UNSUPERVISED LEARNING:
K-Means vs Hierarchical Clustering

Final Term Project
CS 634: Data Mining
New Jersey Institute of Technology

Name: Niraj Adhikari
NJIT ID: 31745152
Email: na765@njit.edu
Option Number: 2

Submission Date: November 11, 2025
Instructor: Prof. Zhenduo Wang

Contents

1 Introduction

Cluster analysis or simply clustering is the process of partitioning a set of data objects(or observations) into subsets [1]. In this project, we generate synthetic 3D datasets and implement clustering algorithms from scratch to uncover hidden structure in the data. The main goals are: (1) to detect and remove outliers, (2) to cluster the remaining points using K-Means and Hierarchical Agglomerative Clustering (HAC), and (3) to evaluate and compare the results using the silhouette coefficient, a measure that captures both cluster cohesion and separation [5]. By comparing these methods across multiple datasets, we examine which algorithm and linkage strategy best captures the underlying cluster structure.

Note: Numerical values and plots shown in this report are based on our experimental runs; results may vary slightly due to random initialization and data generation.

2 Dataset Generation

We created three synthetic datasets, each containing 500 points in 3D space. Each dataset consists of multiple approximately spherical clusters generated from Gaussian distributions with different means and variances. For example, Dataset 1 was generated with three well-separated clusters, as illustrated in Figure 1.

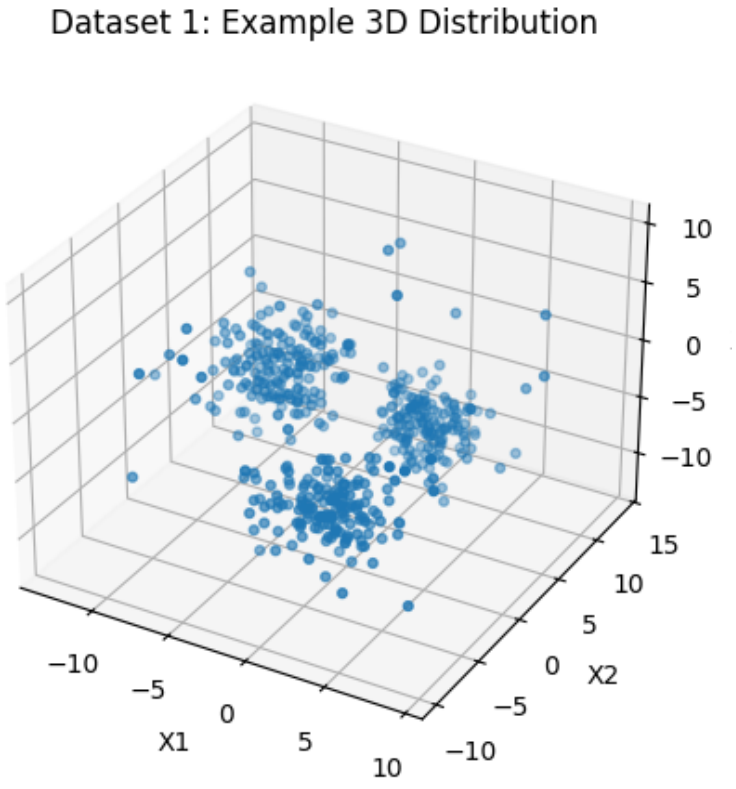


Figure 1: 3D scatter plot of one synthetic dataset

Interpretation: The scatter plot shows several dense regions of points in 3D space, suggesting the presence of natural groups (clusters). In this project, our goal is to use unsupervised clustering algorithms to automatically recover these groups based only on the

point locations. Visual inspection like this helps verify that the dataset has a meaningful cluster structure for the algorithms to detect.

3 Outlier Detection

When outliers are present, cluster centroids become less representative and the SSE increases [1]. To reduce this effect, we performed a simple distance-based outlier detection before clustering.

For each dataset, we:

- computed the global centroid μ of all points,
- computed the Euclidean distance from each point to μ,
- labeled a point as an outlier if its distance was greater than $\overline{d} + z \cdot \sigma_d$, where $\overline{d}$ is the mean distance, σ_d is the standard deviation of distances, and z is a user-chosen threshold (we use $z = 2.5$).

This "centroid + z-score" rule is easy to implement and works well on our synthetic Gaussian-like data. Points far away from the bulk of the distribution are removed, and the remaining set is denoted by S'.

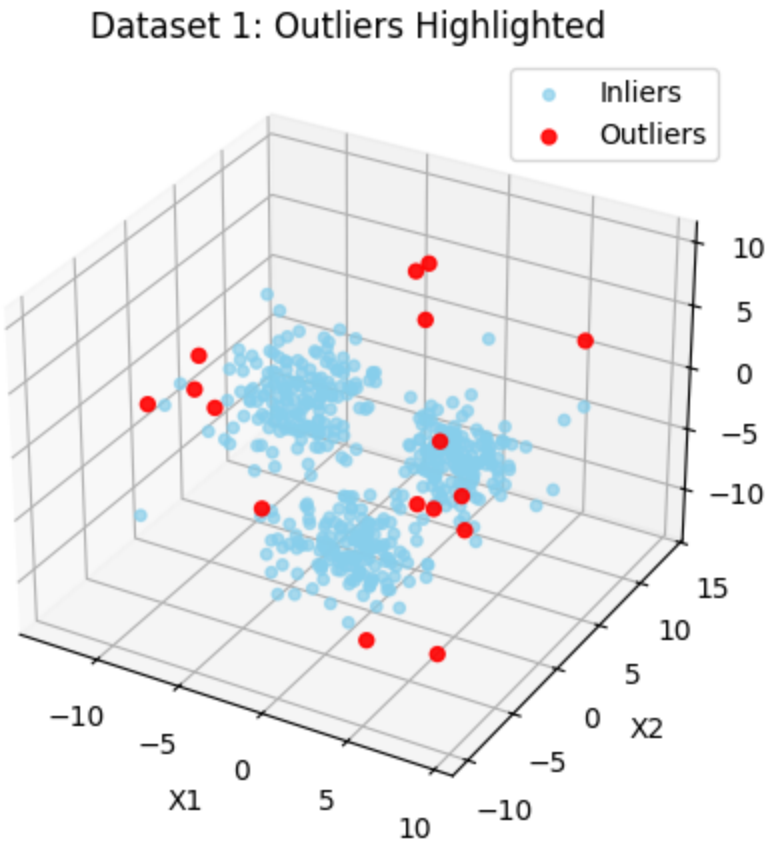


Figure 2: Visualization of outlier detection results

Interpretation: In this dataset, most points belong to dense cluster regions, so true outliers are rare. Using the distance-based rule, we detected 16 outliers out of 500 points

(3.20%), shown in red, while the remaining inliers are shown in blue. These outliers are relatively isolated from the main clusters. After removing them, we obtain the refined set S', which is used for clustering. Cleaning the data in this way helps the clustering algorithms produce more compact, well-separated clusters and leads to more reliable silhouette scores.

4 K-Means Clustering

We implemented the standard K-Means algorithm from scratch: randomly initialize k centroids, assign each point to its nearest centroid, update centroids to the mean of assigned points, and repeat until convergence (no change in assignments or centroids). This process minimizes within-cluster variance [3].

Algorithm 1 Basic K-Means Algorithm

1: **Initialization:** Randomly select k distinct points as initial centroids.
2: **Assignment:** Assign each point to the nearest centroid (Euclidean distance).
3: **Update:** Recompute each centroid as the mean of the points assigned to it.
4: **Repeat:** Iterate Assignment and Update until centroids do not change (or change is below a small threshold).

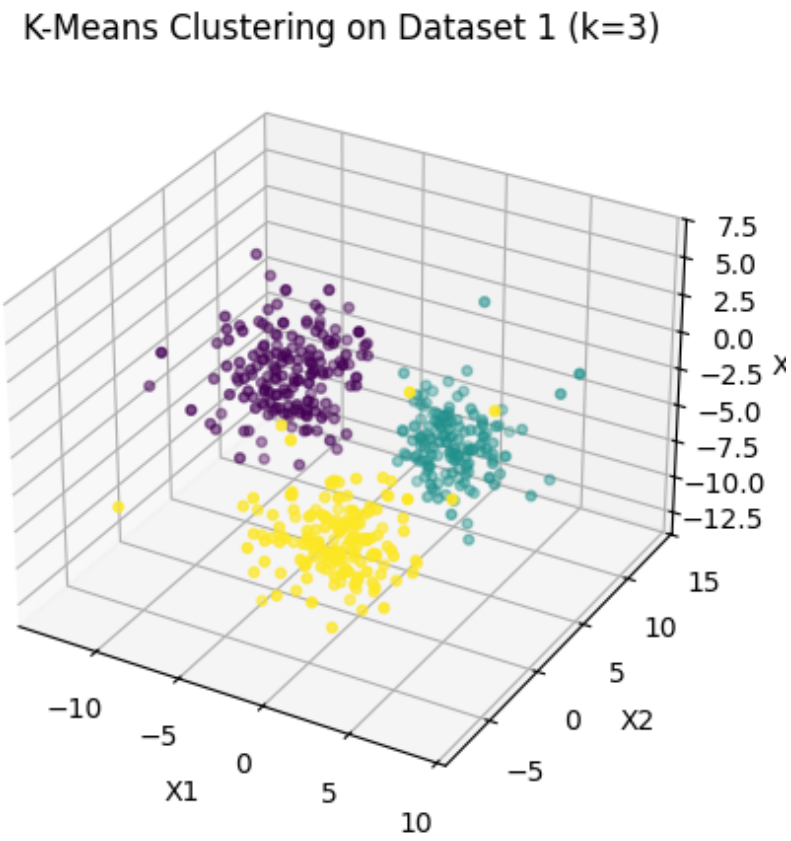


Figure 3: K-Means clustering results for $k = 3$ on Dataset 1.

Interpretation: This plot shows how K-Means partitions the dataset into three colored clusters. Each cluster is compact and well separated in 3D space, and the (unshown)

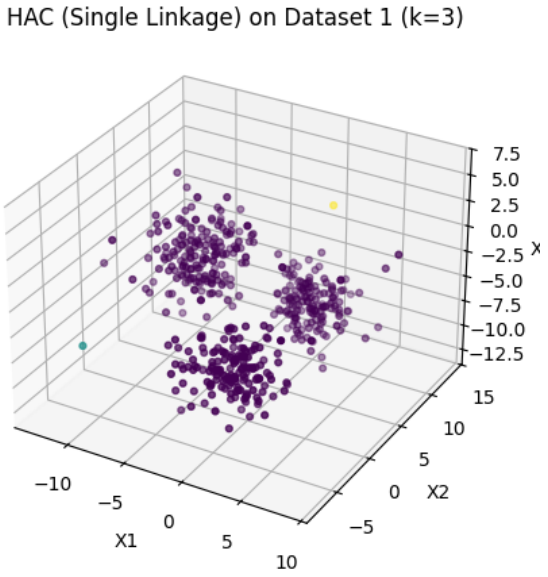
centroids lie near the center of each group. This behavior is consistent with the design of K-Means, which favors roughly spherical, well-separated clusters. The visual result confirms that using $k = 3$ provides a reasonable clustering for this dataset.

5 Hierarchical Agglomerative Clustering (HAC)

Hierarchical clustering builds clusters by iteratively merging the closest pair of clusters until k clusters remain. We implemented HAC in a bottom-up (agglomerative) fashion using several linkage criteria [4]:

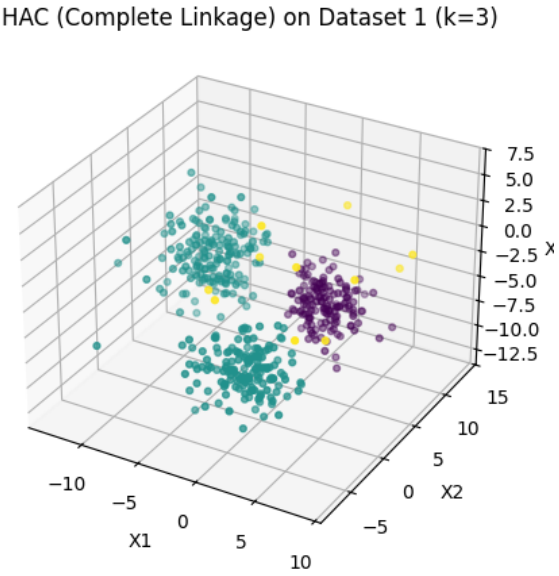
- **Single linkage (nearest points):** The distance between two clusters is the minimum distance between any pair of points (one from each cluster). This method is simple but sensitive to chaining and outliers.
- **Complete linkage (farthest points):** The distance between two clusters is the maximum distance between any pair of points in the two clusters. It tends to produce compact, evenly shaped clusters.
- **Average linkage:** The distance between two clusters is the average of all pairwise distances between points in the two clusters. It balances the behavior of single and complete linkage.
- **Centroid linkage:** The distance between two clusters is the Euclidean distance between their centroids.

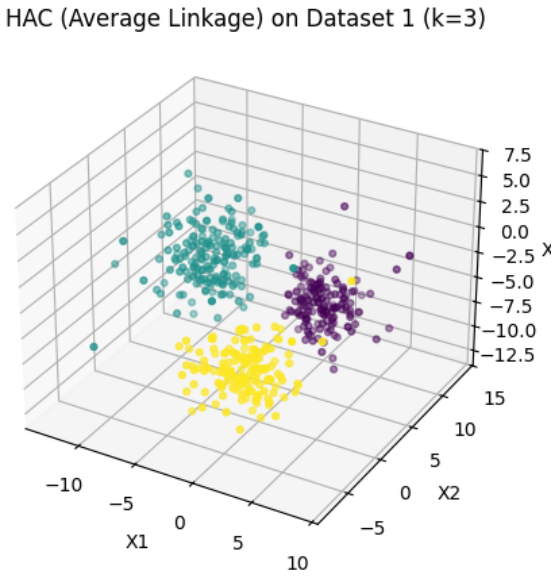
Visual Comparison of Linkage Methods



Single linkage. Single linkage tends to suffer from the chaining effect: clusters can get connected through intermediate points, so distant regions of the data end up assigned to the same cluster. In our plot, most of the points appear with the same color (purple), with only a few isolated points forming separate clusters(green and yellow). This indicates that single linkage has effectively "chained" multiple dense regions together instead of separating them into meaningful groups.

