

**JAYPEE INSTITUTE OF INFORMATION
TECHNOLOGY NOIDA**

**B.TECH V SEMESTER
REPORT FOR PBL**



SOLUTION OF THE SYSTEM OF LINEAR EQUATIONS

SUBMITTED TO:

Dr. DINESH C. S. BISHT

GROUP MEMBERS:

GEETALI AGARWAL- 20103254

KRITARTH BANSAL- 20103256

SAKSHAM GUPTA - 20103268

SWATI AGARWAL- 20103278

AMOL MATHUR- 20104040

Introduction:

A system of equations is a set or collection of equations that are solved together. A collection of linear equations is called a system of linear equations. They are often based on the same set of variables. Various methods have been developed for solving linear equations, but the best method for solving systems of linear equations has not yet been proposed. Different mathematicians have suggested different methods depending on speed and accuracy. Linear equation methods fall into two categories. direct and indirect. Each category contains multiple elimination methods used to solve equations.

Guassian elimination method is a direct method for solving system of linear equations.

B. Literature Survey (History/development)

This brief history of iterative solution methods has been adapted from [3]. It seems that the earliest mention of iterative methods was made by Carl Friedrich Gauss (1777-1855) in 1823. In the mid 1820s Gauss and, later in 1874, Philipp Ludwig von Seidel (1821-1896) developed the Gauss-Seidel iteration method. This was the starting point for iterative solution methods. Traditionally, iterative methods have been used for solving large linear systems with diagonally dominant sparse matrices. The development of over-relaxed iterative methods seems to have been initiated by the PhD work in 1950 of David Young (1923-2008). During the 1950s the field progressed and a stronger theory around iterative methods and linear algebra was developed by, for example, Varga, Ostrowski, Reich and Kahan. Successive over-relaxation methods became the methods of choice in solving large practical problems, such as nuclear diffusion reaction, weather prediction, and oil reservoir modeling. Although they are not quite popular now, they are still used in some instances either as the main iterative solution method or in combination with more recent methods. The successive over-relaxation methods made it possible to efficiently solve systems of order 20,000 by 1960 and five years later systems of order 100,000 could be solved in problems concerning eigenvalue computations in nuclear diffusion codes. The success of these successive over-relaxation methods led to a rich theory of iterative methods that could be successfully used in the analysis of later methods. Some linear systems arising from real-life applications, such as large electric circuits, are not easy to solve using iterative methods in an efficient and reliable manner. While progress is being made, there is much more research to be made.

C. Description of the topic/mathematical foundation

- **Gauss - Seidel Elimination Method**

The **Gauss-Seidel method** is the modification of the gauss-iteration method. This modification reduces the number of iterations. In this method the value of unknown immediately reduces the number of iterations, the calculated value replaces the earlier value only at the end of the iteration. .Because of this, the gauss-seidel methods converge much faster than the Gauss

methods. In Gauss seidel methods the number of iteration methods required to obtain the solution is much less as compared to Gauss method.

Let us take Jacobi's Method one step further. Where the true solution is $\mathbf{x} = (x_1, x_2, \dots, x_n)$, if $x_1^{(k+1)}$ is a better approximation to the true value of x_1 than $x_1^{(k)}$ is, then it would make sense that once we have found the *new* value $x_1^{(k+1)}$ to use it (rather than the old value $x_1^{(k)}$) in finding $x_2^{(k+1)}, \dots, x_n^{(k+1)}$. So $x_1^{(k+1)}$ is found as in Jacobi's Method, but in finding $x_2^{(k+1)}$, instead of using the *old* value of $x_1^{(k)}$ and the old values $x_3^{(k)}, \dots, x_n^{(k)}$, we now use the *new* value $x_1^{(k+1)}$ and the old values $x_3^{(k)}, \dots, x_n^{(k)}$, and similarly for finding $x_3^{(k+1)}, \dots, x_n^{(k+1)}$.

This technique is called the Gauss-Seidel Method -- even though. It is described by,

$$\begin{aligned} a_{11}x_1^{(k+1)} + a_{12}x_2^{(k)} + \dots + a_{1n}x_n^{(k)} &= b_1 \\ a_{21}x_1^{(k+1)} + a_{22}x_2^{(k+1)} + \dots + a_{2n}x_n^{(k)} &= b_2 \\ \vdots & \\ a_{n1}x_1^{(k+1)} + a_{n2}x_2^{(k+1)} + \dots + a_{nn}x_n^{(k+1)} &= b_n \end{aligned}$$

$$\begin{bmatrix} a_{11} & 0 & \dots & 0 \\ a_{21} & a_{22} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ a_{n1} & \dots & a_{nn-1} & a_{nn} \end{bmatrix} \begin{bmatrix} x_1^{(k+1)} \\ x_2^{(k+1)} \\ \vdots \\ x_n^{(k+1)} \end{bmatrix} + \begin{bmatrix} 0 & a_{12} & \dots & a_{1n} \\ 0 & 0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & a_{n-1n} \\ 0 & \dots & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1^{(k)} \\ x_2^{(k)} \\ \vdots \\ x_n^{(k)} \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}.$$

