**Report of TD 2 N-lateration**

Positioning Systems techniques and Applications Master 1,

Internet of Things

The University of Franche-Comté

### By: Uendi MUCA and Omar KHALIL

### Submit To: Dr. Phillipe CANALDA

### Date:25.03.2025

Table of Contents

1.Introduction to N-Lateration

2.Problem Introduction

3.Conceptualization of the Problem

4. Graphical 2D resolution of the N-Lateration problem

5.3D resolution of the N-Lateration problem

6.Architecture of program

7.Class Diagram

8.Data Structure

9.Pseudo-Code

10.Algorithm

11.Mathematical Calculations

12.Comparison of code results and manually computed results

13.Conclusions

# Introduction to N-Lateration

N-Lateration is a positioning method used to estimate the coordinates of an unknown point based on measured distances from some known reference points like GPS satellites or WiFi access points etc. Unlike trilateration, which operates in a two-dimensional space using three reference points to find a 2D coordinates , N-Lateration extends this principle into three-dimensional space, requiring at least four known reference locations (receivers) to determine the position of the device (emiter) accurately.It works by measuring the distance from an unknown position to at least N+1 reference points in an N-dimensional space.

The idea of N-Lateration is to find the coordinates of unknown point by solving some mathematical equations which form a system of nonlinear equations where each equation represent the distance between the object and reference point.

The mathematical equation is:

(x-xi)2+(y-yi)2+(z-zi)2=di2

Where :

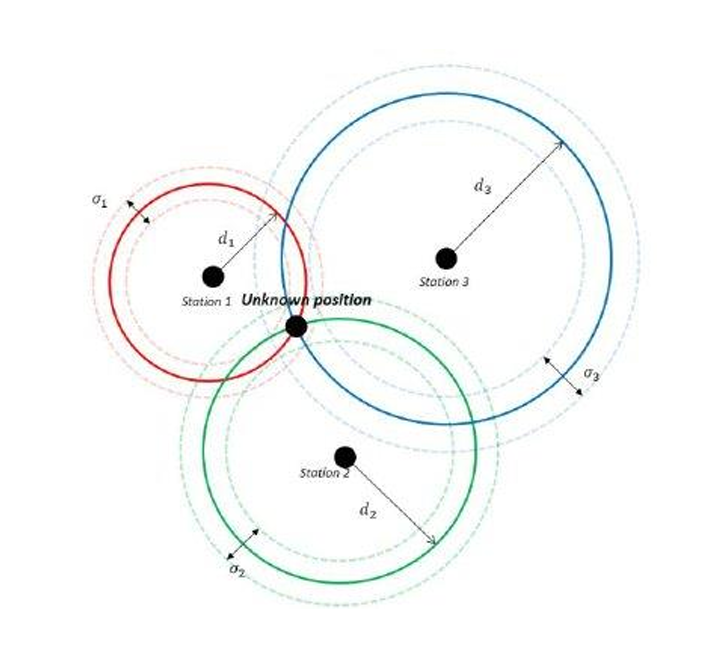
* (xi,yi,zi) are known coordinates of emitter Ei.
* di is the distance between the emitter and receiver i.
* (x,y,z) are unknown position of the receiver that we need to find.

1.There are reference points (Emitters) with known positions.

2.The distance from each reference point to the unknown object is calculated and it is known.This forms a sphere in 3D or a circle in 2D around each reference point.

3.The object is located where all the distance spheres intersect.This is solved mathematically using a system of equations given above.

The solution to this system is obtained by minimizing the sum of the differences between the measured and computed distances.



# Problem Introduction

The task involves designing version 1 of the implementations using agile methodology to prioritize the validation of positioning technique implementation.

The implementation requires the following key steps:

1.Develop an Object Class Diagram to show system structure ,including the prioritized functions.

2.Implement two main algorithms:

* The **N-Lateration algorithm**, which estimates the receivers position position by minimizing the total distance error to four reference points (emitter).
* The **Factory Design Pattern**, which structures and organizes the dataset for efficient handling.

3.Compare the computed position with a geometric resolution of the problem, validating accuracy through visual and analytical methods.

The input data set includes the center of the four APs, the satellite, and the distance, and the receiver,and estimated position minimizes the sum of distances to all spheres.The implementation also incorporates key theoretical models:

* **Celerity Model**, which considers signal travel time for distance calculation.

**d=c\*t** :where d is the distance ,c is speed of signal and is constant 3\*10^8 and t is the time signal takes to go from emitter to receiver.

