**Unit 4: Solving System of Linear Equations**

**Need and Scope**

Analysis of linear equations is significant for a number of reasons. First, mathematical models of many of the real world problems are either linear or can be approximated reasonably well using linear relationships. Second, the analysis of linear relationship of variables is generally easier than that of non-linear relationships.

A linear equation involving two variables x and y has the standard form

ax+by=c

where a,b and c are real numbers and a and b cannot both equal zero. Here the exponent of variables is one. The equation becomes non-linear if any of the variables has the exponent other than one. Similarly, equations containing terms involving a product of two variables are also considered nonlinear.

Some of the examples of linear equations are:

4x+7y=15

-x-2/3y=0

3u-2v=-1/2

Some examples of non-linear equations are:

2x-xy+y=2

x2+y2=25

x+√y=6

In practice, linear equations occur in more than two variables. A linear equation with n variables has the form

a1x1+a2x2+a3x3+a4+x4+…..+xnxn = b

where ai(i=1,2,….n) are real numbers and at least one of them is not zero. The main concern here is to solve for xi (i=1,2…n) given the values of ai and b. note that an infinite set of xi values will satisfy the above equation. There is no unique solution. If we need a unique solution of an equation with n variables then we need a set of n such independent equations. This set of equations is known as system of simultaneous equations or system of equations

A system of n linear equations is represented generally as

a11x1+a12x2+….+a1nxn = b1

a21x1+a22x2+….+a2nxn = b2

……………………………………………

an1x1+an2x2+….+annxn = bn

In matrix form, the above equations can be written as:

Ax = b

Where A is an n\*n matrix, b is a an n vector and x is a vector of n unknowns.

The technique and methods for solving systems of linear algebraic equations belong to two fundamentally different approaches

1. Elimination Approach
2. Iterative Approach

Elimination approach, also known as direct method, educes the given system of equations to form which the solution be obtained by simple substitution. The commonly used elimination methods are:

1. Basic Gauss elimination method
2. Gauss elimination with pivoting
3. Gauss Jordon Method
4. LU decomposition method
5. Matrix inversion method

The solution of direct methods does not contain any truncation errors. However, they may contain round off errors due to floating point operations.

Iterative approach, as usual involves assumption of some initial values which are then refined repeatedly till they reach some accepted level of accuracy.

Existence of Solution

In solving systems of equations, we are interested in identifying values of the variables that satisfy all equations in the system simultaneously. Given an arbitrary systems of equations, it is difficult to say whether the system has a solution or not. Sometimes there may be a solution but it may not be unique. There are four possibilities.

1. System has a unique solution
2. System has no solution
3. System has a solution but not a unique one. That is, it has infinite solutions
4. System is ill conditioned

**Unique Solution**

Given the system

X+2y = 9

2x-3y = 4

The system has a solution

X=5 and y=2

Since no other pair of values of x and y would satisfy the equations, the solution is said to be unique.

**No solution**

The equations

2x-y=5

3x-3/2y=4

Have no solution. These two lines are parallel and therefore they never meet. Such equations are called inconsistent equations.

**No unique Solution**

The system

-2x+3y=6

4x-6y-12

has many different solutions. We see that these are two different forms of the same equations and therefore, they represent the same line. such equations are called dependent equations.

Ill conditioned System

There may a situation where the system has a solution but it is very close to being singular. For example, the system

x-2y = -2

0.45x-0.9y = -1

Has a solution but is very difficult to identify the exact point at which the line intersects. Such systems are said to be ill conditioned. Ill conditioned systems are very sensitive to round off errors and therefore, may pose problems during computation of the solution

**Solution by elimination**

Elimination is a method of solving simultaneous linear equations. This method involves elimination of a term containing one of the unknowns in all but one equation. One such step reduces the order of equations by one. Repeated elimination leads finally to one equation with one unknown. Some rules that are useful in manipulation of the equations are:

1. An equation can manipulated or divided by a constant
2. One equation can be added or subtracted from another equation.
3. Equations can be written in any order

**Naïve Gauss Elimination Method**

One of the most popular techniques for solving simultaneous linear equations is the Gauss elimination method. The approach is designed to solve a general set of n equations and n unknowns.

a11x1+a12x2+….+a1nxn = b1

a21x1+a22x2+….+a2nxn = b2

……………………………………………

an1x1+an2x2+….+annxn = bn

Gauss elimination consists of two stets

1. **Forward Elimination of Unknowns:** In this step, the unknowns are eliminated from each equation starting with the first equation.
2. **Back Substitution:** In this step, starting from the last equation, value of each of the unknowns is found.

**Forward Elimination of Unknowns**

In the first step of forward elimination, the first unknown x1, is eliminated from all rows below the first row. The first equation is selected as the pivot equation to eliminate x1. So, to eliminate x1 in the second equation, one divided the first equation by a11 (hence called pivot element) and then multiplies it by a21. This is the same as multiplying the first equation by a21 /a11 to give

a21x1+a12x2 +….+ a1nxn = b1

Now, this equation can be subtracted from the second equation to give

(a22 - b1

Or a’22x2+…+a’2nxn = b’2

This procedure of elimination x1 is now repeated for the third equation to the nth equation to reduce the set of equations as:

a11x1+a12x2+….+a1nxn = b1

a’22x2+a23x3+….+a’2nxn = b’2

……………………………………………

a’n2x1+an3x2+….+a’nnxn = b’n

This is the end of the first forward elimination. Now, for the second step of forward elimination, we start with the second equation as pivot equation and a’22 as the pivot element. So, to eliminate x2 in the third equation, one divides the second equation by a’22 (the pivot element) and then multiply it by a’32. This is the same as multiplying the second equation by a’32/a’22 and subtracting it from the third equation. This makes the coefficient of x2 zero in the third equation. The same procedure is now repeated for the fourth equation till the nth equation to give

a11x1+a12x2+….+a1nxn = b1

a’22x2+a23x3+….+a’2nxn = b’2

a’33x3+a’34x3+….+a’3nxn = b’3

……………………………………………

A’n3x3+….+a’’nnxn =b’’n

The next steps of forward elimination are conducted by using the third equation as a pivot equation and so on. That is, there will be a total of n-1 steps of forward elimination. At the end of (n-1)th steps of forward elimination, we get a set of equation that looks like

a11x1+a12x2+a13x3+…………..+a1nxn = b1

a’22x2+a23x3+….+a’2nxn = b’2

a’3x3+….+a’3nxn = b’3

……………………………………………

a(n-1)nnxn =bn (n-1)

