Report for exercise 3 from group J

Tasks addressed:	4
Authors:	Jia Long Ji Qiu Jiabo Wang Yilun Liu
Last compiled:	2023–02–25
Source code:	https://github.com/jialongjq/mlcms

The work on tasks was divided in the following way:

Jia Long Ji Qiu	Task 1	1/3
	Task 2	1/3
	Task 3	1/3
	Task 4	1/3
Jiabo Wang	Task 1	1/3
	Task 2	1/3
	Task 3	1/3
	Task 4	1/3
Yilun Liu	Task 1	1/3
	Task 2	1/3
	Task 3	1/3
	Task 4	1/3

Report on task 1: Principal component analysis

In this task, We have successfully implemented a PCA method that can be applied to various domains of data and produce compressed results in an effective and efficient manner. It is estimated to take us around 12 person-hours in total implementing, testing the method and reporting on this task.

For measuring the data, we programmed from scratch the process of calculating energy loss using different number of principal components. The result of energy loss can be shown in graph with a specific threshold, and the quality of the reconstructed data can also be checked through visualization.

After finishing this part, we have deepened our understanding of SVD, the algorithm itself, and how it is applied in our daily life as a common approach to compress data and save storage with little loss in quality.

1.1 Analyze Dataset

In the first part of this task, we implemented PCA using the singular value decomposition method provided in the NumPy library, which returns U, S and V^H respectively (the S returned is in the form as a list of singular values, representing the diagonal elements within the matrix S we got from PCA).

```
# import libraries and data
import numpy as np
import pandas as pd
import scipy
import matplotlib.pyplot as plt
import seaborn as sns

data = pd.read_csv("pca_dataset.txt", sep = " ", header = None)
data = data.rename(columns={0: "x", 1: "f(x)"})

# PCA
data_avg = np.average(data)
data_bar = data - data_avg
U, S, VT = np.linalg.svd(data_bar)
```

The S and V^T result we get from PCA is $S = \begin{pmatrix} 9.94345787 & 0 \\ 0 & 0.83240085 \end{pmatrix}$, $V^T = \begin{pmatrix} -0.88939864 & -0.45713243 \\ 0.45713243 & -0.88939864 \end{pmatrix}$, in which by using each row of V^T as a vector, we can get the direction of all the principal components. Hereby we plot the dataset, along with the two principle components we get represented by arrows starting from the center of the dataset, as it is shown in figure 1.

```
# plot the data with principal components
sns.scatterplot(data, x = "x", y = "f(x)")
plt.text(VT[0][0], VT[0][1], "1st principal component", horizontalalignment="center",
         verticalalignment="top", color="red")
plt.text(VT[1][0], VT[1][1], "2nd principal component", horizontalalignment="center",
         verticalalignment="top", color="green")
plt.annotate("", xy=(VT[0][0], VT[0][1]), xytext=(0, 0),
             arrowprops=dict(color="red",width=2.0,headwidth=5.0,headlength=5.0))
plt.annotate("", xy=(VT[1][0], VT[1][1]), xytext=(0, 0),
             arrowprops=dict(color="green",width=2.0,headwidth=5.0,headlength=5.0))
```

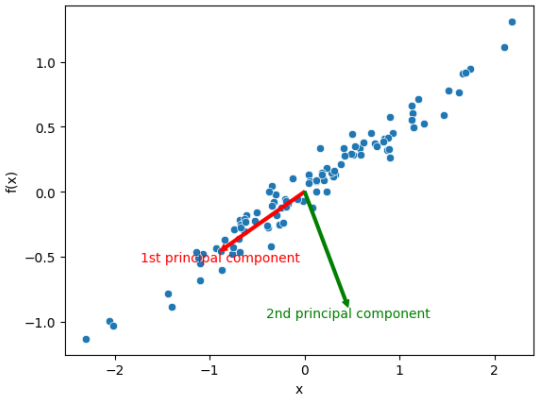


Figure 1: The dataset with computed principle components on the image data

The two principle components we obtained might not appear to be perpendicular to each other due to the aspect ratio of the image.

1.2 Image Data Compression

For the second part, we apply PCA to image data. We obtain the original gray-scale data with size (1024×768) from the scipy.misc.face() method, and then rescale it into 249×185, using interpolation from cv2.INTER_CUBIC, provided in opencv-python package.

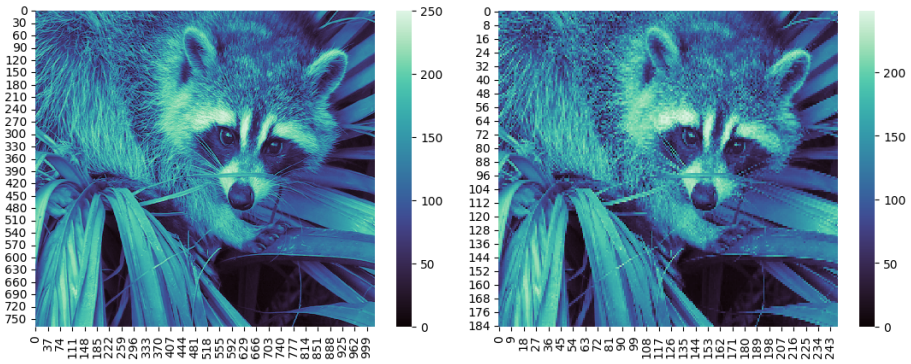


Figure 2: The original (1024×768) and rescaled (249×185) image, represented in pseudo-color

Using columns of the image as the data points in PCA, we apply the same implemented algorithm, and calculate the energy, energy loss, and the number of principal components having energy loss through truncation smaller than the 1% threshold, as is shown in figure 3.

```
U, S, VT = np.linalg.svd(data_bar)
size = S.size

energy = S*S/sum(S*S)
energy_loss = []
energy_loss_threshold = 0.01
energy_loss_index = size
for i in range(size):
    loss =  1-np.sum(energy[:i ])
    if loss < energy_loss_threshold and energy_loss_index == size:
        energy_loss_index = i
        flag = False
    energy_loss.append(loss)
```

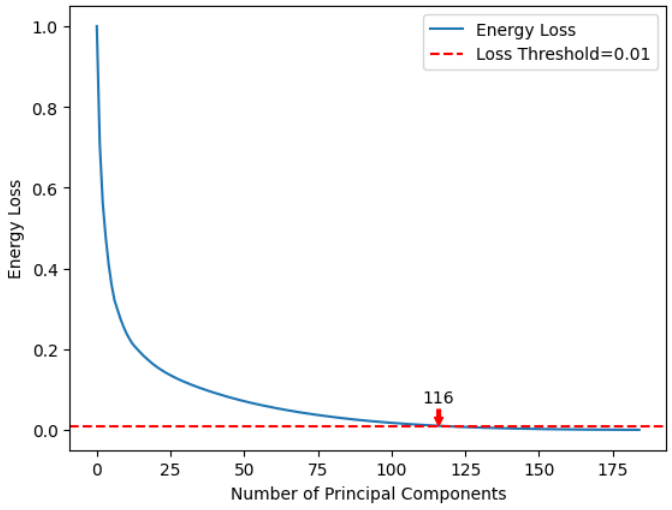


Figure 3: The energy loss with using different number of principal components

From which we can see that using more than 116 principle components can achieve to make the energy lost through truncation smaller than 1%.

