Indian Institute of Information Technology, Design and Manufacturing, Kancheepuram

The Reflection Method for the Numerical Solution of Linear Systems

GROUP 11

T Venumadhava Reddy - ME22B1043
K Susmith - ME22B1044

Contents

1 Abstract

We present Cimmino's reflection algorithm for the numerical solution of linear systems, which starts with an arbitrary point in $\mathbb{R}^n$ that gets reflected to the system's hyperplanes. We discuss the iterative nature of the algorithm, error bounds, a few geometrical analyses of a few simple systems of linear equations, and the means to enhance efficiency and convergence. We provide error estimates for the convergence at each step. A probabilistic argument is also devised to improve this elegant geometrical algorithm. Overall, our study provides insights into the effectiveness and applications of Cimmino's method, emphasizing its evolution into a practical tool and its relevance in numerical computation.

2 Introduction

In 1938, Cimmino published an elegant iterative method for the solution of the system

$$\mathbf{Ax} = \mathbf{b} \tag{1}$$

of n linear equations on $\mathbb{R}^n$, where $\mathbf{A}$ is a real $n \times n$ sparse matrix that is initially supposed nonsingular and $\mathbf{x}, \mathbf{b} \in \mathbb{R}$ are column vectors. We commonly use iterative methods like the Gauss-Seidel method, Gauss-Jordan elimination, or Jacobi method to solve (1). We usually produce a sequence of successive approximations $\mathbf{x}^{(k)}$ which we aim to converge to a solution $\mathbf{x}$ as $k \to \infty$, under certain conditions.

In Cimmino's method, we determine the kth approximation $\mathbf{x}^{(k)}$ by geometric reflections. The n equations of (1) are to be identified with n distinct hyperplanes meeting at a unique point Z, whose coordinates satisfy system (1) and are determined by multiple reflections, which will be explained more clearly below.

The study revisits the classical method for solving a standard problem in linear algebra, which was originally developed by the mathematician Gabriel Cramer (1704-1752). This method is viewed in the light of a simple, yet seemingly hidden, geometrical observation made by Cimmino. It states as follows -

Given two lines on a plane intersecting at Z and a point $P \neq Z$, the mirror points of P concerning the lines lie on the circle of center Z and radius dist(P, Z).

The centroid of the initial collection of points becomes the starting point for the next iteration and again centroid of the resulting collection of points becomes the starting point for the next iteration and so on. To be a bit more concise, we could say, 'The Centroid of each collection of points becomes the starting point for the next iteration'.

3 Methodology

3.1 Cimmino's method for n = 2

Consider, in the plane $\mathbb{R}^2$, the straight line r of equation

$$\langle \boldsymbol{a}_i, \boldsymbol{x} \rangle = b_i, \quad i = 1, 2, \tag{2}$$

where $\mathbf{a}^T = (a_1, a_2)$, $a_i, b \in \mathbb{R}$, and $\langle \mathbf{a}, \mathbf{x} \rangle = \sum_{i=1}^{2} a_i x_i$ (This is the same as previous equation(1)), and $\mathbf{a}_i^T$ denotes the i^{th} row of matrix $\mathbf{A}$.

Let's fix $P^{(0)} = P^{(0)}(\boldsymbol{x}^{(0)}) = P^{(0)}(x_1^{(0)}, x_2^{(0)})$ such that $P^{(0)} \notin r$.

$$\begin{aligned} &\text{Orthogonal projection of } P^{(0)} \text{ onto } r, \text{ R} = \boldsymbol{x}^{(0)} + \frac{b - \langle \boldsymbol{a}, \boldsymbol{x}^{(0)} \rangle}{\|\boldsymbol{a}\|^2} \boldsymbol{a}, \\ &\text{Symmetric point of our } P^{(0)} \text{ w.r.to } R, \text{ Q} = \boldsymbol{x}^{(0)} + 2\frac{b - \langle \boldsymbol{a}, \boldsymbol{x}^{(0)} \rangle}{\|\boldsymbol{a}\|^2} \boldsymbol{a}. \end{aligned} \tag{3}$$

Lemma 3.1. *Given two lines on a plane intersecting at Z and a point $P \neq Z$, the mirror points of P concerning the lines lie on the circle of center Z and radius dist(P, Z).*

Given, Z is the Unique solution of our system. That makes previous notation as $P^{(0)} \neq Z$. and for each $i = 1, 2$, the point $Q^{(i)}$ is symmetric to $P^{(0)}$ with respect to the straight line (2), then dist$(Q^{(i)}, Z)$ = dist$(P^{(0)}, Z)$.

