ML PROGRAMMING ASSIGNMENT 3

CS 545: Machine Learning

Spring 2024

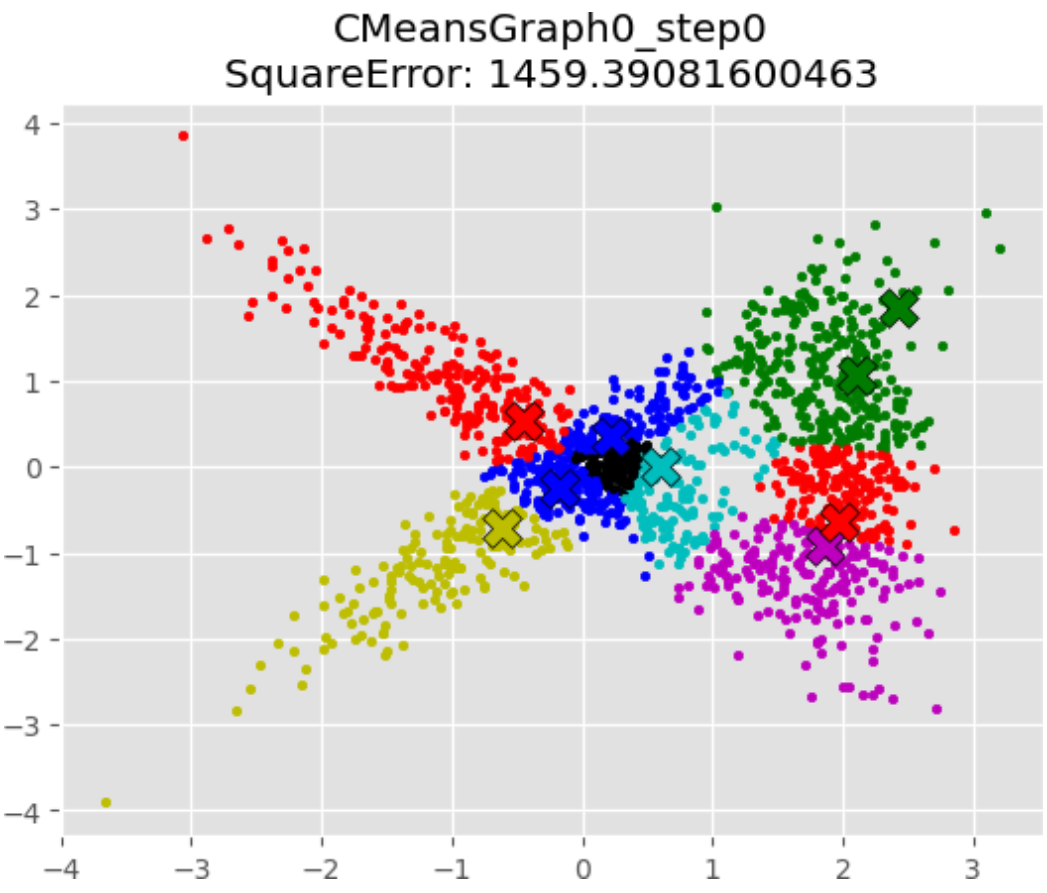
Name: Srihari Tanmay Karthik Tadala

PSU ID: 918597835

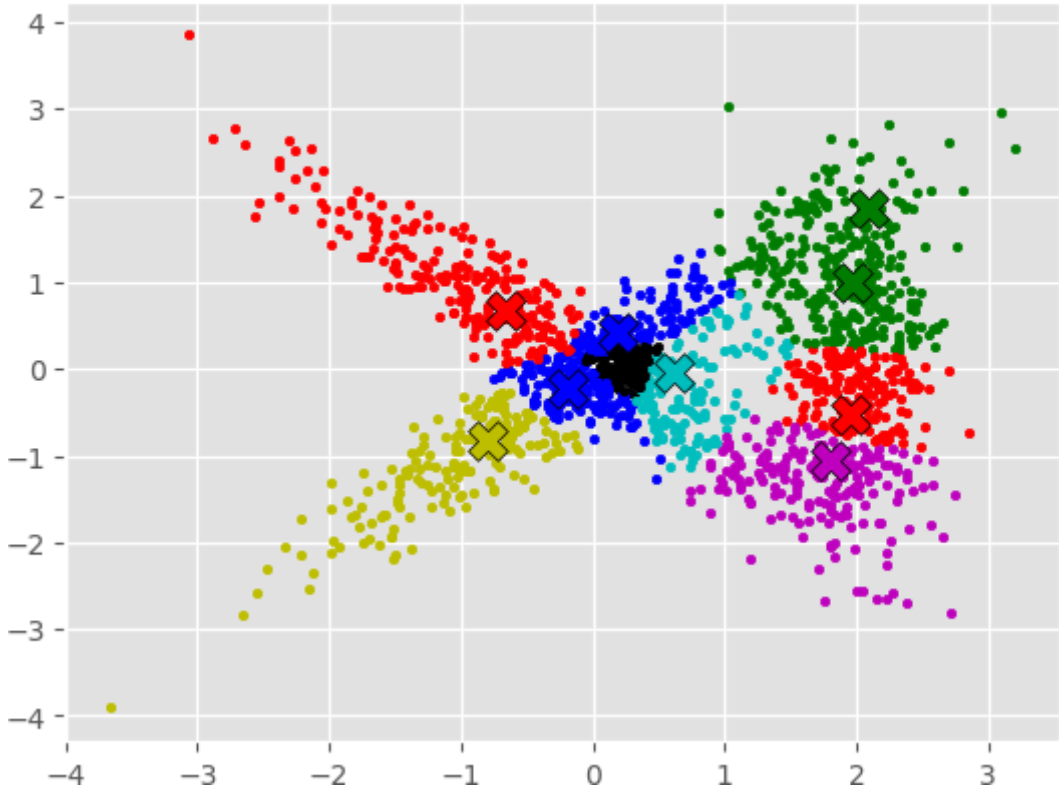
```python
import numpy as np
import matplotlib.pyplot as plt
import random
from math import sqrt
plt.style.use('ggplot')

class FuzzyCluster:
    def __init__(self):
        self.cluster_centers = []
        self.memberships = []
        self.error_history = []
        self.optimal_centers = []
        self.optimal_memberships = []
        self.minimal_error = float('inf')
        self.fuzziness = 2
        self.current_error = 0
        self.tolerance = 0.001
        self.color_palette = ['r', 'g', 'b', 'm', 'c', 'k', 'y'] * 10

    def display_errors(self):
        print("All error sums over iterations:")
        print(self.error_history)
        print("Minimal error sum encountered:")
        print(self.minimal_error)

    def evaluate_errors(self):
        self.error_history.append(self.current_error)
        if self.current_error < self.minimal_error:
            self.minimal_error = self.current_error
            self.optimal_centers = self.cluster_centers.copy()
            self.optimal_memberships = self.memberships.copy()
            print("New lowest error sum found:", self.minimal_error)

    def initialize_memberships(self, data_points, num_clusters):
        self.memberships = []
        for _ in range(len(data_points)):
            weights = random.sample(range(1, 1000), num_clusters)
            total = sum(weights)
            self.memberships.append([weight / total for weight in weights])

    def update_centers(self, data_points, num_clusters):
        self.cluster_centers = [[0, 0] for _ in range(num_clusters)]
        for cluster_index in range(num_clusters):
```

```
    def update_centers(self, data_points, num_clusters):
        self.cluster_centers = [[0, 0] for _ in range(num_clusters)]
        for cluster_index in range(num_clusters):
            num = den = 0
            for data_index, data_point in enumerate(data_points):
                weight = self.memberships[data_index][cluster_index] ** self.fuzziness
                num += weight * data_point
                den += weight
            self.cluster_centers[cluster_index] = [num / den, num / den] if den else [0, 0]

    def update_memberships(self, data_points, num_clusters):
        for data_index, data_point in enumerate(data_points):
            for cluster_index in range(num_clusters):
                num = self.cost_function(data_point, self.cluster_centers[cluster_index])
                den = sum((num / self.cost_function(data_point, center)) ** (2 / (self.fuzziness - 1))
                          for center in self.cluster_centers)
                self.memberships[data_index][cluster_index] = 1 / den if den else 0
                self.current_error += num * self.memberships[data_index][cluster_index]

    def cost_function(self, point, center):
        return sqrt((point[0] - center[0])**2 + (point[1] - center[1])**2)

    def select_initial_centers(self, num_clusters, upper_bound, data_points):
        selected_indices = random.sample(range(0, upper_bound), num_clusters)
        self.cluster_centers = [data_points[i] for i in selected_indices]

    def plot_clusters(self, data_points, iteration, step):
        plt.figure()
        for i, point in enumerate(data_points):
            plt.scatter(*point, color=self.color_palette[np.argmax(self.memberships[i])])
        for center in self.cluster_centers:
            plt.scatter(*center, s=200, color='black', marker='X')
        plt.title(f"Clustering Progress at Iteration {iteration}, Step {step}")
        plt.show()

    def run_clustering(self, iterations, num_clusters, data_points, max_steps):
        self.select_initial_centers(num_clusters, len(data_points), data_points)
        for iteration in range(iterations):
            for step in range(max_steps):
                self.update_memberships(data_points, num_clusters)
                self.update_centers(data_points, num_clusters)
                if step % 10 == 0:  # Optionally display progress every 10 steps
```

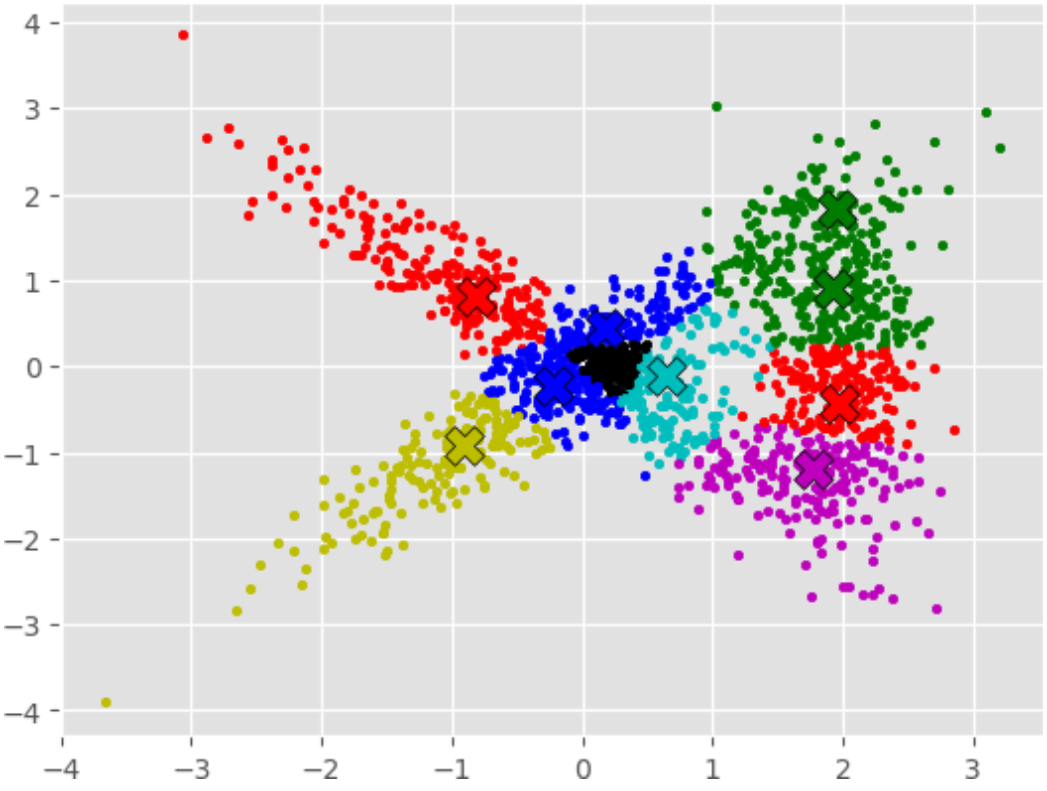
```
    def cost_function(self, point, center):
        return sqrt((point[0] - center[0])**2 + (point[1] - center[1])**2)

    def select_initial_centers(self, num_clusters, upper_bound, data_points):
        selected_indices = random.sample(range(0, upper_bound), num_clusters)
        self.cluster_centers = [data_points[i] for i in selected_indices]

    def plot_clusters(self, data_points, iteration, step):
        plt.figure()
        for i, point in enumerate(data_points):
            plt.scatter(*point, color=self.color_palette[np.argmax(self.memberships[i])])
        for center in self.cluster_centers:
            plt.scatter(*center, s=200, color='black', marker='X')
        plt.title(f"Clustering Progress at Iteration {iteration}, Step {step}")
        plt.show()

    def run_clustering(self, iterations, num_clusters, data_points, max_steps):
        self.select_initial_centers(num_clusters, len(data_points), data_points)
        for iteration in range(iterations):
            for step in range(max_steps):
                self.update_memberships(data_points, num_clusters)
                self.update_centers(data_points, num_clusters)
                if step % 10 == 0:  # Optionally display progress every 10 steps
                    self.plot_clusters(data_points, iteration, step)
                self.evaluate_errors()
            if abs(self.previous_error - self.current_error) <= self.tolerance:
                print("Convergence reached.")
                break
        print("Final clustering achieved.")
        self.display_errors()

def main():
    fuzzy = FuzzyCluster()
    with open("/content/545_cluster_dataset programming 3.txt", 'r') as file:
        data = [list(map(float, line.strip().split())) for line in file.readlines()]
    clusters = int(input("Enter the number of clusters to generate (1-50): "))
    iterations = int(input("Enter the number of iterations to run: "))
    fuzzy.run_clustering(iterations, clusters, data, 100)

if __name__ == "__main__":
    main()
```

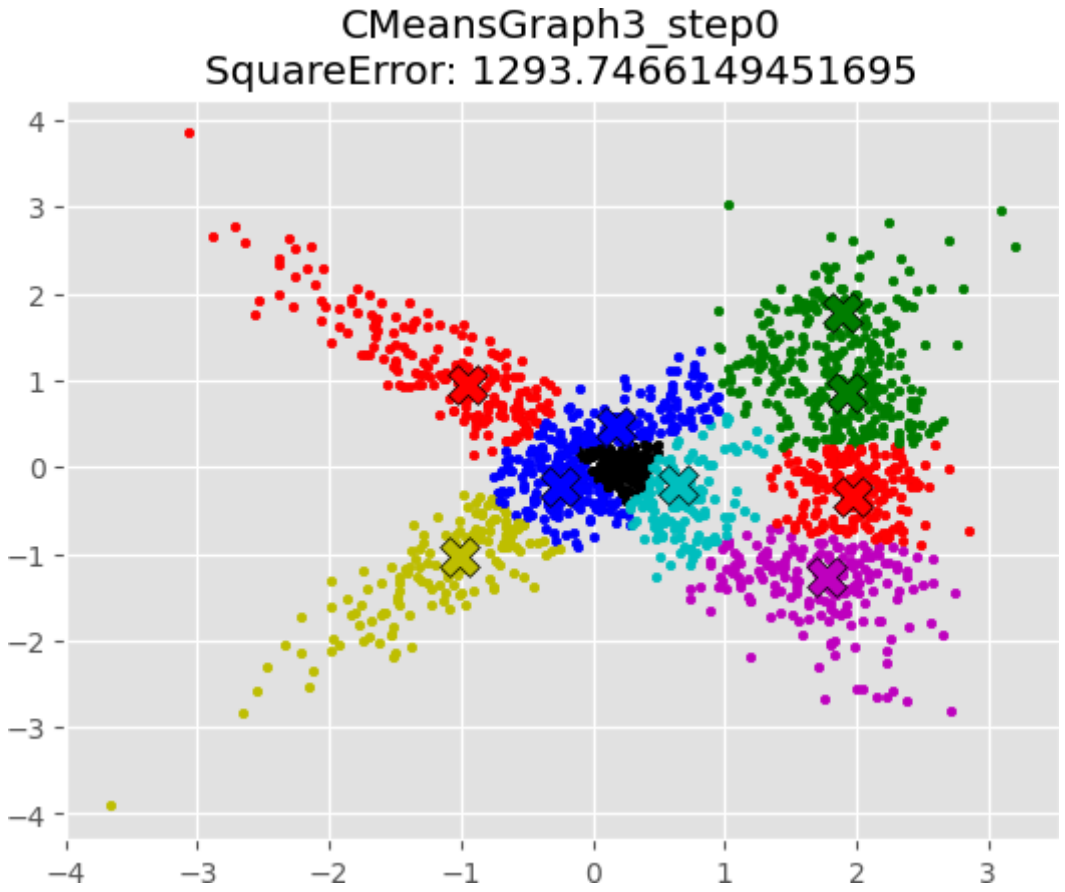


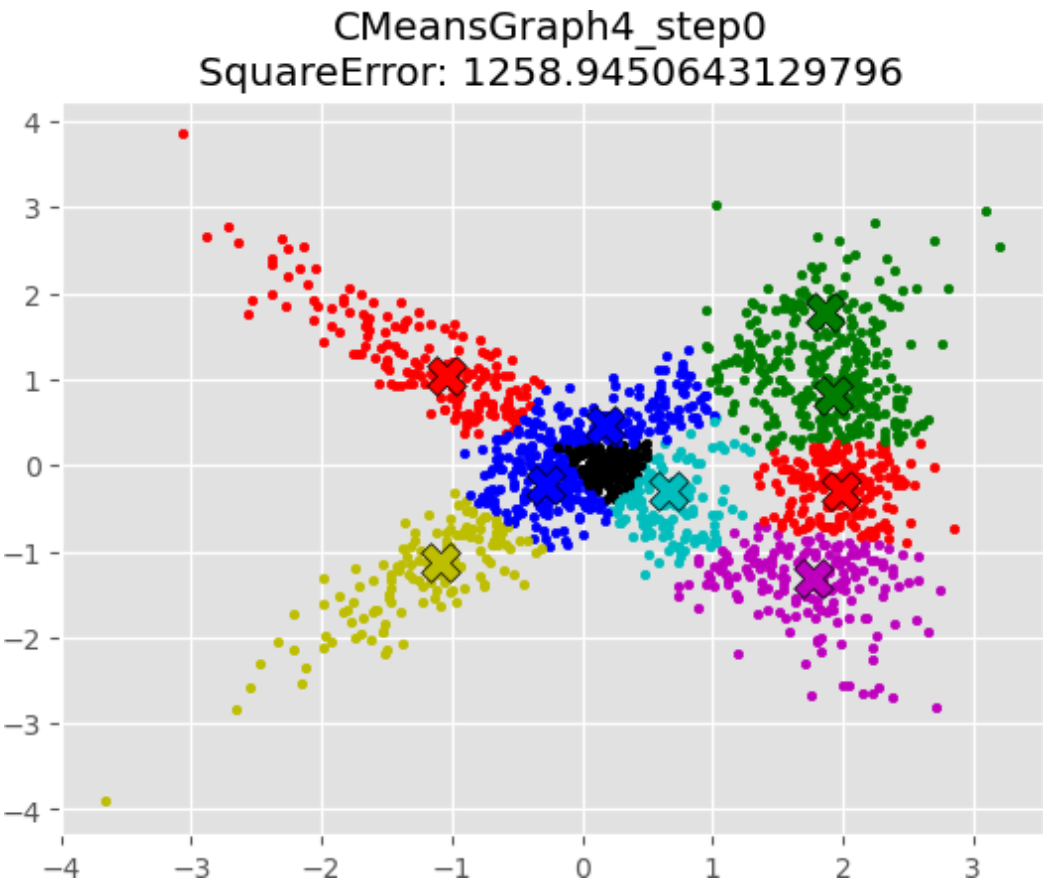
CMeansGraph1_step0
SquareError: 1459.3908153132081



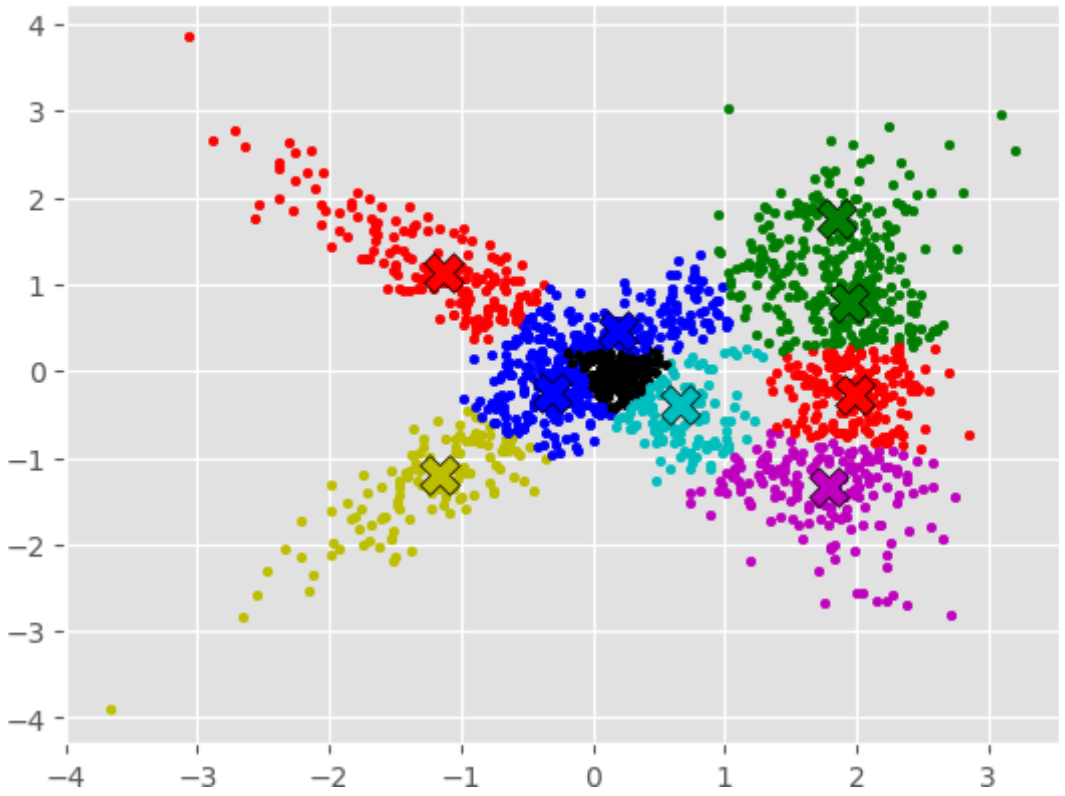
CMeansGraph2_step0
SquareError: 1352.5365514257787



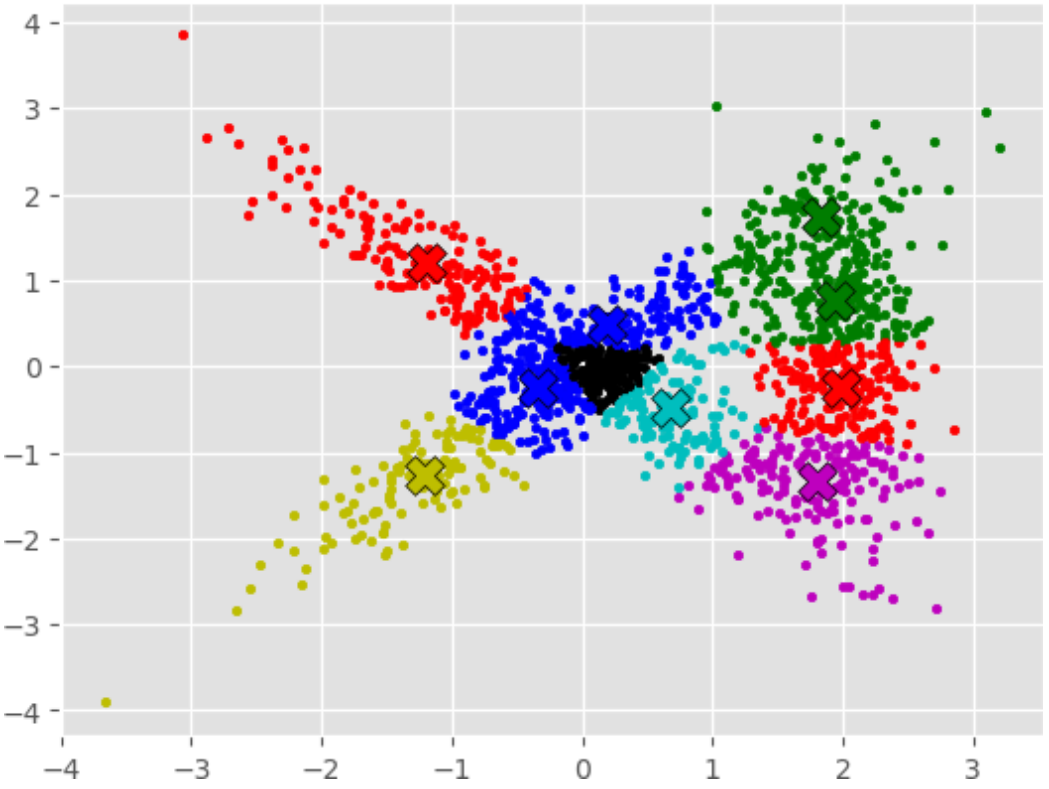

CMeansGraph3_step0
SquareError: 1293.7466149451695


CMeansGraph4_step0
SquareError: 1258.9450643129796

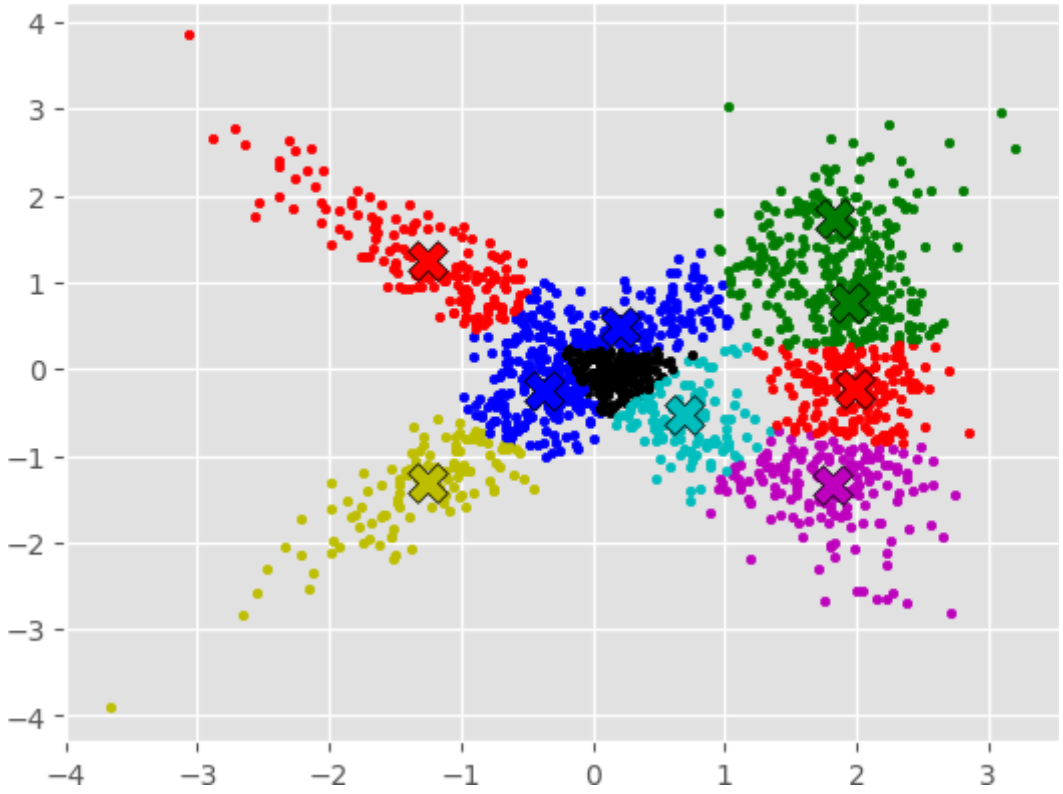
CMeansGraph5_step0
SquareError: 1237.395993146971