That is,

$$(L + D)\mathbf{x}^{(k+1)} + U\mathbf{x}^{(k)} = \mathbf{b},$$

so that

$$\mathbf{x}^{(k+1)} = (L + D)^{-1}[-U\mathbf{x}^{(k)} + \mathbf{b}].$$

Gauss Seidel Code:

We'll be solving the below equation via gauss seidel method program :-

$$2x + 3y + z = 2$$

$$5x + 4y + 6z = 3$$

$$8x + 7y + 9z = 4$$

*** SEIDAL PROGRAM TO FIND SOLUTION OF THE ABOVE SYSTEM OF EQUATIONS**

```
#include<iostream>
```


```
#include<conio.h>
```

```
using namespace std;
```

```

int main(void) {
    float a[10][10], b[10], x[10], y[10];
    int n = 0, m = 0, i = 0, j = 0;
    cout << "Enter size of 2d array(Square matrix) : "; cin >> n;
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            cout << "Enter values no :(" << i << ", " << j << ") ";
            cin >> a[i][j]; } }
    cout << "\nEnter Values to the right side of equation\n";
    for (i = 0; i < n; i++)
    { cout << "Enter values no :(" << i << ", " << j << ") ";
      cin >> b[i]; }
    cout << "Enter initial values of x\n";
    for (i = 0; i < n; i++) {
        cout << "Enter values no. :(" << i << "):";
        cin >> x[i]; }
    cout << "\nEnter the no. of iteration : "; cin >> m;
    while (m > 0) {
        for (i = 0; i < n; i++) {
            y[i] = (b[i] / a[i][i]);
            for (j = 0; j < n; j++) {
                if (j == i) continue;
                y[i] = y[i] - ((a[i][j] / a[i][i]) * x[j]);
                x[i] = y[i]; }
            printf("x%d = %f  ", i + 1, y[i]); }
        cout << "\n";
        m--; }
    return 0; }

```


input

```

Enter size of 2d array(Square matrix) : 3
Enter values no : (0, 0) 2
Enter values no : (0, 1) 3
Enter values no : (0, 2) 1
Enter values no : (1, 0) 5
Enter values no : (1, 1) 4
Enter values no : (1, 2) 6
Enter values no : (2, 0) 8
Enter values no : (2, 1) 7
Enter values no : (2, 2) 9

Enter Values to the right side of equation
Enter values no : (0, 3) 2
Enter values no : (1, 3) 3
Enter values no : (2, 3) 4
Enter initial values of x
Enter values no. : (0):1
Enter values no. : (1):2
Enter values no. : (2):5

Enter the no. of iteration : 5
x1 = -4.500000    x2 = -1.125000    x3 = 5.319445
x1 = 0.027778    x2 = -7.263889    x3 = 6.069445
x1 = 8.861112    x2 = -19.430557    x3 = 7.680557
x1 = 26.305557    x2 = -43.652782    x3 = 11.013891
x1 = 60.972225    x2 = -91.986115    x3 = 17.791664

```

- **Gauss - Jacobi Elimination Method**

Jacobian method or **Jacobi method** is one the iterative methods for approximating the solution of a system of n linear equations in n variables. The Jacobi iterative method is considered as an iterative algorithm which is used for determining the solutions for the system of linear equations in numerical linear Algebra, which is diagonally dominant. In this method, an approximate value is filled in for each diagonal element. Until it converges, the process is iterated. This algorithm was first called the Jacobi transformation process of matrix diagonalization. Jacobi Method is also known as the simultaneous displacement method.

Given a current approximation,

$$\mathbf{x}^{(k)} = (x_1^{(k)}, x_2^{(k)}, x_3^{(k)}, \dots, x_n^{(k)})$$

for \mathbf{x} , the strategy of Jacobi's Method is to use the first equation and the current values of $x_2^{(k)}, x_3^{(k)}, \dots, x_n^{(k)}$ to find a new value $x_1^{(k+1)}$, and similarly to find a new value $x_i^{(k+1)}$ using the i^{th} equation and the old values of the other variables. That is, given current values $\mathbf{x}^{(k)} = (x_1^{(k)}, x_2^{(k)}, \dots, x_n^{(k)})$, find new values by solving for

$$\mathbf{x}^{(k+1)} = (x_1^{(k+1)}, x_2^{(k+1)}, \dots, x_n^{(k+1)})$$

in

$$\begin{array}{cccccc} a_{11}x_1^{(k+1)} & + & a_{12}x_2^{(k)} & + & \dots & + & a_{1n}x_n^{(k)} & = & b_1 \\ a_{21}x_1^{(k)} & + & a_{22}x_2^{(k+1)} & + & \dots & + & a_{2n}x_n^{(k)} & = & b_2 \\ \vdots & & \vdots & & \ddots & & \vdots & & \vdots \\ a_{n1}x_1^{(k)} & + & a_{n2}x_2^{(k)} & + & \dots & + & a_{nn}x_n^{(k+1)} & = & b_n \end{array}$$