We start by fixing an arbitrary point $P^{(0)}$ as our initial approximation and then construct the symmetric points $Q^{(1)}, Q^{(2)}, \ldots, Q^{(n)}$ of $P^{(0)}$ with respect to the n hyperplanes. Let d_1 be the Euclidean distance between $P^{(0)}$ and Z, then d_1 = dist$(P^{(0)}, Z)$. And without losing generality, we can say dist$(P^{(1)}, Z) <$ dist$(P^{(0)}, Z)$.

Let $P^{(1)}$ be the centroid of the points $Q^{(1)}, Q^{(2)}, \ldots, Q^{(n)}$. Now we replace $P^{(0)}$ with the centroid $P^{(1)}$ and iterating to obtain another point $P^{(2)}$ such that dist$(P^{(2)}, Z) <$ dist$(P^{(1)}, Z)$.

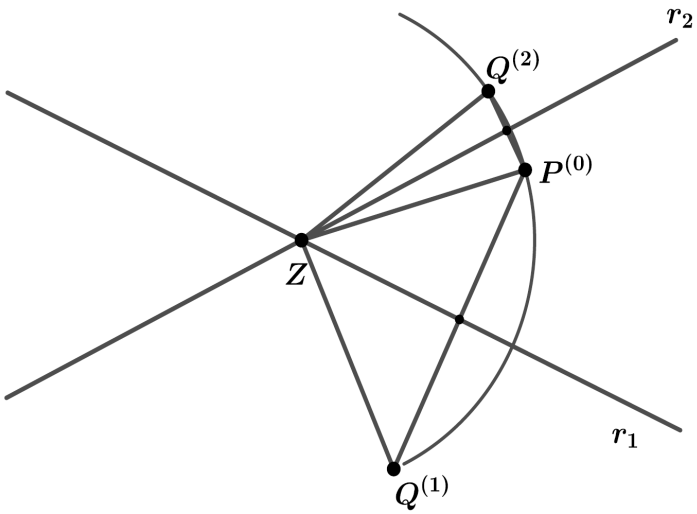
Continuing the process eventually produces the sequence of points, $\{P^{(0)}, P^{(1)}, P^{(2)}, \ldots\}$ and $(P^{(k)})$ converging to Z as $k \to \infty$, i.e. $P^{(k)} \to Z$ for $k \to \infty$.

And this is where we can find Cimmino's geometrical observation, that is

$$\text{dist}(Q^{(i)}, Z) = \text{dist}(P^{(0)}, Z). \tag{4}$$

Considering the triangles $\triangle ZP^{(0)}R^{(1)}$ and $\triangle ZR^{(0)}Q^{(1)}$, there are two same angles and a common side in both the triangles. Therefore, the two triangles will be congruent triangles by the ASA congruency rule (Angle-Side-Angle). Therefore $ZP^{(0)} = ZQ^{(1)}$.

This can be applied to the other reflective point of $P^{(0)}$, i.e., $Q^{(2)}$ also. Therefore we get, $ZP^{(0)} = ZQ^{(1)} = ZQ^{(2)}$, which means the points $P^{(0)}$, $Q^{(1)}$, and $Q^{(2)}$ lie on the same circle with the center as the solution Z and a radius of dist$(Z, P^{(0)})$. See Figure **??** below.



Given the initial approximation $P^{(0)} = P^{(0)}(\boldsymbol{x}^{(0)})$, then $P^{(1)} = P^{(1)}(\boldsymbol{x}^{(1)})$ is the centroid of unit masses placed at $Q^{(i)}$, $i = 1, 2$, of $P^{(0)}$ with respect to the straight line $\langle \boldsymbol{a}_i, \boldsymbol{x} \rangle = b_i$, we get the expression:

$$\boldsymbol{x}^{(1)} = \boldsymbol{x}^{(0)} + \sum_{i=1}^{2} \frac{b_i - \langle \boldsymbol{a}_i, \boldsymbol{x}^{(0)} \rangle}{\|\boldsymbol{a}_i\|^2} \boldsymbol{a}_i. \tag{5}$$

This will be taken as the starting point of the next iteration, and so on. At step $k + 1$,

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \sum_{i=1}^{2} \frac{b_i - \langle \boldsymbol{a}_i, \boldsymbol{x}^{(k)} \rangle}{\|\boldsymbol{a}_i\|^2} \boldsymbol{a}_i. \tag{6}$$

