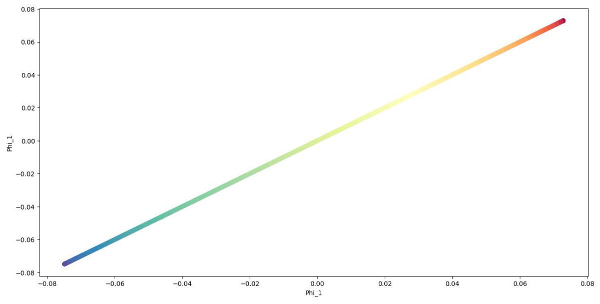


Figure 41: Phi-1 vs Phi-1

From the above plots we can infer that ϕ_1 to ϕ_4 can be represented as some functions of ϕ_1. But from ϕ_5, eigenfunctions cannot be represented as simple functions of ϕ_1.
When PCA analysis is performed using 3 components:

- Explained variance ratio : [0.39451476 0.31812009 0.28736514]
- Sum(Explained variance ratio) : 0.9999999999999999
- Singular values : [501.82729701 450.62802916 428.29173628]

When PCA analysis is performed using 2 components:

- Explained variance ratio : [0.39451476 0.31812009]
- Sum(Explained variance ratio) : 0.7126348568731938

- Singular values : [501.82729701 450.62802916]

It is impossible to only use 2 principal components to represent the data because when 2 components are used the sum of expected variance ratio is 0.7126348568731938 is not equal to 1. That means we have not represented the entire data and have lost a lot of information by not taking in a principal component i.e we reduced the dimensionality too much which led to loss of information.
Instead of $N = 5000$, if we use $N = 1000$, we see that the 10 eigenvalues and eigenfunctions are different.

