Unsupervised Learning - Exercise 2

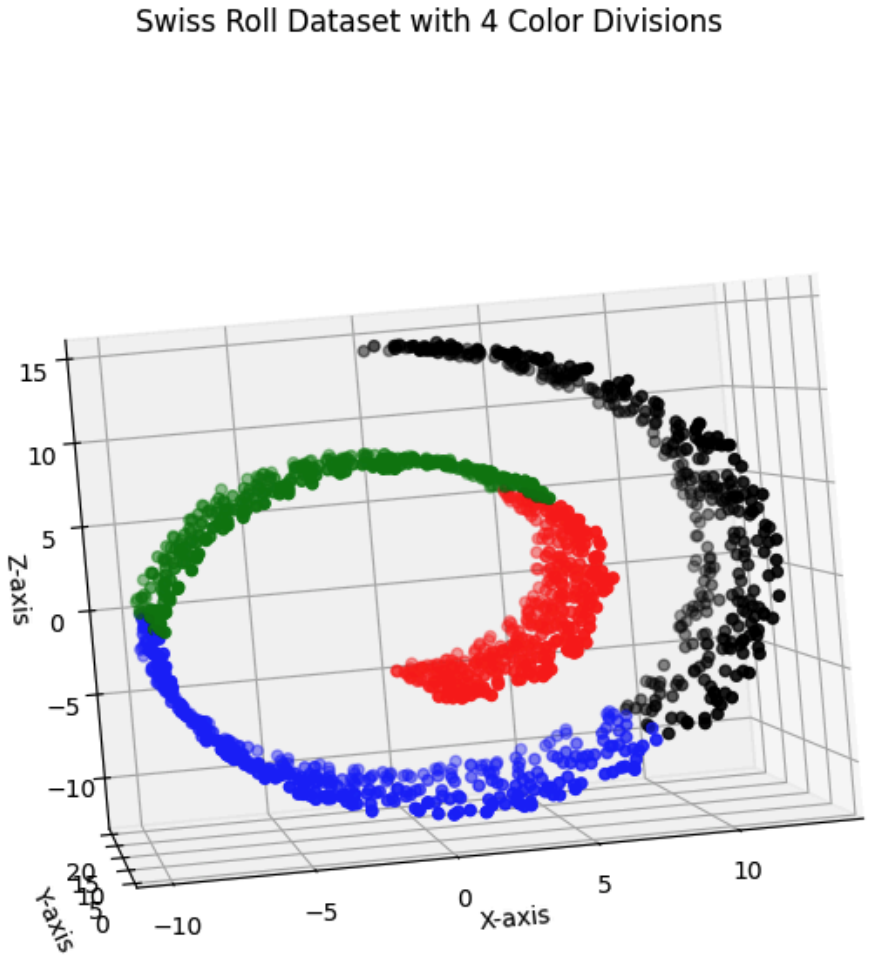
Naama Avni 208523456

Question 1

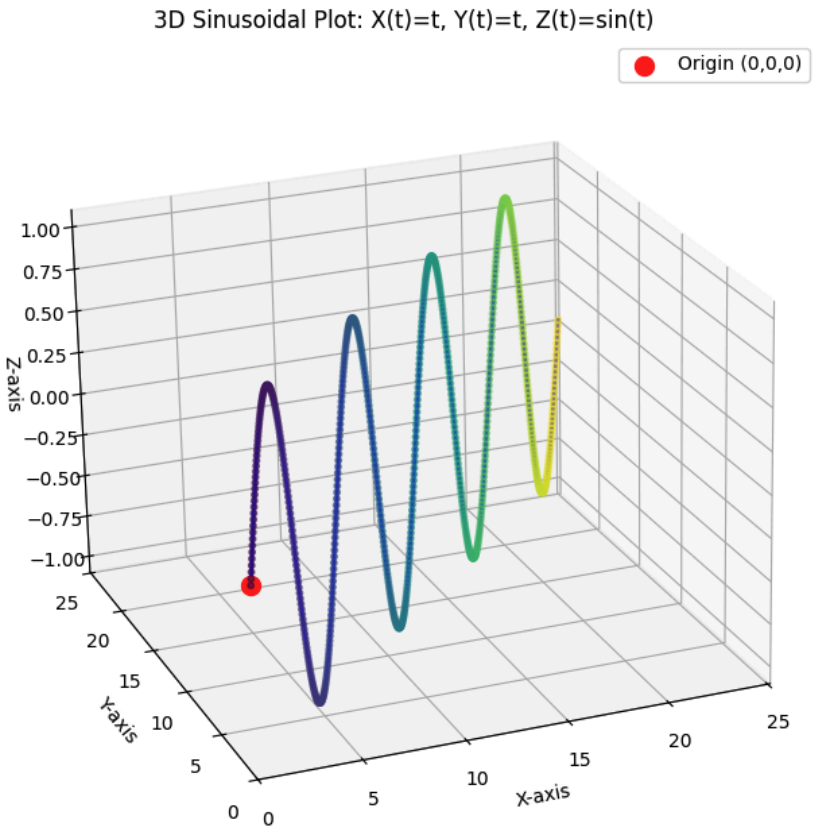
On this question we generate the required data as described, in order to use it on the following question with dimension reduction techniques.