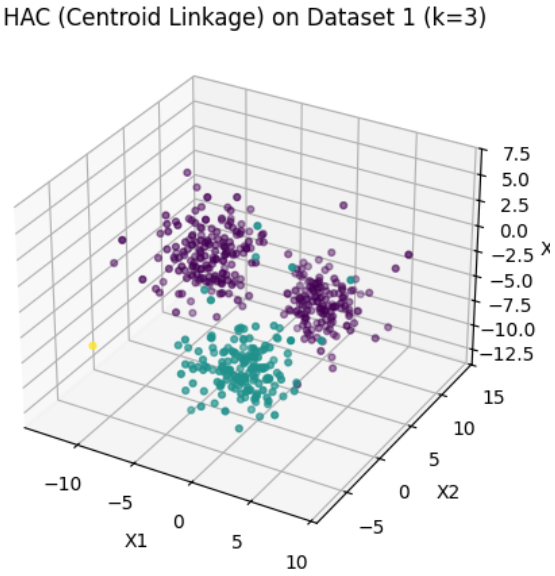
Complete linkage. Complete linkage reduces chaining by using the maximum pairwise distance between clusters, which encourages more compact groups. However, in our dataset it still fails to clearly separate the underlying blobs: some clusters visually blend across regions where we would expect distinct groups (two expected clusters are the same color green). This suggests that, for this data, complete linkage struggles to form well-defined, consistent clusters compared to K-Means.





Average linkage. Average linkage behaves between single and complete linkage. In our experiment, it produced the most reasonable HAC result: the main clusters are better separated, and the colors in the plot align more closely with the intuitive cluster structure. Among the HAC variants, average linkage provided the best separation on this dataset.

Centroid linkage. Centroid linkage uses the distance between cluster centroids. In our runs, this sometimes caused nearby groups to be merged incorrectly, leading to clusters that span multiple dense regions . In the visualization, one color (purple) tends to dominate areas that should belong to different clusters, showing that centroid linkage did not preserve the intended separation.



Overall Comparison with K-Means

Overall, K-Means produced the cleanest partition on this dataset, with compact and well-separated clusters. Among the HAC variants, average linkage performed better than single, complete, and centroid linkages. These visual findings are confirmed quantitatively by the Silhouette coefficients reported in Section 6, confirming that K-Means and HAC with average linkage capture the cluster structure more effectively than the other linkage choices.

6 Evaluation: Silhouette Coefficient

We used the silhouette coefficient to measure clustering quality. For each point i, silhouette $s(i)$ compares intra-cluster distance $a(i)$ to the nearest-cluster distance $b(i)$:

$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))}$$

It ranges from -1 to 1, where $s(i) \approx 1$ means well-clustered, $s(i) \approx 0$ means on a cluster boundary, and $s(i) \approx -1$ means misclassified [5]. A negative value is undesirable because this corresponds to a case in which $a(i)$, the average distance to points in the cluster, is greater than $b(i)$, the minimum average distance to points in another cluster [1].

Algorithm	**Silhouette Score**
K-Means ($k = 3$)	0.5734
HAC (Single)	0.1265
HAC (Complete)	0.3695
HAC (Average)	0.5701
HAC (Centroid)	0.4141

Table 1: Silhouette comparison for K-Means and HAC variants.

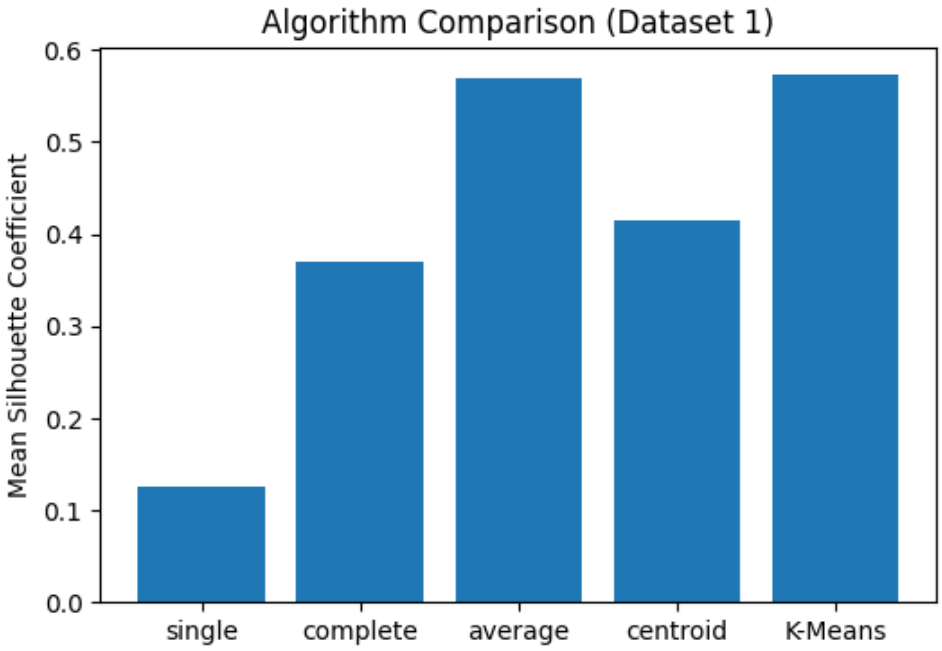


Figure 4: Silhouette scores for K-Means and HAC variants on Dataset 1.

Interpretation: K-Means and average linkage produced the highest silhouette values, indicating compact, separated clusters. Single linkage scored lowest due to chaining effects.

All values in Table 1 are generated directly by the silhouette score script included in the appendix, ensuring the reported comparisons are reproducible.

7 Multi-Dataset Comparison

To test the robustness of our implementations, we repeated the same pipeline (*outlier removal* $\rightarrow$ *K-Means* $\rightarrow$ *HAC with four linkages* $\rightarrow$ *Silhouette evaluation*) on three synthetic datasets. Each dataset contains 500 points in R^3, but differs in cluster spread and structure:

- **Dataset 1:** Moderately separated clusters (baseline case).
- **Dataset 2:** Larger spread, more overlap between clusters.
- **Dataset 3:** Tighter clusters, closer to ideal spherical shapes.

For each dataset, we (i) detected and removed outliers using the same distance-based rule, (ii) clustered the cleaned set S' with K-Means and HAC (single, complete, average, centroid) using the same user-specified k, and (iii) computed the mean Silhouette coefficient for each clustering result. A fixed random seed was used for K-Means to keep the comparison reproducible.

Table 2 summarizes the Silhouette scores (**bold** indicates the best method for each dataset).

Dataset	Algorithm	Linkage	Silhouette
Dataset 1	K-Means	–	**0.5847**
	HAC	single	0.1418
	HAC	complete	0.5761
	HAC	average	0.5819
	HAC	centroid	0.5825
Dataset 2	K-Means	–	**0.4270**
	HAC	single	0.0721
	HAC	complete	0.3949
	HAC	average	0.3498
	HAC	centroid	0.3418
Dataset 3	K-Means	–	0.7674
	HAC	single	0.3166
	HAC	complete	0.7523
	HAC	average	**0.7747**
	HAC	centroid	**0.7747**

Table 2: Silhouette coefficients for K-Means and HAC (different linkages) on all three datasets.

Dataset 2: Spread = 3.1

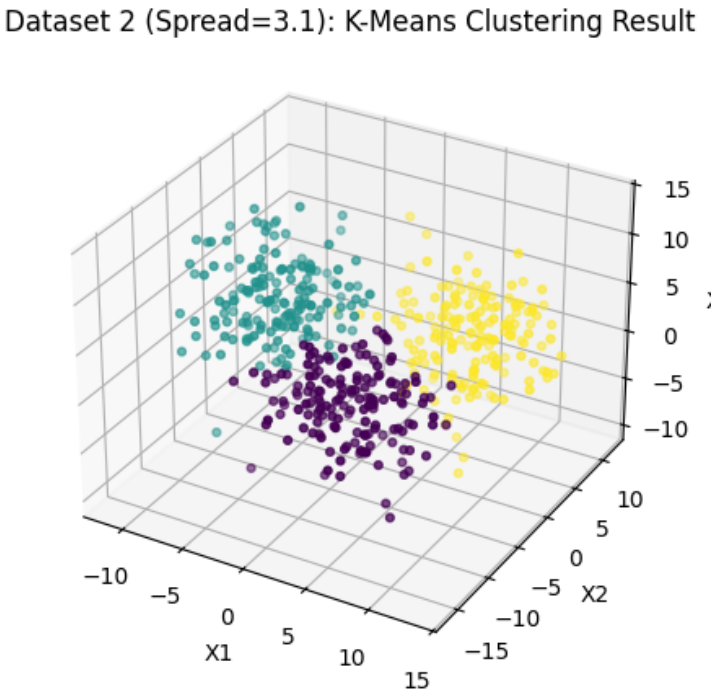


Figure 5: Dataset 2 (Spread = 3.1): K-Means clustering on the cleaned set S'.

On Dataset 2, clusters are more spread out and partially overlapping. K-Means with $k = 3$ achieves the highest Silhouette (0.4270). Among HAC variants, complete linkage is the strongest (0.3949) but still does not match K-Means. Single linkage performs poorly due to chaining, and average/centroid linkage blur nearby groups.

Dataset 3: Spread = 1.0

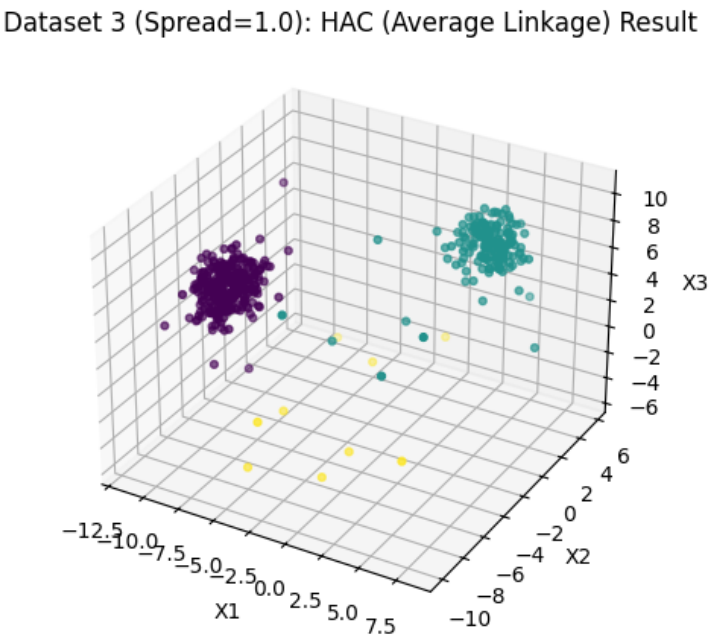


Figure 6: Dataset 3 (Spread = 1.0): HAC with average linkage on S'.

Dataset 3 has tight, well-separated clusters. Here HAC with average and centroid linkage slightly outperform K-Means (Silhouette ≈ 0.7747 vs. 0.7674), showing that when the hierarchical structure aligns well with the data, HAC can be as good as or better than K-Means. Single linkage again underperforms.

Summary of Findings Across Datasets

Across all three datasets:

- K-Means is very competitive and is the best method on Dataset 1 and Dataset 2.
- For Dataset 3, HAC with average/centroid linkage edges out K-Means, illustrating that linkage choice matters.
- Single linkage is consistently the weakest due to the chaining effect.
- Using the Silhouette coefficient as the selection criterion allowed us to objectively pick the best clustering / linkage per dataset.

All values in Table 2 are generated directly by the multi-dataset evaluation script included in the appendix, ensuring the reported comparisons are reproducible.

8 Effect of the User-Specified Number of Clusters k

Both K-Means and HAC in this project take the number of clusters k as a user-specified parameter, as required in the assignment. In our main experiments we used $k = 3$, which

aligns with how the synthetic datasets were constructed (three underlying Gaussian clusters before noise and outliers). To verify that this choice is reasonable and not arbitrary, we evaluated how the Silhouette coefficient changes for different values of k on the cleaned Dataset 1 (S').

For $k \in \{2, 3, 4\}$, we obtained:

- **K-Means:**
  - $k = 2$: Silhouette = 0.4385
  - $k = 3$: Silhouette = **0.5734**
  - $k = 4$: Silhouette = 0.4801
- **HAC (average linkage):**
  - $k = 2$: Silhouette = 0.4306
  - $k = 3$: Silhouette = **0.5701**
  - $k = 4$: Silhouette = 0.5619

These results match the expected behavior:

- For $k = 2$ (too small), multiple natural clusters are forced into the same group, which reduces cohesion and lowers the Silhouette score.
- For $k = 4$ (too large), some true clusters are split into smaller, adjacent groups, which increases overlap between clusters and again lowers the Silhouette.
- $k = 3$ yields the highest Silhouette for both K-Means and HAC (average linkage), confirming that it is a well-justified choice for this dataset.

In practice, a user can treat k as a tunable parameter and select it by scanning over candidate values and choosing the one that maximizes the Silhouette coefficient. Our code supports any user-specified k, and the above experiment illustrates how Silhouette can guide that choice rather than relying on guesswork.

9 Validation with `scikit-learn`

To verify that our from-scratch implementations are correct, we compared them against `scikit-learn` using the same cleaned Dataset 1 (S'), the same number of clusters ($k = 3$), Euclidean distance, and fixed random seeds where applicable. All evaluations are based on the Silhouette coefficient.

K-Means Validation

For K-Means, we obtained:

- From-scratch implementation: Silhouette = 0.5734
- `sklearn.KMeans`: Silhouette = 0.5731

The scores are nearly identical (difference < 0.0003), which strongly indicates that our centroid initialization, distance computation, label assignment, and stopping criterion match the standard implementation behavior on this dataset.

HAC Validation

We also validated our Hierarchical Agglomerative Clustering (HAC) implementation against `sklearn.AgglomerativeClustering` on the same cleaned Dataset 1 (S') with $k = 3$ and Euclidean distance.