Back Substitution

Now the equations are solved starting from the last equation as it has only one unknown.

xn = bnn(n-1)/ann (n-1)

Then the second last equation, that is the last (n-1)th equation has two unknowns: xn and xn-1, but xn is already known. This reduces the (n-1)th equation also to be unknown. Back substitution hence can be represented for all equations by the formula

xi = (bi(i-1) – )/aii(i-1)

and

xn = (bn(n-1)/ann(n-1)

Example 1: Use Naïve Gauss Elimination to solve

x1-3x2+x3 = 4

2x1-8x2+8x3 = -2

-6x1+3x2-15x3=9

Solution:

Given equations are

x1-3x2+x3 = 4

2x1-8x2+8x3 = -2

-6x1+3x2-15x3=9

Forward elimination of Unknowns

Step 1:

Multiply row 1 by 2 and subtract it from row 2. That is, perform (R2 = R2- 2R1)

The resulting equations are:

x1-3x2+x3 = 4

0x1-2x2+6x3 = -10

-6x1+3x2-15x3=9

Multiply row 1 by 6 and add it to row 3. That is, perform (R3 = R3+6R1)

The resulting matrix is:

x1-3x2+x3 = 4

0x1-2x2+6x3 = -10

0x1-15x2-9x3=33

Step 2:

Multiply row 2 by 15/2 and subtract it from row 3. That is, perform (R3 = R3-15/2R2}

The resulting equations are:

x1-3x2+x3 = 4

0x1-2x2+6x3 = -10

0x1-0x2+54x3=-108

This is end of the forward elimination steps

Back Substitution

We can now solve the above equations by back substitution. From the third row

54x3 = -108 🡪x3 = -2

Substituting the value of x3 in the second row

-2x2+6x3 = -10

Or -2x2+6\*-2 = -10🡪x2 = -1

Substituting the value of x3 and x2 in the first row

x1-3x2+x3 = 4

or x1-3\*(-1)-2=4

x1 = 3

Hence the solution is:

x1 = 3

x2 = -1

x3 = -2

**Example 2: Use Naïve Gauss elimination to solve:**

20x1+15x2+10x3 = 45

-3x1-2.249x2+7x3 = 1.751

5x1+x2+3x3 = 9

Solution:

…………………..

**C program for Naïve Gauss Elimination Method**

**…………………………………………………………………………….**

#include<stdio.h>

#include<conio.h>

#include<math.h>

int main()

{

int n,i,k,j,p,q;

float pivot, term, sum=0, a[10][10],b[10],x[10];

printf("Enter the dimension of the system of equations\n");

scanf("%d",&n);

printf("Enter the elements of coefficient matrix row wise\n");

for(i=0;i<n;i++)

{

for(j=0;j<n;j++)

{

scanf("%f",&a[i][j]);

}

}

printf("Enter RHS vector\n");

for(i=0;i<n;i++)

{

scanf("%f",&b[i]);

}

for(k=0;k<=n-2;k++)

{

pivot = a[k][k];

if(fabs(pivot)<0.0001)

printf("Method failed");

else

for(i=k+1;i<n;i++)

{

term = a[i][k]/pivot;

for(j=0;j<n;j++)

{

a[i][j] = a[i][j]-a[k][j]\*term;

}

b[i] = b[i]-b[k]\*term;

}

}

x[n-1]=b[n-1]/a[n-1][n-1];

for(i=n-2;i>=0;i--)

{

sum = 0;

for(j=i+1;j<=n-1;j++)

{

sum = sum+a[i][j]\*x[j];

}

x[i] = (b[i]-sum)/a[i][i];

}

printf("Solution\n");

for(i=0;i<n;i++)

{

printf("x%d = %f\t",i+1,x[i]);

}

getch();

return 0;

}

Output:

Enter the dimension of the system of equations

3

Enter the elements of coefficient matrix row wise

2 1 1

3 2 3

1 4 9

Enter RHS vector

10 18 16

Solution

x1 = 7.000000 x2 = -9.000000 x3 = 5.000000

**Drawbacks of Gauss Elimination Method**

**Division by zero:** Naïve Gauss elimination method may suffer from division by zero problems at the beginning of the each step of forward elimination. This occurs when pivot element is zero

**Round off Error:** The Naïve Gauss elimination method it prone to round off errors. This can be serious problem when there are large numbers of equations as errors propagate. Also, if there is subtraction of floating point numbers from each other, it may create errors.

**Gauss Elimination with Partial and Complete Pivoting**

Round off errors were large in case of Naïve Gauss Elimination method. One method of decreasing the round off error would be to use more significant digits, that is, use double or quad precision for representing the numbers. However, this would not avoid problem of division by zero present in the Naïve Gauss elimination method. To avoid division by zero as well as reduce round off error, Gaussian elimination with partial pivoting is the method of choice. Thus, partial pivoting is required due to following reasons.

* To avoid division by zero problem
* To reduce round off errors

Naïve Gauss elimination method and Gauss elimination with partial are the same, except in the beginning of each of forward elimination; a row switching is done based on the following criterion. If there are n equations, then there are n-1 forward elimination steps. At the beginning of the kth step of forward elimination, one finds the maximum of

|akk|, |ak+1,k|,….|ank|

Then if the maximum of these values if |apk| in the pth row, k≤p≤n, then switch rows p and k. The other steps of forward elimination are the same as the Naïve Gauss elimination method.

