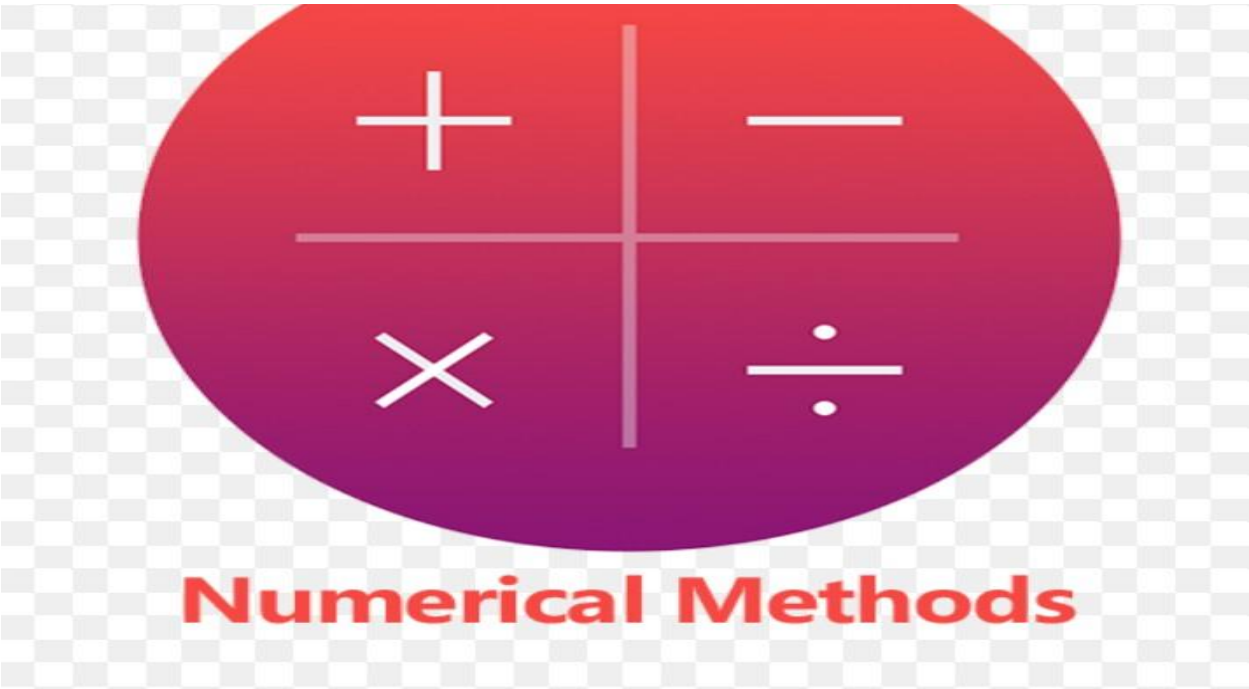



- ***SUBJECT:-*** **NUMERICAL METHOD PRACTICAL WORKBOOK**
- ***COLLEGE NAME:-*** **ASUTOSH COLLEGE**
- ***COLLLEGE ROLL:-*****1050(SEC-A)**
- ***CU REG. NO. :-*** **012-1111-1251-18**
- ***CU ROLL NO. :-*****183012-21-0326**
- ***SEMESTER:-*****VI**
- ***PAPER:-*****CC-14(NUMERICAL. METHODS-PRACTICAL).**

## CONTENT


# ACKNOWLEDGEMENT

**I would like to express my special thanks and gratitude to my Numerical Method (LAB) teacher Miss. Sukanya Banerjee for her able guidance and support in completing my project.**

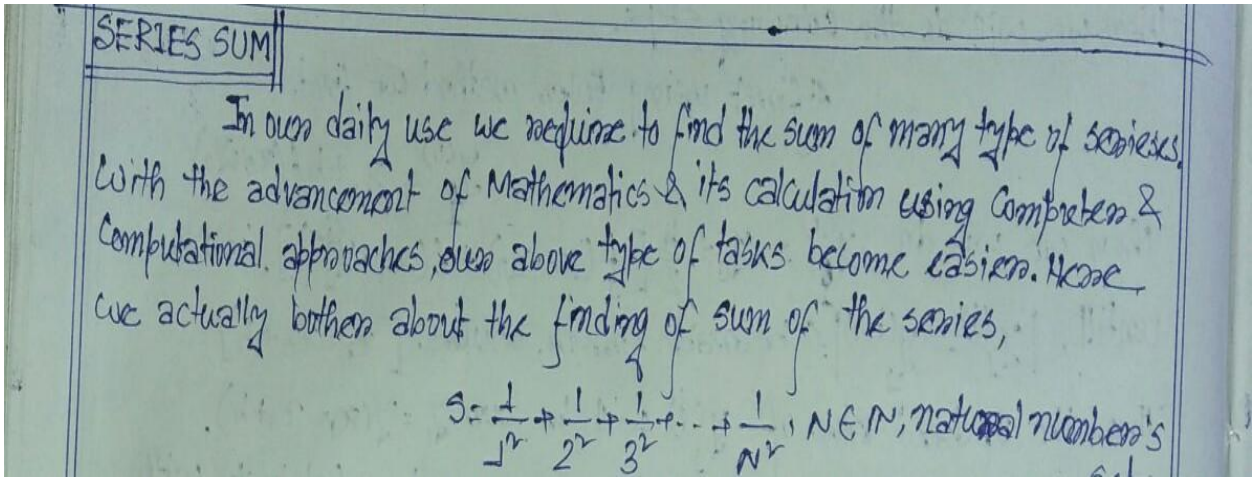
**I would extend my thanks to my classmates for helping me to complete this project work.**

**I am also thankful to all other corresponding faculties for giving me such a beautiful opportunity of doing Numerical Lab-works.**

**Suvendu Kar**
**Student of Mathematics Department(6th sem.)**
**Asutosh College(Kolkata)**
**Date:25/07/2021**

## (01) SUM OF SERIES

- ***Introduction:-***

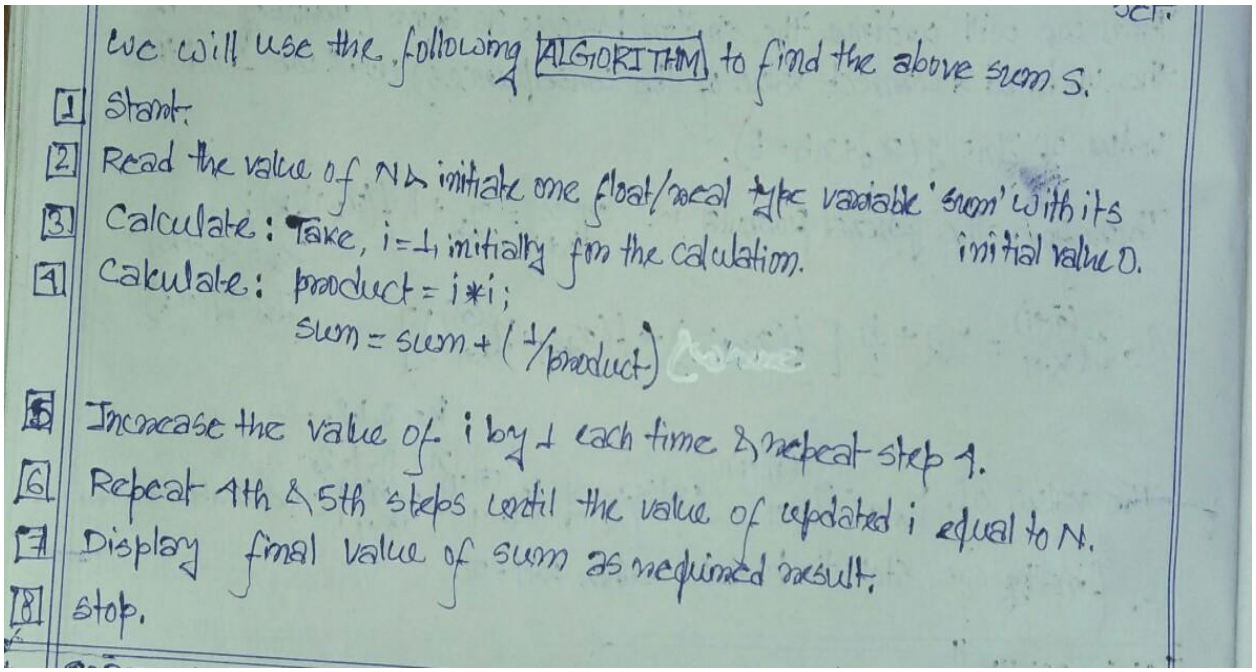


SERIES SUM

In our daily use we require to find the sum of many type of seriesses. With the advancement of Mathematics & its calculation using Computers & Computational approaches, over above type of tasks become easier. Hence we actually bother about the finding of sum of the series,

$$S = \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \dots + \frac{1}{N^2}, \; N \in \mathbb{N}, \text{ natural number's}$$

- ***Algorithm:-***



We will use the following ALGORITHM to find the above sum S.

[1] Start.
[2] Read the value of N & initiate one float/real type variable 'sum' with its initial value 0.
[3] Calculate: Take, i=1, initially for the calculation.
[4] Calculate: product = i*i;
sum = sum + (1/product)
[5] Increase the value of i by 1 each time & repeat step 4.
[6] Repeat 4th & 5th steps until the value of updated i equal to N.
[7] Display final value of sum as required result.
[8] stop.

- ***Question:-***

**Write a C program to evaluate the following Series :**

**S=1/(1*1)+1/(2*2)+.....+1/(N*N);**

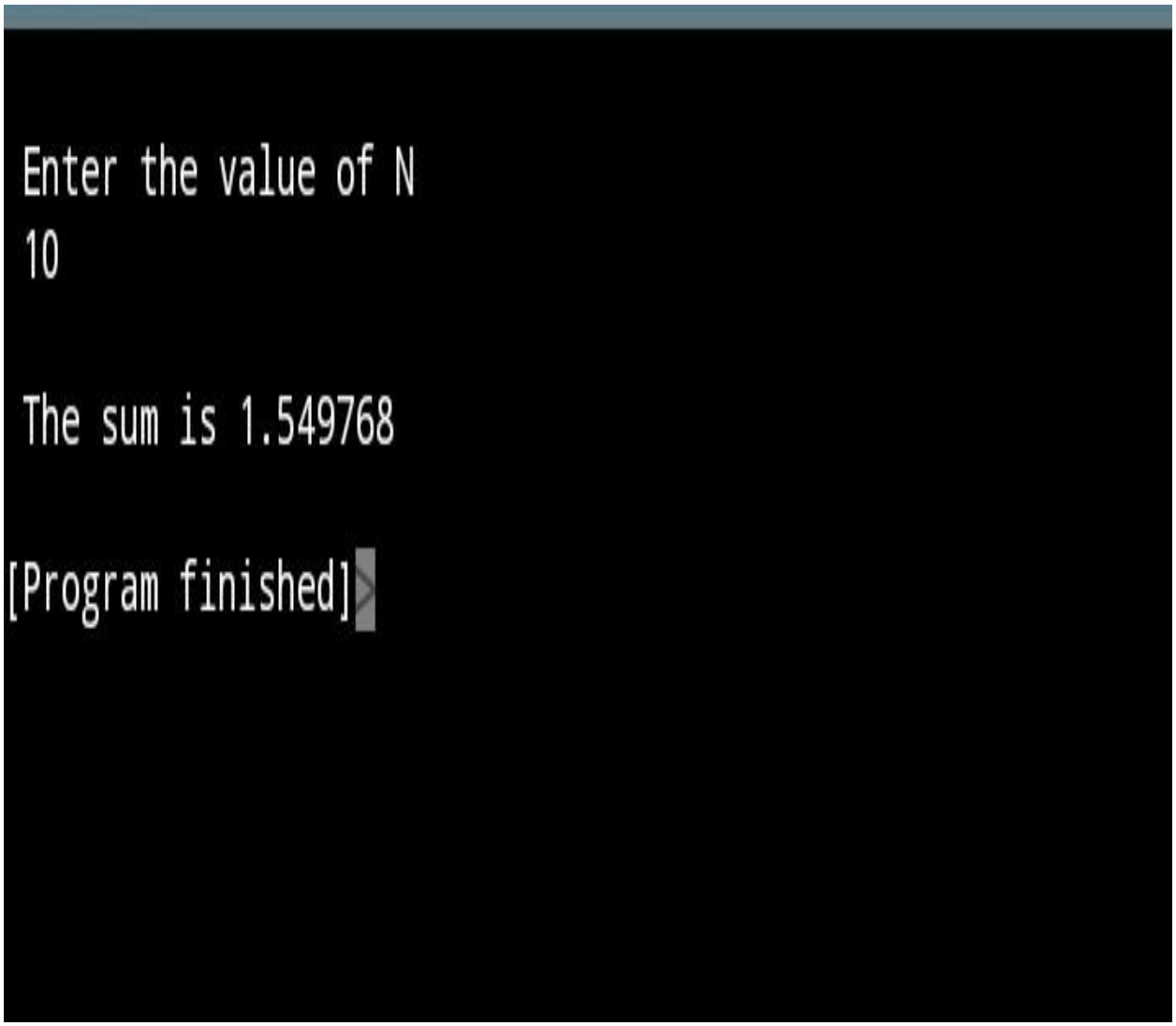
**N being a natural number.**

Sol:

**Input:**

```
//sum of series S=1/(1*1)+1/(2*2)+1/(3*3)+.....+1/(N*N)//
#include<stdio.h>
#include<math.h>
int main()
{
        int N;
        float i,sum;
        printf("\n Enter the value of N\n");
        scanf("%d",&N);
        sum=0.0;
        for(i=1;i<N+1;i++)
        sum=sum+1.0/pow(i,2.0);
        printf("\n The sum is %f\n",sum);
return 0;
}
```

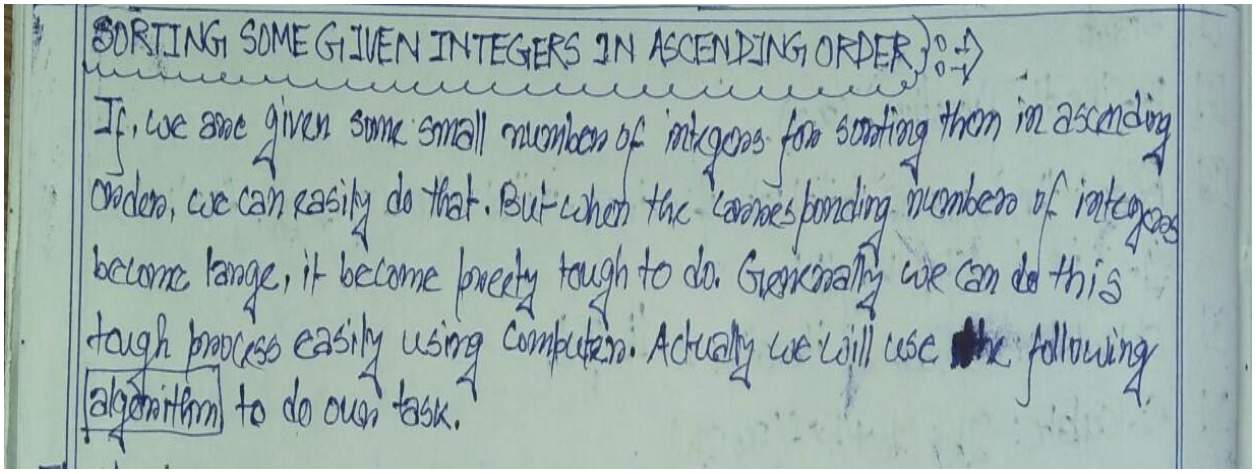
## ***USER INTERFACE:-***



***(2)***

## ***SORTING INTEGERS IN ASCENDING ORDER***

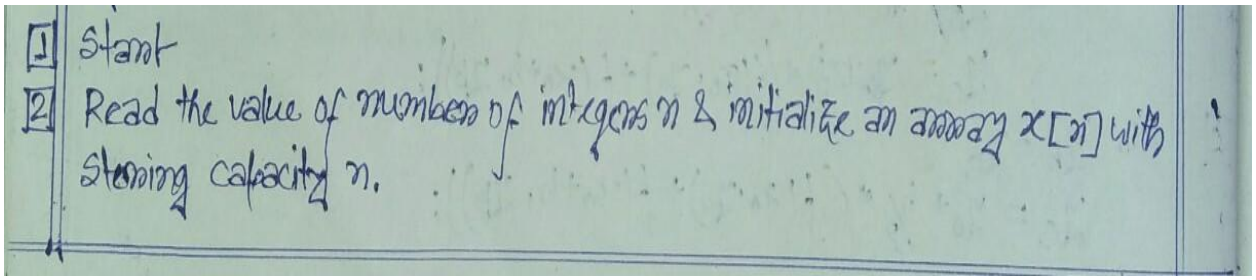
- ***INTRODUCTION:-***

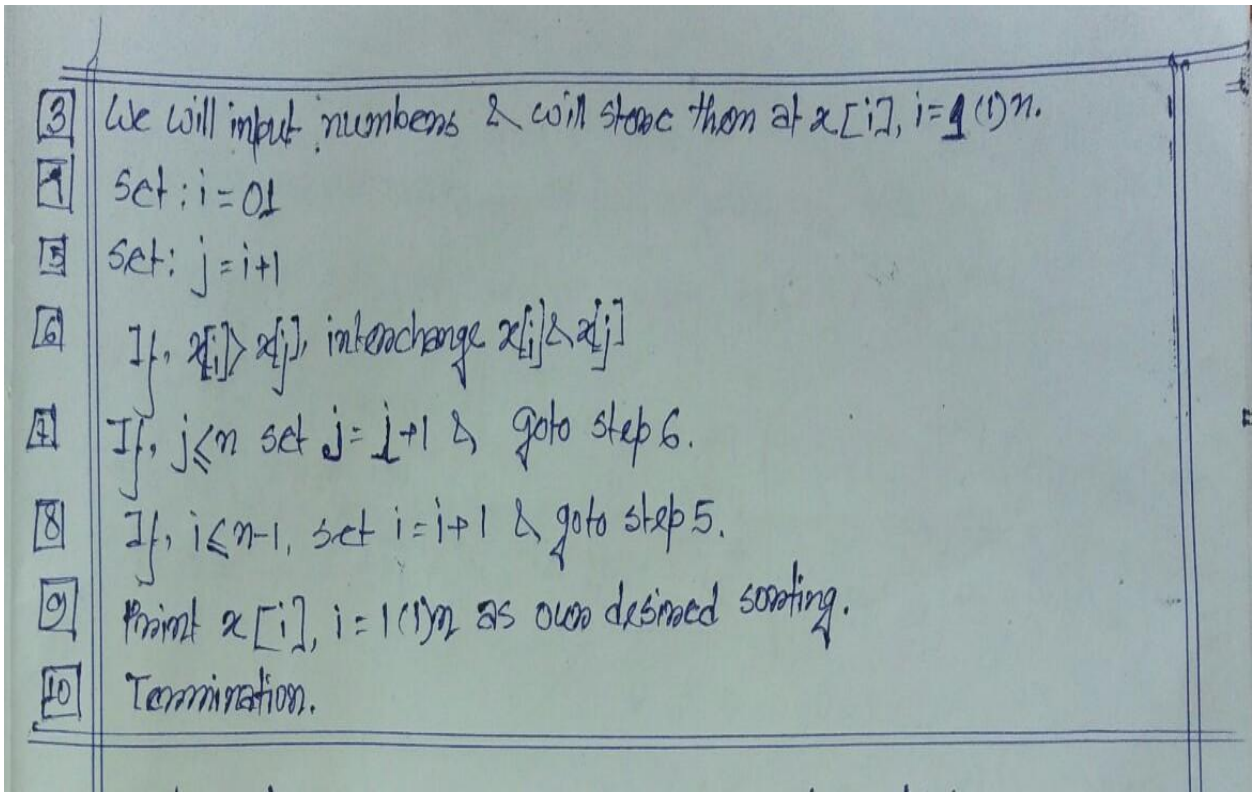


SORTING SOME GIVEN INTEGERS IN ASCENDING ORDER :⇒

If, we are given some small numbers of integers for sorting them in ascending order, we can easily do that. But when the corresponding numbers of integers become large, it become pretty tough to do. Generally we can do this tough process easily using computers. Actually we will use the following algorithm to do our task.