3.2 Error Bounds

In iterative methods, the solution is determined by approaching through a series of approximations. These error bounds are needed to determine when to stop the iteration or to measure the accuracy of the current approximation to the exact solution of a problem. By doing so, we can ensure that the solution is both reliable and computationally feasible

Let's indicate $Z = Z(\xi_1, \xi_2)$ the point which solve our system Ax=b. And fix a point $P^{(0)} \neq Z.Using\langle a_i, \xi\rangle = b_i$, $i = 1, 2$, we define,

$$\eta_i = \langle a_i, x^{(0)}\rangle - b_i = \langle a_i, x^{(0)} - \xi\rangle \tag{7}$$

Now, using the equations (5) and (7),

$$x^{(1)} = x^{(0)} - \sum_{i=1}^{2} \frac{\langle a_i, x^{(0)} - \xi\rangle}{||a_i||^2} a_i. \tag{8}$$

observe,

$$\|x^{(1)} - \xi\|^2 = \|x^{(0)} - \xi\|^2 - 2\sum_{i=1}^{2} \frac{\eta_i^2}{|a_i\|^2} + \left\|\sum_{i=1}^{2} \frac{1}{\|a_i\|^2} \cdot \eta_i \cdot a_i\right\|^2 \tag{9}$$

$$\left\|\sum_{i=1}^{2} p_i \eta_i a_i\right\|^2 \leq \left(\sum_{i=1}^{2} p_i \eta_i^2\right) \cdot \left(\sum_{i=1}^{2} p_i \|a_i\|^2\right), \tag{10}$$

where, $p_i = \frac{1}{\|a_i\|^2}$.

Our equation (7) follows as,

$$\|x^{(1)} - \xi\| \leq \|x^{(0)} - \xi\| \tag{11}$$

Now, the above expression is for $P^{(0)}$ and we continue iterating, we get to obtain a sequence $P^{(\nu)}$ and its corresponding estimates

$$\|x^{(\nu+1)} - \xi\| \leq \|x^{(\nu)} - \xi\| \tag{12}$$

there is equality in (10) iff $x^{(\nu)}$ solves the given system, and hence $x^{(\nu)} = \xi$ and also $x^{(\nu+1)} = x^{(\nu)}$ for any $\nu \in \mathbb{N}$.

4 Code

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define MAX_SIZE 100
#define THRESHOLD 1e-6 // Threshold for residual check
#define MAX_ITERATIONS 10000 // Maximum number of iterations to prevent infinite loop

void inputMatrix(float A[MAX_SIZE][MAX_SIZE], int n) {
    printf("Enter the coefficients of the matrix A:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%f", &A[i][j]);
        }
    }
}

void inputVector(float b[MAX_SIZE], int n) {
    printf("Enter the constants of the vector b:\n");
    for (int i = 0; i < n; i++) {
        scanf("%f", &b[i]);
    }
}

void inputInitialVector(float x[MAX_SIZE], int n) {
    printf("Enter the initial values of the vector x:\n");
    for (int i = 0; i < n; i++) {
        scanf("%f", &x[i]);
    }
}

void printVector(float x[MAX_SIZE], int n) {
    for (int i = 0; i < n; i++) {
        printf("%f ", x[i]);
    }
    printf("\n");
}
```

```
int main() {
    int n;
    printf("Enter the size of the matrix and vectors: ");
    scanf("%d", &n);

    float A[MAX_SIZE][MAX_SIZE], b[MAX_SIZE], x[MAX_SIZE], x_new[MAX_SIZE], r[MAX_SIZE],

    inputMatrix(A, n);
    inputVector(b, n);
    inputInitialVector(x, n);

    int iteration = 0;
    while (iteration < MAX_ITERATIONS) {
        iteration++;

        // Calculate the residual vector r = b - Ax
        for (int i = 0; i < n; i++) {
            temp[i] = 0;
            for (int j = 0; j < n; j++) {
                temp[i] += A[i][j] * x[j];
            }
            r[i] = b[i] - temp[i];
        }

        // Calculate the projection and reflection
        for (int i = 0; i < n; i++) {
            x_new[i] = 0;
            for (int j = 0; j < n; j++) {
                x_new[i] += A[i][j] * r[i];
            }
            x_new[i] /= (pow(A[i][i], 2));
        }

        // Update x with the new values
        for (int i = 0; i < n; i++) {
            x[i] += x_new[i];
        }