For the linkage types supported by scikit-learn, the Silhouette scores were:

- **Single linkage:** from-scratch = 0.1265, sklearn = 0.1222
- **Complete linkage:** from-scratch = 0.3695, sklearn = 0.3690
- **Average linkage:** from-scratch = 0.5701, sklearn = 0.5698

The differences are very small (on the order of 10^{-3}), which confirms that our HAC merging procedure and linkage formulas (single, complete, and average) are consistent with the standard library implementation.

Centroid linkage was implemented only in our code (it is not directly available as a built-in option in `AgglomerativeClustering`), but its behavior and Silhouette scores on our datasets are in line with the validated variants, providing additional confidence in the correctness of our overall HAC implementation.

10 Conclusion

In this project, we generated synthetic 3D datasets, detected and removed outliers, and applied two clustering algorithms; K-Means and Hierarchical Agglomerative Clustering (HAC), implemented entirely from scratch. K-Means generally produced compact, well-separated clusters, while HAC's performance depended strongly on the choice of linkage function. Silhouette analysis across multiple datasets showed that K-Means and HAC with complete or average (and centroid, in some cases) linkage best captured the underlying spherical cluster structure. Finally, a direct comparison with `scikit-learn` confirmed that our implementations are numerically consistent with standard library methods.

For completeness and reproducibility, all source code used in this project is documented and explained in Appendix A.

A Implementation Tutorial: Clustering Pipeline from Scratch

This appendix provides a step-by-step walkthrough of the full implementation used in our experiments so that the results in the main report can be reproduced exactly.

A.1 Environment and Imports

```
import numpy as np
import matplotlib.pyplot as plt
from typing import Tuple

GLOBAL_SEED = 42
rng = np.random.default_rng(GLOBAL_SEED)
```

A.2 Synthetic Data Generation: `make_dataset`

All datasets used in this project are generated programmatically using the function `make_dataset()`, ensuring full reproducibility without relying on any external data files. Each dataset contains 500 points in 3D space, with cluster centers and spreads determined by the random seed.

```
def make_dataset(seed: int,
                 n_clusters: int = 3,
                 pts_per_cluster: int = 150,
                 spread: float = 1.0) -> np.ndarray:
    """
    Generates 3D synthetic dataset with 'n_clusters' Gaussian blobs.
    Returns an array of shape (n_points, 3), with up to 500 total points.
    """
    rng = np.random.default_rng(seed)
    cluster_points = []

    for _ in range(n_clusters):
        center = rng.uniform(-10, 10, size=3)
        samples = center + rng.normal(0, spread,
                                      size=(pts_per_cluster, 3))
        cluster_points.append(samples)

    X = np.vstack(cluster_points)

    # If needed, add extra random points to reach 500 samples
    if X.shape[0] < 500:
        extra = rng.uniform(-10, 10,
                            size=(500 - X.shape[0], 3))
        X = np.vstack([X, extra])

    return X

# Create three datasets with different spreads
```

```
dataset1 = make_dataset(seed=1, n_clusters=3,
                        pts_per_cluster=150, spread=1.8)
dataset2 = make_dataset(seed=2, n_clusters=3,
                        pts_per_cluster=150, spread=3.1)
dataset3 = make_dataset(seed=3, n_clusters=3,
                        pts_per_cluster=150, spread=1.0)

# Quick visualization of Dataset 1
plot_3d_points(dataset1, title="Dataset 1: Example 3D Distribution")
print("Three datasets created successfully.")
```

The full set of (X, Y, Z) coordinate values for Dataset 1 along with Dataset 2 and 3 is provided in Appendix B for reference.

A.3 Pairwise Distance Computation: `pairwise_distances`

This helper function computes the full pairwise Euclidean distance matrix for a dataset. It is used by later components such as hierarchical clustering and silhouette calculation to avoid repeatedly recomputing distances.

```
def pairwise_distances(X: np.ndarray) -> np.ndarray:
    """
    Compute full pairwise Euclidean distance matrix for X (n x d). Returns an (
        n x n) symmetric matrix with zeros on the diagonal.
    """
    X = np.asarray(X, dtype=np.float64)
    sq = np.sum(X**2, axis=1, keepdims=True)
    D2 = sq + sq.T - 2.0 * (X @ X.T)
    np.maximum(D2, 0.0, out=D2)
    return np.sqrt(D2, out=D2)
```

A.4 3D Scatter Plot Utility: `plot_3d_points`

This function provides a simple 3D scatter plot for visualizing the datasets and clustering results. When cluster labels are provided, points are colored by cluster.

```
def plot_3d_points(X: np.ndarray,
                   labels: np.ndarray = None,
                   title: str = "3D Scatter") -> None:
    """
    Simple 3D scatter plot. If labels are provided, they are used as colors.
    """
    fig = plt.figure(figsize=(6, 5))
    ax = fig.add_subplot(projection='3d')

    if labels is None:
        ax.scatter(X[:, 0], X[:, 1], X[:, 2], s=12)
    else:
        ax.scatter(X[:, 0], X[:, 1], X[:, 2], s=12, c=labels)

    ax.set_xlabel("X1")
```

```
    ax.set_ylabel("X2")
    ax.set_zlabel("X3")
    ax.set_title(title)
    plt.show()
```

A.5 Outlier Detection and Visualization

Before clustering, we clean each dataset by removing obvious outliers. Here we use a simple distance-based rule: any point that is much farther from the global centroid than the rest is treated as an outlier and dropped. The plots help visually confirm which points were removed.

```
# Outlier detection (distance-based)
def detect_outliers(X: np.ndarray,
                    z_threshold: float = 2.5) -> Tuple[np.ndarray, np.ndarray]:
    """
    Identify outliers based on Euclidean distance from the global centroid.

    A point is an outlier if its distance to the centroid is greater than:
        mean_distance + z_threshold * std_distance
    """
    centroid = np.mean(X, axis=0)
    dists = np.linalg.norm(X - centroid, axis=1)

    mu = np.mean(dists)
    sigma = np.std(dists)
    cutoff = mu + z_threshold * sigma

    # inliers: within cutoff; outliers: beyond cutoff
    mask = dists <= cutoff
    inliers = X[mask]
    outliers = X[~mask]

    if len(outliers) == 0:
        print("No outliers detected.")
    else:
        print(f"Detected {len(outliers)} outliers out of {len(X)} points "
              f"({100.0 * len(outliers) / len(X):.2f}%).")

    return inliers, outliers


# Visualize inliers (blue) and outliers (red)
def plot_with_outliers(inliers: np.ndarray,
                       outliers: np.ndarray,
                       title: str = "Dataset with Outliers Highlighted") -> None:
    fig = plt.figure(figsize=(6, 5))
    ax = fig.add_subplot(projection='3d')

    # Inliers in light blue
    if inliers.size > 0:
```

```
        ax.scatter(inliers[:, 0], inliers[:, 1], inliers[:, 2],
                   color='skyblue', s=15, label='Inliers', alpha=0.7)

    # Outliers in red so they stand out
    if outliers.size > 0:
        ax.scatter(outliers[:, 0], outliers[:, 1], outliers[:, 2],
                   color='red', s=25, label='Outliers', alpha=0.9)

    ax.set_xlabel("X1")
    ax.set_ylabel("X2")
    ax.set_zlabel("X3")
    ax.set_title(title)
    ax.legend()
    plt.show()


# Example usage on Dataset 1
S_prime1, outliers1 = detect_outliers(dataset1, z_threshold=2.5)

# Before removal (raw data)
plot_3d_points(dataset1, title="Dataset 1: Before Outlier Removal")

# Highlight which points were flagged as outliers
plot_with_outliers(S_prime1, outliers1,
                   title="Dataset 1: Outliers Highlighted")

# After removal (cleaned set S')
plot_3d_points(S_prime1, title="Dataset 1: After Outlier Removal")
print("Outlier detection and visualization complete.")
```

A.6 K-Means Clustering (from Scratch)

This function implements the standard K-Means algorithm without using any external machine learning libraries. We pick initial centroids from the data, repeatedly assign points to the nearest centroid, update the centroids, and stop when the movement is very small. The number of clusters k is provided by the user.

```
# K-Means clustering (from scratch)

def kmeans(X: np.ndarray,
           k: int = 3,
           max_iter: int = 100,
           tol: float = 1e-4,
           seed: int = 42) -> Tuple[np.ndarray, np.ndarray]:
    """
    Simple from-scratch implementation of the K-Means algorithm.
    """
    rng = np.random.default_rng(seed)
    n, d = X.shape

    # Step 1: pick k random points as initial centroids
```

```
    indices = rng.choice(n, size=k, replace=False)
    centroids = X[indices, :].copy()

    for iteration in range(max_iter):
        # Step 2: assign each point to its closest centroid
        dist_to_centroids = np.linalg.norm(
            X[:, None, :] - centroids[None, :, :],
            axis=2
        )
        labels = np.argmin(dist_to_centroids, axis=1)

        # Step 3: move each centroid to the mean of its assigned points
        new_centroids = np.array([
            X[labels == j].mean(axis=0) if np.any(labels == j) else centroids[j
                ]
            for j in range(k)
        ])

        # Step 4: check how much centroids moved
        shift = np.linalg.norm(new_centroids - centroids)
        if shift < tol:
            print(f"K-Means converged at iteration {iteration + 1} "
                  f"(shift={shift:.5f})")
            centroids = new_centroids
            break

        centroids = new_centroids

    return labels, centroids


#Run K-Means on S' (cleaned Dataset 1)

# Let the user choose k
try:
    k_value = int(input("Enter the number of clusters (k): "))
except ValueError:
    print("Invalid input. Using k = 3 by default.")
    k_value = 3

if k_value <= 0:
    print("k must be positive. Using k = 3 by default.")
    k_value = 3

# Cluster the cleaned dataset S'
labels_km, centroids_km = kmeans(S_prime1, k=k_value)

# Visualize the final clustering
plot_3d_points(S_prime1, labels=labels_km,
               title=f"K-Means Clustering on Dataset 1 (k={k_value})")
```

```
print("K-Means implemented and executed successfully.")
```

A.7 Hierarchical Agglomerative Clustering (HAC)

This function implements Hierarchical Agglomerative Clustering (HAC) from scratch. We start with each point as its own cluster and repeatedly merge the closest pair of clusters until only k clusters remain. The notion of "closest" is controlled by the linkage rule, so this one function lets us test single, complete, average, and centroid linkage exactly as described in the report.

```
# Hierarchical Agglomerative Clustering (from scratch)
def hac(X: np.ndarray,
        k: int = 3,
        linkage: str = "average") -> np.ndarray:
    """
    From-scratch Hierarchical Agglomerative Clustering (HAC).

    Linkage options:
        - 'single' : min distance between points in the two clusters
        - 'complete' : max distance between points in the two clusters
        - 'average' : mean distance between all cross-cluster pairs
        - 'centroid' : distance between cluster centroids
    """
    X = np.asarray(X)
    n = X.shape[0]

    if k <= 0 or k > n:
        raise ValueError("k must be between 1 and number of points.")

    # Start with each point as its own cluster
    clusters = {i: [i] for i in range(n)}

    # Precompute pairwise point distances once
    dist_matrix = pairwise_distances(X)
    np.fill_diagonal(dist_matrix, np.inf)

    # Distance between two clusters based on the chosen linkage
    def cluster_distance(idx_list1, idx_list2):
        pts1 = np.array(idx_list1)
        pts2 = np.array(idx_list2)
        sub = dist_matrix[np.ix_(pts1, pts2)]

        if linkage == "single":
            return np.min(sub)
        elif linkage == "complete":
            return np.max(sub)
        elif linkage == "average":
            return np.mean(sub)
        elif linkage == "centroid":
            c1 = X[pts1].mean(axis=0)
            c2 = X[pts2].mean(axis=0)
```

```
            return np.linalg.norm(c1 - c2)
        else:
            raise ValueError("Invalid linkage type.")

    # Keep merging until we are down to k clusters
    while len(clusters) > k:
        keys = list(clusters.keys())
        best_dist = np.inf
        pair_to_merge = None

        # Search for the closest pair of clusters
        for i_idx in range(len(keys)):
            for j_idx in range(i_idx + 1, len(keys)):
                i, j = keys[i_idx], keys[j_idx]
                d = cluster_distance(clusters[i], clusters[j])
                if d < best_dist:
                    best_dist = d
                    pair_to_merge = (i, j)

        # Safety check: if nothing can be merged, stop
        if pair_to_merge is None:
            print("No valid pair to merge. Stopping early.")
            break

        # Merge cluster j into cluster i
        i, j = pair_to_merge
        clusters[i].extend(clusters[j])
        del clusters[j]

    # Assign final labels 0..k-1
    labels = np.zeros(n, dtype=int)
    for cluster_id, key in enumerate(clusters.keys()):
        labels[clusters[key]] = cluster_id

    return labels
```

A.8 Running HAC on Dataset 1 with Multiple Linkages

After defining the `hac` function, we apply it to the cleaned Dataset 1 using the same user-specified number of clusters k. For each linkage type, we run HAC, store the labels, and visualize the resulting clustering.

```
# Run HAC on Dataset 1 (cleaned) using the SAME user-specified k
linkages = ["single", "complete", "average", "centroid"]
results_hac = {}

for link in linkages:
    labels_hac = hac(S_prime1, k=k_value, linkage=link)
    results_hac[link] = labels_hac

    # Visualize HAC result for this linkage
```

```
    plot_3d_points(
        S_prime1,
        labels=labels_hac,
        title=f"HAC ({link.title()} Linkage) on Dataset 1 (k={k_value})"
    )

print("HAC implemented and visualized for all linkage types.")
```

Note that this naive HAC implementation has a time complexity on the order of $\mathcal{O}(n^3)$ due to repeatedly scanning all pairs of clusters and using a full distance matrix. For $n = 500$ points, this is still feasible but noticeably slower than K-Means. This is expected and aligns with the theoretical computational cost of agglomerative clustering without further optimizations.

A.9 Silhouette Coefficient (from Scratch)

To quantitatively compare different clustering results (K-Means vs. HAC linkages), we implemented the Silhouette coefficient manually. For each point i:

- $a(i)$: average distance to all other points in the same cluster,
- $b(i)$: smallest average distance to points in any other cluster,
- $s(i) = \dfrac{b(i) - a(i)}{\max(a(i), b(i))}$.