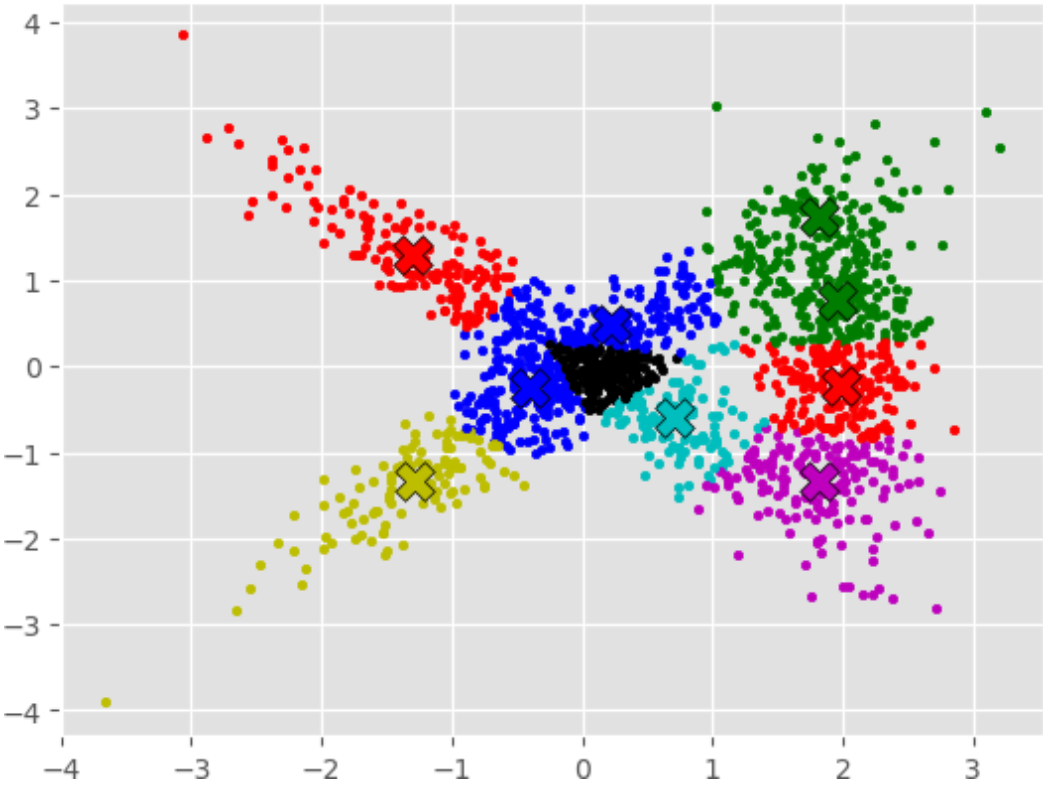
CMeansGraph6_step0
SquareError: 1224.4004588785558

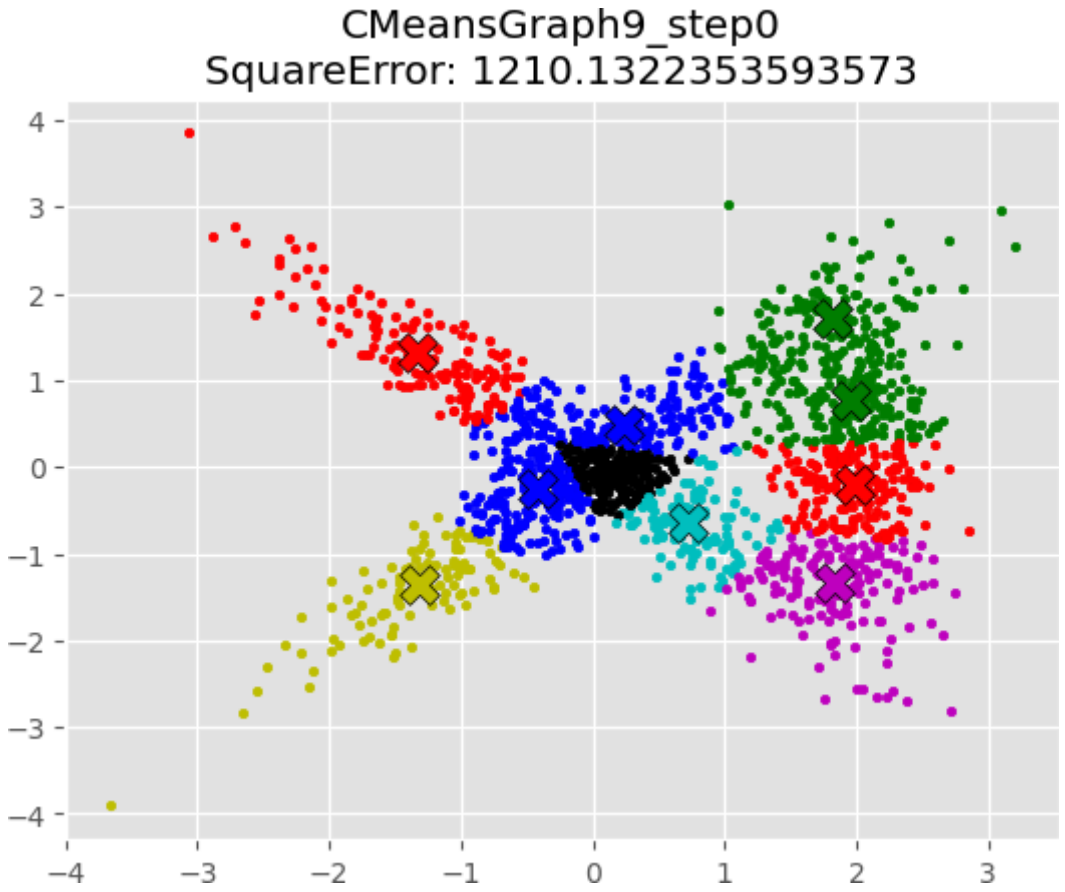


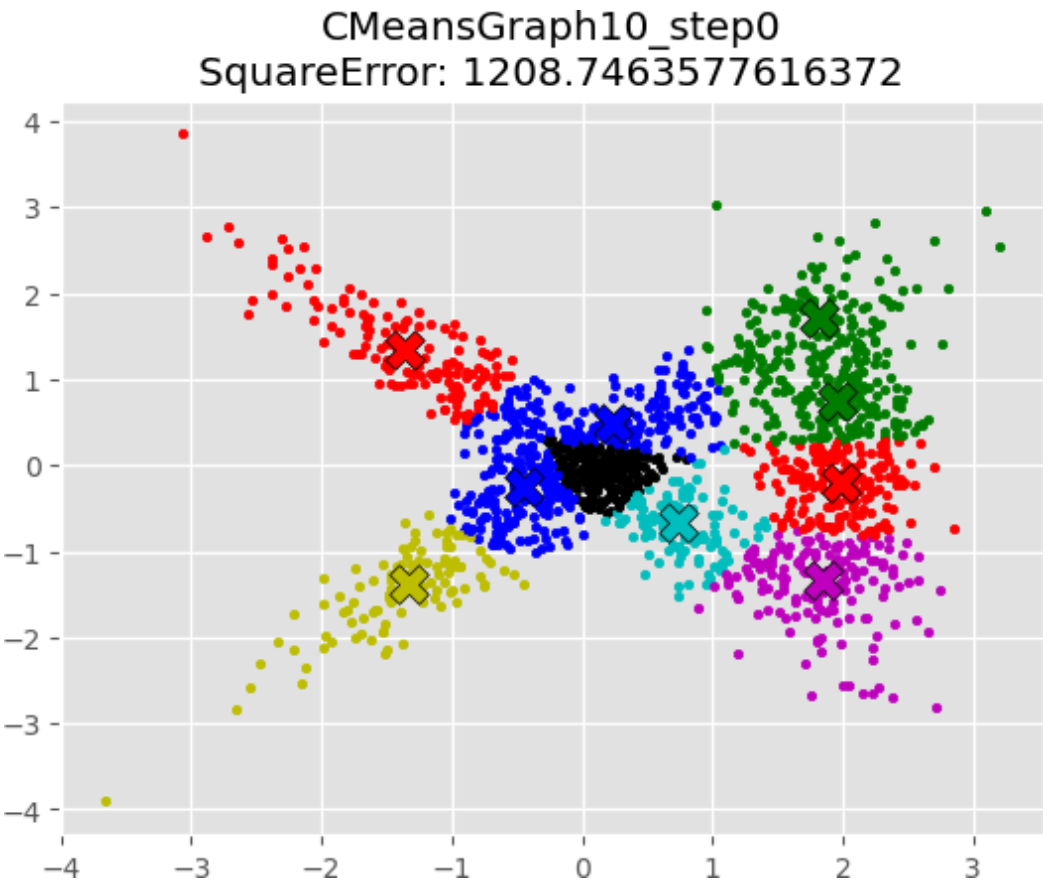
CMeansGraph7_step0
SquareError: 1216.7964124715488

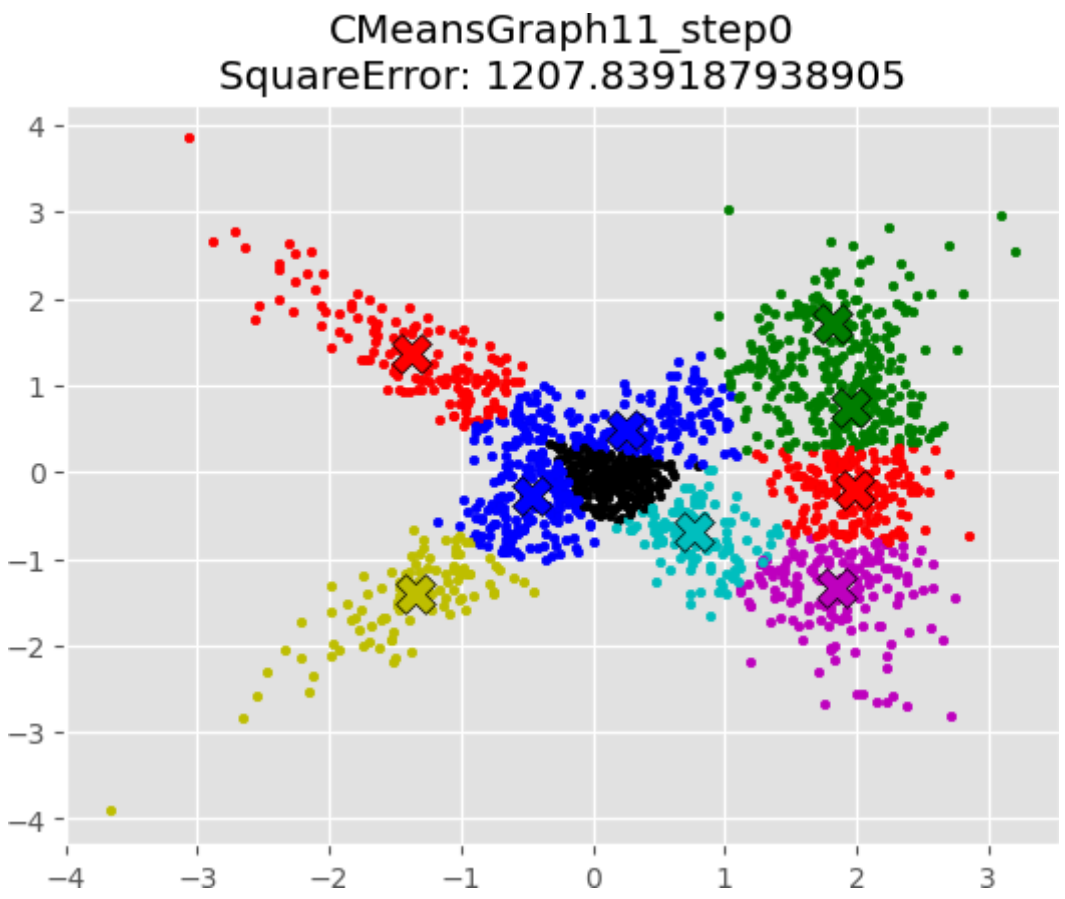


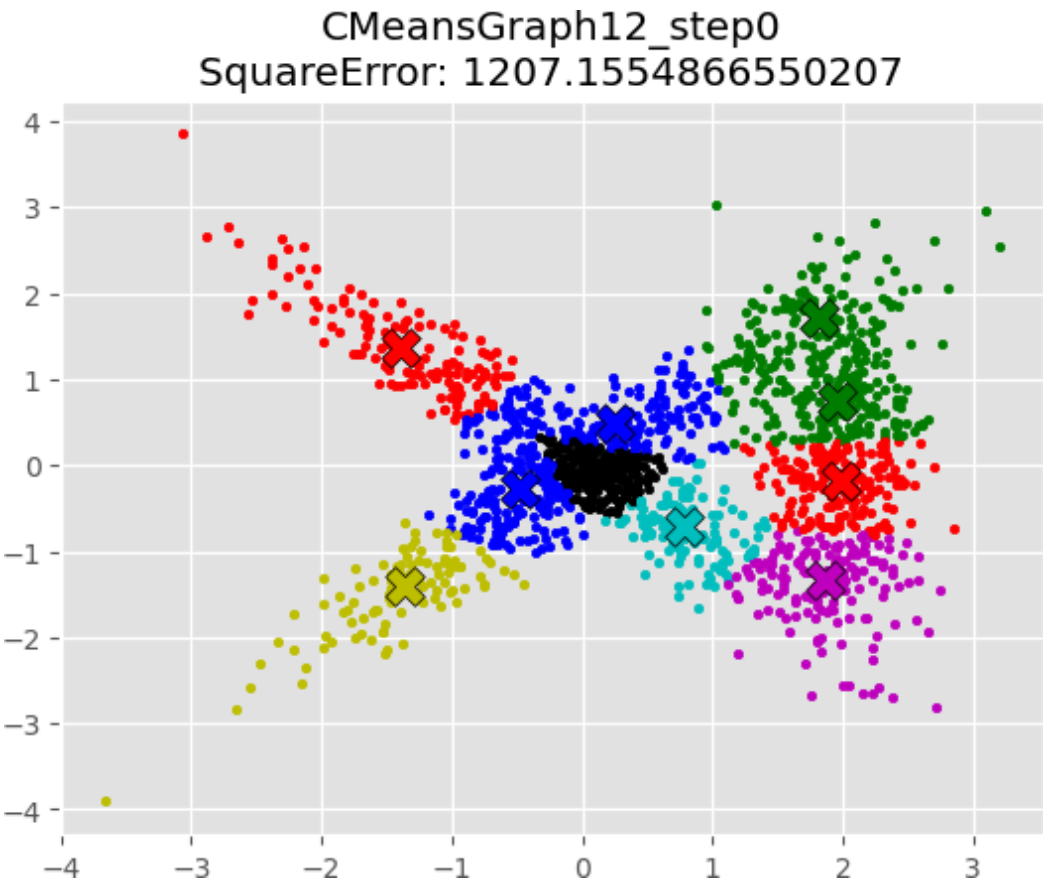
CMeansGraph8_step0
SquareError: 1212.5202214177473

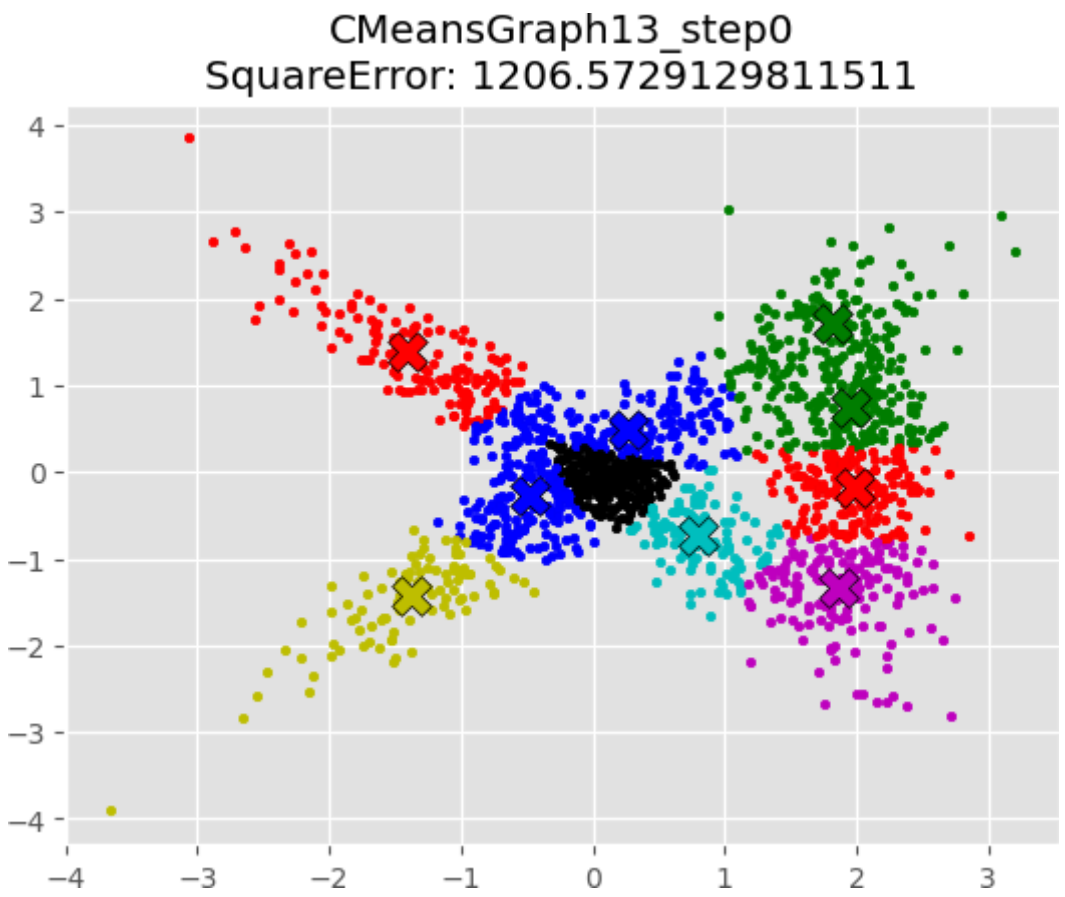


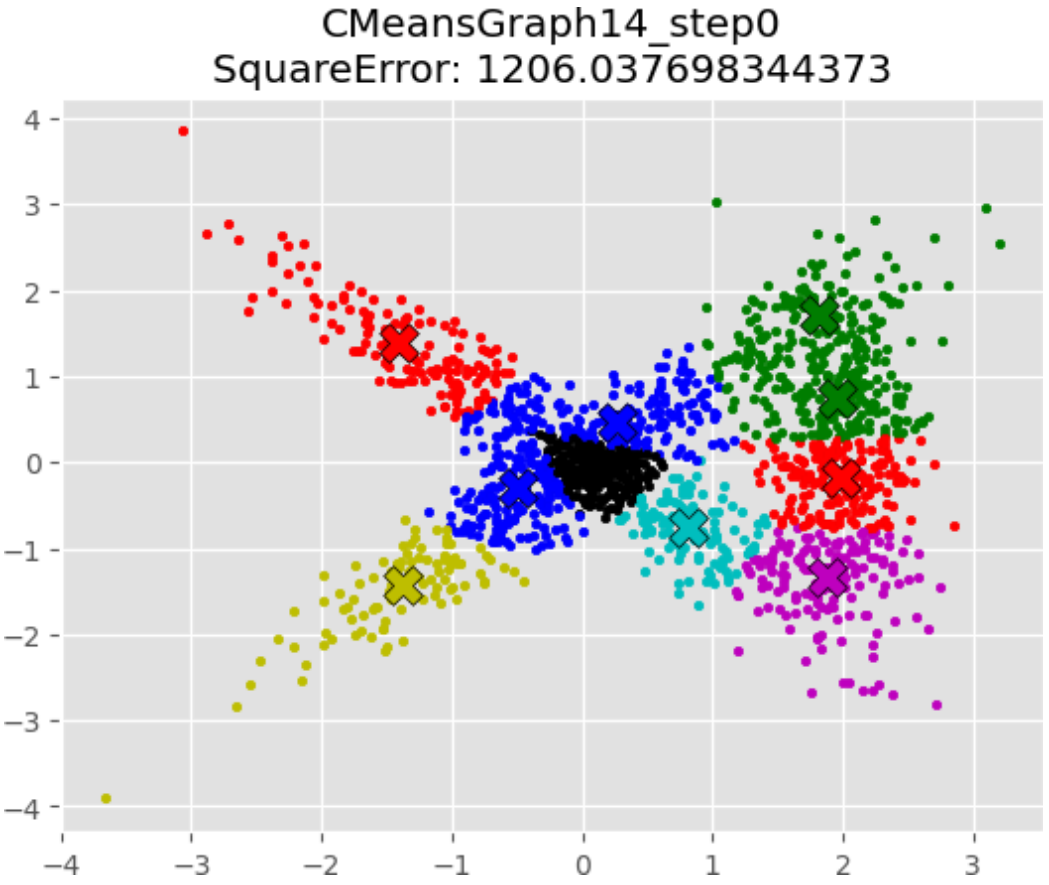

CMeansGraph9_step0
SquareError: 1210.1322353593573
4
3
2
1
0
−1
−2
−3
−4
−4
−3
−2
−1
0
1
2
3

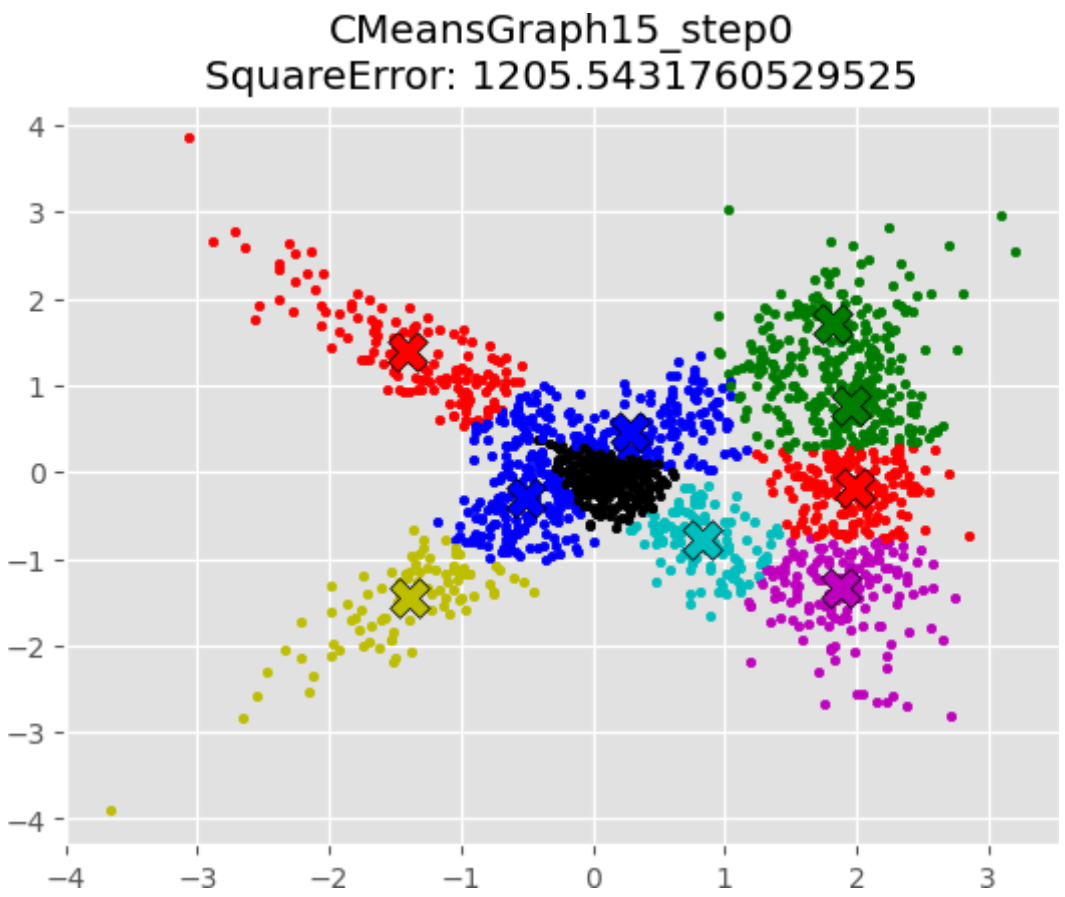

CMeansGraph10_step0
SquareError: 1208.7463577616372
4
3
2
1
0
−1
−2
−3
−4
−4
−3
−2
−1
0
1
2
3

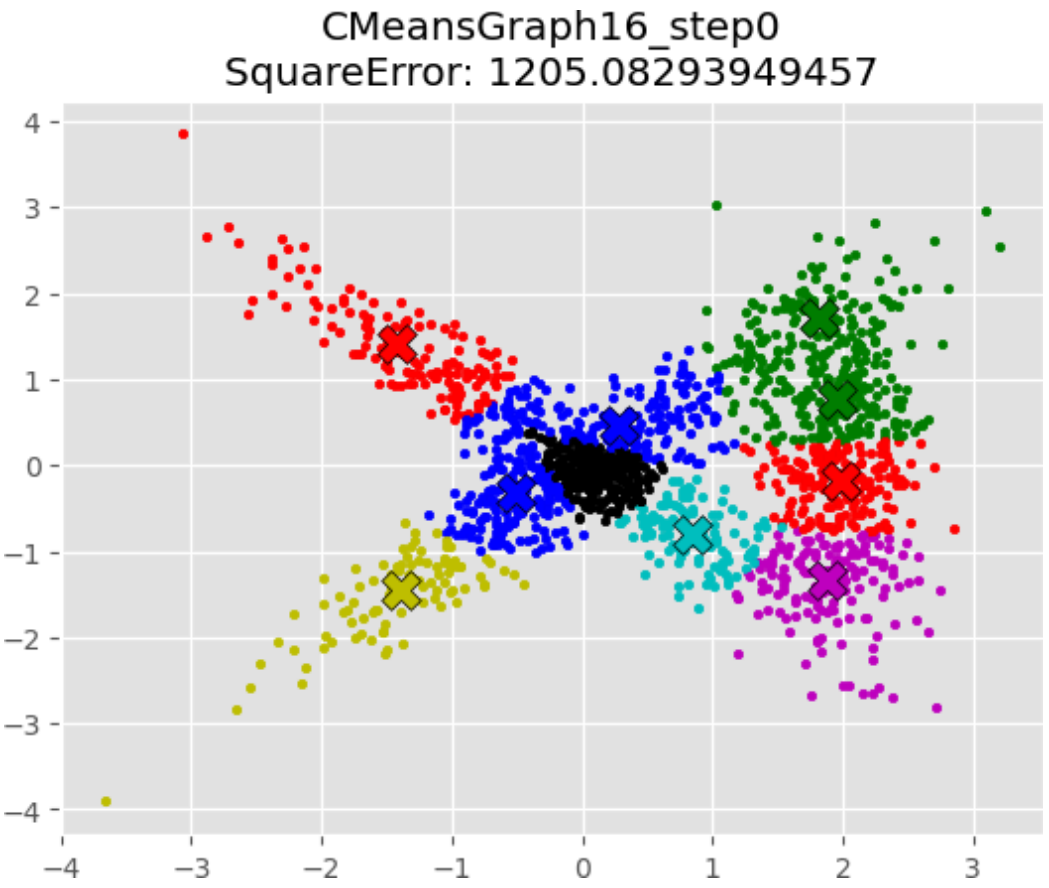

CMeansGraph11_step0
SquareError: 1207.839187938905
4
3
2
1
0
−1
−2
−3
−4
−4
−3
−2
−1
0
1
2
3