Now for the visualization, we apply the aforementioned PCA method to the rescaled image with all (185), 120, 50 and 10 principle components by truncating the U, S, V^T matrices we got into corresponding sizes. The reconstructed images we got is shown in figure 4.

```
U_r = U[:, :r]
S_r = S[:r]
VT_r = VT[:r, :]
```

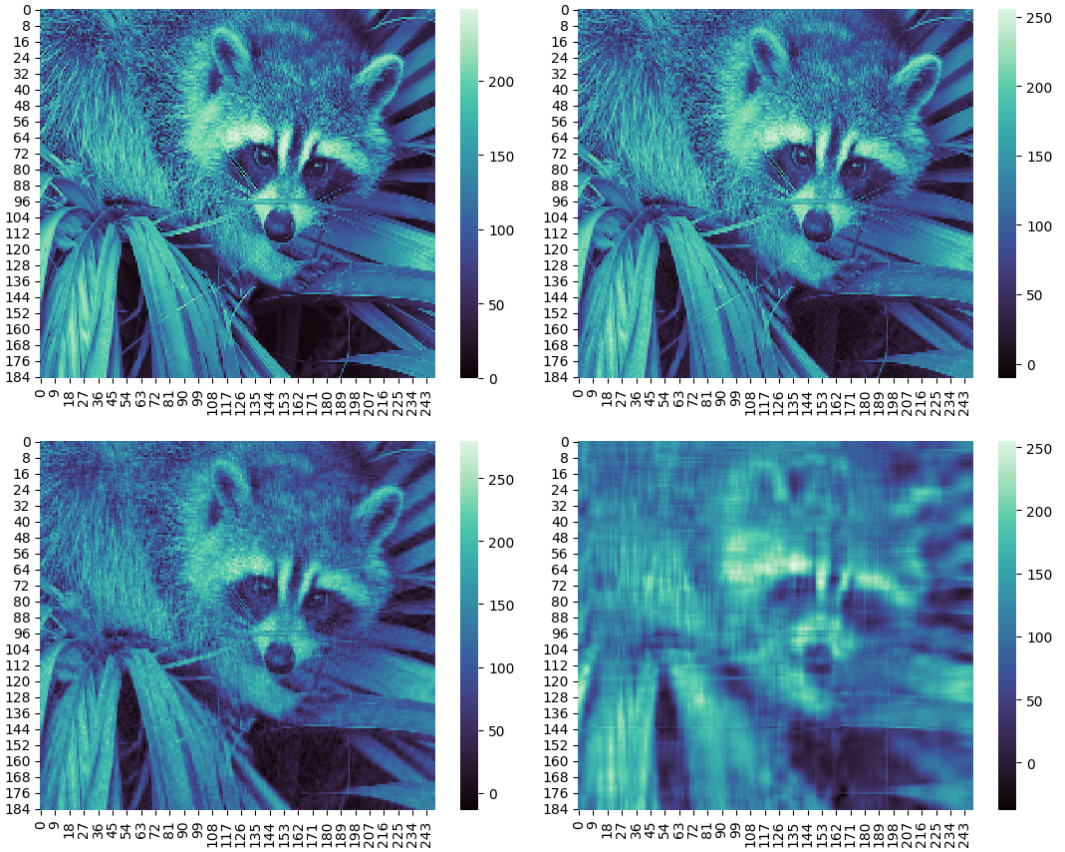


Figure 4: The reconstructed images using 185 (upper left), 120 (upper right), 50 (lower left) and 10 (lower right) principle components

As we can see the difference between the original image and reconstructed results using 185 (all) and 120 components is hardly visible, as their energy loss is all below the threshold of 0.01. Whereas many details of the image reconstructed from using 50 principle components are lost, such as the raccoon's hair, whiskers, and the edges of the surrounding foliage become blurred, while noticeable noise appears in large areas that should be smooth blocks of color. And the result using only 10 components is almost unrecognizable.

1.3 Vadere Trajectory Data Compression

For this part, we are doing PCA on the trajectory data of 15 pedestrians over 1000 time steps. As the form the original data is provided, pedestrians' locations are stored on a 2-dimensional plane, therefore for each pedestrian, we consider their (x, y) coordinates as two independent variables within the time sequence. This can also be observed from visualizing the path of the first two pedestrians in the 2-dimensional space (figure 5).

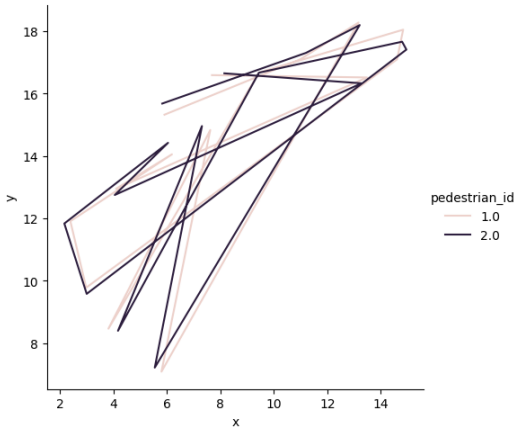


Figure 5: The path of the first two pedestrians in the 2-dimensional space

Hence we have a dataset with 30 variables and 1000 records. Now we apply PCA on this dataset and analyze the energy loss (figure 6).