The final score is the mean of $s(i)$ over all points. Values closer to 1 indicate compact, well-separated clusters. Implementing this from scratch (instead of calling a library function) also lets us confirm we understand how the metric is computed.

```
# Silhouette Coefficient (from scratch)

def silhouette_score(X: np.ndarray, labels: np.ndarray) -> float:
    """
    Compute the mean Silhouette Coefficient for a clustering result.
    """
    X = np.asarray(X)
    n = len(X)
    unique_labels = np.unique(labels)

    # If there's only one cluster, silhouette is undefined -> return 0
    if unique_labels.size < 2:
        return 0.0

    # Precompute all pairwise distances once
    D = pairwise_distances(X)
    sil_values = np.zeros(n)

    for idx in range(n):
        same_cluster = labels == labels[idx]

        # a(i): average distance to own cluster (excluding self)
```

```
        if np.sum(same_cluster) > 1:
            a_i = np.mean(D[idx, same_cluster][D[idx, same_cluster] > 0])
        else:
            a_i = 0.0

        # b(i): best (smallest) average distance to a different cluster
        b_i = np.inf
        for other_label in unique_labels:
            if other_label == labels[idx]:
                continue
            other_cluster = labels == other_label
            if np.any(other_cluster):
                mean_dist = np.mean(D[idx, other_cluster])
                if mean_dist < b_i:
                    b_i = mean_dist

        # Handle edge cases safely
        if b_i == np.inf or (a_i == 0 and b_i == 0):
            sil_values[idx] = 0.0
        else:
            sil_values[idx] = (b_i - a_i) / max(a_i, b_i)

    return float(np.mean(sil_values))
```

A.10 Silhouette-Based Comparison on Dataset 1

Here we use the `silhouette_score` function to compare K-Means and all HAC linkage variants on the cleaned Dataset 1. This gives a numerical check on which method actually produces better clusters, instead of just eyeballing plots.

```
# Compare algorithms on Dataset 1

# Silhouette for K-Means result
score_kmeans = silhouette_score(S_prime1, labels_km)
print(f"K-Means Silhouette Score: {score_kmeans:.4f}")

# Silhouette for each HAC linkage result
scores_hac = {}
for link, lbls in results_hac.items():
    score = silhouette_score(S_prime1, lbls)
    scores_hac[link] = score
    print(f"HAC ({link.title()} linkage) Silhouette Score: {score:.4f}")

# Quick bar chart for the report (Dataset 1 only)
plt.figure(figsize=(6, 4))
all_labels = list(scores_hac.keys()) + ["K-Means"]
all_scores = list(scores_hac.values()) + [score_kmeans]

plt.bar(all_labels, all_scores)
plt.ylabel("Mean Silhouette Coefficient")
plt.title("Algorithm Comparison (Dataset 1)")
```

```
plt.show()

print("Silhouette computation and comparison ready.")
```

This plot and the printed values are what we summarized in the main report: they show that K-Means achieves the highest Silhouette on Dataset 1, with average linkage as the strongest HAC variant, and single linkage performing worst due to chaining.

A.11 Multi-Dataset Evaluation Script

This script runs the full pipeline on all three synthetic datasets: (i) detect outliers and construct S', (ii) run K-Means with the chosen user-specified k, (iii) run HAC with all four linkage types, and (iv) compute Silhouette scores to compare methods and identify the best linkage per dataset.

```
# Multi-Dataset Evaluation

datasets = {
    "Dataset 1": dataset1,
    "Dataset 2": dataset2,
    "Dataset 3": dataset3
}

linkages = ["single", "complete", "average", "centroid"]

summary = [] # will store: (dataset, algorithm, linkage, silhouette)

print("\n Multi-Dataset Evaluation")
for name, data in datasets.items():
    print(f"\nProcessing {name}...")

    # 1. Outlier detection -> cleaned set S'
    S_prime, _ = detect_outliers(data, z_threshold=2.0)

    # 2. K-Means on S' with user-specified k
    labels_km, _ = kmeans(S_prime, k=k_value)
    score_km = silhouette_score(S_prime, labels_km)
    summary.append((name, "K-Means", "-", score_km))
    print(f"{name} - K-Means (k={k_value}): Silhouette = {score_km:.4f}")

    # 3. HAC with all 4 linkages -> track the best one by silhouette
    best_linkage = None
    best_hac_score = -1.0

    for link in linkages:
        labels_hac = hac(S_prime, k=k_value, linkage=link)
        score_hac = silhouette_score(S_prime, labels_hac)
        summary.append((name, "HAC", link, score_hac))
        print(f"{name} - HAC ({link}, k={k_value}): Silhouette = {score_hac:.4f
            }")

        if score_hac > best_hac_score:
```

```
            best_hac_score = score_hac
            best_linkage = link

    print(f"--> Best HAC linkage for {name}: "
          f"{best_linkage} (Silhouette = {best_hac_score:.4f})")

# 4. Compact text summary
print("\nSummary (text)")
for ds, algo, link, score in summary:
    if algo == "K-Means":
        print(f"{ds}: {algo:7s} | Silhouette = {score:.4f}")
    else:
        print(f"{ds}: {algo:7s} ({link:8s}) | Silhouette = {score:.4f}")
```

A.12 Example Visualizations for Dataset 2 and Dataset 3

For completeness, the following snippet shows how we generated the example cluster plots used in the multi-dataset discussion. The idea is the same: clean the data with outlier removal, then run the chosen clustering method and visualize the assigned labels in 3D.

```
# Example plots for Dataset 2 and Dataset 3

# Dataset 2: use K-Means on the cleaned set S2'
S2_prime, _ = detect_outliers(dataset2, z_threshold=2.0)
labels_km2, _ = kmeans(S2_prime, k=3)
plot_3d_points(
    S2_prime,
    labels=labels_km2,
    title="Dataset 2 (Spread=3.1): K-Means Clustering Result"
)

# Dataset 3: use HAC (average linkage) on the cleaned set S3'
S3_prime, _ = detect_outliers(dataset3, z_threshold=2.0)
labels_hac3 = hac(S3_prime, k=3, linkage="average")
plot_3d_points(
    S3_prime,
    labels=labels_hac3,
    title="Dataset 3 (Spread=1.0): HAC (Average Linkage) Result"
)
```

A.13 Effect of Different Values of k on Dataset 1

To see how the user-specified number of clusters k affects clustering quality, we ran both K-Means and HAC (average linkage) on the cleaned Dataset 1 (S') for $k \in \{2, 3, 4\}$ and compared Silhouette scores.

```
# Effect of Different k on Dataset 1 (cleaned S')

# Recompute S_prime1 just to be safe (same parameters as before)
S_prime1, _ = detect_outliers(dataset1, z_threshold=2.5)
```

```
k_values = [2, 3, 4]

print("\nSilhouette scores for different k on Dataset 1 (S')")
for k_test in k_values:
    # K-Means for this k
    labels_km_k, _ = kmeans(S_prime1, k=k_test)
    score_km_k = silhouette_score(S_prime1, labels_km_k)
    print(f"\nK-Means (k={k_test}): Silhouette = {score_km_k:.4f}")

    # HAC with average linkage for this k
    labels_hac_avg_k = hac(S_prime1, k=k_test, linkage="average")
    score_hac_avg_k = silhouette_score(S_prime1, labels_hac_avg_k)
    print(f"HAC (average, k={k_test}): Silhouette = {score_hac_avg_k:.4f}")
```

A.14 K-Means Validation with `scikit-learn`

To double-check that our from-scratch K-Means behaves as expected,we ran a small validation experiment on the same cleaned Dataset 1 (S'). The idea is simple: use our own implementation and `sklearn.KMeans` on exactly the same data (with the same k and random seed) and compare the Silhouette scores. This step is for verification only; the main project results are based purely on the custom implementation.

```
# K-Means validation against scikit-learn (using cleaned Dataset 1)

# Reuse the cleaned Dataset 1 as the validation set
X_val = S_prime1

# Our from-scratch K-Means
labels_km_custom, _ = kmeans(X_val, k=k_value)
score_km_custom = silhouette_score(X_val, labels_km_custom)

# scikit-learn KMeans (used only for validation, not in the main solution)
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score as skl_silhouette

km_sklearn = KMeans(
    n_clusters=k_value,
    random_state=GLOBAL_SEED, # keep it reproducible
    n_init=10
)
labels_km_sklearn = km_sklearn.fit_predict(X_val)
score_km_sklearn = skl_silhouette(X_val, labels_km_sklearn)

print(f"Our K-Means silhouette: {score_km_custom:.4f}")
print(f"sklearn KMeans silhouette: {score_km_sklearn:.4f}")
```

A.15 HAC Validation with `scikit-learn`

To confirm that our Hierarchical Agglomerative Clustering (HAC) implementation matches standard behavior, we compared it against `sklearn.AgglomerativeClustering` on the same cleaned Dataset 1 (S'), using the same number of clusters k and Euclidean distance. Only linkage types that are directly supported by scikit-learn (single, complete, average) are checked here. Again, this validation is used purely for sanity-checking; all main results in the report are based on the custom HAC implementation.

```
from sklearn.cluster import AgglomerativeClustering

X_val = S_prime1
k_val = k_value # same k used in the main experiments

linkages_to_check = ["single", "complete", "average"]

print("HAC Validation against scikit-learn (Dataset 1, S')")
for link in linkages_to_check:
    # Our HAC (from-scratch)
    labels_custom = hac(X_val, k=k_val, linkage=link)
    sil_custom = silhouette_score(X_val, labels_custom)

    # sklearn HAC (for validation only)
    hac_sklearn = AgglomerativeClustering(
        n_clusters=k_val,
        linkage=link,
        metric="euclidean"
    )
    labels_sklearn = hac_sklearn.fit_predict(X_val)
    sil_sklearn = skl_silhouette(X_val, labels_sklearn)

    print(f"Linkage = {link:8s} | ours = {sil_custom:.4f} | "
          f"sklearn = {sil_sklearn:.4f}")
```

References

[1] J. Han, M. Kamber, and J. Pei. *Data Mining: Concepts and Techniques*. 3rd ed., Morgan Kaufmann, 2011.

[2] Packt Publishing. *4 Popular Algorithms for Distance-based Outlier Detection.* https://www.packtpub.com/en-us/learning/how-to-tutorials/4-popular-algorithms-distance-based-outlier-detection

[3] Wikipedia. *K-means Clustering.* https://en.wikipedia.org/wiki/K-means_clustering

[4] GeeksforGeeks. *Types of Linkages in Hierarchical Clustering.* https://www.geeksforgeeks.org/machine-learning/ml-types-of-linkages-in-clustering/

[5] Wikipedia. *Silhouette (Clustering).* https://en.wikipedia.org/wiki/Silhouette_(clustering)

[6] Python Software Foundation. *PEP 8 – Style Guide for Python Code.* https://peps.python.org/pep-0008/

