**SIMPLEX METHOD C CODE**

**CAT 1 PROJECT**

**TEAM MEMBERS:**

ABISHEK 1933002

RACHITHA.K 1933024

SUWETHA.G 1933032

**AIM:**

Implementation of simplex method in C programming language.

**ALGORITHM:**

**Simplex Algorithm**

1. Start with the initial basis associated with identity matrix.

2. Calculate the relative profits.

**For MAX problem-**

If all the relative profits are less than or equal to 0, then the current basis is the optimal one. STOP.

Else continue to 3.

**For MIN problem-**

If all the relative profits are greater than or equal to 0, then the current basis is the optimal one. STOP.

Else continue to 3.

3. Find the column corresponding to max relative profit. Say column k has the max

Rel. profit. So xk will enter the basis.

4. Perform a min ratio test to determine which variable will leave the basis.

Index of the min element i.e 'r' will determine the leaving variable.

The basic variable at index r, will leave the basis.

**NOTE: Min ratio test is always performed on positive elements.**

5. It's evident that the entered variable will not form an identity matrix, so

we will have to perform row operations to make it identity again.

Find the pivot element. The element at index (r, k) will be the pivot element and

row r will be the pivot row.

6. Divide the rth row by pivot to make it 1. And subtract c\*(rth row) from other

rows to make them 0, where c is the coefficient required to make that row 0.

**SOURCE CODE:**

#include <stdio.h>

#include <conio.h>

#define INFINITY 999

#define N 3

#define M 6

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\* Solves the LPP by "SIMPLEX" method i.e. by table \*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void minimum(float \*arr,int \*arrminpos,int n);

/\* Calculates the minimum valued position among the array arr having n elements. \*/

void display (float c[],float b[],float a[][M],int basic[]);

/\* Display the table \*/

void displayframe(float c[M]);

/\* Displays the frame of the table \*/

void calctemp(float \*,float [][M],float [],int []);

/\* Calculates Zj-Cj \*/

/--------------------------------------------------------------------------\

Cj 5 4 3 0 0 0 miniRatio

cB xB b a1 a2 a3 a4 a5 a6 bi/aij

0 x4 5 2 3 1 1 0 0 2.5

0 x5 11 4 1 2 0 1 0 2.75

0 x6 8 3 4 2 0 0 1 2.66

----------------------------------------------------------------------------

Zj-Cj -5 -4 -3 0 0 0

----------------------------------------------------------------------------

5 x1 2.5 1 1.5 0.5 0.5 0 0 5

0 x5 1 0 -5 0 -2 1 0 infinity

0 x6 105 0 -0.5 0.5 -1.5 0 1 1

----------------------------------------------------------------------------

Zj-Cj 0 3.5 -0.5 2.5 0 0

----------------------------------------------------------------------------

5 x1 2 1 2 0 2 0 -1

0 x5 1 0 -5 0 -2 1 0

3 x3 1 0 -1 1 -3 0 2

----------------------------------------------------------------------------

Zj-Cj 0 3 0 1 0 1

----------------------------------------------------------------------------

So the solution is :-

x1=2 x2=0 x3=1 x4=0 x5=1 x6=0

max(z) = 5\*2 + 4\*0 + 3\*1 = 13.

\--------------------------------------------------------------------------/

void main()

{

float c[M]={{5},{4},{3},{0},{0},{0}};

/\* Stores co-efficient of the objective function Max(z) \*/

float a[N][M]={

{2,3,1,1,0,0},

{4,1,2,0,1,0},

{3,4,2,0,0,1}

};

/\* Stores the co-efficent of the constraints \*/

float b[N]={{5},{11},{8}};

/\* Stores the values on RHS of constraints \*/

float temp[M]={{0},{0},{0},{0},{0},{0}};

/\* Stores the values of Zj-Cj\*/

int tempminpos; /\* Stores the minimum valued position

of {Zj-Cj} i.e. coming in variable \*/

float miniratio[N]; /\* Stores the value of the ratio b[i]/a[i][j] \*/

int miniratiominpos; /\* Stores the minimum valued position of

b[i]/a[i][j] i.e. going out variable \*/

float key; /\* Stores the key element \*/

int gooutcol; /\* Stores the column number which goes out \*/

float z; /\* Stores the value of the objective function \*/

float x[M]; /\* Stores the value of the variables \*/

int i,j; /\* Loop variables \*/

int basic[N]; /\* Stores the basic variable \*/

int nonbasic[N]; /\* Stores the non-basic variable \*/

int flag=0; /\* Terminating variable \*/

//clrscr();