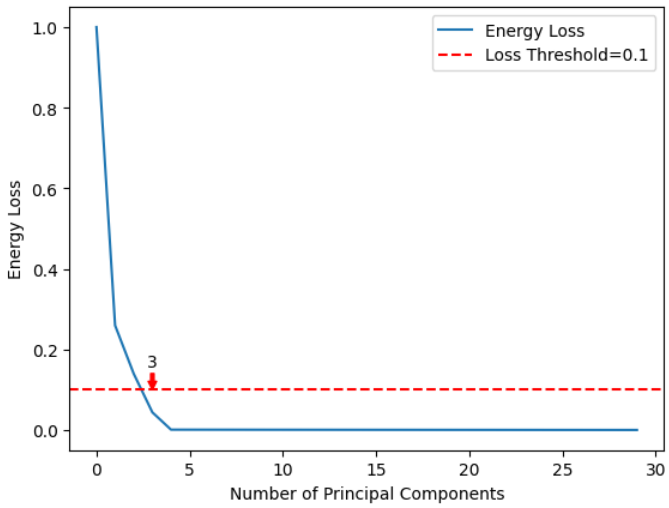


Figure 6: The energy loss with using different number of principle components on the trajectory data

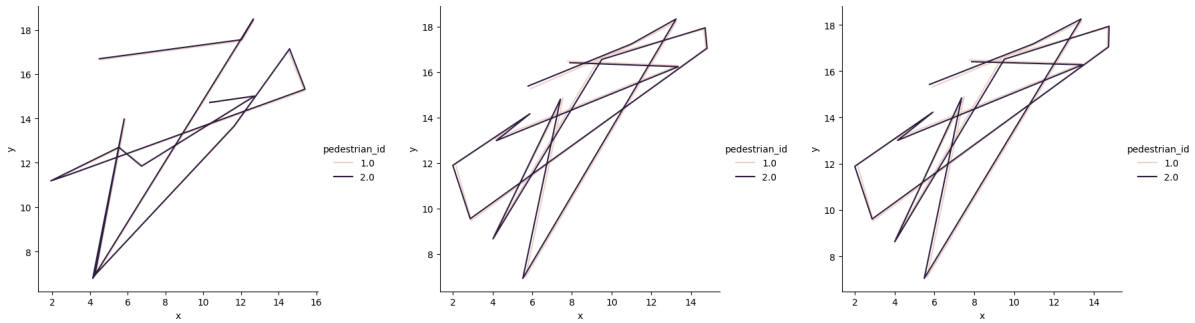


Figure 7: The reconstructed path of the first two pedestrians in the 2-dimensional space using 2 (left), 3 (middle) and 4 (right) principal components

From figure 6 we can see that two components are not enough to capture more than 90% energy of the dataset. To do so, we need at least 3 principal components, which can reduce the loss lower below the threshold.

The visualization result of the path of the first two pedestrians from both is shown in figure 7. The result with 2 principal components much deviates from the original curves, and the ones with 3 and 4 principal components are nearly identical to each other, both of which managed to capture a lot of details. Further increase the amount of principal components used can hardly improve the energy loss.

Report on task 2: Diffusion Maps

This algorithm was very clearly described in the statements, to implement it we just followed step by step the description. The implementation of the algorithm is located in the file DiffusionMaps.py and it takes as parameters the data and the number L + 1 of eigenfunctions wanted. As output, it returns two values: the eigenvector and the corresponding eigenvalues. For the distance matrix computation, we have followed the recommendation of using a sparse version and only considering points in a reasonable neighborhood, so we use the library scipy to implement the distance matrix.

It took us around two-three days to implement it and test it. To test it, we have used the library datafold to compare the results and plots. The plots obtained with our algorithm have the same shape of the ones obtained with the Diffusion Maps algorithm of the datafold library, although the values are slightly different. We think this is maybe caused because the datafold uses another method to implement it.

Finally, different from PCA that only performs well for datasets close to a hyperplane (linear dimensionality reduction method), Diffusion Maps can reduct datasets that are non-linear. Given data in a high-dimensional space, the method focus on discovering the underlying manifold (lower-dimensional constrained "surface" upon which the data is embedded).

2.1 Similarity of Diffusion Maps and Fourier analysis

First, we have created the periodic data set with N = 1000 points given by

$$X = \{x_k \in \mathbb{R}^2\}_{k=1}^{N}, x_k = (cos(t_k), sin(t_k)), t_k = (2\pi k)/(N+1)$$

A plot of this data set in 3D (the two parameters of x_k and t_k) can be observed in the Figure 8.

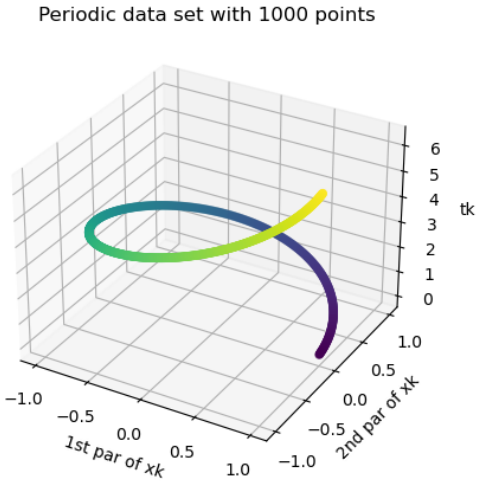
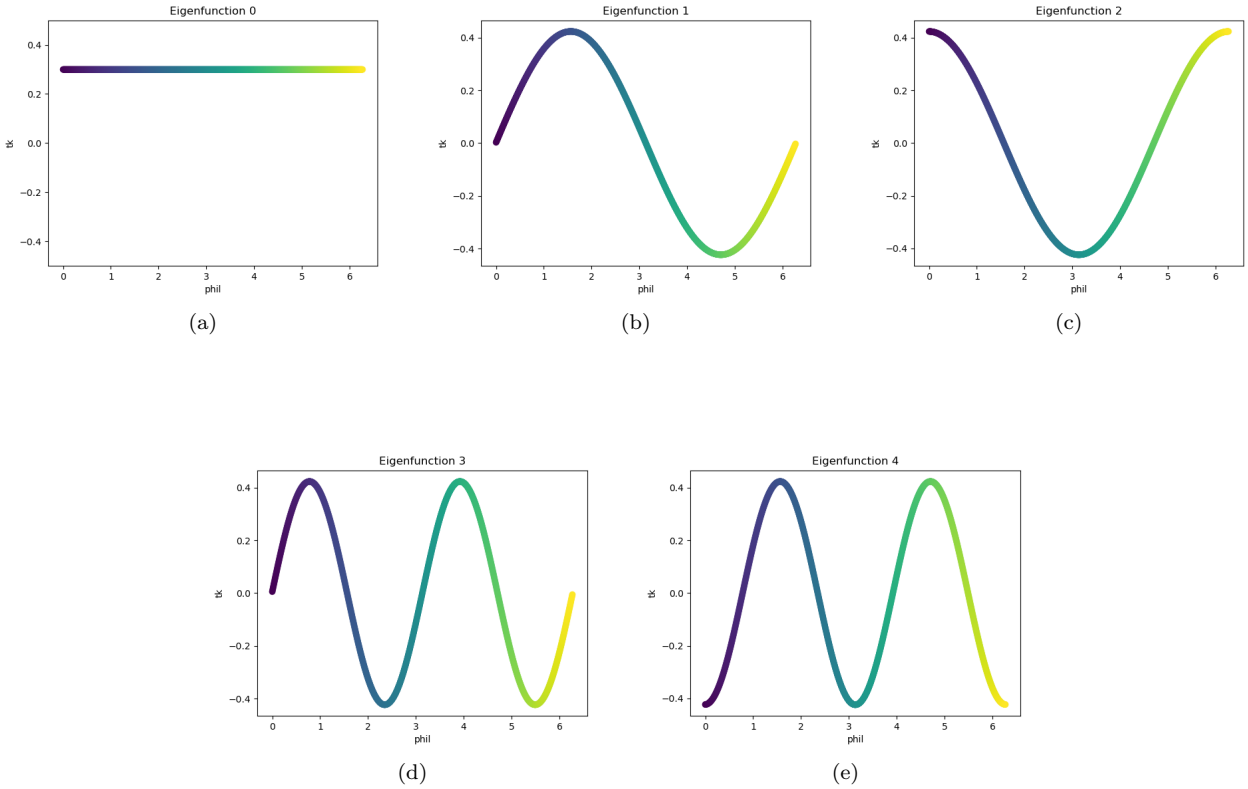