Output plots:

4 groups swiss roll



3D Sinusoidal Curve



Question 2

Question 2, part 1

We'll describe the algorithm for PCA using SVD while N≫D. It was more convenient for me to do it in Hebrew.

לביצוע PCA בעזרת SVD למטריצה X בגודל N*D, במצב שבו N≫D כלומר, יש הרבה יותר דוגמאות (שורות) מאשר ממדים (עמודות), נפרט את השלבים הבאים:

1. ננרמל את כל הנתונים במטריצה X כך שהממוצע עבור כל הפיצ׳רים(עמודות) יהיה אפס.
2. נבצע עבור המטריצה המנורמלת X פירוק SVD(שממחיש את חישוב מטריצת הcovarience ומוצא את הeigenvectors וeigenvalues) כך שנקבל עבור X:

Unset

$$X = U\Sigma V^T$$

a. U is an N×N orthogonal matrix

b. Σ(Sigma) is an N×D diagonal matrix with singular values - the eigenvalues

c. V is a D×D orthogonal matrix which it's columns are the eigenvectors represent the principal components.

3. כדי לבצע את הורדת המימד מהמימד הקיים D למימד נמוך יותר K, ניקח את k הeigenvalues הגדולים ביותר (הראשונים) במטריצה Σ, ואת K הeigenvectors (העמודות) התואמים אליהם ממטריצה V.
4. ונבצע את 'הקרנת' הנתונים למימד החדש באמצעות הכפלת k הוקטורים במטריצה המנורמלת X לקבלת מטריצה חדשה Xproj בגודל N*K.

Unset

```
Xproj = X*Vk
```

5. המטריצה שהתקבלה Xproj בגודל N*K משמרת את מירב הנתונים שניתן במימד מוקטן K עבור המטריצה המקורית X.

Question 2, part 2

On this part, we implement two different approaches to PCA using SVD:

1. pca_a_N_bigger_than_D(X, K=2) - PCA implementation for cases where N >> D (number of samples is much larger than dimensions):
   - Centers the data by subtracting the mean
   - Performs SVD directly on the centered data
   - Uses the right singular vectors (V) as principal directions
   - Projects the data onto the top K components
2. pca_b_D_bigger_than_N(X, K=2) - PCA implementation for cases where D >> N (dimensions are much larger than number of samples):
   - Centers the data
   - Computes the Gram matrix (X^T X) which is more efficient when D >> N
   - Performs SVD on the Gram matrix

- Computes principal components using the formula: X^T @ u_i / sqrt(s_i)
- Normalizes the components
- Projects the data onto K dimensions

Question 2, part 3

This part demonstrates the application of PCA on different datasets. Here's what each part does:

1. pca_swiss_roll(): Applies PCA to the Swiss roll dataset. The function visualizes how both PCA methods (N>>D and D>>N) project this 3D data into 2D space.
2. pca_sinusoidal_curve(): Applies PCA to a sinusoidal curve dataset. The function visualizes how both PCA methods (N>>D and D>>N) project this 3D data into 2D space.
3. pca_5_5(): Applies PCA to a specific dataset from a previous exercise (the "5,5" dataset), reducing it to 1D and visualizing the results with some jitter for better visualization.
4. pca_D_0_3_8(): Applies PCA to a subset of the MNIST dataset, specifically focusing on digits 0, 3, and 8. This demonstrates how PCA can be used to reduce the dimensionality of image data while preserving the main features that distinguish these digits.
5. mean_shift_analysis_D_0_3_8(): Takes the results from the MNIST PCA and performs a clustering analysis using Mean Shift. This part compares how well clustering works on:
    - The original high-dimensional data (64 features)
    - The PCA-reduced data (2D)

    It analyzes different bandwidth values and compares the clustering performance between the original and reduced data.

Each of these steps generates visualizations that are saved as PNG files, making it easy to compare how different PCA methods perform on various types of data.