/\* Initializing basic variables to 3,4,5 i.e. x4,x5,x6 \*/

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

{

basic[i]=(i+N);

nonbasic[i]=i;

}

printf("\nMax z = c1x1 + c2x2 + c3x3\n");

printf("\na11x1 + a12x2 + a13x3 <= b1\n");

printf("\na21x1 + a22x2 + a23x3 <= b2\n");

printf("\na31x1 + a31x2 + a32x3 <= b3\n");

printf("\nEnter values of ci's\n");

/\* Inputing requisite amount of data \*/

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

{

printf("\nEnter c[%d]\t",i+1);

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

}

printf("\nEnter values of ai's\n");

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

{

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

{

printf("\nEnter a[%d][%d]\t",i+1,j+1);

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

}

}

printf("\nEnter values of bi's\n");

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

{

printf("\nEnter b[%d]\t",i+1);

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

}

/\* Calculation for actual table \*/

while(flag==0)

{

z=0;

calctemp(temp,a,c,basic);

printf("\n");

/\* Determining the incoming column \*/

minimum(temp,&tempminpos,M);

display(c,b,a,basic);

printf("\nZj-Cj\t\t\t");

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

printf("%.4g\t",temp[i]);

printf("\n\n");

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

{

x[basic[i]]=b[i];

x[nonbasic[i]]=0;

printf("x[%d]=%g\n",basic[i]+1,b[i]);

}

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

z=z+c[i]\*x[i];

printf("Max(z) = %g",z);

/\* Determining the outgoing column \*/

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

{

if(a[i][tempminpos]==0)

{

miniratio[i]=INFINITY;

continue;

}

if(a[i][tempminpos]<0)

{

miniratio[i]=INFINITY;

continue;

}

miniratio[i]=b[i]/a[i][tempminpos];

}

minimum(miniratio,&miniratiominpos,N);

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

if(miniratiominpos==i)

gooutcol=basic[i];

printf("\nComing in variable = X%d\t",tempminpos+1);

printf("Going out variable = X%d\n",gooutcol+1);

/\* Changing the basic and non-basic variable \*/

basic[miniratiominpos]=tempminpos;

nonbasic[tempminpos]=gooutcol;

/\* Performing the operations to bring similar expressions in

in-coming variable as out-going variable by row operations \*/

key=a[miniratiominpos][tempminpos];

b[miniratiominpos]=b[miniratiominpos]/key;

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

a[miniratiominpos][i]=a[miniratiominpos][i]/key;

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

{

if(miniratiominpos==i)

continue;

key=a[i][tempminpos];

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

{

a[i][j]=a[i][j]-a[miniratiominpos][j]\*key;

}

b[i]=b[i]-b[miniratiominpos]\*key;

}

getch();

/\* Terminating condition \*/

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

{

flag=1;

if(temp[i]<0)

{

flag=0;

break;

}

}

}

printf("\nPress any key to exit...\n");

getch();

}

void calctemp(float \*temp,float a[N][M],float c[M],int basic[N])

{

int i,j;

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

{

temp[i]=0;

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

temp[i]=temp[i]+c[basic[j]]\*a[j][i];

temp[i]=temp[i]-c[i];

}

}

void minimum(float \*arr,int \*arrminpos, int n)

{

int i;

float arrmin;

arrmin=arr[0];

\*arrminpos=0;

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

if(arr[i]<arrmin)

{

arrmin=arr[i];

\*arrminpos=i;

}

printf("\n%d\n",\*arrminpos);

}

void display (float c[N],float b[N],float a[N][M],int basic[N])

{

int i,j;

displayframe(c);

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

{

printf("\n%.4g\tX%d\t%.4g\t",c[basic[i]],basic[i]+1,b[i]);

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

printf("%.4g\t",a[i][j]);

printf("\n");

}

}

void displayframe(float c[M])