- ***ALGORITHM:-***


1 Start
2 Read the value of numbers of integers n & initialize an array x[n] with storing capacity n.


3 We will input numbers & will store them at x[i], i=1(1)n.
4 Set: i=0
5 Set: j=i+1
6 If, x[i]>x[j], interchange x[i]&x[j]
7 If, j≤n set j=j+1 & goto step 6.
8 If, i≤n-1, set i=i+1 & goto step 5.
9 Print x[i], i=1(1)n, as our desired sorting.
10 Termination.

***Question:-***

**Write a C program to arrange 25 given numbers in ascending order:**

***Answer:-***

```
//Sorting Integers in Ascending Order//
#include <stdio.h>
int main()
{
int i, j, a, n, x[30];
printf("\n Enter the number of entries(<30):
");
scanf("%d",&n);
printf("Enter the numbers \n");
for (i = 1; i < n+1; i++)
scanf("%d", &x[i]);
for (i = 1; i < n; i++)
{
for (j = i + 1; j < n+1; j++)
{
if (x[i] > x[j])
{
a =  x[i];
x[i] = x[j];
x[j] = a;
```

```
}
}
}
printf("The numbers arranged in ascending
order are given below \n");
for (i = 1; i < n+1; i++)
printf("%d\n", x[i]);
}
```

## User Interface:-

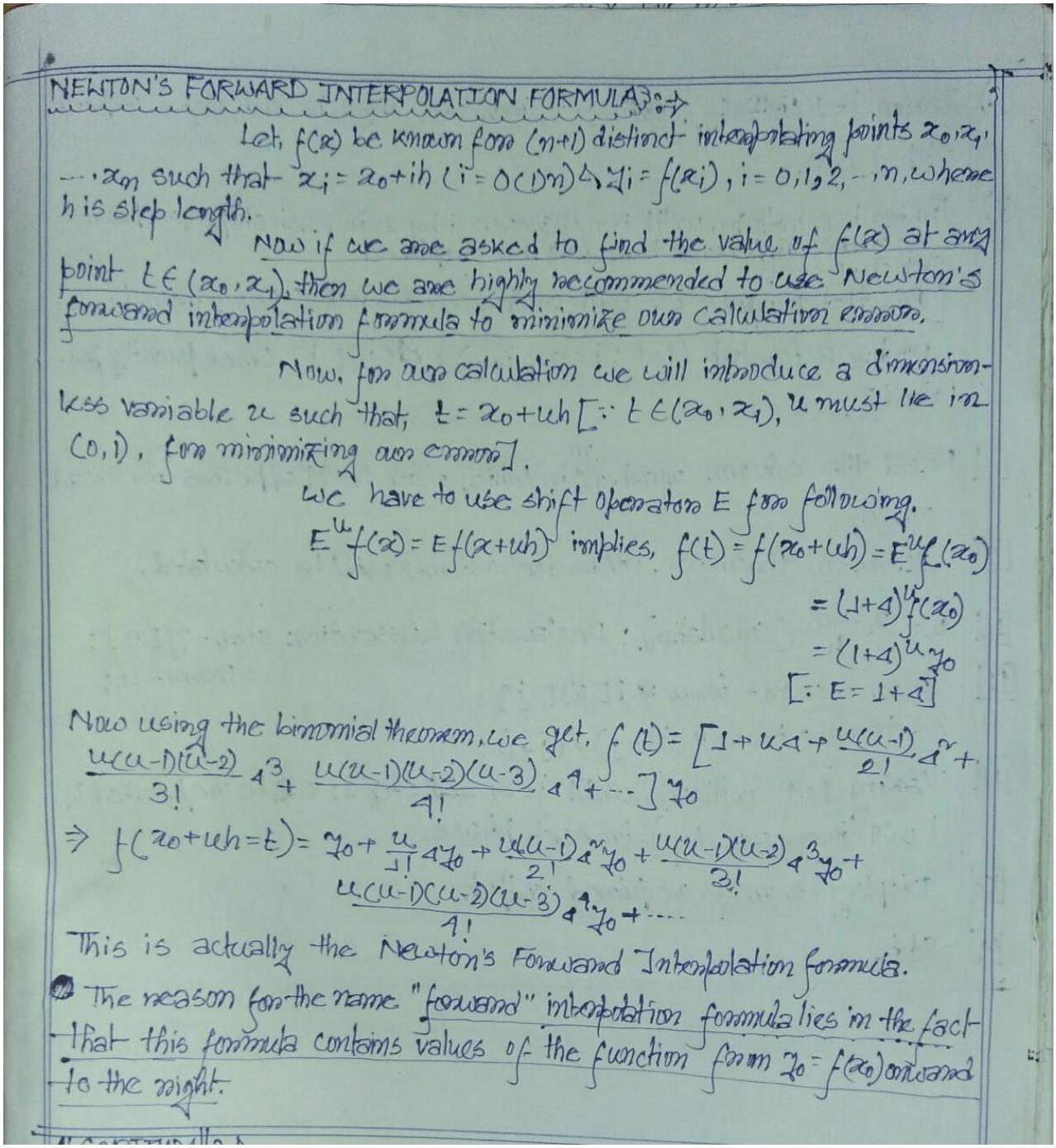
```
 Enter the number of entries(<30): 25
Enter the numbers
 25
 25
 563
 1
 5
 5
 8965
 1000
 53
 15
 56
 +56
 -1
 -56325
 -56
 -8
 8
 0
 1
 2
 5
 5
 412
 500
 720
The numbers arranged in ascending order are given below
-56325
-56
-8
-1
0
1
1
2
5
5
5
5
8
15
25
25
53
```

```
56
56
412
500
563
720
1000
8965

[Program finished]
```

## (3)

# NEWTON FORWARD INTERPOLATION METHOD

- ## INTRODUCTION:-



NEWTON'S FORWARD INTERPOLATION FORMULA:⇒

Let, $f(x)$ be known for $(n+1)$ distinct interpolating points $x_0, x_1, \dots, x_n$ such that $x_i = x_0 + ih$ $(i = 0(1)n) \Rightarrow y_i = f(x_i)$, $i = 0, 1, 2, \dots, n$, where $h$ is step length.

Now if we are asked to find the value of $f(x)$ at any point $t \in (x_0, x_1)$, then we are highly recommended to use Newton's forward interpolation formula to minimize our calculation errors.

Now, for our calculation we will introduce a dimensionless variable $u$ such that, $t = x_0 + uh$ [$\because t \in (x_0, x_1)$, $u$ must lie in $(0,1)$, for minimizing our errors].

We have to use shift operator $E$ for following.

$E^u f(x) = Ef(x+uh)$ implies, $f(t) = f(x_0+uh) = E^u f(x_0)$

$= (1+\Delta)^u f(x_0)$

$= (1+\Delta)^u y_0$

[$\because E = 1+\Delta$]

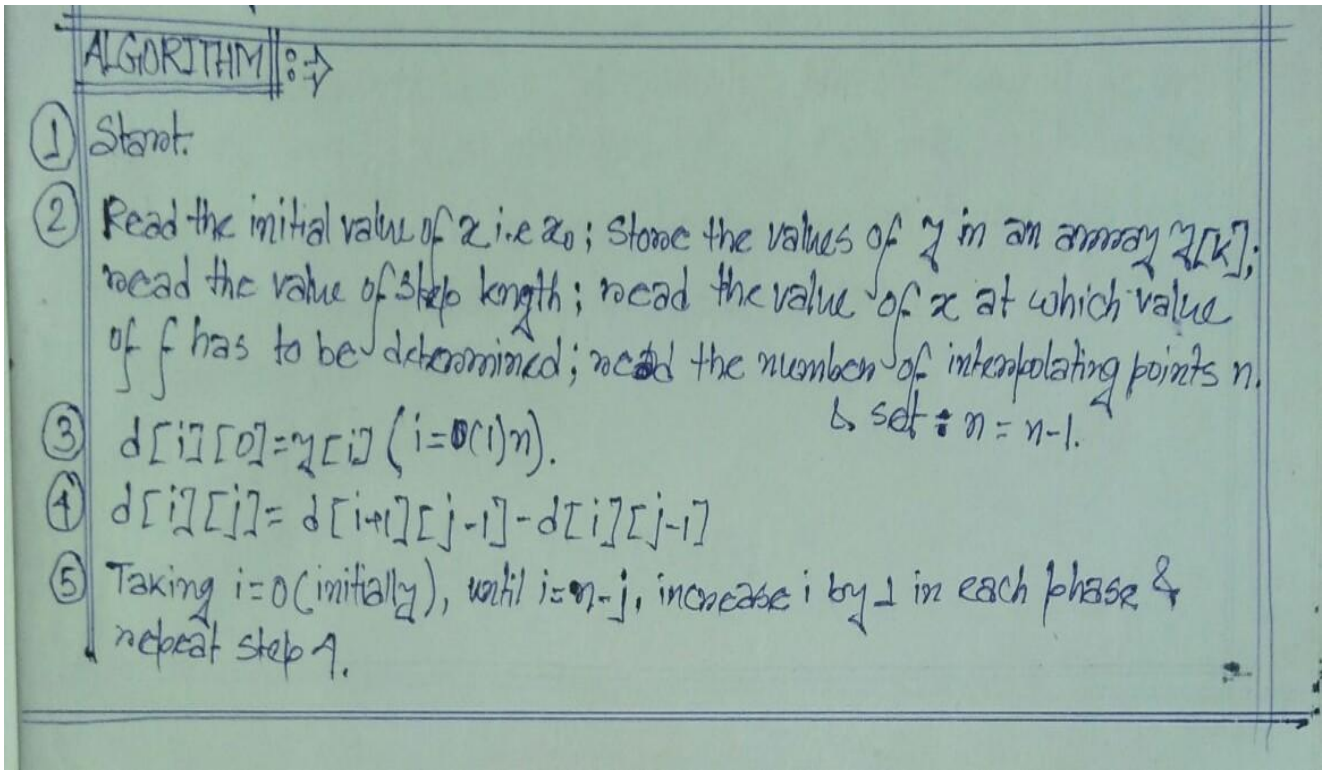
Now using the binomial theorem, we get, $f(t) = \left[1 + u\Delta + \frac{u(u-1)}{2!}\Delta^2 + \frac{u(u-1)(u-2)}{3!}\Delta^3 + \frac{u(u-1)(u-2)(u-3)}{4!}\Delta^4 + \dots\right] y_0$

$\Rightarrow f(x_0+uh=t) = y_0 + \frac{u}{1!}\Delta y_0 + \frac{u(u-1)}{2!}\Delta^2 y_0 + \frac{u(u-1)(u-2)}{3!}\Delta^3 y_0 + \frac{u(u-1)(u-2)(u-3)}{4!}\Delta^4 y_0 + \dots$

This is actually the Newton's Forward Interpolation formula.

The reason for the name "forward" interpolation formula lies in the fact that this formula contains values of the function from $y_0 = f(x_0)$ onward to the right.

- ***ALGORITHM***



ALGORITHM :→

1. Start.
2. Read the initial value of x i.e $x_0$; Store the values of y in an array y[k]; read the value of step length; read the value of x at which value of f has to be determined; read the number of interpolating points n.
   & set : n = n-1.
3. d[i][0] = y[i] (i = 0(1)n).
4. d[i][j] = d[i+1][j-1] - d[i][j-1]
5. Taking i = 0 (initially), until i = n-j, increase i by 1 in each phase & repeat step 4.

⑥ Taking j = 1 (initially), until j = n, increase j by 1 in each step & repeat step 4 & 5.

⑦ Now, the difference table will be displayed
[For that purpose we will continue a for loop (j = 0; j <= n-i; j++) under a for loop (i = 0; i <= n; i++) & d[i][j]. Consequently printed].

8 Read the column number (m) in which noise level appears or read m = 0.

9 If, m = 0, then m = n, otherwise m = m-2 will be calculated.

10 u = (x - x0)/h (Calculation).; Declaration & assertion: sum = y[0]; term = u;

11 sum = sum + term * d[0][j]
& term = term * (u-j)/(j+1);

12 Taking j = 1 initially, until j = m step 11 will be repeated & j will increase by 1 in each phase.

13 Display sum as required output.

14 Stop.

- ***QUESTION***

**Compute the values of $f(x)$ at $x = 0.20 + \frac{R+1}{100}$ from the following table using suitable interpolation formula:**

| $x$ | $f(x)$ |
|---|---|
| 0.20 | 1.2922071606 |
| 0.35 | 1.3397750591 |
| 0.50 | 1.3890939964 |
| 0.65 | 1.4402284308 |
| 0.80 | 1.4932451930 |
| 0.95 | 1.5482135742 |
| 1.10 | 1.6052054161 |
| 1.25 | 1.6642952050 |

**[R being last digit of university roll number]**

**Answer:**

**The last digit of my university roll number being R=6,according to the definition of the interpolating point x at which the value of f(x)has to be determined,its value is x=0.20+(1+R)/100=0.27.**

**//NEWTON FORWARD INTERPOLATION METHOD//**

```
#include<stdio.h>

int main()

{

float y[10],d[10][10],sum, x0,x,h,u,term;

int n, m, i, j, t;
```

```
printf("\n Enter the starting value x0,step size & the number of interpolating
points\n ");

scanf("%f%f%d",&x0,&h,&t);

printf("\n Input the values of y\n");

n=t-1;

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

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

printf("\n Input x for which interpolation is required\n");

scanf("%f",&x);

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

d[i][0]=y[i];

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

{

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

d[i][j]=d[i+1][j-1]-d[i][j-1];

}

printf("\nDifference Table is \n");

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

{
```

```
for(j=0;j<n-i+1;j++)

printf("%0.6f",d[i][j]);

printf("\n");

}

printf("\n Enter the column number in which noise level appears otherwise press
0\n");

scanf("%d",&m);

if(m==0)

m=n;

else

m=m-2;

u=(x-x0)/h;

sum=y[0];

term=u;

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

{

sum=sum+term*d[0][j];

term=term*(u-j) /(j+1);

}
```

```
printf("y(%0.5f)=%0.10f",x, sum);

return 0;

}
```

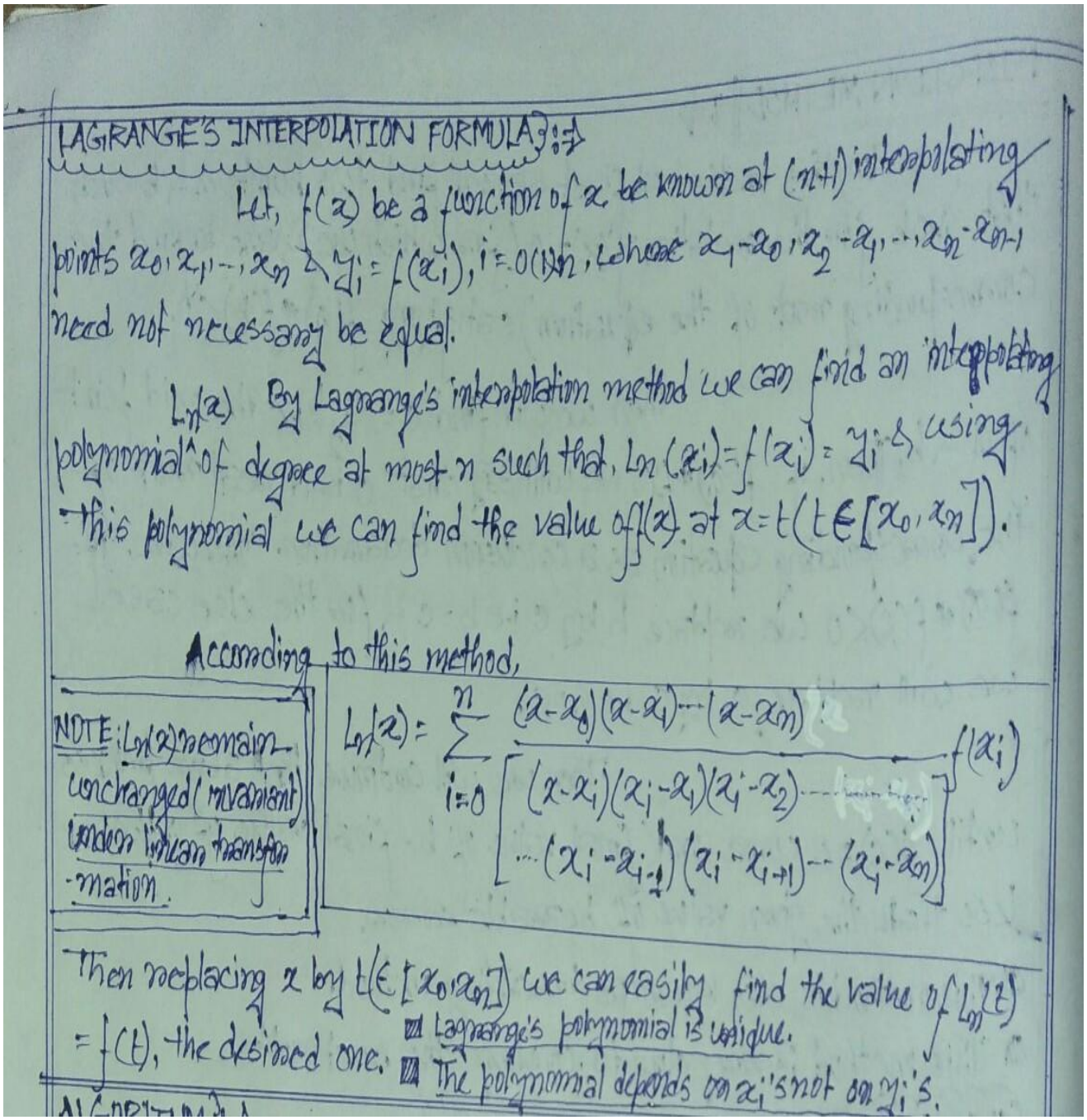
- ***USER INTERFACE***

```
 Enter the starting value x0,step size & the number of inter
polating points
 0.20
 0.15
 8

 Input the values of y
 1.2922071606
 1.3397750591
 1.3890939964
 1.4402284308
 1.4932451930
 1.5482135742
 1.6052054161
 1.6642952050

 Input x for which interpolation is required
 0.27

Difference Table is
1.2922070.0475680.0017510.0000650.0000020.000000-0.0000000.0
00000
1.3397750.0493190.0018160.0000670.0000030.000000-0.000000
1.3890940.0511340.0018820.0000690.0000030.000000
1.4402280.0530170.0019520.0000720.000003
1.4932450.0549680.0020230.000075
1.5482140.0569920.002098
1.6052050.059090
1.664295

 Enter the column number in which noise level appears otherw
ise press 0
 0
y(0.27000)=1.3141915798
[Program finished]
```

## (04)

# LAGRANGE INTERPOLATION METHOD

- **INTRODUCTION:-**



LAGRANGE'S INTERPOLATION FORMULA:-

Let, $f(x)$ be a function of $x$ be known at $(n+1)$ interpolating points $x_0, x_1, \dots, x_n$ & $y_i = f(x_i)$, $i = 0(1)n$, where $x_1 - x_0, x_2 - x_1, \dots, x_n - x_{n-1}$ need not necessary be equal.