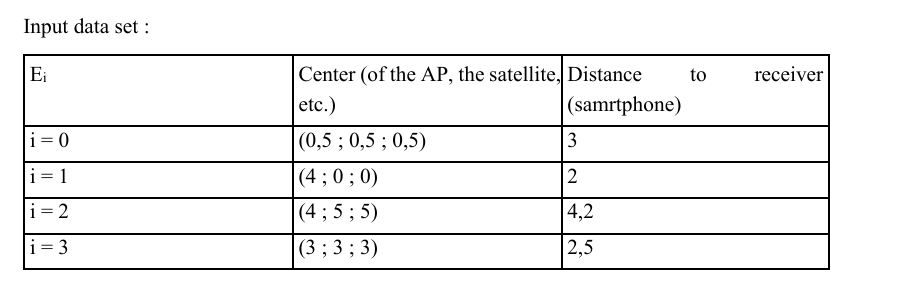
* **Friis Attenuation Model**, which accounts for signal weakening over distance.



## Conceptualization of the Problem

As explained also above, the N lateration algorithm is used to estimate the position of the receiver using the known distances to the reference point (in this case 4). The algorithm involves computing the intersection of spheres with radii equal to the distances from the emitter to each of the receivers. The intersection of at least 4 spheres (in 3D) or 3 spheres (in 2D) gives the estimated position of the receiver.

To create a graphical resolution for the data provided, we can plot the positions of the 4 known points called access points in 3D space. Then, for each point, we can plot a sphere with a radius equal to the distance between the access point and the receiver. The estimated position of the receiver can then be found by computing the intersection of the 4 spheres



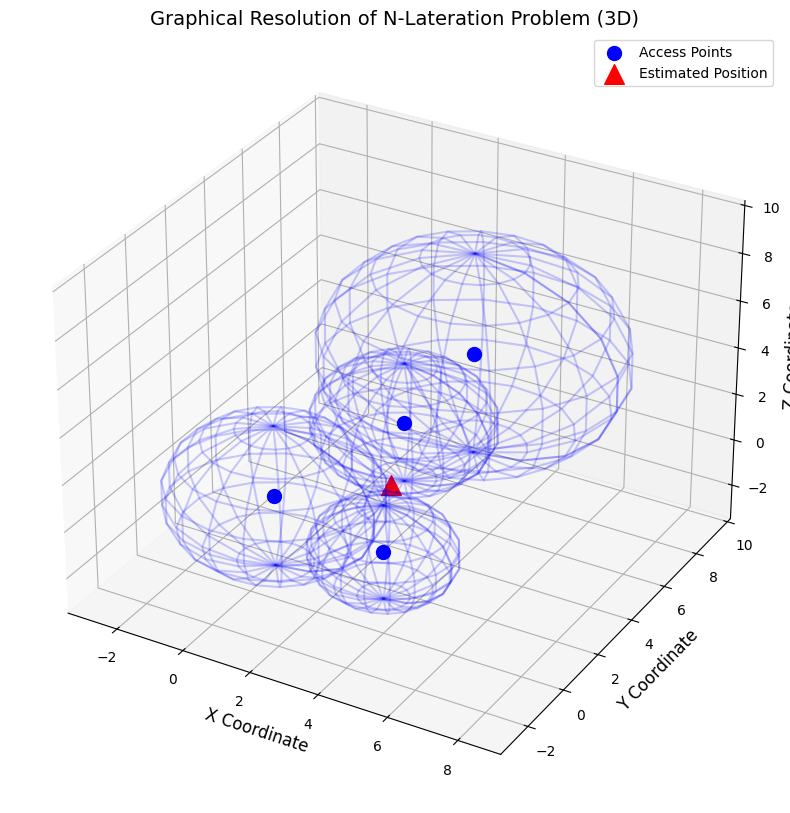
This input data set represents the coordinates of four different points, denoted as Ei, along with the distance from each of these points to receivers(smartphone). The coordinates of each point are given as (x, y, z) values, and the distances are in arbitrary units.

## Graphical 2D resolution of the N-Lateration problem



If we observe the visualization we have a triangle instead of a single point for intersection of 4 circles.In theory, with perfect measurements, the circles would intersect at one exact point, giving the exact location of the receiver.But in practice we get a triangle shape intersection due to some reasons like environment effects (walls,people moving etc),interference,noise etc.The actual estimated coordinates are somewhere in the center of that triangle, computed numerically.

## 3D resolution of the N-Lateration problem



## 

## Architecture and flow of the program

The purpose of this program is to estimate the position of a receiver based on the distances to four emitters using the N-Lateration algorithm. The program is implemented in Python and follows the Scrum/Agile methodology.

