Data Generation & Clustering

**Imports**

import matplotlib.pyplot as plt

import numpy as np

import random as r

import math as m

**Data Generation**

**Deterministic Data Generation**

def determDataGen(numClusters, centX, centY, clustSize, clustRad):

    # cluster x and y coordinates

    px = []

    py = []

    for j in range(numClusters):

        # cluster centers

        cx = centX[j]

        cy = centY[j]

        ang = 0

        radInc = clustRad / 5

        pointsPerRad = 4

        angInc = 360 / pointsPerRad

        for k in range(clustSize):

            myX = cx + clustRad \* np.cos(np.deg2rad(ang))

            myY = cy + clustRad \* np.sin(np.deg2rad(ang))

            # append new cluster point

            px.append(myX)

            py.append(myY)

            ang = ang + angInc

            if(ang > 360):

                ang = 0

                clustRad += radInc

                pointsPerRad \*= 2

                angInc = 360 / pointsPerRad

            #

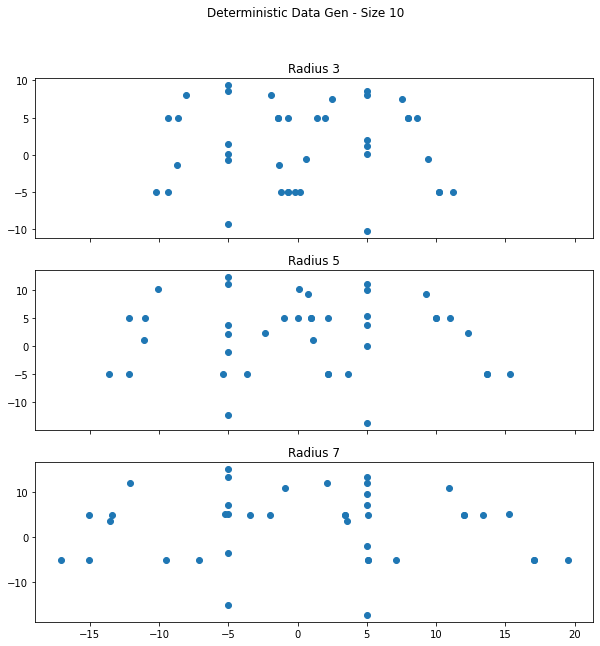
        #

    #

    return px, py

# determDataGen

Calendar

Description automatically generated

A picture containing shape

Description automatically generated

***Non-Deterministic Data Generation***

def nonDetermDataGen(numClusters, centX, centY, clustSize, clustRad):

    # cluster x and y coordinates

    px = []

    py = []

    for clust in range(numClusters):

        # cluster centers

        cx = centX[clust]

        cy = centY[clust]

        for clust\_pnt in range(clustSize):

            angD = r.randint(0,359)

            magnitude = r.random() \* clustRad

            myX = cx + magnitude \* np.cos(np.deg2rad(angD))

            myY = cy + magnitude \* np.sin(np.deg2rad(angD))

            # append new cluster point

            px.append(myX)

            py.append(myY)

        #

    #

    return px, py

# nonDetermDataGenA picture containing text, light, different, wire

Description automatically generatedCalendar

Description automatically generated

A picture containing qr code

Description automatically generated

***Function Calls to Generate Plots:***

pop\_size = [10, 30, 100]

rad\_size = [3, 5, 7]

determData = {}

nonDetermData = {}

for ind\_pop, pop in enumerate(pop\_size):

    det\_fig, det\_axs = plt.subplots(len(pop\_size), sharex = True, figsize = (10, 10))

    det\_fig.suptitle(f'Deterministic Data Gen - Size {pop}')

    nonDet\_fig, nonDet\_axs = plt.subplots(len(pop\_size), sharex = True, figsize = (10, 10))

    nonDet\_fig.suptitle(f'Non Deterministic Data Gen - Size {pop}')

    for ind\_rad, rad in enumerate(rad\_size):

        key = f's{pop}\_r{rad}'

        det\_data\_gen = determDataGen(4, [5, -5, -5, 5], [5, 5, -5, -5], pop, rad)

        det\_axs[ind\_rad].scatter(det\_data\_gen[0], det\_data\_gen[1])

        det\_axs[ind\_rad].set\_title(f'Radius {rad}')

        determData[key] = det\_data\_gen

        nonDet\_data\_gen = nonDetermDataGen(4, [5, -5, -5, 5], [5, 5, -5, -5], pop, rad)

        nonDet\_axs[ind\_rad].scatter(nonDet\_data\_gen[0], nonDet\_data\_gen[1])

        nonDet\_axs[ind\_rad].set\_title(f'Radius {rad}')

        nonDetermData[key] = nonDet\_data\_gen

    #

#

**Clustering**

***Ad Hoc Clustering***

def adHocClustering(x, y, maxDist):

    nClusts = 0 # number of clusters

    clusters = [] # the clusters

    pntsPerCluster = [] # number of pnts per cluster

    for j in range(len(x)): # loop through each data pnt

        pnt = [x[j], y[j]] # for convenience

        if(nClusts == 0): # no clusters?