$L_n(x)$ By Lagrange's interpolation method we can find an interpolating polynomial of degree at most $n$ such that, $L_n(x_i) = f(x_i) = y_i$ & using this polynomial we can find the value of $f(x)$ at $x = t (t \in [x_0, x_n])$.

According to this method,

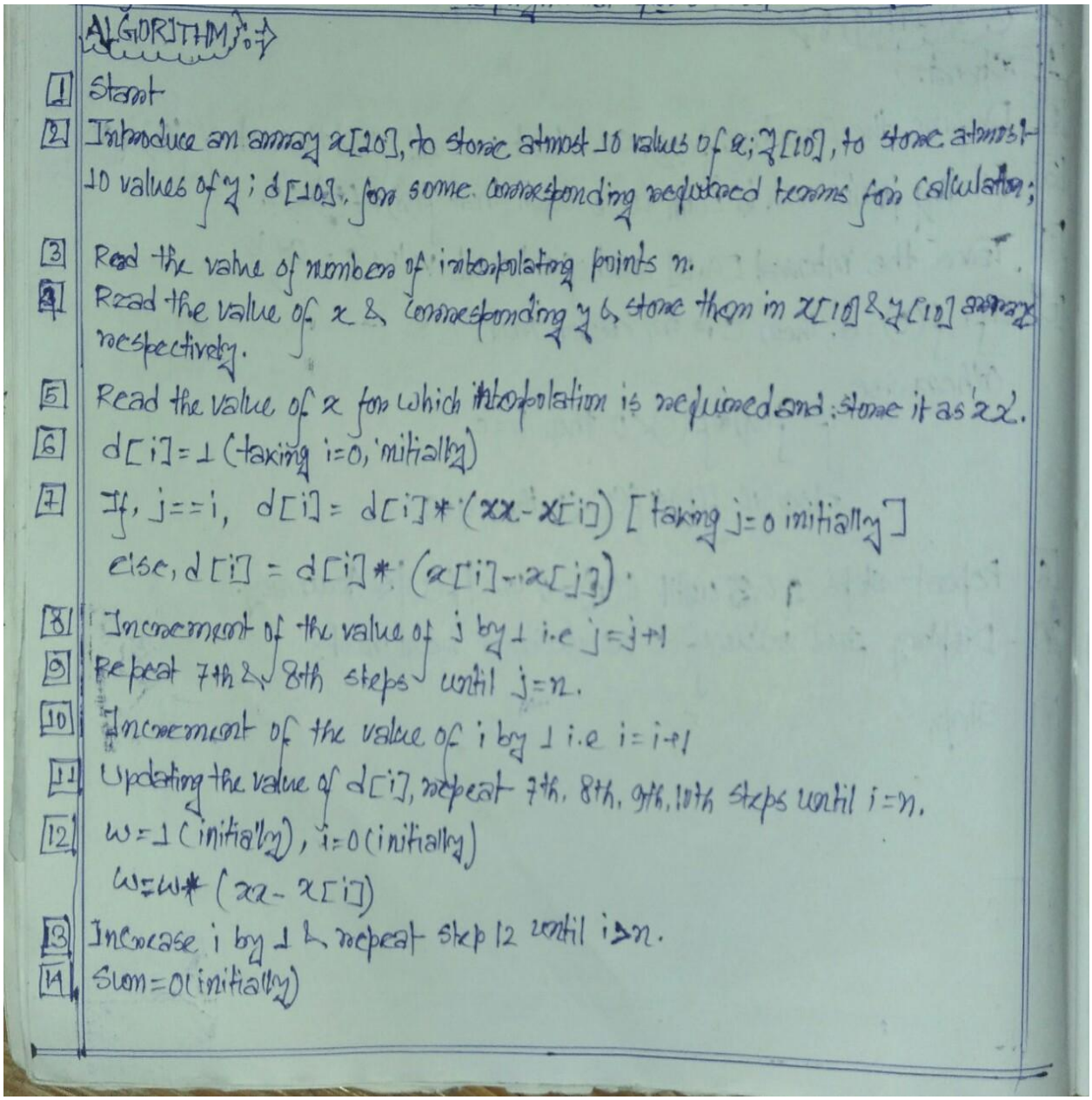
$$L_n(x) = \sum_{i=0}^{n} \frac{(x-x_0)(x-x_1)\cdots(x-x_n)}{\left[(x-x_i)(x_i-x_1)(x_i-x_2)\cdots\cdots\cdots(x_i-x_{i-1})(x_i-x_{i+1})\cdots(x_i-x_n)\right]} f(x_i)$$

NOTE: $L_n(x)$ remain unchanged (invariant) under linear transformation.

Then replacing $x$ by $t (\in [x_0, x_n])$ we can easily find the value of $L_n(t) = f(t)$, the desired one.

- Lagrange's polynomial is unique.
- The polynomial depends on $x_i$'s not on $y_i$'s.

- ***ALGORITHM***:-



ALGORITHM :→

1. Start
2. Introduce an array x[20], to store atmost 10 values of x; y[10], to store atmost 10 values of y; d[10], for some corresponding required terms for calculation;
3. Read the value of number of interpolating points n.
4. Read the value of x & corresponding y & store them in x[10] & y[10] arrays respectively.
5. Read the value of x for which interpolation is required and store it as 'xx'.
6. d[i] = 1 (taking i=0, initially)
7. If, j==i, d[i] = d[i] * (xx - x[i]) [taking j=0 initially]
   else, d[i] = d[i] * (x[i] - x[j]).
8. Increament of the value of j by 1 i.e j = j+1
9. Repeat 7th & 8th steps until j=n.
10. Increment of the value of i by 1 i.e i = i+1
11. Updating the value of d[i], repeat 7th, 8th, 9th, 10th steps until i=n.
12. w = 1 (initially), i=0 (initially)
    w = w * (xx - x[i])
13. Increase i by 1 & repeat step 12 until i>n.
14. sum = 0 (initially)

15 i:=0(initially), increase i by 1 & repeat step 16 until i>n.

16 sum=sum+ x[i]/d[i].

17 sum = w * sum

18 Print the value of sum as final result.

19 Stop.

- ***QUESTION:-***

***(01)Write a C program to find the value of f(x) at x=0.75+(R+1)/100,and also at x=0.90+(R+1)/100 using suitable interpolation formula from the following interpolation table:***

| *x* | *f(x)* |
|---|---|
| *0.15* | *0.1407825446* |
| *0.30* | *0.1470414286* |
| *0.45* | *0.2131583482* |
| *0.60* | *0.3113257143* |
| *0.75* | *0.4254520089* |
| *0.90* | *0.5550671429* |
| *1.05* | *0.7192278125* |
| *1.20* | *0.9604228571* |
| *1.35* | *1.3484786161* |
| *1.50* | *1.9844642857* |

***[R being last digit of university roll number]***

***Answer:***

***[The last digit of my university roll number being 6,I will use R=6;and according to the definition of x in the first case it will take value x=0.82 and secondly it will take value x=0.97.So in the following program I will give input for xx as 0.82 to find the corresponding f(x) and will find same for second time by inputting xx=0.97]***

***[NOTE:We will use Lagrange interpolation formula here]***

```
//Lagrange Interpolation Formula //
#include<stdio.h>
#include<math.h>
void main()
{
	float x[10],y[10],d[10],sum,w,xx;
	int n,i,j;
	printf("\n Enter the number of interpolating points:
");
	scanf("%d",&n);
	n=n-1;
	printf("\n Enter the interpolating points \n ");
	for(i=0;i<n+1;i++)
	scanf("%f",&x[i]);
	printf("\n Enter the values of y\n");
	for(i=0;i<n+1;i++)
	scanf("%f",&y[i]);
	printf ("\n Enter the point at which interpolation is
required\n ");
	scanf("%f",&xx);
	for(i=0;i<n+1;i++)
	{
		d[i]=1;
		for(j=0;j<n+1;j++)
		{
			if(j==i)
			d[i]=d[i]*(xx-x[i]);
			else
			d[i]=d[i]*(x[i]-x[j]);
		}
```

```
        }
        w=1;
        for(i=0;i<n+1;i++)
        w=w*(xx-x[i]);
        sum=0;
        for(i=0;i<n+1;i++)
        sum=sum+(y[i]/d[i]);
        sum=sum*w;
        printf("\n The value of y(%0.4f) is %0.10f(correct
upto 10 decimal places)",xx,sum);

}
```

- ***USER INTERFACE:-***

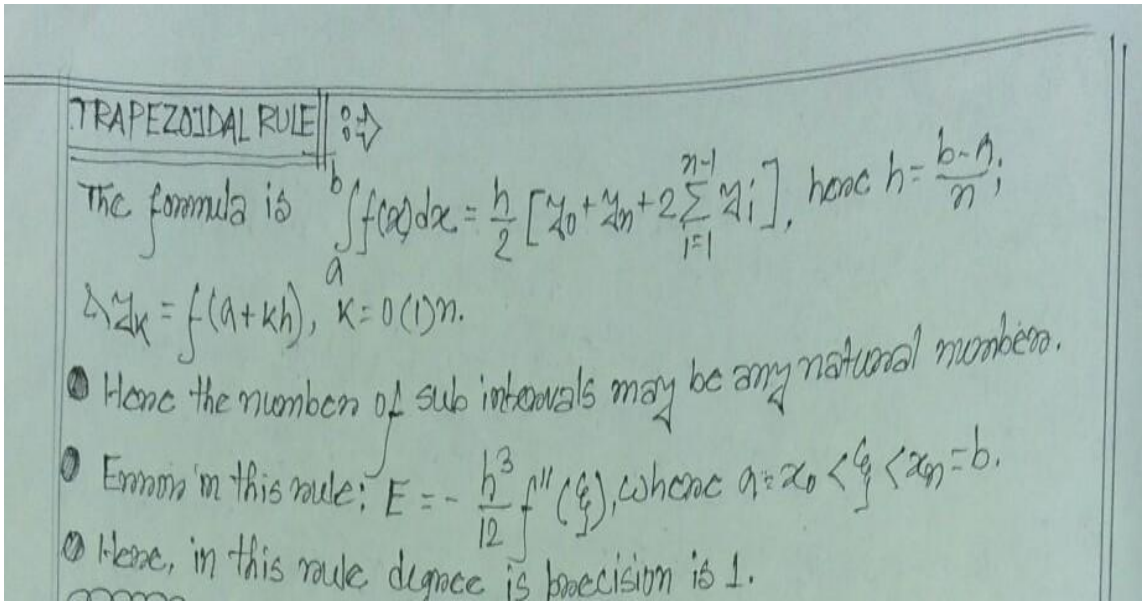
```
Enter the number of interpolating points: 10

Enter the interpolating points
0.15
0.30
0.45
0.60
0.75
0.90
1.05
1.20
1.35
1.50

Enter the values of y
0.1407825446
0.1470414286
0.2131583482
0.3113257143
0.4254520089
0.5550671429
0.7192278125
0.9604228571
1.3484786161
1.9844642857

Enter the point at which interpolation is required
0.82

The value of y(0.8200) is 0.4833524227(correct upto 10 deci
mal places)
[Program finished]
```

```
Enter the number of interpolating points: 10

Enter the interpolating points
0.15
0.30
0.45
0.60
0.75
0.90
1.05
1.20
1.35
1.50

Enter the values of y
0.1407825446
0.1470414286
0.2131583482
0.3113257143
0.4254520089
0.5550671429
0.7192278125
0.9604228571
1.3484786161
1.9844642857

Enter the point at which interpolation is required
0.97

The value of y(0.9700) is 0.6253829598(correct upto 10 deci
mal places)
[Program finished]
```

**(05)**

## TRAPEZOIDAL RULE

- ***Introduction and Working Formula:-***



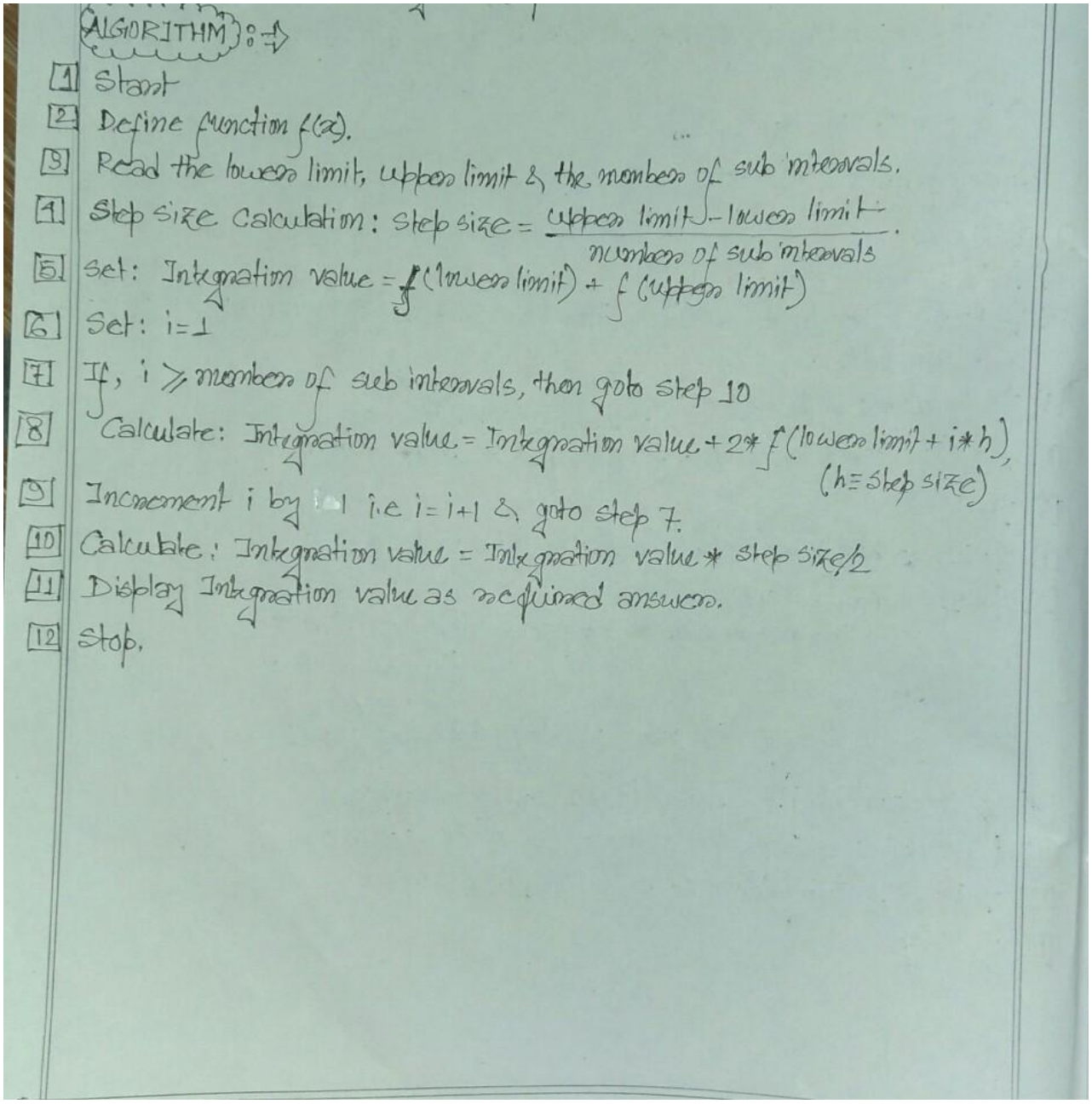
TRAPEZOIDAL RULE ⇒

The formula is $\int_a^b f(x)\,dx = \frac{h}{2}\left[y_0 + y_n + 2\sum_{i=1}^{n-1} y_i\right]$, hence $h = \frac{b-a}{n}$;

$y_k = f(a+kh)$, $k = 0(1)n$.

- Hence the number of sub intervals may be any natural numbers.
- Error in this rule: $E = -\frac{h^3}{12} f''(\xi)$, where $a = x_0 < \xi < x_n = b$.
- Here, in this rule degree of precision is 1.

- ***Algorithm:-***



ALGORITHM :⇒

1. Start
2. Define function $f(x)$.
3. Read the lower limit, upper limit & the number of sub intervals.
4. Step size Calculation: Step size = $\frac{\text{upper limit} - \text{lower limit}}{\text{number of sub intervals}}$.
5. Set: Integration value = $\int f(\text{lower limit}) + f(\text{upper limit})$
6. Set: $i = 1$
7. If, $i \geq$ number of sub intervals, then goto step 10
8. Calculate: Integration value = Integration value + $2 * f(\text{lower limit} + i * h)$, ($h$ = step size)
9. Increment $i$ by 1 i.e $i = i+1$ & goto step 7.
10. Calculate: Integration value = Integration value * step size/2
11. Display Integration value as required answer.
12. Stop.