**Requirements:**

1. **Programming language:**Python 3 or above as the programming language.
2. **Libraries:**NumPy and SciPy libraries for numerical computations and optimization and matplotlib for graphical visualization 2D/3D,mpl\_toolkits.mplot3d FOR 3D object rendering for plotting spheres.
3. **IDEs**:We worked VS Code and google collab as the recommended IDE, but other IDEs can be used.

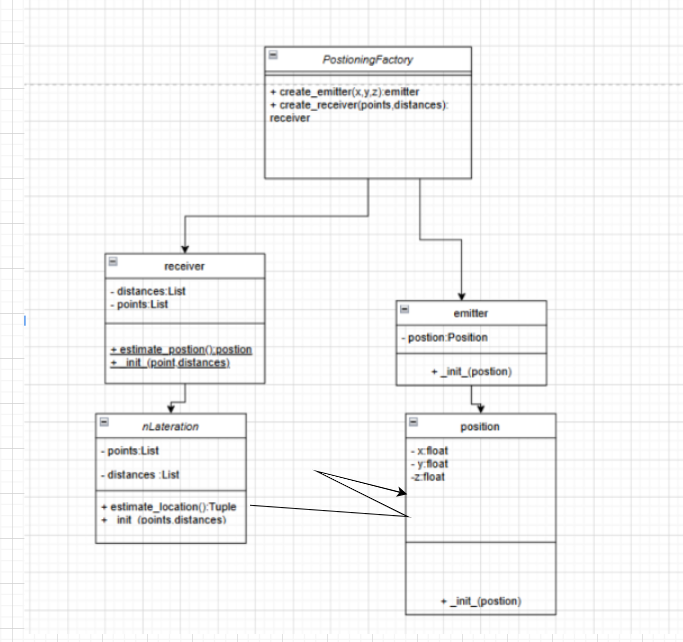
### 

### Design of the Program

The program consists of four classes: Emitter, Position, Receiver, and N-Lateration. The Receiver class is the main component as it takes the distances to known emitters as input and estimate the receiver position.The N-Lateration class has implemented the positioning algorithm using the given distances.In the Emitter class we store the coordinates of reference points and the position class has the estimated location of the receiver in 3D.

### Class Diagram with Positioning Factory

We have done the test and have been focused on the example dataset given in the task but the code also has the functionality to put the data that the user wants dynamically.



The class diagram has 5 classes where each is responsible for a specific thing in the N-Lateration algorithm for finding the position of the object.**PositioningFactory** is the main execution point of the program where user can create the data dynamically or use default values by program .**Emitter** class depend on **position** class for passing the coordinates of the reference points and the known distances.**Receiver** class uses **nLateration** class to instantiate an object and call method to *estimate\_location* for finding position of unknown object by using data that are passed from receiver class.nLateration return the position of unknown object as position.

**1.nLateration** is responsible for solving non-linear least squares optimization problems to estimate the location of a receiver.It has two attributes both are private **points** and **distances.**

* Points to store the coordinates of the known emitters (x,y,z)
* Distances to store the distances from the receiver to each emitter.

Methods:

* estimate\_location() :Uses least\_squares optimization technique to compute the receiver's estimated position.

We decided to use least\_squares optimization instead of Brute Force as this is more efficient.Brute Force checks every point in fixed grid while least squares uses mathematical optimization to find best fit.It is much fasters,more precise and it has lower computational costs compared to brute force.Directly minimizes total error using optimization.

**2.Emitter** class(Signal Transmitting device class). A class representing an emitter that emits signals used to estimate the location of a user. It has a private attribute position representing the center of the emitter (x,y,z).Method \_ \_init \_ \_() which initializes the emitter’s location in 3D space.

**3.Receiver** also can be named as Signal Receiving and Position Estimation class.Receiver class represents a receiver that collects signals from multiple emitters and calculates the receiver’s position,coordinates.It has two private attributes which is private distances.

* Distances is used to store the measured distances from receiver to the emitters.
* Points to store the known (x,y,z) positions of emitters

Method:

* estimate\_position() :Creates an instance of the NLateration class, passing the emitter coordinates and distances as input, and then computes the receiver’s position.

**4.Position** for representing three-dimensional points in space.It has 3 private attributes and no method.

Attributes:

* x (private) → The X-coordinate of the position.
* y (private) → The Y-coordinate of the position.
* z (private) → The Z-coordinate of the position.

Methods:

* \_\_init\_\_() :Initializes the position values (x, y, z).

5. **PositioningFactory Class (Factory Design Pattern)**.The PositioningFactory class is a factory class that dynamically creates Emitter and Receiver objects.We added this just if user want to put other data expect our test.

Methods:

* create\_emitter(x, y, z) → Emitter → Creates an Emitter at (x, y, z).
* create\_receiver(points, distances) → Receiver → Creates a Receiver with given emitter positions and distances.

