Assignment - 1
ARNAB BIR - 14MA20009

Gauss-Seidel Method:

Introduction: Gauss-Seidel method is an iterative method used to find the solution of a system of linear simultaneous equations using fixed point iteration method. Some initial assumptions are made for the initial values of the variables and then this algorithm is implemented until the relative change in the value of the variable reaches some limiting value (er).

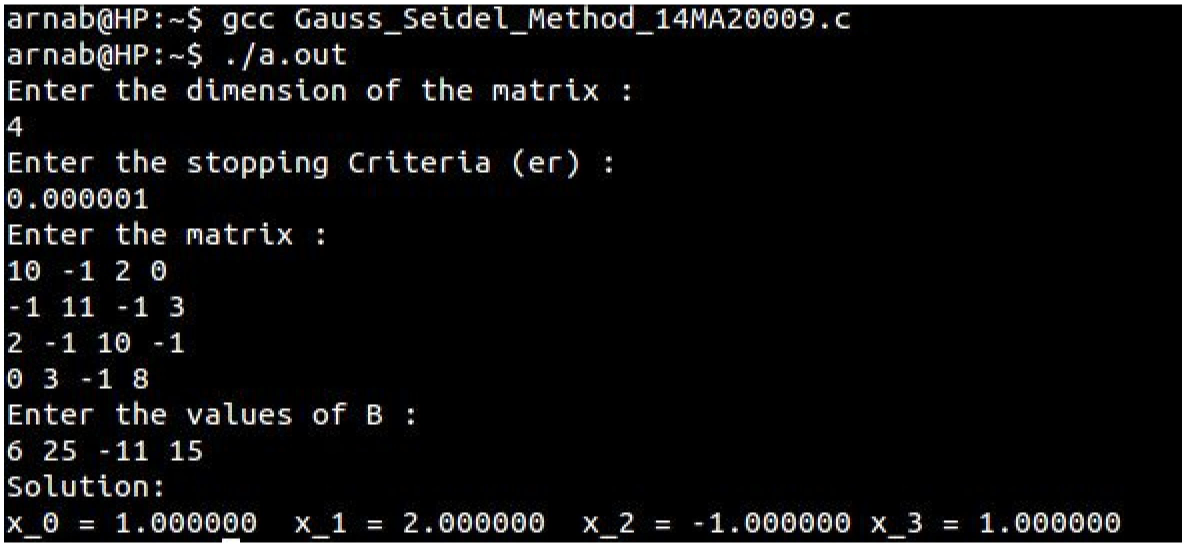
Convergence: The Gauss-Seidel method converges for any starting vector x if the coefficient matrix is **diagonally dominant**. Otherwise the solution diverges. Hence all system of equations can not be solved by Gauss-Seidel method.

Code:

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

int main(){
      int key, i, j, k, n;
      double er, sum;
      printf("Enter the dimension of the matrix :\n");
      scanf("%d", &n);

      printf("Enter the stopping Criteria (er) :\n");
      scanf("%lf", &er);

      printf("Enter the matrix : \n");
      double a[n][n];
      for(i = 0; i < n; ++i)
            for(j = 0; j < n; ++j)
                  scanf("%lf", &a[i][j]);

        printf("Enter the values of B : \n");
        double b[n];
      for(i = 0; i < n; ++i)
```

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

      double x[n];

      double x0[n];
      for(i = 0; i < n; ++i)
                  x0[i] = 0;
      do{
            key = 0;

            for(i = 0; i < n; ++i){
                  sum = b[i];
                  for(j = 0; j < n; ++j)
                        if(j != i)
                              sum = sum - a[i][j] * x0[j];

                  x[i] = sum / a[i][i];
                  if(fabs((x[i] - x0[i]) / x[i]) > er){
                        key = 1;
                        x0[i] = x[i];
                  }
            }

      }while(key == 1);

      printf("Solution:\n");
      for(i = 0; i < n; ++i)
            printf("x_%d = %lf\t", i, x[i]);
      printf("\n");
      return 0;
}
```

Result:



```
arnab@HP:~$ gcc Gauss_Seidel_Method_14MA20009.c
arnab@HP:~$ ./a.out
Enter the dimension of the matrix :
4
Enter the stopping Criteria (er) :
0.000001
Enter the matrix :
10 -1 2 0
-1 11 -1 3
2 -1 10 -1
0 3 -1 8
Enter the values of B :
6 25 -11 15
Solution:
x_0 = 1.000000  x_1 = 2.000000  x_2 = -1.000000 x_3 = 1.000000
```

Basic Solutions Using Gauss Seidel Method:

In this problem, while finding each solution, we assign (n - m) number of variables equal to 0 and rest m variables are calculated accordingly. Each solution represents a basic solution for the system of equations. Hence, during each iteration we select m out of n variables and solve the system of m equations and m unknowns. The function **makeDominant** is used to make the m x m matrix diagonally dominant. In case it is not possible, we can not get any basic solution using this algorithm.