- ***Question:-***

**Use trapezoidal rule to find out the value of the integration up to 5 decimal places with 13 points:**

$$\int_{15^o}^{60^o} \frac{2x+psinx}{\sqrt{x^2+qcos^2x+1.0}}dx$$

$$p = 1 + 0.1R$$

$$q = 1 - 0.1R$$

[Here R is the last digit of my university roll number. ]

***Answer:-***

***[The last digit of my CU roll-number being 6,according to the definitions of p, q their value will be 1.6 and 0.4 respectively and the time of writing function definition in the program I will use these values of p and q respectively in corresponding positions]***

```
//Trapezoidal Rule//
#include<stdio.h>
#include<math.h>
float p=1.6,q=0.4;
int main()
{
double pi, a, b, h, sum=0.0, f(double);
int i, n;
```

```
pi=4*atan(1);
printf("\n Enter the lower limit: ");
scanf("%lf",&a);
a=pi*(a/(180*1.0));
printf("\n Enter the upper limit: ");
scanf("%lf",&b);
b=pi*(b/(180*1.0));
sum=f(a)+f(b);
printf("\n Enter the number of intervals: ");
scanf("%d",&n);
h=(b-a) /(n*1.0);
for(i=1;i<n;i++)
{
sum=sum+2*(f(a+i*h));
}
printf("\n The value of the integral correct upto 5
decimal places is %0.5f",((h/(2*1.0))*sum));
return 0;
}
double f(double x)
{
return
(((2*x)+(p*sin(x)))/sqrt(pow(x,2)+(q*pow(cos(x),2))+1.0))
;}
```

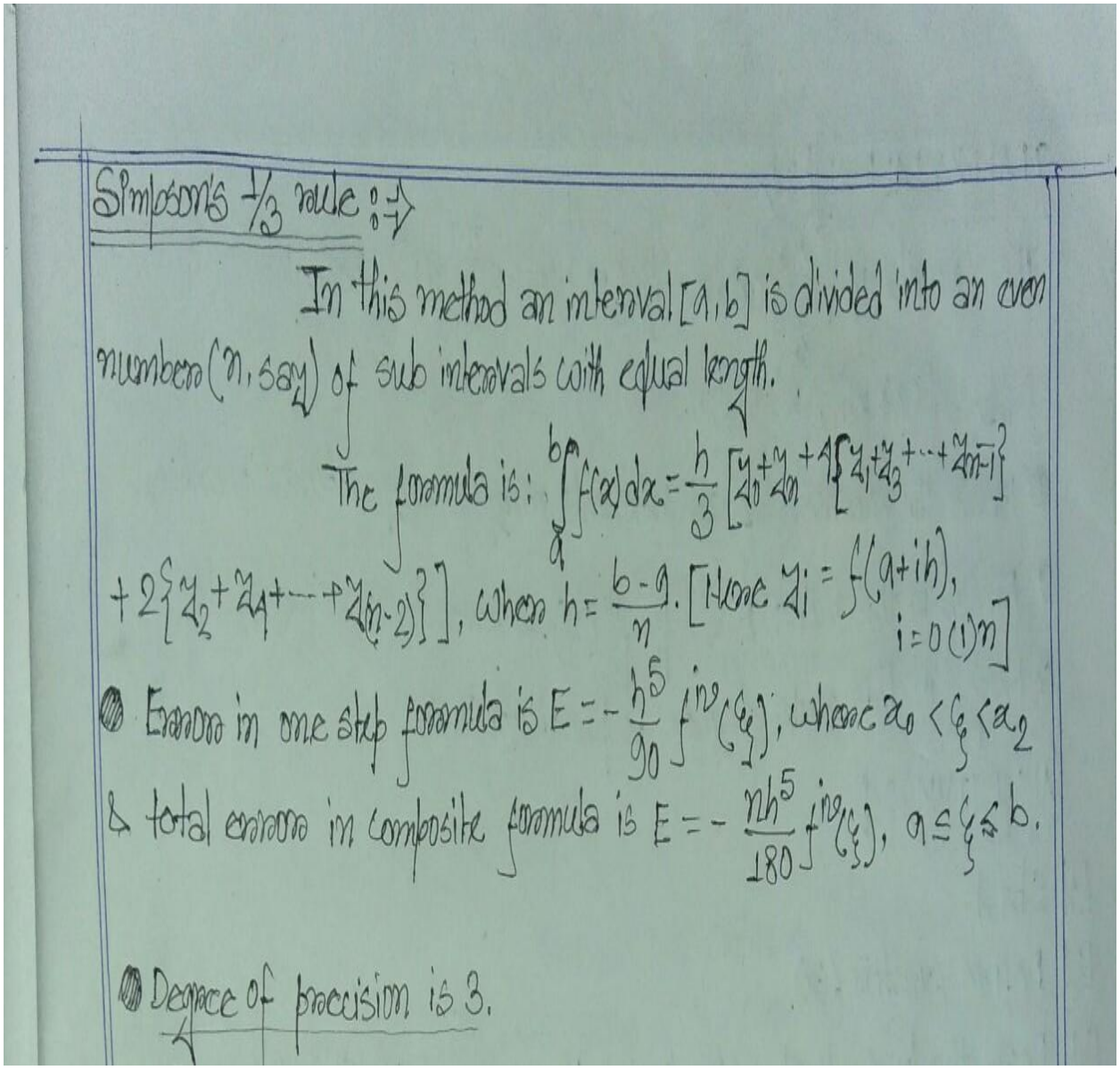
- ***USER SCREEN:-***

```
 Enter the lower limit: 15

 Enter the upper limit: 60

 Enter the number of intervals: 12

 The value of the integral correct upto 5 decimal places is
1.33013
[Program finished]
```

***(06)***

## SIMSON ONE THIRD RULE

- ***INTRODUCTION AND THEORY:-***



Simpson's 1/3 rule :⇒

In this method an interval $[a, b]$ is divided into an even number $(n, \text{say})$ of sub intervals with equal length.

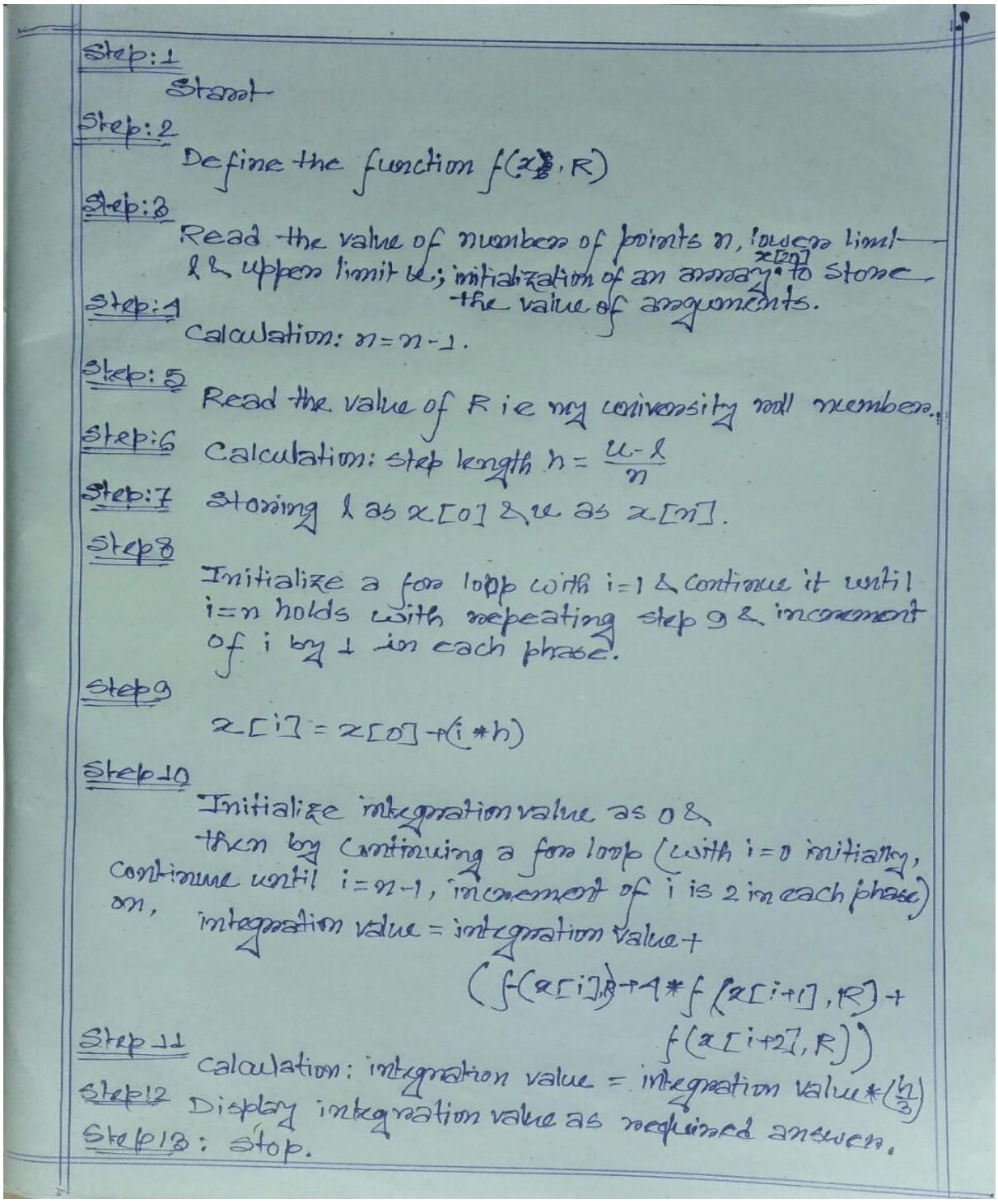
The formula is: $\int_a^b f(x)\,dx = \frac{h}{3}\left[y_0 + y_{2n} + 4\{y_1 + y_3 + \cdots + y_{2n-1}\} + 2\{y_2 + y_4 + \cdots + y_{(n-2)}\}\right]$, where $h = \frac{b-a}{n}$. [Here $y_i = f(a+ih)$, $i = 0(1)n$]

- Error in one step formula is $E = -\frac{h^5}{90} f^{iv}(\xi)$, where $x_0 < \xi < x_2$

& total error in composite formula is $E = -\frac{nh^5}{180} f^{iv}(\xi)$, $a \le \xi \le b$.

- Degree of precision is 3.

- ***ALGORITHM:-***



Step:1
Start
Step:2
Define the function $f(x, R)$
Step:3
Read the value of number of points $n$, lower limit $l$ & upper limit $u$; initialization of an array $x[20]$ to store the value of arguments.
Step:4
Calculation: $n = n - 1$.
Step:5
Read the value of R i.e my university roll number.
Step:6
Calculation: step length $h = \frac{u-l}{n}$
Step:7
storing $l$ as $x[0]$ & $u$ as $x[n]$.
Step8
Initialize a for loop with $i=1$ & continue it until $i=n$ holds with repeating step 9 & increment of i by 1 in each phase.
Step9
$x[i] = x[0] + (i * h)$
Step10
Initialize integration value as 0 & then by continuing a for loop (with $i=0$ initially, continue until $i=n-1$, increment of i is 2 in each phase) on, integration value = integration value + $(f(x[i], R) + 4 * f(x[i+1], R) + f(x[i+2], R))$
Step11
calculation: integration value = integration value $* (\frac{h}{3})$
Step12 Display integration value as required answer.
Step13: stop.

- ***Question:-***

**Use Simpson's 1/3rd rule to find out the value of the integration up to 5 decimal places with 13 points:**

$$\int_{0.2}^{2.6} \frac{1+sinbxcos(1.1+bx)}{1+bx+x^2} dx$$

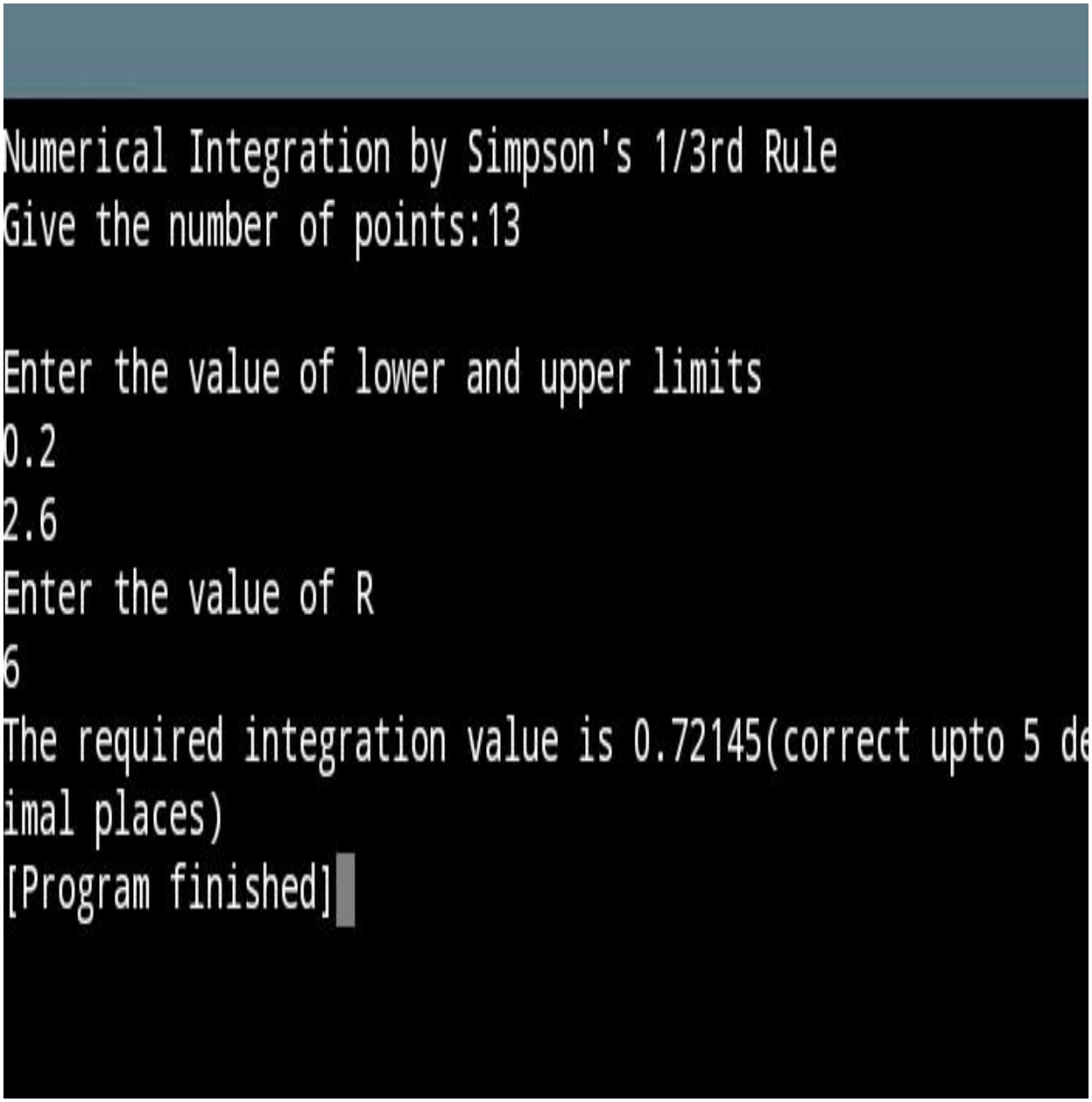
(Where b=0.4+(1+R)/50;R being last digit of my university roll number.)

## C Program for Solving Above Problem:-

```
//Simpson One third rule//
#include <stdio.h>
#include<math.h>
float f(float x,int R)
{
    float y,b;
    b=0.4+((1.+R)/50);
    //printf("%f ",b);
    y=(1.+sin(b*x)*cos(1.1+(b*x)))/(1+(b*x)+pow(x,3));
    return (y);
}
int main()
{
```

```
float a, c, h, sum;
int n, i;
printf("\n Enter the lower and upper limits: ");
scanf("%f%f",&a, &c);
printf("\n Enter number of intervals:.");
scanf("%d",&n);
h=(c-a) /n;
sum=f(a)+f(c);
for(i=1;i<n;i=i+1)
{
if(i%2==0)
sum=sum+(2*f(a+(i*h)));
else
sum=sum+(4*(f(a+(i*h))));
}
printf("\n The value of the integral correct upto 5 decimal
places =%0.5f",sum);
return 0;
}
```