B Raw 3D Dataset

B.1 Dataset 1

X	Y	Z
-2.109251	10.638914	-6.313334
-0.730083	10.055287	-6.460577
0.765871	9.060434	-6.132724
-1.089185	8.716036	-7.984623
1.314356	9.080774	-7.643230
-1.171003	8.546328	-7.102152
-0.259653	11.338589	-5.304704
-4.643660	5.609050	-7.431398
-0.523510	9.393831	-6.725628
4.048542	7.007637	-7.796497
3.913421	10.173339	-5.923294
-0.688779	6.042739	-6.815371
0.432658	6.800040	-8.346616
0.106754	7.308721	-7.293694
0.408302	9.073329	-8.028133
1.305179	10.613374	-6.539281
-1.236382	10.326248	-8.019400
1.818922	7.080057	-5.470767
0.200318	6.761526	-7.681827
0.333817	9.500298	-8.884746
-1.756839	9.368526	-7.956957
0.660343	10.376409	-10.084625
0.694331	11.213638	-7.652356
-1.223034	10.363313	-6.660604
1.849022	8.387886	-9.784081
0.038413	8.206783	-5.721225
0.584972	6.073745	-9.268101
1.827253	10.232851	-8.269246
0.234545	9.811306	-6.273680
1.813668	9.470948	-7.287499
-0.229494	10.909611	-11.168345
-0.013147	9.068674	-9.682436
0.835497	7.836968	-5.564407
0.010367	10.213750	-4.922889
0.925706	7.432976	-9.842581
3.392524	8.808948	-8.356225
0.496095	8.664734	-5.582952
0.297503	9.034023	-8.403051
1.081655	7.148314	-5.918207
2.979520	6.264839	-11.556020
1.346814	13.595490	-8.918472
-2.014820	10.069418	-8.630107
-0.674413	8.382662	-6.159204
-0.493112	9.509463	-7.434568
-1.283975	8.433587	-8.827527
0.248159	6.986315	-9.084018
2.858964	8.913542	-7.213831
1.157198	8.251731	-7.528171
1.001700	9.517622	-9.203542
1.736449	7.946491	-9.017750
-1.384423	8.306292	-4.187667
-1.879532	9.297411	-10.964892
0.233612	10.628493	-7.542802
-0.896406	9.425994	-5.856535
1.431016	12.559727	-6.740306
-0.829906	8.782511	-7.247305
0.432160	8.955224	-6.803669
-2.771098	10.502606	-8.151338
-1.875253	10.157226	-4.745621
1.123883	9.299361	-8.794804
5.405254	10.593739	-9.167538
-1.166916	9.165837	-9.915324
0.539967	8.182945	-4.909521
1.968311	4.128960	-7.041743
-2.675009	11.006622	-6.814217
1.223562	7.092049	-3.825633
3.872565	7.092686	-6.445741
-0.975512	8.966848	-9.394954
3.597294	7.264751	-7.649759
1.139102	7.843665	-7.547570
-0.778119	8.769045	-9.223784
-0.551946	8.636867	-7.717515
0.338474	8.481690	-5.761027
-0.345320	8.763305	-8.313414
-0.711294	6.733187	-6.182995
-1.820100	7.666732	-6.470167
0.961065	8.289067	-10.751486
0.993356	9.476488	-9.659094
1.623012	7.747294	-9.143946
0.408748	8.688027	-6.752085
-2.653914	12.271288	-8.201593
-2.534954	10.123190	-7.755455
0.821178	8.397979	-7.224340
0.678824	7.665299	-5.895076
-0.609388	7.443837	-6.978149
1.037507	8.596931	-8.669343
1.352047	5.840682	-8.972363
0.307574	6.559367	-7.066418
0.137679	10.627006	-8.763430
-0.890199	9.609001	-11.540422
5.816509	7.751703	-8.430511
1.786462	8.937577	-10.319779
1.364902	10.548954	-7.926711
-0.270448	9.884046	-8.752612
1.025532	9.368011	-8.331686
-2.269351	8.603183	-8.692568
2.038971	9.268628	-5.709056
0.478752	9.482496	-8.526206
1.438918	12.221731	-7.674245
-0.830562	8.725168	-7.983112
-1.026230	9.258022	-7.640459
2.826405	9.009637	-6.533766
1.950072	8.467913	-4.530682
-0.902417	7.554284	-7.776088
0.029941	6.486901	-7.179978
-2.765043	11.515123	-7.262602
-0.919091	7.374265	-7.808783
-0.165111	7.129154	-8.772439
-0.100483	8.071863	-5.426224
2.284599	9.038112	-6.264329
-2.166907	10.156626	-7.171858
1.108835	11.889915	-11.222352
0.706139	7.030860	-6.050854
-2.127532	8.117555	-6.751887
1.340749	9.143903	-8.543903
-0.759925	10.601379	-7.126707
-2.794433	10.527867	-6.367560
1.808580	8.403344	-5.626126
-1.673484	10.035274	-7.999492
1.450266	10.819433	-8.441590
0.144219	9.079392	-4.975411
1.515437	6.814574	-6.293113
1.577593	12.832125	-10.139276
-0.729002	11.409342	-9.555934
-1.922597	9.940022	-5.283672
-0.967192	9.981503	-6.906288
2.970181	9.006541	-5.334363
-1.389180	8.676492	-7.290876
2.286827	10.052577	-8.469963
1.463974	10.396410	-7.317770
-0.227360	8.660428	-10.167794
0.576170	9.431514	-8.674759
1.572836	6.537933	-8.110071
-0.618962	12.586792	-9.995339
1.249167	10.705096	-6.436902
2.363687	7.204346	-11.221273
1.616705	6.856783	-7.704328
-1.885490	10.903489	-5.559537
-1.059682	10.635559	-6.898509
-0.014467	9.112624	-7.481757
1.343442	9.567900	-7.745849
2.047693	7.908618	-6.600327
0.996312	11.671127	-8.026602
3.373981	9.326371	-7.465966
-0.974018	10.064602	-7.028724
-1.750064	6.979077	-8.116742
-0.980378	10.971818	-4.640839
1.799840	9.653438	-7.887637
0.334502	10.601747	-3.281058
1.879651	8.504532	-7.048363
-6.402147	7.708858	-2.017011
-4.311009	7.585716	-1.968815
-3.860215	5.691119	-2.862233
-6.276929	4.445416	-2.528880
-7.112060	4.663812	-1.329143
-8.347041	4.387338	-4.850908
-6.284821	5.526715	-1.815551
-6.239718	5.466133	-3.307851
-7.184541	5.323895	-0.307275
-6.043395	8.999383	-2.275118
-4.954865	6.684345	-2.351455
-8.231176	2.275550	-1.818571
-4.755728	4.893239	-1.799528
-7.436838	5.414089	-5.955165
-6.335981	4.454863	-4.577282
-2.838177	1.906892	-4.050206
-8.915807	6.402898	-3.709222
-5.550672	5.461754	-1.643263
-9.107895	3.302444	0.900938
-9.077611	2.907257	-2.402713
-5.298071	4.857186	-0.774990
-8.675383	6.104833	-1.086418
-9.624470	7.176085	2.015647
-8.137693	4.504411	0.404691
-9.420486	0.822487	-4.445169
-7.774974	3.363040	-1.017444
-6.270888	2.161296	-1.089385
-3.380907	6.558933	-0.326676
-6.717933	6.174772	-3.849708
-5.402427	4.247188	-0.075039
-5.915434	2.442452	-1.880282
-4.561455	2.371615	-3.117121
-8.137865	1.712073	-0.380387
-4.394458	5.844313	-1.674043
-6.847910	4.835120	-3.427470
-5.145331	6.286122	-0.432091
-7.709960	4.545363	-2.409075
-3.534819	4.727689	-5.498397
-6.036943	7.792315	-0.877353
-5.171190	4.469146	-5.659170
-10.012717	2.161530	-2.338967
-6.196256	5.645181	-2.723494
-5.031781	3.900688	-3.378751
-5.221298	2.758235	-7.023022
-8.662175	4.573723	-7.654989
-7.397102	3.806473	-4.678290
-9.422760	3.571804	-3.095450

```
-4.486400 5.078989 -4.956209
-8.292626 5.673490 1.302089
-6.058554 4.963518 1.238685
-6.795062 3.847854 -4.654517
-7.677682 8.325919 -4.669804
-6.734108 1.869246 -1.879413
-5.156488 3.965667 -0.796044
-5.456734 5.208880 0.978577
-5.352346 3.855308 -3.336152
-8.275805 5.260797 -2.693369
-1.841753 7.720641 -2.496813
-7.346690 7.446553 -5.500717
-7.567045 6.115598 -3.752445
-7.612547 3.772181 -0.921214
-6.506705 5.127993 -1.039159
-6.700365 6.403255 -3.422795
-9.339272 1.205866 -5.028487
-7.745182 6.258320 -1.797169
-7.831637 2.478496 -3.062664
-7.480853 3.387057 -4.689249
-7.610707 6.235081 -6.024966
-7.187349 0.585809 -1.781184
-8.099734 0.649998 -1.929405
-3.482924 4.600240 -0.019445
-6.882961 0.520572 -1.227799
-7.507156 1.772483 -0.697963
-6.237910 3.458498 -0.548144
-8.323983 5.562414 -3.803498
-5.793135 2.422889 0.569869
-6.296500 6.983411 -3.591072
-7.611238 5.915632 -7.049360
-8.674477 6.263954 -1.273473
-4.992468 3.846666 -1.095471
-8.069773 1.934707 -5.161300
-7.747331 3.416141 -3.243549
-13.142102 0.806749 -3.069408
-6.237623 3.362409 -3.689084
-10.909505 8.042934 -4.092670
-3.724139 7.213470 -1.617597
-7.784950 4.642224 -4.186145
-6.284939 4.409549 -1.153127
-5.043467 7.523345 -2.809638
-4.876154 2.979230 -3.720886
-7.866755 2.636697 -0.346194
-5.092419 6.694397 -2.156210
-6.260366 3.363067 -4.182400
-5.949621 4.887307 -4.103293
-5.700655 1.269126 -2.432867
-7.888818 3.451538 -4.060893
-5.039764 5.422942 0.019487
-6.519358 2.067739 -2.792872
-8.898985 5.208467 0.047001
-6.185291 3.301698 -2.800658
-6.411443 1.775592 -1.868386
-6.423957 3.395798 -2.891933
-6.477333 2.895316 -3.623502
-5.386306 4.770303 0.874925
-8.568614 1.162497 -0.456867
-5.072793 3.032306 -4.797812
-6.884196 1.204006 -2.774806
-10.733895 3.921933 -5.983100
-7.238165 7.076356 -2.651921
-5.430824 3.185497 -3.781939
-9.517394 6.208391 -2.285233
-3.906080 2.208562 -2.587836
-6.689117 6.866945 1.087813
-8.004521 2.279725 -5.321591
-4.434552 4.159446 -3.366128
-6.936845 -0.291800 -5.308609
-5.163409 5.023494 -1.934126
-8.650890 5.367857 0.995960
-9.053336 4.335542 -2.989400
-6.200552 4.333270 -2.471785
-8.438721 4.278982 -3.313593
-7.114967 4.192711 -1.030137
-7.899625 3.724879 -0.278329
-7.244442 3.835500 -0.652183
-6.360906 4.471272 -1.811518
-9.081690 5.139002 0.182167
-5.800404 1.513117 -1.687201
-8.469158 3.912224 -1.959632
-8.980908 6.730025 -2.311660
-5.403522 5.472499 0.235373
-4.681070 1.680547 -2.102010
-10.664583 3.533437 -0.889509
-5.173319 6.661122 -2.851869
-6.239518 4.518848 0.109967
-3.954713 3.699476 4.642274
-6.819127 3.686236 -1.088557
-5.445307 5.332341 -3.334087
-4.155412 4.450719 0.608177
-4.294442 2.066296 -3.918433
-8.597873 4.812975 -2.052864
-6.308618 2.905356 -1.377804
-4.261065 1.967765 -2.495501
-6.338443 4.139545 -2.496099
-3.600178 6.684957 -0.610559
-8.153213 8.410328 -1.488045
-7.821610 3.769956 -3.018461
-2.962487 4.748372 -2.021894
-10.654155 5.707094 -4.039362
-8.811575 5.483439 -3.688205
-5.273150 6.400243 -5.354980
1.855053 -5.088130 -4.176721
-0.347857 -4.765463 -7.647980
1.634691 -5.117121 -7.422887
-2.265166 -0.472258 -7.149037
-0.560336 -0.778365 -9.749109
-0.504540 0.679693 -7.654728
2.644106 -2.293516 -8.692533
-0.242702 -2.507359 -8.171233
-1.283967 -4.054146 -10.105251
0.396882 -1.940095 -6.088069
-3.053159 -1.198879 -7.467599
-2.374612 -2.473338 -5.970538
1.137254 -1.343852 -5.735206
1.809620 -2.713251 -11.184824
-1.169720 -2.007568 -8.280780
1.432385 -1.211568 -7.347968
3.583033 -4.401800 -5.757303
-0.650158 -4.050680 -6.373341
-0.907608 -1.099121 -9.532537
0.264548 -4.163115 -7.855269
-0.468044 -4.676186 -6.251905
-2.039142 -4.408202 -5.826256
1.955029 -1.440839 -6.817535
4.304987 -4.819170 -8.766987
-1.583803 -1.023215 -6.909391
2.205671 -0.889469 -7.832133
-1.267217 -4.425051 -3.444996
1.949779 -1.570283 -8.035003
-0.147620 -3.256207 -8.007021
-4.116802 -2.747000 -8.112049
0.731418 -2.259130 -5.887516
0.186728 -1.201463 -12.033006
-1.771722 0.470756 -9.617542
-0.291452 -1.363154 -9.011304
1.511571 -5.205902 -10.866720
-2.181718 -5.612002 -7.454555
-0.360994 -2.044141 -9.287019
4.788260 -2.807158 -7.505060
1.031485 -3.073520 -8.559146
0.365734 -0.981241 -6.224042
1.907041 -3.649846 -8.816152
3.820597 -3.063242 -6.558960
1.543156 -0.074536 -9.851019
-0.649228 -3.411369 -7.960754
-4.127127 -2.803514 -12.626520
0.753265 -1.661515 -6.339848
-1.950650 -0.908755 -8.970242
-1.989792 -4.391533 -8.939877
0.530814 -1.141021 -11.195309
-0.752202 -1.000692 -7.044058
1.139689 -5.318248 -5.689481
-2.587603 -2.267712 -4.946945
3.619327 -1.689075 -5.811464
-0.854900 -3.476440 -5.292246
0.239388 -2.839006 -9.936213
0.507560 -1.447287 -8.826714
-0.676247 0.234981 -7.076329
-0.282972 -4.656410 -8.512451
-0.213497 -0.145411 -8.496099
3.466301 -4.281479 -8.040865
1.754875 -3.042192 -8.891680
-0.000822 -1.802158 -10.485530
-0.210869 -2.806647 -8.123954
1.560138 -3.633235 -8.053335
0.304253 -1.297912 -7.795301
2.966372 -4.243734 -5.081483
-1.307582 -2.903360 -6.741925
3.935208 -0.997354 -8.127762
2.574911 -2.994510 -8.821811
1.044129 -2.439793 -5.680609
-0.753032 -1.923134 -6.255020
0.736979 -2.662011 -8.652922
1.863250 -2.362837 -6.503861
-0.394904 -2.586420 -9.264219
0.390442 -1.232462 -7.229821
1.783404 -2.046602 -4.942267
-0.105593 -5.559375 -6.618730
-0.060844 -4.363569 -6.224931
0.712289 -1.808231 -7.995485
0.591181 -2.152030 -4.202829
-0.994631 -2.628730 -6.742261
-2.868120 -5.495184 -3.498735
-2.944318 -1.574968 -8.924014
-1.137192 -6.178386 -9.018538
1.648378 -5.109578 -6.325440
0.800433 1.023103 -8.251369
2.510642 -1.009877 -7.913212
-2.756824 0.107152 -9.853108
-0.615652 -1.020123 -5.388488
-1.285307 -2.667444 -7.862650
2.363302 1.898499 -10.691005
1.886058 -7.754556 -7.371889
-1.759316 0.975433 -7.097745
-1.089693 -3.508604 -8.432741
0.575457 -2.894508 -6.216614
3.008967 -1.347299 -9.604014
-2.686356 -3.969596 -5.501421
-0.366590 -3.811472 -7.148300
-2.311450 -4.660631 -6.478623
-0.309180 -3.425666 -6.825465
-0.816272 -3.845165 -8.224867
1.321214 -2.502165 -9.547484
-0.309006 -1.443038 -4.029855
-1.819741 -2.000477 -7.862955
1.722432 -1.800642 -4.747290
1.171508 -3.591045 -8.366676
0.966487 -1.898943 -3.080921
3.253895 -4.722282 -6.766804
-2.159283 -5.590059 -6.058826
1.229642 -4.479263 -4.438572
-1.523129 -4.711752 -6.222496
0.634186 -2.898755 -7.517927
0.639338 -2.861085 -7.707717
-3.176225 -5.025371 -6.297499
-1.031245 -1.628493 -5.173373
-0.402415 -2.100495 -3.659551
```

```
1.487381 0.705532 -4.637076
1.592545 -2.811249 -8.912797
-1.764560 -1.734449 -6.554478
-0.077823 -1.082601 -8.485375
2.234891 -2.537828 -7.462596
0.236005 -2.787305 -2.918333
-2.664239 -2.290840 -5.157817
1.528020 -3.153588 -4.264991
-1.686791 -4.961174 -6.374681
-2.381549 -6.890150 -5.808013
1.606927 -1.291724 -8.274657
0.565800 -4.347429 -4.275376
1.660340 -1.273780 -10.907181
1.689640 0.079309 -5.207937
1.656931 -2.611564 -8.012508
1.319452 -1.920856 -6.971734
-0.320108 0.416421 -9.499326
0.195758 -3.654166 -5.539531
2.811363 -0.370481 -6.037476
-2.288187 1.136378 -6.467156
0.041263 -2.571140 -8.008343
0.999482 -2.051539 -8.045389
-4.389308 -2.673059 -6.538904
1.183599 -2.886078 -6.581242
2.087471 -3.456291 -4.757906
-3.282788 -4.879728 -9.026469
-4.163543 -4.409413 -5.150139
2.471249 -2.434595 -4.299405
0.667579 -3.505902 -6.962273
-0.010367 -1.285160 -10.102700
2.035443 -3.436989 -5.857627
-2.537376 -3.364506 -7.256695
1.859111 -4.801397 -4.395809
1.105291 -3.556236 -5.206065
-2.151774 1.568609 -8.316207
-0.804082 1.106353 1.290786
-9.395946 5.464686 4.439888
-4.059776 -0.573091 4.783638
6.681495 -5.045672 5.416642
-1.516033 9.063210 7.567310
-2.840170 -2.803350 -0.283283
9.212896 -7.120389 0.978025
2.954215 4.401101 -8.657703
1.986306 -0.991179 -3.211662
9.410715 -7.879418 4.173836
-5.912791 8.962046 0.006291
-0.932248 -5.029966 0.675588
7.286579 -8.558534 3.164530
-7.068912 -5.498591 7.077050
-8.554569 4.633418 -8.842938
0.190866 2.470160 -7.060029
6.042192 -5.579784 -0.486350
-5.194376 5.271718 -4.977174
-4.756348 -9.756464 7.996752
-9.328061 -3.708271 2.940143
-3.358285 1.944122 -2.708703
7.599022 -6.820416 -9.631791
-8.006000 0.479841 1.845374
6.678056 9.092960 -1.973912
6.473903 -1.896605 -2.586860
-7.346114 7.373068 -3.277855
2.933822 0.343310 9.936750
-4.887320 1.029973 5.792645
-6.178380 6.463476 1.163395
-7.929619 9.361135 -1.762969
-0.491876 5.689246 9.292746
-5.089229 -7.276690 4.894618
3.540750 -1.069589 2.952138
8.883000 6.838220 1.364394
8.172535 8.258229 5.423112
-7.628781 1.234178 2.145338
7.586696 0.324598 2.251796
4.303278 -8.322574 -8.873141
-9.392043 -7.863291 -4.890317
-9.032264 1.928927 1.445784
-0.894171 -9.821013 0.383655
-7.472628 -9.897886 5.924548
8.213060 -8.553598 3.179963
-8.934262 1.643775 2.478233
1.735040 9.806002 2.273027
-6.677331 8.656218 -4.019773
0.788521 -1.349756 6.134777
-8.319939 -2.647740 -1.145689
1.948277 8.158158 -3.970037
```