**Algorithm for Gauss Elimination with Partial Pivoting**

1. Start
2. Read Dimension of System of equations say n
3. Read coefficient matrix row-wise
4. Read RHS vector
5. Perform forward elimination as below

For k=1 to n-1

Find largest of a[p][k] for p=k, k+1, ….n

Swap row k and row p in coefficient matrix

Swap row k and p in RHS vector

Pivot = a[k][k]

For i=k+1 to n

Term = (a[i][k]/pivot)

Multiply row k of B matrix by “term” and subtract it from row i

Multiply row k of B matrix by “term” and subtract it from row i

End for

End for

1. Perform back substitution as below

X[n] = b[n]/a[n][n]

For i=n-1 to 1

Sum = 0

For j=i+1 to n

Sum sum+ a[i][j]\*x[j]

End for

X[i] = (b[i]-sum)/a[i][i]

End for

1. Display the solution vector
2. Stop

**Example:** Solve following systems of linear equations by using Gauss elimination with partial pivoting

2x1+x2+x3 = 5

4x1-6x2=-2

-2x1+7x2+2x3 = 9

**Solution:**

Representing above equations in matrix form we get

=

Forward Elimination of Unknowns

Step1:

Since, largest absolute value among a11, a21, and a31 is 4, switch row 1 and row 2. The resulting matrix is R1🡨🡪R2

=

Multiply Row 1 by ½ and subtract it from row 2. That is perform ( R2 = R2- 1/2R1)

The resulting matrix is:

=

Multiply row 1 by ½ and add it to row 2. That is perform: (R3 = r+1/2R1}

The resulting matrix is:

=

**Step 2:**

Since largest absolute value among a22, a32 is 4, we do not need to switch rows.

=

Now, subtract row 2 from Row 3. That is perform: (R3 = R3-R2}

The resulting matrix is:

=

This is the end of the forward elimination steps

Back Substitution

We can now solve the above equations by back substitution. From the third row,

x3 = 2

Substituting the value of x3 in the second row

4x2+x3=6

4x2+2 =6

🡪X2 = 1

Substituting the value of x3 and x2 in the first equation,

4x1-6x2=-2

🡪4x1-6\*1 = -2

🡪x1 = 1

Hence the solution is:

=

**Example: Use Gauss elimination with partial pivoting to solve following equations**

20x1+15x2+10x3=45

-3x1-2.249x2+7x3 = 1.751

5x1+x2+3x3 = 9

C program for Gauss Elimination with Pivoting

#include<stdio.h>

#include<conio.h>

#include<math.h>

int main()

{

int n,i,k,j,p,q,row;

float pivot, term, temp,largest,sum=0, a[10][10],b[10],x[10];

printf("Enter the dimension of the system of equations\n");

scanf("%d",&n);

printf("Enter the elements of coefficient matrix row wise\n");

for(i=0;i<n;i++)

{

for(j=0;j<n;j++)

{

scanf("%f",&a[i][j]);

}

}

printf("Enter RHS vector\n");

for(i=0;i<n;i++)

{

scanf("%f",&b[i]);

}

for(k=0;k<=n-2;k++)

{

largest = fabs(a[k][k]);

for(p=k+1;p<=n-1;p++)

if(fabs(a[p][k])>largest)

{

largest = fabs(a[p][k]);

row = p;

}

for(p=0;p<n;p++)

{

temp = a[k][p];

a[k][p]=a[row][p];

a[row][p]=temp;

}

temp = b[k];

b[k]=b[row];

b[row]=b[p];

pivot = a[k][k];

if(fabs(pivot)<0.0001)

printf("Method failed");

else

for(i=k+1;i<n;i++)

{

term = a[i][k]/pivot;

for(j=0;j<n;j++)

{

a[i][j] = a[i][j]-a[k][j]\*term;

}

b[i] = b[i]-b[k]\*term;

}

}

x[n-1]=b[n-1]/a[n-1][n-1];

for(i=n-2;i>=0;i--)

{

sum = 0;

for(j=i+1;j<=n-1;j++)

{

sum = sum+a[i][j]\*x[j];

}

x[i] = (b[i]-sum)/a[i][i];

}

printf("Solution\n");

for(i=0;i<n;i++)

{

printf("x%d = %f\t",i+1,x[i]);

}

getch();

return 0;

}

Output:

Enter the dimension of the system of equations

3

Enter the elements of coefficient matrix row wise

2 2 1

4 2 3

1 1 1

Enter RHS vector

6 4 0

Solution

x1 = 3.000000 x2 = -1.000000 x3 = -2.000000

**Gauss Jordon Method**

It is a variation of Gauss elimination. The difference between Gauss Elimination method and Gauss Jordan method are described below

* When unknown is eliminated from an equation, it is also eliminated from all other equations. All rows are normalized by dividing them by their pivot element. Hence, the elimination step results in an identity matrix rather than a triangular matrix
* Back substitution is not required. We can obtain solution directly from identity matrix obtained from elimination step

All other techniques developed for Gauss elimination are still valid for Gauss Jordon elimination. However, Gauss Jordon requires more computational work than Gauss elimination. The goal in Gauss Jordan elimination is to use row operations. In Gauss elimination method, a variable is eliminated below the pivot equation. But in Gauss Jordan method, it is eliminated from all other rows (both below and above). This process thus eliminated all the off diagonal terms producing a diagonal matrix rather than a triangular matrix. Further, all rows are normalized by dividing them by their pivot elements. Consequently, we can obtain the values of unknowns directly from b vector, without employing back substitution.

