Ring-like Clustering on Noisy Data

Luis Mellado Díaz
Artificial Intelligence Department
Escuela Técnica Superior de Ingeniería Informática
Seville, Spain
melladodíazluis@gmail.com

Jose María Moyano Murillo
Artificial Intelligence Department
Escuela Técnica Superior de Ingeniería Informática
Seville, Spain
jmoyano1@us.es

Agustin Riscos Núñez
Artificial Intelligence Department
Escuela Técnica Superior de Ingeniería Informática
Seville, Spain
ariscosn@us.es

***Abstract*— This study addresses a simplified version of a problem encountered in the LHCb experiment at CERN involving the RICH (Ring Imaging Cherenkov) detectors. These detectors aim to identify circular or elliptical patterns resulting from Cherenkov radiation. We abstract the problem to finding optimal ring configurations for a given dataset within a defined spatial range, accounting for noise. Our goal is to minimize error in fitting the data.**

Keywords—cluster, GUI, K-means, iteration, noise

I. Introduction

The problem of fitting a circle to a collection of points in the plane holds significance in various domains. The main goal of this study is to design, implement, understand and analyze an algorithm that searches for circumference and takes uncertainty explicitly into account.

A. Description of the Problem

In general, suppose that we have a collection of points of n $\geqslant$ 3 points in a 100x100 2D-space labeled (x1, y1), (x2, y2), . . . , (xn, yn).
Our basic problem is to find a circle that best represents the data in some sense. With the circle described by $(x - a)^2 + (y - b)^2 = r^2$, we need to determine values for the center (a,b) and the radius r for the best fitting circle.

B. Proposed Solution, Clustering.

Clustering is an unsupervised learning technique used to group data points into classes (clusters) based on similarity. The partition into subsets aims to discover patterns or hidden insights previouly unknown.

Each cluster will be represented by a circumference defined by a center (a,b) and a radius r, while each point (x,y) is associated with all clusters but with a different membership degree.

II. Structure of the Work

In this report, we commence by describing the general operational framework of the algorithm, followed by an in-depth exploration of its insights, including initialization, halting conditions, center recalculation, and result representation.

Subsequently, we explore the methodologies for experiment generation and evaluation. We then see the algorithm's performance across various scenarios, and discuss its reliability upon initialization and noise levels.

Lastly, we elucidate the design decisions made and provide a comprehensive reference section, including figures and additional content, located at the conclusion of this work.

III. Ring-Like Clustering

A. The algorithm

The implemented algorithm is inspired in the K-means algorithm. Given the number of clusters K and given a distance function (e.g. Euclidean distance), group the points by proximity to each cluster. Next, every cluster center is updated using the mean of its members.

Notice that in our case, clusters are circumferences. The distance function is [1] the euclidean distance from a point P to a circumference c having center O and radius r:

$$d(P,c) = | d(P,O) - r |$$

And instead of using the mean of the points to recompute the cluster's centers we will use the [2] Least-Squares Regression.

Thus, the general scheme of the algorithm is the following:

1. Cluster Initialization
2. Main Loop (until halting condition)
   - 2.1 Compute membership degree of each point (Euclidean distance)
   - 2.2 Update the center and radious of each cluster (Least-Squares Regression)
3. Assign to each point a cluster and return final solution

B. Cluster Initialization

Regarding the initialization of the cluster the designed algorithm provides 3 different options:

1. Predefinded values (BASIC): Selecting this option centers the clusters in any point of a predefined list. That list include points like: (25, 25), (50, 25), (75, 25) … Radius is always 10.

2. Random (ADVANCED): The K clusters will be centered at random points inside the 100x100 square 2d-space with a random radius.
3. Heuristic (ADVANCED): Points are first sorted and divided in K groups. From each group 3 points are selected. The 3 points form a triangle, [5][6] which circumcenter will be the initial center of that cluster.

C. *Halting condition*

The algorithm counts with 2 different halting conditions:

1. Number of iterations (BASIC): The algorithm stops when it reaches the number of iterations provided by the user. By default is set to 10 iterations.
2. Convergence rate (ADVANCED): Optionally a convergence rate can be set. When the cluster variation from one iteration to the other is smaller than the convergence rate the algorithm halts.

D. *Updating the Clusters: Least-Squares Circle*

[2] In the situation in which you have data points (x_i,y_i) that are distributed in a ring-shape on a 2-dimensional space the least-squares regresion can be used to determine the equation of a circle that will best fit with the available data points.