B.2 Dataset 2

```
X          Y          Z
-12.336306 1.548916 9.831429
-5.776568 -1.631377 7.156268
-6.484608 -0.999718 5.321790
-5.787111 -6.485832 7.694885
-5.075271 -2.339782 4.402239
-4.374591 -6.796227 8.893056
-4.184849 -3.005407 7.557077
-7.901106 -1.602316 12.660294
-9.846929 -9.391353 1.619537
-2.159235 -3.631159 9.627376
-2.528222 -3.377405 7.165033
-5.294015 -1.337950 2.782395
-6.075520 -3.277067 11.868919
-7.137596 -7.375265 4.538324
-1.763013 -4.758694 10.389991
-10.572599 -0.531755 9.492600
-9.165621 -3.554073 10.053363
-4.495208 -0.931103 13.646664
-3.918567 -4.899362 3.894253
-2.758757 -4.640040 5.730401
-5.094063 -2.015581 2.978862
-9.510374 -11.575195 10.000395
-4.538998 0.651193 6.256750
-7.068439 -2.548596 6.047091
-8.655736 -6.773884 11.761531
-3.669270 -2.739379 5.427204
-6.905888 -1.266044 5.960173
-7.120992 -4.445976 3.477136
-4.178615 -0.529571 3.692651
-0.339271 -6.099799 6.760093
-7.360556 -4.719106 6.431468
-6.115281 -6.209130 4.183068
-7.313353 -8.897425 5.469309
-3.524167 -1.214131 8.290533
2.849986 -3.042264 4.869880
1.034310 -7.276553 9.286795
-7.728706 -2.932431 0.182485
-1.980009 -4.520745 3.284702
0.435342 -1.657578 6.426519
-7.078632 -4.164601 5.776336
-2.520950 -1.556144 4.214470
-6.470806 -5.679580 2.101631
-6.601639 -4.316642 8.426466
-0.674581 -6.536049 7.992542
-6.145573 2.402905 6.133472
-3.211164 -6.932280 3.771377
-4.142502 -5.220746 7.379709
-9.609412 -1.977962 3.496126
0.571755 -4.924426 9.686036
-9.092206 -2.700444 3.531623
-6.277113 -3.953928 5.301582
-3.948565 -1.712445 8.308139
-5.038950 -8.256812 3.790112
-5.374602 -10.607838 8.615965
-5.510636 -4.809267 9.224481
-2.697235 -3.345074 3.180381
-3.805450 -2.957027 3.541717
-1.533491 -2.455727 0.178679
-4.389714 -7.950741 9.807092
-2.265971 -7.451308 3.541409
-4.452012 -4.770491 11.012644
-2.470738 -4.905702 8.074442
-11.067823 -3.069597 8.970377
-5.322510 -6.231841 8.038599
1.303948 -3.395083 2.810414
-0.285762 -9.853674 7.844450
-6.571817 -1.145322 3.651144
-0.663829 -3.099331 0.931309
-10.886578 -5.010073 10.296571
-4.373964 -3.822103 5.553213
-2.480267 -2.860464 7.153587
-7.820398 -0.347832 10.510064
-8.801953 3.104458 8.524285
-6.280576 -9.933680 8.320739
-3.039472 -5.876636 0.733800
-1.281879 -5.853966 4.822105
4.053742 2.566537 9.241099
-6.179562 -2.016847 -0.788041
-4.437506 -6.385614 2.299139
-9.674451 -2.402238 6.561346
-4.272614 -1.418811 3.661655
-8.598069 -9.536910 5.245405
-2.411273 -4.493390 8.272012
-8.429024 -5.484453 3.910111
-8.637729 1.734191 8.926151
-7.129339 -6.404830 6.107205
-9.238124 -2.679113 2.272375
-5.916100 -7.207127 4.796985
-8.894345 -8.905925 3.289478
-6.875788 -4.944334 6.498364
-0.034379 -1.512196 5.500527
-2.517410 -6.165759 10.774838
-3.064379 -8.233279 8.626210
-2.237871 -12.355523 3.965420
-6.236607 -10.008565 2.976265
-7.707668 -2.458180 7.215951
3.373382 -5.252381 9.859531
-4.523333 -3.190307 2.689662
-10.691547 -6.408337 8.647237
0.138940 -2.157805 7.829186
-7.367935 0.726134 10.450314
-6.413338 -4.173926 3.196754
-0.219416 -2.770479 5.989814
-3.506367 -2.664248 5.008875
-8.616609 -8.009788 0.895579
-3.337873 2.847303 1.017400
-4.339066 -2.756127 3.232175
-4.134037 -7.430682 10.005761
-6.609929 -3.718440 8.405664
-6.100197 -9.582993 4.637622
-2.402472 -4.292228 6.996861
-8.117003 -3.586186 8.541832
-4.654768 1.884668 5.640355
```

```
-7.995943 -9.044921 9.855733
-4.950714 -0.894150 7.703288
0.270004 -0.105244 5.114096
-3.918714 -0.665524 2.452682
-0.269980 -0.272998 9.331585
-5.390647 -3.093540 2.821073
-4.588099 -5.851054 6.552863
-6.301099 -3.150386 9.940445
-8.827527 -1.161465 0.109896
-11.466961 -4.400220 6.305376
-3.395075 -9.137071 6.098705
-4.375531 -0.139042 5.962612
-3.127883 -7.060196 5.703095
-6.148917 -1.849934 3.219175
-7.354157 -2.244614 11.831768
-3.581377 -3.089140 5.211473
-1.393639 -2.356435 11.657412
-3.742552 -6.971076 4.517696
-7.828711 -7.907722 10.202039
-2.265013 -7.869975 11.806609
-4.009619 -6.996542 7.481803
-4.982260 -6.425407 0.261419
-4.653926 -3.124990 -1.710625
-3.047545 -2.115514 4.828148
-4.420176 -4.900217 3.689497
-5.883928 0.159795 1.787602
-4.032461 -3.459412 4.090861
-6.132809 -8.371459 3.643432
-4.721407 -3.017358 3.499547
4.706904 -5.798245 6.679238
-5.482556 -5.269494 10.317828
-6.252073 0.106428 9.824145
-8.009319 -0.820147 4.268343
3.715265 -5.226251 3.367643
-10.692094 -8.038617 4.484503
-7.861644 -7.839107 4.128962
-5.408055 0.462800 1.004301
-0.645399 -11.145379 3.719972
2.678532 -10.005354 3.392141
4.150010 -9.029920 2.688423
0.711736 -6.392718 2.944604
2.001481 -12.117977 6.209337
3.571647 1.121321 -0.528431
5.923438 -9.683680 2.283508
5.863137 -10.815455 1.186857
3.778543 -7.985812 3.346135
6.458870 -5.753353 2.312476
2.202768 -11.839620 -0.782908
1.641123 -6.221187 0.369946
-0.571626 -6.395924 -0.992940
0.905556 -9.039042 2.963988
-4.201245 -11.814108 2.041463
2.150725 -11.017689 -0.181367
7.573912 -9.750358 3.706398
5.734418 -9.733286 2.871798
1.883888 -10.229895 -1.993445
5.991989 -9.571895 -0.214504
-0.215455 -2.730181 3.796734
2.867930 -10.491277 -3.245243
4.184247 -7.443493 -2.868905
6.880430 -5.932136 -1.485292
1.556965 -9.662295 1.995242
2.828359 -13.418312 6.066839
4.260186 -4.429203 5.387086
2.635132 -7.277518 3.780797
5.665426 -8.845749 -0.419238
3.389013 -10.822383 -0.226258
-3.013932 -8.078308 -1.686226
1.556521 -8.500972 2.470026
9.136057 -7.703972 -2.883984
3.684938 -10.184351 2.735232
1.154488 -10.533662 -0.087781
1.334180 -13.994422 -0.574505
3.236190 -4.616085 0.588285
2.127223 -6.540102 3.169285
8.238153 -6.523755 1.436841
7.428444 -3.596207 0.023464
3.769858 -13.547586 0.368342
0.539994 -9.001225 0.004070
2.764870 -8.657008 3.026888
3.995533 -10.079194 -2.499268
7.107665 -3.943963 0.088358
5.020009 -13.068526 -3.345729
2.442982 -3.775387 -1.747173
0.693839 -3.354057 -4.372551
5.282913 -6.825377 0.857698
3.864362 -0.210201 -1.996077
2.306410 -9.075783 -0.640874
-0.880213 -3.398370 2.179432
-2.170151 -10.516294 2.753879
4.530227 -12.802615 1.213714
4.088358 -7.120628 2.720396
-2.371720 -2.611954 2.151651
8.878192 0.538006 0.999409
-0.545040 -5.658796 0.936553
1.880254 -11.540788 6.729269
1.907333 -12.676313 0.109666
-0.307878 -7.848098 0.197107
0.001609 -6.472126 -0.497508
5.523284 -4.428552 0.375083
1.275430 -7.719783 4.260631
3.270966 -7.903737 1.413776
4.244390 -6.388497 -0.691658
1.783769 -8.202475 -0.566633
3.701613 -6.675538 1.429799
1.408620 -10.746928 6.925936
6.002992 -11.809357 -1.110046
9.088543 -3.652578 1.894484
6.868956 -12.947243 1.501204
5.576133 -14.103499 -0.930145
5.706474 -4.829548 -0.342116
4.178033 -3.375392 0.313145
-0.853804 -8.527964 0.278348
7.312874 -7.841242 -0.989356
10.721312 -9.887963 2.961719
5.375942 -5.906202 2.033176
5.098313 -8.548097 -2.902327
2.598211 -8.952678 0.946204
3.701509 -10.274180 5.599235
8.136440 -15.158722 1.526048
1.076781 -16.416957 2.768558
1.361345 -9.502378 -0.522755
6.649341 -4.495777 2.355203
5.428262 -8.686128 1.931631
0.980160 -8.230518 -0.715241
4.436006 -5.772874 2.461370
6.950424 0.002160 -5.033203
3.322045 -10.100510 0.081381
-0.930719 -8.174741 0.696492
1.318035 -9.126001 -0.340369
-3.983434 -4.997633 0.556372
0.335748 -7.387723 -0.556422
-3.459768 -5.588203 -3.700661
2.602562 -7.965116 5.681109
6.968812 -7.823782 -0.869377
1.990343 -4.965725 7.035130
4.965369 -10.336619 4.688191
2.116380 -9.601934 0.632334
5.109086 -7.484248 4.574821
7.298443 -6.746539 -5.732880
-1.979828 -7.845977 1.296544
9.973132 -8.431264 1.024046
6.205677 -5.037857 0.767366
4.746551 -11.851595 -1.426633
7.102151 -12.699721 4.356306
5.381078 -5.951359 4.326915
5.027600 -14.976123 -1.616439
1.415558 -5.501657 -1.999059
1.094450 -9.462318 5.578076
-1.187639 -12.534782 3.441337
-3.309512 -7.308469 0.089610
-1.608925 -9.640442 -1.609488
3.799757 -7.209752 6.112918
3.239990 -8.025885 -0.292526
3.206355 -5.083833 -4.472582
5.322924 -9.996649 1.726197
4.287263 -5.700759 2.236187
2.062843 -7.371348 1.533944
2.645675 -7.701643 4.205237
3.708454 -7.855646 -2.494145
2.885889 -14.097455 3.729170
0.367938 -4.321918 10.776792
2.047687 -4.215747 2.748294
1.552964 -10.945666 3.239156
7.603572 -7.241114 -3.189747
0.048620 -9.044321 5.704353
4.158212 -7.323089 5.249699
2.645378 -8.613860 0.933352
-2.232054 -6.890896 2.770737
6.303496 -12.023200 -1.032039
6.176480 -7.336110 5.759717
0.604224 -7.715111 -0.598455
4.239860 -9.841239 3.158635
6.631852 -6.244262 -4.108509
6.194875 -6.950220 -1.736227
0.533749 -8.311850 3.275078
1.882451 -13.367387 2.370032
3.806031 -0.770089 -2.497397
11.103251 -2.929023 2.436818
8.422475 -8.038263 4.528432
1.004950 -8.090537 -1.648108
-1.604336 -9.179748 -0.545434
5.055342 -4.398108 0.972763
3.462314 -7.755050 -3.704564
4.113775 -8.302414 1.353536
-0.131538 -14.481429 0.721330
4.259710 -4.180770 5.003783
6.516443 7.514996 5.022964
15.491984 6.319006 -1.810627
11.426049 3.113825 1.946240
6.243011 2.752807 6.272134
9.002866 3.669573 2.349289
5.812047 7.473958 0.960812
4.779970 3.569499 0.780093
8.052115 2.718339 1.857293
8.530178 12.812405 4.788700
-2.094595 3.982516 0.903871
8.148272 10.431948 4.434773
0.016056 6.518482 2.469335
8.435321 9.779346 -1.247301
10.574595 0.439874 2.291889
8.334929 12.648902 -0.042887
1.903878 7.696941 0.084523
12.002937 3.371876 -0.585199
11.013022 5.831688 5.236769
2.721410 10.685227 0.308733
9.695347 2.451539 -2.672797
6.358924 6.107685 7.021948
8.240225 7.784951 4.176983
16.796556 3.030608 6.025326
6.692384 6.375270 0.439407
10.758540 1.676782 -1.964723
6.510813 1.976928 4.314500
10.012953 3.232564 4.564231
5.102739 5.429734 2.155300
12.972083 5.565526 -0.692742
4.310681 9.286843 -3.039600
5.697322 4.388191 -5.462252
5.700371 8.673538 2.848338
```