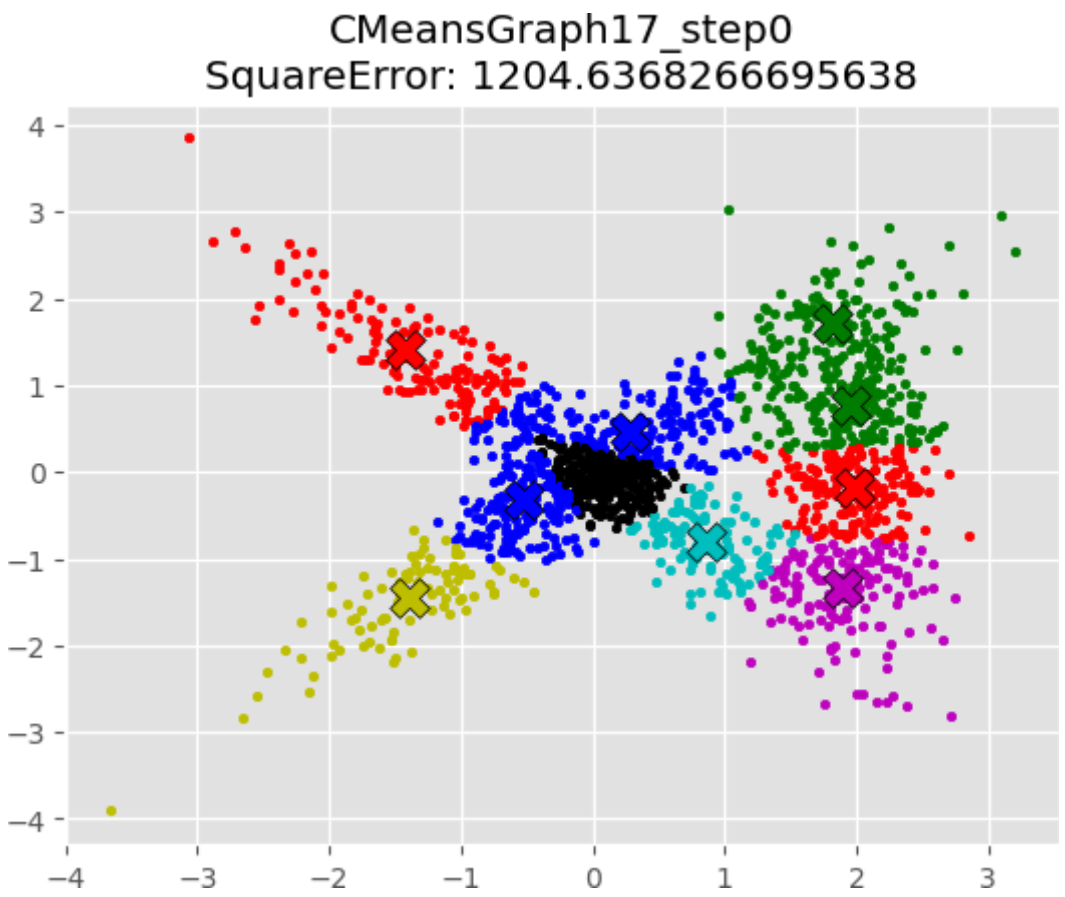

CMeansGraph12_step0
SquareError: 1207.1554866550207
4
3
2
1
0
−1
−2
−3
−4
−4
−3
−2
−1
0
1
2
3

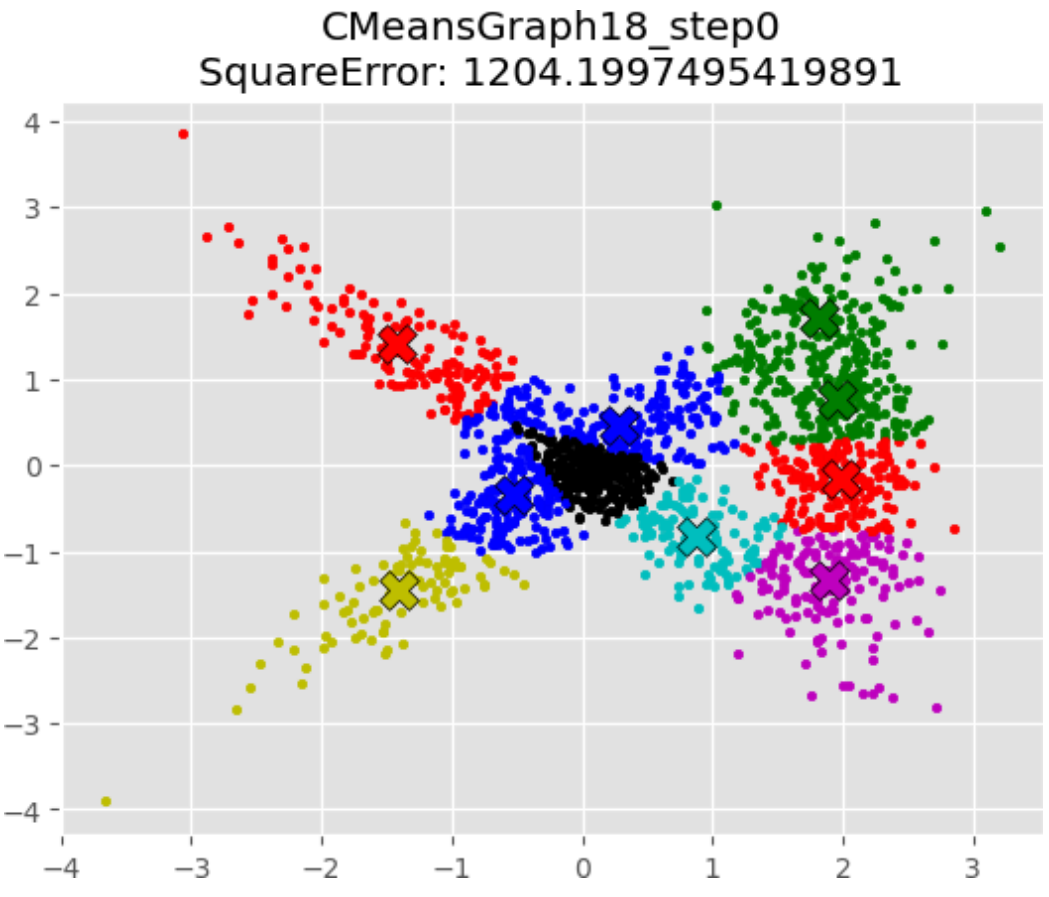

CMeansGraph13_step0
SquareError: 1206.5729129811511


CMeansGraph14_step0
SquareError: 1206.037698344373

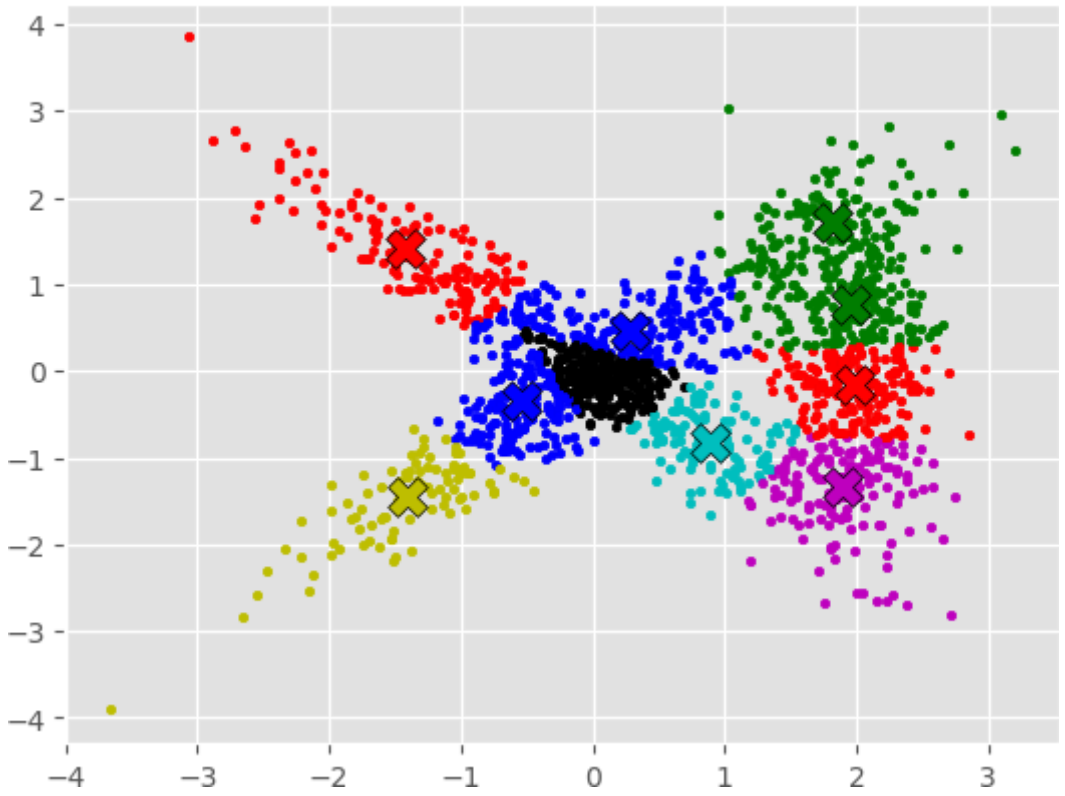

CMeansGraph15_step0
SquareError: 1205.5431760529525
4
3
2
1
0
−1
−2
−3
−4
−4
−3
−2
−1
0
1
2
3


CMeansGraph16_step0
SquareError: 1205.08293949457
4
3
2
1
0
−1
−2
−3
−4
−4
−3
−2
−1
0
1
2
3

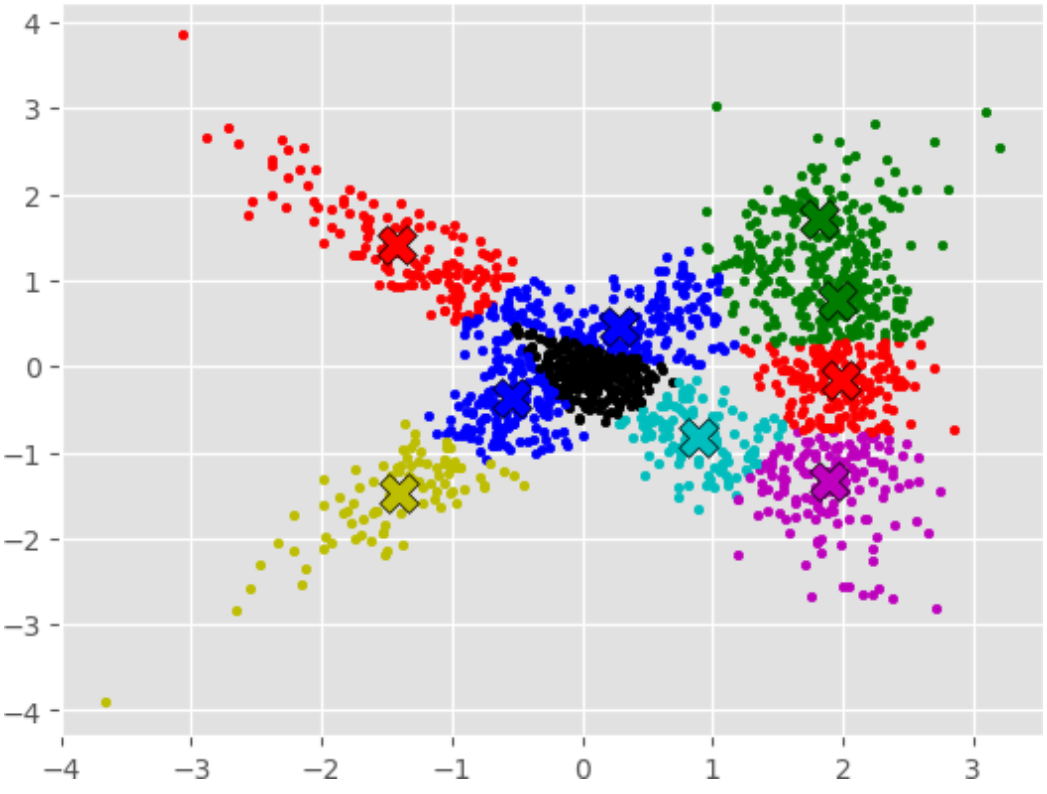

CMeansGraph17_step0
SquareError: 1204.6368266695638


CMeansGraph18_step0
SquareError: 1204.1997495419891

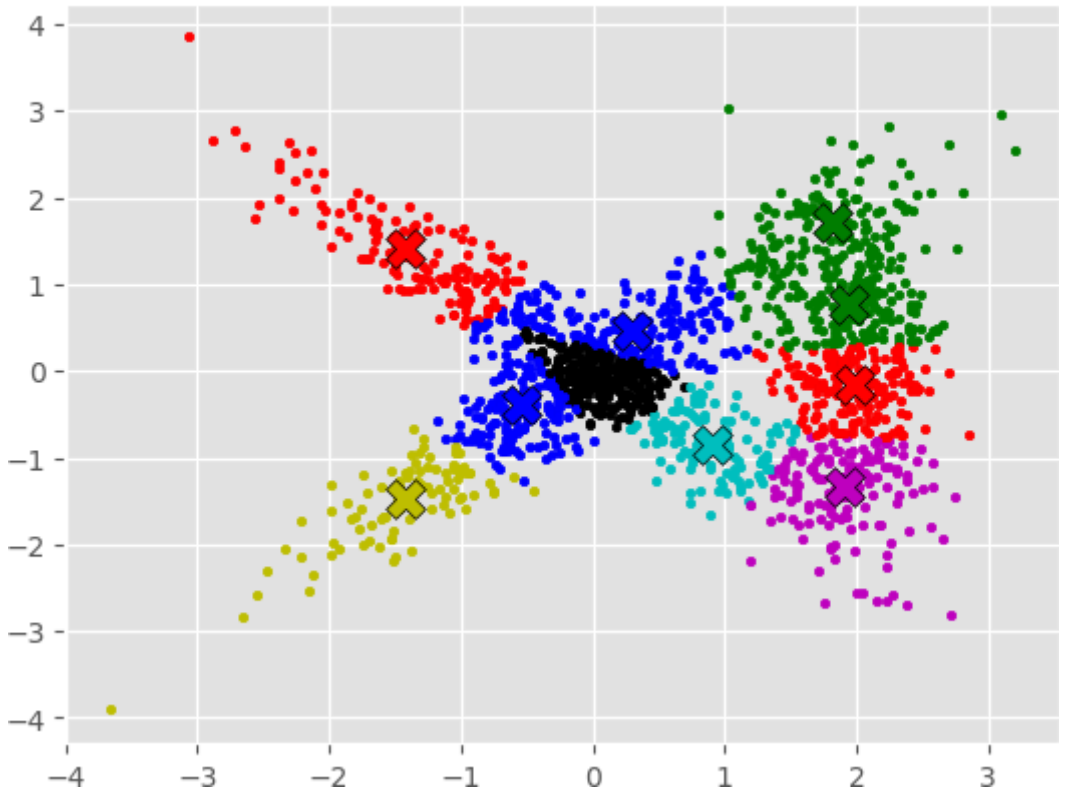
CMeansGraph19_step0
SquareError: 1203.7704618509392



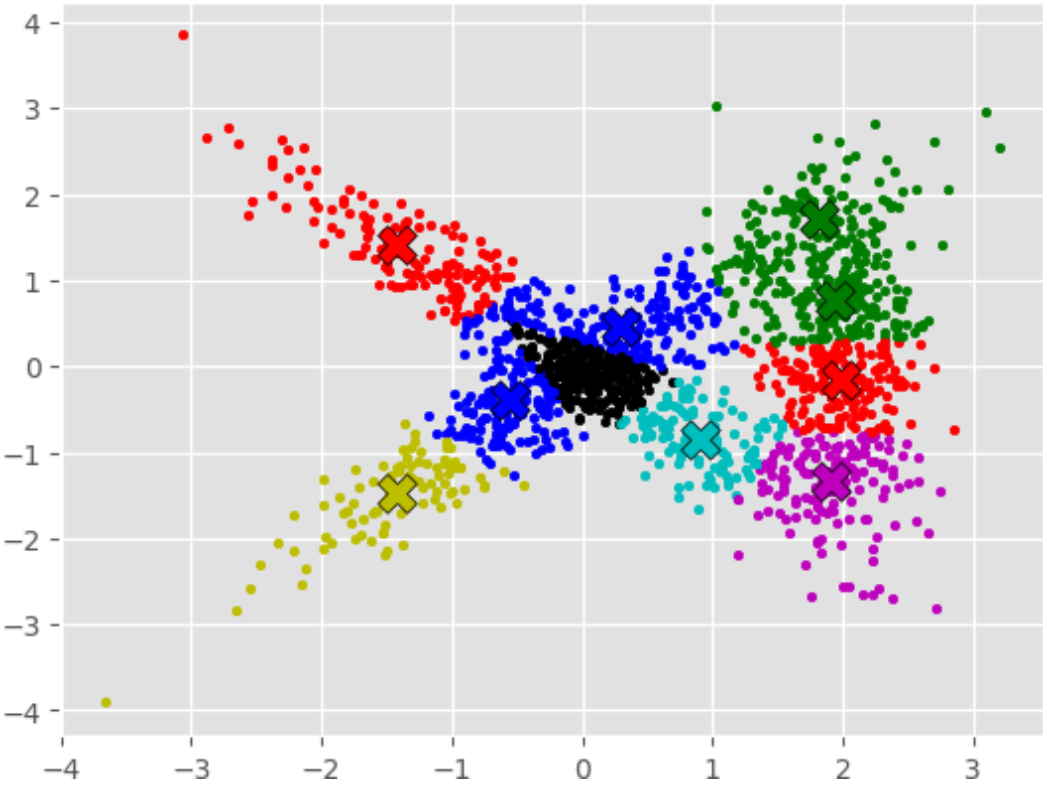
CMeansGraph20_step0
SquareError: 1203.3489731679908

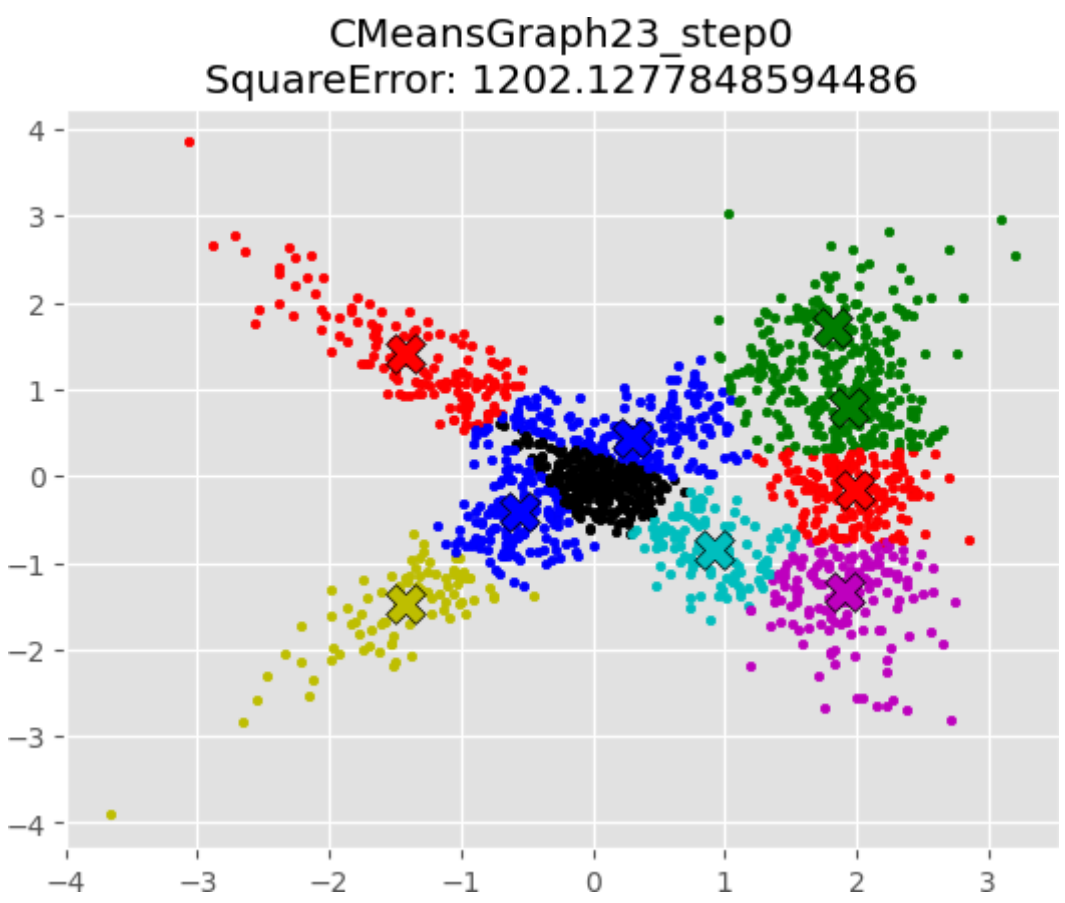


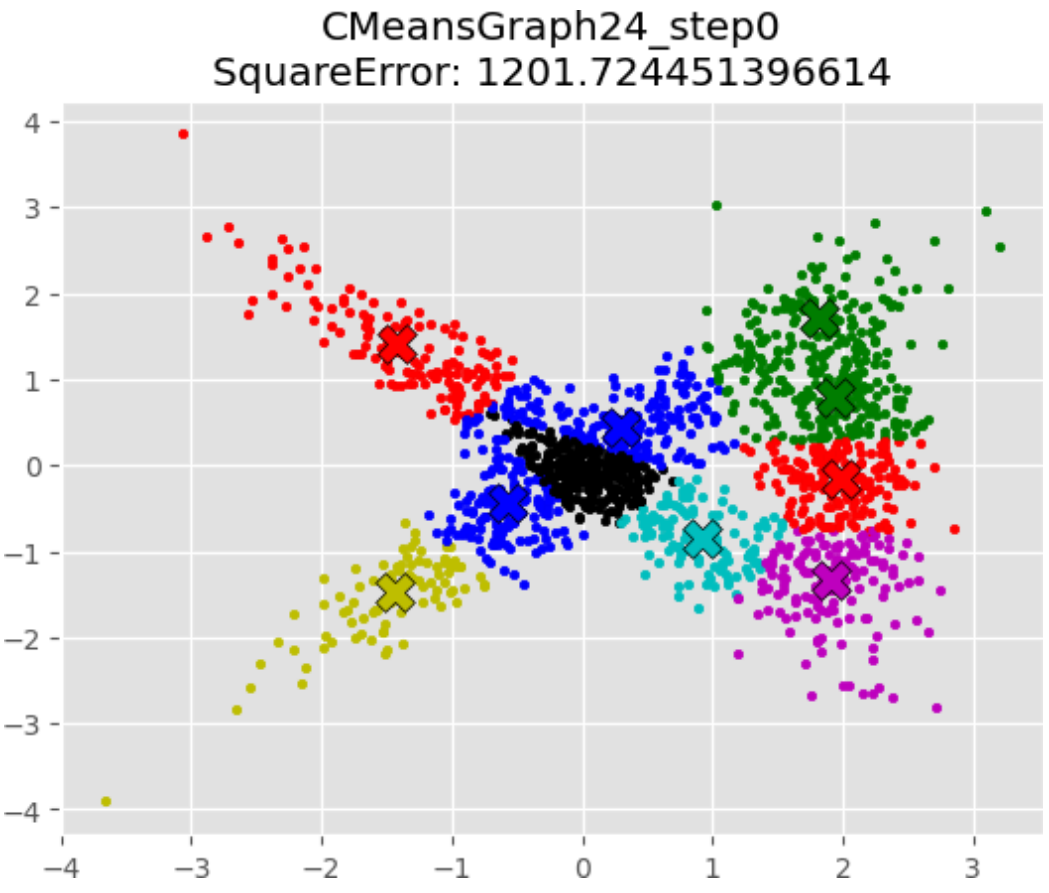
CMeansGraph21_step0
SquareError: 1202.9358547998354

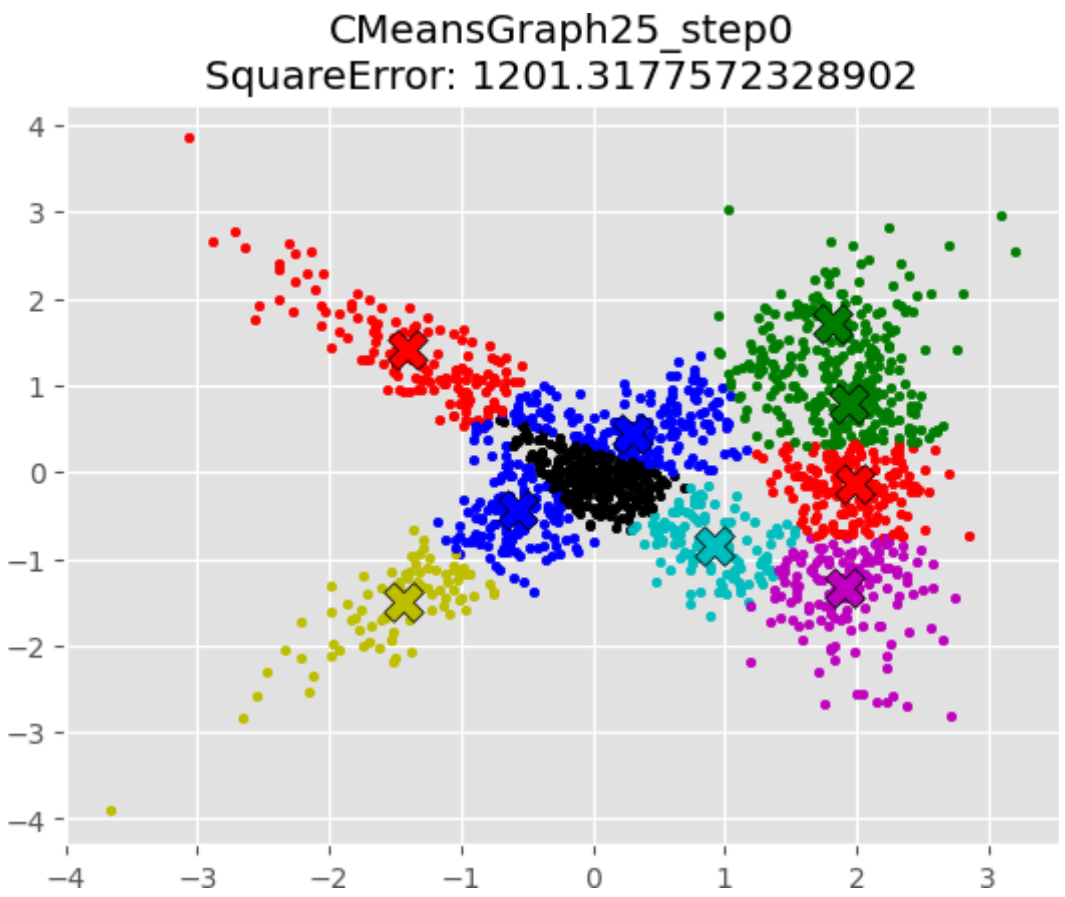


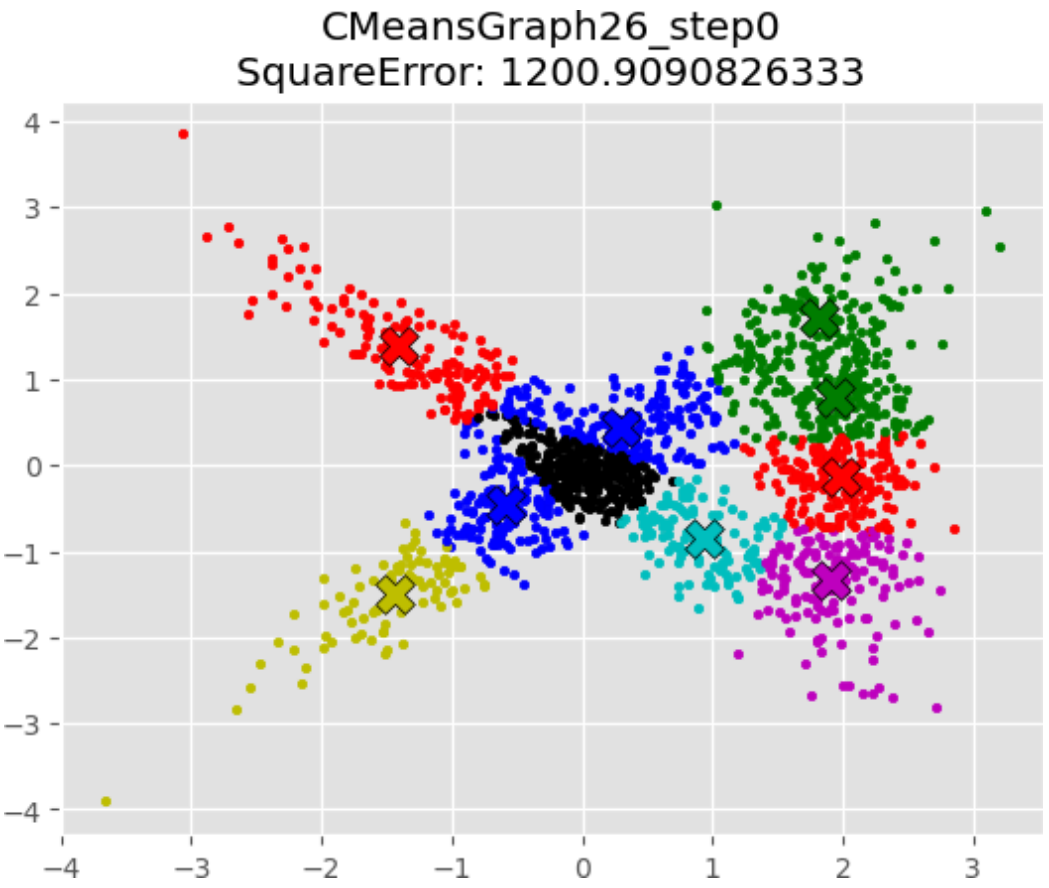
CMeansGraph22_step0
SquareError: 1202.5301797580933