[3] The idea is to define a circumference C with center (k,m) with radious r for a set of points S such that the sum of the distance from every point in S to C is the minimum possible. Therefore the function to minimize is the following:

$$F(k, m, r) = \sum [(x_i - k)^2 + (y_i - m)^2 - r^2]^2$$

However, it is convinient to linearize the model in order to solve the least-squares problem using matrices. As such, we have the following:

$$(x - k)^2 + (y - m)^2 = r^2$$

$$x^2 + y^2 = 2kx + 2my + r^2 - k^2 - m^2$$

$$x^2 + y^2 = Ax + By + C$$

[4] Now we are left with the task of solving a system of 3 equations.

$$\begin{bmatrix} \sum x_i^2 & \sum x_i y_i & \sum x_i \\ \sum x_i y_i & \sum y_i^2 & \sum y_i \\ \sum x_i & \sum y_i & n \end{bmatrix} \begin{bmatrix} A \\ B \\ C \end{bmatrix} = \begin{bmatrix} \sum x_i(x_i^2 + y_i^2) \\ \sum y_i(x_i^2 + y_i^2) \\ \sum x_i^2 + y_i^2 \end{bmatrix}$$

Fig 1. Matricial form of the least sqares system of equations.

Finally we can deduce the center (k,m) and radius r of the best fist circle C returning the variable change:

$$k = A/2$$

$$m = B/2$$

$$r = (\sqrt{4C + A^2 + B^2})/2$$

E. *Updating the Clusters (Weighted version)*

A different version of the Least-Squares regression method has been developed specifically for obtaining the best fit circle not only considering the points belonging to a cluster but also incorporating the rest of points, albeit with reduced significance.

It must be stated that in the Least-Sqares regression there is no such a thing of a point with less importance. Either it is consider or it is not consider. Then the problem on how to weight the points arises.

The approach to solve the problem is the following: All points are considered to build de best fit circle but with a probability inversely proportional to the distance from the point to the cluster. That is, points that are close to the cluster are very likely to be considered while points that are far away from the cluster are probably neglected.

In order to compute the likelyhood of a point participating in the construction of the cluster a probability function p(d(a,c)) is used. Where d(a,c) is a function that returns the normalized distance between a point "a" and a cluster "c".

$$d(a,c) = \text{normalize}(\text{dist}(a,c))$$

$$p(d(a,c)) = e^{-5d}$$

Once the list of important points is computed, the normal Least-Squares method is applied over it.

F. *Presentation of the solution*

When the algorithm halts it provides the center and radius of each cluster and the classification of each point. The solution comes with a graphical representation that shows every cluster and its members with a particular color.

This solution can be processed to remove what might be noise. For that two optional processing tools are offered:

1. Removal of noise points: Removes every point that is further from its cluster than a given distance.
2. Removal of noise clusters: Remove clusters that are "too similar" to others according to a given equivalence rate based on the radius similarity and the center proximity. The equivalence rate ranges from 0.00 to 1.00. It is recommended to use values from 0.90 to 1.00, otherwise you will probably eliminate clusters that are not noise but just close to another.

IV. Experiment

For the purpose of facilitating the study and produce a high number of experiments a generation function was designed. This tool receives the following inputs:

1. Maximum number of points per circular pattern: Integer number that limits how many points any circular pattern can have.
2. Number of circular patterns: Input here an integer number representing how many circular patterns are going to be generated.
3. Maximum uncertainty: A float number that represent the distance a point can deviate from the circular pattern. In other words, the maximum allowed noise.

4. Experiment title: A string that will serve as a name to identify the experiment.

The tool outputs a .csv file containing a series of randomly generated points that follow the restrictions imposed by inputs 1, 2 and 3. This points compose an experiment over which you can run the algorithm and evaluate the results.

A. *Running the Algorithm Over the Experiment*

To run the algorithm over a generated or handcrafted experiment you need to provide:

- The title of the experiment, in order to scan the points from the .csv file.
- Number of clusters K.
- Initialization type.

Optionally the halting conditions and the result processing parameter can be introduced. Those are:

- Maximum number of iterations.
- Maximum convergence rate.
- Maximum allowed noise.
- Maximum cluster equivalence rate.

What these parameter are and how they relate to the different stages of the algorithm is explained in depth in section II.

When the algorithm is run over an experiment a .csv file is generated so the results of the experiment can be revisited. This time the .csv do not only contain the x and y coordinates of a series of poins but also the cluster they belong to (represented by a color).