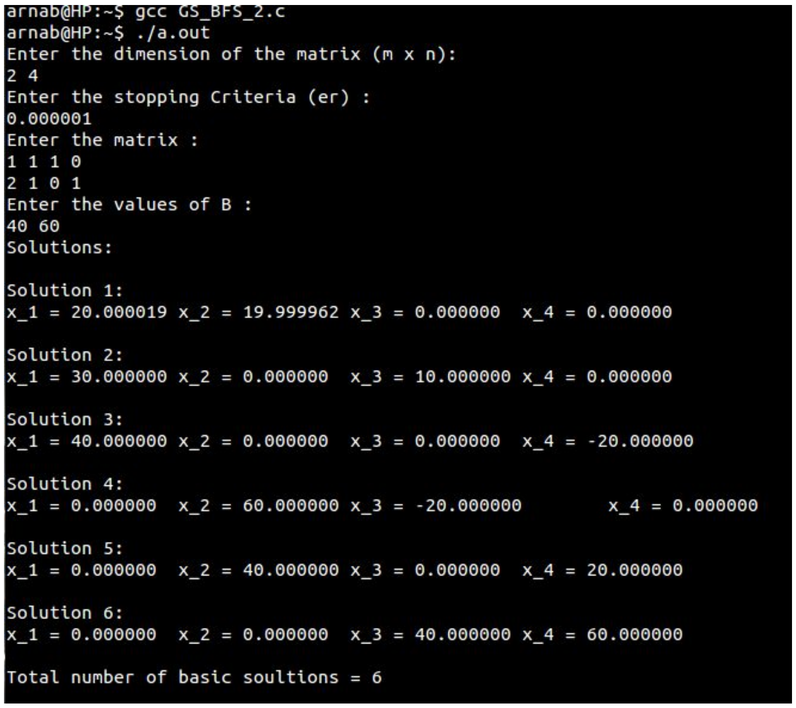
Code:

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

int combos[100][10];
int count;

long long int nCr(int n, int r){
      int i;
      long long int result = 1;
      if(r > n/2) r = n - r;
      for(i = 1; i <= r; ++i){
            result *= (n-i+1);
            result /= i;
      }
      return result;
}
```

```
int transformToDominant(int m, int r, double M[m][m], double N[m], int V[], int
R[])
    {
      int i, j;
        int n = m;
        if (r == m)
        {
            double T[n][n+1];
            double P[n];
            for (i = 0; i < n; i++)
            {
            P[i] = N[R[i]];
                for (j = 0; j < n; j++)
                    T[i][j] = M[R[i]][j];
            }

            for (i = 0; i < n; i++)
            {
            N[i] = P[i];
                for (j = 0; j < n; j++)
                    M[i][j] = T[i][j];
            }

            return 1;
        }

        for (i = 0; i < n; i++)
        {
            if (V[i]) continue;

            double sum = 0;

            for (j = 0; j < n; j++)
                sum += fabs(M[i][j]);

            if (2 * fabs(M[i][r]) >= sum)
            {
                V[i] = 1;
                R[r] = i;

                if (transformToDominant(m, r + 1, M, N, V, R))
                    return 1;

                V[i] = 0;
            }
        }

        return 0;
    }

int makeDominant(int m, double M[m][m], double N[m])
{
    int i;
    int visited[m];
    for(i = 0; i < m; ++i) visited[i] = 0;
    int rows[m];
    return transformToDominant(m, 0, M, N, visited, rows);
}

double * gaussSeidel( int m, int n, double a[m][n], double b[m], double er, int
itr){
      int key, i, j;
      double x0[n], sum;
      double * x = (double *) malloc(n * sizeof(double));
      double M[m][m];
      double N[m];
      for(i = 0; i < n; ++i){
                  x0[i] = 0;
                  x[i] = 0;
      }

      for(i = 0; i < m; ++i){
            N[i] = b[i];
            for(j = 0; j < m; ++j){
                  M[i][j] = a[i][combos[itr][j]];
            }
      }

    makeDominant(m, M, N);

      do{
            key = 0;

            for(i = 0; i < m; ++i){
                  sum = N[i];
                  for(j = 0; j < m; ++j)
                        if(j != i)
                              sum -= M[i][j] * x0[combos[itr][j]];

                  x[combos[itr][i]] = sum / M[i][i];
                  if(fabs((x[combos[itr][i]] - x0[combos[itr][i]]) /
x[combos[itr][i]]) > er){
                        key = 1;
                        x0[combos[itr][i]] = x[combos[itr][i]];
                  }
            }
      }while(key == 1);