        // Calculate the residual to check for convergence
        float residual = 0.0;
```

```
        for (int i = 0; i < n; i++) {
            float sum = 0.0;
            for (int j = 0; j < n; j++) {
                sum += A[i][j] * x[j];
            }
            residual += fabs(b[i] - sum);
        }

        if (residual < THRESHOLD) {
            break; // Converged, exit the loop
        }
    }

    if (iteration == MAX_ITERATIONS) {
        printf("Maximum iterations reached. Solution may not have converged.\n");
    } else {
        printf("The final approximation x is: ");
        printVector(x, n);
    }

    return 0;
}
```

Output:

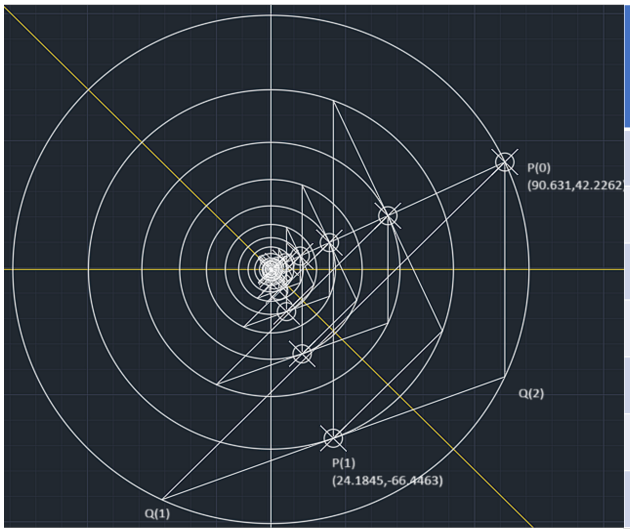
```
    Enter the size of the matrix and vectors: 2
Enter the coefficients of the matrix A:
1 1
2 -5
Enter the constants of the vector b:
5 -11
Enter the initial values of the vector x:
1 1
The final approximation x is: 2.000000 3.000000
```

The code is working properly.

5 Simple Analysis

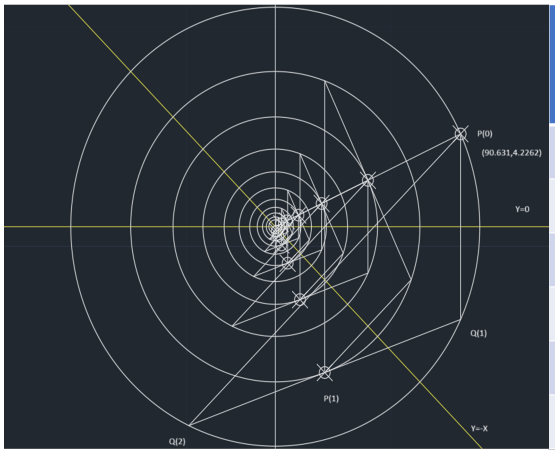
We took a few simple cases with simple equations of different cases of choosing the initial approximation to find out the effect it'll have on our calculations.

We are choosing $P^{(0)}$ at a distance of 100 and 10 units from Z at both acute and obtuse angles. We are using AutoCAD software to implement this analysis.



No of ITERATIONS	ITERATION POINT	ERROR
1	(90.6308,42.2618)	100
2	(24.1845,-66.4463)	70.7107
3	(45.3154,21.1309	50
....	(....,....)	....
16	(0.1889,-0.5191)	0.5524
17	(0.354,0.1654)	0.3096
18	(0.0945,-0.2596)	0.2762

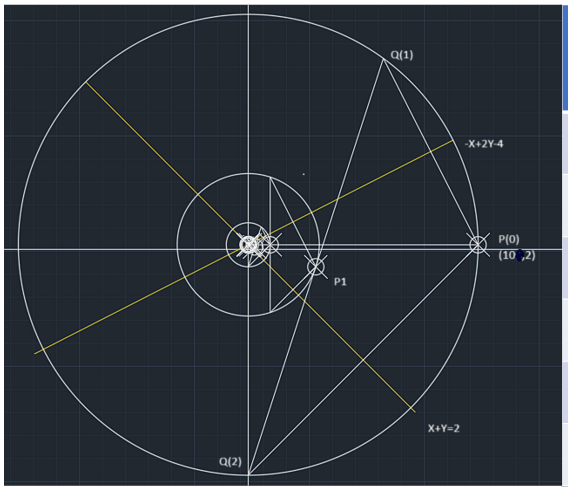
Figure 1: y=0, y=-x and $P^{(0)} = (90.631, 42.262)$



NO . of ITERATIONS	ITERATION POINT	ERROR
1	(9.0631,4.2262)	10
2	(2.4184,-6.6446)	7.0711