B. *Evaluating new points*

Once the results of an experiment have been produced it is natural to introduce new points and evaluate them to know to which cluster or class they belong to.

To do so, we can use the "evaluate new points" tool that was implemented. It is necessary to provide:

1. The experiment results title. Necessary to scan the experiment resuls and load the points and clusters. Please do not confuse a experiment file with a experiment results file.
2. A list of the new points to evaluate in the format [(x1,y1), (x2,y2), … , (xn,yn)]

Finally, the classification of the new points will be shown. When you evaluate new points the output of the evaluation is not stored in the experiment results file.

V. PERFORMANCE

In this section wi will analyze the performance and reliabilty of the algorithm in 4 different experiments. Each experiment is firstly described in detail and then studied. The study of an experiment consist in running the algorithm over the experiment varying the initialization and the cluster computation method in order to find the most optimal and reliable configuration.

If a methods or configuration involves randomness the expirement will be run 5 times and an average performance will be concluded.

It must be clarified that whenever I refer to "low noise" it means that the circular pattern is clear enough to be distinguished by a human. When I refer to "high noise" it means that the circular pattern is hardly distinguishable.

Noise can be mesured in how far a point can deviate from the circular trayectory (5 units of noise means that points can at most be 5 units of distance from the intended pattern).

A. *Ideal Experiment*

This experiment counts with 3 circular patterns with low noise. Patterns do not overlap and are placed next to each other.

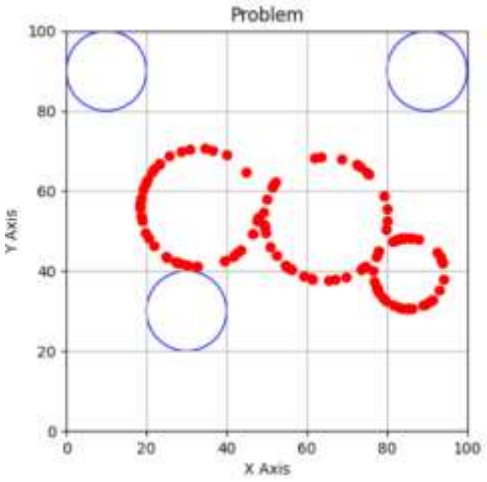

Fig 2. Problem A, the ideal problem.

Predefined initialization behaves poorly, on average, it takes 6 iterations to find only one of the patterns after halting. Random intialization does not work much better, however, in some of the experiments it has found 2 of the 3 circumferences.

On the other hand, when the algorithm is run with heuristic, it is common that it finds the 3 circular patterns perfectly. On average it finds 80% of the patterns (between 2 and 3 circumferences) on 6 iterations.

	A	B	C	D	E
1	Experiment	Initialization	Success Rate	Iterations	Performance
2	A	Predefined	0,33	10,00	0,17
3	A	Predefined	0,33	6,00	0,39
4	A	Predefined	0,33	5,00	0,44
5	A	Predefined	0,00	5,00	0,28
6	A	Predefined	0,33	4,00	0,50
7	A	Predefined	0,264	6	0,35
8	A	Random	0,00	5,00	0,28
9	A	Random	0,33	7,00	0,33
10	A	Random	0,66	6,00	0,55
11	A	Random	0,66	9,00	0,39
12	A	Random	0,33	5,00	0,44
13	A	Random	0,396	6,4	0,40
14	A	Heuristic	0,66	5,00	0,61
15	A	Heuristic	1,00	8,00	0,61
16	A	Heuristic	1,00	6,00	0,72
17	A	Heuristic	1,00	8,00	0,61
18	A	Heuristic	0,33	5,00	0,44
19	A	Heuristic	0,796	6,4	0,60

Fig 2. Results of experiment A..

B. *Overlapping circular patterns*

This experiment is conformed by 3 circular patters with low noises. The circular patterns do overlap.

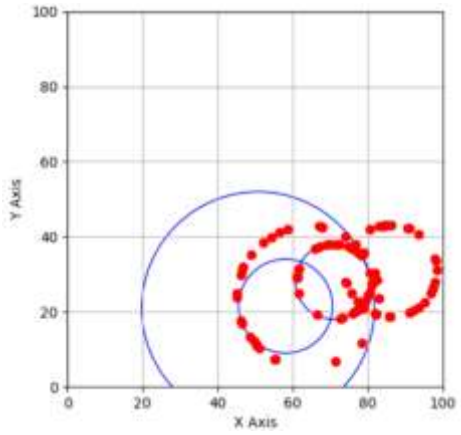

Fig 4. Problem B, Overlapping circular patterns.