4.362706 7.706680 3.735626
13.431026 3.265605 0.929307
7.282427 5.136922 2.402106
6.822017 3.412708 -2.839336
5.416123 1.249890 -1.126910
5.455668 7.125454 4.762368
3.917323 6.488771 4.589384
6.976256 0.435663 -4.597447
9.563210 8.349085 2.932060
3.824712 10.004174 5.828519
6.941571 4.802096 -4.009755
9.407715 0.485862 4.165262
9.336216 5.777390 6.950793
6.075433 7.315225 4.525446
7.847123 4.190343 5.870006
8.387488 4.891717 1.542140
7.481216 12.560331 2.386331
6.336345 2.188735 -1.724146
1.194811 6.215738 2.143794
3.886995 0.876710 1.040386
5.446279 1.340095 2.983848
7.510496 5.321963 4.437771
5.042072 9.692221 -1.102416
8.657980 7.443799 1.233717
8.150209 1.490831 -2.621871
6.434443 3.206556 3.871306
7.151083 7.381911 3.025381
7.689269 5.757486 0.880223
5.615761 11.776978 -3.424616
6.476154 8.578333 3.333257
11.421231 7.167500 -1.884732
6.028946 1.106757 5.081770
11.521407 4.088122 3.704998
5.765793 0.988447 0.628121
2.125055 3.678201 2.662161
9.277414 9.452193 0.968553
7.751145 6.736406 3.719228
-1.823370 3.012299 4.458429
3.537519 6.185217 4.568217
1.121993 4.820015 2.711648
7.028552 0.660641 3.044183
5.247878 7.946410 3.294602
5.272802 5.930805 -0.191818
4.794401 0.474610 1.195292
9.444347 5.706983 1.842566
4.967475 3.695307 4.188459
2.264722 11.698553 1.667633
13.616597 8.929866 0.553788
9.141424 6.065640 3.264499
9.307274 3.554620 -0.104119
5.530866 0.996884 0.983700
7.524833 5.996372 2.381924
5.573035 7.647651 -4.593061
5.549623 4.166381 4.525183
8.820243 7.647598 1.770147
9.681349 2.032137 1.923274
0.166318 10.431942 6.617601
4.843962 2.476946 -4.832737
1.941131 5.477870 1.179281
2.113356 3.550319 -0.393687
8.144931 7.303864 0.992273
7.052199 5.092129 -4.799481
7.255394 4.911851 4.006984
1.010704 7.486172 -0.295291
7.534756 8.875539 -2.886316
2.644590 8.207584 -1.844020
3.740249 6.994662 1.065305
9.130516 6.060801 1.619663
9.775157 11.920077 1.394981
11.313712 8.653081 2.048354
8.145667 2.233858 -0.202798
7.590370 4.996224 5.744820
2.702011 4.234785 -0.265156
6.262219 8.755751 -2.026449
9.372665 3.319989 0.712983
9.836754 4.275022 3.791999
7.098837 4.621605 0.609640
1.380652 2.848311 2.250434
7.384033 0.546360 4.850438
8.542996 1.141881 -1.485739
8.345591 -0.270627 -0.547332
6.614024 4.801008 1.588464
8.001755 2.727864 0.263675
13.166656 0.862301 0.388401
7.149815 3.213977 5.580622
7.039702 7.579738 -6.313958
10.506741 8.799743 -0.540114
10.881813 6.068137 4.693738
12.462303 5.567626 -1.316248
6.815773 1.306857 -0.652296
4.489481 3.068029 5.463767
9.464612 2.503361 5.367479
1.671538 3.260889 1.438314
10.583999 7.569106 3.296615
10.953653 7.551032 6.139004
4.485067 0.096935 0.616488
9.512426 6.157742 4.680006
9.148335 6.957962 4.466302
4.728176 8.753634 -1.190622
6.597472 0.973967 3.501569
2.138137 9.727856 -2.238100
7.759483 4.933105 -0.786483
10.833034 4.038288 6.486062
3.773162 2.879342 -4.269142
8.674229 8.215785 2.635876
6.457945 2.513862 9.234768
9.267647 -0.349362 0.979734
-1.134001 10.277602 8.082969
11.505758 5.213226 0.110018
2.612430 7.035340 -0.364085
5.804946 3.398944 0.994636
9.763755 10.635430 2.426933
7.040935 1.934403 -1.857852
3.679037 8.147175 2.735886
9.999349 6.720740 0.721876
3.128475 4.261466 -2.468408
13.225660 6.250071 0.836992
13.209669 4.219496 3.717495
-7.330921 0.342493 9.153135
8.073817 -7.721147 -0.127562
7.455380 2.120527 -9.590491
-3.000946 -4.373955 -3.230727
9.545396 6.347968 9.942945
5.888963 9.371209 -1.669605
8.734304 0.917094 6.237229
-5.972610 7.537379 -2.061821
8.831473 -3.868641 -2.092278
0.204850 7.676152 4.644466
-4.778378 -0.022018 7.003093
7.749855 4.876606 -3.430427
8.436680 -2.293254 4.593714
6.085982 3.746046 -8.936503
-5.741400 5.623232 3.052342
6.912261 4.081523 3.732992
7.613192 8.574176 -4.262436
6.669372 9.300007 -7.609860
-1.163105 0.872522 1.246592
7.350769 -9.844899 -8.670342
5.646411 -4.495377 -0.767826
-7.093240 9.181814 -0.134587
-8.615986 8.122427 1.789899
-4.921552 -8.158994 -9.398825
7.337864 -7.570210 -5.599539
1.095523 -8.670660 4.438330
2.918037 2.984859 -2.858530
-4.318831 5.426273 9.612534
1.913539 8.266542 -6.841856
5.250707 -5.308670 -4.433651
-9.382769 8.760746 -6.976341
-5.436702 -3.069899 -1.397641
6.276620 -8.373912 -8.543192
-5.590807 7.593916 6.544241
7.965868 0.165678 -9.884276
2.148984 -4.402173 -7.355830
-3.995888 4.504433 1.770295
4.767536 2.484077 1.756728
1.046319 5.478482 3.666757
3.373392 -3.137418 -5.698525
5.551244 2.689869 5.224560
-3.652149 -9.653293 5.623532
-6.596457 5.035353 -8.929488
-8.080850 -7.657539 -7.139089
6.242183 -9.510069 -3.880598
0.973404 0.710130 3.573529
0.746364 -7.280267 -8.427422
5.546878 -4.284904 -7.005796
-4.241812 -7.503171 5.292118
-5.749966 1.677797 -1.741499