{

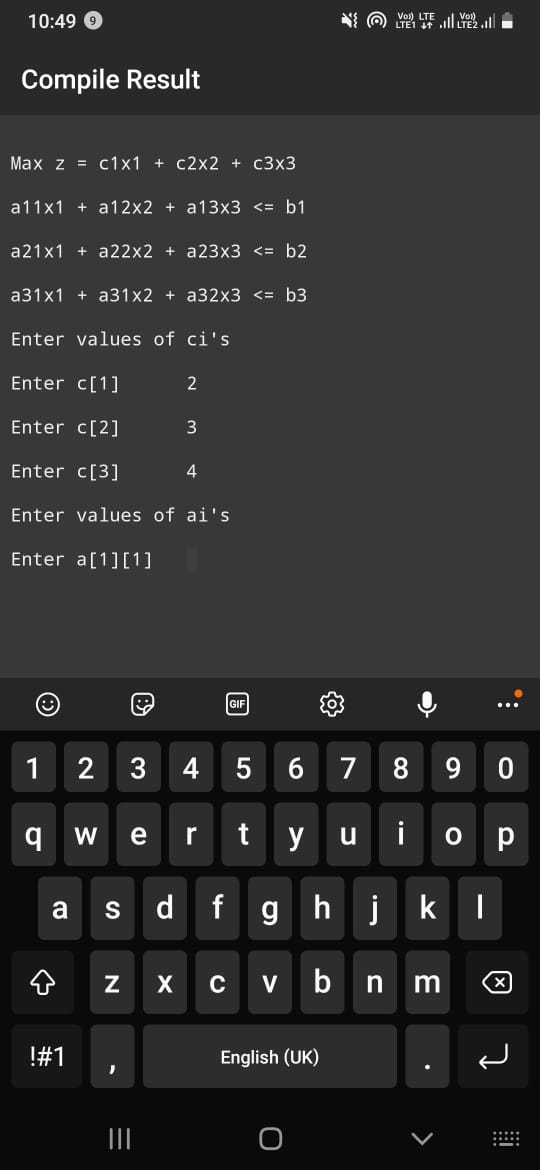
printf("\t\tc[j]\t");

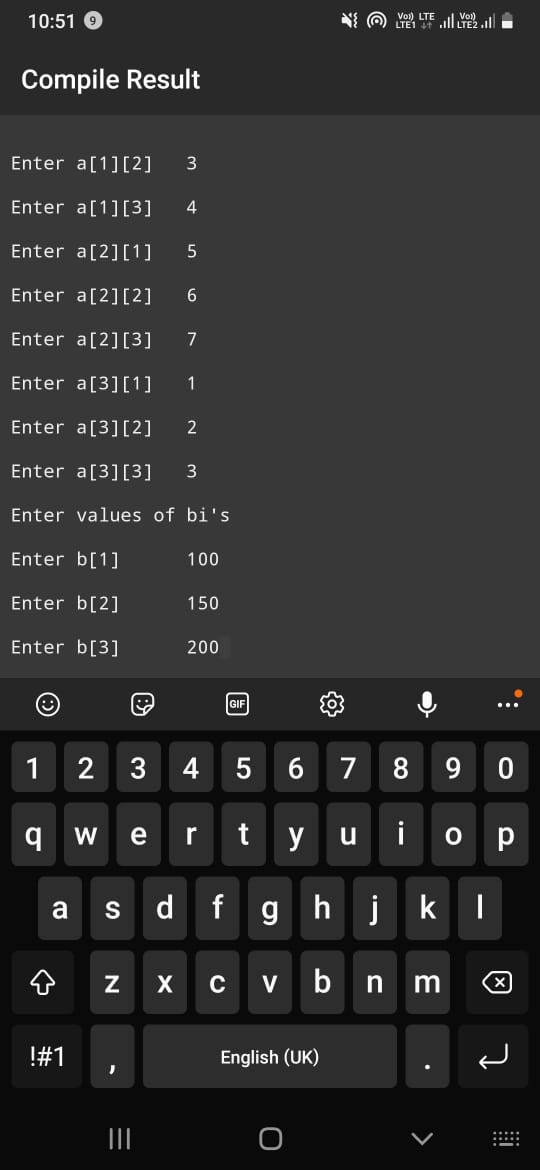
printf("%g\t%g\t%g\t%g\t%g\t%g\n",c[0],c[1],c[2],c[3],c[4],c[5]);