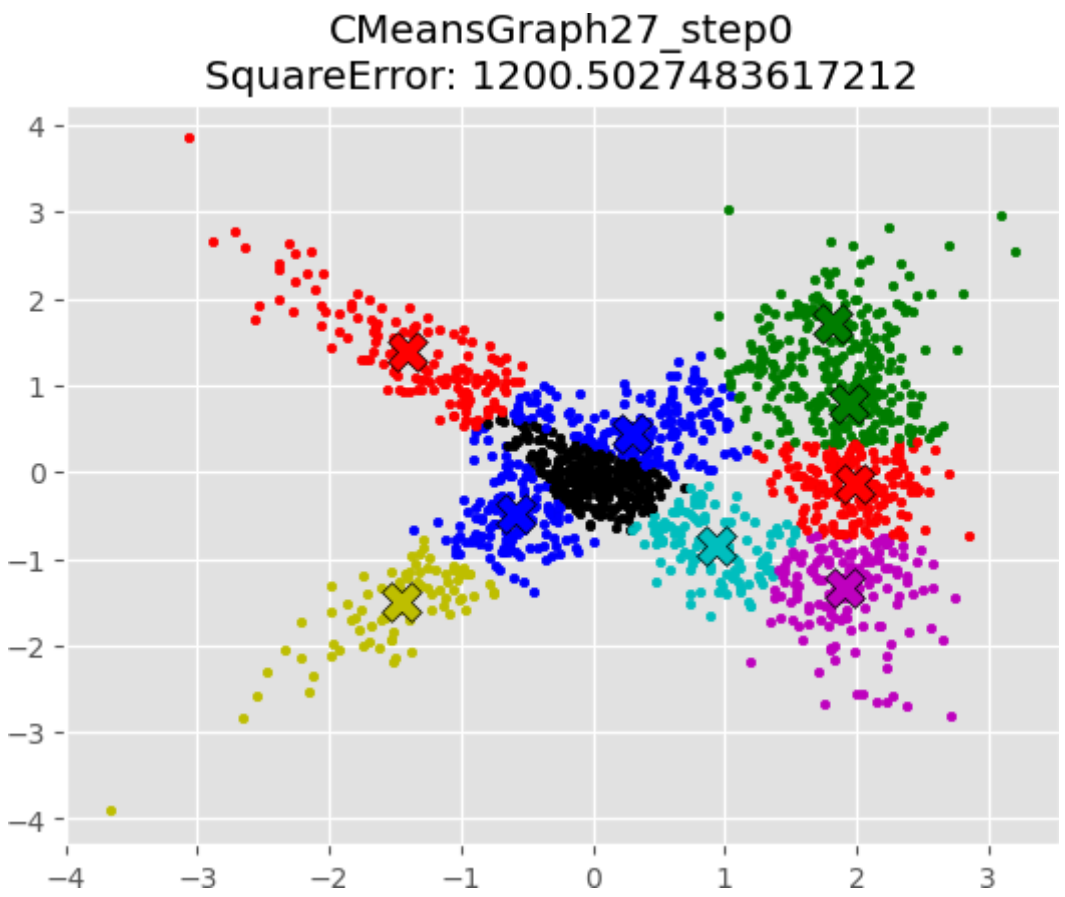


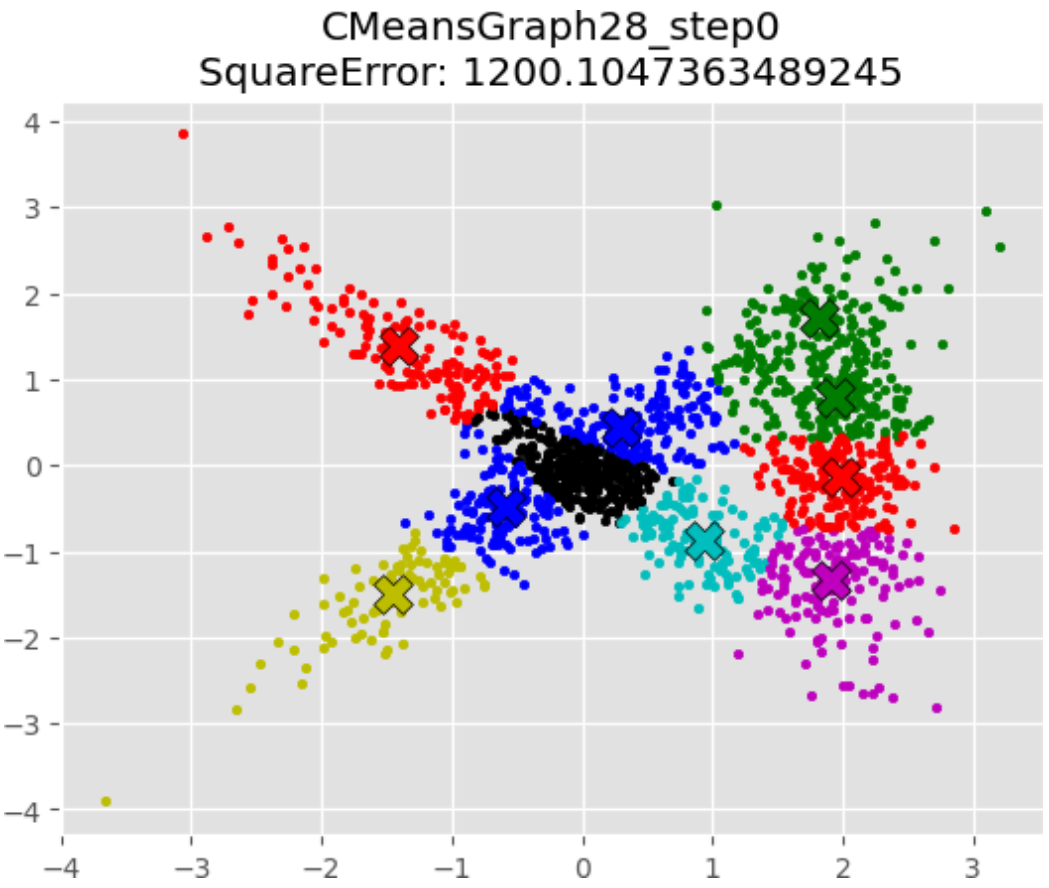

CMeansGraph23_step0
SquareError: 1202.1277848594486


CMeansGraph24_step0
SquareError: 1201.724451396614

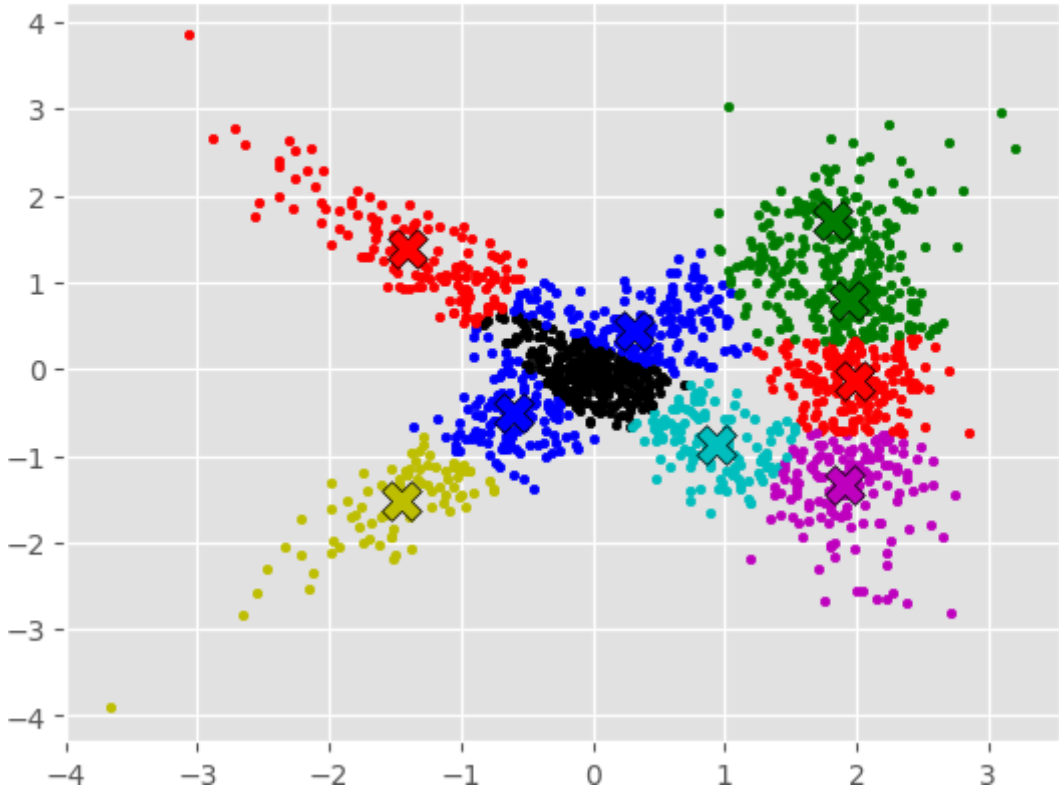

CMeansGraph25_step0
SquareError: 1201.3177572328902
4
3
2
1
0
−1
−2
−3
−4
−4
−3
−2
−1
0
1
2
3


CMeansGraph26_step0
SquareError: 1200.9090826333
4
3
2
1
0
−1
−2
−3
−4
−4
−3
−2
−1
0
1
2
3

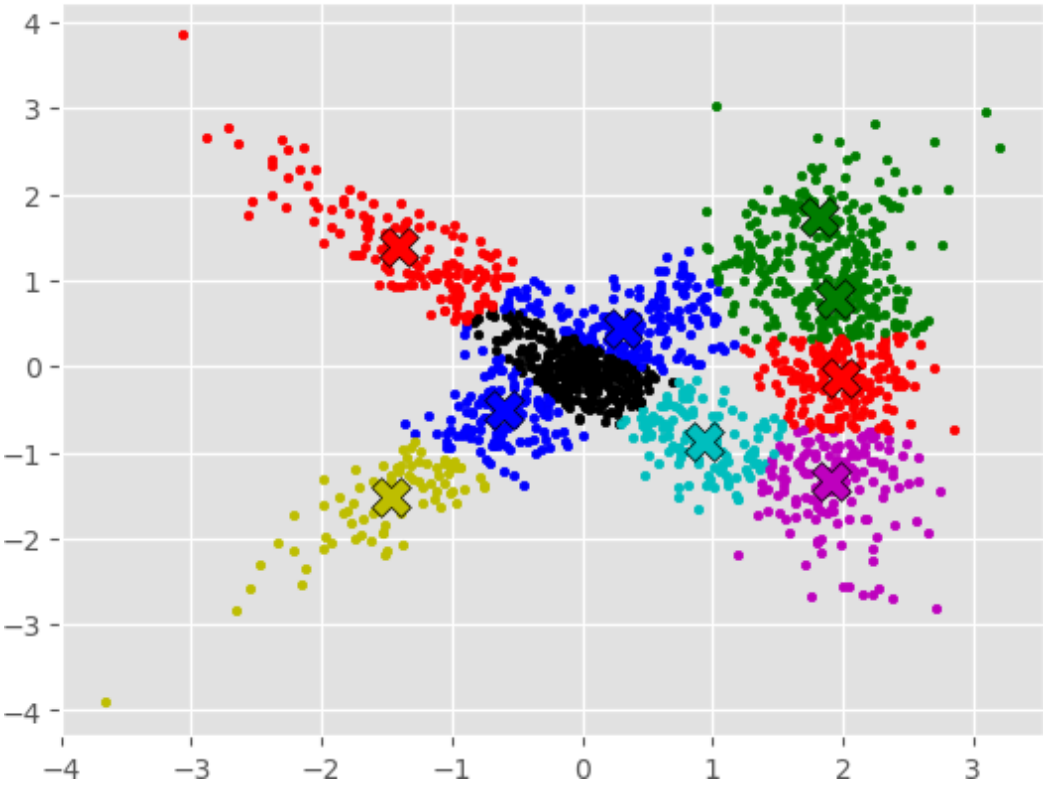

CMeansGraph27_step0
SquareError: 1200.5027483617212
4
3
2
1
0
−1
−2
−3
−4
−4
−3
−2
−1
0
1
2
3


CMeansGraph28_step0
SquareError: 1200.1047363489245
4
3
2
1
0
−1
−2
−3
−4
−4
−3
−2
−1
0
1
2
3

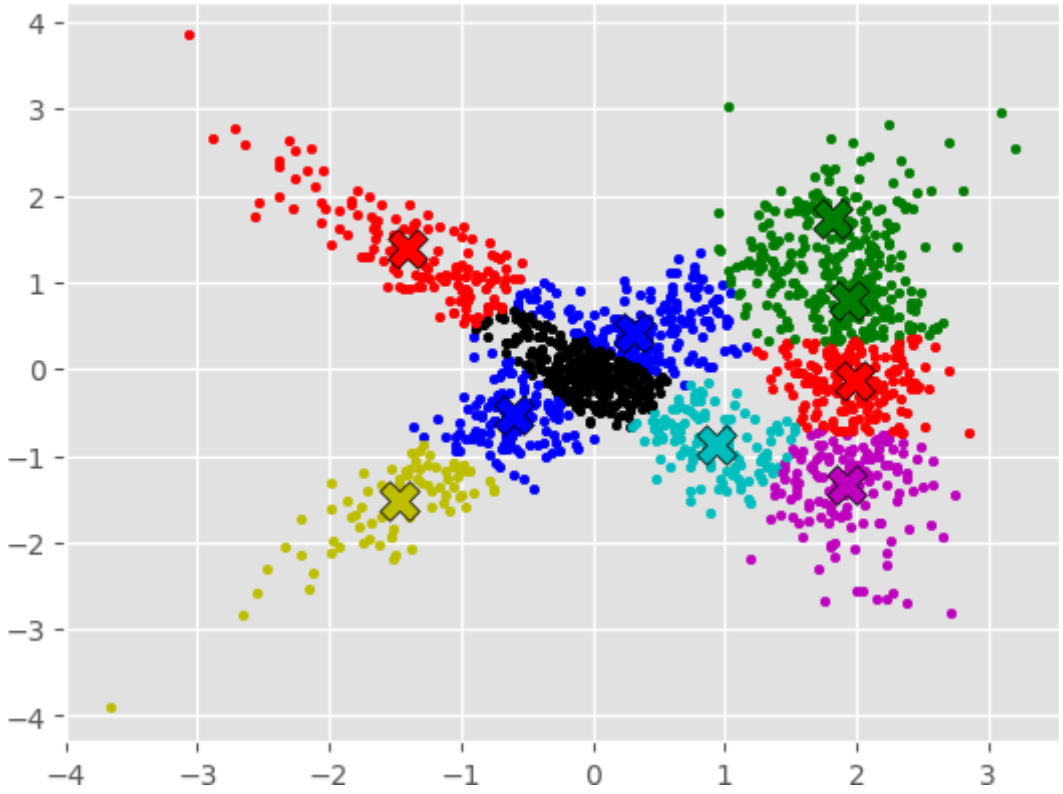
CMeansGraph29_step0
SquareError: 1199.7209767331835



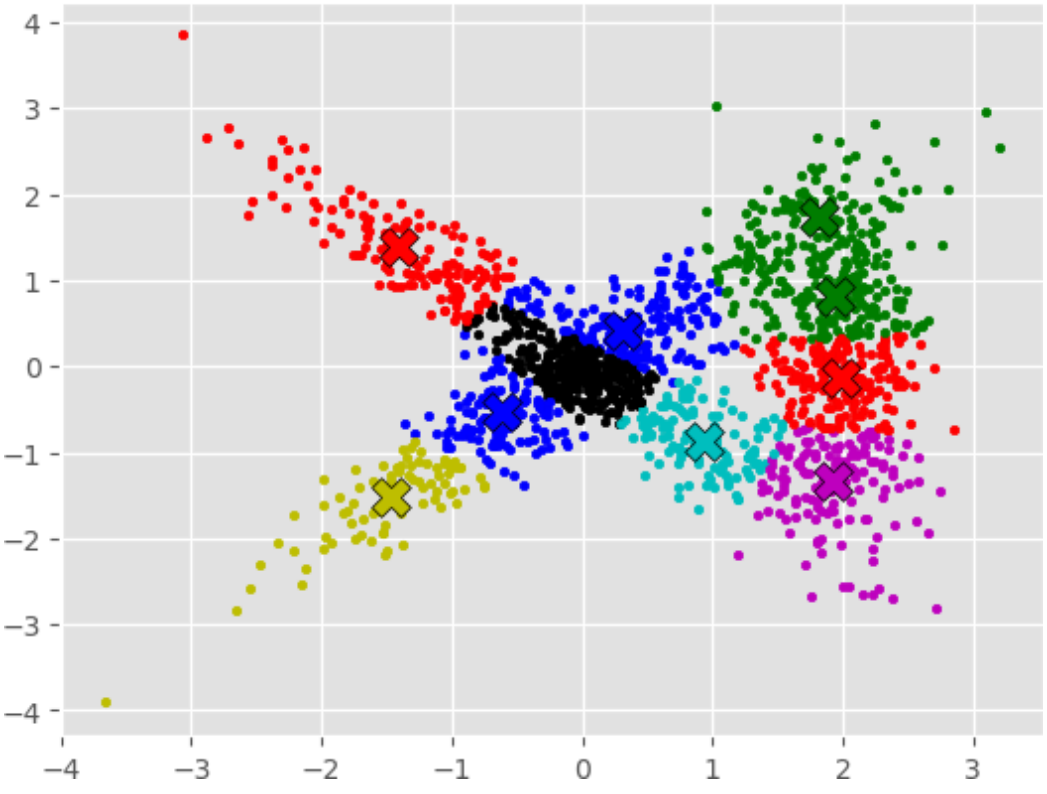
CMeansGraph30_step0
SquareError: 1199.3561843895536

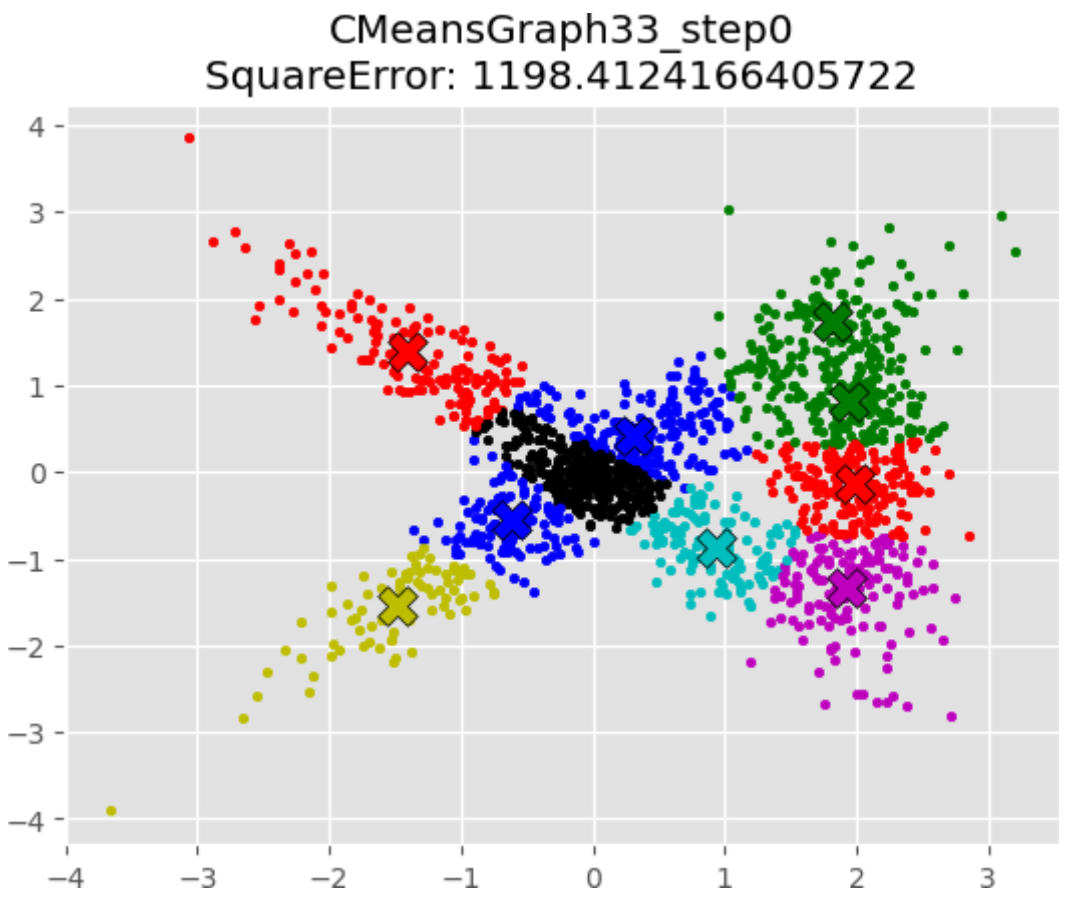


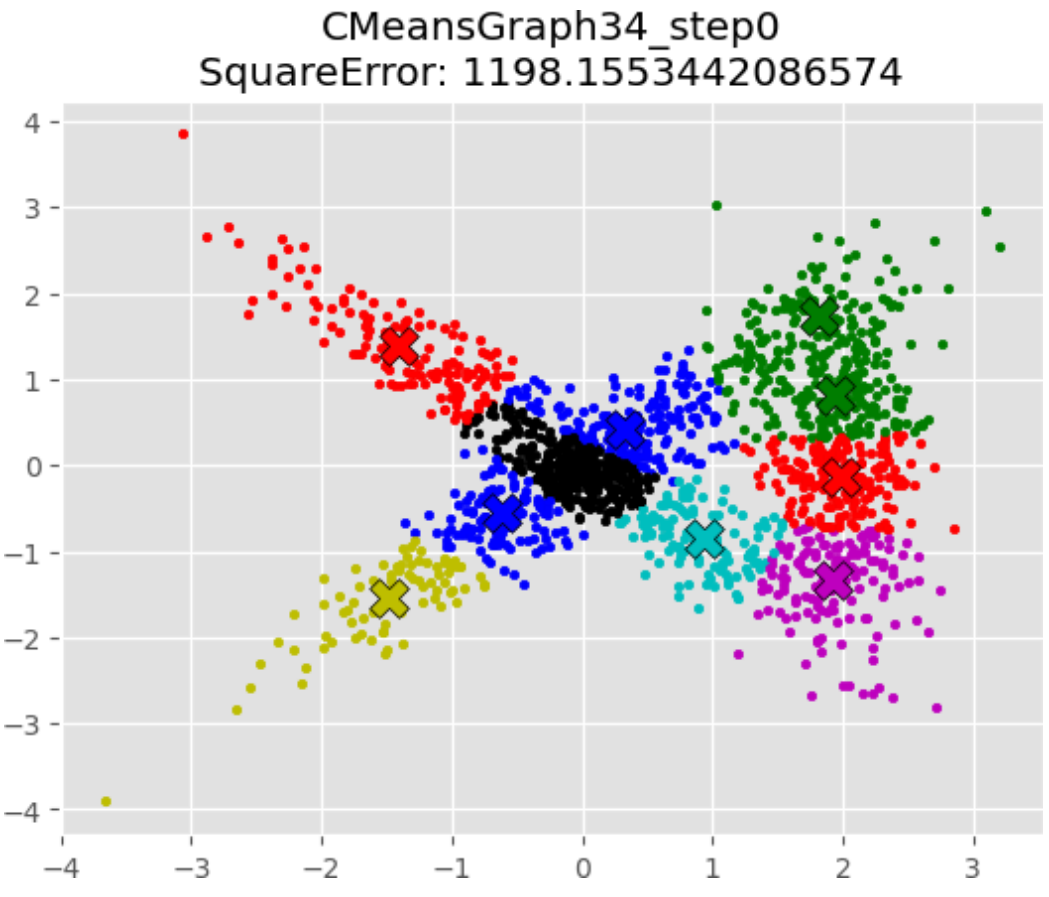
CMeansGraph31_step0
SquareError: 1199.0145038678925



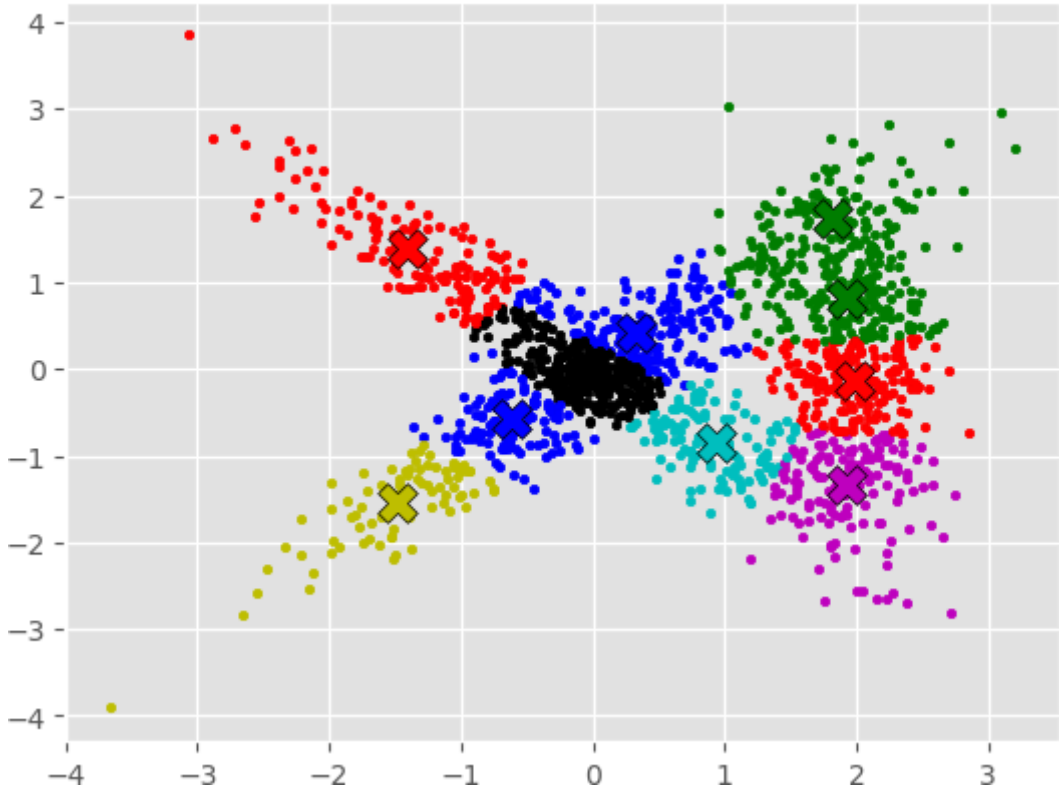
CMeansGraph32_step0
SquareError: 1198.6992395545149