The difference between Gauss elimination and Gauss Jordan method is as shown below

=

= =

Result of Gauss elimination Result of Gauss Jordan elimination

**Algorithm for Gauss Jordan Elimination**

1. Start
2. Normalize the first equation by dividing it by its pivot element
3. Eliminate x1 term from all the other equations
4. Now, normalize the second equation by dividing it by its pivot element
5. Eliminate x2 from all the equations, above and below the normalized pivotal equation
6. Repeat this process until xn is eliminated from all but the last equation
7. The resultant b vector is the solution vector
8. Stop

The Gauss Jordan method requires approximately 50 percent more arithmetic operations compared to Gauss method. Therefore this method is rarely used

**Example: Solve the system**

**2x1+4x2-6x3 = -8**

**x1+3x2+x3 =10**

**2x1-4x2-2x3 = -12**

using Gauss Jordan method

Solution:

Normalize the first equation by dividing it by 2 (pivot element). The result equations set is

x1+2x2-3x3 = -4

x1+3x2+x3 =10

2x1-4x2-2x3 = -12

Step2: Eliminate x1 from the second subtracting 1 time the first equation from it. Similarly, eliminate x1 from the third equation by subtracting 2 times the first equation from it. The result is:

x1+2x2-3x3 = -4

0+x2+4x3 =14

0-8x24x3 = -4

Step 3

Normalize the second equation. (It is already in normalized form)

Step 4: Following similar approach, eliminate x2 from first and third equations. This gives:

x1+0-11x3 = -32

0+x2+4x3 = 14

0+0+36x3 = 108

Step 5: Normalize the third equation by dividing it by 36

x1+0-11x3 = -32

0+x2+4x3 = 14

0+0+x3 = 108

Step 6:

Eliminate x3 from the first and second equations. We get

x1+0+0 = 1

0+x2+0 = 2

0+0+x3 = 3

…………………

**C program for Gauss Jordan Method**

#include <stdio.h>

#include <stdlib.h>

#include<math.h>

int main()

{

int n,i,k,j,p,q;

float pivot, term, a[10][10];

printf("Enter dimension of system of equations\n");

scanf("%d",&n);

printf("Enter coefficient augmented matrix\n");

for(i=0;i<n;i++)

{

for(j=0;j<n+1;j++)

{

scanf("%f",&a[i][j]);

}

}

for(k=0;k<n;k++)

{

pivot = a[k][k];

for(p=0;p<n+1;p++)

a[k][p]=a[k][p]/pivot;

for(i=0;i<n;i++)

{

term = a[i][k];

if(k!=i)

{

for(j=0;j<n+1;j++)

{

a[i][j] = a[i][j]-a[k][j]\*term;

}

}

}

}

printf("Solution Vector\n");

for(i=0;i<n;i++)

{

printf("x%d = %f\n",i+1,a[i][3]);

}

getch();

return 0;

}

Output:

Enter dimension of system of equations

3

Enter coefficient augmented matrix

2 4 -6 -8

1 3 1 10

2 -4 -2 -12

Solution Vector

x1 = 1.000000

x2 = 2.000000

x3 = 3.000000

**Matrix Inversion Method**

Although Gauss Jordan method is more time consuming than Gauss Elimination method, it provides simple approach for computing inverse of a method. A matrix X is said to be inverse of M is MX = XM =I, where I is identity matrix of same order as that of matrix M. Matrix inverse can be computed by using Gauss Jordan method in the following two steps:

**Step 1:** Augment the coefficient matrix with identity matrix as below

**Step 2:** Apply Gauss Jordan method to the augmented matrix to reduce coefficient matrix to indemnity matrix as below

Now the right hand side of above augmented matrix is the inverse of original coefficient matrix.

**Algorithm:**

1. Start
2. Read dimension of Matrix say n
3. Read coefficient matrix row-wise
4. Augment the coefficient matrix by identity matrix
5. Perform elimination operation as below

For k=1 to n

Pivot = a[k][k]

For p =1 to 2n

a[k][p] = a[k][p]/pivot

end for

for i=1 to n

term = a[i][k]

if(i!=k)

multiply row k by “term” and subtract it from row i

End for

End for

1. Display inverse matrix which is second half of augmented matrix
2. Stop

**Write a C program to find the inverse of matrix of using Gauss Jordan Method**

#include<stdio.h>

#include<conio.h>

#include<math.h>

int main()

{

int n,i,j,k,p,q;

float pivot, term, a[10][10];

printf("Enter dimension of System of equations\n");

scanf("%d",&n);

printf("Enter coefficients matrix\n");

for(i=0;i<n;i++)

{

for(j=0;j<n;j++)

{

scanf("%f",&a[i][j]);

}

}

for(i=0;i<n;i++)

{

for(j=n;j<2\*n;j++)

{

if(i==j-n)

a[i][j]=1;

else

a[i][j]=0;

}

}

for(k=0;k<n;k++)

{

pivot = a[k][k];

for(p=0;p<2\*n;p++)

a[k][p]=a[k][p]/pivot;

for(i=0;i<n;i++)

{

term = a[i][k];

if(k!=i)

for(j=0;j<2\*n;j++)

{

a[i][j]=a[i][j]-a[k][j]\*term;

}

}

}

printf("Inverse Matrix\n");

for(i=0;i<n;i++)

{

for(j=n;j<2\*n;j++)

{

printf("%0.2f\t",a[i][j]);

}

printf("\n\n");

}

getch();

return 0;

}

**Output:**

Enter dimension of System of equations

3

Enter coefficients matrix

2 1 1

3 2 3

1 4 9

Inverse Matrix

-3.00 2.50 -0.50

12.00 -8.50 1.50

-5.00 3.50 -0.50

**Example: Find the inverse of the following matrix:**

**Solution:**

Step 1:

Augmented Matrix is:

Step 2:

Perform the operation R1 = 1/7R1

Step 3: Perform operation R2 = R2-5R1

Step 4: Perform operation R2 = -7R2

Step 5: Perform operation R1 = R1-3/7R2

Therefore, the inverse of matrix A is:

**Example 2:** Find the inverse of the matrix

Solution:

Step 1:

The augmented matrix is:

Step 2:

Perform the operation R2 = R2-2R1

Step 3: Perform the operation R2 = 1/5R2

Sep 4: Perform the operations R1 = R1+R2 and R3 = R3+2R2

Step 4: Perform the operation R3 = 5R3

**Step 5:**

Perform the operation R1 = R1-3/5R3 and R2 = R2+2/5R3

Therefore, the inverse of the matrix A is:

**C program for matrix Inversion by using Gauss Jordan Method**

#include<stdio.h>

#include<conio.h>

#include<math.h>

int main()