All the plots available in the question directory, here is the code output:

```
Unset
=== Mean Shift Analysis on D_0_3_8 Dataset ===
Original data shape: (535, 64)
PCA data shape: (535, 2)

--- Original High-Dimensional Data (64D) ---
Using bandwidth values: [15, 30, 32, 33, 35]
r = 15.00: 397 clusters, similarity = 0.012
r = 30.00: 3 clusters, similarity = 0.182
r = 32.00: 3 clusters, similarity = 0.193
r = 33.00: 2 clusters, similarity = 0.243
```

```
r = 35.00: 1 clusters, similarity = 0.000

--- PCA-Reduced Data (2D) ---
Using bandwidth values: [1, 5, 10, 13, 15]
r = 1.00: 270 clusters, similarity = 0.316
r = 5.00: 12 clusters, similarity = 0.431
r = 10.00: 3 clusters, similarity = 0.646
r = 13.00: 3 clusters, similarity = 0.647
r = 15.00: 2 clusters, similarity = 0.572

Best bandwidth for Original Data: r = 33 (similarity = 0.243)
Best bandwidth for PCA Data: r = 13 (similarity = 0.647)
→ PCA-reduced data generally performs better for clustering
```

Question 3

Question 3, part 1

The function **lle(X, K=10, n_components=2, reg=1e-3)** implements the Locally Linear Embedding algorithm, which is a nonlinear dimensionality reduction technique using the following steps:

1. Takes high-dimensional data (X) and reduces it to a lower dimension (specified by n_components, default is 2)
2. For each point, it:
    - Finds K nearest neighbors (default K=10)
    - Computes weights that best reconstruct each point from its neighbors
    - Uses these weights to create a lower-dimensional embedding that preserves local relationships

As seen from the following algorithm uses, LLE is particularly useful for visualizing high-dimensional data while preserving the local structure of the manifold on which the data lies.

Question 3, part 2

The **main_3_2** function demonstrates the LLE algorithm on the former created datasets. For each dataset, it applies LLE to reduce the data to 2 dimensions and creates scatter plots of the results, saving them as PNG files.
As can be seen, LLE function well on the manifold based sets.

Question 3, part 3

The **main_3_3** function analyzes MNIST digits dataset (containing only digits 1, 2, 4, and 7) by applying LLE with different K values (10, 15, 20, 25, 30), creating visualizations for each K value, and then performing clustering analysis using Mean Shift with various bandwidth values to evaluate the quality of the embeddings using silhouette scores -Similarity score and Ncut metrics.

As can be seen, LLE doen't perform that well on that type on non manifold data

Best K: 30, Best r: 0.04, Best Ncut Score: 0.53

```
Unset
Bandwidth(r): 0.01
Number of clusters: 24
Similarity(Silhouette Score): 0.367
Ncut Score: 14.2701
------
Bandwidth(r): 0.02
Number of clusters: 6
Similarity(Silhouette Score): 0.609
Ncut Score: 1.5837
------
Bandwidth(r): 0.03
Number of clusters: 4
Similarity(Silhouette Score): 0.639
Ncut Score: 0.9298
------
Bandwidth(r): 0.04
Number of clusters: 3
Similarity(Silhouette Score): 0.518
Ncut Score: 0.9773
------
Bandwidth(r): 0.05
Number of clusters: 2
Similarity(Silhouette Score): 0.383
Ncut Score: 0.6603
------
K: 10, r: 0.01, Number of clusters: 13, Ncut Score: 5.5589
------
K: 10, r: 0.02, Number of clusters: 7, Ncut Score: 3.0965
------
K: 10, r: 0.03, Number of clusters: 4, Ncut Score: 0.8055
------
K: 10, r: 0.04, Number of clusters: 3, Ncut Score: 0.8254
------
K: 10, r: 0.05, Number of clusters: 3, Ncut Score: 1.1432
```

```
------
K: 15, r: 0.01, Number of clusters: 25, Ncut Score: 12.809
------
K: 15, r: 0.02, Number of clusters: 8, Ncut Score: 2.8986
------
K: 15, r: 0.03, Number of clusters: 5, Ncut Score: 2.1004
------
K: 15, r: 0.04, Number of clusters: 3, Ncut Score: 0.9795
------
K: 15, r: 0.05, Number of clusters: 2, Ncut Score: 0.6301
------
K: 20, r: 0.01, Number of clusters: 23, Ncut Score: 13.3965
------
K: 20, r: 0.02, Number of clusters: 6, Ncut Score: 1.5837
------
K: 20, r: 0.03, Number of clusters: 4, Ncut Score: 0.9298
------
K: 20, r: 0.04, Number of clusters: 3, Ncut Score: 0.9773
------
K: 20, r: 0.05, Number of clusters: 2, Ncut Score: 0.6603
------
K: 25, r: 0.01, Number of clusters: 48, Ncut Score: 27.4405
------
K: 25, r: 0.02, Number of clusters: 9, Ncut Score: 3.9055
------
K: 25, r: 0.03, Number of clusters: 3, Ncut Score: 1.2836
------
K: 25, r: 0.04, Number of clusters: 1 - Skipping (insufficient clusters)
------
K: 25, r: 0.05, Number of clusters: 1 - Skipping (insufficient clusters)
------
K: 30, r: 0.01, Number of clusters: 59, Ncut Score: 29.693
------
K: 30, r: 0.02, Number of clusters: 13, Ncut Score: 6.8815
------
K: 30, r: 0.03, Number of clusters: 5, Ncut Score: 2.2043
------
K: 30, r: 0.04, Number of clusters: 2, Ncut Score: 0.5331
------
K: 30, r: 0.05, Number of clusters: 1 - Skipping (insufficient clusters)
------
Best K: 30, Best r: 0.04, Best Ncut Score: 0.5331383199915276
```