Again, the behaviour of the predefined initialization is kind of poor. This time it takes about 6 iterations to find between 1 and 2 of the patterns. Random initialization is slightly better, it covered, at least, 2 of the circumferences in most of the experiments with a similar number of iterations.

As for heuristic initalization, we see almost the same performance than in the ideal experiment A. What is interesting is that it required les iterations on average (4 iterations instead of 6). This could be due to the fact that the circumferences are really close to each other.

	A	B	C	D	E
19	Experiment	Initialization	Success Rate	Iterations	Performance
20	B	Predefined	0,66	7,00	0,50
21	B	Predefined	0,33	6,00	0,39
22	B	Predefined	0,66	8,00	0,44
23	B	Predefined	0,00	5,00	0,28
24	B	Predefined	0,66	5,00	0,61
25	B	Predefined	0,462	6,2	0,44
26	B	Random	0,66	3,00	0,72
27	B	Random	0,33	9,00	0,22
28	B	Random	0,33	9,00	0,22
29	B	Random	0,66	5,00	0,61
30	B	Random	1,00	7,00	0,67
31	B	Random	0,596	6,6	0,49
32	B	Heuristic	0,66	4,00	0,66
33	B	Heuristic	1,00	4,00	0,83
34	B	Heuristic	0,66	4,00	0,66
35	B	Heuristic	1,00	4,00	0,83
36	B	Heuristic	0,66	4,00	0,66
37	B	Heuristic	0,796	4	0,73

Fig 5. Results of experiment B.

C. *High number of circular patterns*

In this case we count with 10 circular patterns with low noise. The patterns are arder to distinguish not becouse of noise but due to overlapping.

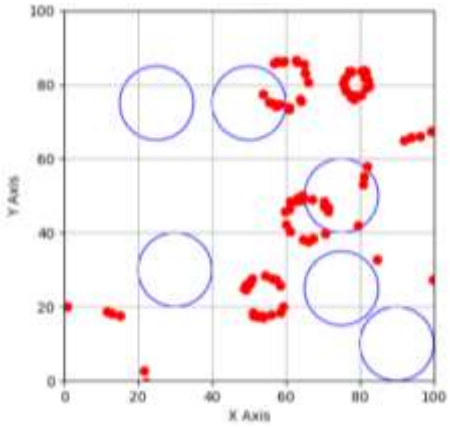

Fig 6. Problem C, High number of circular patterns.

Predefined initialization found on average 1 or none circumference in 4 iterations, showing the worst performance up to the moment. The solution that it provided did not covered the circumferences propperly. What you would see is how one of the estimations covered several real circumferences from the outside. It is also common that several clusters are discarded in the process due to beeing fruther than the rest from the action.

As for random initialization, you would see a very similar scenario. But this time, since the initialization is purely random, better solutions have been provided. On average it covered 27% of the circumferences within 5 iterations.

Heuristic does much better. It found 4 out of 6 circumferences perfectly in all the experiments. The other 2 circumferences were partially covered, It always took exactly 6 iterations. This is, on average, 66% of the patterns found in 6 iterations.

	A	B	C	D	E
37	Experiment	Initialization	Success Rate	Iterations	Performance
38	C	Predefined	0,17	6,00	0,31
39	C	Predefined	0,00	6,00	0,22
40	C	Predefined	0,17	8,00	0,20
41	C	Predefined	0,17	3,00	0,47
42	C	Predefined	0,33	4,00	0,50
43	C	Predefined	0,16733333	5,4	0,34
44	C	Random	0,17	5,00	0,36
45	C	Random	0,17	4,00	0,42
46	C	Random	0,17	5,00	0,36
47	C	Random	0,50	6,00	0,47
48	C	Random	0,33	5,00	0,44
49	C	Random	0,268	5	0,41
50	C	Heuristic	0,66	6,00	0,55
51	C	Heuristic	0,66	6,00	0,55
52	C	Heuristic	0,66	6,00	0,55
53	C	Heuristic	0,66	6,00	0,55
54	C	Heuristic	0,66	6,00	0,55
55	C	Heuristic	0,66	6	0,55

Fig 7. Results of experiment C.