Figure 8: Periodic data set of 1000 points

After plotting the data set, the five eigenfuctions ϕ_l associated to the largest eigenvalues λ_l have been computed with the Diffusion Maps algorithm on the periodic data set mentioned before. Below we can observe the five plots of the eigenfunctions $\phi_l(x_k)$ (y-axis) against t_k (x-axis).



(a) (b) (c)

(d) (e)

Figure 9: Plots of the values of the eigenfunctions against t_k

Analysis: After observing the plots, we can see that the eigenfunction associated to the largest eigenvalue Figure 9a is constant while the others have a sinusoidal form, but with different period. And the Fourier transformations also converts periodic complex functions to combination of more simple sinusoidal functions. In conclusion, both methods tries to transform the original given data (that used to be complex periodic functions) to a combination of simple sinusoidal functions with different periods (or frequency).

2.2 Swiss roll manifold

First, with the sklearn Python library we have easily generated a Swiss roll data set with 5000 data points in three dimensional space. The Swiss roll manifold is defined through

$$X = \{x_k \in \mathbb{R}^3\}_{k=1}^{N}, x_k = (ucos(u), v, usin(u)),$$

where (u, v) $\in [0, 10]^2$ are chosen uniformly at random. A plot of this data set can be observed in the Figure 10.

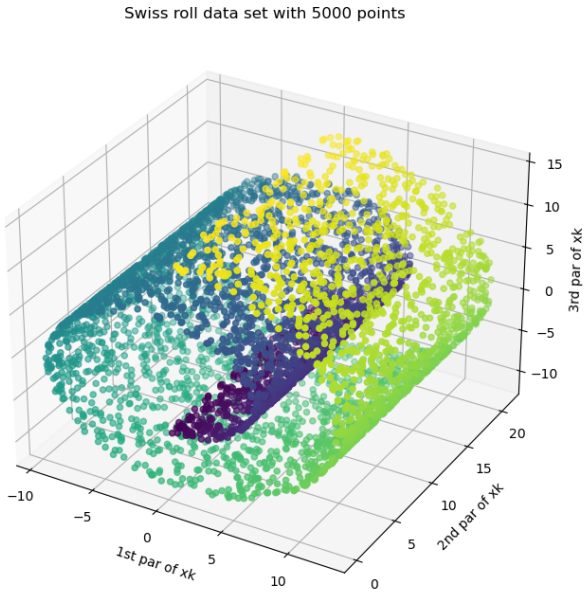
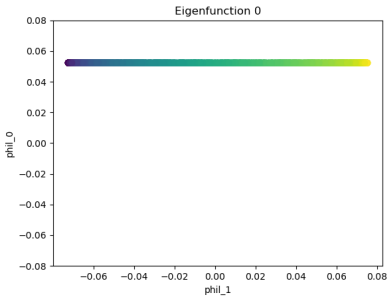
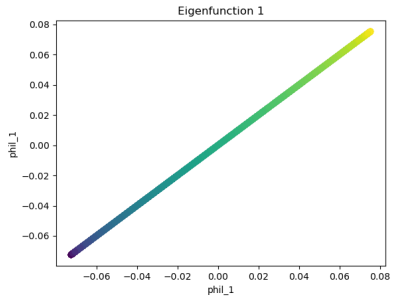


Figure 10: Swiss roll data set with 5000 points

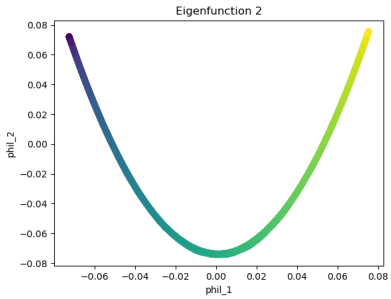
After that, we have obtained the first ten eigenfunctions of the Laplace Beltrami on the Swiss roll data set and observed that, in fact, the first non-constant eigenfunction was ϕ_1, so next we have plotted the other eigenfunctions against this one in 2D plots (with ϕ_1 in the horizontal axis). The plots can be observed in the next figure.