### Data Structure

1. Lists (list) :Used to store multiple values, such as emitter coordinates and distances.
2. NumPy Arrays (np.array) : Used in the NLateration class for efficient mathematical computations.
3. Dictionaries (dict) :Used in get\_user\_input() to store user-defined or default datasets.
4. Objects (Classes) :Encapsulate data and methods, organizing the program into Position, Emitter, Receiver, NLateration, and PositioningFactory.
5. Tuples (tuple) : Used for immutable storage of emitter positions (x, y, z).

### Pseudo-Code for N-Lateration

The pseudo-code below explains how the program is executed in each step.

**START**

FUNCTION get\_user\_input()

PROMPT user: "Do you want to use the default dataset? (yes/no)"

IF user chooses "yes":

RETURN default emitters & distances

ELSE:

WHILE user has not finished entering:

PROMPT: "Enter emitter position (x y z distance) or 'done'"

IF user enters "done":

BREAK

VALIDATE input format

ADD emitter position and distance to lists

IF less than 4 emitters entered:

PRINT: "Not enough emitters! Using default values."

RETURN default dataset

RETURN entered dataset

FUNCTION PositioningFactory.create\_emitter(x, y, z)

RETURN new Emitter(Position(x, y, z))

FUNCTION PositioningFactory.create\_receiver(points, distances)

RETURN new Receiver(points, distances)

CLASS Position

INIT(x, y, z)

SET self.x = x

SET self.y = y

SET self.z = z

CLASS Emitter

INIT(position)

SET self.position = position

CLASS Receiver

INIT(points, distances)

SET self.points = points

SET self.distances = distances

FUNCTION estimate\_position()

CREATE NLateration instance

CALL estimate\_location()

RETURN estimated position

CLASS NLateration

INIT(points, distances)

SET self.points = points

SET self.distances = distances

FUNCTION estimate\_location()

INITIAL GUESS = Average of emitter positions

DEFINE error function

USE least\_squares optimization

RETURN estimated position as Position object

MAIN EXECUTION:

CALL get\_user\_input() → dataset

CREATE emitters using PositioningFactory

EXTRACT emitter positions

CREATE receiver using PositioningFactory

CALL estimate\_position() from receiver

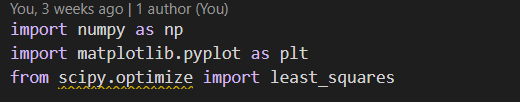
PRINT estimated position

**END**

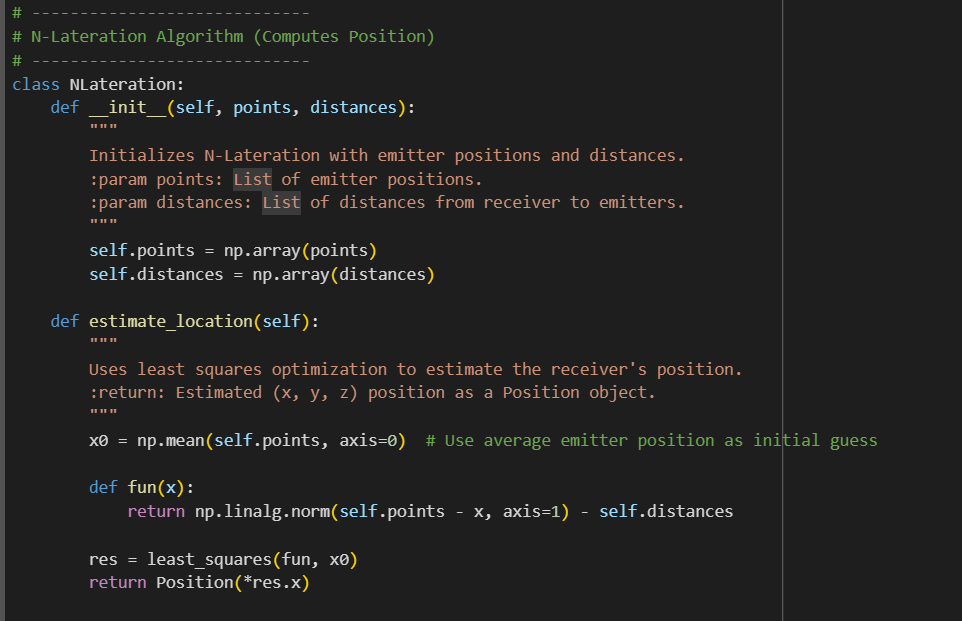
### Algorithm

The N-Lateration algorithm is used to estimate the position of the receiver. The algorithm involves minimizing the sum of distances to all four emitters, which are represented as spheres. This is done by finding the point that lies on the intersection of all four spheres. The optimization problem is solved using the least squares method.