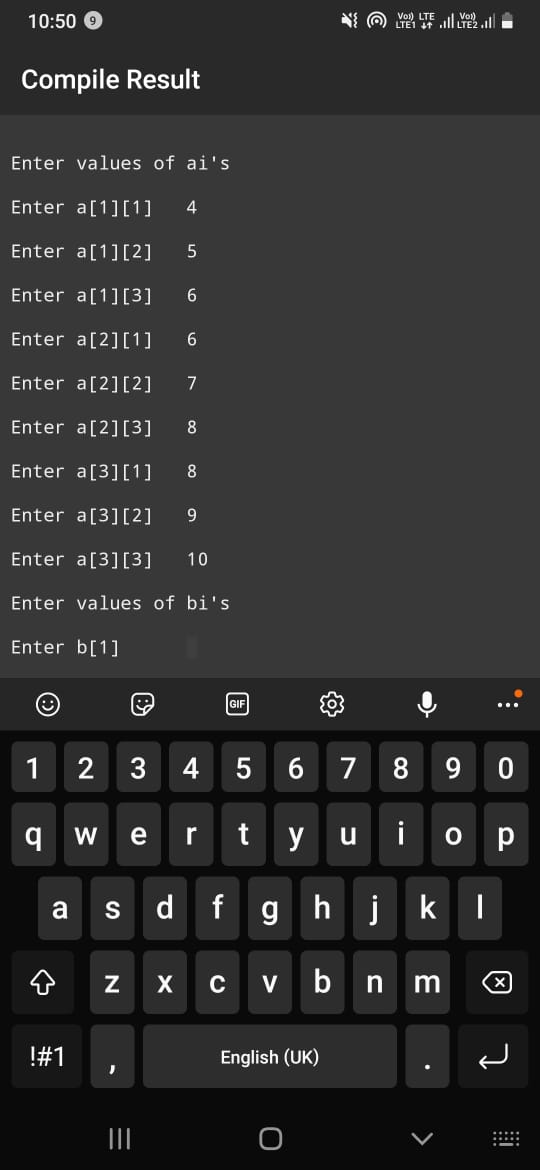
printf("\nc[B]\tB\tb\ta1\ta2\ta3\ta4\ta5\ta6\n");

