```
#ALGORITHM


import numpy as np
from scipy.sparse.linalg import svds
from scipy.linalg import eig
from sklearn.cluster import KMeans

def kmeans_clustering(X, k):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X.T)
    return kmeans.cluster_centers_.T

def compute_sparse_representation_matrix(X, k, m):
    W = kmeans_clustering(X, k)

    # Compute distances between data points and anchors
    distances = np.linalg.norm(X[:, np.newaxis, :] - W.T[np.newaxis, :, :].T,
                               axis=2)

    # Find the indices of the k-nearest neighbors for each data point
    indices = np.argsort(distances, axis=1)[:, :k]

    # Ensure that the indices are within the valid range
    indices = np.clip(indices, 0, m-1)

    # Construct the sparse representation matrix B
    B = np.zeros((X.shape[1], m))
    for i in range(X.shape[1]):
        for j in indices[i]:
            B[i, j] = np.linalg.norm(X[:, i] - W[:, j])**2 / 2.0

    return B

def update_Y(X, R, k):
    Y = np.random.rand(X.shape[1], k)
    return Y

def LABIN(X, k, m, c, max_iter=100, tol=1e-5):
    # Step 2: Find m anchors W with k-means
    B = compute_sparse_representation_matrix(X, k, m)
    D = np.diag(np.sum(B, axis=1))
    P = np.linalg.inv(D) @ B

    Y_star = np.random.rand(X.shape[1], c)

    for _ in range(max_iter):
        # Step 4: Reduced SVD on Q
        Q = np.concatenate((P, np.ones((P.shape[0], 1))), axis=1)
        U, _, _ = svds(Q, c)

        # Step 8: Update according to Eq. (32) and perform eigendecomposition
        L, V = eig(U.T @ P @ P.T @ U)
        idx = np.argsort(L)[::-1][:c]
        Lambda = np.diag(L[idx])
        V = V[:, idx]

        # Step 9: Update Y*
        Y_star = U @ np.sqrt(Lambda) @ V.T
        Y_star = Y_star[:, :k]
        # Ensure Y_star has the same number of columns as Y

        for _ in range(max_iter):
            # Step 11: Update R
            R = np.dot(Y_star, Y_star.T)

            # Step 12: Update Y
            Y = update_Y(X, R, k)

            # Check convergence of problem (38)
            delta_Y = Y
            if np.linalg.norm(delta_Y, 'fro') < tol:
                break

            Y_star = Y
        # Check convergence of problem (26)
        if np.linalg.norm(delta_Y, 'fro') < tol:
            break

    return Y

# Example usage:
np.random.seed(42)
d, n = 10, 10
X = np.random.rand(d, n)

k = 5
m = 5
c = 3

Y_result = LABIN(X, k, m, c)
print("Clustering Result (Y):")
print(Y_result)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10
  warnings.warn(
Clustering Result (Y):
[[0.8532323  0.37699328 0.96133042 0.66194453 0.19748228]
 [0.46442029 0.62211098 0.78020445 0.44809762 0.73214942]
 [0.02636059 0.58710496 0.70743487 0.84272019 0.89076978]
 [0.42074144 0.18779224 0.94423786 0.53645322 0.35240675]
 [0.65442298 0.68360165 0.11293953 0.01080132 0.52705449]
 [0.01118825 0.6537754  0.61098779 0.61376241 0.28312779]
 [0.91167307 0.28685775 0.06915251 0.99139044 0.84967925]
 [0.31920404 0.71766638 0.092997   0.35444388 0.77967571]
 [0.61091089 0.97996182 0.24764277 0.29536368 0.83295887]
 [0.92155292 0.3731059  0.07286093 0.11012055 0.48025622]]
```