{

int n,i,k,j,p,q;

float pivot, term, a[10][10];

printf("Enter dimension of System of equations\n");

scanf("%d",&n);

printf("Enter coefficient matrix\n");

for(i=0;i<n;i++)

{

for(j=0;j<n;j++)

{

scanf("%f",&a[i][j]);

}

}

for(i=0;i<n;i++)

{

for(j=n;j<2\*n;j++)

{

if(i==j-n)

a[i][j]=1;

else

a[i][j]=0;

}

}

for(k=0;k<n;k++)

{

pivot = a[k][k];

for(p=0;p<2\*n;p++)

a[k][p]=a[k][p]/pivot;

for(i=0;i<n;i++)

{

term = a[i][k];

if(k!=i)

for(j=0;j<2\*n;j++)

{

a[i][j]=a[i][j]-a[k][j]\*term;

}

}

}

printf("Matrix Inverse\n");

for(i=0;i<n;i++)

{

for(j=n;j<2\*n;j++)

{

printf("%f ",a[i][j]);

}

printf("\n");

}

getch();

return 0;

}

Output:

Enter dimension of System of equations

3

Enter coefficient matrix

2 1 1

3 2 3

1 4 9

Matrix Inverse

-3.000000 2.500000 -0.500000

12.000000 -8.500000 1.500000

-5.000000 3.500000 -0.500000

**Matrix Factorization**

The coefficient matrix A of a system of linear equations can be factorized (or decomposed) into two triangular matrices L and U such that

A = LU

Where

L =

And

U =

L is known as lower triangular matrix and U is known as upper triangular matrix

Once A is factored into L and U, the system of equations AX = C

Can be written as

LUX = C

Multiplying both sides by L-1 we get

L-1LUX = L-1C

IUX = L-1C

Let L-1C = Z

The we have

UX = Z

We also use matrix factorization methods to solve system of linear equations. We can solve system of equations in two steps:

1. Solve equation (1) first for [Z] by using the forward substitution
2. Use equation (2) to calculate the solution vector [x] by back substitution

**Doolittle LU Decomposition**

Coefficient matrix A of a system of linear equations can be decomposed into triangular matrices L and U such that

\*

If L has 1’s on it’s diagonal then it is called Doolittle factorization, thus Doolittle algorithm assume that l11 = 1, l22 = 2,..,lnn=1

From the above matrices,

a11 = l11.u11 🡪u11 = a11

a12 = l11.u12 ->>u12 = a12

……

a1n = l11u1n 🡪u1n = a1n

a21 = l21u11 🡪l21 = a21/a11

a22 = l21u12+l22u22 🡪u22 = a22-l21u12

…..

a2n = …a2n-l21u1n

Algorithm for LU matrix factorization using Doolittle LU Decomposition

1. Start
2. Read dimension of matrix say n
3. Read elements of matrix row-wise
4. Assign values to first row of U matrix as below

For j=1 to n

U[1][j]=a[1][j]

End for

1. Assign values to first row of L matrix as below

For i=1 to n

L[i][i]=1

End for

1. Compute and assign values to first column of L matrix as below

for j=2 to n

l[i][1] = a[i][1]/[u[1][1]

End for

1. Compute and assign values to 2nd to nth rows of L and U as below
2. For j=2 to n

For i=2 to j

U[i][j] = a[i][j]-

End for

For i=j+1 to n

L[i][j] = {a[i]j]-

End for

End for

1. Display L and U Matrices
2. Stop

**C program for matrix factorization using Do Little LU Decomposition**

#include<stdio.h>

#include<conio.h>

#include<math.h>

int main()

{

int n, i,j,k;

float sum, a[10][10],b[10],x[10],z[10],u[10][10],l[10][10];

printf("Enter dimension of System of equations\n");

scanf("%d",&n);

printf("Enter coefficient matrix\n");

for(i=0;i<n;i++)

{

for(j=0;j<n;j++)

{

scanf("%f",&a[i][j]);

}

}

printf("Enter RHS vector\n");

for(i=0;i<n;i++)

{

scanf("%f",&b[i]);

}

printf("Computing elements of L and U matrix...\n");

for(j=0;j<n;j++)

{

u[0][j]=a[0][j];

}

for(i=0;i<n;i++)

l[i][i]=1;

for(i=1;i<n;i++)

l[i][0]=a[i][0]/u[0][0];

for(j=1;j<n;j++)

{

for(i=1;i<=j;i++)

{

for(k=0;k<=i-1;k++)

{

sum = sum+(l[i][k]\*u[k][j]);

}

u[i][j]= a[i][j]-sum;

sum=0;

}

for(i=j+1;i<n;i++)

{

for(k=0;k<=j-1;k++)

{

sum = sum+(l[i][k]\*u[k][j]);

}

l[i][j] = (a[i][j]-sum)/u[j][j];

sum = 0;

}

}

printf("Solving for Z using Forward Substitution..\n");

getch();

z[0]=b[0];

for(i=1;i<n;i++)

{

for(j=0;j<i;j++)

sum = sum+(l[i][j]\*z[j]);

z[i] = b[i]-sum;

sum = 0;

}

printf("Solving for X using Backward Substitution...\n");

getch();

x[n-1]=z[n-1]/u[n-1][n-1];

for(i=n-2;i>=0;i--)

{

for(j=i+1;j<n;j++)

sum = sum+(u[i][j]\*x[j]);

x[i] = (z[i]-sum)/u[i][i];

sum = 0;

}

printf("Solution:\n");

for(i=0;i<n;i++)

{

printf("x%d = %f\t",i+1,x[i]);

}

getch();

return 0;

}

**Example:** Factorize the following matrix using Doolittle LU decomposition

A=

Solution:

A = LU = \*

Where

u11 = a11 = 25

u12 = a12 = 5

u13 = a13 =1

l21 = a21/u11 = 64/25 = 2.56

u22 = a22-l21\*u12 = 8-2.25\*5 = -4.81

u23 = a23-l21.u13 = 1-2.56\*1 = -1.56

l31 = a31/u11 = 144/25 = 5.76

l32 = (a32-l31.u12)/u22 = (12-5.76\*5)/4,8 = 3.5

u33 = (a33-l31\*u13-l32\*u23) = 1-5.76\*1-3.5\*(-1.56) = 0.7

Now,

A = LU = \*

**Example : Solve the following system of equations by using Doolittle LU Decomposition method**

**3x1 + 2x2 + x3 = 10**

**2x1 + 3x2 + 3x3 = 14**

**X1 + 2x2 + 3x3 = 14**

**Solution:**

**Coefficient Matrix**

**A =**

**Decompose the coefficient matrix using Doolittle decomposition as below**

**A = LU = \***

**u11 = a11 = 3**

**u12 = a12 =2**

**u13 = a13 =1**

**l21 = a21/u11 = 2/3**

**u22 = a22-l21\*u12 = 3-2/3\*2 = 5/3**

**u23 = a23-l21.u13 =2-2/3 = 4/3**

**l31 = a31/u11 = 1/3**

**l32 = (a32-l31.u12)/u22 = (2-2/3)/(5/3) = 4/5**

**u33 = (a33-l31\*u13-l32\*u23) = 3-1/3-16/15 = 24/15 = 8/5**

**Thus**

**A = LU = \***

**Now we solve LZ = C by using forward elimination. That is, solve**

**=**

**We get,**

**Z1 = 10**

**Z2 = 14 – 2/3 \*z1 = 22/3**

**Z3 = 14 – 1/3\*z1 – 4/5\*z2 = 14-10/3-88/15 = 72/15 = 24/5**

**Again solve UX = Z using backward substitution method**

**=**

**We get**

**x1 = 3**

**x2 = (22/3-4/3\*3)/5/3 = 2**

**x3 = (10-2\*2-1\*3)/3 = 1**

**Cholesky Method**

For many practical systems of linear equations, the coefficients matrix **A** is symmetric. In such case, Cholesky method can be used to factorize the coefficient matrix and so solve systems of linear equations in more efficient way. If coefficient matrix is symmetric the upper factor is transpose of lower factor or vice versa. Thus, we can factorize matrix A as:

A = LLT or A = UT U

Thus,

A = \*

Like Doolittle LU decomposition the matrices of right hand side and comparing it with coefficient matrix of left hand side, we get

uii = for i=1,2…n

uij =

Example: Factorize the given matrix using Cholesky Factorization

A =

Solution:

Let,

A =LU = \*

Where

u11 = = = 1

u12 = a12/u11 = 4/1 = 4

u13 - = a13/u11 = 7/1 = 7

u22 = 2 = = 8

u33 = √(a33-a213-u223) = √(89-49-4) = 6

Thus two factors of coefficient matrix are:

A = LU = \*

Example: Solve the following system of equations by using Cholesky decomposition technique

4x1+10x2+8x3 = 44

10x1+26x2+26x3 = 128

x1+26x2+61x3 = 214

Solution:

Let

A = LU = \*

Where

u11 =√a11 = √4 =2

u12 = a12/u11 =10/2 = 5

u13 =a13/2 = 4

u22 = √(a22-u212) = √(26-52) = 1

u23 = = 26-5\*4 = 1

u33 = √(a33-u213-u223) =√(61-42-62) = 3

Thus two factors of coefficient matrix are:

A = LU = \*

Now, solve the equation

LZ = C

=

This gives,

z1 = 22

z2 = 18

z3 = 6

Now we solve the equation

=

Thus, we get,

x1 = -8

x2 = 6

x3 = 2

**Iterative Solution of Linear Equations**

Direct methods for solving system of equations pose some problems when the systems grow larger or when most of the coefficients are zero. They require prohibitively large number of floating point operations and therefore, not only become time consuming but also severely affect the solution due to round off errors. In such cases, iterative methods provide an alternative. For example, ill conditioned systems can be solved by iterative methods without facing the problems of round off errors.

The following three iterative methods are used

1. Jacobi Iteration method
2. Gauss Seidel Iteration method
3. Successive over relaxation method

**Jacobi Iteration Method**

Jacobi method is one of the simplest iterative methods. The basic idea behind this method is essentially the same as that for the fixed point iteration method. Let us consider an equation of the form

f(x) = 0

can be rearranged into a form

x= g(x)

The function g(x) can be evaluated iteratively using an initial approximation x as follows:

xi+1 = g(xi)

Jacobi method extends this idea to a system of equations. It is a direct substitution method where the values of unknowns are improved by substituting directly the previous values

Let us consider a system of n equations in n unknowns

a11x1+a12x2+….+a1nxn = b1

a21x1+a22x2+….+a2nxn = b2

……………………………………………

an1x1+an2x2+….+annxn = bn

we can rewrite the original systems as

x1 =

x2 =

…………………………………………………..

xn =

Now, we can compute x1,x2,…xn by using guesses for these values. These new values are again used to compute the next set of x values. The process can continue till we obtain a desired level of accuracy in the x values.

In general, iteration for x1 can be obtained from the ith equation as follows:

xi(k+1) =

**Example:** Obtain the solution of the following system using Jacobi iteration method

2x1+x2+x3 = 5

3x1+5x2+2x3 = 15

2x1+x2+4x3 = 8

Solution:

The given set of equations is:

2x1+x2+x3 = 5

3x1+5x2+2x3 = 15

2x1+x2+4x3 = 8

First, solve the equations for unknowns as the diagonal. That is,

x1 =

x2 =

x3 =

If we assume the initial values of x1 , x2 and x3 to be zero, then we get

Iteration 1:

x1 = = 2.5

x2 = = 3

x3 = = 2

Iteration 2:

x1 = = 0

x2 = = 0.7

x3 = = 0

Iteration 3:

x1 = = 2.15

x2 = = 3

x3 = = 1.825

Iteration 4:

x1 = = 0.0875

x2 = = 1.222

x3 = = 0.175

The process continued till the values of x reach a desired level of accuracy.

