MIDTERM WORKING QUESTIONS

1) Assume that we have three points (i.e. P1, P2, and P3) with x, y, and z coordinates. Write a Python program to determine whether those points lie in the same surface or not. Your program must perform the following steps:

i) The program must include **main** function. In the main function, define three **empty** lists for points. Then, call **initalizeCoordinates** function.

ii) The program must include **initalizeCoordinates** function that receives the lists. Coordinates must be in interval [7, 15]. In the main function, print the coordinates.

iii) The program must include **calculateNormals** function that receives the lists and returns the normal vector of those points with given formulas:

$$n_x = \left(p2_y - p1_y\right) * \left(p3_z - p1_z\right) - \left(p2_z - p1_z\right) * \left(p3_y - p1_y\right)$$
$$n_y = \left(p2_z - p1_z\right) * \left(p3_x - p1_x\right) - \left(p2_x - p1_x\right) * \left(p3_z - p1_z\right)$$
$$n_z = \left(p2_x - p1_x\right) * \left(p3_y - p1_y\right) - \left(p2_y - p1_y\right) * \left(p3_x - p1_x\right)$$

iv) In the main function, call **calculateNormals** function and print the normal vector.

v) The program must include **calculateAngles** function that receives the normal vector and returns an angle vector with given formulas:

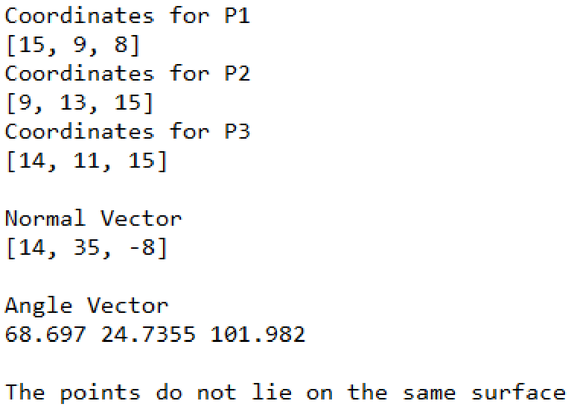
$$\text{angleX} = \cos^{-1}\left(\frac{n_x}{\sqrt[2]{{n_x}^2 + {n_y}^2 + {n_z}^2}}\right) * 180/M\_PI$$
$$\text{angleY} = \cos^{-1}\left(\frac{n_y}{\sqrt[2]{{n_x}^2 + {n_y}^2 + {n_z}^2}}\right) * 180/M\_PI$$
$$\text{angleZ} = \cos^{-1}\left(\frac{n_z}{\sqrt[2]{{n_x}^2 + {n_y}^2 + {n_z}^2}}\right) * 180/M\_PI$$

vi) In the main function, call **calculateAngles** function and print the angle vector.

vii) The program must include **isSameSurface** function that the angle vector and a threshold value in terms of degree, then returns true if all criteria given below are met. Otherwise return false. Test your program using 5 degrees as a threshold value.

$$|angleX - angleY| < thres,$$
$$|angleX - angleZ| < thres,$$
$$|angleZ - angleY| < thres$$

viii) In the main function, call **isSameSurface** function and print the result.

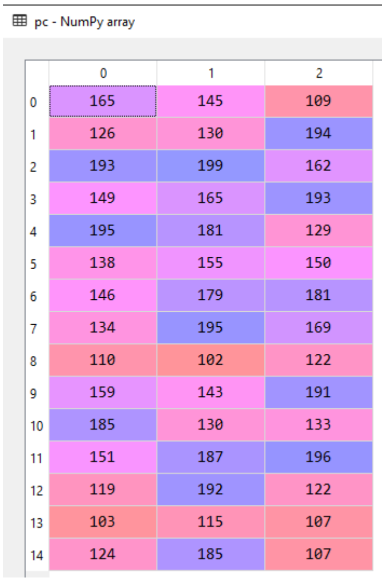


```
Coordinates for P1
[15, 9, 8]
Coordinates for P2
[9, 13, 15]
Coordinates for P3
[14, 11, 15]

Normal Vector
[14, 35, -8]

Angle Vector
68.697 24.7355 101.982

The points do not lie on the same surface
```

2) In this question, you will write a python program to determine inlier and outlier points depending on a distance threshold (thresh) and the number of neighbors (K) in that threshold. The program must include the following steps:

i) The program must include ***getParameters*** function. In the function, prompt the number of points (NOP), thresh and the K and return these values**.**

ii) The program must include ***generatePointCloud*** function. The function receives the NOP and generates points with x, y, and z coordinates, which will be integer values between 100 and 200 according to the NOP. Then, store these values in a **numpy array** and returns the numpy array. An example output for 15 points is given in Figure 1.