Question 3, part 4

The specified constraint $$\frac{1}{n}\sum_{i=1}^{n} \boldsymbol{y}_i \boldsymbol{y}_i^\top = \boldsymbol{I},$$ * the equations was taken from the slides we learned
where:

- Yi is the low-dimensional embedding of the i-th data point.
- N is the number of data points.
- I is the d×d identity matrix.

says: the average outer product of the embeddings equals the identity matrix.

And used to prevent degenerate solution and ensure a meaningful embedding in the algorithm second step because:
Our minimization function is:

$$\underset{\boldsymbol{Y}}{\text{minimize}} \quad \sum_{i=1}^{n} \left|\left| \boldsymbol{y}_i - \sum_{j=1}^{n} w_{ij} \boldsymbol{y}_j \right|\right|_2^2,$$

And the minimal trivial solution is all Yi values = 0, causing the cost function to be equal to zero as well. By defining the above constraint we are preventing this trivial non-informative solution. And also validating that the embeddings have unit variance ensuring uncorrelated spread out dimensions, that aren't collapse to a single point by forcing the embedding to span all directions equally — with no dimension dominates.

Question 4

Question 4, part 1

The **sne** function implements dimensionality reduction while preserving the pairwise similarities between data points. It uses a Gaussian distribution for similarity calculations, includes momentum-based optimization, and runs for a specified number of iterations (default 1000) with a given perplexity (default 30.0) and learning rate (default 10.0).

Question 4, part 2

The **main_4_2** function demonstrates the t-SNE algorithm on created former datasets. For each dataset, it applies t-SNE to reduce the data to 2 dimensions and creates scatter plots of the results, saving them as PNG files. All the plots available on the exercise directory.
As can be seen from the plots, the t-SNE algorithm doesn't perform well on the manifold based datasets.
Output:

```
Unset
Swiss roll dataset shape: (1000, 3)
Running t-SNE (v=1)...
Iteration 0, KL divergence: 3.1946
Iteration 100, KL divergence: 0.9500
Iteration 200, KL divergence: 0.7941
Iteration 300, KL divergence: 0.6989
Iteration 400, KL divergence: 0.6337

Sinusoidal curve dataset shape: (721, 3)
Running t-SNE (v=1)...
Iteration 0, KL divergence: 2.8843
Iteration 100, KL divergence: 0.5093
Iteration 200, KL divergence: 0.3801
Iteration 300, KL divergence: 0.3303
Iteration 400, KL divergence: 0.3067
```

Question 4, part 3

The main_4_3 function compares SNE and t-SNE with varying degrees of freedom on MNIST digits dataset (containing only digits 1, 2, 4, and 7). It applies each technique to reduce the data to 2 dimensions and creates a 2x2 grid of scatter plots to visualize the results, saving the comparison as a PNG file.
It can be seen that these algorithms perform very well on thia task, creating well defined clusters of digits.

Output:

```
Unset
D_1_2_4_7 dataset shape: (719, 64)
Running SNE...
Iteration 0, KL divergence: 3.0790
Iteration 100, KL divergence: 3.0790
Iteration 200, KL divergence: 3.0070
```

```
Iteration 300, KL divergence: 1.2789
Iteration 400, KL divergence: 1.0080

Running t-SNE (v=1)...
Iteration 0, KL divergence: 3.0790
Iteration 100, KL divergence: 0.8670
Iteration 200, KL divergence: 0.7468
Iteration 300, KL divergence: 0.6668
Iteration 400, KL divergence: 0.6117

Running t-SNE (v=3)...
Iteration 0, KL divergence: 3.0790
Iteration 100, KL divergence: 0.4974
Iteration 200, KL divergence: 0.4737
Iteration 300, KL divergence: 0.4658
Iteration 400, KL divergence: 0.4628

Running t-SNE (v=10)...
Iteration 0, KL divergence: 3.0790
Iteration 100, KL divergence: 0.6546
Iteration 200, KL divergence: 0.6277
Iteration 300, KL divergence: 0.6169
Iteration 400, KL divergence: 0.6143
```