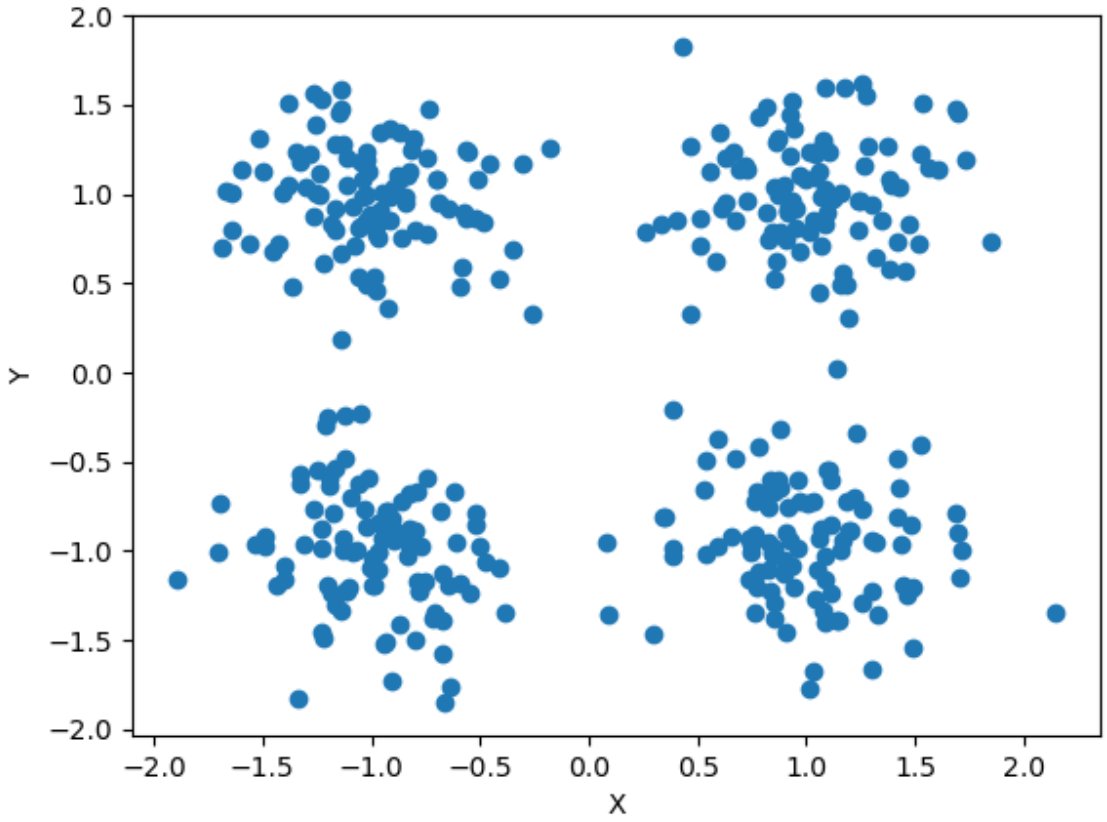
Nicholas Kiprop
Matrikel number 1748461

1 Load the Data

A. Download the provided data file `cluster data.csv`
B. Load the contents into your program:
C. Recommended way is `pandas → pd.read csv("cluster data.csv", header=0, index col=0)`
D. Plot the data with `matplotlib` (See Scatter-plot for examples)



DDA_exercise3.py

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from mpi4py import MPI

# initialize MPI
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()

#load data form csv file into pandas dataframe called pandas
#specify column header is the first row in data file
# index_col 0 specifies that the first column in the CSV file should be used as the DataFrame index.
pandas = pd.read_csv("cluster_data.csv", header=0, index_col=0)

plt.scatter(pandas['x'], pandas['y'])
plt.xlabel('X')
plt.ylabel('Y')
plt.show()
```

DDA_exercise3.py

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from mpi4py import MPI

# initialize MPI
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()

#load data form csv file into pandas dataframe called pandas
#specify column header is the first row in data file
# index_col 0 specifies that the first column in the CSV file should be used as the
DataFrame index.
pandas = pd.read_csv("cluster_data.csv", header=0, index_col=0)

plt.scatter(pandas['x'], pandas['y'])
plt.xlabel('X')
plt.ylabel('Y')
plt.show()



# spplit data into equal chinks
chunk_size = len(pandas) // size
chunk_start = rank * chunk_size
chunk_end = chunk_start + chunk_size
if rank == size - 1:
    chunk_end = len(pandas)
chunk = pandas[chunk_start:chunk_end]


# initialize the centroids
centroids = np.random.uniform(low = 0, high=10, size=(4, 2))

# Assign points to the closest centroids
distance = np.sqrt(((chunk.values - centroids[:, np.newaxis])**2).sum(axis=2))
assignment = np.argmin(distance, axis=0)

# compute the local centroids for this MPI Process
local_centroids = np.zeros((4, 2))
counts = np.zeros(4)
for i in range(len(chunk)):
    local_centroids[assignment[i]] += chunk.iloc[i][['x', 'y']]
    counts[assignment[i]] += 1
local_centroids /= np.where(counts == 0, 1e-9, counts)[:, np.newaxis]


#Reduce the local centroids to the global centroids using mpi
global_centroids = np.zeros((4, 2))
global_counts = np.zeros(4)
for i in range(4):
    comm.Reduce(local_centroids[i], global_centroids[i], op=MPI.SUM, root=0)
    comm.Reduce(counts[i], counts, op=MPI.SUM, root=0)
#Divide the global centroids by the number of assigned points to get the avg centroids
for i in range(4):
```

```
        if global_counts[i] > 0:
            global_centroids /= global_counts[i]

    #print the global cenroids from the coordinator(rank 0)
    if rank == 0:
        print("Global centroids:", global_centroids)
```

Kmeans.py

```
import pandas as pd
from mpi4py import MPI
import numpy as np

# Define colors
RED = "\033[31m"
GREEN = "\033[32m"
YELLOW = "\033[33m"
RESET = "\033[0m"

#initialize communicator , rank to get rank of current processes and size for total number of processes
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()


#load data from cluster_data csv on rank 0
if rank == 0:
    data = pd.read_csv("cluster_data.csv", header = 0, index_col=0)
else:
    # initialize data variable to None
    data = None

# Flatten data array & scatter the data evenly on all ranks
flat_data = np. array([]) if rank !=0 else data.values.flatten()
_data = np.empty(len(flat_data) // size, dtype=np.float64)
comm.Scatter(flat_data, _data , root=0)


k = 4 #number of clusters

 # initialize centroids randomly on rank 0
if rank == 0:
    centroids = np.random.rand(k, 2)
else:
    centroids = None

#broadcast centroids to all ranks
centroids = comm.bcast(centroids, root=0)

#print _data on each rank
print(f"{RED}Rank {rank} {RESET}")
print(f"{GREEN}centroids:{centroids} {RESET}")
print(f"{YELLOW}local data: {_data} {RESET}")


# compute the distances between each data point and the centroids
distances = np.zeros((_data.shape[0], k))
for i in range(k):
    distances[:, i] = np.linalg.norm(_data - centroids[i], axis=1)

#collect distances computed by all ranks onto each rank
all_distances = comm.allgather(distances)

#flatten and transpose all_distances into shape (n, k) where n is the total number of
```

```
    data points
    all_distances = np.concatenate(all_distances, axis=0)
    all_distances = np.transpose(all_distances)

    #print all_distances on rank 0
    if rank == 0:
        print(f"{RED}all distances{RESET}: {GREEN}{all_distances} {RESET}")
```