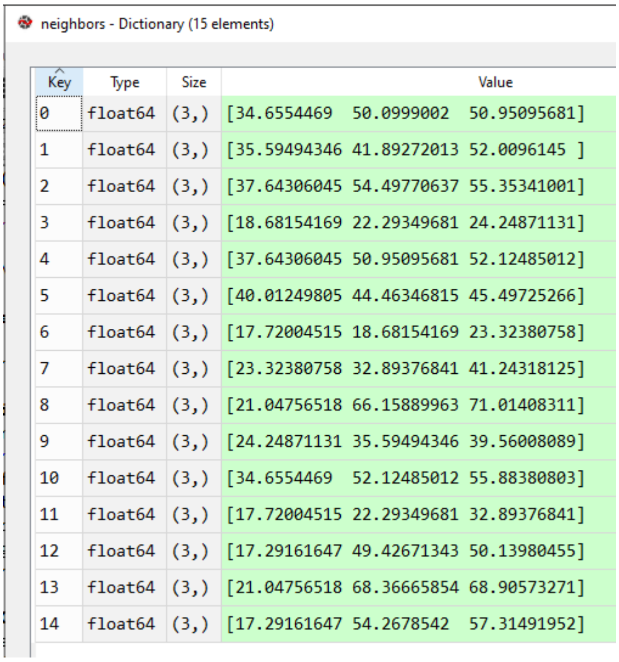


pc - NumPy array

	0	1	2
0	165	145	109
1	126	130	194
2	193	199	162
3	149	165	193
4	195	181	129
5	138	155	150
6	146	179	181
7	134	195	169
8	110	102	122
9	159	143	191
10	185	130	133
11	151	187	196
12	119	192	122
13	103	115	107
14	124	185	107

Fig. 1 An example numpy array for 15 points

iii) The program must include ***findKNeigbors*** function. The function receives the numpy array and the K. In the function, first define an empty dictionary (i.e neighbors). Then, for each point in the numpy array, calculate the Euclidean distance between the current point and other points. Sort the distances in ascending order and assign K-nearest distances, excluding itself, into dictionary as values and the key of these values must be indices of the points in numpy array. The function must return the dictionary. **In the function, use numpy library.** An example dictionary with K is 3 is given in Figure 2.

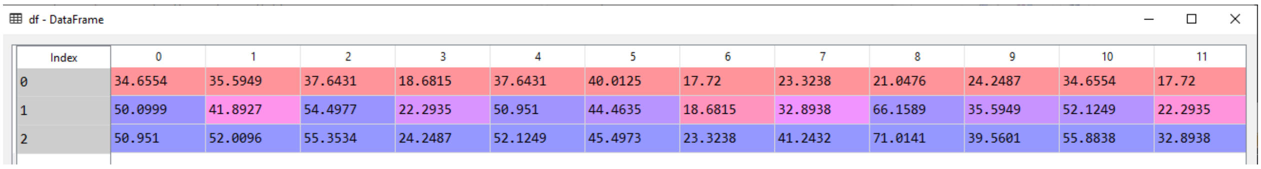


neighbors - Dictionary (15 elements)

Key	Type	Size	Value
0	float64	(3,)	[34.6554469 50.0999002 50.95095681]
1	float64	(3,)	[35.59494346 41.89272013 52.0096145]
2	float64	(3,)	[37.64306045 54.49770637 55.35341001]
3	float64	(3,)	[18.68154169 22.29349681 24.24871131]
4	float64	(3,)	[37.64306045 50.95095681 52.12485012]
5	float64	(3,)	[40.01249805 44.46346815 45.49725266]
6	float64	(3,)	[17.72004515 18.68154169 23.32380758]
7	float64	(3,)	[23.32380758 32.89376841 41.24318125]
8	float64	(3,)	[21.04756518 66.15889963 71.01408311]
9	float64	(3,)	[24.24871131 35.59494346 39.56008089]
10	float64	(3,)	[34.6554469 52.12485012 55.88380803]
11	float64	(3,)	[17.72004515 22.29349681 32.89376841]
12	float64	(3,)	[17.29161647 49.42671343 50.13980455]
13	float64	(3,)	[21.04756518 68.36665854 68.90573271]
14	float64	(3,)	[17.29161647 54.2678542 57.31491952]

Fig. 2 An example dictionary.