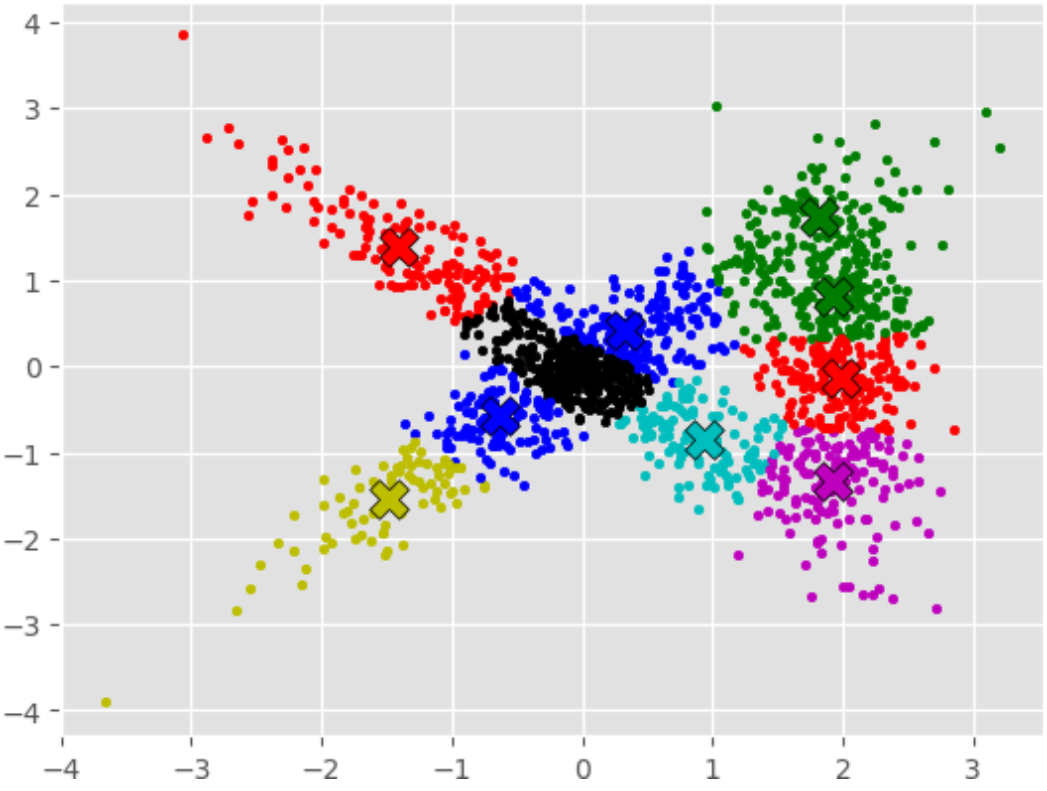

CMeansGraph33_step0
SquareError: 1198.4124166405722

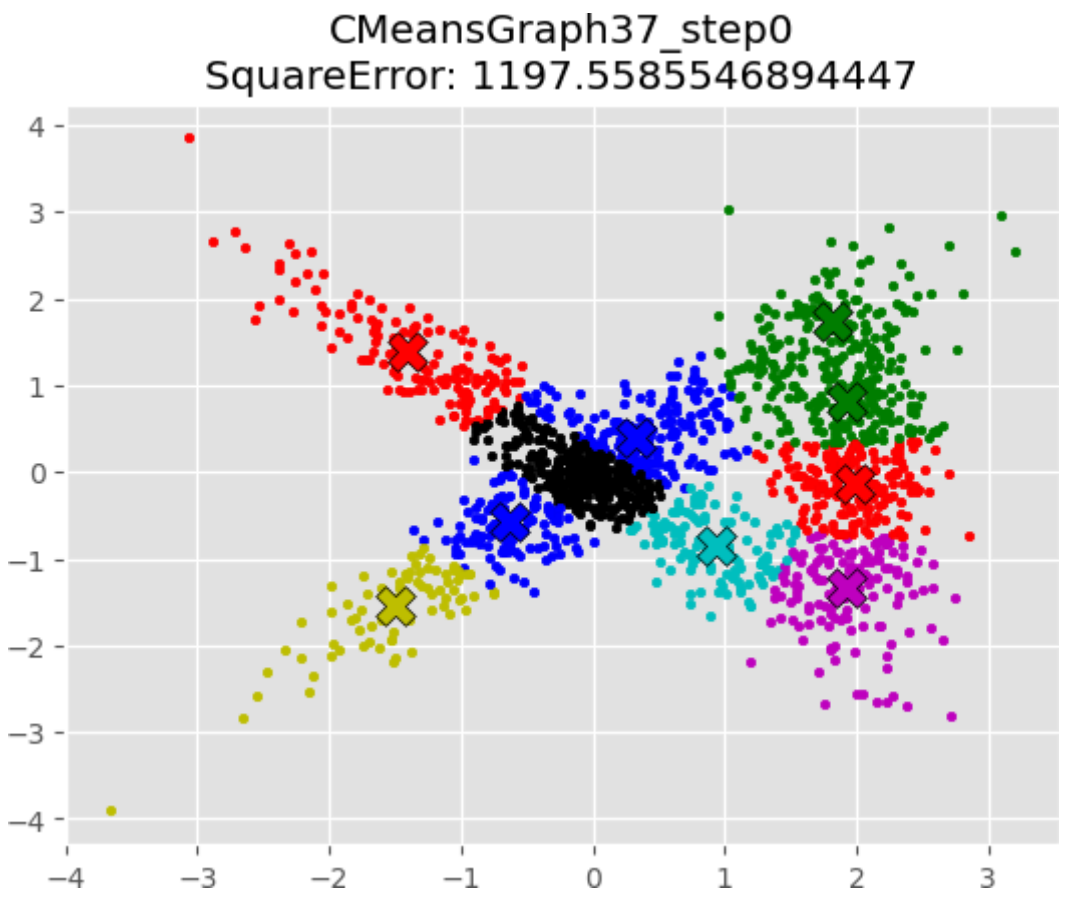

CMeansGraph34_step0
SquareError: 1198.1553442086574

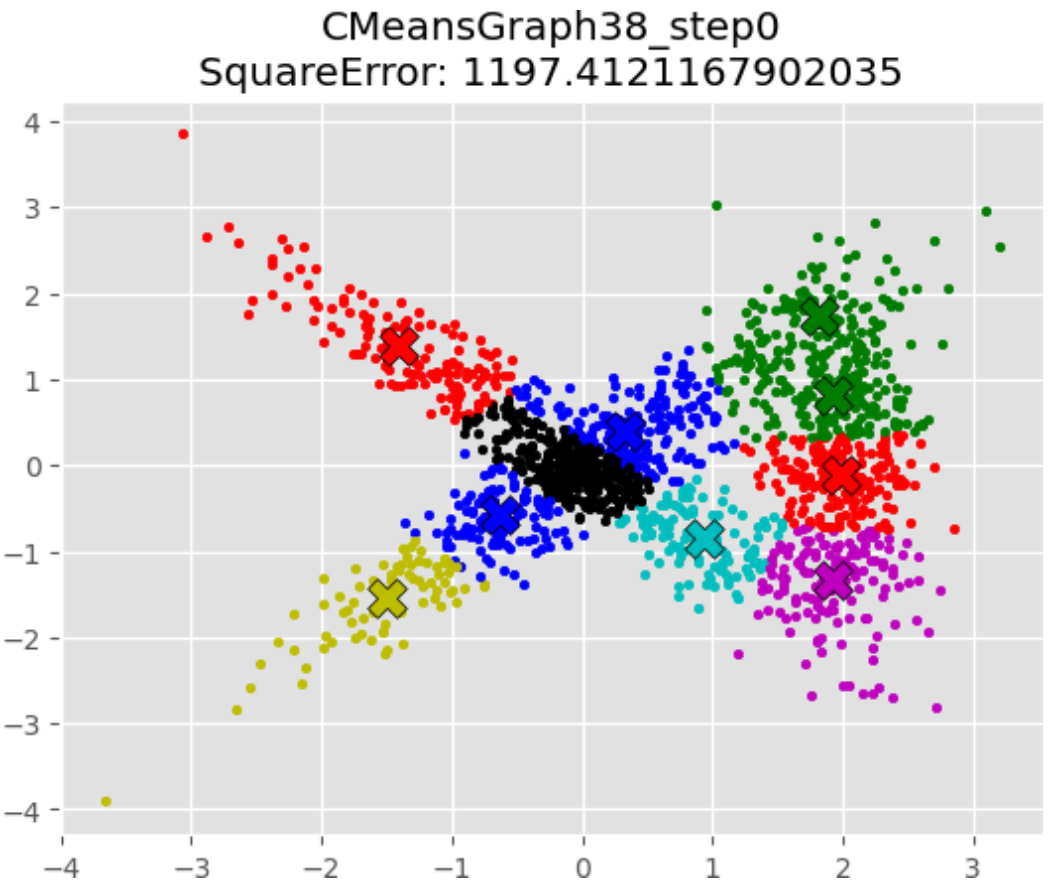
CMeansGraph35_step0
SquareError: 1197.9280364068366

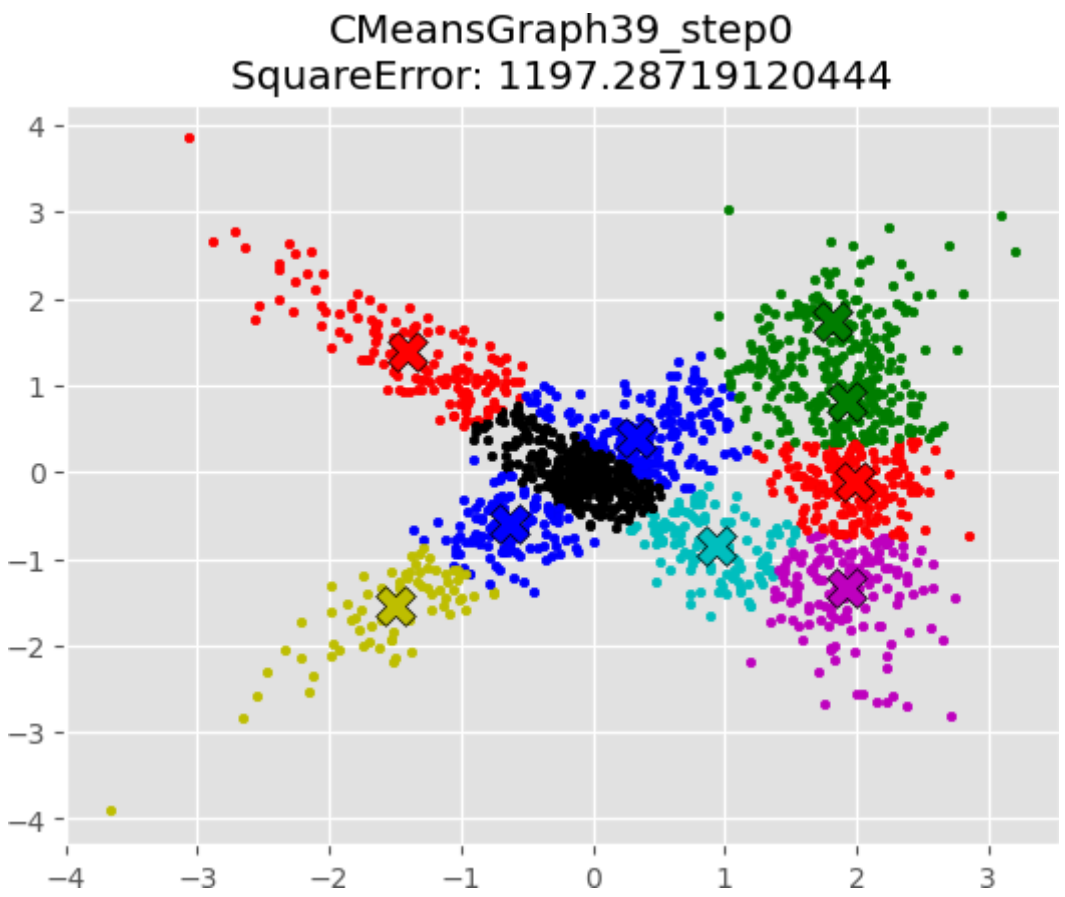


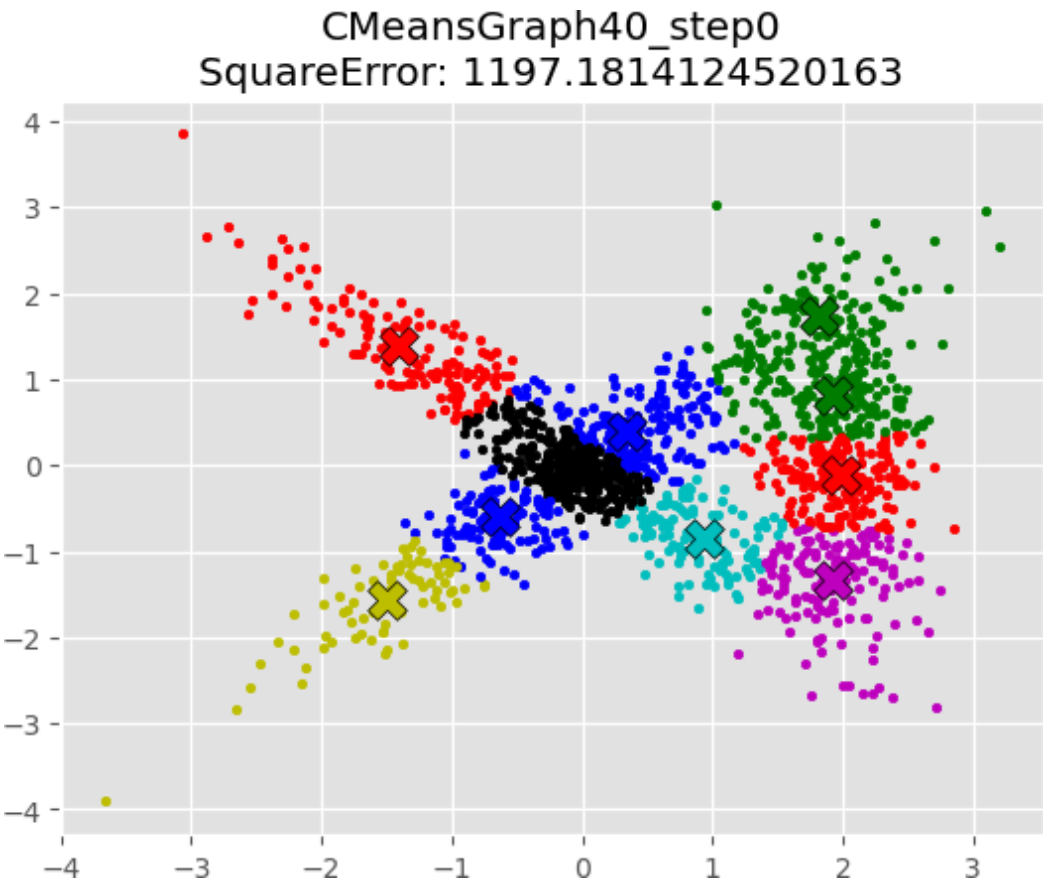
CMeansGraph36_step0
SquareError: 1197.7295718285782



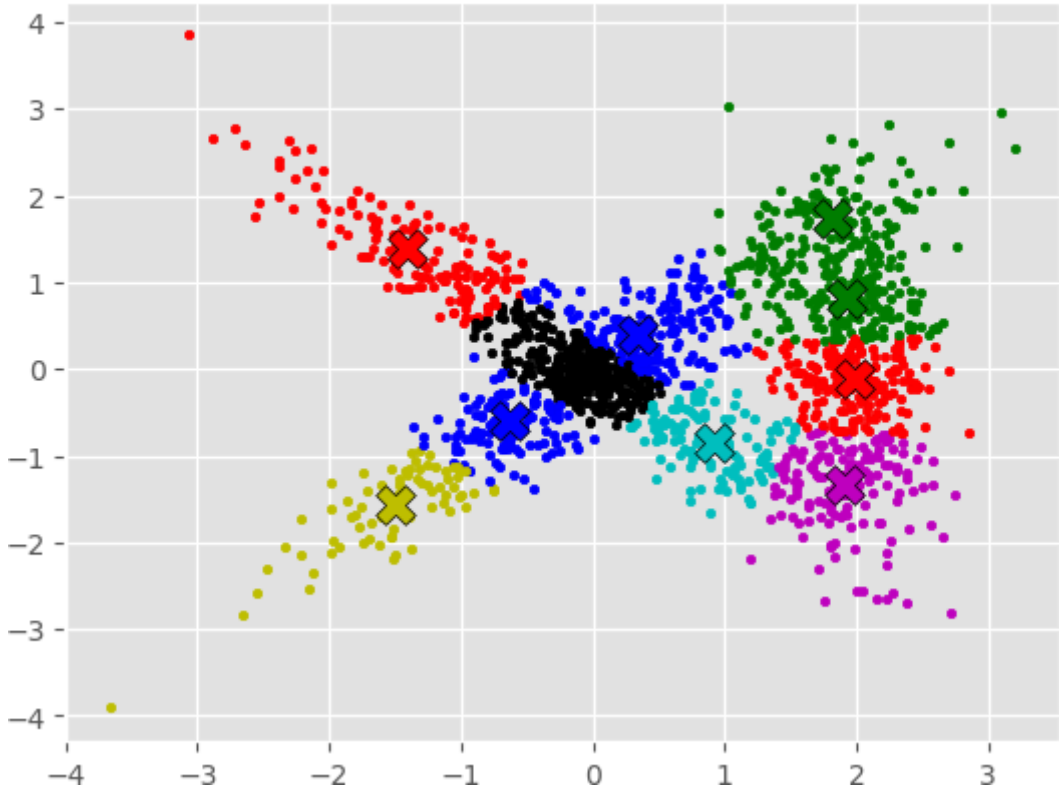

CMeansGraph37_step0
SquareError: 1197.5585546894447
4
3
2
1
0
−1
−2
−3
−4
−4
−3
−2
−1
0
1
2
3


CMeansGraph38_step0
SquareError: 1197.4121167902035
4
3
2
1
0
−1
−2
−3
−4
−4
−3
−2
−1
0
1
2
3

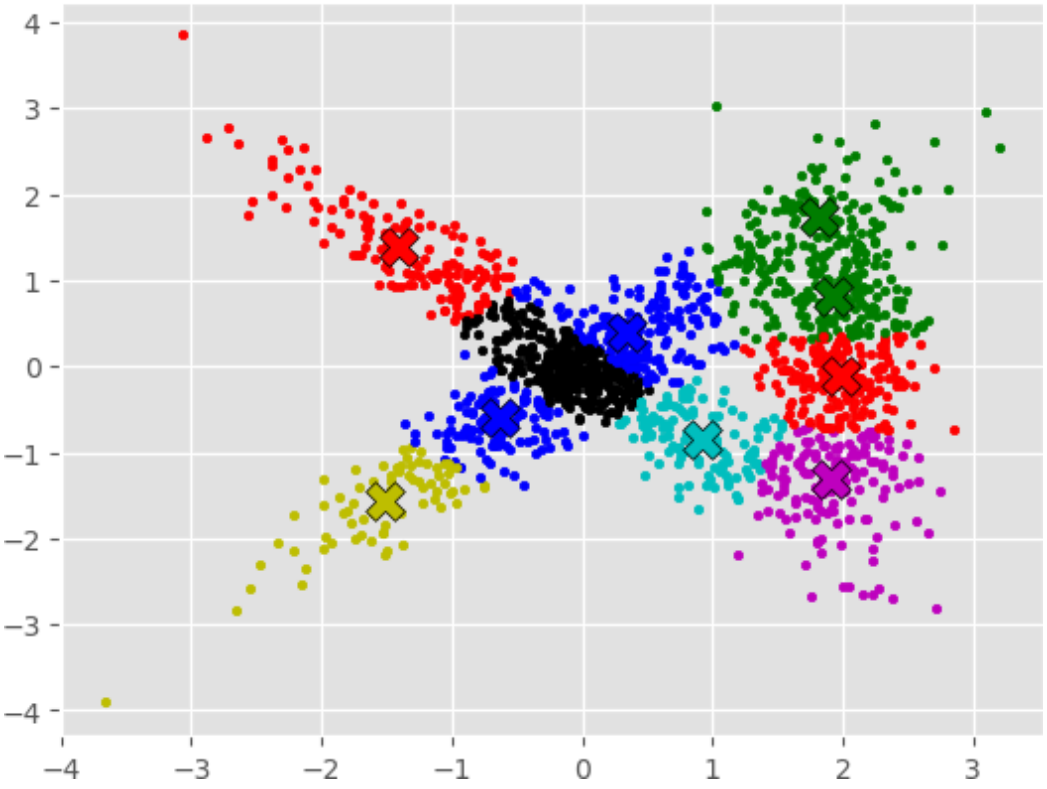

CMeansGraph39_step0
SquareError: 1197.28719120444
4
3
2
1
0
−1
−2
−3
−4
−4
−3
−2
−1
0
1
2
3

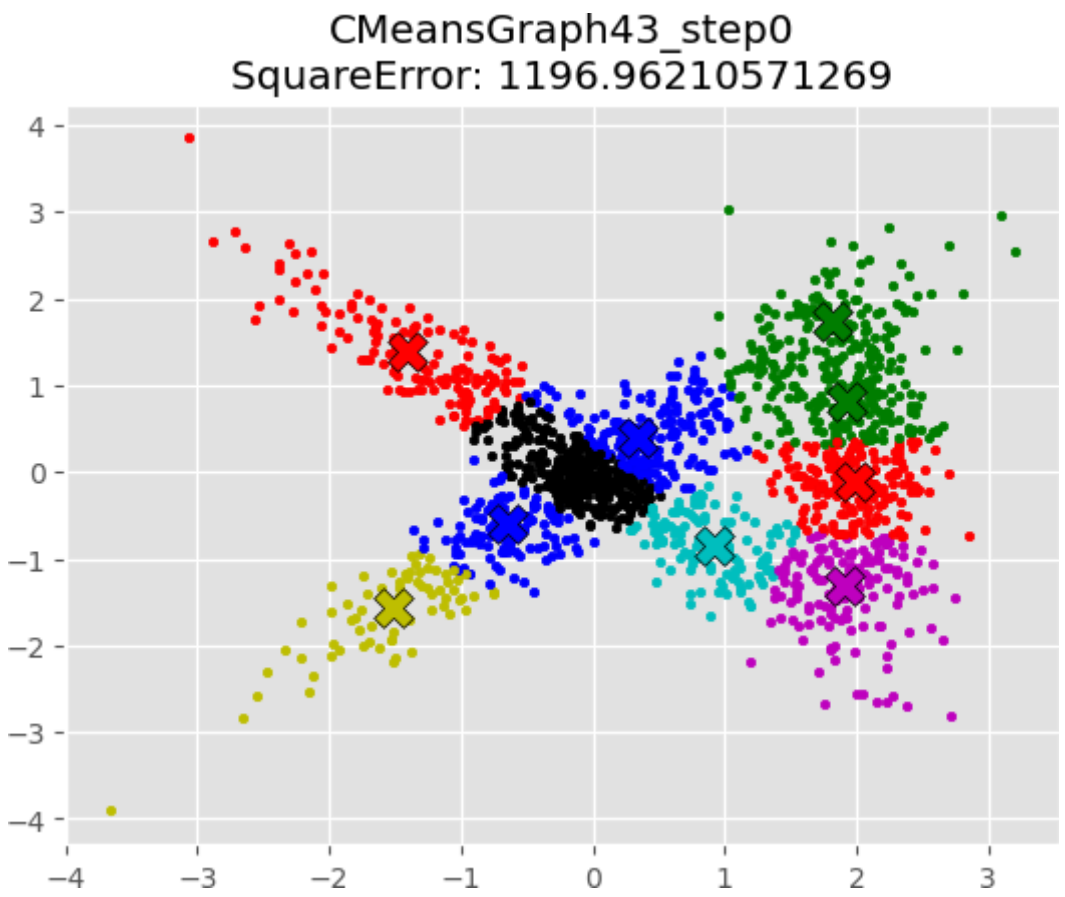

CMeansGraph40_step0
SquareError: 1197.1814124520163
4
3
2
1
0
−1
−2
−3
−4
−4
−3
−2
−1
0
1
2
3

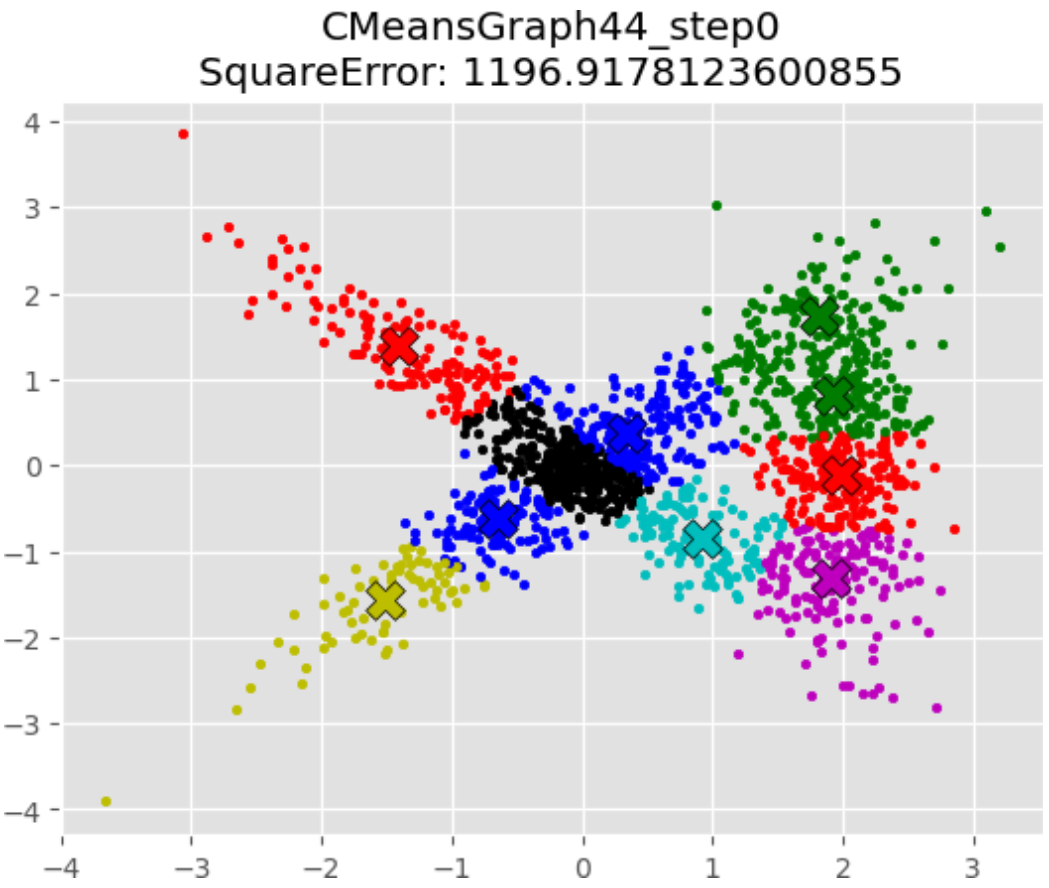
CMeansGraph41_step0
SquareError: 1197.0929114277023



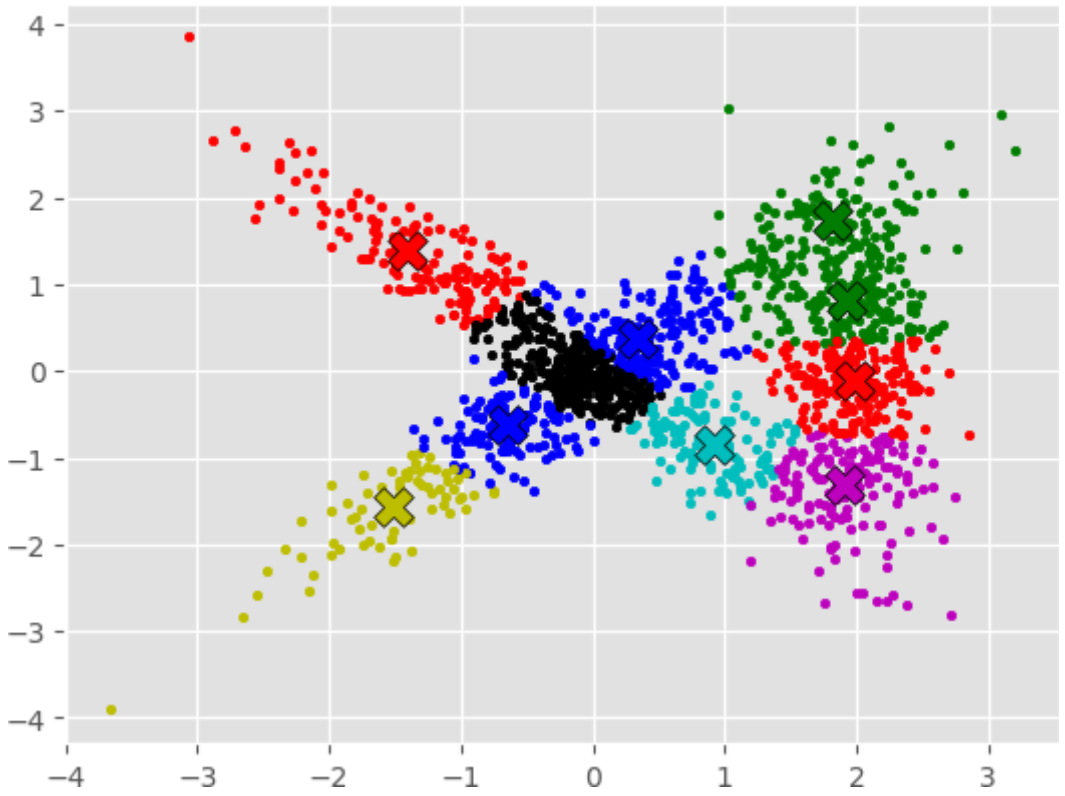
CMeansGraph42_step0
SquareError: 1197.0201920344089