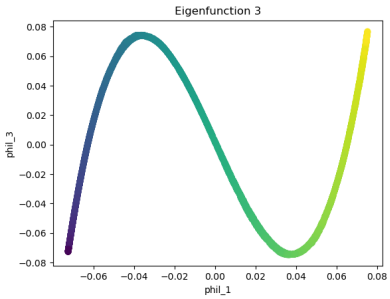
(a)



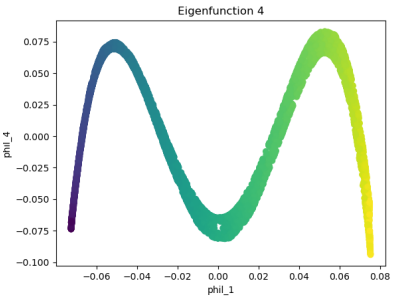
(b)



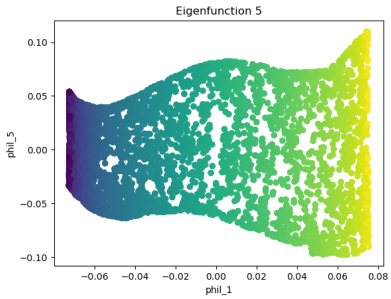
(c)



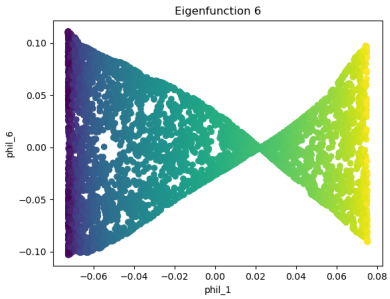
(d)



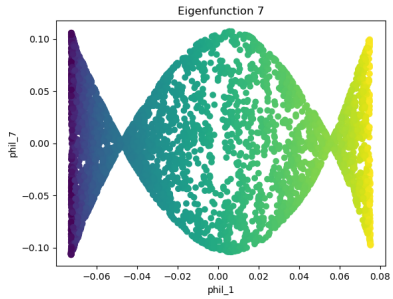
(e)



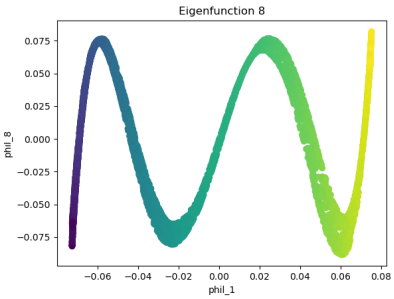
(f)



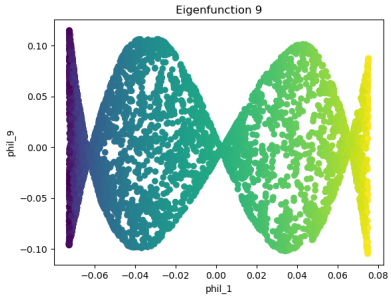
(g)



(h)



(i)



(j)

Figure 11: Plots of the eigenfunction ϕ_1 against the others (5000 data points)

As we can observe in the different plots, the first one ϕ_0 is constant, so after plotting it against ϕ_1 we obtain a straight line. Obviously ϕ_1 against ϕ_1 is a diagonal. ϕ_2 and ϕ_3 are still functions of ϕ_1 because each value of the x-axis (ϕ_1) corresponds to one value of the y-axis (ϕ_2 or ϕ_3). We don't have this one-one correspondence since ϕ_4 and so on. This method is only by looking visually at the graphs, but there is also another mathematically based method, this one uses the by using the Local Regression Selection method of the datafold software and gives us the results. The results it gives is that the first value of l is 5. It confuses us a little bit, because of course phi_5 is not longer a function of phi_1, but phi_4 also have not that one-one correspondence during all the graph. The shape of phi_4 does look like phi_3 and we believe more to the numerical method, so, 5 is the first value of l such that ϕ_5 is not longer a function of ϕ_1.

Next, we need to compute the three principal components of the Swiss roll data set. First, we decomposed the centred data matrix into singular vectors U, V and values S with the svd method. To check if these values are correct, we have reconstructed the initial data set with these values, and after plotting it, they are quite similar, so we have assumed that it is correct. The three principal components are located in the V^T vector and have the following values:

- Principal Component 1: [0.33333601 -0.72495673 0.60276434]
- Principal Component 2: [-0.38033221 -0.68839766 -0.6176213]
- Principal Component 3: [-0.86269028 0.02337527 0.50519212]

Regarding at the question why is it impossible to only use two principal components to represent the data is that we will loss a lot of energy just by removing one component as we can see in the plot of the Figure 12 (removing the third principal component, by setting the corresponding value of the S vector to 0). So we need all three principal components for the Swiss roll data set in order to not lose energy and be able to visualize well the data set.

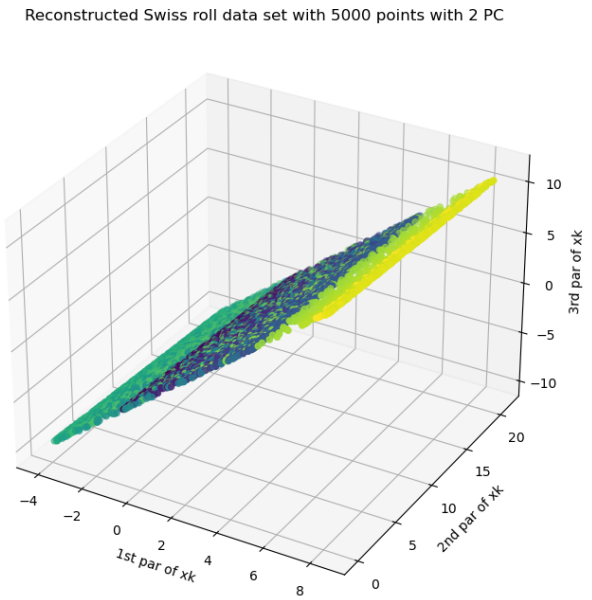


Figure 12: Plot of the Swiss roll data set with 2 PC (5000 data points)

The final question of this part is about what happens if only 1000 data points are used.
First, let's analyse it for the Diffusion Maps. Visualizing the plot of the data set (Figure 13), we can observe that obviously there are less points than the previous one so it is less representative of the manifold. Also this

reduction of data points can be observed in the eigenfunctions plots created with the Diffusion Maps algorithm, where there are much more noise than the previous one as we can see in the Figure 13.