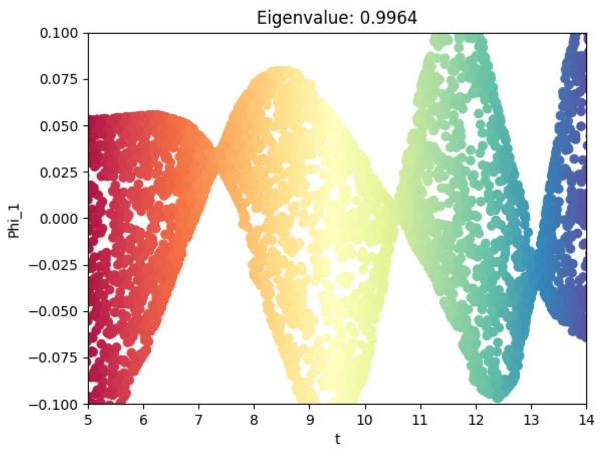


Figure 42: phi-1 vs t when $N = 5000$

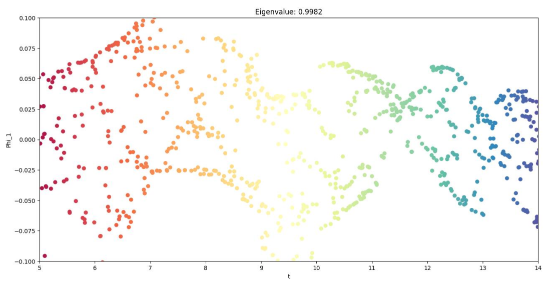


Figure 43: phi-1 vs t when $N = 1000$

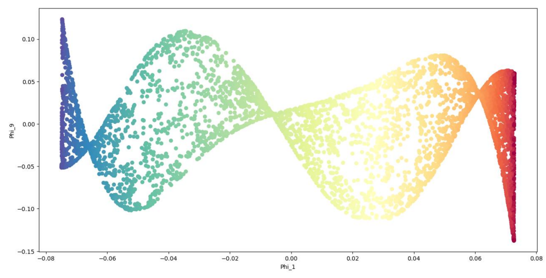


Figure 44: phi-9 vs phi-1 when $N = 5000$

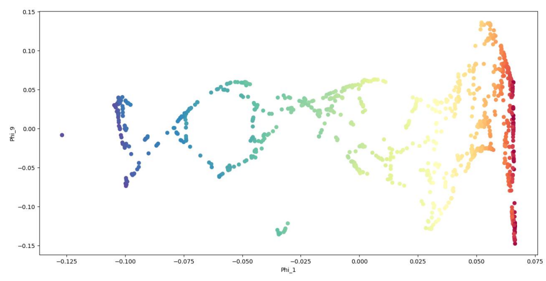


Figure 45: phi-9 vs phi-1 when $N = 1000$

If we run a PCA analysis as well, we see clearly that the singular values obtained are different. When PCA analysis is performed using 3 components for $N = 1000$:

- Explained variance ratio : [0.39826856 0.31804516 0.28368628]
- Sum(Explained variance ratio) : 1.000000000000
- Singular values : [228.60155984 204.28457406 192.9347002]

The above singular values are clearly different from the ones obtained when $N = 5000$ ([501.82729701 450.62802916 428.29173628])

Task 2 part3: We performed PCA analysis on the data in the file data_DMAP_PCA_vadere.txt using both 2 and 3 principal components. Following are the diffusion maps for both the cases.
When 2 principal components are used:

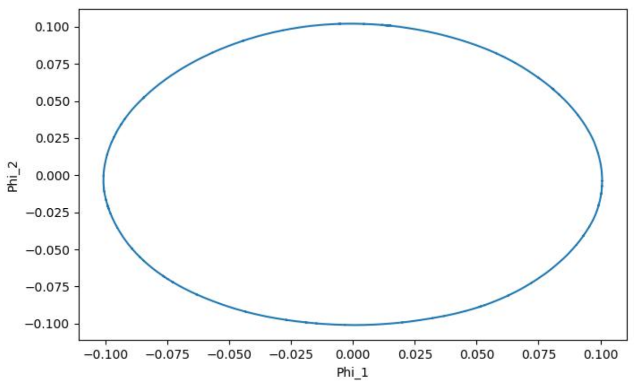


Figure 46: Phi-2 vs Phi-1

While when 3 principal components were used:

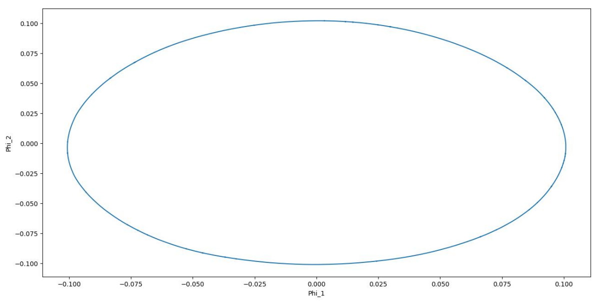


Figure 47: Phi-2 vs Phi-1

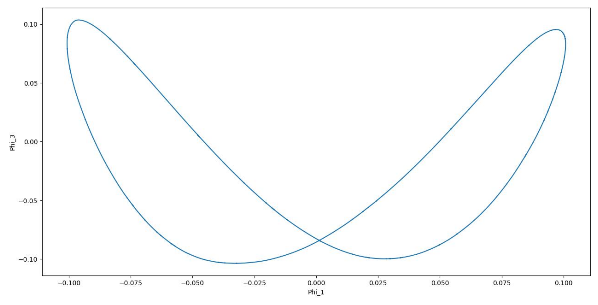


Figure 48: Phi-3 vs Phi-1

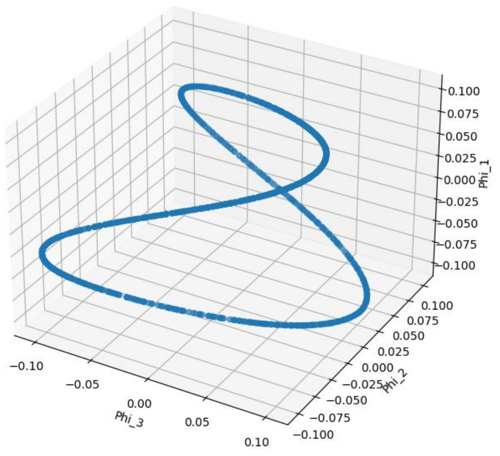


Figure 49: 3D visualization of eigenfunctions

From above visualizations it is clear that we need 3 principal components not 2 components to represent the data accurately.