D. *Low number of points*

Now there are 3 circular patterns with low noise. Patterns do not overlap and there is low noise. The circumferences are hard to spot since they are conformed by a lower number of points.

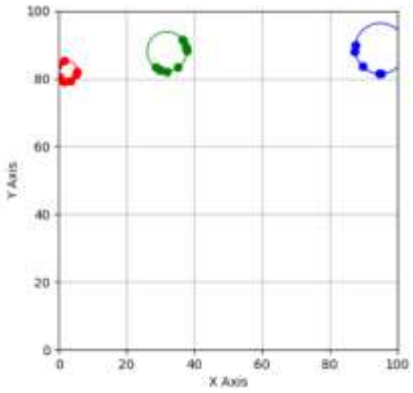

Fig 8. Problem D, Low number of points, solved.

Notice that, due to the lower number of poinnts, this time the algorithm fights against circumference arcs.

Predefined initialization does fairly well, finding at least one of the arcs and in some cases the three of them on 5 iterations average.

This is the first time we notieced random initialization doing worse than predefined. It found between 0 and 1 of the patterns on 4 iterations before halting.

The performance of the heuristic is excellent, providing a 100% success rate in less than 4 iterations. The lower number of points helps creating an initialization really close to the real solution.

	A	B	C	D	E
55	Experiment	Initialization	Success Rate	Iterations	Performance
56	D	Predefined	1,00	4,00	0,83
57	D	Predefined	0,33	3,00	0,55
58	D	Predefined	0,33	3,00	0,55
59	D	Predefined	0,33	5,00	0,44
60	D	Predefined	0,33	10,00	0,17
61	D	Predefined	0,464	5	0,51
62	D	Random	0,33	3,00	0,55
63	D	Random	0,33	4,00	0,50
64	D	Random	0,00	6,00	0,22
65	D	Random	0,33	4,00	0,50
66	D	Random	0,00	5,00	0,28
67	D	Random	0,198	4,4	0,41
68	D	Heuristic	1,00	3,00	0,89
69	D	Heuristic	1,00	3,00	0,89
70	D	Heuristic	1,00	3,00	0,89
71	D	Heuristic	1,00	3,00	0,89
72	D	Heuristic	1,00	3,00	0,89
73	D	Heuristic	1	3	0,89

Fig 9. Results of experiment D.

E. *Moderate noise*

Experiment E counts with 3 patterns with moderate noise (5-20 units). Points are scattered around but circular patterns are still distinguishable.

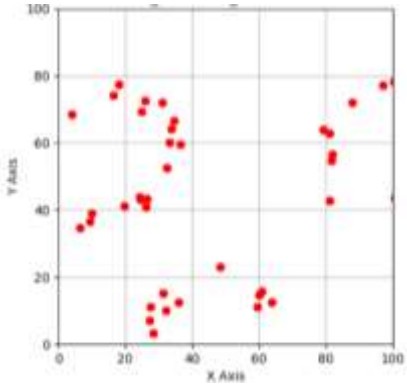

Fig 10. Problem E, moderate noise..

Predefined and random initialization perform really similar here. Finding on average between 30% and 40% of the circumferences on 6-7 iterations.

Heuristic initialization always found 2 of the patterns in 5 iterations average.

	A	B	C	D	E
73	Experiment	Initialization	Success Rate	Iterations	Performance
74	E	Predefined	0,66	5,00	0,61
75	E	Predefined	0,33	4,00	0,50
76	E	Predefined	0,33	5,00	0,44
77	E	Predefined	0,33	9,00	0,22
78	E	Predefined	0,00	10,00	0,00
79	E	Predefined	0,33	6,6	0,35
80	E	Random	0,33	9,00	0,22
81	E	Random	0,33	8,00	0,28
82	E	Random	0,66	5,00	0,61
83	E	Random	0,66	4,00	0,66
84	E	Random	0,00	5,00	0,28
85	E	Random	0,396	6,2	0,41
86	E	Heuristic	0,66	3,00	0,72
87	E	Heuristic	0,66	5,00	0,61
88	E	Heuristic	0,66	3,00	0,72
89	E	Heuristic	0,66	10,00	0,33
90	E	Heuristic	0,66	6,00	0,55
91	E	Heuristic	0,66	5,4	0,59

Fig 11.Results of experiment E.

F. *High noise*

Finally, we have an experiment with high noise (more than 20 units). Circular patterns are hardly distinguishable and we can only see a chaotic distribution of poins.