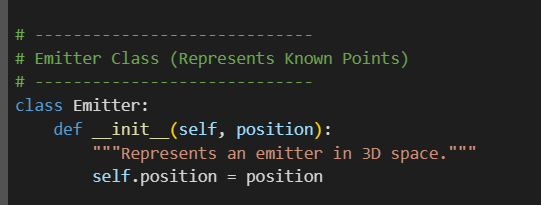
Source Code Documentation



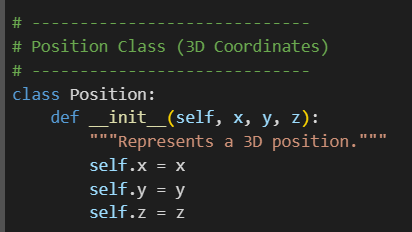
Libraries we have imported for our algorithm



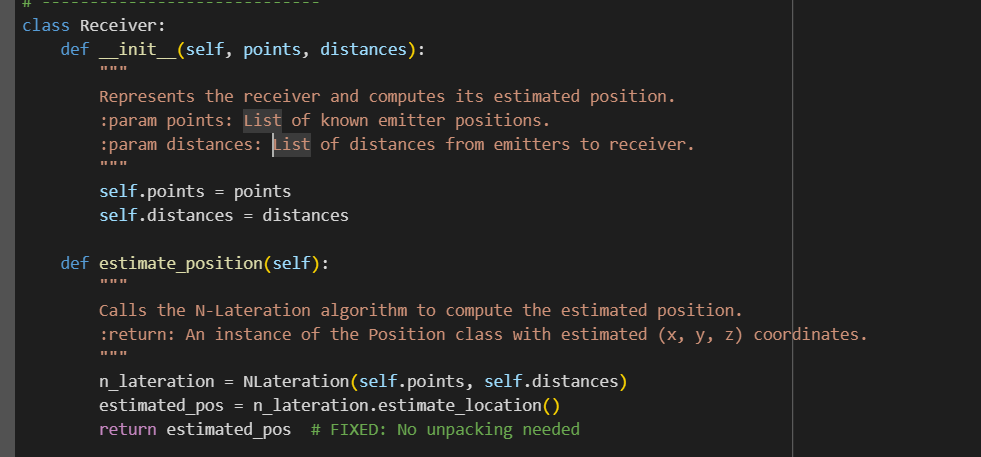
* This class is used to calculate the estimated location of a user based on the distances to several points (emitters) in a 3D space.
* The constructor takes in two arguments: a list of points and a list of distances.
* The method estimate\_location() takes in no arguments, and returns an instance of the Position class, representing the estimated location of the user.



* This class is used to represent an emitter in a 3D space.
* The constructor takes in one argument: the center point of the emitter.



* This class is used to represent a position in a 3D space.
* This class represents only a data holder class,does not computation and do not contain a logic
* The constructor takes in three arguments: the x, y, and z coordinates of the position.



* This class is used to estimate the position of a receiver in a 3D space, based on distances from multiple emitters.
* The constructor takes in two arguments:
  + A list of known emitter positions (each represented as (x, y, z)).
  + A list of distances from the receiver to each emitter.
* The method estimate\_position():
  + Call the N-Lateration algorithm.
  + Computes the receiver's estimated position.
  + Returns an instance of the Position class, which contains the estimated (x, y, z) coordinates.

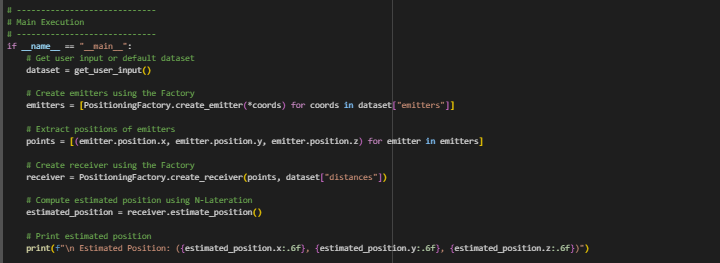


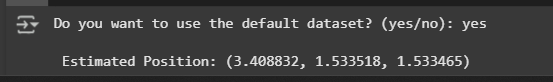
Factory class for creating emitter and receiver objects dynamically.

Implements Factory Design Pattern to separate object creation from logic.