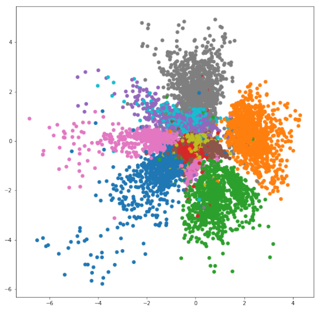
Report on task TASK 3

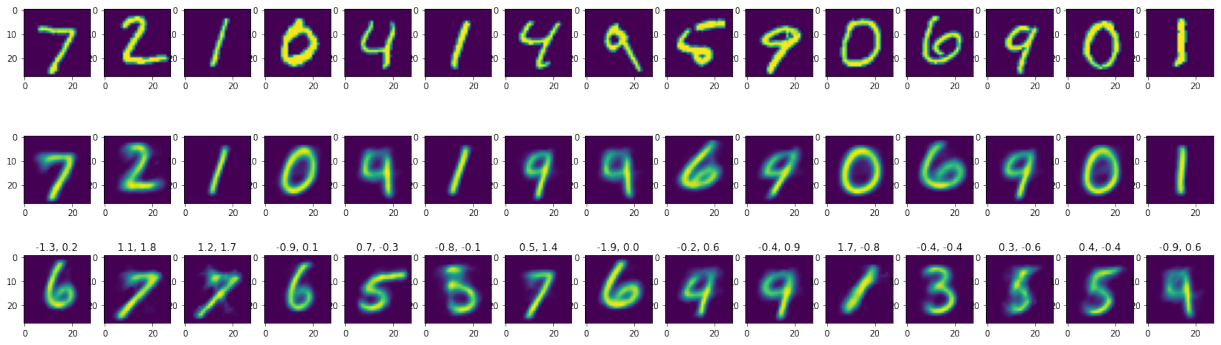
Answers to the questions:

1. The means of the approximate prior and of the likelihood can be computed without using any activation function since the they can be any vectors without restrictions. The standard deviation of the approximate prior should be computed using an exponential activation function since it can take any positive value but no negative or the 0 value. This makes the computed values valid for a standard deviation. Also it is a nice activation since the log which is also required is just the unactivated output of the linear layer.

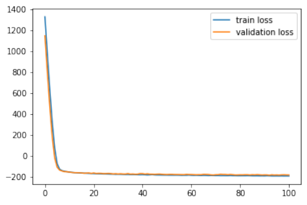
2. One reason for this problem is that the sampled latent vectors for generation are far from any latent distribution and therefore they do not represent meaningful data. This might happen if the encoder learns to encode too small sigmas or too far apart means. Another reason can be the weights that are used to add the KL divergence term and the log probability term. When the weight of the KL divergence is too small, the model comes closer to an autoencoder which can not generate very well. A third problem can be the high dimensionality of the latent space. I this case the relevant distributions cover a very small volume of the space and sampling results are very likely outside any of the learned distributions.

3. The latent space, reconstructed, and generated digits are plotted in the notebook. The plots after 100 epochs are:



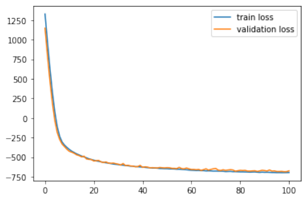


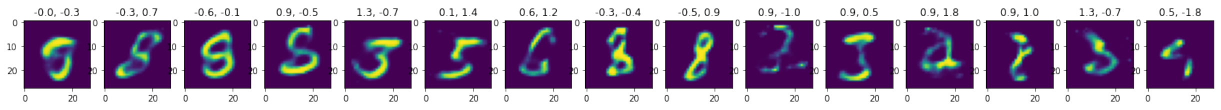
4. The loss curve is:



It is necessary to note that the loss becomes negative which is very intuitive. This happens since the log probability term can become positive (hence negative when multiplying the elbo my -1 to obtain the loos). It becomes positive since the probability distribution function is different than the probability mass function in the sense that it can take values larger than 1.