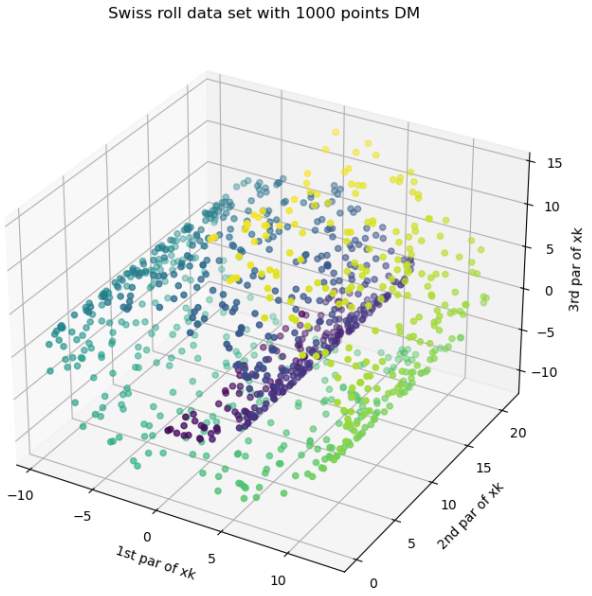


Figure 13: Plot of the Swiss roll data set with DM (1000 data points)

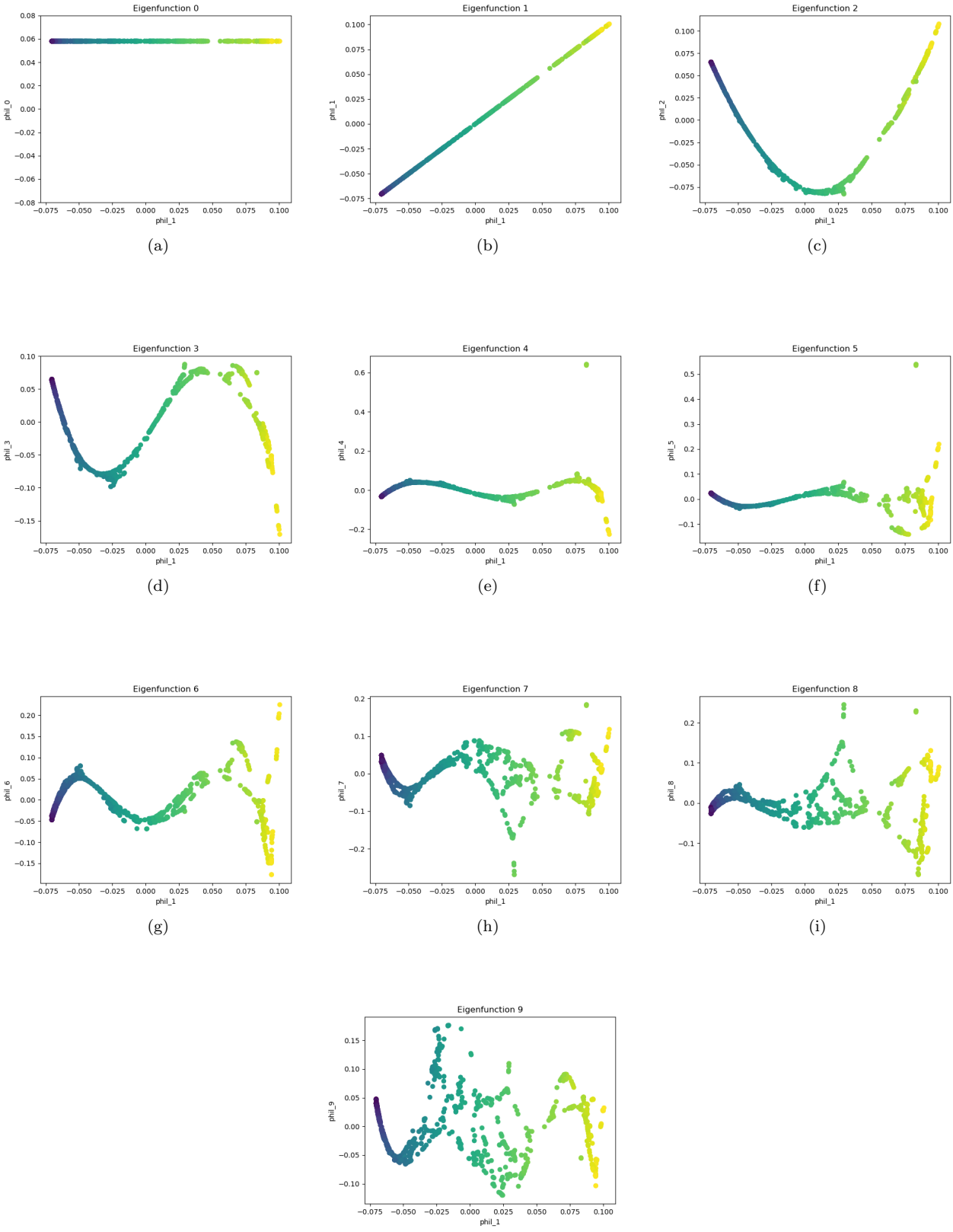


Figure 14: Plots of the eigenfunction ϕ_1 against the others (1000 data points)

In the PCA case, again we have used the svd method to obtain the U and Vt vectors and the values S. By reconstructing it, the plot of the Figure 13 has been obtained, so the same of the Diffusion Maps. As well as the previous case, it has three principal components, and it is not possible to just use two of them because again we will lose energy. A plot of the data set with 2 principal components can be observed in the next figure.

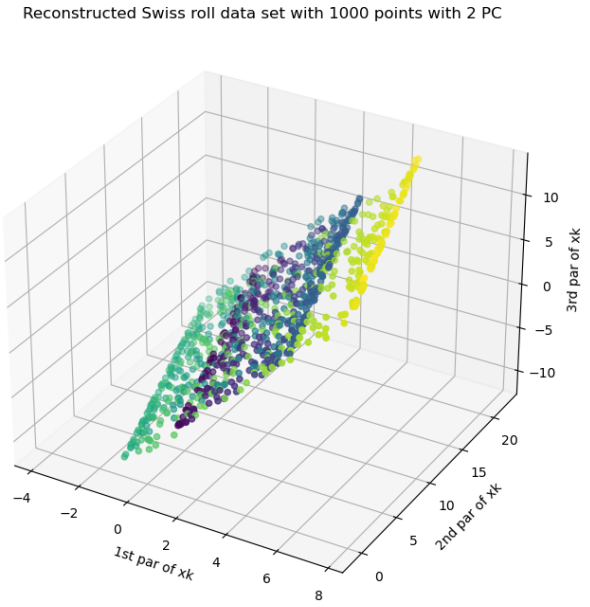


Figure 15: Plot of the Swiss roll data set with 2 PC (1000 data points)

BONUS

The bonus consists on downloading the datafold software and compute the eigenvectors of the Swiss roll data set of the part 2, plot them against each other and analyze it.

