```
        new_centroids = update_centroid(data, grouping, centroids)
        J_obj = clustering_objective(data, grouping,
        ↪  new_centroids)
        J_obj_vector.append(J_obj)
        iteration += 1
        if np.linalg.norm(np.array(new_centroids) -
        ↪  np.array(centroids)) < 1e-6:
            Stop = True
        else:
            centroids = new_centroids
    return new_centroids, grouping, J_obj_vector, iteration
```

Convergence. Here we use a `while` loop, which executes the statements inside the loop as long as the condition `Stop == False` is true. We terminate the algorithm when the improvement in the clustering objective becomes very small (`1e-6`).

Alternatively, we can use the `Kmeans` function in the `cluster` module of the `sklearn` package.

In [ ]:
```
from sklearn.cluster import KMeans
import numpy as np
kmeans = KMeans(n_clusters=4, random_state=0).fit(data)
labels = kmeans.labels_
group_representative = kmeans.cluster_centers_
J_clust = kmeans.inertia_
```

Here we try to apply the k-means algorithm on `data`, clustering the vectors into 4 groups. Note that the `sklearn.cluster.KMeans` function initialize the algorithms with random centroids and thus the initial values of centroids are not required as an argument but the random state to draw the random initialization is.

4.4. Examples

We apply the algorithm on a randomly generated set of $N = 300$ points, shown in Figure 4.1. These points were generated as follows.

In [ ]:
```
import matplotlib.pyplot as plt
plt.ion()
```

```
X = np.concatenate([[0.3*np.random.randn(2) for i in range(100)],
↪  [[1,1] + 0.3*np.random.randn(2) for i in range(100)], [[1,-1]
↪  + 0.3* np.random.randn(2) for i in range(100)]])
plt.scatter( X[:,0],X[:,1])
plt.xlim(-1.5,2.5)
plt.ylim(-2,2)
plt.show()
```

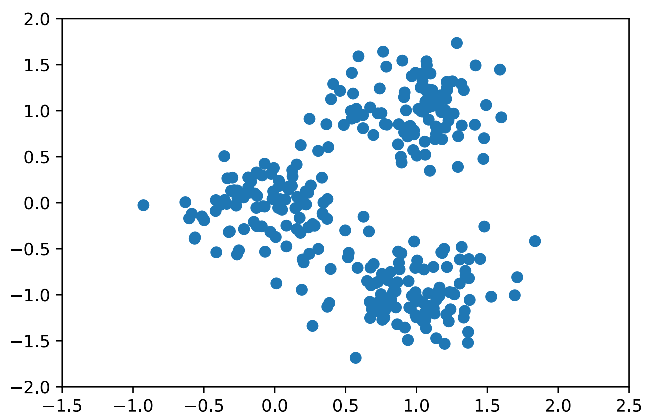

Figure 4.1.: 300 points in a plane.

On the first line, we import the `matplotlib` package for plotting. Then we generate three arrays of vectors. Each set consists of 100 vectors chosen randomly around one of the three points (0,0),(1,1), and (1,-1). The three arrays are concatenated using `np.concatenate()` to get an array of 300 points. Next, we apply the `KMeans` function and make a figure with the three clusters (Figure 4.2).

```
In [ ]: from sklearn.cluster import KMeans
        import numpy as np
        kmeans = KMeans(n_clusters=3, random_state=0).fit(X)
        labels = kmeans.labels_
        group_representative = kmeans.cluster_centers_
        J_clust = kmeans.inertia_

        grps = [[X[i,:] for i in range(300) if labels[i]==j] for j in
        ↪  range(3)]
```

```
plt.scatter([c[0] for c in grps[0]],[c[1] for c in grps[0]])
plt.scatter([c[0] for c in grps[1]],[c[1] for c in grps[1]])
plt.scatter([c[0] for c in grps[2]],[c[1] for c in grps[2]])
plt.xlim(-1.5,2.5)
plt.ylim(-2,2)
plt.show()
```

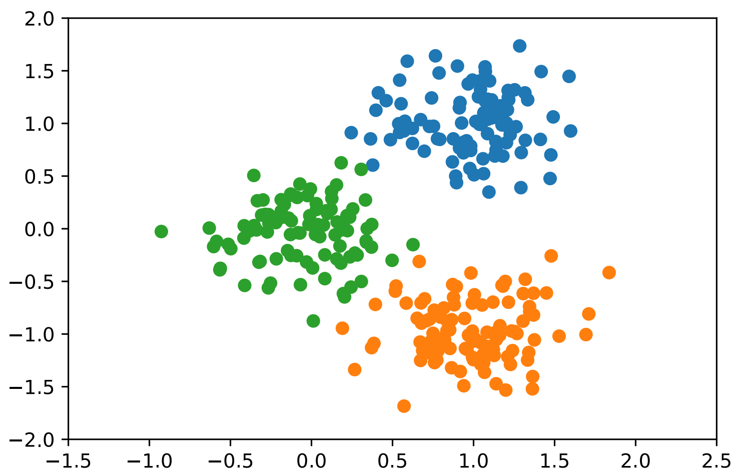

Figure 4.2.: Final clustering.

4.5. Applications