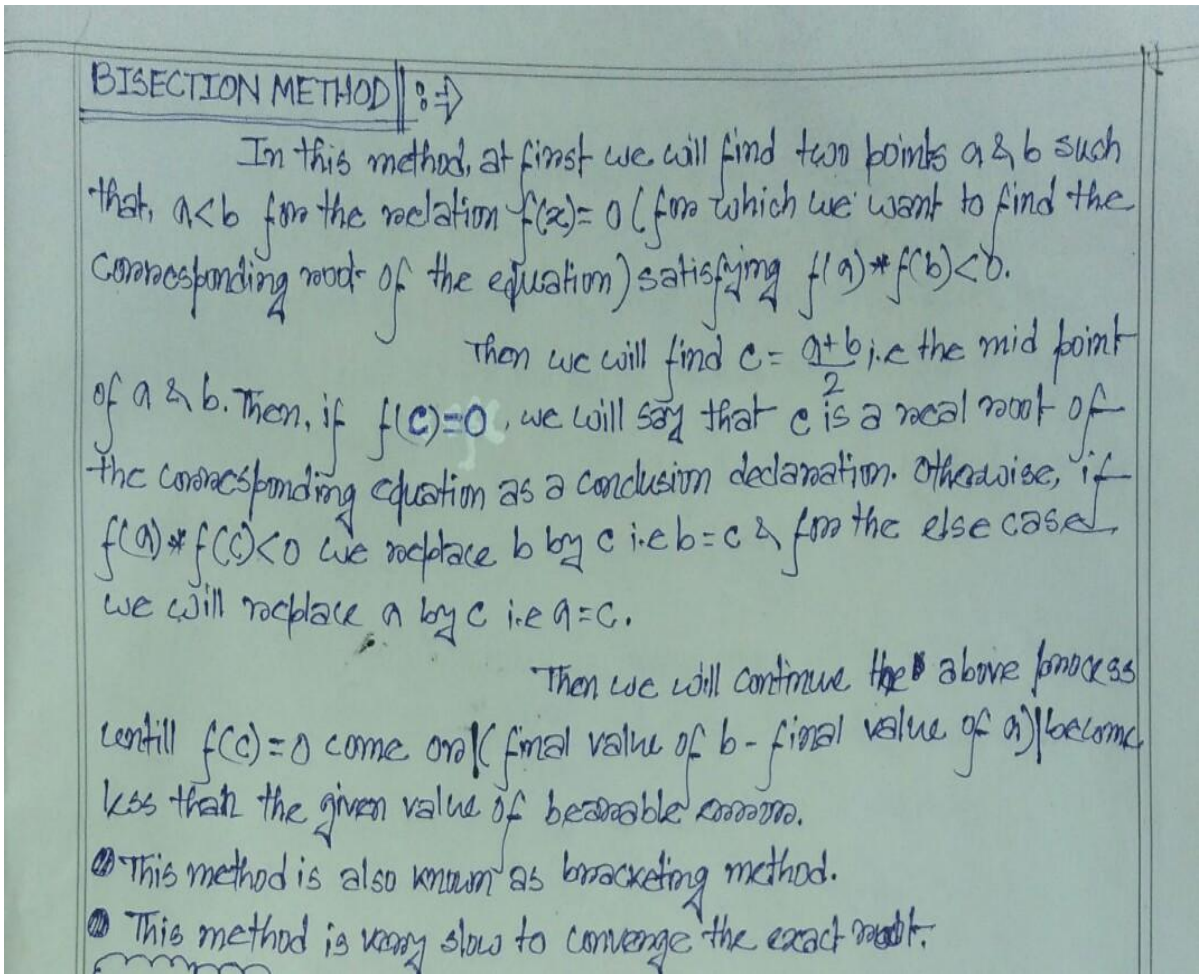
- ***User Interface:-***



**(07)**

# BISECTION METHOD

- ***Introduction and Working Formula:-***



BISECTION METHOD :⇒

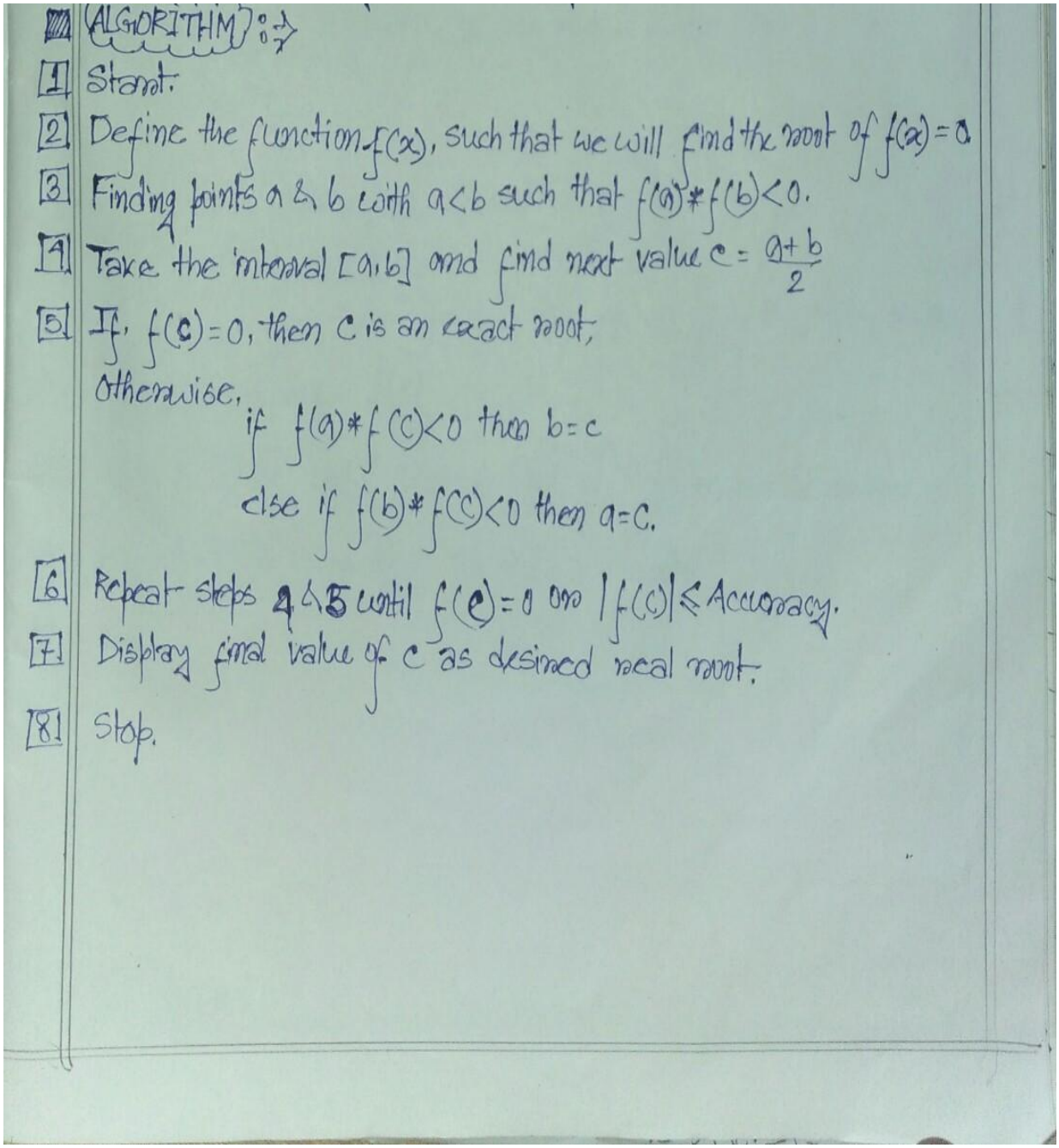
In this method, at first we will find two points a & b such that, a<b for the relation f(x)= 0 (for which we want to find the corresponding root of the equation) satisfying f(a)*f(b)<0.

Then we will find c = (a+b)/2 i.e the mid point of a & b. Then, if f(c)=0, we will say that c is a real root of the corresponding equation as a conclusion declaration. Otherwise, if f(a)*f(c)<0 we replace b by c i.e b=c & for the else case we will replace a by c i.e a=c.

Then we will continue the above process until f(c)=0 come on |(final value of b - final value of a)| becomes less than the given value of bearable error.

- This method is also known as bracketing method.
- This method is very slow to converge the exact result.

- ***Algorithm:-***



[ALGORITHM] :->

1. Start.
2. Define the function $f(x)$, such that we will find the root of $f(x) = 0$
3. Finding points $a$ & $b$ with $a < b$ such that $f(a) * f(b) < 0$.
4. Take the interval $[a, b]$ and find next value $c = \frac{a+b}{2}$
5. If, $f(c) = 0$, then $c$ is an exact root,
   Otherwise,
   if $f(a) * f(c) < 0$ then $b = c$
   else if $f(b) * f(c) < 0$ then $a = c$.
6. Repeat steps 4 & 5 until $f(c) = 0$ or $|f(c)| \leq$ Accuracy.
7. Display final value of $c$ as desired real root.
8. Stop.

- ***Question:-***

**Find the smallest possible root of the following equation with four decimal places with Bisection method:**

$$3(2 - px)^2 + xcos(2 + px) = 2, p = 1.5 + \frac{R}{20}$$

**R is the last of the University Roll Number.**

***Answer*:-**

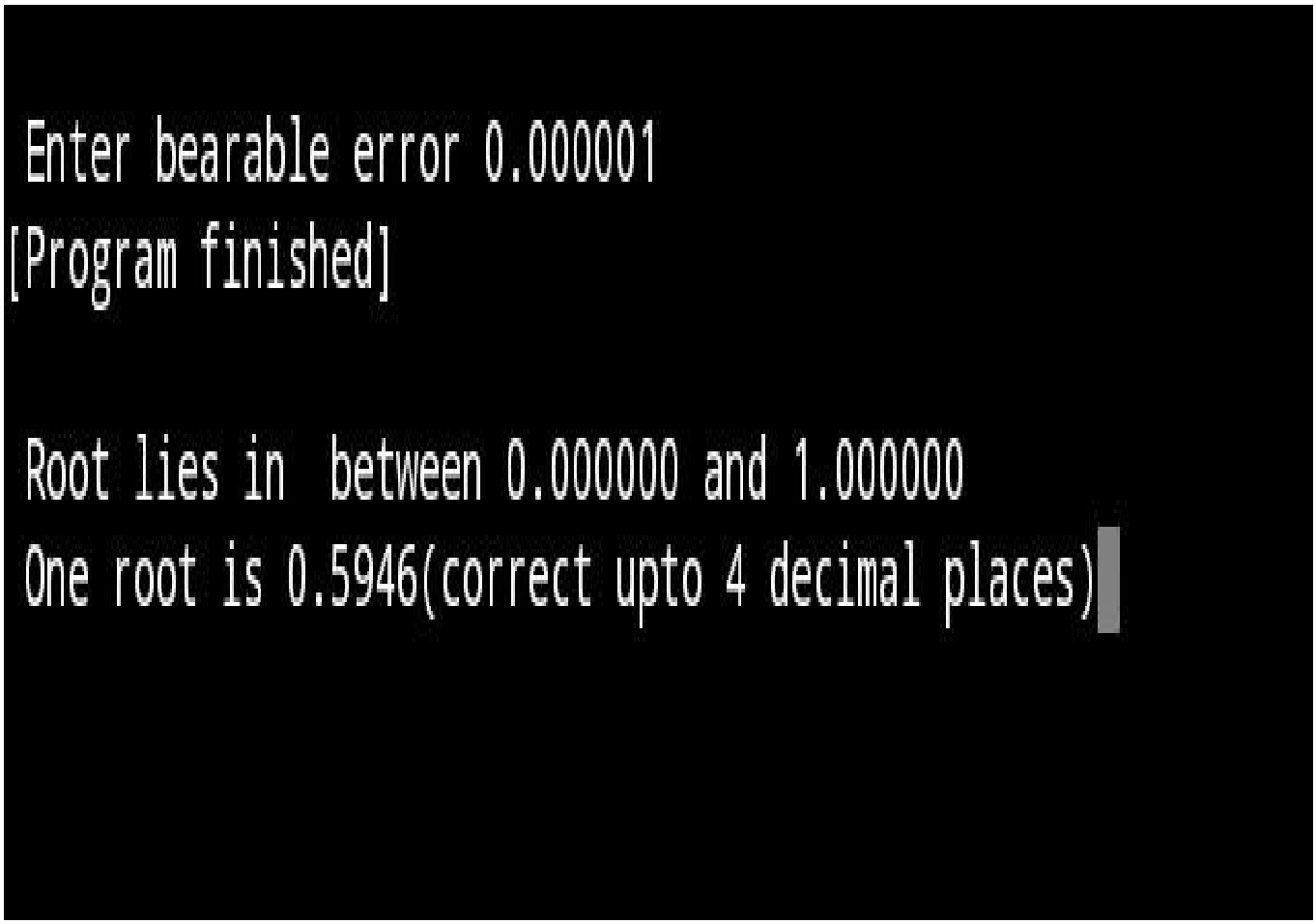
**[The last digit of my CU roll number being R=6,according to the definition of p, the value of p will be 1.8.]**

```
//Bisection Method//
#include<stdio.h>
#include<math.h>
float p=1.8;
float f(float x)
{
    return 3*pow(2-(p*x),2)+x*cos(2+(p*x))-2;
}
int main()
{
    float a=0.0,b,c,error;
    int i=0;
    printf("\n Enter bearable error");
    scanf("%f",&error);
```

```
    do
    {
        a=a+pow(-1,i)*i;
        b=a+1;
        if(f(a)==0)
        {
            printf("\n One root is %0.4f(correct
upto 4 decimal places)",a);
            break;
        }
        else
        {
        if(f(b)==0)
        {
            printf("\n One root is %0.4f(correct
upto 4 decimal places)",b);
            break;
        }
        }
        i++;
    }while(f(a)*f(b)>0);
    printf("\n Root lies in  between %f and
%f",a,b);
    do
```

```
    {
        c=(a+b)/2.0;
        if(f(c)==0)
        {
                printf("\n One root is
%0.4f(correct upto 4 decimal places)",c);
                break;
        }
        else
        if(f(a)*f(c)<0)
        b=c;
        else
        a=c;
    }while(fabs(f(c))>error);
    printf("\n One root is %0.4f(correct upto 4
decimal places)",c);
    return 0;
}
```

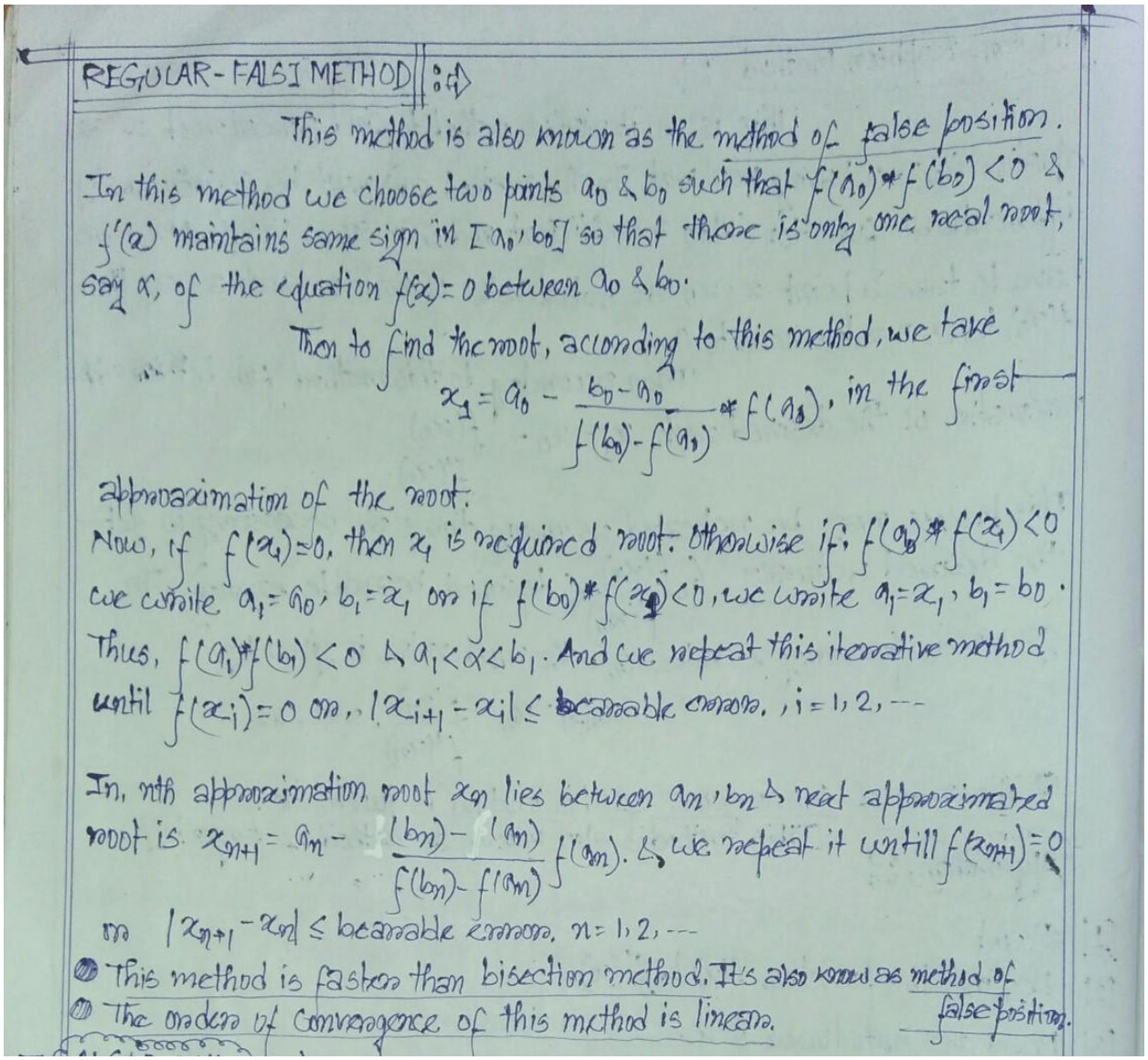
- ***User Interface:-***



**(08)**

## REGULAR FALSI METHOD

- ***Introduction and Working Formula:-***



REGULAR-FALSI METHOD :-

This method is also known as the method of false position.

In this method we choose two points $a_0$ & $b_0$ such that $f(a_0) * f(b_0) < 0$ & $f'(x)$ maintains same sign in $I[a_0, b_0]$ so that there is only one real root, say $\alpha$, of the equation $f(x) = 0$ between $a_0$ & $b_0$.

Then to find the root, according to this method, we take

$$x_1 = a_0 - \frac{b_0 - a_0}{f(b_0) - f(a_0)} * f(a_0),$$ in the first approximation of the root.

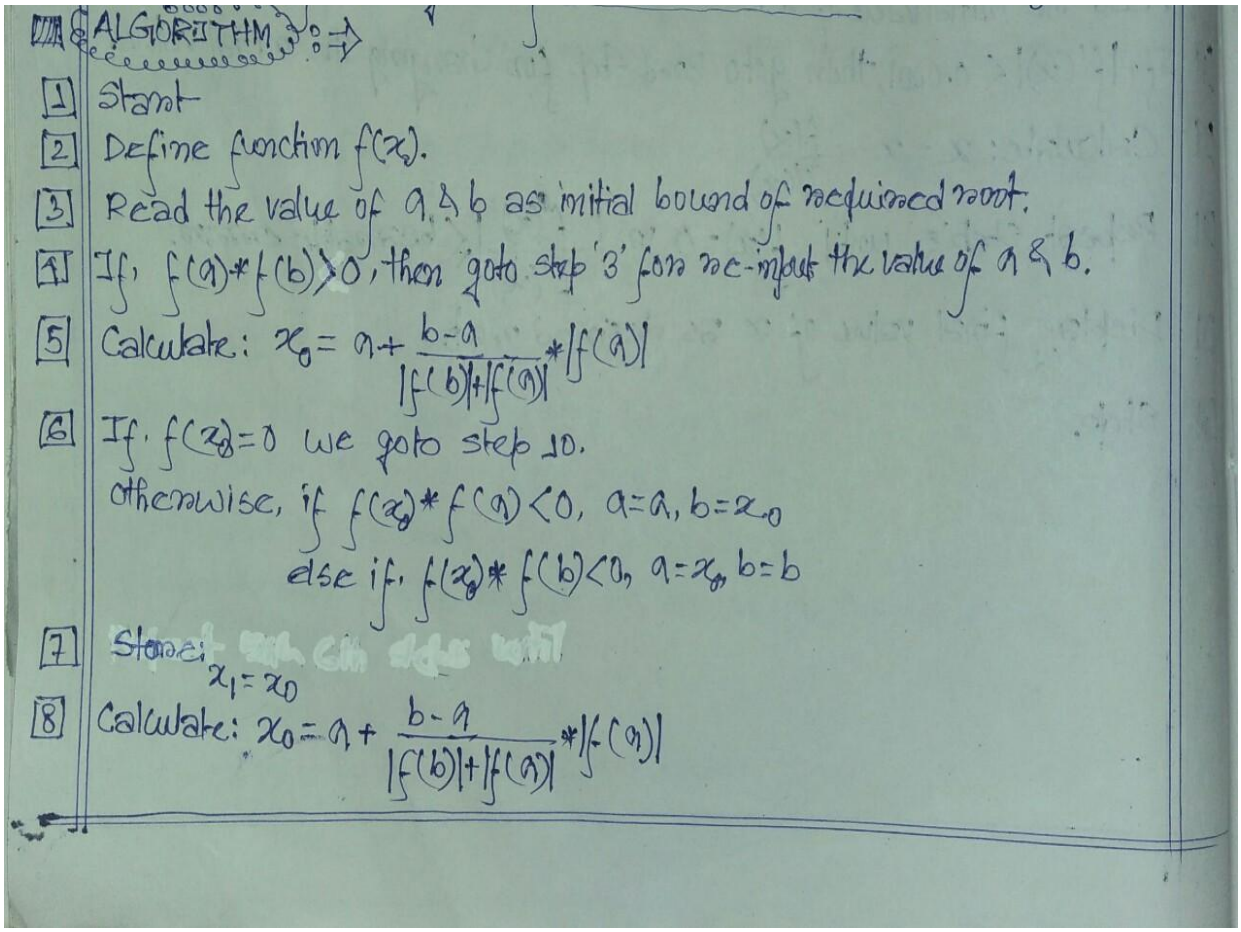
Now, if $f(x_1) = 0$, then $x_1$ is required root. Otherwise if $f(a_0) * f(x_1) < 0$ we write $a_1 = a_0$, $b_1 = x_1$ or if $f(b_0) * f(x_1) < 0$, we write $a_1 = x_1$, $b_1 = b_0$.