3	(4.5315,2.1131)	5
...	(....,....)	...
11	(0.2832,1.1321)	0.3125
12	(0.0756,-0.2076)	0.221

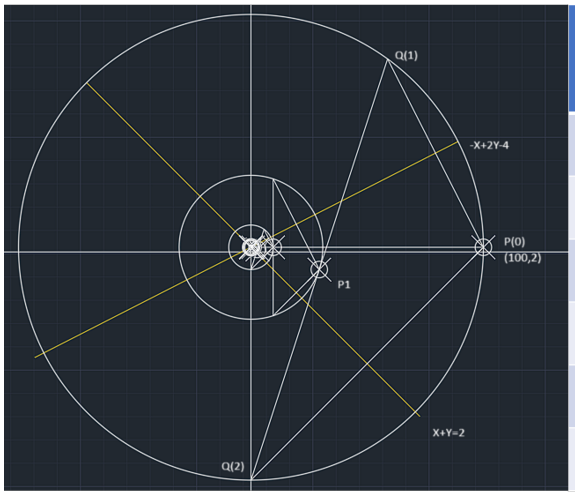
Figure 2: y=0, y=-x and $P^{(0)} = (9.0631, 4.2262)$

INDIAN INSTITUTE OF INFORMATION TECHNOLOGY, DESIGN AND MANUFACTURING, KANCHEEPURAM



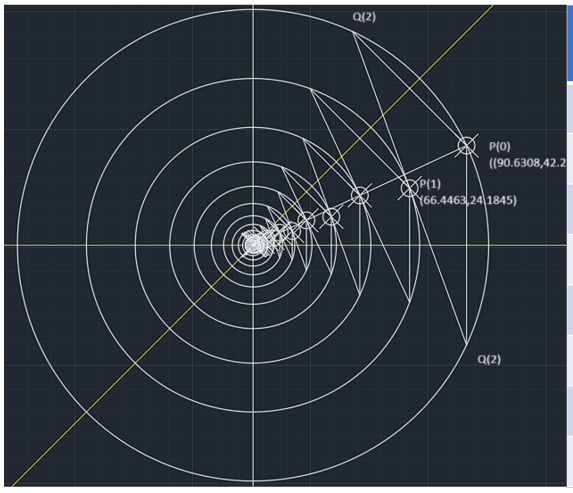
NO OF ITERATIONS	ERROR
1	10
2	3.0902
3	0.9549
.....	.....
9	0.0008
10	0.0002

Figure 3: x+y=2, -x+2y=4 and $P^{(0)} = (10, 2)$



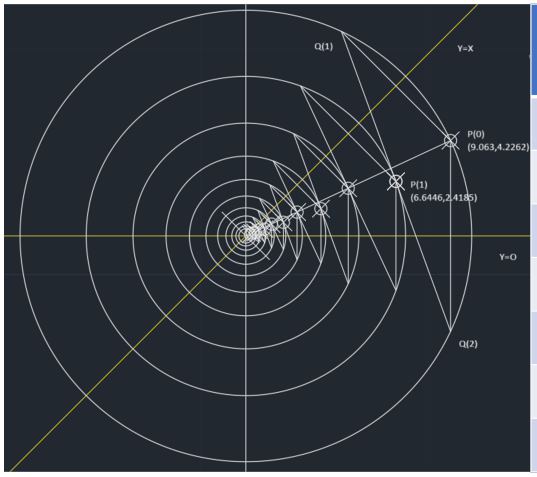
NO OF ITERATIONS	ERROR
1	100
2	3.090175
3	9.5492
.....	.....
11	0.0008
12	0.0002

Figure 4: x+y=2, -x+2y=4 and $P^{(0)} = (100, 2)$



No of ITERATIONS	ITERATION POINT	ERROR
1	(90.6308,42.2618)	100
2	(66.4463,24.1845)	70.7107
3	(45.3154,21.1309	50
4	(33.2232,12.0922	35.3553
.....	(.....,.....)	.....
16	(0.51991,0.1889)	0.5524
17	(0.354,0.1654)	0.3096
18	(0.2596,0.0945)	0.2762

Figure 5: y=0, y=x and $P^{(0)} = (90.631, 42.262)$



NO .OF ITERATIONS	ITERATIVE POINT	ERROR
1	(9.0631,4.2262)	10
2	(6.6446,2.4184)	7.0711
3	(4.5315,2.1131)	5
.....	(.....,.....)	.....
10	(0.4153,0.1512)	0.4419
11	(0.2832,0.1321)	0.3125
12	(0.2076,0.0756)	0.221

Figure 6: y=0, y=x and $P^{(0)} = (9.0631, 4.2262)$

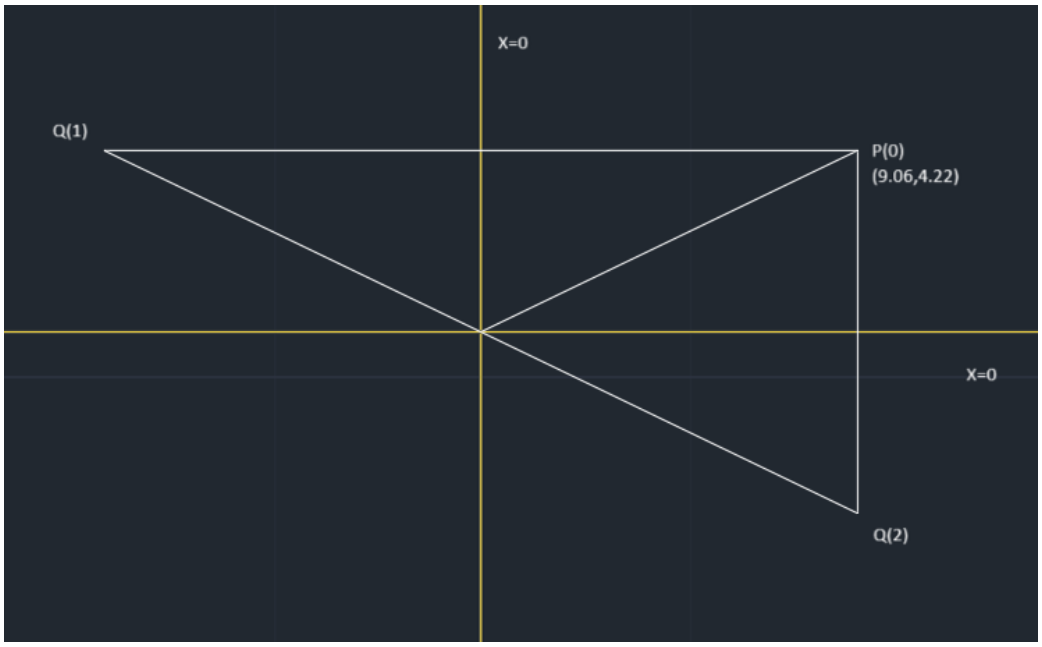


Figure 7: x=0, y=0 and $P^{(0)} = (9.0631, 4.2262)$

The convergence analysis provides error estimates for the convergence of the algorithm at each iteration. The relative error is the same even if the distance between the arbitrarily chosen point and solution Z changes. The current iterative point and the reflective points of the current iterative point to the line equations always lie on the same circle, which was already proven, with the error values as the radius of the circle and center as the exact solution Z of the system of linear equations.