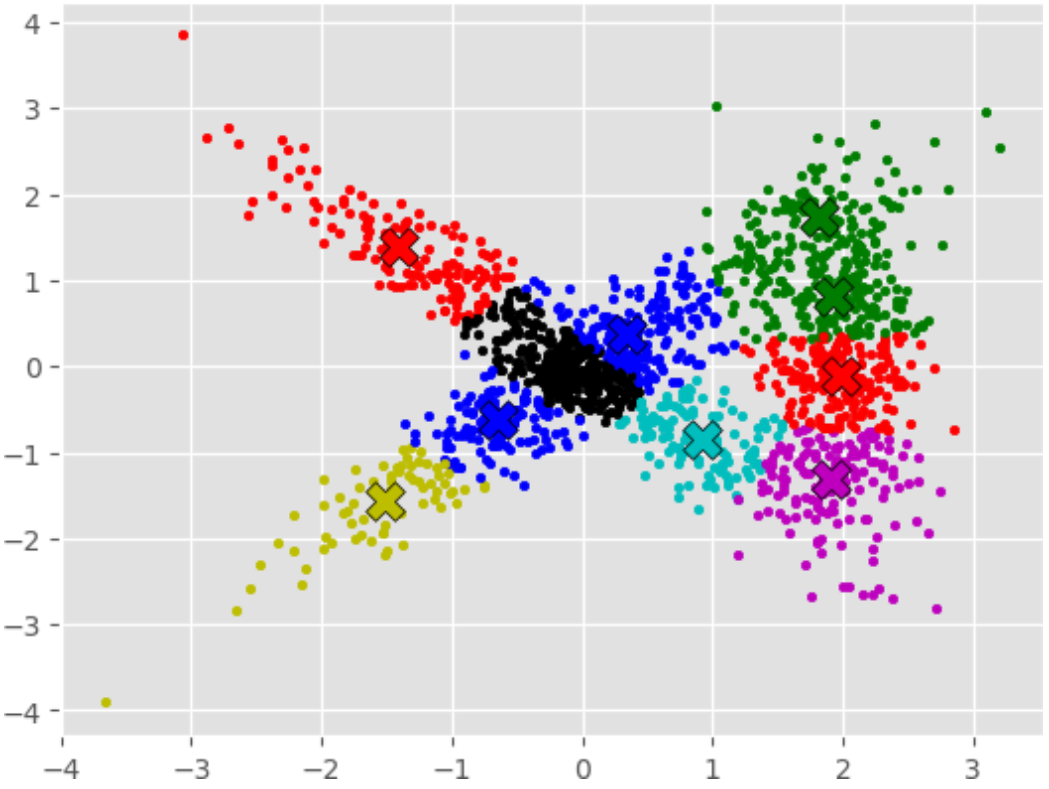

CMeansGraph43_step0
SquareError: 1196.96210571269
4
3
2
1
0
−1
−2
−3
−4
−4
−3
−2
−1
0
1
2
3

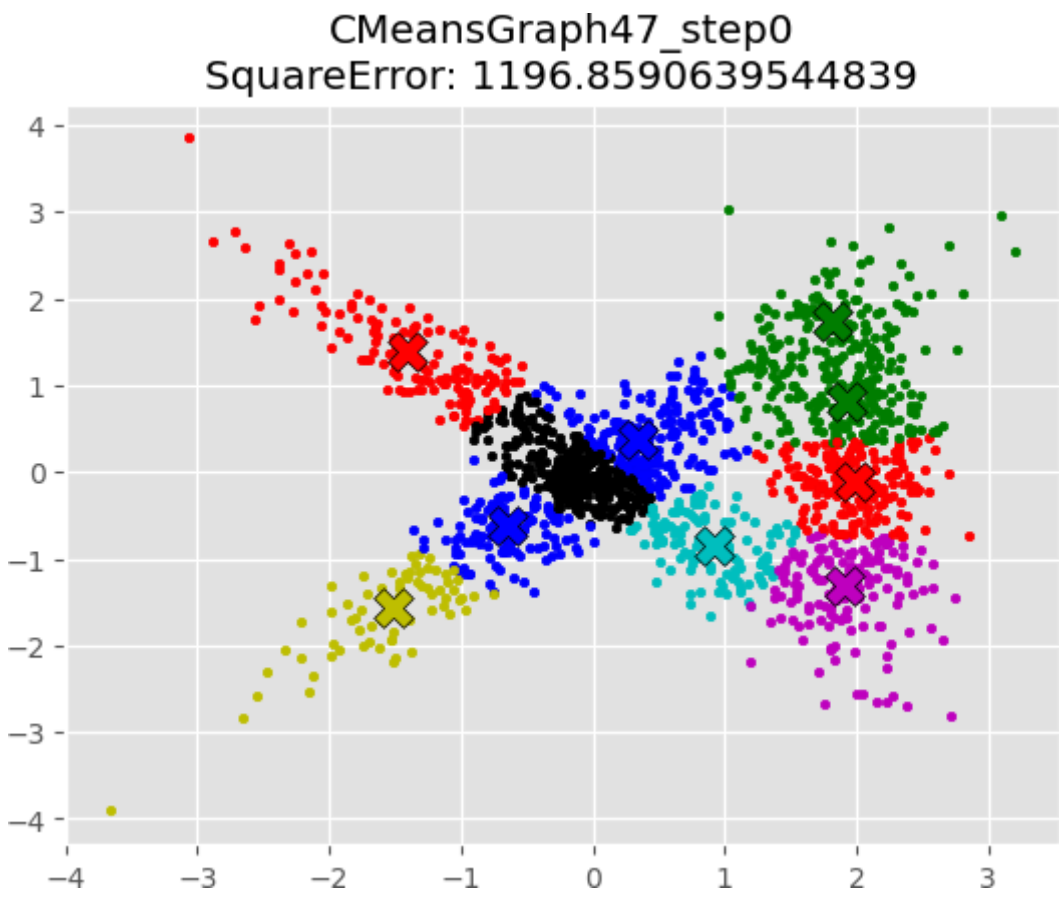

CMeansGraph44_step0
SquareError: 1196.9178123600855
4
3
2
1
0
−1
−2
−3
−4
−4
−3
−2
−1
0
1
2
3

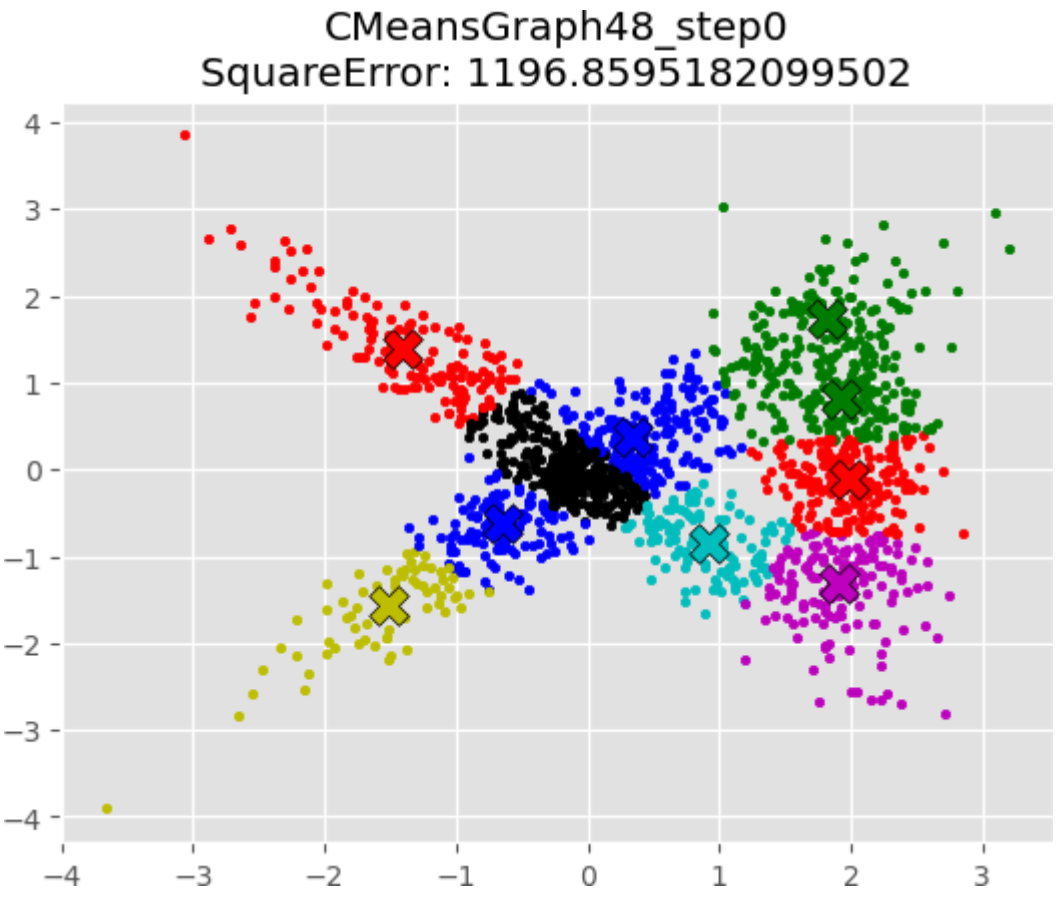
CMeansGraph45_step0
SquareError: 1196.886612731284



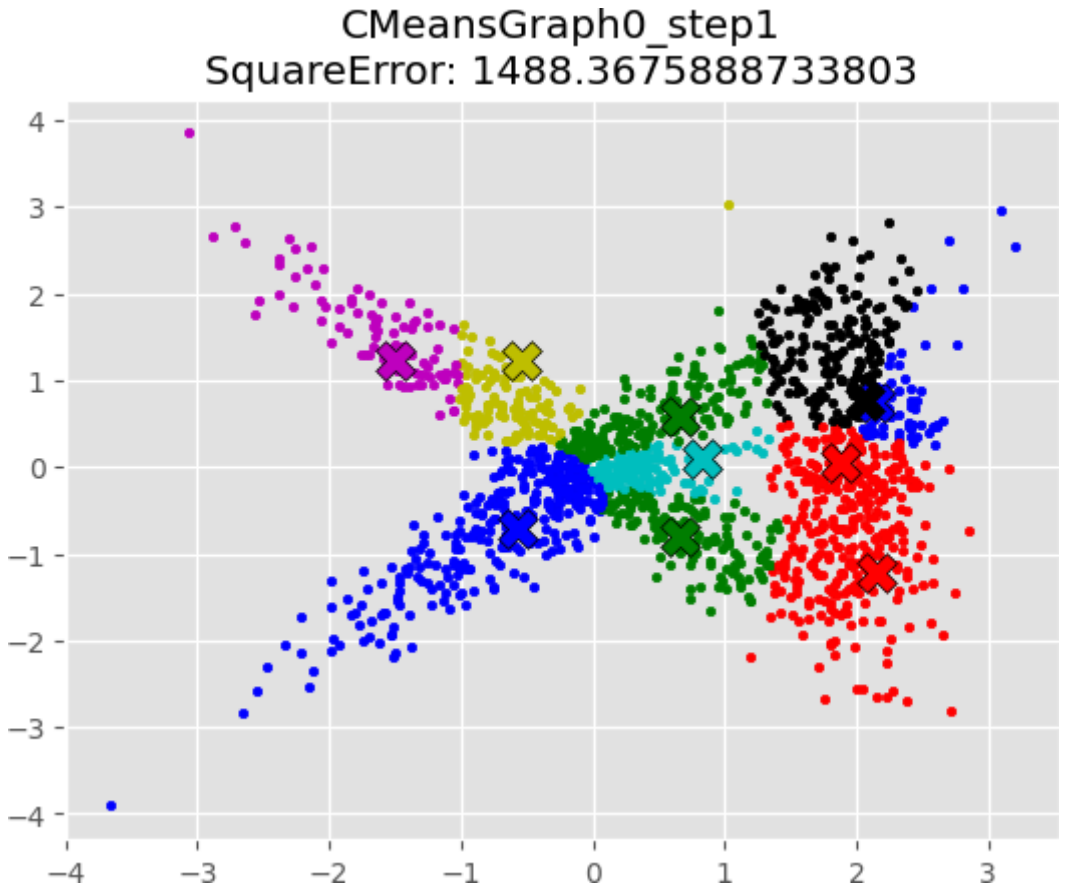
CMeansGraph46_step0
SquareError: 1196.8675357231757

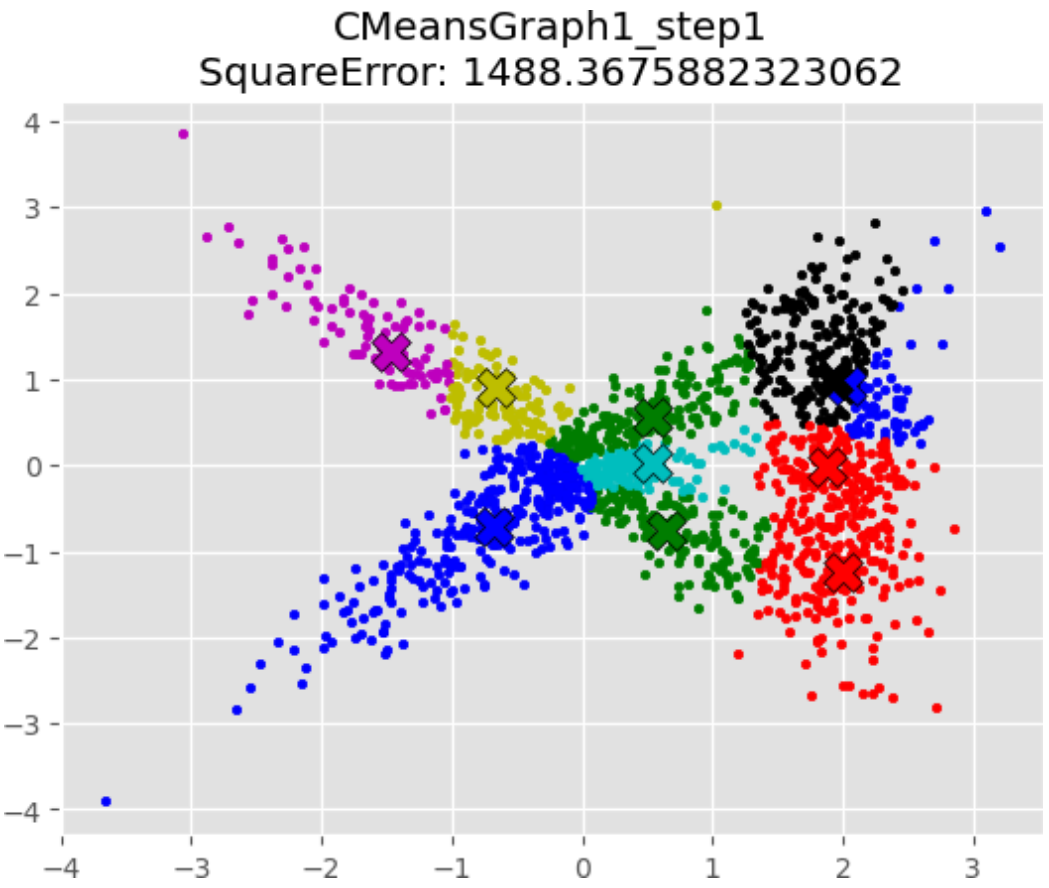


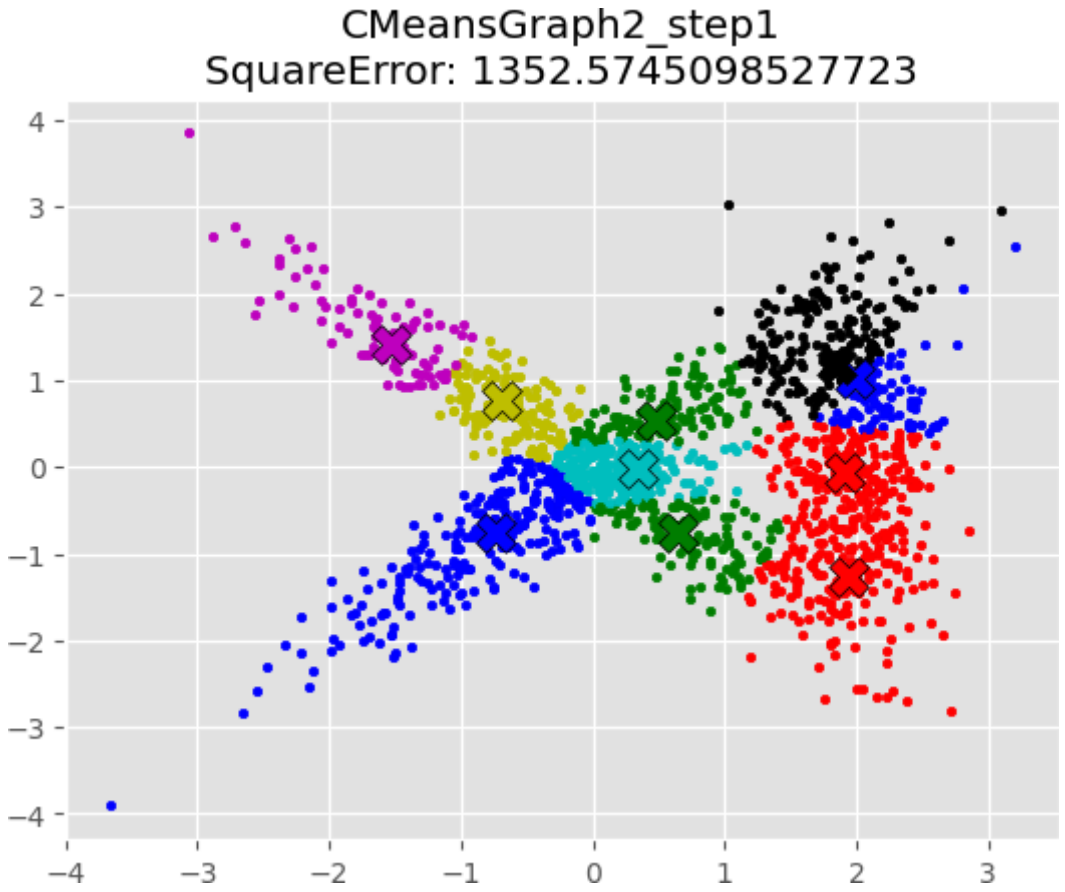


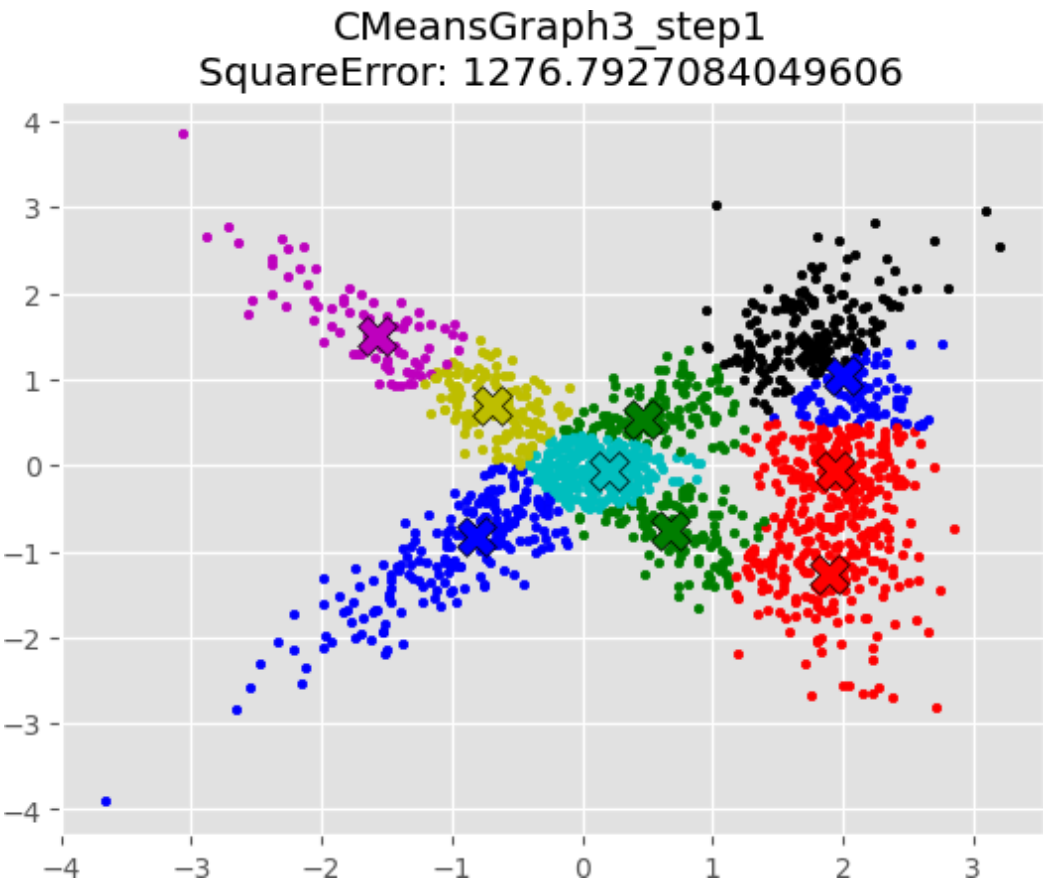


The stopping criterion of reaching epsilon has been met. The new best square error is 1.

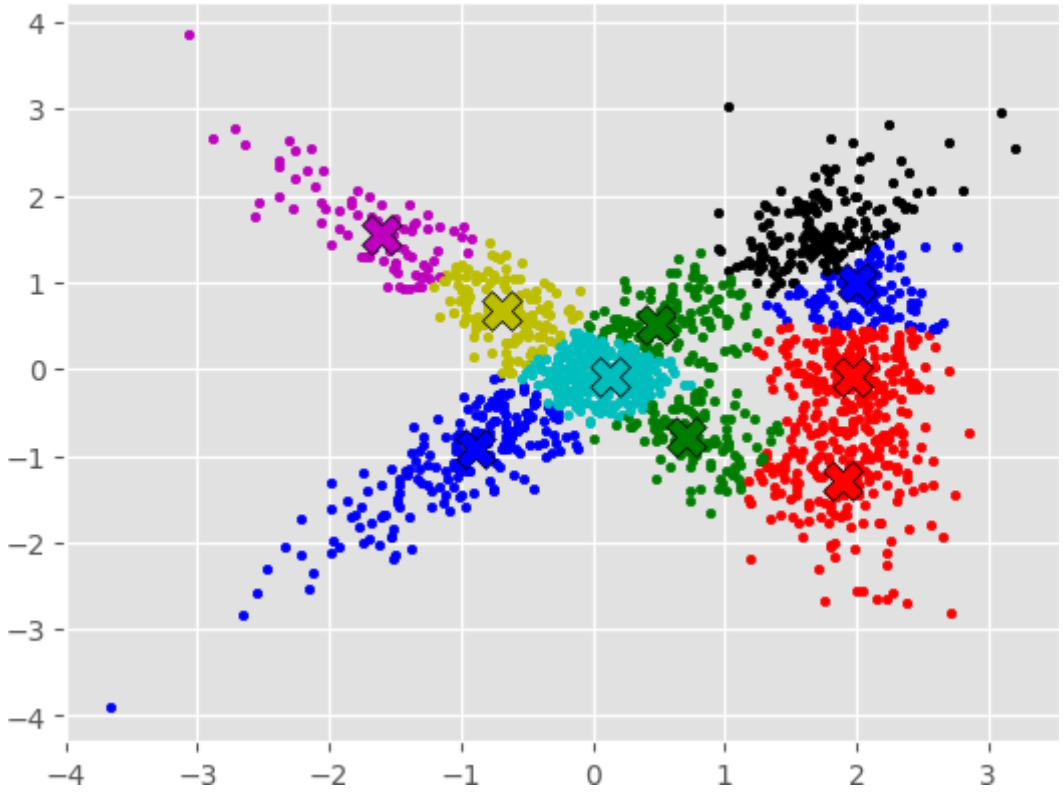

CMeansGraph0_step1
SquareError: 1488.3675888733803
4
3
2
1
0
−1
−2
−3
−4
−4
−3
−2
−1
0
1
2
3


CMeansGraph1_step1
SquareError: 1488.3675882323062
4
3
2
1
0
−1
−2
−3
−4
−4
−3
−2
−1
0
1
2
3

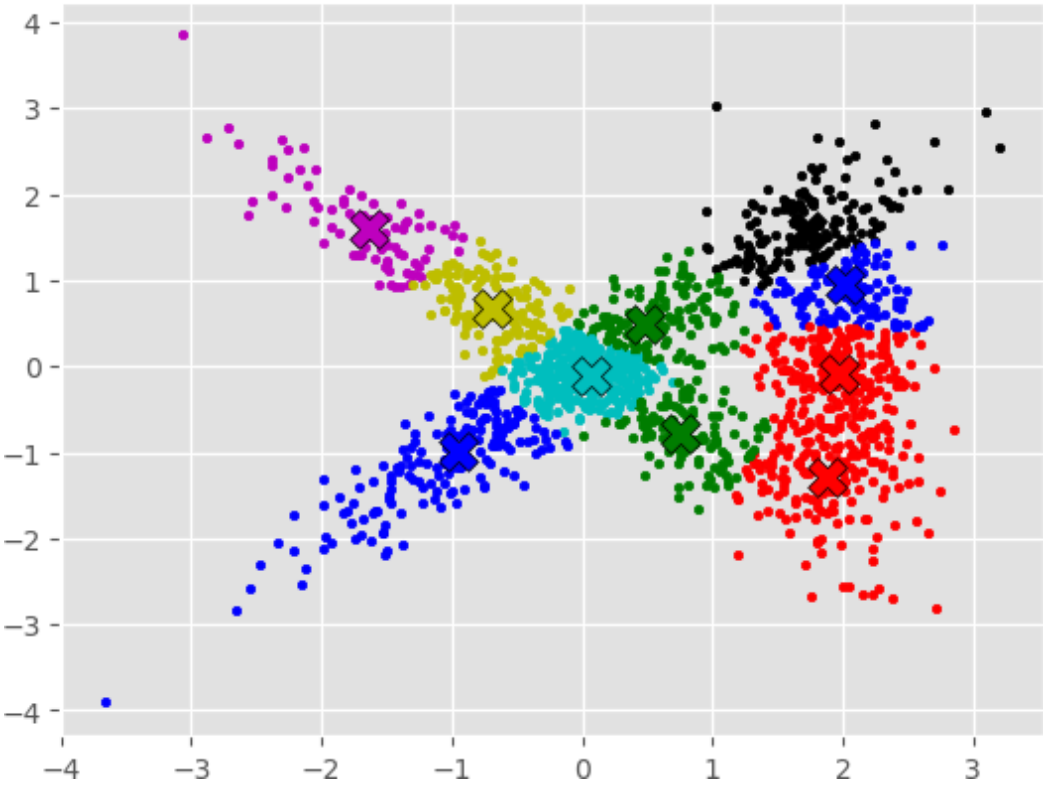

CMeansGraph2_step1
SquareError: 1352.5745098527723
4
3
2
1
0
−1
−2
−3
−4
−4
−3
−2
−1
0
1
2
3


CMeansGraph3_step1
SquareError: 1276.7927084049606
4
3
2
1
0
−1
−2
−3
−4
−4
−3
−2
−1
0
1
2
3

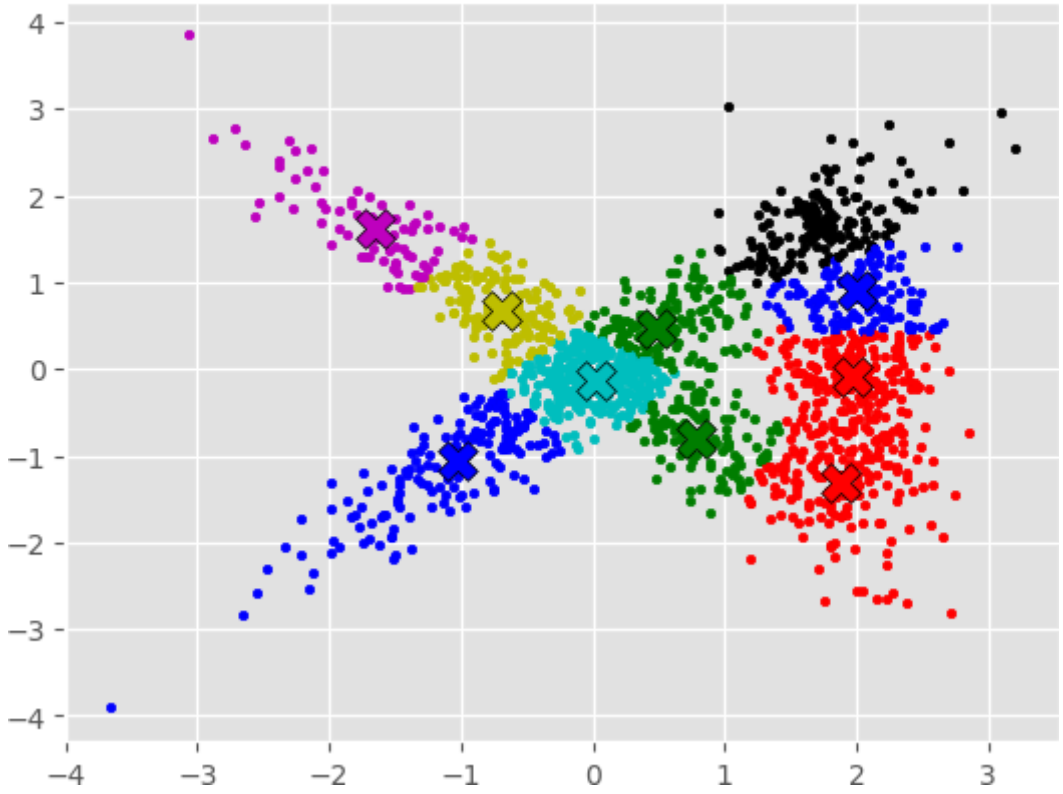
CMeansGraph4_step1
SquareError: 1236.4404211126578