      return x;
}

void concatCombination(int arr[], int data[], int start, int end,
                     int index, int r)
{
    int i, j;
    if (index == r){
      for(j = 0; j < r; ++j){
            combos[count][j] = data[j];
            }
      ++count;
      return;
    }

    for (i = start; i <= end && end-i+1 >= r-index; ++i)
    {
        data[index] = arr[i];
        concatCombination(arr, data, i+1, end, index+1, r);
    }
}

void getCombination(int arr[], int n, int r)
{
    int data[r];
    concatCombination(arr, data, 0, n-1, 0, r);
}

int main(){
      int i, j, n, m;
      double er;

      printf("Enter the dimension of the matrix (m x n):\n");
      scanf("%d%d", &m, &n);

      long long int solCount = nCr(n, m);

      printf("Enter the stopping Criteria (er) :\n");
      scanf("%lf", &er);

      printf("Enter the matrix : \n");
      double a[m][n];
      for(i = 0; i < m; ++i)
            for(j = 0; j < n; ++j)
                  scanf("%lf", &a[i][j]);
```

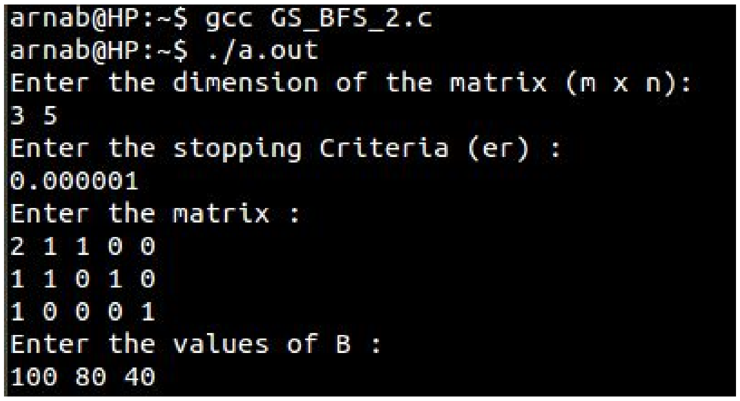
```
          printf("Enter the values of B : \n");
          double b[m];
        for(i = 0; i < m; ++i)
              scanf("%lf", &b[i]);

        count = 0;
        int indices[n];
        for(i = 0; i < n; ++i)  indices[i] = i;

        getCombination(indices, n, m);

        if(count != solCount) printf("Error : All combinations not found!\n");

        double * x[solCount];
        printf("Solutions:\n\n");

        for(j = 0; j < solCount; ++j){
              printf("Solution %d:\n", j+1);
              x[j] = gaussSeidel(m, n, a, b, er, j);
          for(i = 0; i < n; ++i)
              printf("x_%d = %lf\t", i+1, x[j][i]);
          printf("\n\n");
        }
        printf("Total number of basic soultions = %d\n\n", count);
        return 0;
}
```

Result:



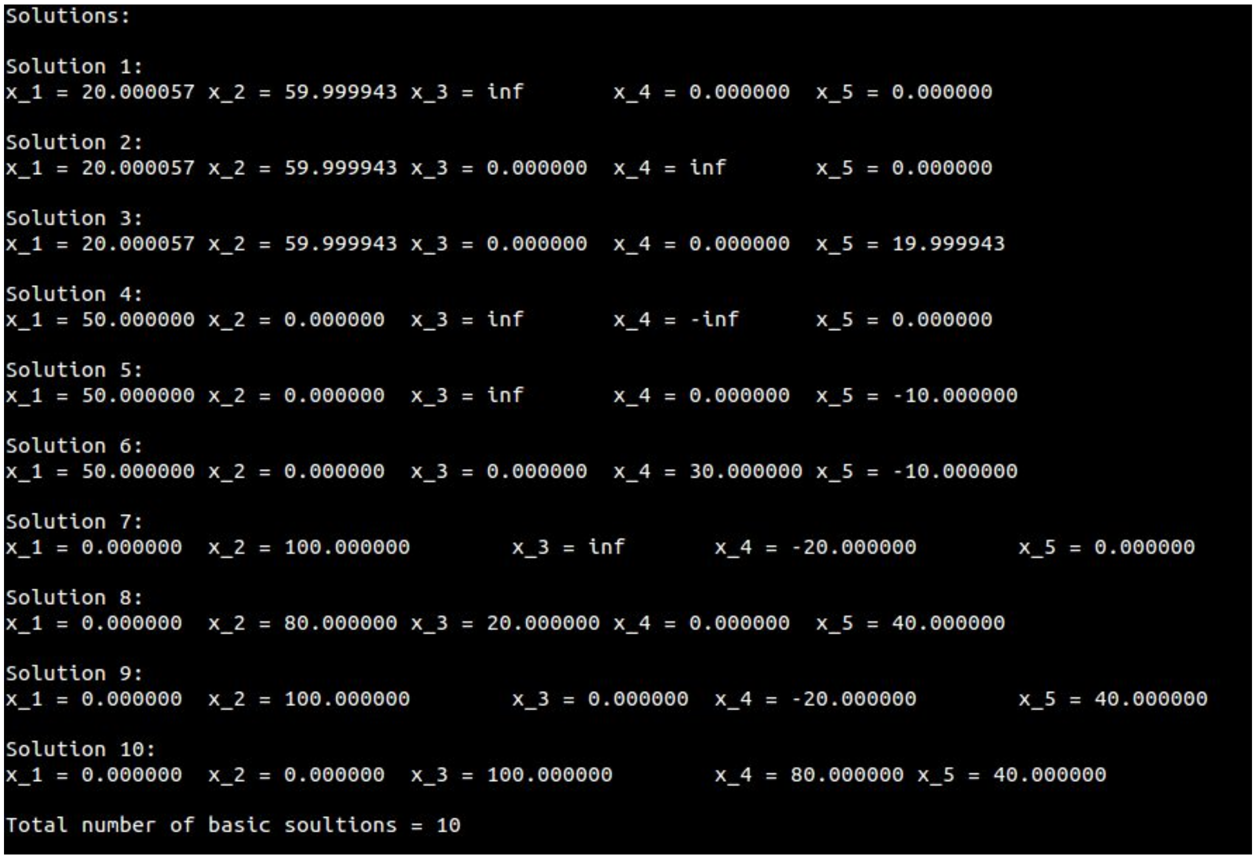
```
arnab@HP:~$ gcc GS_BFS_2.c
arnab@HP:~$ ./a.out
Enter the dimension of the matrix (m x n):
2 4
Enter the stopping Criteria (er) :
0.000001
Enter the matrix :
1 1 1 0
2 1 0 1
Enter the values of B :
40 60
Solutions:

Solution 1:
x_1 = 20.000019 x_2 = 19.999962 x_3 = 0.000000  x_4 = 0.000000

Solution 2:
x_1 = 30.000000 x_2 = 0.000000  x_3 = 10.000000 x_4 = 0.000000

Solution 3:
x_1 = 40.000000 x_2 = 0.000000  x_3 = 0.000000  x_4 = -20.000000

Solution 4:
x_1 = 0.000000  x_2 = 60.000000 x_3 = -20.000000        x_4 = 0.000000

Solution 5:
x_1 = 0.000000  x_2 = 40.000000 x_3 = 0.000000  x_4 = 20.000000

Solution 6:
x_1 = 0.000000  x_2 = 0.000000  x_3 = 40.000000 x_4 = 60.000000

Total number of basic soultions = 6
```

(a) In this case, the m x m coefficient matrix is always diagonally dominant. Hence we get convergent solution in every case.



```
arnab@HP:~$ gcc GS_BFS_2.c
arnab@HP:~$ ./a.out
Enter the dimension of the matrix (m x n):
3 5
Enter the stopping Criteria (er) :
0.000001
Enter the matrix :
2 1 1 0 0
1 1 0 1 0
1 0 0 0 1
Enter the values of B :
100 80 40
```

(b) In this case, the m x m coefficient matrix is not always diagonally dominant. Hence we do not get convergent solution in every case.



```
Solutions:

Solution 1:
x_1 = 20.000057 x_2 = 59.999943 x_3 = inf       x_4 = 0.000000  x_5 = 0.000000

Solution 2:
x_1 = 20.000057 x_2 = 59.999943 x_3 = 0.000000  x_4 = inf       x_5 = 0.000000

Solution 3:
x_1 = 20.000057 x_2 = 59.999943 x_3 = 0.000000  x_4 = 0.000000  x_5 = 19.999943

Solution 4:
x_1 = 50.000000 x_2 = 0.000000  x_3 = inf       x_4 = -inf      x_5 = 0.000000

Solution 5:
x_1 = 50.000000 x_2 = 0.000000  x_3 = inf       x_4 = 0.000000  x_5 = -10.000000

Solution 6:
x_1 = 50.000000 x_2 = 0.000000  x_3 = 0.000000  x_4 = 30.000000 x_5 = -10.000000

Solution 7:
x_1 = 0.000000  x_2 = 100.000000        x_3 = inf       x_4 = -20.000000        x_5 = 0.000000

Solution 8:
x_1 = 0.000000  x_2 = 80.000000 x_3 = 20.000000 x_4 = 0.000000  x_5 = 40.000000

Solution 9:
x_1 = 0.000000  x_2 = 100.000000        x_3 = 0.000000  x_4 = -20.000000        x_5 = 40.000000

Solution 10:
x_1 = 0.000000  x_2 = 0.000000  x_3 = 100.000000        x_4 = 80.000000 x_5 = 40.000000

Total number of basic soultions = 10
```