This system can also be written as -

$$\begin{bmatrix} a_{11} & 0 & \dots & 0 \\ 0 & a_{22} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & a_{nn} \end{bmatrix} \begin{bmatrix} x_1^{(k+1)} \\ x_2^{(k+1)} \\ \vdots \\ x_n^{(k+1)} \end{bmatrix} + \begin{bmatrix} 0 & a_{12} & \dots & a_{1n} \\ a_{21} & 0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & a_{n-1n} \\ a_{n1} & \dots & a_{nn-1} & 0 \end{bmatrix} \begin{bmatrix} x_1^{(k)} \\ x_2^{(k)} \\ \vdots \\ x_n^{(k)} \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}.$$

To be clear, the subscript i means that $x_i^{(k)}$ is the i^{th} element of vector

$$\mathbf{x}^{(k)} = (x_1^{(k)}, x_2^{(k)}, \dots, x_i^{(k)}, \dots, x_n^{(k)}),$$

and superscript k corresponds to the particular iteration (not the k^{th} power of x_j).

Here,

$D = \text{Diagonal Matrix}$

$L = \text{Strict Lower Triangular}$

$U = \text{Strict Upper Triangular Matrix}$

$$D = \begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & a_{nn} \end{bmatrix}, L = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ a_{21} & 0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ a_{n1} & \cdots & a_{nn-1} & 0 \end{bmatrix} \text{ and } U = \begin{bmatrix} 0 & a_{12} & \cdots & a_{1n} \\ 0 & 0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & a_{n-1n} \\ 0 & \cdots & 0 & 0 \end{bmatrix},$$

then Jacobi's Method can be written in matrix-vector notation as

$$D\mathbf{x}^{(k+1)} + (L+U)\mathbf{x}^{(k)} = \mathbf{b}$$

so that

$$\mathbf{x}^{(k+1)} = D^{-1}[(-L-U)\mathbf{x}^{(k)} + \mathbf{b}].$$

* JACOBI PROGRAM TO FIND SOLUTIONS

```
#include<stdio.h>
#include<math.h>
int main() {
    int i,j,m,n,l;
    float x[10],a[10][10],b[10],c[10];
    printf("\nEnter the value of n : \n"); scanf("%d",&n);
    printf("\nEnter the number of iterations : \n"); scanf("%d",&l);
    printf("\nEnter the right hand side constants : \n");
    for(i=0;i<n;i++) scanf("%f",&b[i]);

    printf("\nEnter the coefficients row wise : \n");
    for(i=0;i<n;i++) { x[i]=0;
        for(j=0;j<n;j++) scanf("%f",&a[i][j]); }
    m=1;
line:
    for(i=0;i<n;i++) {
        c[i]=b[i];
        for(j=0;j<n;j++) {
            if(i!=j) {
                c[i]=c[i]-a[i][j]*x[j]; } }
        for(i=0;i<n;i++) x[i]=c[i]/a[i][i];
        m++;
        if(m<=l) goto line;
    else {
        printf("\nThe Solution is : \n");
        for(i=0;i<n;i++) printf("\nx(%d) = %f\n",i,x[i]); }
}
```

```
input
Enter the value of n :
4
Enter the number of iterations :
13
Enter the right hand side constants :
3 15 27 -9
Enter the coefficients row wise :
10 -2 -1 -1
-2 10 -1 -1
-1 -1 10 -2
-1 -1 -2 10
The Solution is :
x(0) = 0.999990
x(1) = 1.999990
x(2) = 2.999990
x(3) = -0.000010
```

D. Future aspects

In the future we will try to find methods for faster calculations of systems of Linear Equations . The world of Gaussian processes will remain exciting for the foreseeable as research is being done to bring their probabilistic benefits to problems currently dominated by deep learning — sparse and minibatch Gaussian processes increase their scalability to large datasets while deep and convolutional Gaussian processes put high-dimensional and image data within reach. Watch this space.

E. Conclusion

Gauss-Seidel method is more efficient than Jacobi method as Gauss-Seidel method requires less number of iterations to converge to the actual solution with a certain degree of accuracy.

The disadvantage of the Jacobi method is even after the modified value of a variable is evaluated in the present iteration, it is not used until the next iteration. In other words, the value of all the variables which are used in the current iteration are from the previous iteration therefore an increase in the number of iterations to reach the exact solution.

F. Bibliography:-

- Feige *J., Reichman D. On Systems of Linear Equations with Two Variables per Equation. In: Approximation, Randomization, and Combinatorial Optimization: Algorithms and Techniques. Lecture Notes in Computer Science.
- Lay D.C. (2005) Linear Algebra and Its Applications. 3rd edition. Addison Wesley.
- Atkinson K. (1989) An Introduction to Numerical Analysis. 2nd edition. New York: John Wiley & Sons.
- Mittal R.C., Al-Kurdi A. LU-decomposition and numerical structure for solving large sparse nonsymmetric linear systems. In: Computers & Mathematics with Applications. 2002