First, following the tutorial, we have generated the plot of the Swiss roll using 5000 data points, as we can see in the next figure:

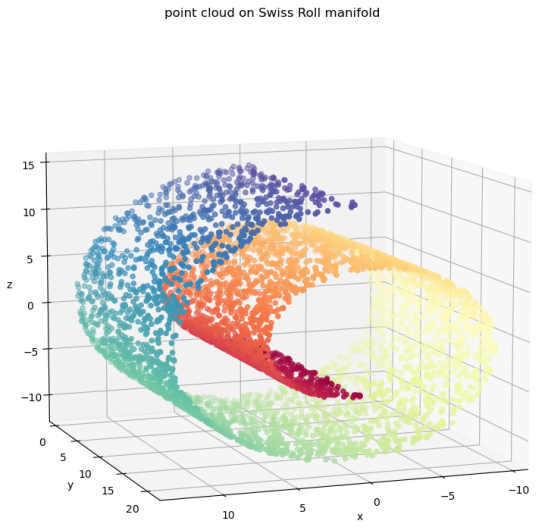


Figure 16: Plot of the Swiss roll data set generated with datafold (5000 data points)

The next step is to optimize the kernel parameters to get the epsilon and cut-off.

Next, we have fit a DiffusionMaps model of the datafold with the optimized parameters and then compared potential two-dimensions embeddings, as we have done in the part 2. The plots can be seen in the Figure 17.

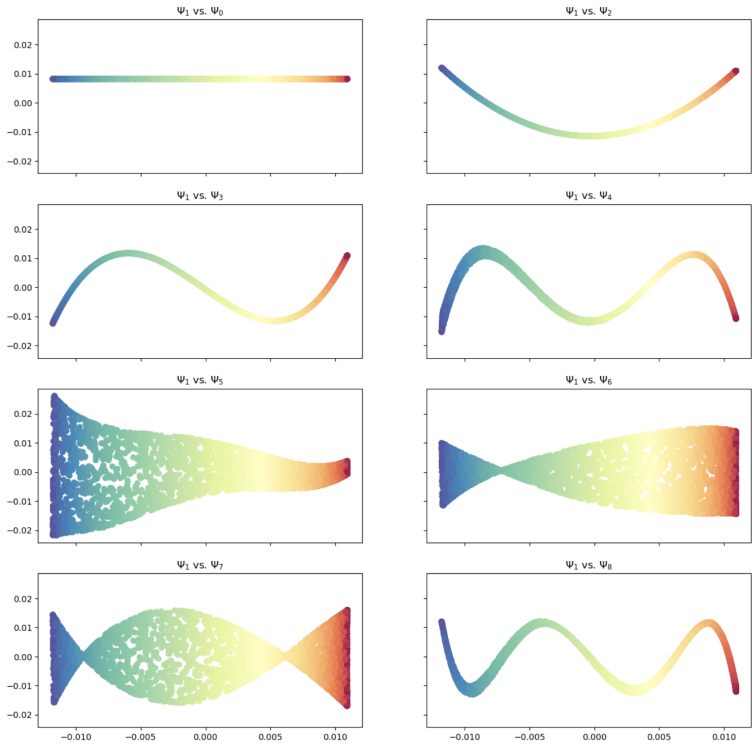


Figure 17: Plots of the eigenfunction ϕ_1 against the others datafold version (5000 data points)

As can be observed, this plots are very similar to the ones we have obtained in the part 2. The same problem occurs here with the plot of the ϕ_1 against ϕ_4, where we can see an irregularity at the beginning of the plot, where there is no one-one corresponding. However, as before, using the Local Regression Selection, the first l such that ϕ_l is not longer a function of ϕ_1 is again 5.

Finally, there is a plot of the unfolded point cloud:

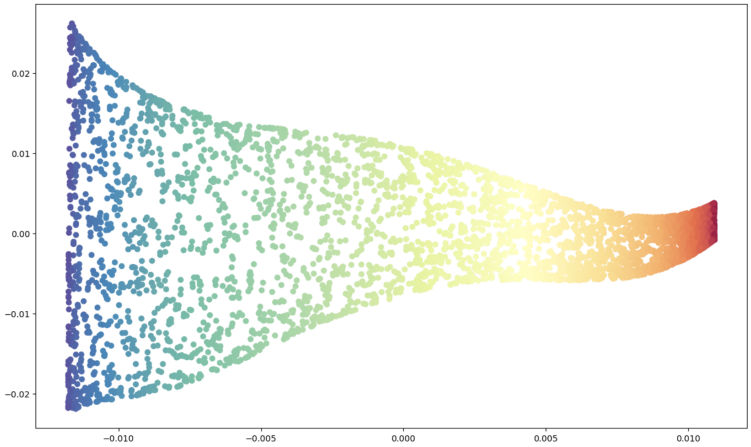


Figure 18: Unfolded point cloud of the Swiss roll data set

In conclusion, we have basically followed the steps of the tutorial and the results obtained are the expected ones, because they are almost the same than the ones we have obtained in the part 2 of the task.

2.3 Trajectory data analysis

In the last part of this task, we have to analyse the trajectory data in the file data_DMAP_PCA_vadere.txt on Moodle as we did in the task 1, but in this case with Diffusion Map. Let's take a look at the path of the first two pedestrians in 2D space (Figure 19)

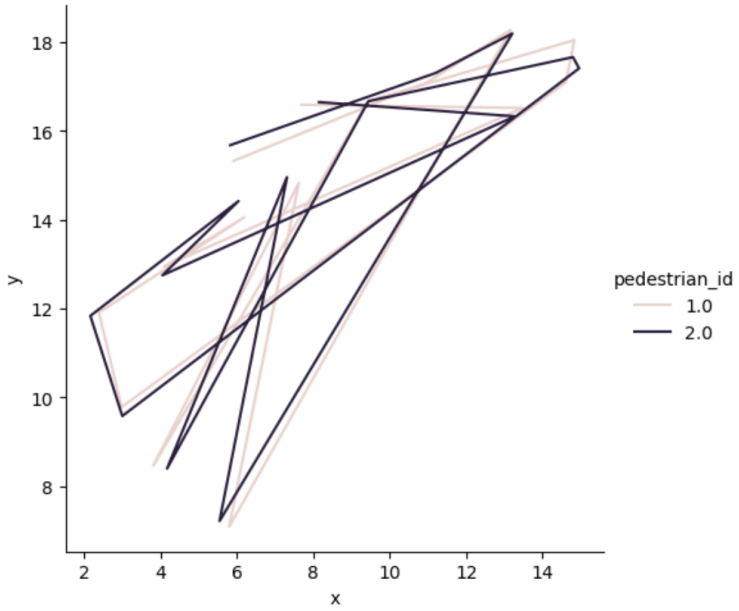


Figure 19: Path of the two pedestrians in 2D space

We notice that Diffusion Maps do not have exactly the same energy interpretation of the eigenvalues (singular values) than PCA, so the task we have to do is finding a good combination of eigenfunctions to accurately

represent the data set. A good combination of eigenfunctions is that one which leads to a subspace where there are no "intersections" of the curves of the unfold data.