            clusters.append(pnt) # this is the first pnt of the first cluster

            nClusts = nClusts + 1 # increment clusters count and...

            pntsPerCluster.append(1) # create item to increment pnts for that cluster

        else: # find closest cluster

            dists = [m.dist(pnt, cluster) for cluster in clusters]

            closeDists = min(dists) # closest cluster

            closeIndex = dists.index(min(dists)) # index of the closest cluster

            if(closeDists < maxDist): # close enough

                # most and rest counts for merging

                most = pntsPerCluster[closeIndex] / (pntsPerCluster[closeIndex] + 1)

                rest = 1 / (pntsPerCluster[closeIndex] + 1)

                # merge centroid

                clusters[closeIndex] = [((clusters[closeIndex][0] \* most) + (pnt[0] \* rest)), ((clusters[closeIndex][1] \* most) + (pnt[1] \* rest))]

                pntsPerCluster[closeIndex] = pntsPerCluster[closeIndex] + 1 # increment pntsPerCluster

            else: # new cluster

                clusters.append(pnt) # first pnt of a new cluster

                nClusts = nClusts + 1 # increment clusters and...

                pntsPerCluster.append(1) # add new item to increment points for that cluster

            #

        #

    #

    return nClusts, clusters, pntsPerCluster

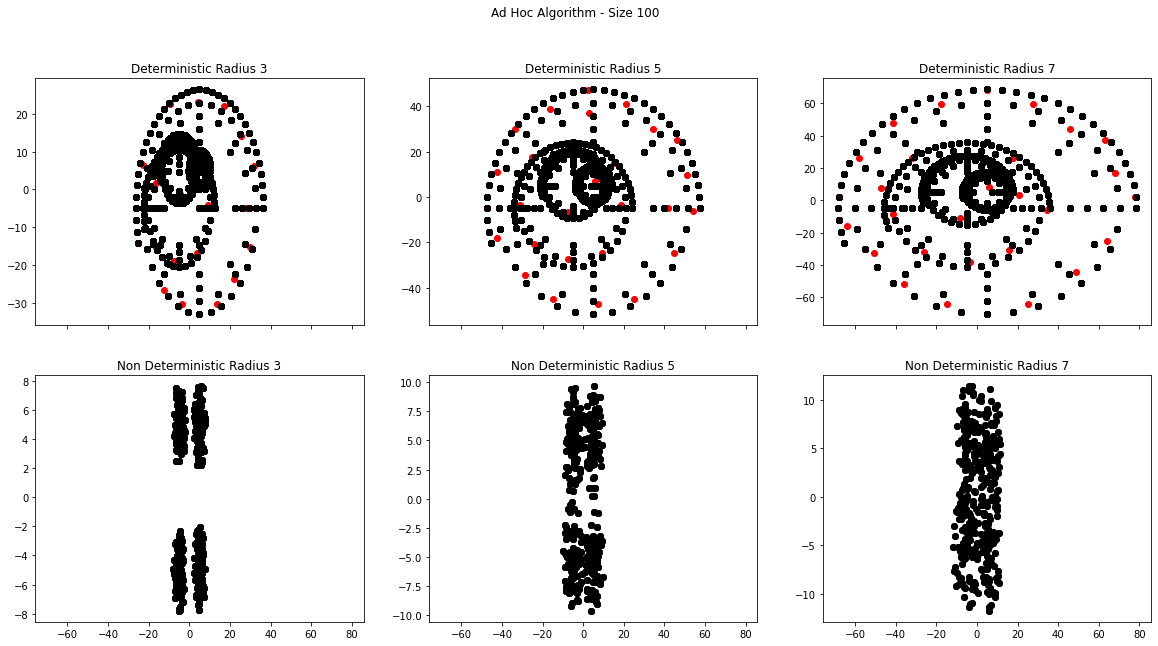
# adHocClustering

Calendar

Description automatically generated

A picture containing text, clipart, vector graphics

Description automatically generated



***Function Calls to Generate Plots:***

for ind\_pop, pop in enumerate(pop\_size):

    fig\_10, axs\_10 = plt.subplots(2, 3, sharex = True, figsize = (20, 10))

    fig\_10.suptitle(f'Ad Hoc Algorithm - Size {pop}')

    max\_dist = 8

    for ind\_rad, rad in enumerate(rad\_size):

        nClusts\_determ, clusters\_determ, pntsPerCluster\_determ = adHocClustering(determData[f's{pop}\_r{rad}'][0], determData[f's{pop}\_r{rad}'][1], max\_dist)

        for j in range(nClusts\_determ):

            axs\_10[0][ind\_rad].scatter(clusters\_determ[j][0], clusters\_determ[j][1], color = 'r')

            axs\_10[0][ind\_rad].scatter(determData[f's{pop}\_r{rad}'][0], determData[f's{pop}\_r{rad}'][1], color = 'k')