Methods:

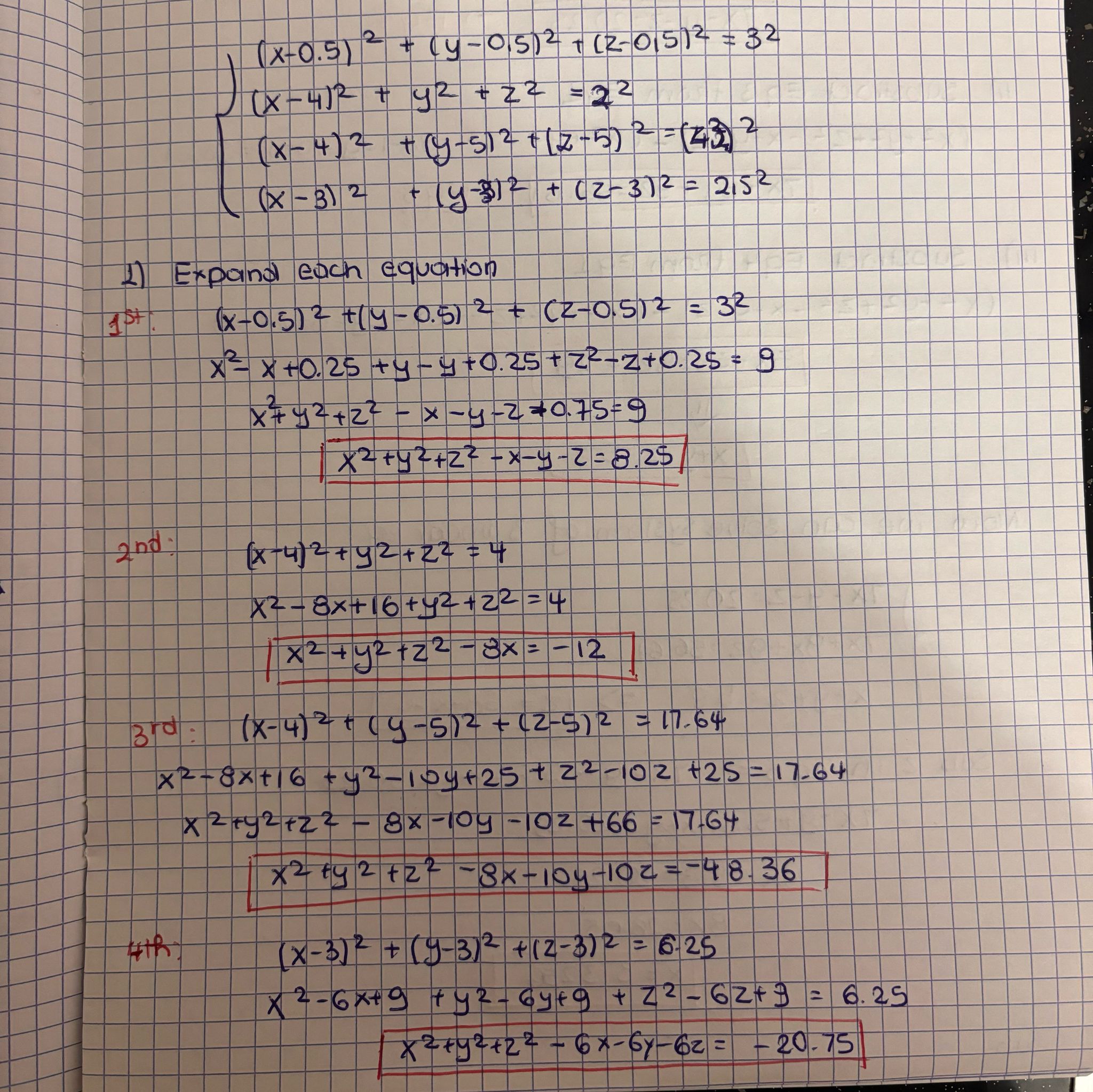
* create\_emitter(x, y, z) → Returns an Emitter object with given (x, y, z).
* create\_receiver(points, distances) → Returns a Receiver object with given emitter positions and distances.
* The get\_user\_input() function handles user input for emitter positions and distances, allowing them to choose between a default dataset (predefined emitters and distances) or entering custom values (x, y, z, distance). It ensures valid input by checking the correct number of values (4 per emitter) and converting them to numerical format. If fewer than 4 emitters are provided, it defaults to the predefined dataset. The function returns a dictionary containing "emitters" and "distances", ensuring flexibility and error-free execution.