- Case 1: The angle between the lines = $90°$. Then the first iteration itself converges to the solution (considering the arbitrary point is not coinciding with the solution).
- Case 2: The angle between the lines is not equal to $90°$ (considering the arbitrary point is not coinciding with the solution).

The results are observed as:

1. The point which is chosen arbitrarily lies in the region where the lines make an acute angle, and then the convergence happens normally.
2. The point that is chosen arbitrarily lies in the region where the line makes an obtuse angle, and then the convergence happens such that the centroid of the reflective points lies on the opposite side of the solution.

One must remember that there may be errors in our analysis as AutoCAD is used instead of high numerical computational software.

6 Conclusion

Cimmino's method has been extensively discussed in various publications, often alongside the Kaczmarz method, with comparisons and extensions. It has also been independently rediscovered by researchers in applied sciences. Between 1970 and 1980, there was a resurgence of interest in these algorithms due to two technological advances: the widespread use of tomography (CAT scans) in radiology, which required regularization effects for image reconstruction from projections, and the rapid development of parallel computing, especially concerning the Cimmino algorithm.

Over the years, Cimmino's method has found applications in several areas, including convex mathematical programming, fast adaptation of radiation therapy planning, solution of "inverse problems" in medical physics, and filtering in signal processing. Many of these problems are nonlinear and are formulated in terms of inequalities.