**Algorithm for Jacobi Iteration Method**

1. Start
2. Read dimension of system of equations say n
3. Read coefficients of matrix row-wise
4. Read elements of RHS vector
5. Read accuracy limit say E
6. Read initial guess

For i=1 to n

Old\_x[i] = 0

End for

1. Compute new set of approximate root as below

For i=1 to n

Sum = b[i]

For j=1 to n

If(i!=j)

Sum = sum-a[i][j] \*old\_x[j]

End for

New\_x[i] = sum/a[i][i]

E[i] = abs(new\_x[i]-old\_x[i])/new\_x[i]

End for

1. Compute error with specified precision

For i=1 to n

If(E[i]>E

For j=1 to n

Old\_x[j] = new\_x[j];

End for

Got to step 7

End for

1. Display results in new\_x vector
2. Stop

**C program for Jacobi method**

#include<stdio.h>

#include<conio.h>

#include<math.h>

/\* Arranging given system of linear

equations in diagonally dominant

form:

20x + y - 2z = 17

3x + 20y -z = -18

2x - 3y + 20z = 25

\*/

/\* Equations:

x = (17-y+2z)/20

y = (-18-3x+z)/20

z = (25-2x+3y)/20

\*/

/\* Defining function \*/

#define f1(x,y,z) (17-y+2\*z)/20

#define f2(x,y,z) (-18-3\*x+z)/20

#define f3(x,y,z) (25-2\*x+3\*y)/20

int main()

{

float x0=0, y0=0, z0=0, x1, y1, z1, e1, e2, e3, e;

int count=1;

printf("Enter tolerable error:\n");

scanf("%f", &e);

printf("\nCount\tx\ty\tz\n");

do

{

x1 = f1(x0,y0,z0);

y1 = f2(x0,y0,z0);

z1 = f3(x0,y0,z0);

printf("%d\t%0.4f\t%0.4f\t%0.4f\n",count, x1,y1,z1);

e1 = fabs(x0-x1);

e2 = fabs(y0-y1);

e3 = fabs(z0-z1);

count++;

x0 = x1;

y0 = y1;

z0 = z1;

}while(e1>e && e2>e && e3>e);

printf("\nSolution: x=%0.3f, y=%0.3f and z = %0.3f\n",x1,y1,z1);

getch();

return 0;

}

**Ill Conditioned Systems**

Systems where small changes in the coefficient result in large deviations in the solution are said to be ill conditioned systems. A wide range of answers can satisfy such equations. This means that a completely erroneous set of answers may produce zero or near zero residuals

Ill conditioned systems are very sensitive to round off errors. These errors during computing process may induce small changes in the coefficients which in turn, may result in a large error in the solution.

We can decide the condition of a system either graphically or mathematically. Graphically, if two lines appear almost parallel, then we can say the system is ill conditioned, since it is hard to decide just at which point they intersect.

The problem of ill-condition can be mathematically described as follows. Consider a two equation system

a11x1+a12x2 = b1

a21x1+a22x2 = b2

if these two lines are almost parallel, their slopes must be nearly equal. That is,

=

Alternatively,

a11a22 = a12a21

Or a11a22 - a12a21=0

Note that a11a22 - a12a21 is the determinant of the coefficient matrix

A =

This shows that the determinant of an ill conditioned system is very small or nearly equal to zero

It partial pivoting technique, we try to interchange the rows so that the largest element becomes the pivot element. This is done basically to avoid a division by zero or nearly zero point. Even the largest element in that column may happen to be zero (or nearly zero). Such situations arise when the systems are ill-conditioned. Solution of these systems may not be meaningful.

**Example:** Solve the following systems

2x1+x2=25

2.001x1+x2 = 25.01

And thereby discuss the effect of ill-conditioning.

Solution:

x1 = = 10

x2= = 5

Let us change the coefficient of x1 in the second equation to 2.0005. Now the values of x1 and x2 are

x1 = = 20

x2= = -15

Now, let us compare the results. A small change in one of the coefficients has resulted in a large change in the result.

**Gauss Seidel Method**

Gauss Seidel method is also iterative approach for solving system of linear equations. It is similar to Jacobi iteration method only the difference is that the Jacobi method, the values of xi (k) obtained in the kth iteration remains unchanged until the entire (k+1)th iteration has been calculated. But with the Gauss Seidel method, we use the new values as soon as they are known.

Let us consider a system of n equations in n unknowns

a11x1+a12x2+….+a1nxn = b1

a21x1+a22x2+….+a2nxn = b2

……………………………………………

an1x1+an2x2+….+annxn = bn

we can rewrite the original systems as

x1 =

x2 =

…………………………………………………..

xn =

Now, we start with initial guesses and use above equations to find new estimates for xi’s . Remember, unlike, Jacobi iteration Gauss Seidel method always uses the most recent estimates to calculate the next estimates for xi. This means Gauss Seidel method is used to find new values of unknown as below.

**Algorithm**

1. Start
2. Read dimension of System of equations, say n
3. Read coefficients of matrix row-wise
4. Read elements of RHS vector
5. Read accuracy limits say E
6. Set initial guesses as below

For i=1 to n

New\_x[i]=0

End for

1. Compute new estimate as below

For i=1 to n

Sum = b[i]

For j=1 to n

If(i≠j)

Sum = sum-a[i]\*new\_x[i]

New\_x[i]= sum/a[i][i]

E[i] =

End for

1. Check for tolerance level as below

For i=1 to n

If (E[i]>e)

Go to step 7

1. Display Result in “new\_x” vector
2. Stop

Example: Use Gauss Seidel method to obtain the solution of the following equations:

6x1-2x2+x3=11

x1+2x2-5x3=-1

-2x1+7x2+2x3 = 5

Solution:

The given set of equations is:

6x1-2x2+x3=11

x1+2x2-5x3=-1

-2x1+7x2+2x3 = 5

Step 1:

Rewrite the equations such that each equation has the unknown with largest coefficient on the left hand side. (Not necessarily always)

x1 =

x2 =

x3 =

Step 2: Assume the initial guesses (x2)0 = (x3)0 = 0 and then calculate (x1)0 as

(x1)1 = = 1.833

Use (x1)1 = 1.833 and (x3)0 =0 to calculate (x2)1 as

(x2)1 = = 1.238

Similarly, use (x1)1 =1.833 and (x2)1 = 1.238 to calculate (x3)1

(x3)1 = = 1.062

Step 3:

Repeat the same procedure for the 2nd iteration

(x1)2 = = 2.069

Use (x1)2 = 2.069 and (x3)0 =1.062 to calculate (x2)1 as

(x2)2 = = 1.002

Similarly, use (x1)2 =2.0562 and (x2)2 = 1.002 to calculate (x3)2

(x3)1 = = 1.015

And continue the above iterative procedure until {(xk)i+1-(xk)i}/(xk)i+1<e for k=1,2, and 3. The results for 5 iterations are

|  |  |  |  |
| --- | --- | --- | --- |
| Iteration | x1 | x3 | x3 |
| 1 | 1.833 | 1.238 | 1.062 |
| 2 | 2.069 | 1.002 | 1.015 |
| 3 | 1.998 | 0.995 | 0.998 |
| 4 | 1.999 | 1.000 | 1.000 |
| 5 | 2.000 | 1.000 | 1.000 |

Thus, the solution is x1 = 2, x2 = 1, and x3=1

Example 2: Solve the following system of equations using Gauss Seidel method

5x1-2x2+3x3 = -1

-3x1+9x2+x3=2

2x1-x2-7x3 = 3

Solution:

The given set of equations are:

5x1-2x2+3x3 = -1

-3x1+9x2+x3=2

2x1-x2-7x3 = 3

From the given equations, we can obtain

x1 =

x2 =

x3 =

Let initial guess are x1 = 0, x2=0 and x3=0

|  |  |  |  |
| --- | --- | --- | --- |
| Iteration | x1 | x2 | x3 |
| 1 | 0 | 0 | 0 |
| 2 | -0.200 | 0.1554 | -0.507 |
| 3 | 0.1664 | 0.334 | -0.428 |
| 4 | 0.190 | 0.333 | -0.421 |
| 5 | 0.185 | 0.331 | -0.422 |
| 6 | 0.185 | 0.331 | -0.422 |

C Program for Gauss Seidel Method

……………………………………………………….

**Eigen Vectors and Eigen Values of a Matrix**

The Eigen vectors of a square matrix are non-zero vectors that always remains proportional to original vector when multiplied by the matrix. This means any vector x that satisfies the equation Ax = λx, is called eigen vector. Here A is the matrix , x is the eigen vector and λ is the associated eigen value. Eigenvectors of a matrix A are exceptional vectors x that are in the same direction as that of a Ax. Eigenvectors are not unique in the sense that any eigenvector can be multiplied by a constant to form another eigenvector. For d each eigenvector there is only one associated eigenvalue.

**Power Method**

Eigen values can be ordered in terms of their absolute values to find the dominant or largest eigen value of a matrix. Thus, if two distinct hypothetical matrices have the following set of eigen values

* 5, 8,-7 then |8|>|-7|>5 and 8 is the dominant eigen value
* 0.2, -1, 1 then |1| = [-1|>|0.2| and since |1| = |-1| there is not dominant eigenvalue

One of the simplest methods for finding the largest eigen value and eigen vector of a matrix is the power method, also called the Vector Iteration Method. The method fails if there is no dominant eigenvalue. It uses iterative approach and starts with initial guess for vector x. Power method can be implemented as follows:

Y = AX

X = Y

New value of X can be obtained by using equation (2) and then value of Y is calculated by using equation (1). This process is repeated until the desired level of accuracy is obtained. The parameter k is known as sealing factor and it is the element of Y with the largest magnitude.

Inverse power method, which is similar to power method, is used to calculate the smallest eigenvalue and its corresponding eivenvector. Here, we simply use the property: If AX = λX

then

A-1 = X

**Example:** Find the dominant eigenvalue and corresponding eigen vectors of the matrix given below by using power method.

A =

Solution: Assume that initial guess for eigenvector as

X =

**Iteration 1:**

Y = AX = =

🡪k=2. New value of X can be calculated as:

X = Y = 1/2 =

**Iteration 2:**

Y = AX = =

k=2. New value of X can be calculated as:

X = Y = =

**Iteration 3:**

Y = AX = =

k=2.8 New value of X can be calculated as:

X = Y = =

**Iteration 4:**

Y = AX = =

k=2.93 New value of X can be calculated as:

X = Y = =

**Iteration 5:**

Y = AX = =

k=2. 99 New value of X can be calculated as:

X = Y = =

**Iteration 6:**

Y = AX = =

k=3. New value of X can be calculated as:

X = Y = =

Thus, the largest eigenvalue is 3, and the corresponding eigenvector is

X =

**Example 2:** Find the smallest eigenvalue and corresponding eigenvector of the matrix given below by using power method. Note that finding the smallest eigenvalue of A is equivalent to finding the dominant eigenvalue of A-1

A =

Solution:

Find the inverse of given matrix:

A =

We can easily obtain inverse of A by using Gauss Jordan method

B = A-1 =

Now, we use power method to find the dominant eigen value of A-1

Assume initial Eigen vector is

Iteration 1:

Y = BX = =

🡪k=1. New value of X can be calculated as

X = Y = = =

Iteration 2:

Y = BX = =

🡪k=1. New value of X can be calculated as

X = Y = = =

Iteration 2:

Y = BX = =

🡪k=1. New value of X can be calculated as

X = Y = = =

Iteration 3:

Y = BX = =

🡪k=1. New value of X can be calculated as

X = Y = = =

Iteration 4:

Y = BX = =

🡪k=1. New value of X can be calculated as

X = Y = = =

Algorithm for power method

1. Start
2. Read dimension of matrix say n
3. Read elements of matrix row-wise
4. Read guess vector say x
5. Compute Y vector as
6. Set k =
7. Stop