B.3 Dataset 3

X Y Z
-8.854786 -5.716439 5.809892
-10.307003 -5.495722 5.160276
-4.964017 -5.038003 5.672859
-8.568304 -5.931836 4.970339
-8.677818 -4.781844 5.786936
-7.329258 -5.463592 6.049749
-6.741196 -4.718684 5.520261
-8.469856 -4.723265 7.960577
-8.556637 -5.507349 7.027803
-9.173477 -5.555510 6.908028
-7.706667 -5.172273 6.695594
-11.115179 -4.242483 5.065845
-9.955636 -4.987344 6.726034
-8.731784 -6.340196 6.051614
-8.339764 -3.858192 6.772897
-8.093201 -4.152157 5.819966
-9.212916 -4.679732 6.608028
-8.501846 -6.046598 6.254643
-10.780911 -4.573665 6.516858
-9.925874 -5.202436 5.061390
-7.529796 -7.297957 5.110995
-7.577437 -4.107389 3.867484
-8.785057 -4.935770 5.416273
-6.696376 -6.455017 6.380021
-9.335422 -3.857827 6.003838
-8.659267 -6.981975 7.707315
-7.534238 -4.510226 7.163371
-7.937790 -5.903036 5.225248
-9.087217 -3.893718 4.565108

```
-8.883386 -5.585034 6.250108
-7.711667 -6.512887 4.295476
-8.291431 -4.050226 6.782547
-8.071366 -5.580946 6.318723
-8.530352 -4.446583 5.231042
-8.152777 -5.374570 6.568849
-8.062378 -2.713755 7.524144
-6.790279 -7.303294 5.685173
-8.895627 -4.731068 3.746463
-7.112518 -4.196807 4.723418
-9.265565 -6.064962 6.068785
-7.646046 -3.215904 5.828044
-7.519514 -5.108372 7.785416
-7.544859 -3.895239 4.947814
-8.479257 -6.077562 7.530437
-7.629377 -5.568934 5.573021
-7.802352 -5.965285 5.094900
-7.805742 -2.800658 5.779356
-8.842882 -6.434947 4.690478
-7.762033 -4.412987 6.034664
-7.954441 -5.147873 6.164144
-9.813176 -5.721908 6.136969
-9.070183 -5.740220 5.206369
-8.620513 -4.410682 5.618909
-8.440887 -4.450072 6.670260
-6.591809 -7.354275 6.882348
-8.769301 -5.129098 6.863209
-7.203764 -4.224439 6.180596
-6.677354 -5.546764 5.884507
-7.487665 -5.815162 8.186396
-7.267810 -3.088215 5.998900
-8.670105 -5.096742 6.760066
-8.874444 -4.884088 6.008685
-6.671335 -5.926492 7.071651
-8.930880 -6.224424 5.315190
-9.477203 -5.117440 7.056750
-8.122702 -4.639465 7.657663
-8.016990 -5.068628 5.750428
-9.895259 -4.504051 4.269029
-7.634340 -5.278110 7.152271
-8.354787 -6.086990 6.383400
-8.847253 -5.444747 6.067360
-8.421566 -5.452592 5.193044
-8.476150 -7.402130 5.868157
-9.485073 -4.143526 7.295416
-10.238042 -5.118873 5.899125
-9.333732 -4.732137 5.563800
-10.054616 -5.530467 5.877236
-8.180586 -6.495023 6.641170
-7.551560 -6.409617 5.366634
-8.367354 -5.829715 7.769244
-8.078645 -6.274372 5.237380
-8.344487 -2.968011 5.847225
-8.159540 -4.749744 5.985363
-6.006356 -5.795315 6.769692
-8.126579 -4.642593 4.848846
-6.518904 -5.385317 6.059653
-9.221993 -5.993485 6.589729
-7.300517 -4.510535 7.234460
-7.572528 -5.235328 6.862015
-7.693413 -5.364365 6.751562
-7.001319 -5.029219 5.669284
-7.568305 -3.363068 5.814927
-8.379326 -5.400187 7.248507
-10.124215 -4.897944 7.217753
-9.100400 -3.774812 6.560798
-8.836053 -5.059650 4.490997
-8.852588 -3.436886 5.131685
-6.442445 -5.345363 7.020152
-8.243120 -7.505271 5.452838
-8.097163 -6.414740 4.678079
-7.920831 -5.913205 4.329662
-8.971132 -4.403052 5.551743
-7.352244 -3.736517 6.058005
-9.416700 -6.565355 5.999235
-7.405961 -4.994775 6.572746
-8.730358 -4.963367 5.554181
-8.612627 -6.556036 4.453276
-8.768973 -6.371389 5.066744
-7.615969 -5.370871 8.939615
-7.360675 -5.896977 6.803497
-7.930213 -5.949471 7.128084
-9.062345 -8.322171 6.819554
-8.904276 -3.777752 5.375817
-9.468371 -3.715328 4.949991
-8.083623 -3.814063 6.139987
-8.367602 -5.161366 6.864543
-9.093829 -4.653149 6.602565
-5.954305 -5.600315 5.110659
-7.551330 -5.660947 6.314605
-8.154440 -4.095396 5.368652
-8.379296 -7.815903 5.135375
-6.750678 -5.945296 5.324012
-9.504924 -4.683956 6.203479
-9.473884 -4.596339 6.285595
-7.532983 -5.905269 7.763498
-7.877149 -5.380679 5.506655
-8.531844 -4.404998 5.756523
-7.208873 -4.683114 7.371984
-7.472912 -5.326729 4.116149
-7.988446 -4.474714 6.188382
-7.730585 -5.735410 5.645012
-9.229406 -3.901889 6.183884
-8.779503 -6.595642 5.872182
-6.844385 -4.801868 5.300968
-6.857522 -3.835280 7.006265
-7.498200 -4.108138 6.568748
-7.512016 -6.043815 7.525047
-8.055836 -3.239578 3.983241
-7.581608 -4.282328 5.682557
-7.834572 -4.515463 6.602871
-8.826676 -5.405728 7.044529
-7.009697 -5.164813 6.102522
-8.086520 -4.226125 4.971451
-9.616082 -5.139475 4.918961
-8.874126 -5.177494 6.507994
-9.359108 -4.462184 4.251775
-7.741601 -6.841522 4.750566
-7.644795 -5.670152 5.229931
-8.731793 -4.798757 5.971443
-8.316399 -4.209013 5.309099
-8.269754 -3.216905 4.090395
-9.417963 -3.651208 3.729734
-8.430494 -3.779689 3.425828
-9.344355 -4.623232 5.857486
-6.673337 -4.603909 4.138210
-11.542744 -4.292449 4.117113
-10.334004 -3.718387 2.558392
-9.476013 -4.365125 2.875146
-8.404419 -3.448420 2.737725
-10.668999 -3.901722 4.093649
-10.472563 -4.799208 4.016629
-9.404199 -3.676493 4.508344
-7.491696 -3.129621 4.802620
-8.184081 -3.180162 4.830141
-10.486902 -3.644564 4.196418
-7.642349 -4.410507 4.650138
-8.931520 -2.822756 4.301328
-9.530700 -4.194890 3.684638
-9.099121 -4.511453 4.395768
-7.337876 -4.909672 4.161119
-9.460652 -3.831370 2.942571
-9.953890 -4.543471 4.670294
-9.307441 -5.004268 3.257318
-8.749407 -3.061783 4.447021
-9.073025 -4.060108 3.181493
-9.572835 -4.771908 4.419386
-9.293186 -5.720381 5.249375
-7.568752 -4.767178 4.522987
-8.993998 -3.719318 2.644363
-7.648150 -4.971834 5.388372
-9.180489 -4.893537 3.983693
-8.363987 -5.367257 3.748968
-9.743340 -6.219430 5.424571
-8.682271 -4.525296 5.287809
-10.217590 -4.577431 5.849549
-8.281494 -6.729432 0.792065
-10.299064 -3.442698 3.651700
-9.478904 -3.895710 3.238944
-9.048282 -3.711883 4.591159
-8.865215 -5.106027 4.638433
-9.688790 -3.199034 6.694239
-10.001066 -6.678483 3.698706
-8.884315 -3.453850 5.203161
-10.045111 -5.297217 5.391447
-10.353548 -3.413750 3.897258
-8.494756 -3.797817 4.900798
-8.440733 -5.493372 4.421667
-9.247178 -4.195573 5.357394
-9.532268 -5.215952 3.396030
-9.571978 -3.751371 3.309330
-8.101902 -4.213530 3.898035
-10.592811 -4.259048 4.815615
-11.026926 -3.312022 5.771893
-8.922658 -4.012200 6.537841
-8.029269 -5.401739 4.107501
-9.418091 -4.654858 1.759878
-8.504155 -5.140717 4.287797
-9.397299 -3.632886 3.828337
-10.056588 -3.971559 5.529032
-9.580971 -3.134033 4.386329
-9.793890 -2.978824 3.305519
-8.247112 -5.002167 4.574364
-10.495308 -5.341574 5.189198
-9.073043 -4.465688 3.684338
-8.814679 -5.901251 3.004211
-9.418950 -5.338262 3.423552
-9.332663 -4.641503 3.901639
-7.595884 -3.492906 5.353880
-9.289322 -5.008666 2.670247
-9.609297 -6.589937 3.540404
-8.496921 -2.882451 5.089753
-10.122760 -4.011363 3.334982
-8.438186 -4.615342 4.982346
-6.688264 -3.581457 6.456596
-9.083718 -4.833699 5.457850
-7.536803 -5.058996 4.200778
-10.695690 -4.704027 5.012478
-9.347545 -5.073201 2.716906
-8.892906 -3.706441 4.720646
-9.005844 -5.778047 5.885954
-9.317696 -4.648778 4.857074
-11.424346 -4.568449 3.987356
-8.888877 -3.156821 4.596346
-11.093258 -4.843058 4.329678
-9.113993 -5.785244 2.460387
-8.712149 -4.160446 4.102845
-10.986582 -2.729931 1.223912
-9.625257 -2.412830 3.966720
-9.029090 -5.683128 5.482786
-10.553783 -4.249562 4.259051
-10.311885 -3.617002 2.708188
-9.944645 -4.532684 3.711648
-8.793353 -3.579926 3.830372
-9.802626 -5.020960 3.758552
-8.451511 -4.721566 4.531785
-9.352313 -5.348446 4.987866
-10.299201 -4.497784 5.611170
-10.354267 -4.559170 2.539365
```

```
-9.770576 -5.998581 3.762559
-9.799457 -4.372010 5.708447
-8.978176 -3.845807 4.073444
-9.505684 -5.337963 2.323289
-9.192540 -2.874858 4.600446
-9.795149 -2.854575 3.691522
-9.880000 -5.716676 3.162095
-9.507523 -2.504961 3.917970
-10.292676 -5.109400 3.901284
-10.972912 -4.377353 4.824814
-8.940348 -3.903249 2.834599
-10.959752 -3.218407 5.218477
-10.801424 -4.297600 4.149674
-8.716274 -4.295395 3.622014
-9.065903 -4.346934 4.075518
-11.113579 -7.499182 3.333957
-10.895932 -4.251419 4.840243
-8.998045 -4.873509 4.194627
-8.093819 -5.232789 5.046441
-8.553544 -4.242297 5.576076
-9.978979 -3.561475 3.195194
-10.028192 -5.563692 4.569977
-10.200920 -4.451821 2.987434
-8.194902 -2.667808 4.350222
-7.617795 -3.943318 6.293456
-9.161446 -6.074405 5.614730
-9.387527 -5.539826 3.467880
-10.887492 -4.671683 3.052882
-7.776979 -3.839489 2.463740
-10.770532 -5.070957 2.199105
-9.370966 -4.547429 3.351339
-9.807654 -4.005128 5.545830
-7.765487 -4.469672 4.979995
-9.999515 -4.092028 3.164474
-8.531587 -5.742383 3.159063
-9.269505 -3.946992 2.685971
-9.797799 -4.368830 4.353192
-10.099959 -4.407099 3.856909
-7.857331 -3.947083 2.800050
-11.462717 -4.404100 5.306230
-9.087420 -3.621915 3.639472
-10.625081 -3.755388 4.223088
-8.577477 -5.885547 4.305699
-9.886661 -3.697392 3.543458
-9.868555 -5.089702 5.387863
-9.232296 -5.263978 3.101562
-10.305282 -3.041354 2.309371
-10.111518 -3.866091 5.302642
-9.409437 -3.718465 4.883763
-8.274973 -4.457566 5.779114
-9.045035 -2.955781 3.041357
-8.300547 -5.739018 2.648562
3.707938 3.162093 8.163918
3.652921 1.737377 9.042584
4.020309 2.083467 10.211852
5.662680 1.560965 7.598979
5.376333 0.057312 5.132026
4.133798 0.493251 9.918344
4.943081 0.593314 7.013297
5.305336 0.750506 7.596204
4.716825 1.672080 8.279050
4.926617 2.889549 9.887183
3.162280 4.168001 8.317635
6.062055 0.853501 8.515817
5.034701 1.707848 9.044219
5.452930 0.590302 8.075082
5.550657 2.283633 7.857535
4.541584 1.443006 7.089415
3.534965 3.166340 7.798676
4.917878 0.849000 9.491283
6.574829 1.377132 8.708249
4.067539 3.362065 9.119646
3.982576 2.844891 8.553759
5.235901 1.953964 7.524232
4.283162 2.217926 8.142008
3.871384 3.625067 7.980389
4.258786 0.894536 8.878943
5.791860 1.837112 9.056930
3.317049 1.218582 7.716132
4.819871 2.077039 6.145058
7.938466 0.719062 8.527092
5.637861 2.434960 8.420034
1.896156 0.492299 8.933549
3.879942 0.518315 5.813549
6.013764 2.078245 7.269530
4.656258 1.460688 7.039337
4.508100 1.808965 8.702265
6.042859 1.578766 7.682753
5.175140 0.916259 6.890103
5.913012 1.162055 8.260393
5.417299 0.434215 8.779575
5.217202 1.443022 8.532333
4.009619 -0.664910 7.854083
3.854579 2.408725 8.401345
4.235583 1.587313 9.489590
5.900261 1.908620 8.567031
4.486787 1.408854 8.725174
4.645010 1.837852 7.019581
3.094872 -0.160974 7.697239
6.091485 2.721313 7.137075
3.206388 2.330615 8.597490
3.393927 2.595280 8.308751
6.086647 1.879690 9.024444
5.084039 0.584071 7.900614
4.084176 2.366970 7.136740
3.227762 2.218528 9.279436
6.207104 2.628517 6.624009
4.063433 0.986212 8.108482
2.605223 1.524184 8.407305
3.808159 1.651349 9.321547
4.804736 2.590107 9.143940
5.168843 1.916650 7.758030
5.999641 1.061974 7.432455
5.464518 0.806963 8.555868
5.815820 1.614073 8.236021
5.677916 2.288004 6.591062
3.449541 2.866652 8.522629
2.679990 0.187547 8.158599
3.782576 2.006940 8.213686
5.255073 0.371074 8.757839
6.725523 0.745853 9.452328
5.749312 2.186708 8.073925
3.606624 0.665547 8.904632
5.748802 2.767374 9.102022
5.673700 1.110456 6.934096
4.974955 2.045782 8.228207
3.250786 1.025473 6.665465
4.917180 0.495865 7.976028
5.094458 1.683822 9.512114
3.667471 -0.636344 9.089398
6.175719 2.980503 8.430354
5.313986 1.672372 6.851843
4.204836 2.047445 7.570731
6.684436 0.010737 7.714717
5.887725 0.944442 7.016530
3.418512 0.825532 9.659464
5.187245 2.069545 7.630773
5.623264 1.660561 7.229652
4.327266 1.756803 9.321975
4.721910 2.054777 10.171356
3.981838 0.314543 9.135467
5.219330 1.457517 8.416705
4.451949 2.072138 7.817278
5.581858 0.385964 7.336850
4.898730 1.087191 8.849601
4.740719 1.015352 8.364470
4.255729 0.000888 7.019537
3.983077 1.089614 8.254611
5.697183 2.113061 9.939123
6.063285 1.375147 9.812095
5.971394 1.111976 8.473945
5.886077 1.808343 9.179948
5.188171 2.271018 7.846302
5.172447 1.204360 8.841112
3.105134 1.903454 9.415833
5.374643 1.543356 8.704654
3.948747 0.737411 7.806821
4.646541 2.534905 7.271505
5.794972 -0.004117 7.378462
4.757287 2.726273 7.551614
2.650341 1.988355 8.136040
3.711570 0.524364 7.469308
4.891819 2.231519 6.686398
7.082207 1.529920 7.791550
5.465841 1.676233 10.500292
5.888691 1.198440 7.773018
4.294131 1.224854 8.835926
4.145729 3.179044 8.080651
4.645513 2.514448 6.182855
3.854739 2.163881 8.173092
4.778099 1.961480 7.560138
4.618883 0.202979 6.285410
6.053918 0.630557 7.443835
6.184866 2.622249 7.632790
4.256715 1.479997 7.203489
5.335231 1.706317 8.287470
5.539806 2.279525 9.999531
6.302853 -0.149465 9.055138
3.667307 0.132224 8.868219
4.083449 1.244257 9.226989
4.122247 0.664875 8.483714
6.008794 2.088910 6.603270
4.134540 1.408015 10.087200
5.678955 2.411589 9.359847
4.476551 2.039863 8.647297
3.808751 1.543420 8.721133
3.705934 2.368362 8.915714
5.819402 0.785349 8.264211
6.238852 2.352719 7.368480
5.010497 2.980246 8.080155
5.165908 4.127897 8.118917
4.676341 0.552611 7.495945
6.268298 1.417578 6.966386
3.849517 1.560804 8.916292
4.034866 2.266083 7.660189
5.131996 2.068829 8.752273
4.340460 1.376228 8.646903
3.473954 1.712809 8.649297
5.806443 1.498325 7.946499
4.899687 0.798455 7.917858
5.163180 1.809110 7.079215
4.424536 -0.590038 8.920207
8.398523 9.081708 2.044797
8.277949 -1.597589 -4.059854
6.226973 -3.431815 -4.706017
-0.453883 -1.891314 9.162305
7.980638 4.455965 7.887827
-4.966201 -4.751069 -3.099365
1.926607 -9.511409 -8.020028
-4.954474 -8.071841 -4.967504
-7.951491 -4.063394 -1.295444
0.902570 6.088687 -4.189880
-2.305399 7.361593 -9.634558
0.273564 -7.633183 -9.730670
7.727879 3.314514 -0.686556
6.205495 9.119961 1.622141
-0.882354 -9.984418 8.961357
5.537448 -7.885743 -1.052403
-5.069915 8.943153 -3.937564
```

```
-7.149070 7.973976 -3.921904
-7.404706 -8.756661 -6.740832
-1.320487 -4.983619 3.519950
7.796405 -7.068599 -9.238528
5.202339 3.261394 7.363333
-7.630175 3.967844 -4.354636
-6.292782 -2.408364 6.987902
6.347308 6.785827 -5.092601
-0.962806 -6.480744 -5.564292
6.860419 -7.863044 8.353159
9.273942 -1.235304 1.954633
9.272880 4.422808 5.786735
4.866068 -8.885014 5.666868
3.966051 -3.204477 -9.126338
-2.933987 1.065583 -2.784215
1.626031 -2.058082 3.902894
8.724619 -9.700046 2.373529
6.210639 -9.517860 -0.829457
6.133550 3.681714 -2.335204
-3.172214 -9.349532 0.007656
6.409706 6.497637 4.270334
-0.104180 -5.076945 -4.363854
-6.547575 -3.410910 -9.876245
-0.924677 5.653264 -3.385621
7.092045 1.139581 5.687071
9.917849 -7.896258 -1.888752
-6.700397 -7.306821 4.316609
-9.905844 1.875265 8.329052
-1.576753 3.735620 -7.368841
5.070794 5.492258 1.717103
8.334698 1.012019 1.907738
-9.666818 6.432783 -3.117010
3.859745 -0.154717 -6.894812
```