Thus, $f(a_1) * f(b_1) < 0 \Rightarrow a_1 < \alpha < b_1$. And we repeat this iterative method until $f(x_i) = 0$ or, $|x_{i+1} - x_i| \leq$ bearable errors, $i = 1, 2, \ldots$

In, nth approximation root $x_n$ lies between $a_n$, $b_n$ & next approximated root is $x_{n+1} = a_n - \frac{(b_n) - f(a_n)}{f(b_n) - f(a_n)} f(a_n)$. & we repeat it untill $f(x_{n+1}) = 0$ or $|x_{n+1} - x_n| \leq$ bearable errors, $n = 1, 2, \ldots$

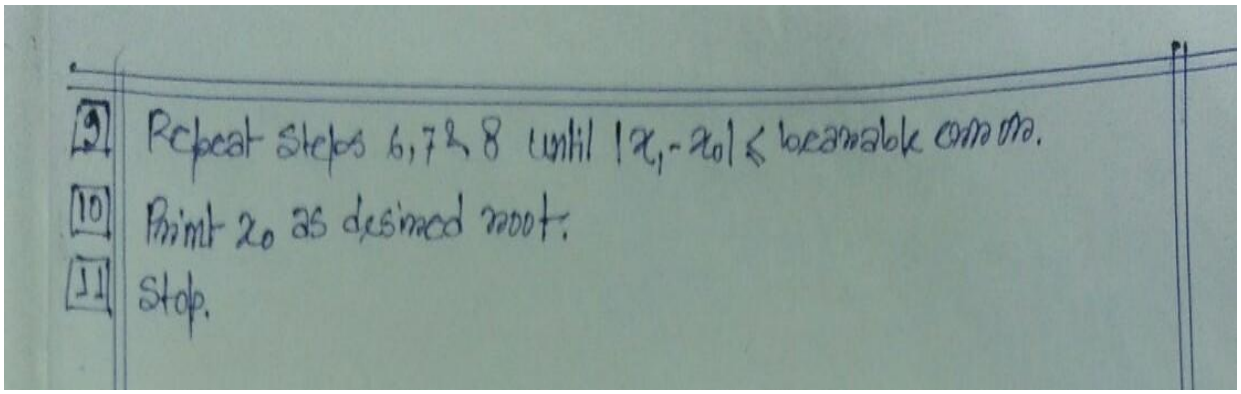
- This method is faster than bisection method. It's also known as method of false position.
- The order of convergence of this method is linear.

- ***Algorithm:-***



ALGORITHM :⇒

1. Start
2. Define function $f(x)$.
3. Read the value of $a$ & $b$ as initial bound of required root.
4. If, $f(a)*f(b)>0$, then goto step '3' for re-input the value of $a$ & $b$.
5. Calculate: $x_0 = a + \frac{b-a}{|f(b)|+|f(a)|} * |f(a)|$
6. If, $f(x_0)=0$ we goto step 10.
   otherwise, if $f(x_0)*f(a)<0$, $a=a, b=x_0$
   else if, $f(x_0)*f(b)<0$, $a=x_0, b=b$
7. Store: $x_1 = x_0$
8. Calculate: $x_0 = a + \frac{b-a}{|f(b)|+|f(a)|} * |f(a)|$



9. Repeat Steps 6, 7 & 8 until $|x_1 - x_0| \leq$ tolerable error.
10. Print $x_0$ as desired root.
11. Stop.

- ***Question:-***

**Write a C program to find to compute a real root of the followings equation by Regular Falsi Method correct upto 4 decimal places.**
**sin(p*x)-(p*x*x)=x-0.5**
**Where p=1.5+(R/20)(R being last digit of university roll number).**

**Answer:-**

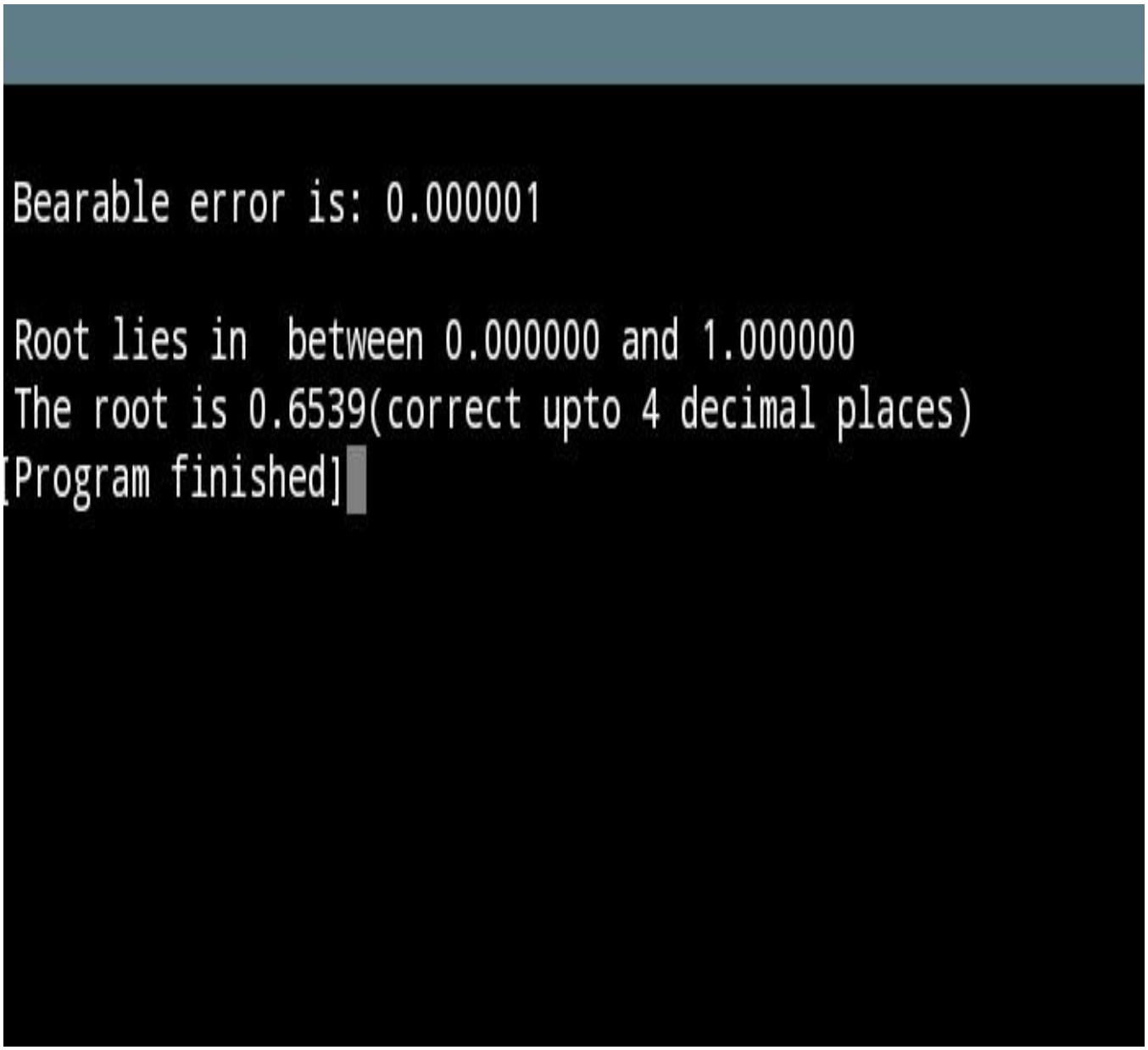
**[The last digit of my university roll number being 6,I will take R=6 and according to the definition of p, it's value will be p=1.8]**

```
//Regular Falsi Method//
#include<stdio.h>
#include<math.h>
float f(float x)
{
return sin(1.8*x)-(1.8*(x*x))-x+0.5
}
int main()
{
float a, b, x1,x0,error;
int i=0;
printf("\n Bearable error is: ");
scanf("%f",&error);
```

```
do
    {
        a=a+pow(-1,i)*i;
        b=a+1;
        if(f(a)==0)
        {
            printf("\n One root is %0.4f(correct
upto 4 decimal places)",a);
            break;
        }
        else
        {
        if(f(b)==0)
        {
            printf("\n One root is %0.4f(correct
upto 4 decimal places)",b);
            break;
        }
        }
        i++;
    }while(f(a)*f(b)>0);
    printf("\n Root lies in  between %f and
%f",a,b);
x0=a+((b-a) *(fabs(f(a)))/(fabs(f(a)+fabs(f(b)))));
```

```
do
{
if(f(a)*f(x0)<0)
b=x0;
else
a=x0;
x1=x0;
x0=a+((b-a) *(fabs(f(a)))/(fabs(f(a)+fabs(f(b)))));
}while(fabs(x1-x0)>error);
printf("\n The root is %0.4f(correct upto 4
decimal places) ",x0);
return 0;
}
```

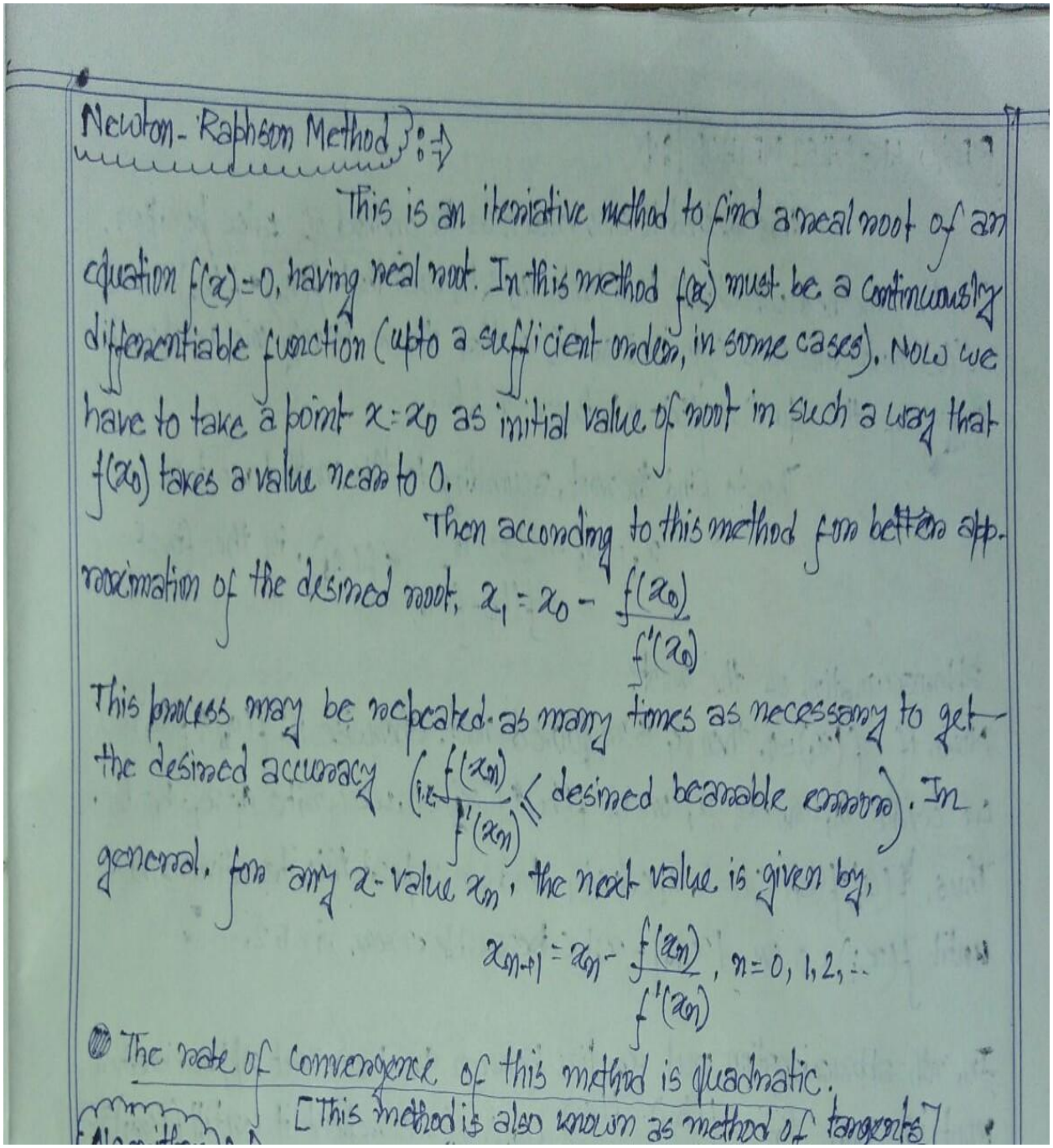
- ***User Interface:-***



**(09)**

# NEWTON RAPHSON METHOD

- ***Introduction and Working Formula:-***



Newton-Raphson Method :⇒

This is an iterative method to find a real root of an equation $f(x)=0$, having real root. In this method $f(x)$ must be a continuously differentiable function (upto a sufficient order, in some cases). Now we have to take a point $x=x_0$ as initial value of root in such a way that $f(x_0)$ takes a value near to 0.

Then according to this method fom better approximation of the desired root, $x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$

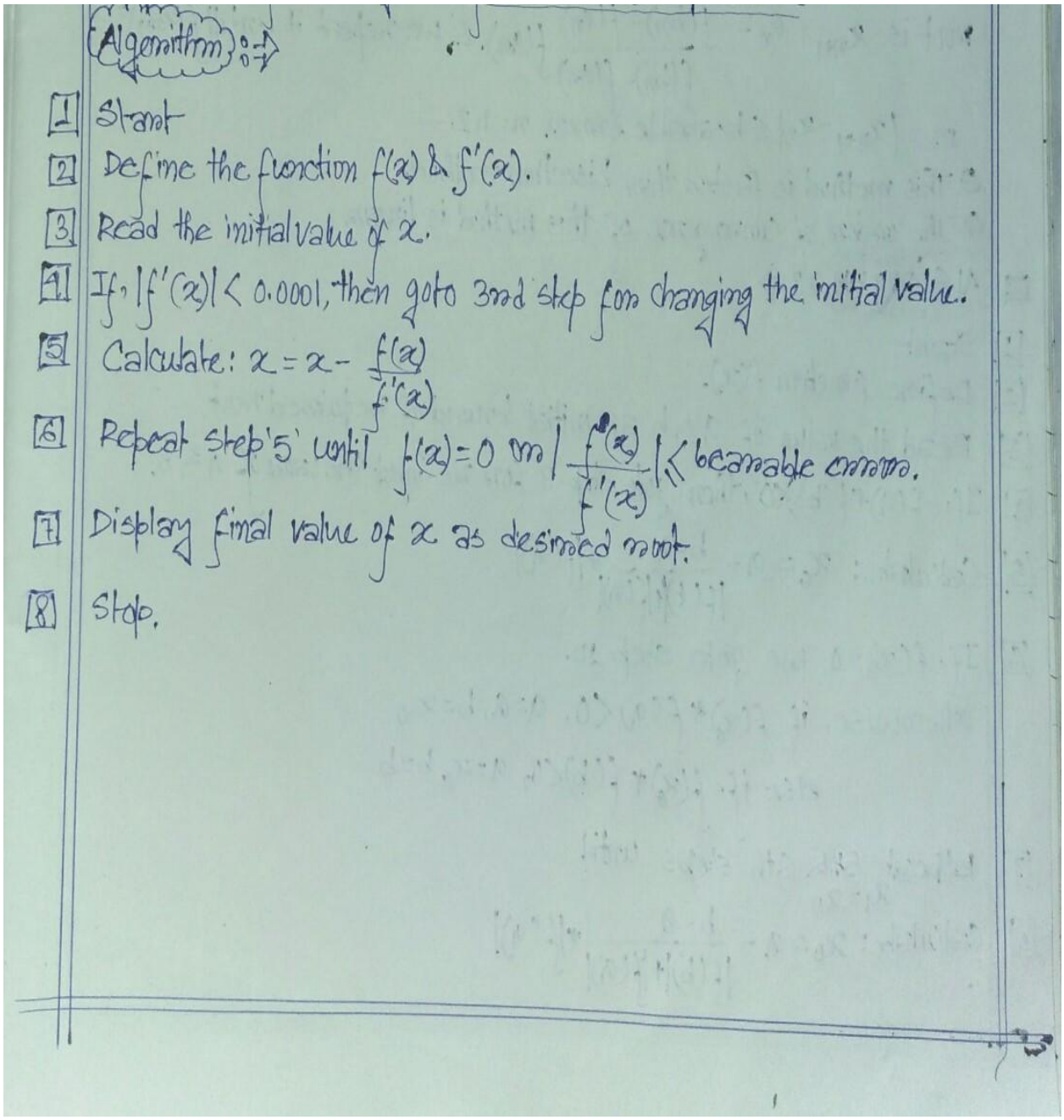
This process may be repeated as many times as necessary to get the desired accuracy $\left(\text{i.e. } \left|\frac{f(x_n)}{f'(x_n)}\right| < \text{desired bearable error}\right)$. In general, for any x-value $x_n$, the next value is given by,

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, \quad n = 0, 1, 2, \ldots$$

The rate of convergence of this method is quadratic.

[This method is also known as method of tangents]

- ***Algorithm:-***



Algorithm :⇒

1. Start
2. Define the function $f(x)$ & $f'(x)$.
3. Read the initial value of $x$.
4. If, $|f'(x)| < 0.0001$, then goto 3rd step for changing the initial value.
5. Calculate: $x = x - \frac{f(x)}{f'(x)}$
6. Repeat step '5' until $f(x) = 0$ or $\left|\frac{f(x)}{f'(x)}\right| <$ desirable error.
7. Display final value of $x$ as desired root.
8. Stop.