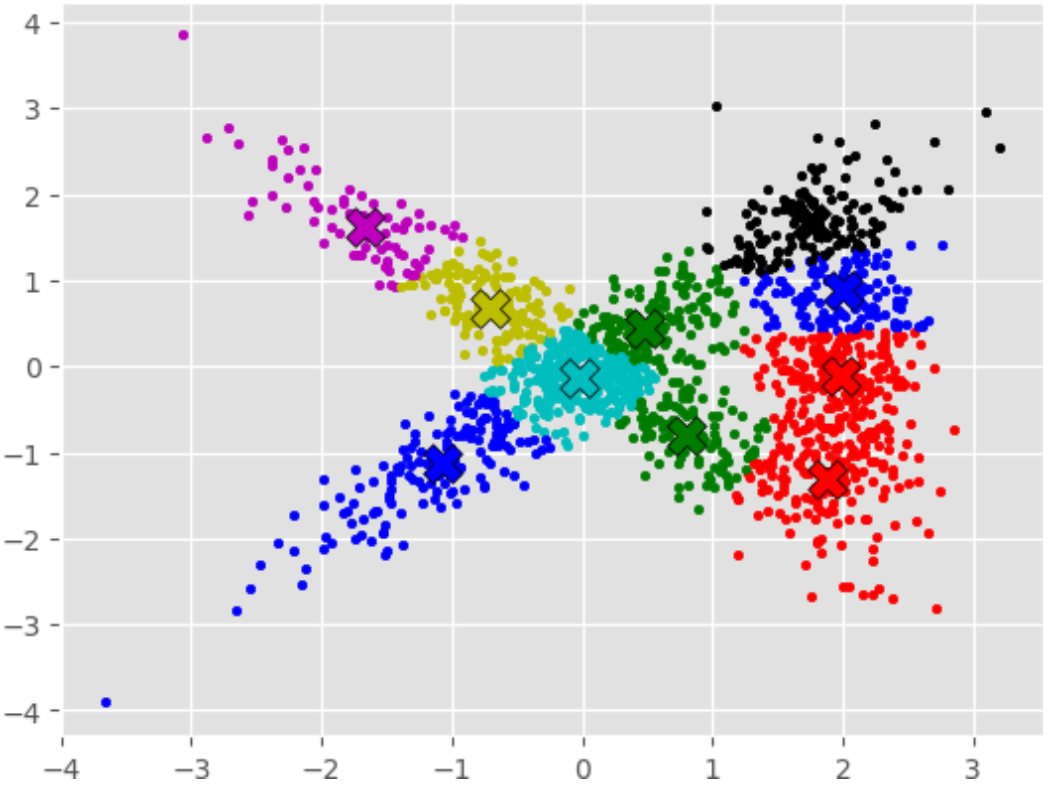
CMeansGraph5_step1
SquareError: 1212.9645232344499



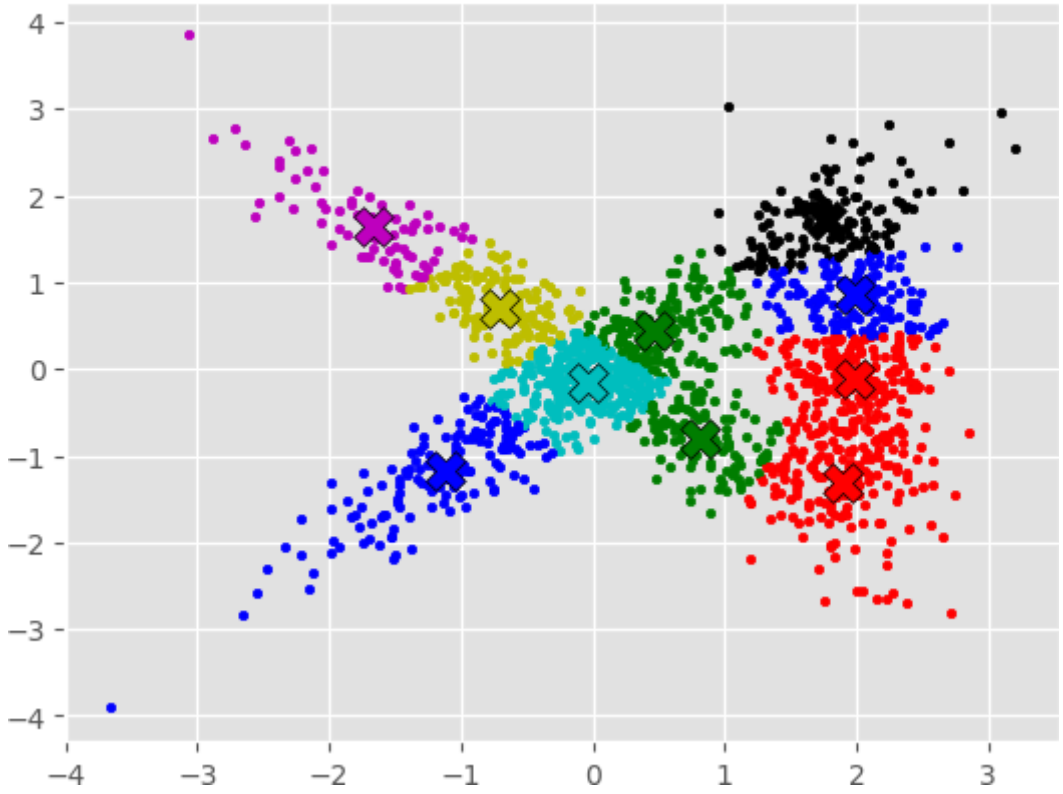
CMeansGraph6_step1
SquareError: 1198.9235552253808



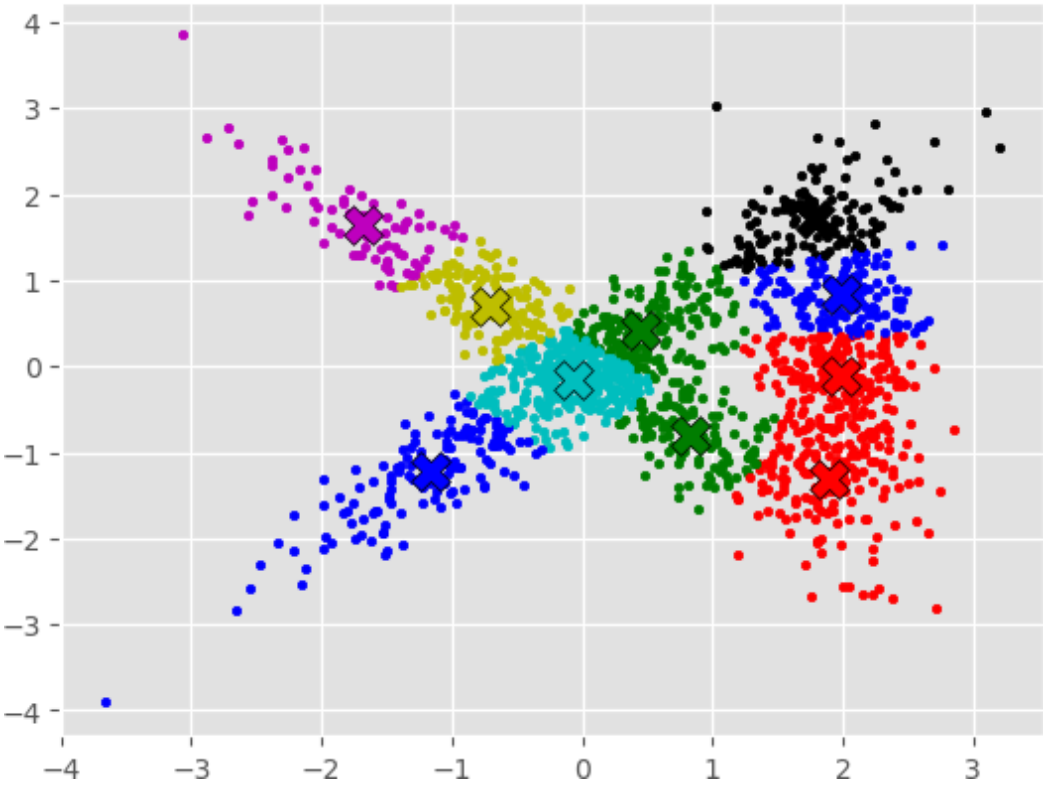
CMeansGraph7_step1
SquareError: 1190.581113657334



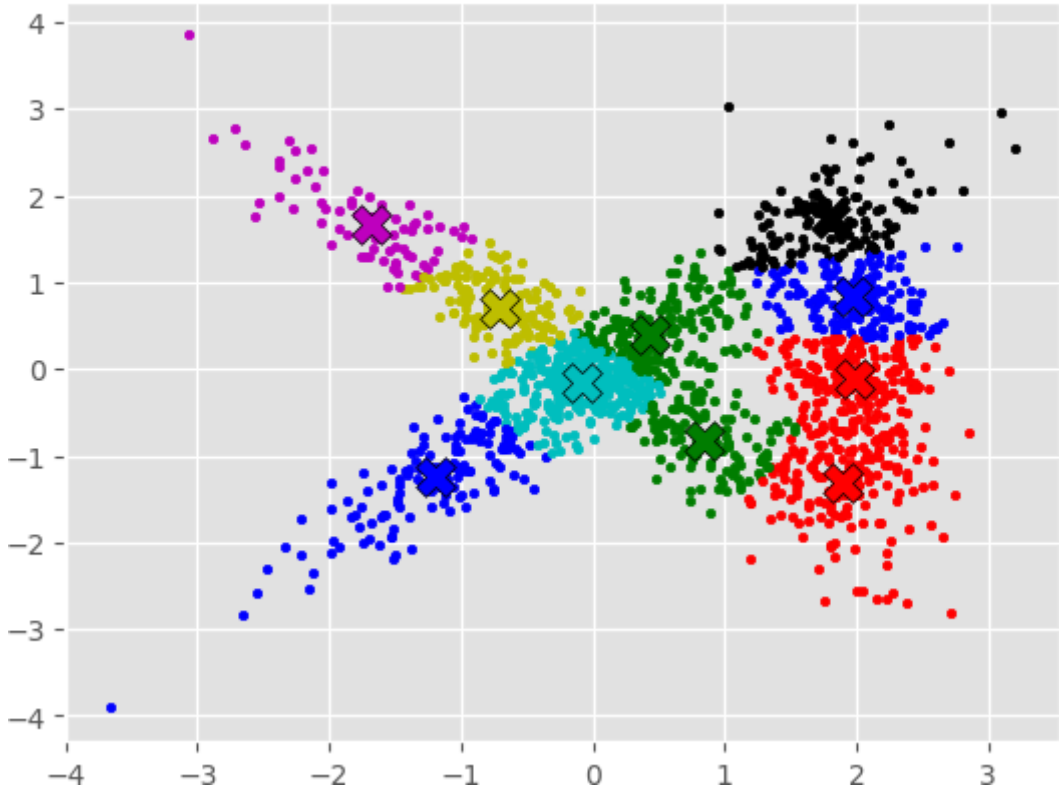
CMeansGraph8_step1
SquareError: 1185.8152238595153



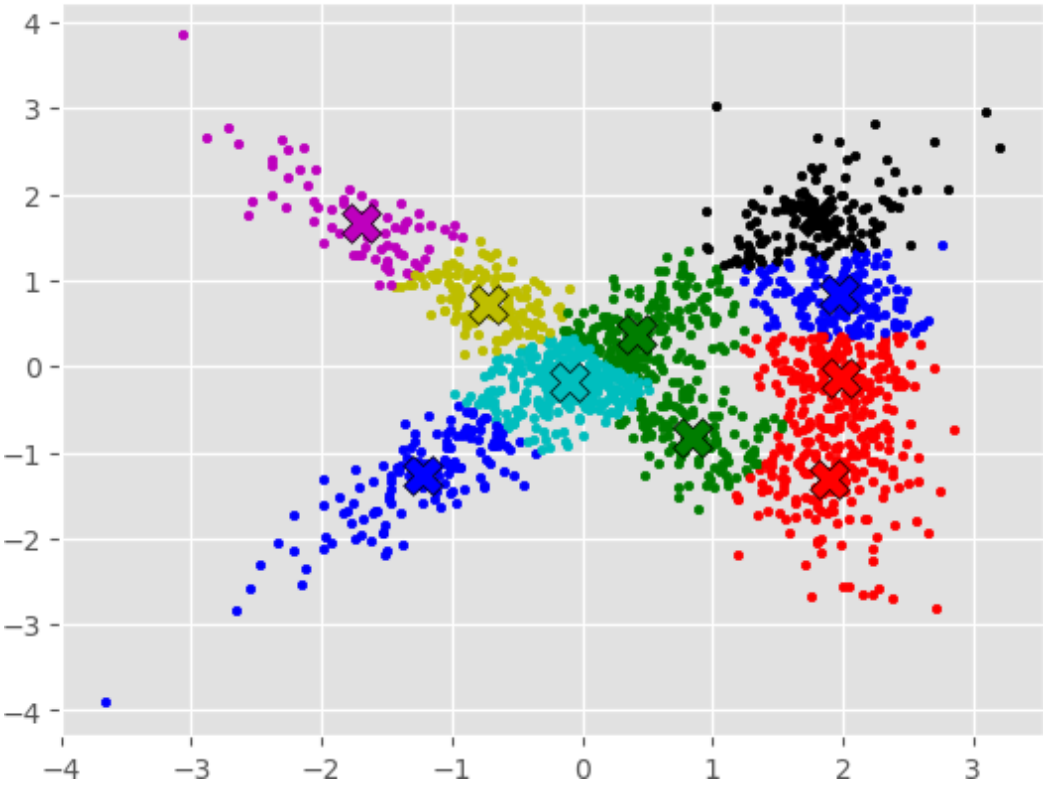
CMeansGraph9_step1
SquareError: 1183.2474779422985



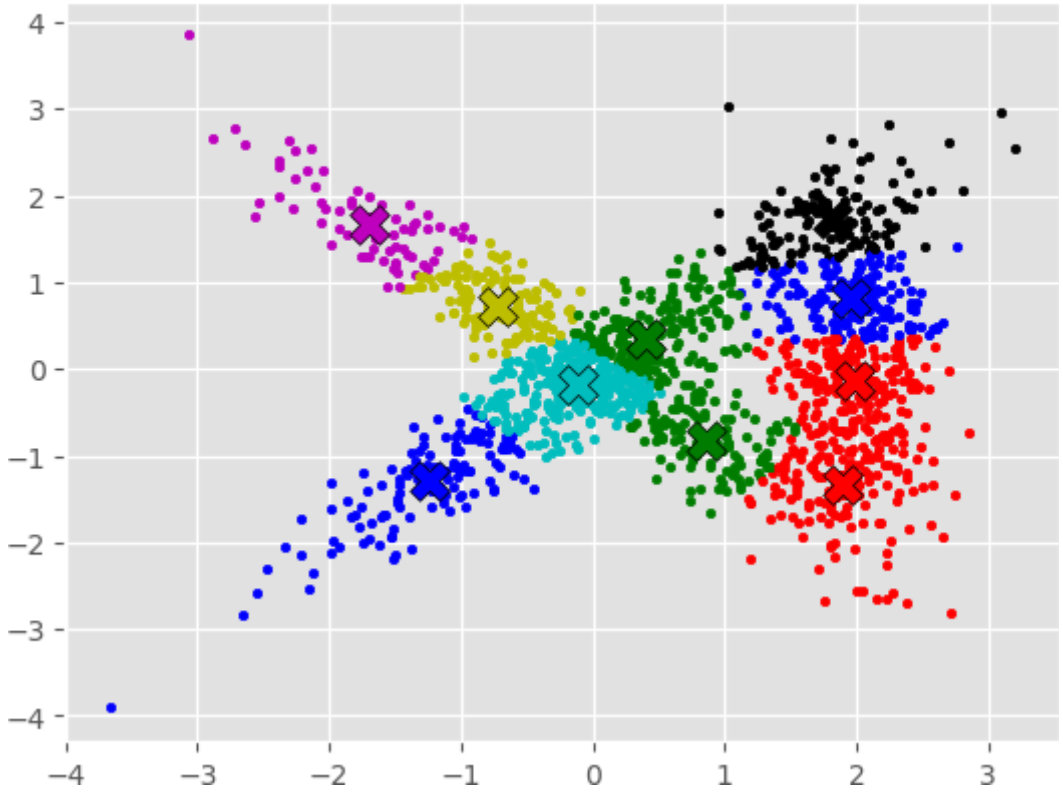
CMeansGraph10_step1
SquareError: 1181.8132169294431



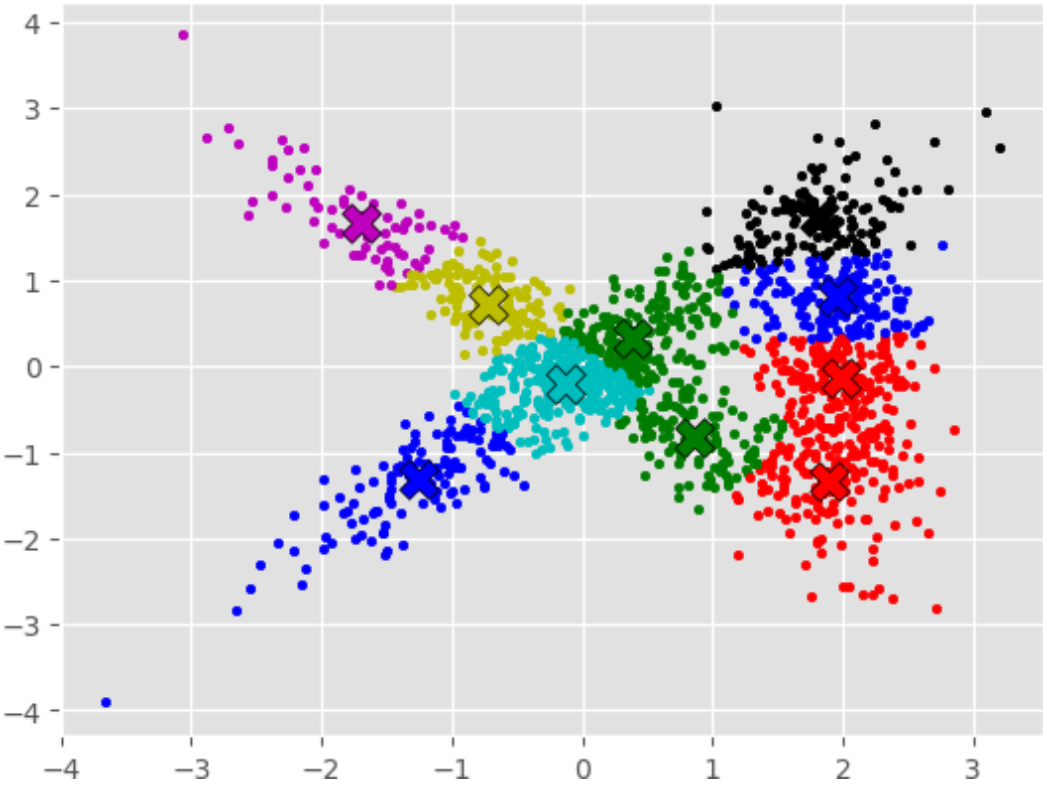
CMeansGraph11_step1
SquareError: 1180.8717720313414



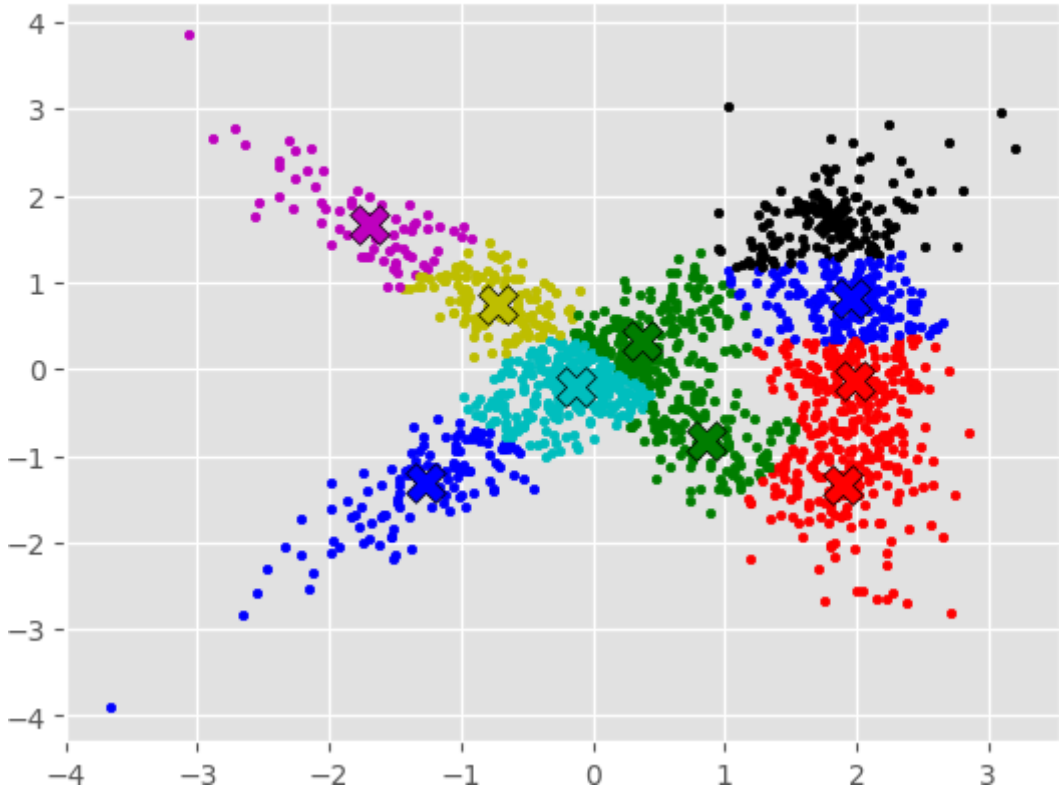
CMeansGraph12_step1
SquareError: 1180.1467177977734



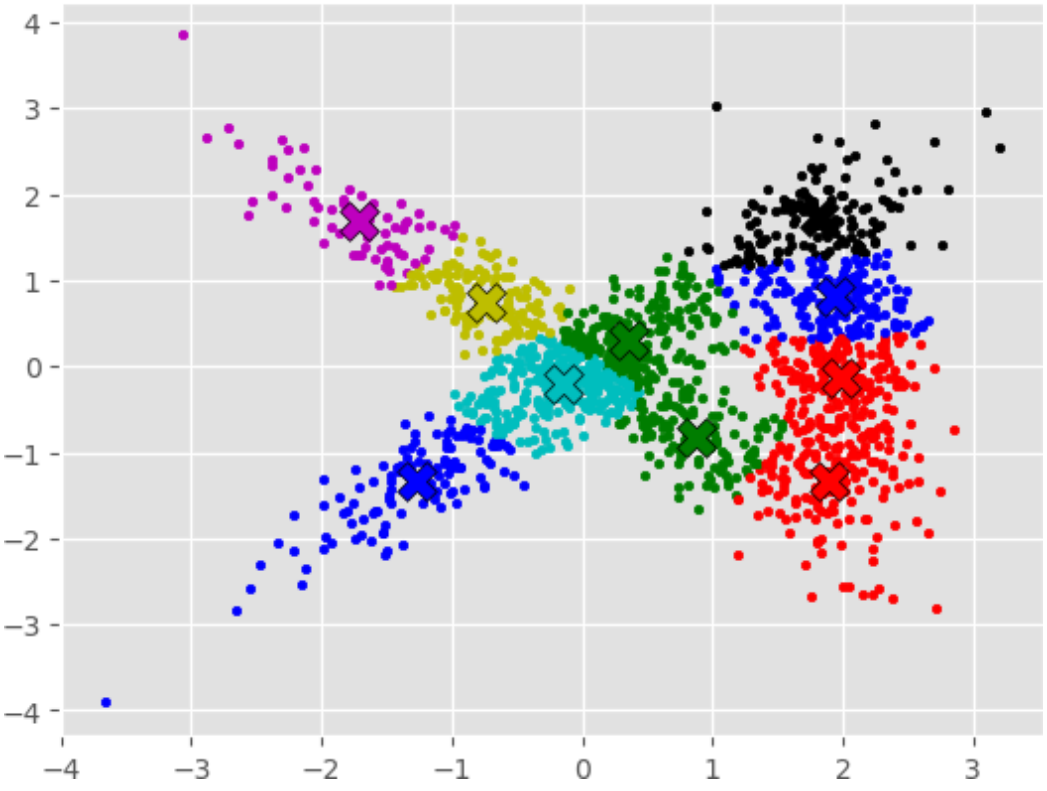
CMeansGraph13_step1
SquareError: 1179.505632109272



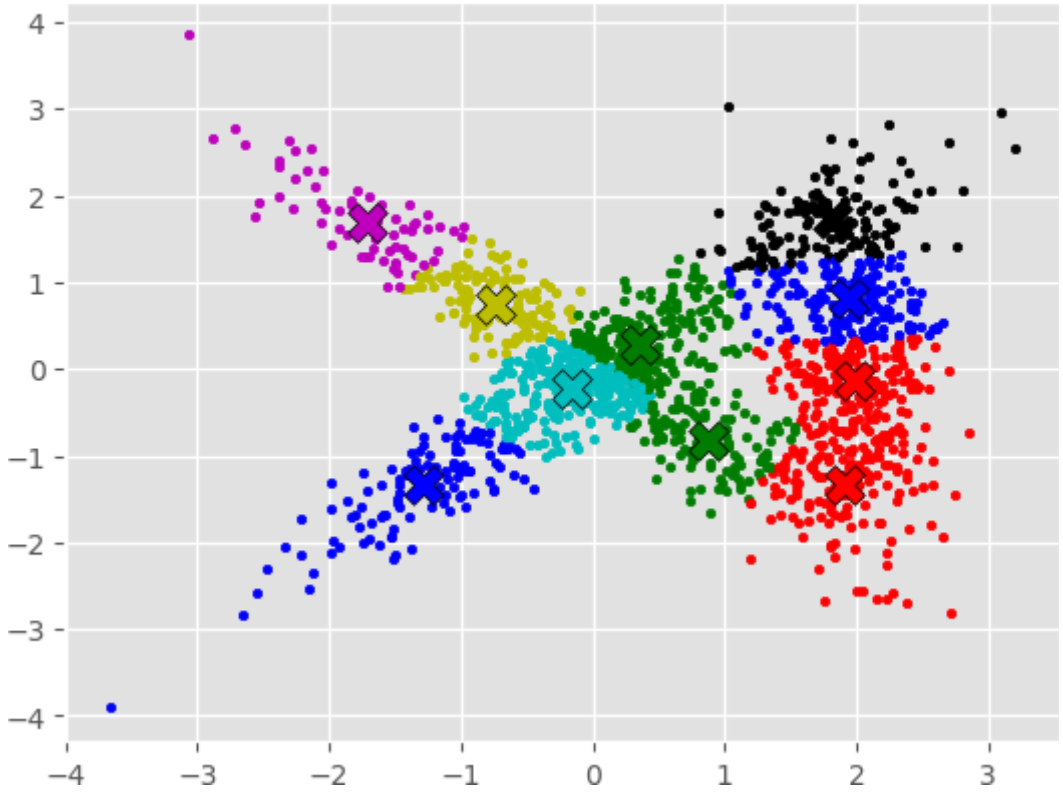
CMeansGraph14_step1
SquareError: 1178.9568579644726