5. The plots after convergence are:





It can be seen that the generated digits are not as good as in the previous case. The problem is the too-high dimensionality of the latent space as described above.

Report on task Fire Evacuation Planning for the MI Building

The final goal entails applying Machine Learning to analyze the distribution of people in the MI Building in the case of a fire evacuation. It is critical to know where the biggest population concentrations are in advance so that proper escape routes can be supplied to varied populations. The time that was left in order to complete the task was not enough to fully go through it, but we captioned most of the things. The dataset must be downloaded as the initial step in the assignment (which is in .npy format). The dispersion in the train and test sets is depicted in Figure 50 and Figure 51.

As per the task description it was suggested to normalize and to rescale the input data to a range of [1, 1] before the training of the model the model itself. This was done with the help of the `MinMaxScaler` that was imported from the `sk.learn` Library of python. We set the `feature_range` of the `MinMaxScaler` to be equal to (-1,1) as this is the normalization we are looking for.

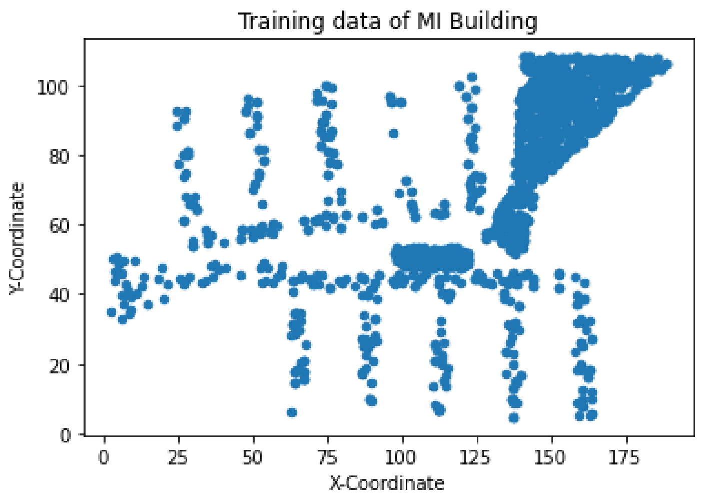


Figure 50: Training data of MI Building

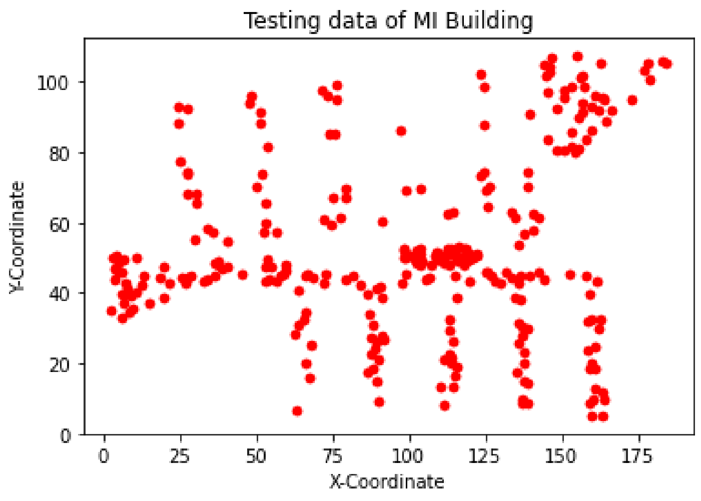


Figure 51: Testing data of MI Building

Because this time the data is 2 dimensional we had to adjust the input and output neurons of the variational autoencoder from the initial 748 to 2.