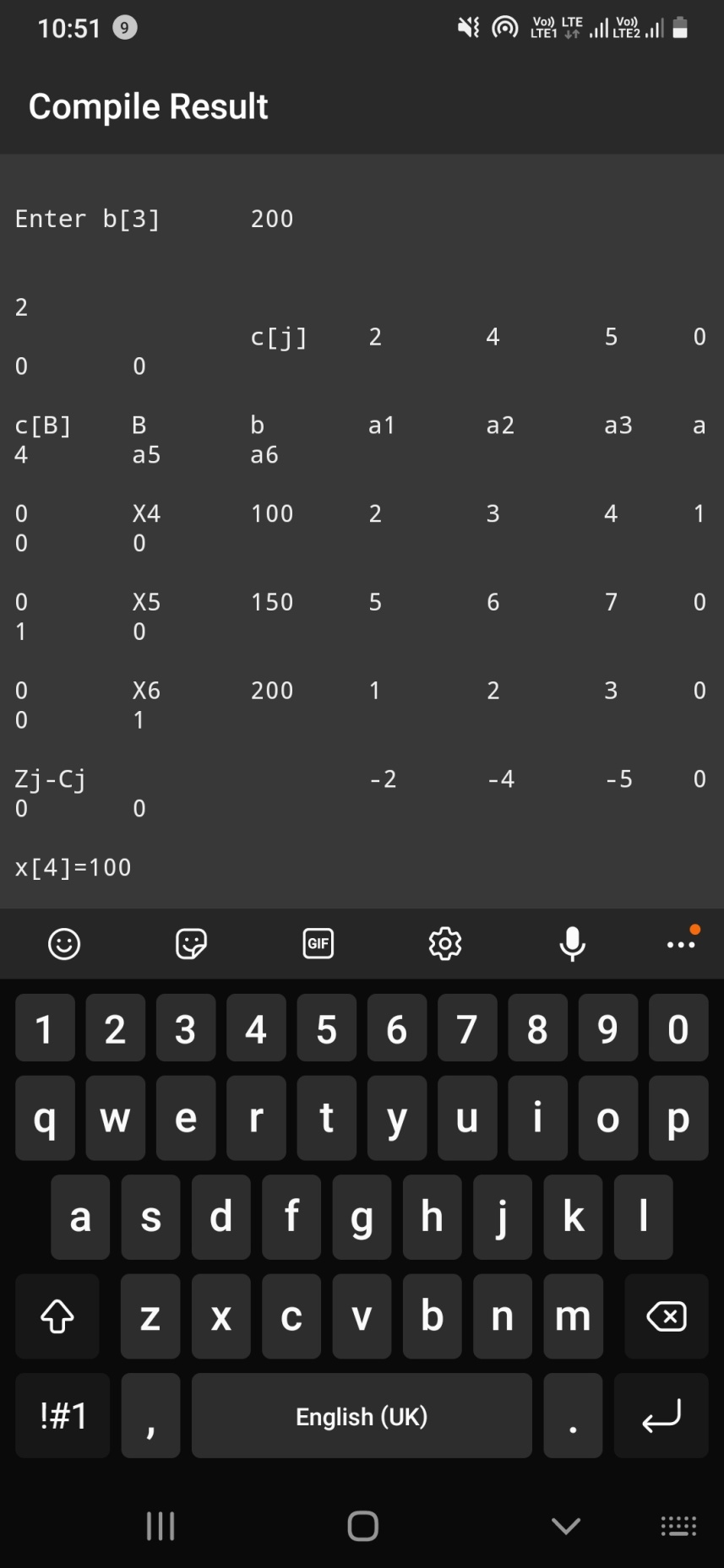
}

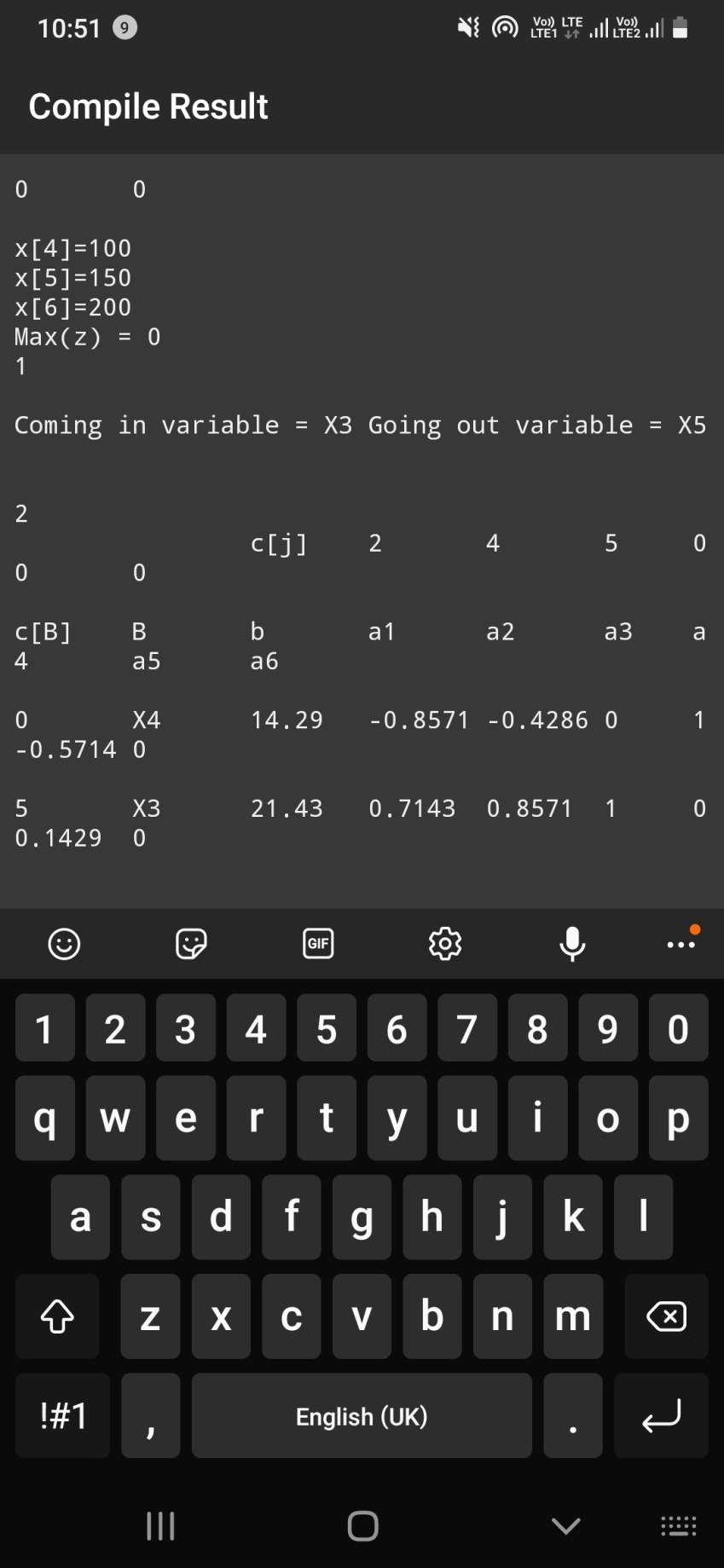
OUTPUT:

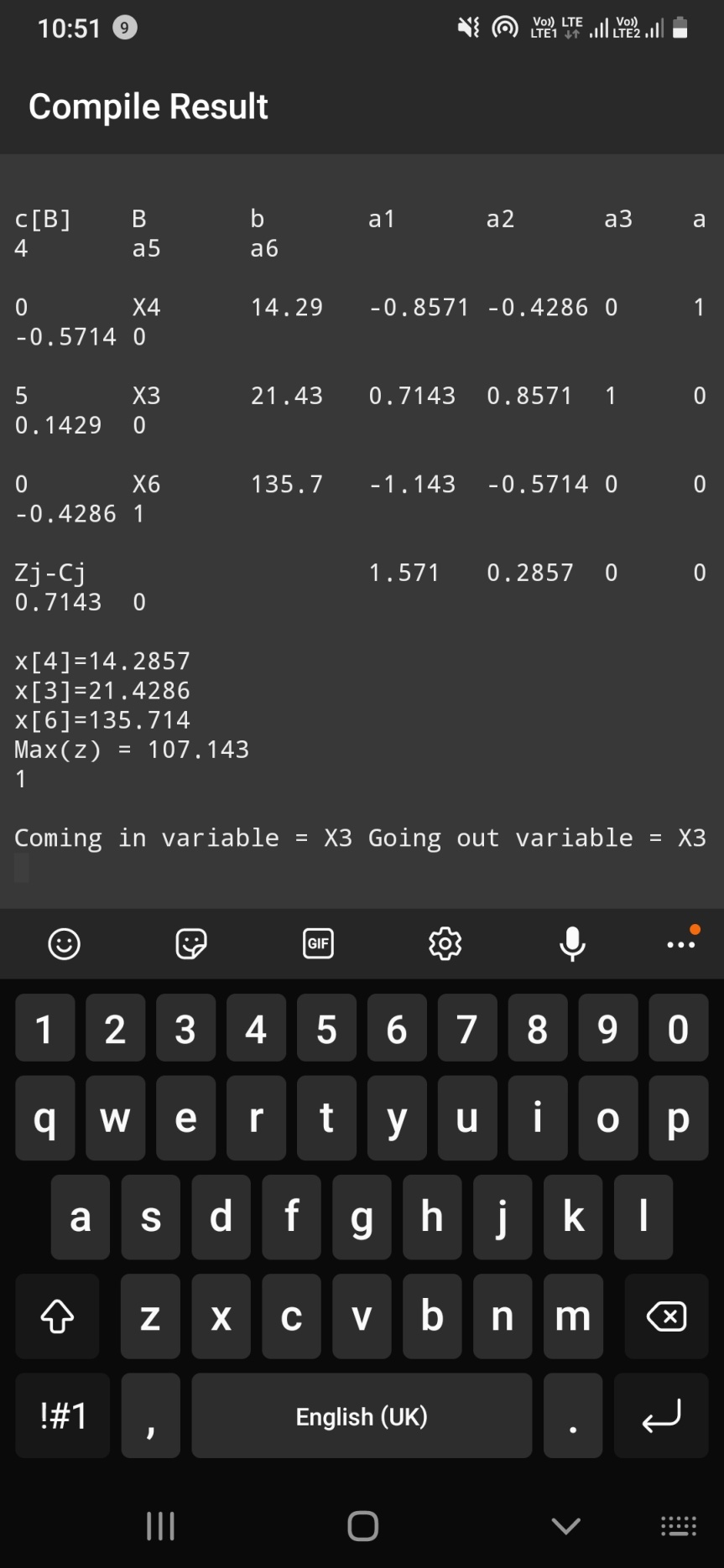
****

****

****

****

****

****