- ***Question:-***
- **Question:-Write a C program to compute a real root of the following equation correct upto 5 decimal places, by Newton Raphson method.**

**(x*x)+tanh(x)-exp(p*sinx)-3=0,where p=1+(R/20),R being last digit of university roll number.**

*Answer:-*

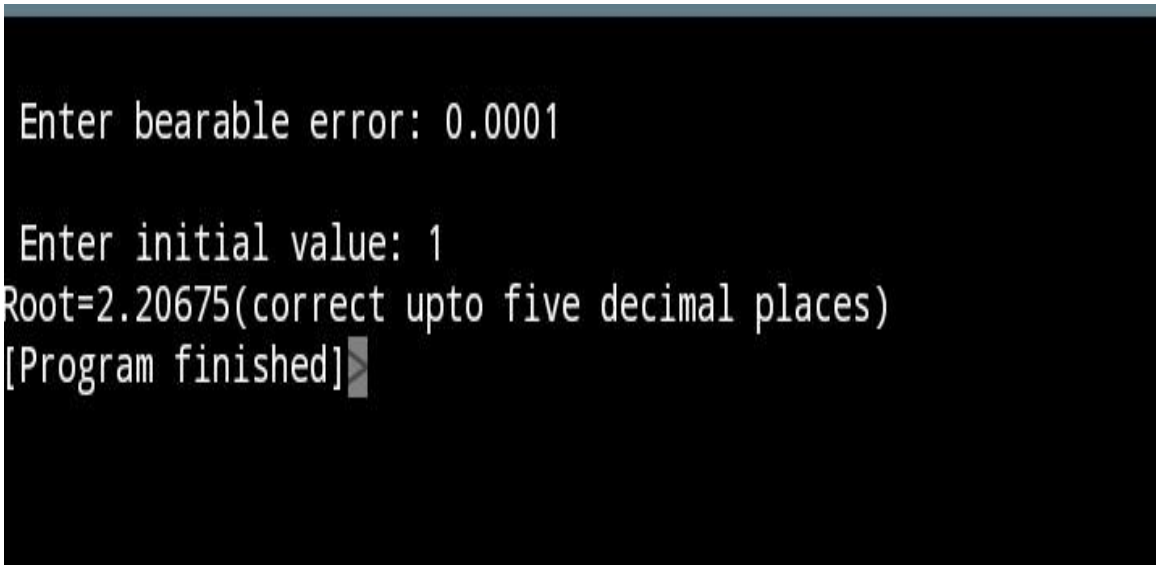
*The last digit of my roll number being 6,R=6 and consequently p=1.3.*

```
//Newton Raphson Method//

#include<stdio.h>

#include<math.h>

int main()

{

float x, h, error,f(float),f1(float);

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

scanf("%f",&error);

printf("\n Enter initial value: ");
```

```
scanf("%f",&x);

if(fabs(f1(x))<0.00001)

{printf("\n change the initial value \n");}

else

{

h=-f(x)/f1(x);

while(fabs(h)>error)

{

x=x+h;

h=-f(x)/f1(x);

}

printf("Root=%0.5f(correct upto five decimal places) ",x);

}

return 0;

}
```

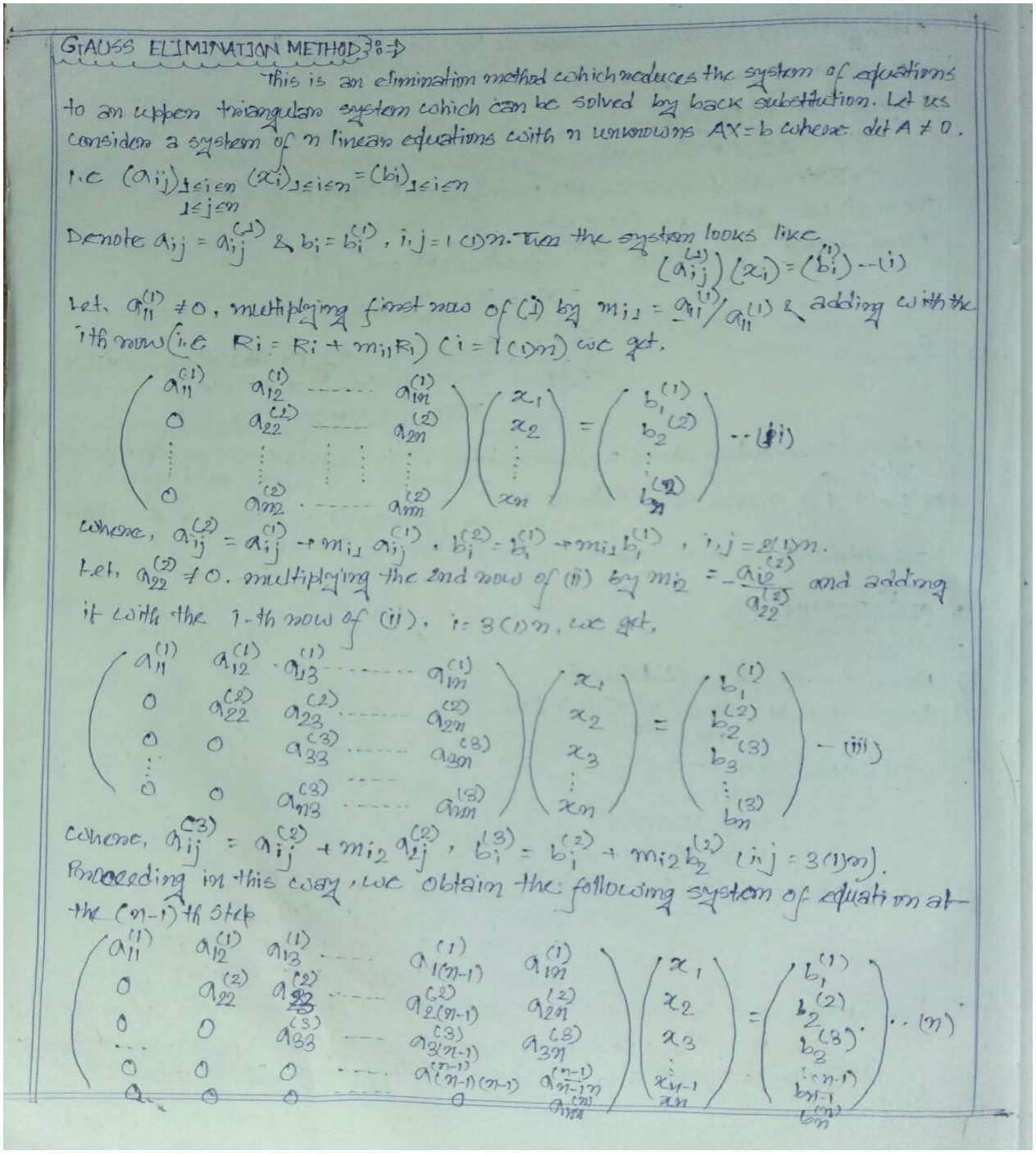
```
float f(float x)

{

return((x*x)+tanh(x)-exp(1.3*sin(x))-3);

}

float f1(float x)

{

return((2*x)+pow(cosh(x),-2)-1.3*cos(x)*exp(1.3*sin(x)));

}
```

***USER SCREEN:-***


Enter bearable error: 0.0001
Enter initial value: 1
Root=2.20675(correct upto five decimal places)
[Program finished]

## (10)GAUSS ELIMINATION METHOD

- ***Introduction and Working Formula***



GAUSS ELIMINATION METHOD :⇒

This is an elimination method which reduces the system of equations to an upper triangular system which can be solved by back substitution. Let us consider a system of n linear equations with n unknowns $AX = b$ where $\det A \neq 0$.

i.e $(a_{ij})_{\substack{1\le i\le n\\ 1\le j\le n}} (x_i)_{1\le i\le n} = (b_i)_{1\le i\le n}$

Denote $a_{ij} = a_{ij}^{(1)}$ & $b_i = b_i^{(1)}$, $i,j = 1(1)n$. Then the system looks like

$$(a_{ij}^{(1)})(x_i) = (b_i^{(1)}) \quad \text{--(i)}$$

Let, $a_{11}^{(1)} \neq 0$, multiplying first row of (i) by $m_{i1} = -a_{i1}^{(1)}/a_{11}^{(1)}$ & adding with the ith row (i.e $R_i = R_i + m_{i1}R_1$) $(i = 1(1)n)$ we get,

$$\begin{pmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & a_{1n}^{(1)} \\ 0 & a_{22}^{(2)} & \cdots & a_{2n}^{(2)} \\ \vdots & \vdots & \vdots & \vdots \\ 0 & a_{n2}^{(2)} & \cdots & a_{nn}^{(2)} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1^{(1)} \\ b_2^{(2)} \\ \vdots \\ b_n^{(2)} \end{pmatrix} \quad \text{--(ii)}$$

where, $a_{ij}^{(2)} = a_{ij}^{(1)} + m_{i1}a_{ij}^{(1)}$, $b_i^{(2)} = b_i^{(1)} + m_{i1}b_1^{(1)}$, $i,j = 2(1)n$.

Let, $a_{22}^{(2)} \neq 0$. multiplying the 2nd row of (ii) by $m_{i2} = -\frac{a_{i2}^{(2)}}{a_{22}^{(2)}}$ and adding it with the i-th row of (ii), $i = 3(1)n$, we get,

$$\begin{pmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} & \cdots & a_{1n}^{(1)} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & \cdots & a_{2n}^{(2)} \\ 0 & 0 & a_{33}^{(3)} & \cdots & a_{3n}^{(3)} \\ \vdots & 0 & a_{n3}^{(3)} & \cdots & a_{nn}^{(3)} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1^{(1)} \\ b_2^{(2)} \\ b_3^{(3)} \\ \vdots \\ b_n^{(3)} \end{pmatrix} \quad \text{--(iii)}$$

where, $a_{ij}^{(3)} = a_{ij}^{(2)} + m_{i2}a_{2j}^{(2)}$, $b_i^{(3)} = b_i^{(2)} + m_{i2}b_2^{(2)}$ $(i,j = 3(1)n)$.

Proceeding in this way, we obtain the following system of equation at the $(n-1)$th step

$$\begin{pmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} & \cdots & a_{1(n-1)}^{(1)} & a_{1n}^{(1)} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & \cdots & a_{2(n-1)}^{(2)} & a_{2n}^{(2)} \\ 0 & 0 & a_{33}^{(3)} & \cdots & a_{3(n-1)}^{(3)} & a_{3n}^{(3)} \\ \vdots & 0 & 0 & \cdots & a_{(n-1)(n-1)}^{(n-1)} & a_{(n-1)n}^{(n-1)} \\ 0 & 0 & 0 & \cdots & 0 & a_{nn}^{(n)} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} b_1^{(1)} \\ b_2^{(2)} \\ b_3^{(3)} \\ \vdots \\ b_{n-1}^{(n-1)} \\ b_n^{(n)} \end{pmatrix} \quad \text{--(iv)}$$

where $a_{ij}^{(k+1)} = a_{ij}^{(k)} + m_{ik}\, a_{kj}^{(k)}$

$b_i^{(k+1)} = b_i^{(k)} + m_{ik}\, b_k^{(k)}$

$m_{ik} = -\dfrac{a_{ik}^{(k)}}{a_{kk}^{(k)}}$, $i, j = k+1, k+2, \cdots, n$. $k = 1, 2, \cdots, n-1$.

The non-zero coefficient (by assumption) $a_{11}^{(1)}, a_{22}^{(2)}, \cdots, a_{nn}^{(n)}$ of the above set of equations are known as pivots & the corresponding equations are pivotal equations.

From the last equation of the above system, we have

$$x_n = \frac{b_n^{(n)}}{a_{nn}^{(n)}}.$$

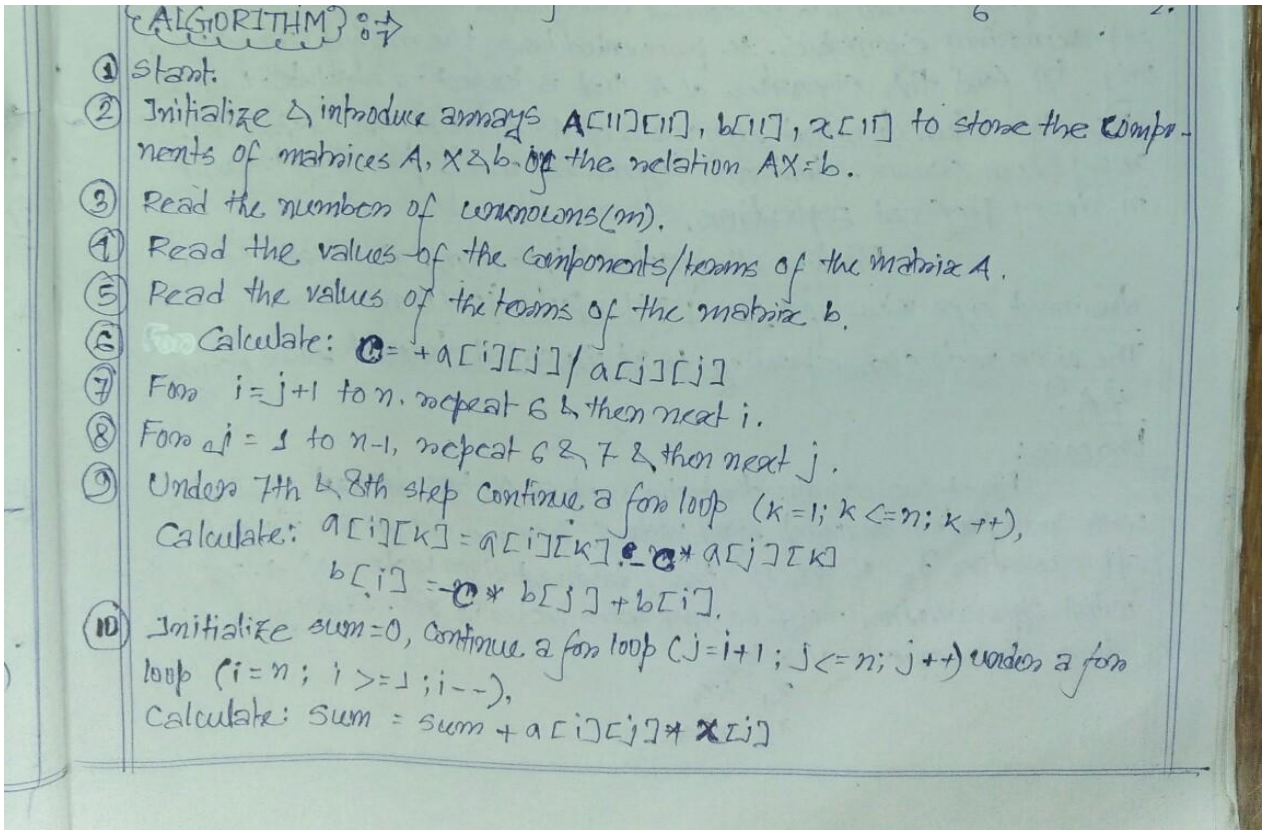
This is then substituted in the $(n-1)$-th equation to obtain $x_{n-1}$ & the process is repeated to compute the other unknowns.

Thus, $x_i = \dfrac{1}{a_{ii}^{(i)}}\left(b_i^{(i)} - \sum_{j=i+1}^{n} a_{ij}^{(i)} x_j\right)$ $j = n-1, n-2, \cdots$.

NOTE : The total number of addition & subtractions $= \dfrac{n}{6}(n-1)(2n+5)$.

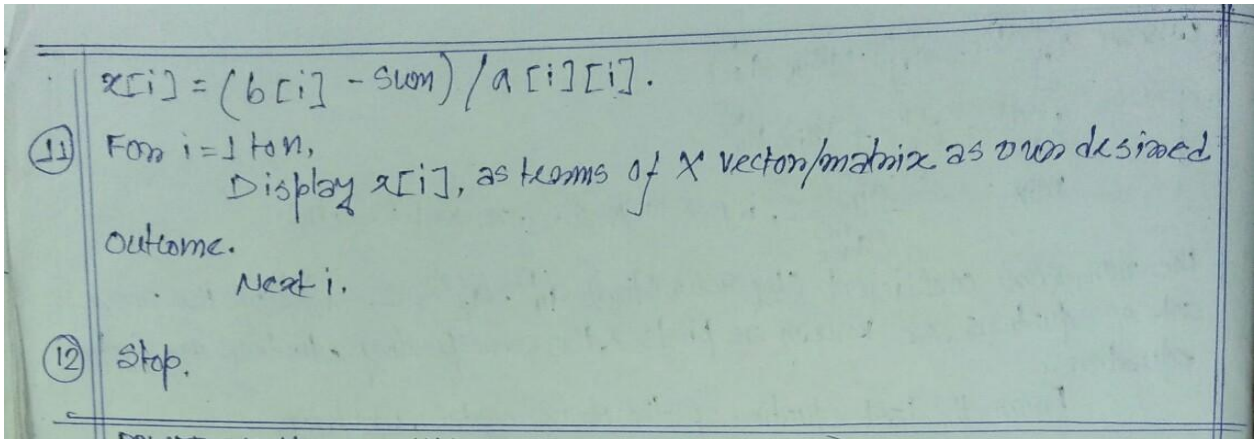
(ALGORITHM) :→

## Algorithm:-



ALGORITHM :→

1. Start.
2. Initialize & introduce arrays A[1][1], b[1], x[1] to store the components of matrices A, X & b of the relation AX=b.
3. Read the number of rows/columns(n).
4. Read the values of the components/terms of the matrix A.
5. Read the values of the terms of the matrix b.
6. Calculate: c = +a[i][j]/a[j][j]
7. For i=j+1 to n, repeat 6 & then next i.
8. For j = 1 to n-1, repeat 6 & 7 & then next j.
9. Under 7th & 8th step continue a for loop (k=1; k<=n; k++),
   Calculate: a[i][k] = a[i][k] - c * a[j][k]
   b[i] = -c * b[j] + b[i].
10. Initialize sum=0, continue a for loop (j=i+1; j<=n; j++) under a for loop (i=n; i>=1; i--),
    Calculate: sum = sum + a[i][j] * x[j]



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

11. For i=1 to n,
    Display x[i], as terms of 'x' vector/matrix as user desired outcome.
    Next i.
12. Stop.

- ***Question:-***

***Write a C program to solve the following system of linear equation by GAUSS ELIMINATION METHOD.***

***AX=b;***

***A=***

| | | | |
|---|---|---|---|
| ***3.91+a*** | ***1.13*** | ***0.15*** | ***0.85*** |
| ***1.13*** | ***-3.81+a*** | ***-0.39*** | ***0.97*** |
| ***0.15*** | ***-0.39*** | ***-5.01+a*** | ***1.03*** |
| ***0.85*** | ***0.97*** | ***1.03*** | ***4.11+a*** |