We tried out different hyperparameters and run several iterations of training the autoencoder but there were not any significant changes to the end result. The final values for the hyper-parameters we trained our network on were the following:

- number of hidden layers: 2 for the encoder, 2 for the decoder
- epochs: 100
- batch size: 64
- learning rate: 0.001

The plot of the training and testing plot could be observed more precisely on Figure 52

As we can see there are a lot of discrepancies when it comes to both the training and the validation loss. It is important to mention however that the ”instability” of the validation loss is much more prominent that the train loss. An idea would be to utilize the concept called early stopping. Early stopping is an optimization technique

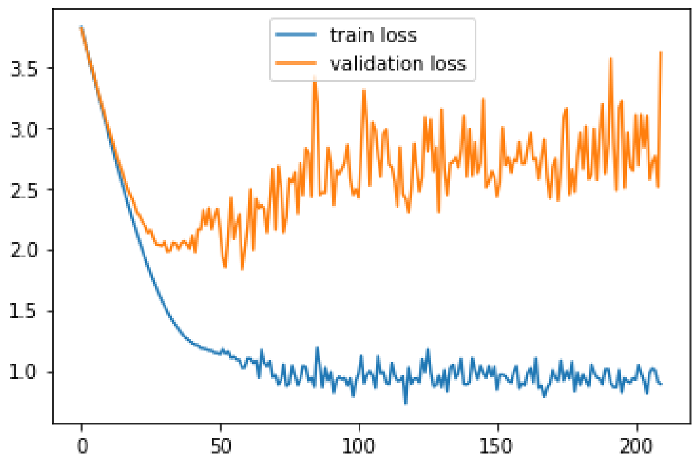


Figure 52: Training and Validation loss of the model after running 100 epochs

used to reduce the effects of overfitting without sacrificing model precision. Early stopping is based on the fundamental principle of discontinuing training prior to the onset of overfitness in a model. But unfortunately we did not have any time in order to be able to integrate early stopping in our code.

One of our next subtasks was to visualize by making a scatter plot of the reconstructed data. The result of this procedure could be seen in Figure 53, Figure 54

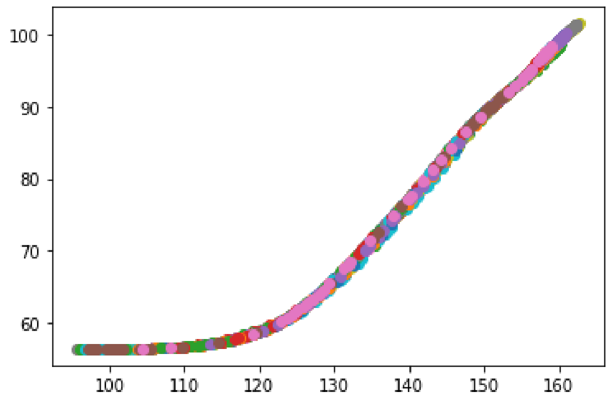


Figure 53: Reconstruction after training the VAE on the datasets: Training set

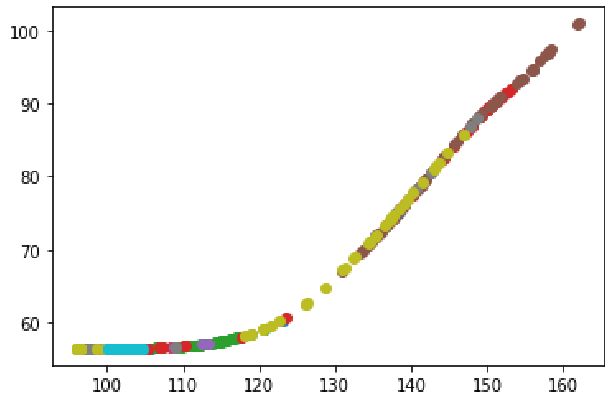


Figure 54: Reconstruction after training the VAE on the datasets: Validation set

We also had two more things to do:

- Make a scatter plot of 1000 generated samples
- Answer the question: ”Generate data to estimate the critical number of people for the MI building: how many samples (people)are approximately needed to exceed the critical number at the main entrance?”

But unfortunately we did not have the time to inspect throughly these two subtasks.