First, we have tried with 2 eigenfunctions in a range of 8 total eigenfunctions, the plots are shown below:

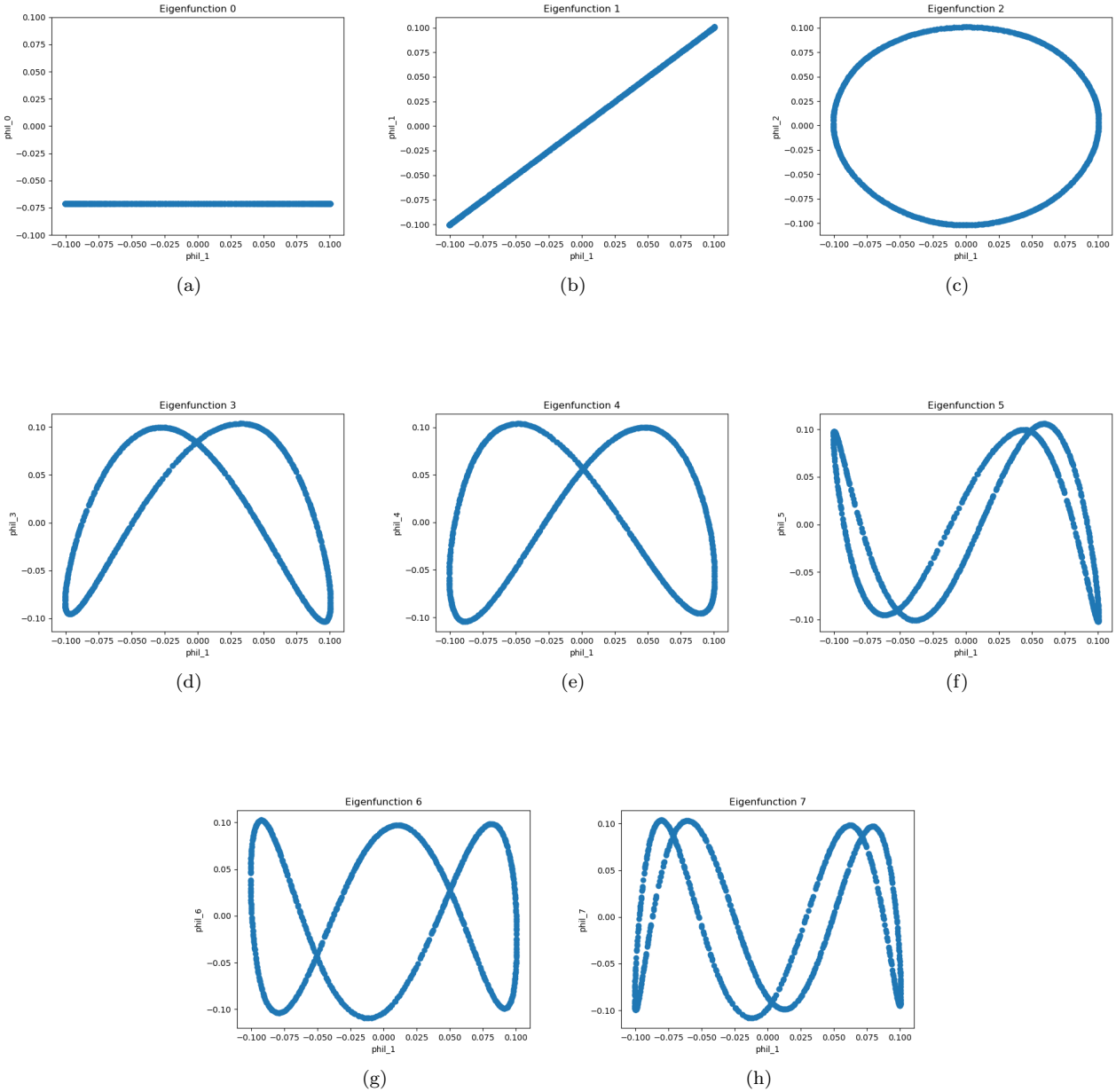


Figure 20: Plots of the eigenfunction ϕ_1 against the others

As we can observe in the plots, the eigenfunction ϕ_1 against the eigenfunction ϕ_2 is accurate to represent the data set because there are no "intersections" of the curves so just using two eigenfunctions it's enough to get a proper embedding.

Report on task 3: Training a Variational Autoencoder on MNIST

This third task has been really challenging since it was our first time working with Variational Autoencoders while having little knowledge of statistics and deep learning. In fact, we could not implement the proper model of the VAE described in the exercise sheet. The implementation of the encoder was pretty clear. However, the decoder part and the ELBO loss was a bit tricky and we could not really understand it. In the end, after several days of trying to implementing it, we ended up following a tutorial which implementation was a bit different from what we were asked to do.

- What activation functions should be used for the mean and standard deviation of the approximate posterior and the likelihood—and why?

The standard deviation should be positive by convention, therefore positive-valued activation functions such as ReLU, Sigmoid, Softplus, etc. should be used for both the approximate posterior and the likelihood standard deviations. Moreover, for the standard deviation of the likelihood, since it is needed to calculate the density function and it is a divisor of this formula, a non-zero activation function must be used, such as Sigmoid.

As for the mean, since it can take any real value, it is not compulsory to use any activation function.

- What might be the reason if we obtain good reconstructed but bad generated digits?

The performance of the digit reconstruction and generation depends mainly on how well the latent space is modeled.

The reason for obtaining good reconstructed but bad generated digits might be due to a unbalance between the reconstruction and the KL divergence losses, more specifically for this case, the reconstruction being better than the KL divergence. This means that every input is well sampled in the latent space but badly distributed over the prior, which does not allow a good generation of digits from arbitrary points of this space, since it is very restricted to the training set used for modeling it.

Report on task 4: Fire Evacuation Planning for the MI Building