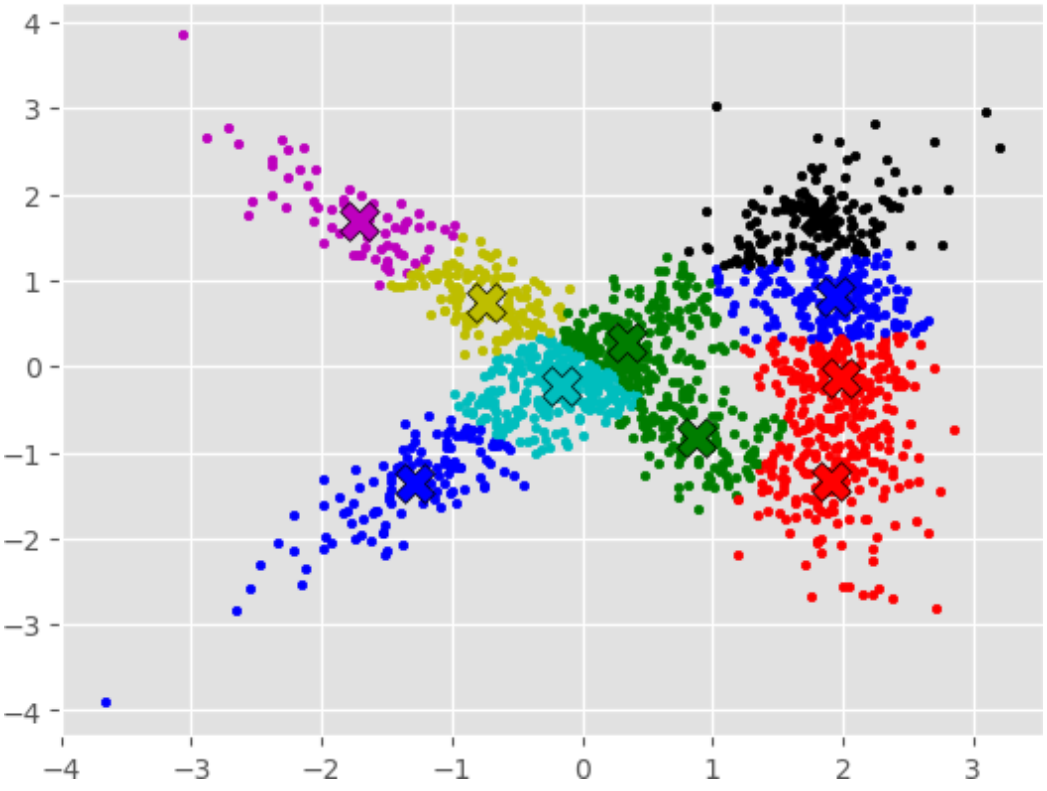
CMeansGraph15_step1
SquareError: 1178.5091308715412



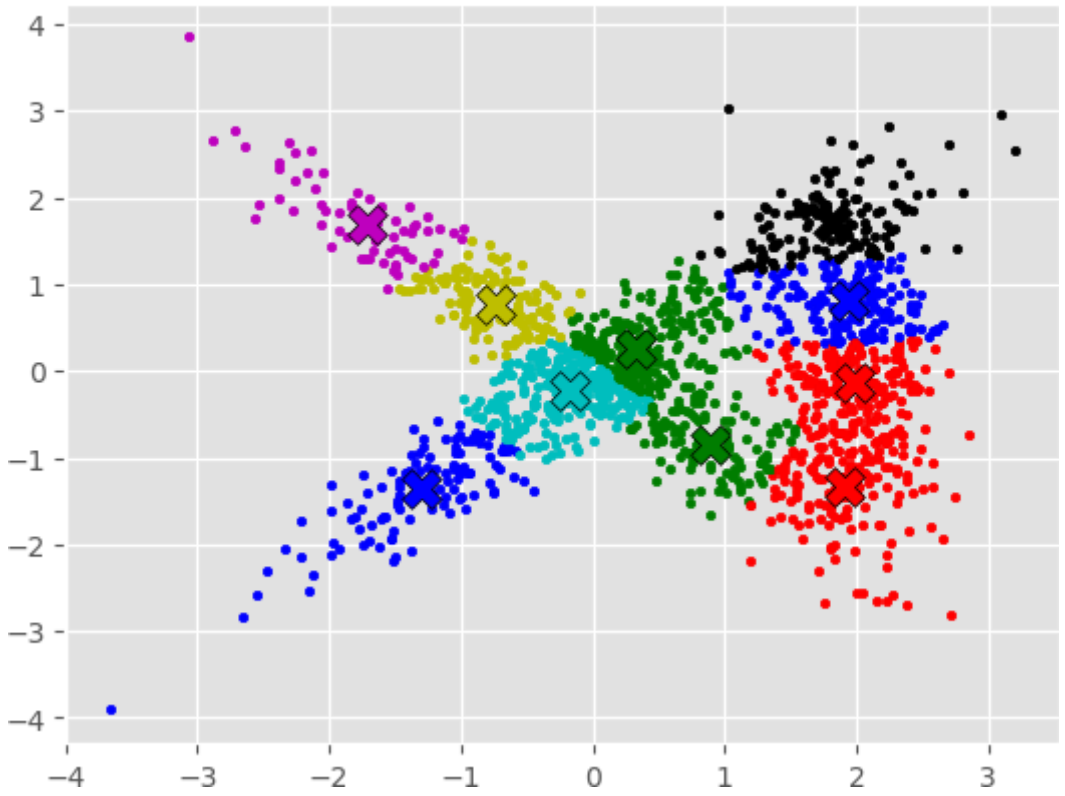
CMeansGraph16_step1
SquareError: 1178.1445333033867



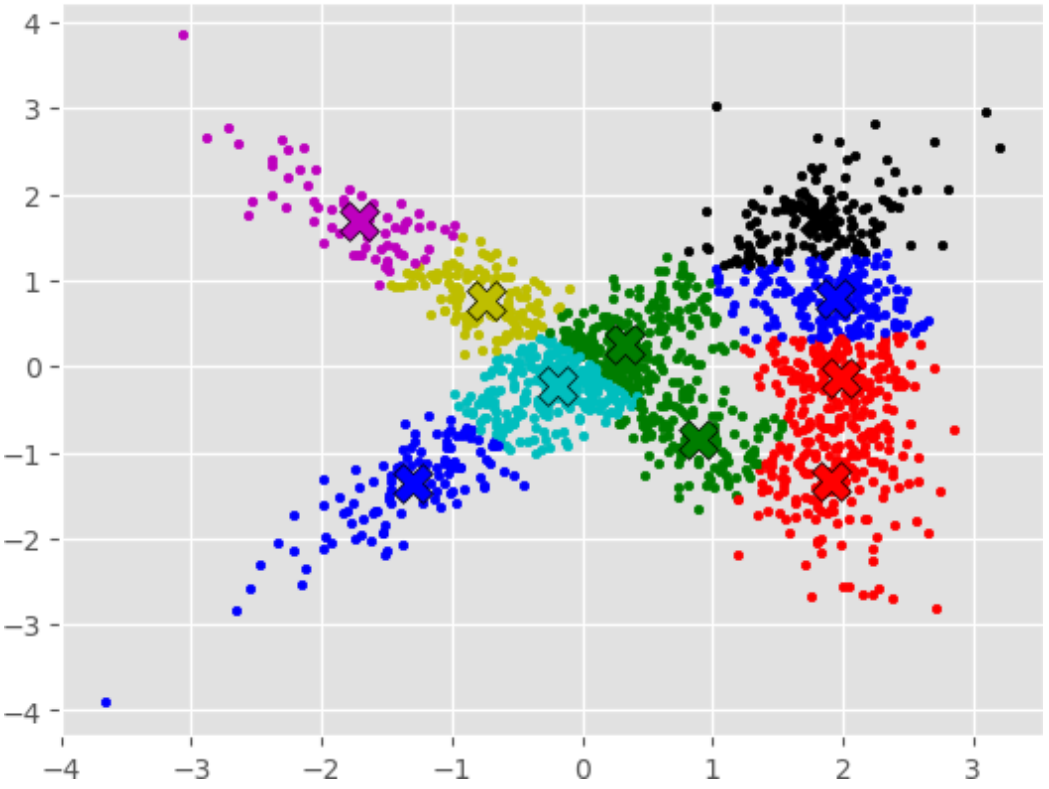
CMeansGraph17_step1
SquareError: 1177.847267381007



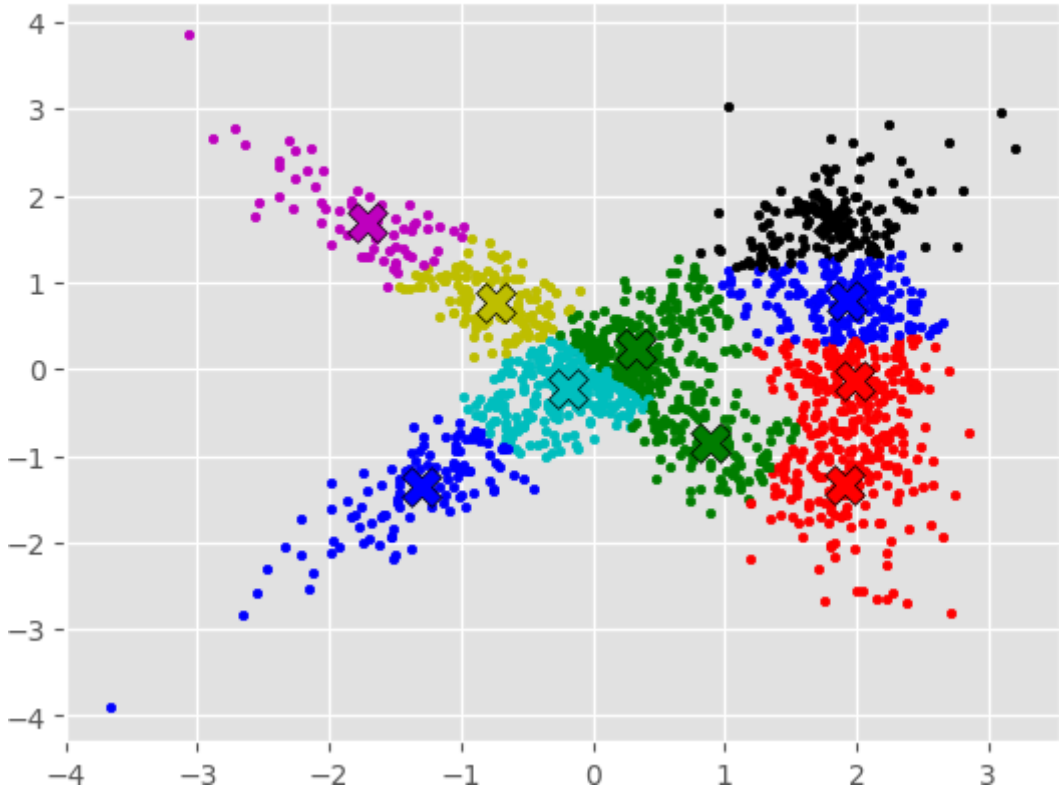
CMeansGraph18_step1
SquareError: 1177.611888539641



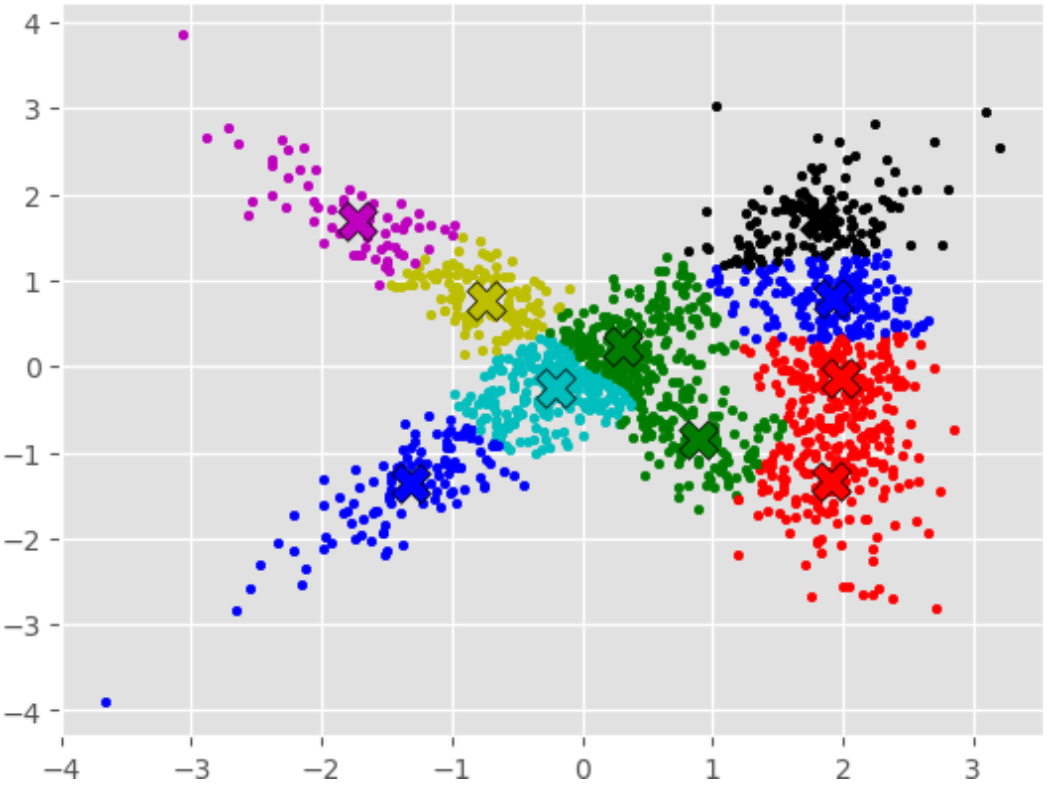
CMeansGraph19_step1
SquareError: 1177.4358967584244



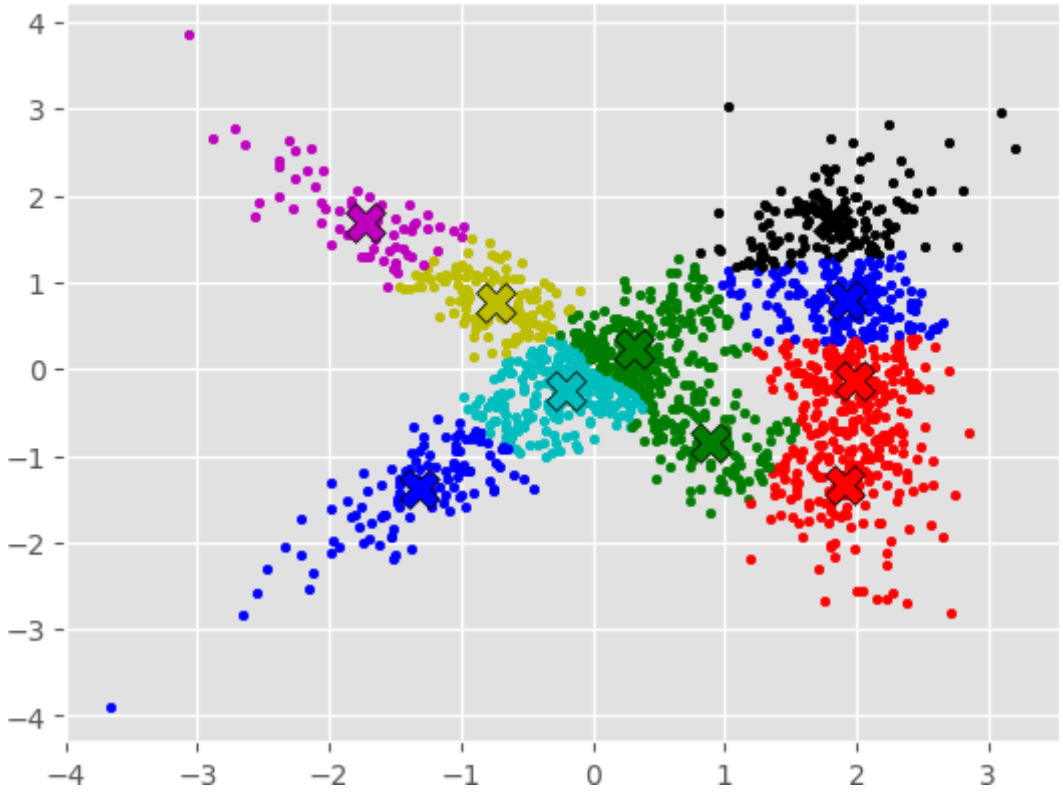
CMeansGraph20_step1
SquareError: 1177.3180264769173



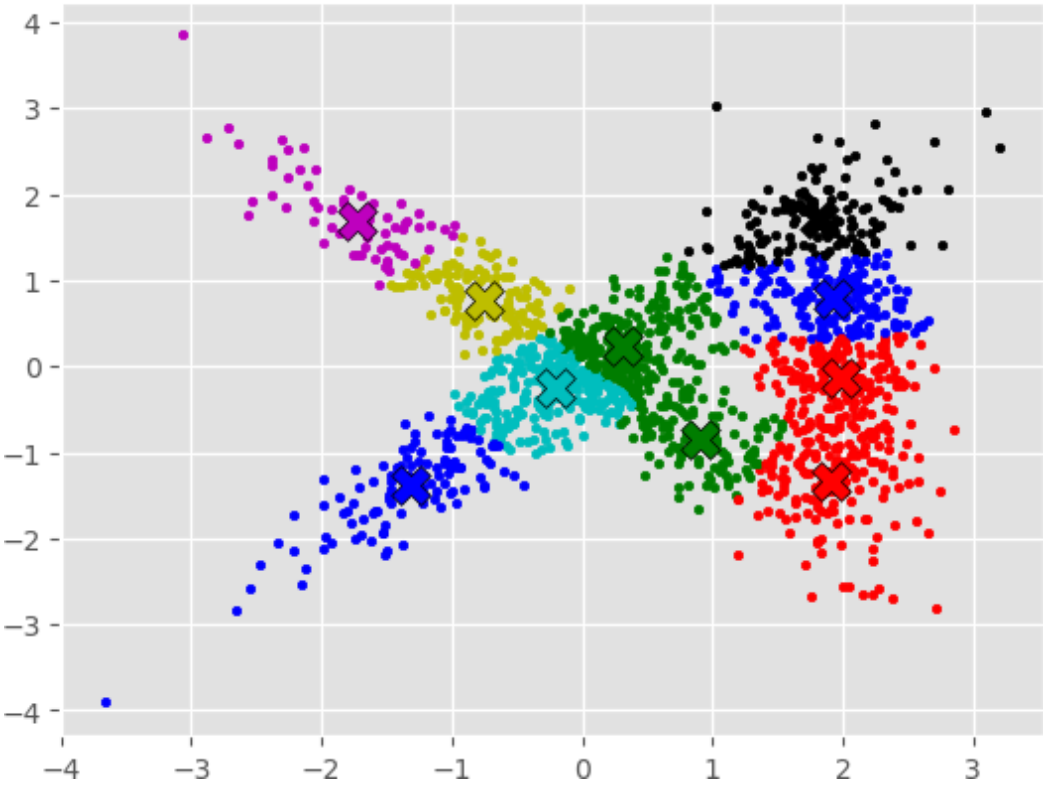
CMeansGraph21_step1
SquareError: 1177.247365708964



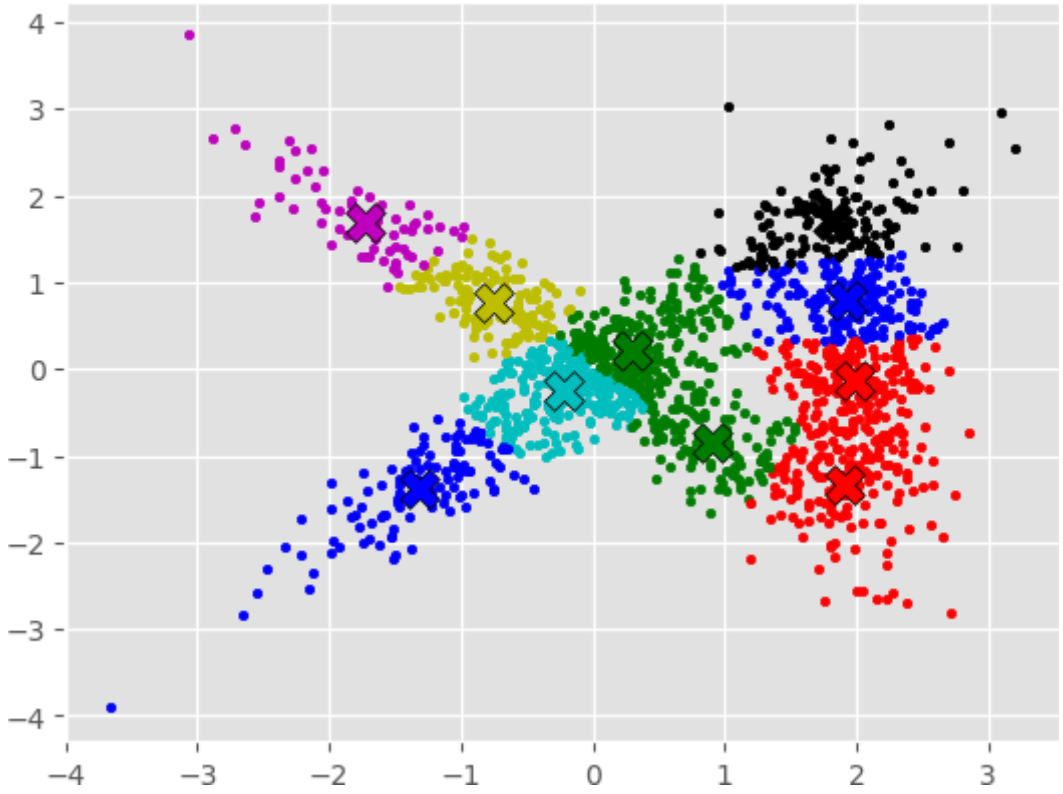
CMeansGraph22_step1
SquareError: 1177.2331628684608



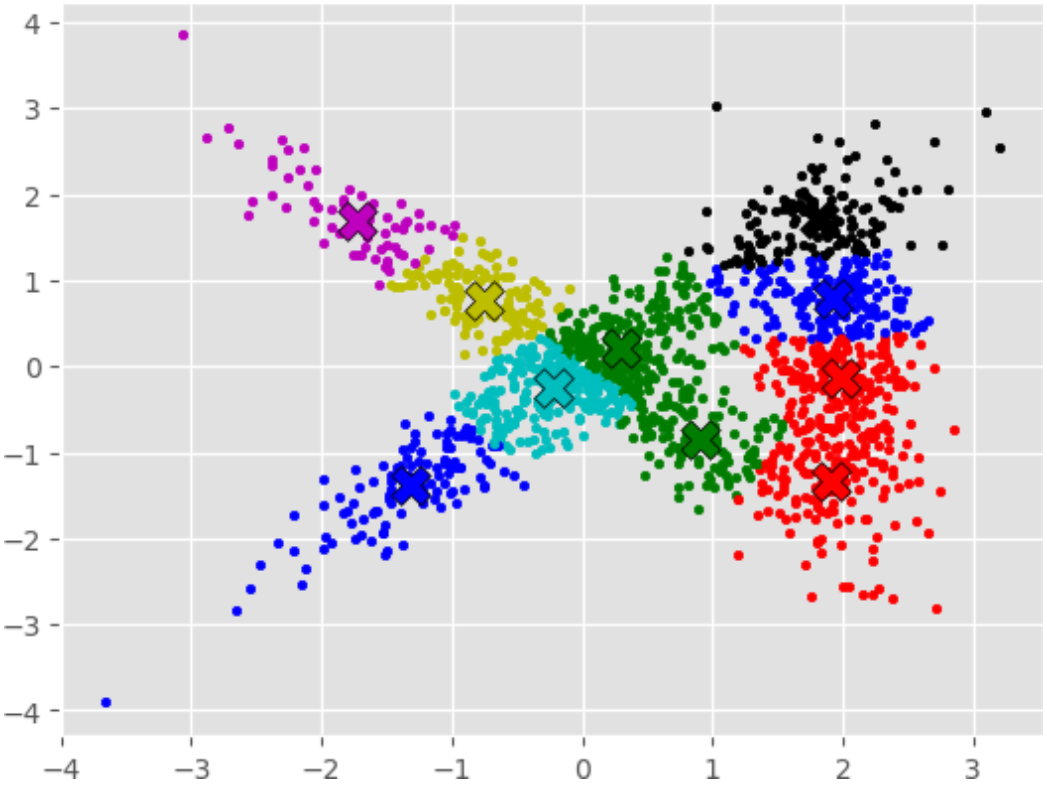
CMeansGraph23_step1
SquareError: 1177.257861398735



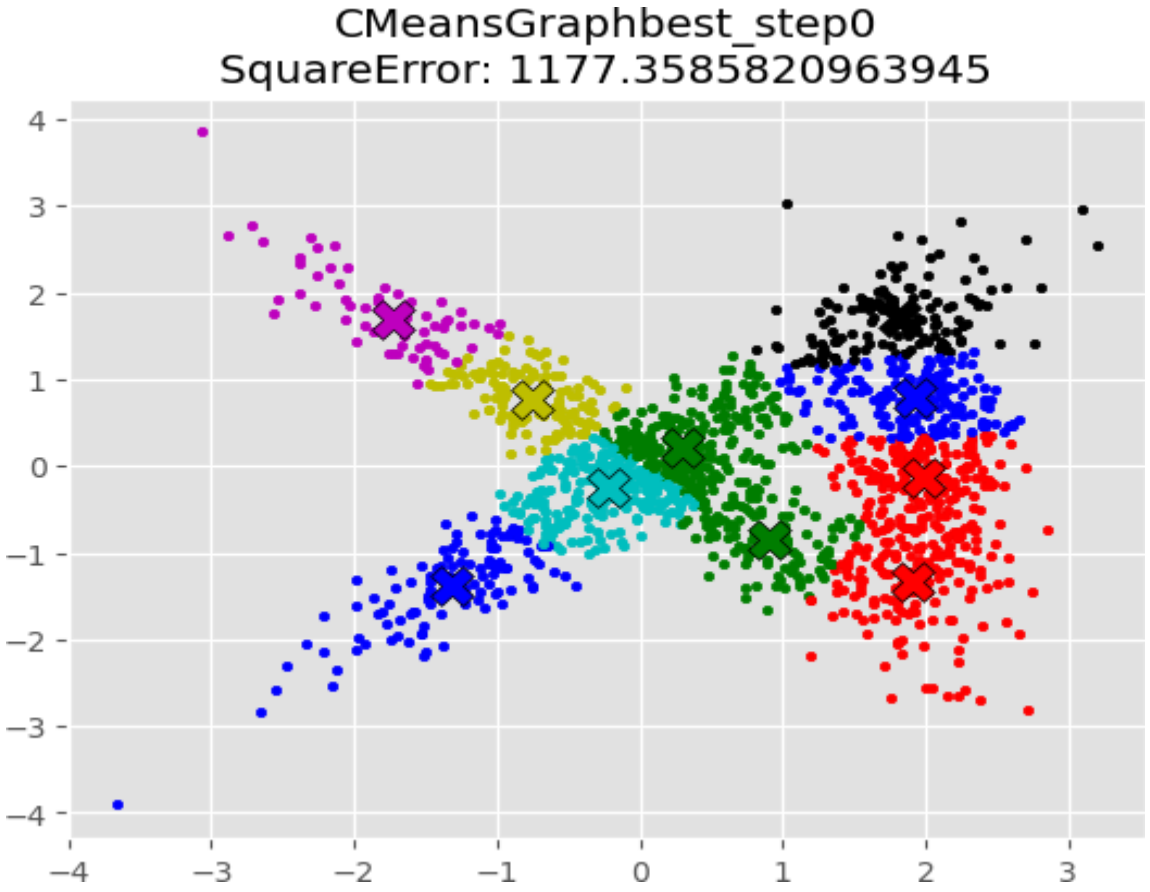
CMeansGraph24_step1
SquareError: 1177.3025326609034



CMeansGraph25_step1
SquareError: 1177.3585820963945



r


CMeansGraphbest_step0
SquareError: 1177.3585820963945
4
3
2
1
0
−1
−2
−3
−4
−4
−3
−2
−1
0
1
2
3