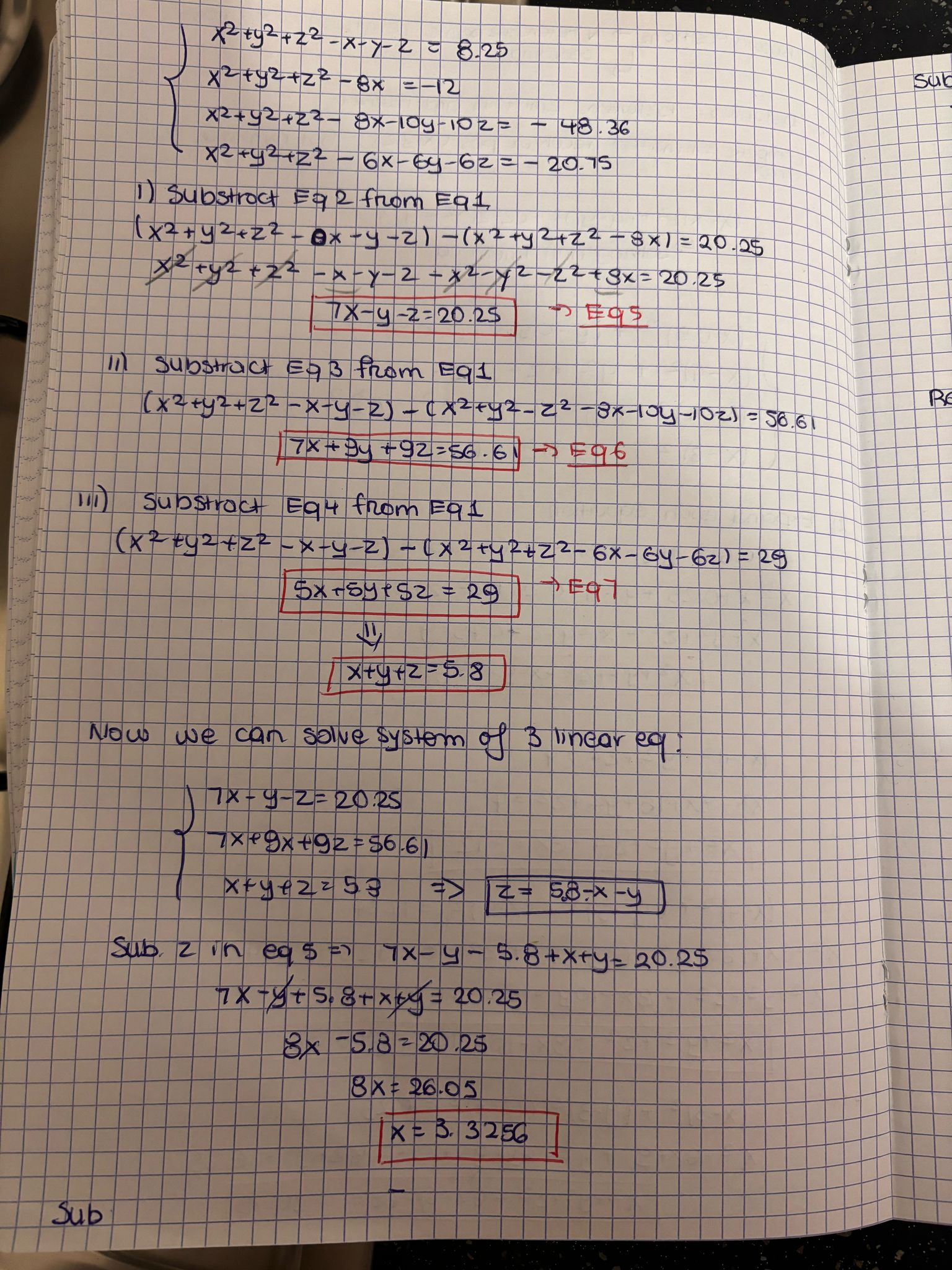
Results:

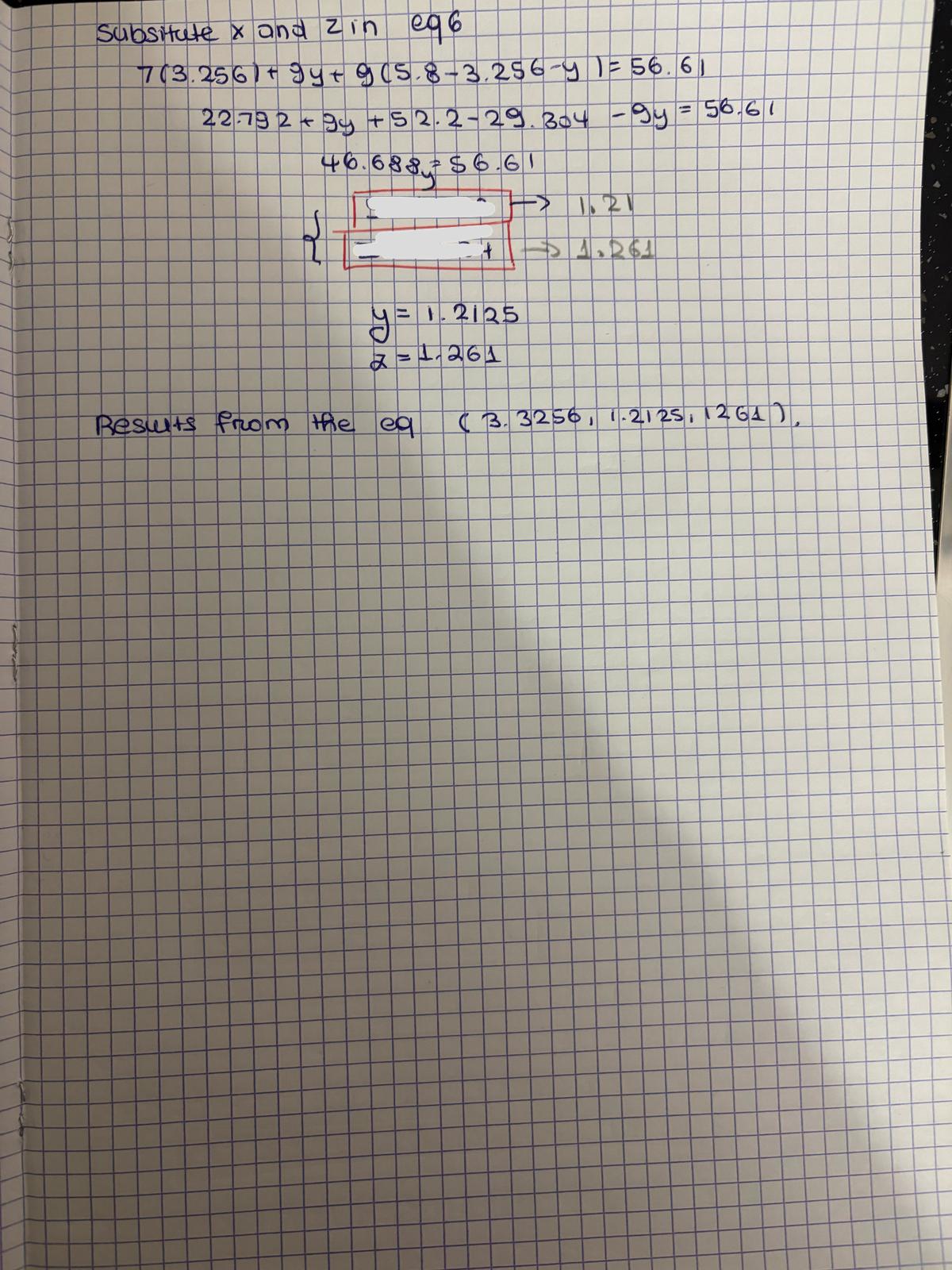


### Comparison with manually calculation results

We solved the system of equations also mathematically and we got this results:(3.3256,1.2125,1.2661)







If we compare the results they do not have much difference x is approximately the same while y and z difference with 0.2-0.3.This shows that the model N-lateration perform well overall and this small difference happens for some reason that can be explained.

Firstly it can be due to Numerical Approximation of the python code.As when we solve the system manually we provide an extract algebraic solution by directly computing the intersection of spheres.While in python source code the least\_squares method in your code minimizes the total error iteratively, which means it approximates the best possible solution but doesn’t necessarily match the exact system solution.

Secondly it can be due to numerical precision and optimization stopping criteria in the least\_squares solver means it stops iterating once it reaches a minimum error threshold, meaning it may not reach the exact analytical solution.

The result of the optimization depends on the starting guess. A different initial guess closer to the actual position might improve accuracy.In our case the least\_squares method starts with an initial guess x0 = [0, 0, 0].If this starting point is far from the actual solution, the optimizer may converge to a slightly different minimum.

Since we are solving an over-constrained system (more equations than unknowns), the algorithm finds the best numerical approximation instead of a strict algebraic intersection.

To conclude The difference in results arises because the Python implementation uses least\_squares optimization, which finds an approximate best-fit solution by minimizing the total squared error. The manual solution is an exact mathematical intersection of the spheres. Additionally, numerical precision limitations, the choice of initial guess, and the over-constrained nature of the problem contribute to small differences in the estimated coordinates.

### Conclusion:

The program successfully estimates the position of the receiver using the N-Lateration algorithm. Compare the result provided by your implementation with a geometric resolution of the scenario provided and we find that our program gets the results in your proposed search area.