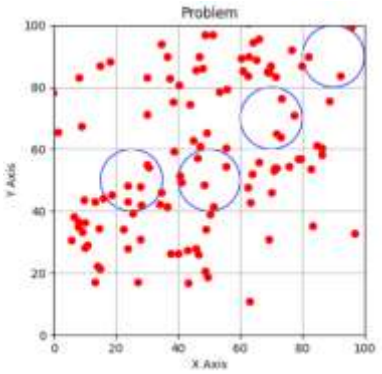

Fig 12. Problem F, High noise.

There is no a real way to measure how well the algorithm is doing since it is impossible to distinguish where the circumferences are. The algorithm just groups the scattered points in circular patterns and later removes the noise. See the following image:

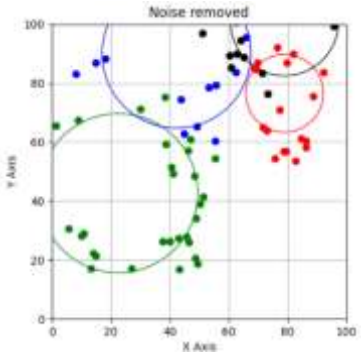

Fig 13.Solution of proble F after removing noise.

G. *Conclusion*

After performing all the experiments and collecting the date we found out that:

1. Predefined initialization found 33.75% of the patterns on 5.84 iterations before halting.
2. Random initialization found 38.08% of the patterns on 5.72 iterations before halting.
3. Heuristic initialization found 78.28% of the patterns on 4.96 iterations before halting.

Some interesting observations on the algorithm behaviour are:

1. The closer the circumferences the lower iterations the algorithm needs to converge and halt. Overlapping circumferences result in faster but not perfectly exact solutions.
2. The higher number of circumferences to find the slower the algorithm is, both in time and number of iterations. Here is where the algorithm has shown the worst precission.
3. The lower number of points the easier it is for the heuristic to create an initialization really close to the real solution.
4. Low noise and providing circumference arcs did not have a considerable negative repercussion it the algorithm.

The experimental results confirm that the choice of initialization strategy in k-means is crucial for its performance. Heuristic strategies that leverage prior knowledge of the dataset are significantly more effective in terms of accuracy and speed.

VI. Design Decissions

A. *Project Directory*

This section aims to explain how the folders are organized within the project and what you can expect to find in each one.

In the source file you can find the following folders:

src

1. **experiments:** Contains the necessary functions to generate experiments. Stores the points csv files.
   1.1 **data:** Contains the points csv files corresponding to the genereated experiments.
      1.1.1 **results:** Stores the points csv files corresponding to the produced results of running the algorithm over an experiment.
2. **maths:** Provides mathematical functions to compute distances, complex sums and the Least-Squares Regression.
3. **ring_clustering:** Contains the different stages of the main algorithm, that is: initialization, main loop, halting and post_processing.
4. **user_interface:** Here the functions to draw graphical representation of points an circles can be found. Also the general tkinter GUI is developed in this folder.
5. **utils:** other functions that did not fit in any of the previous folders. That includes functions to color points and circles and random point generation.

main.py

B. *Programming Language*

Python was the programming language chosen for this project due to its simplicity and readability, which facilitate quick development and comprehension of algorithms like K-means clustering. My familiarity with libraries such as numPy, math, random, tkinter… was also a key point in this decision.

Another reason to choose Python insteado of other popular languages like Java was my functional approach. Functional programming promotes the use of higher-order functions, enhancing code modularity and reusability.

REFERENCES

[1] "Shortest Distance Between a Point and a Circle," Varsity Tutors, Available: https://www.varsitytutors.com/hotmath/hotmath_help/topics/shortest-distance-between-a-point-and-a-circle.

[2] K. Jones, "Best Fit Circle," Ball State University, Available: https://www.cs.bsu.edu/homepages/kjones/kjones/circles.pdf

[3] "Best Fit Circle Least Squares Calculator," Good Calculators, Available: https://goodcalculators.com/best-fit-circle-least-squares-calculator/.

[4] "From Matrix to System of Equations," goodcssm.live, Available: https://goodcssm.live/product_details/29794835.html.

[5] "Circumcenter of the Triangle," Stack Overflow, Available: https://stackoverflow.com/questions/56224824/how-do-i-find-the-circumcenter-of-the-triangle-using-python-without-external-lib.

[6] "Circumcircle," Wikipedia, Available: https://en.wikipedia.org/wiki/Circumcircle.