        #

        axs\_10[0][ind\_rad].set\_title(f'Deterministic Radius {rad}')

        nClusts\_nonDeterm, clusters\_nonDeterm, pntsPerCluster\_nonDeterm = adHocClustering(nonDetermData[f's{pop}\_r{rad}'][0], nonDetermData[f's{pop}\_r{rad}'][1], max\_dist)

        for j in range(nClusts\_nonDeterm):

            axs\_10[1][ind\_rad].scatter(clusters\_nonDeterm[j][0], clusters\_nonDeterm[j][1], color = 'r')

            axs\_10[1][ind\_rad].scatter(nonDetermData[f's{pop}\_r{rad}'][0], nonDetermData[f's{pop}\_r{rad}'][1], color = 'k')

        #

        axs\_10[1][ind\_rad].set\_title(f'Non Deterministic Radius {rad}')

        max\_dist += 4

    #

#

***K Means Clustering***

def kMeans(x, y, k):

    centroids\_x = []

    centroids\_y = []

    # choose k random points >> new centers

    for i in range(k):

        rndm\_indx = r.randint(0, len(x)-1)

        centroids\_x.append(x[rndm\_indx])

        centroids\_y.append(y[rndm\_indx])

    #

    clusters\_x = {}

    clusters\_y = {}

    for i in range(100):

        clusters\_x = {}

        clusters\_y = {}

        # assign data to clusters

        for centroid\_indx in range(len(centroids\_x)):

            pnts\_x = [] # holds the list of points that match the current cluster

            pnts\_y = []

            # loop through the points

            for pnt\_indx in range(len(x)):

                # find dist from centers

                dist = [m.dist([centroids\_x[j], centroids\_y[j]], [x[pnt\_indx], y[pnt\_indx]]) for j in range(len(centroids\_x))]

                # if the min(dist) is this centroid assign it to that cluster

                if centroid\_indx == dist.index(min(dist)):

                    pnts\_x.append(x[pnt\_indx])

                    pnts\_y.append(y[pnt\_indx])

                #

                # add list of cluster pnts to the clusters dict

                clusters\_x[centroid\_indx] = pnts\_x

                clusters\_y[centroid\_indx] = pnts\_y

            #

        #

        for key in clusters\_x:

            avg\_x = np.mean(clusters\_x[key])

            avg\_y = np.mean(clusters\_y[key])

            centroids\_x[key] = avg\_x

            centroids\_y[key] = avg\_y

        #

    #

    return centroids\_x, centroids\_y, clusters\_x, clusters\_y

#

Calendar

Description automatically generated

A picture containing calendar

Description automatically generated

Logo

Description automatically generated

***Function Calls to Generate Plots:***

for ind\_pop, pop in enumerate(pop\_size):

    fig, axs = plt.subplots(2, 3, sharex = True, figsize = (20, 10))

    fig.suptitle(f'K Means Clustering - Size {pop}')

    for ind\_rad, rad in enumerate(rad\_size):

        centersX\_determ, centersY\_determ, clustX\_determ, clustY\_determ = kMeans(determData[f's{pop}\_r{rad}'][0], determData[f's{pop}\_r{rad}'][1], 4)

        axs[0][ind\_rad].scatter(centersX\_determ, centersY\_determ, color = 'k')

        for j in range(len(clustX\_determ)):

            axs[0][ind\_rad].scatter(clustX\_determ[j], clustY\_determ[j])

        axs[0][ind\_rad].set\_title(f'Deterministic Radius {rad}')

        centersX\_nonDeterm, centersY\_nonDeterm, clustX\_nonDeterm, clustY\_nonDeterm = kMeans(nonDetermData[f's{pop}\_r{rad}'][0], nonDetermData[f's{pop}\_r{rad}'][1], 4)

        axs[1][ind\_rad].scatter(centersX\_nonDeterm, centersY\_nonDeterm, color = 'k')

        for j in range(len(clustX\_nonDeterm)):

            axs[1][ind\_rad].scatter(clustX\_nonDeterm[j], clustY\_nonDeterm[j])

        axs[1][ind\_rad].set\_title(f'Deterministic Radius {rad}')

    #

#