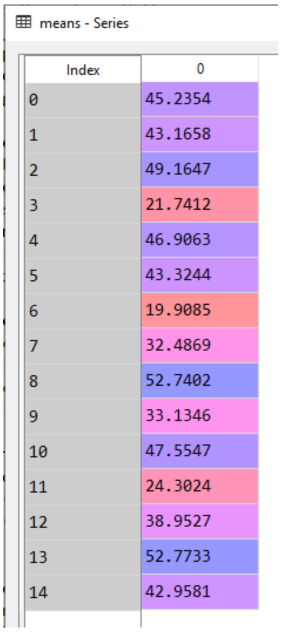
iv) The program must include ***filterPC*** function. The function receives the numpy array, the dictionary and the thresh. In the function, first convert the dictionary to a DataFrame (Figure 3). **Notice that, the distances are now the columns of the DataFrame.** Then, calculate the mean of each point, in other words columns. The result of mean calculation must be a Series which is given in Figure 4. At that stage, you have to produce a numpy array to mask the mean values according to the distance threshold. **To do this, use pandas library to filter the points in the point cloud. You can use values data attribute of the Series.** Assume that the thresh is 40. If the mean value is greater than the thresh a False value must be assigned. Otherwise, a True value is assigned (Figure 5). The points corresponding to True values must assign a numpy array inlier and points corresponding to False values must assign a numpy array outlier. The function must return inlier and outlier numpy arrays.



df - DataFrame

Index	0	1	2	3	4	5	6	7	8	9	10	11
0	34.6554	35.5949	37.6431	18.6815	37.6431	40.0125	17.72	23.3238	21.0476	24.2487	34.6554	17.72
1	50.0999	41.8927	54.4977	22.2935	50.951	44.4635	18.6815	32.8938	66.1589	35.5949	52.1249	22.2935
2	50.951	52.0096	55.3534	24.2487	52.1249	45.4973	23.3238	41.2432	71.0141	39.5601	55.8838	32.8938

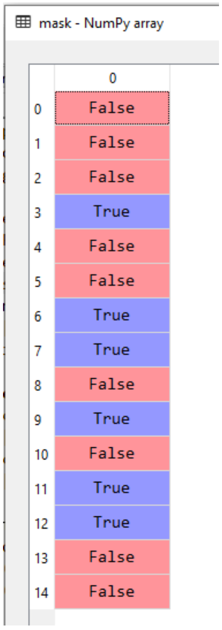
Fig. 3 An example DataFrame corresponding to the dictionary given in Fig. 2.



means - Series

Index	0
0	45.2354
1	43.1658
2	49.1647
3	21.7412
4	46.9063
5	43.3244
6	19.9085
7	32.4869
8	52.7402
9	33.1346
10	47.5547
11	24.3024
12	38.9527
13	52.7733
14	42.9581

Fig. 4 An example Series corresponding to the mean calculation of the DataFrame given in Fig. 3.



mask - NumPy array

	0
0	False
1	False
2	False
3	True
4	False
5	False
6	True
7	True
8	False
9	True
10	False
11	True
12	True
13	False
14	False

Fig. 5 An example mask for means values according to distance threshold which given in Fig. 4

v) Write inlier and outlier numpy arrays to files with .csv extension. The filenames must be point_cloud_inlier_yourname_yoursurname.csv and point_cloud_outlier_yourname_yoursurname.csv. The values in the files must be separated with the minus character ("-").

vi) The program must include ***plotFilteredPoints*** function. The function receives the filenames and plots the inlier and outlier points via matplotlib.pyplot library. In the function, read the files to DataFrames. Change columns headers with i_x, i_y, and i_z for inliers and o_x, o_y, and o_z for outliers. Then, add the following statements to your function. Use **scatter** method to plot the points. An example output is given Figure 6. In the figure, inlier points and outlier points are shown with red and green colors, respectively.

```
from mpl_toolkits.mplot3d import Axes3D   // add the lib to your code
fig = plt.figure()                        // add the statement to your code
ax = fig.add_subplot(111, projection='3d') // add the statement to your code
```

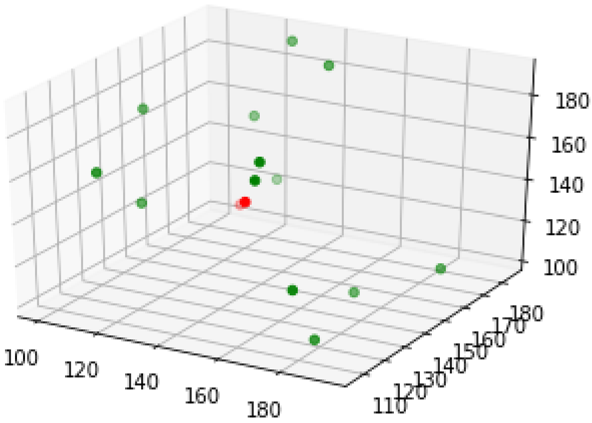


Fig. 6 An example output for plotFilteredPoints function

vii) The program must include ***main*** function. In the function, call the functions ***getParameters, generatePointCloud, findKNeigbors, filterPC, plotFilteredPoints*** and writing to files steps.