***and X=***

| |
|---|
| ***x1*** |
| ***x2*** |
| ***x3*** |
| ***x4*** |

***; b=***

| |
|---|
| ***1.11*** |
| ***-2.12*** |
| ***3.13*** |
| ***1.23*** |

***[a=0.5+(R/100);R being last digit my university roll number]***

- ***Answer:-***

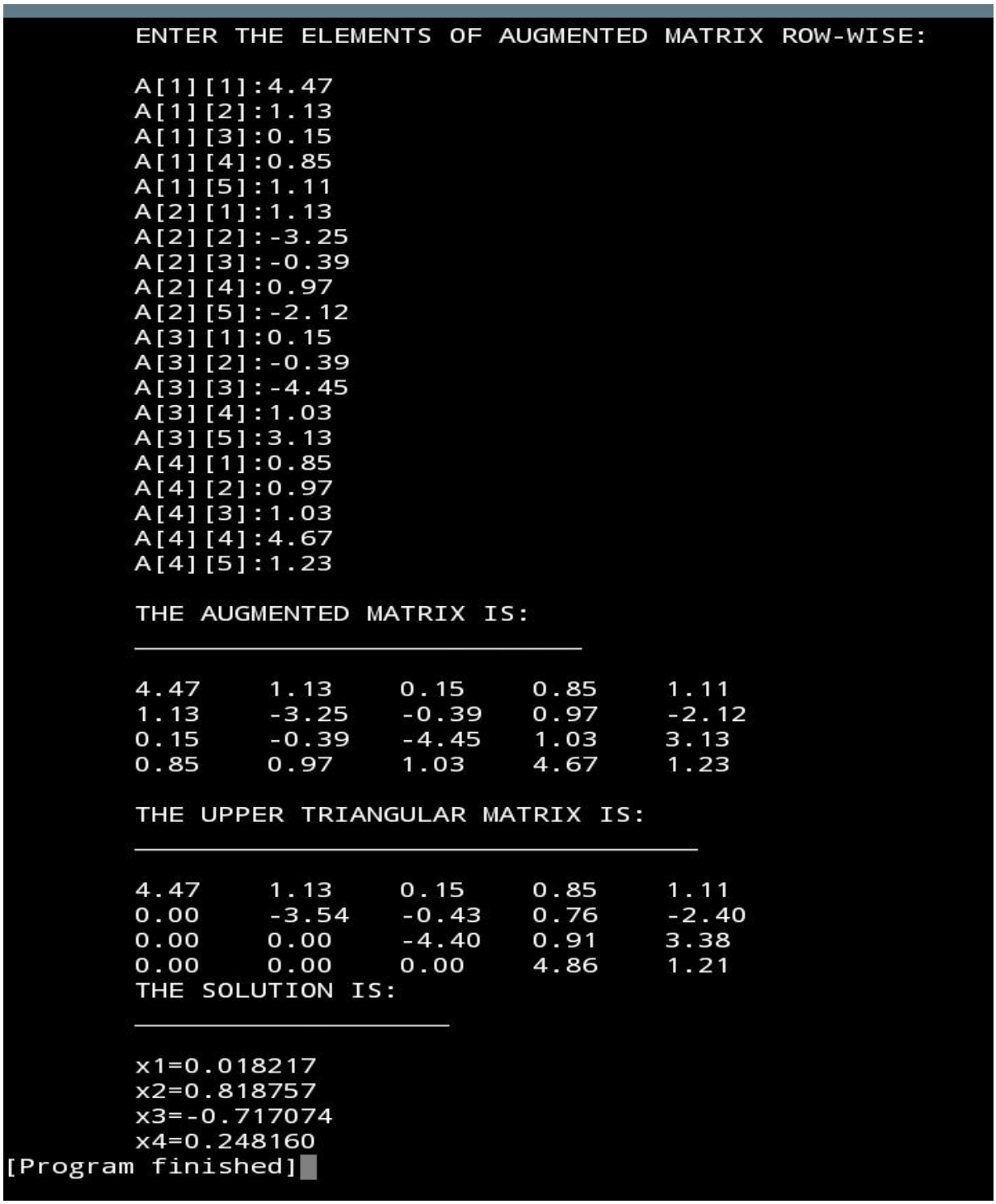
```
//SOLUTION OF EQUATION BY GAUSS
ELIMINATION METHOD//
#include<stdio.h>
#include<conio.h>
#include<math.h>
int main()
{
  int i,j,k,n;
  float A[11][11],c,x[11],sum=0.0;
  printf("\n\tENTER THE NUMBER OF VARIABLES:
");
  scanf("%d",&n);
  printf("\n\tENTER THE ELEMENTS OF
AUGMENTED MATRIX ROW-WISE:\n\n");
  for(i=1;i<=n;i++)
  {
       for(j=1;j<=n+1;j++)
       {
            printf("\tA[%d][%d]:",i,j);
            scanf("\t%f",&A[i][j]);
       }
  }
  printf("\n\tTHE AUGMENTED MATRIX IS:\n");
```

```
printf("\t_____\n");
for(i=1;i<=n;i++)
{
    printf("\n");
    for(j=1;j<=n+1;j++)
    {
        printf("\t%0.2f",A[i][j]);
    }
}
for(j=1;j<=n;j++)
{
    for(i=1;i<=n;i++)
    {
        if(i>j)
        {
            c=A[i][j]/A[j][j];
            for(k=1;k<=n+1;k++)
            {
             A[i][k]=A[i][k]-c*A[j][k];
            }
        }
    }
}
printf("\n\n\tTHE UPPER TRIANGULAR MATRIX IS:
\n");
```

```
printf("\t_____\n")
;
  for(i=1;i<=n;i++)
  {
        printf("\n");
        for(j=1;j<=n+1;j++)
        {
              printf("\t%0.2f",A[i][j]);
        }

  }
  x[n]=A[n][n+1]/A[n][n];
  /*THIS LOOP IS FOR BACKWARD
SUBSTITUTION*/
  for(i=n-1;i>=1;i--)
  {
         sum=0;
        for(j=i+1;j<=n;j++)
        {
             sum=sum+A[i][j]*x[j];
        }
        x[i]=(A[i][n+1]-sum)/A[i][i];
  }
  printf("\n\tTHE SOLUTION IS:\n");
```

```
 printf("\t_____\n");
        for(i=1;i<=n;i++)
        {
         printf("\n\tx%d=%f\t",i,x[i]);/*x1,x2,x3 are
the required solutions*/
        }
        return 0;
 }
```

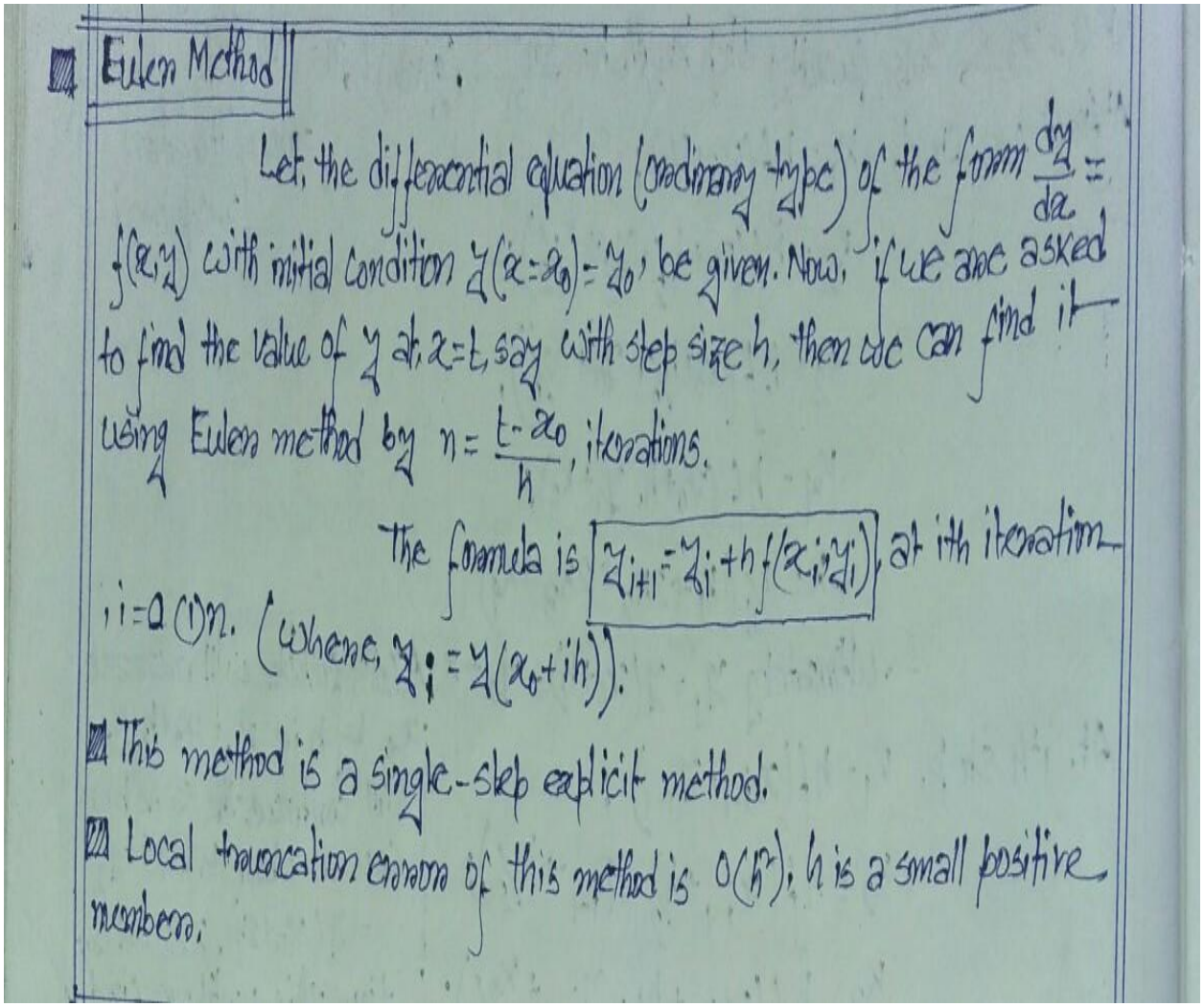
- ***User Interface***



**(11)**

## EULER METHOD

- ***Introduction and Working Formula:-***



Euler Method

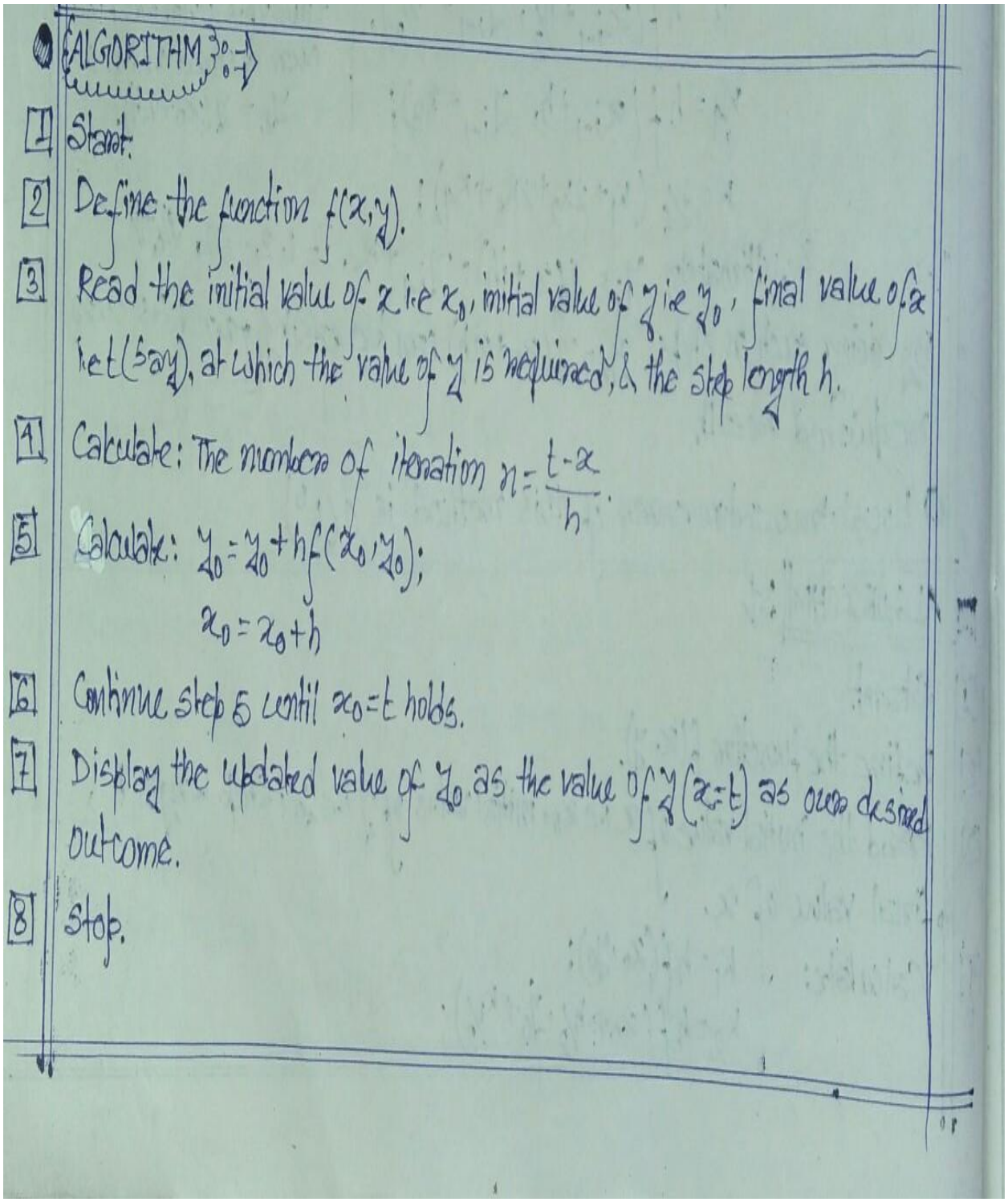
Let the differential equation (ordinary type) of the form $\frac{dy}{dx} = f(x,y)$ with initial condition $y(x=x_0)=y_0$, be given. Now, if we are asked to find the value of $y$ at $x=t$, say with step size $h$, then we can find it using Euler method by $n = \frac{t-x_0}{h}$, iterations.

The formula is $\boxed{y_{i+1} = y_i + h f(x_i, y_i)}$ at ith iteration, $i = 0(1)n$. (where, $y_i = y(x_0 + ih)$).

This method is a single-step explicit method.

Local truncation error of this method is $O(h^2)$. $h$ is a small positive number.

- ***Algorithm:-***



ALGORITHM :-

1. Start.
2. Define the function $f(x,y)$.
3. Read the initial value of $x$ i.e $x_0$, initial value of $y$ i.e $y_0$, final value of $x$ i.e $t$ (say), at which the value of $y$ is required, & the step length $h$.
4. Calculate: The number of iteration $n = \frac{t-x}{h}$.
5. Calculate: $y_0 = y_0 + hf(x_0, y_0)$;
   $x_0 = x_0 + h$
6. Continue step 5 until $x_0 = t$ holds.
7. Display the updated value of $y_0$ as the value of $y(x=t)$ as our desired outcome.
8. Stop.

- ***Question:-***

**Write a C program to solve the following differential equation for x=0(0.1)0.5 by Euler method correct up to five decimal places.**

$$\frac{dy}{dx} = 1 - \sin sin\,(ax + y) + \frac{by}{2+x}, y(0) = 0$$

$$a = 1 + 0.25R,$$

$$b=-0.3+0.2R$$

**R is the last of the University Roll Number.**

***Answer:-***

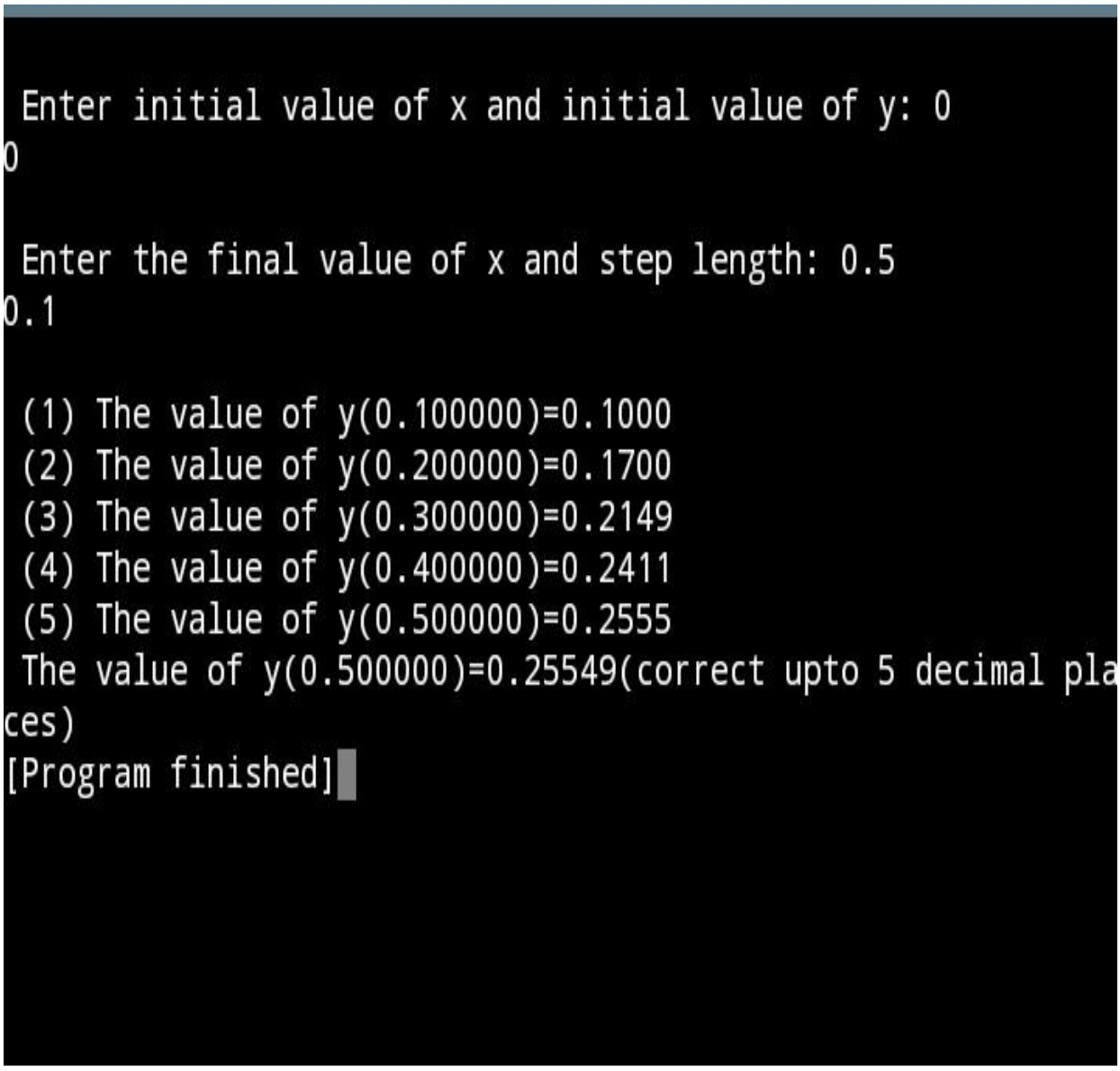
**[The last digit of my university roll number being 6,I will take R=6 and according to the definition of a and b their value will be a=2.5 and b=0.9]**

**INPUT**

```
//EULER METHOD//
#include<stdio.h>
#include<math.h>
float f(float x,float y)
{
    return 1-sin((2.5*x)+y)+((0.9*y)/(2+x));
}
int main()
```

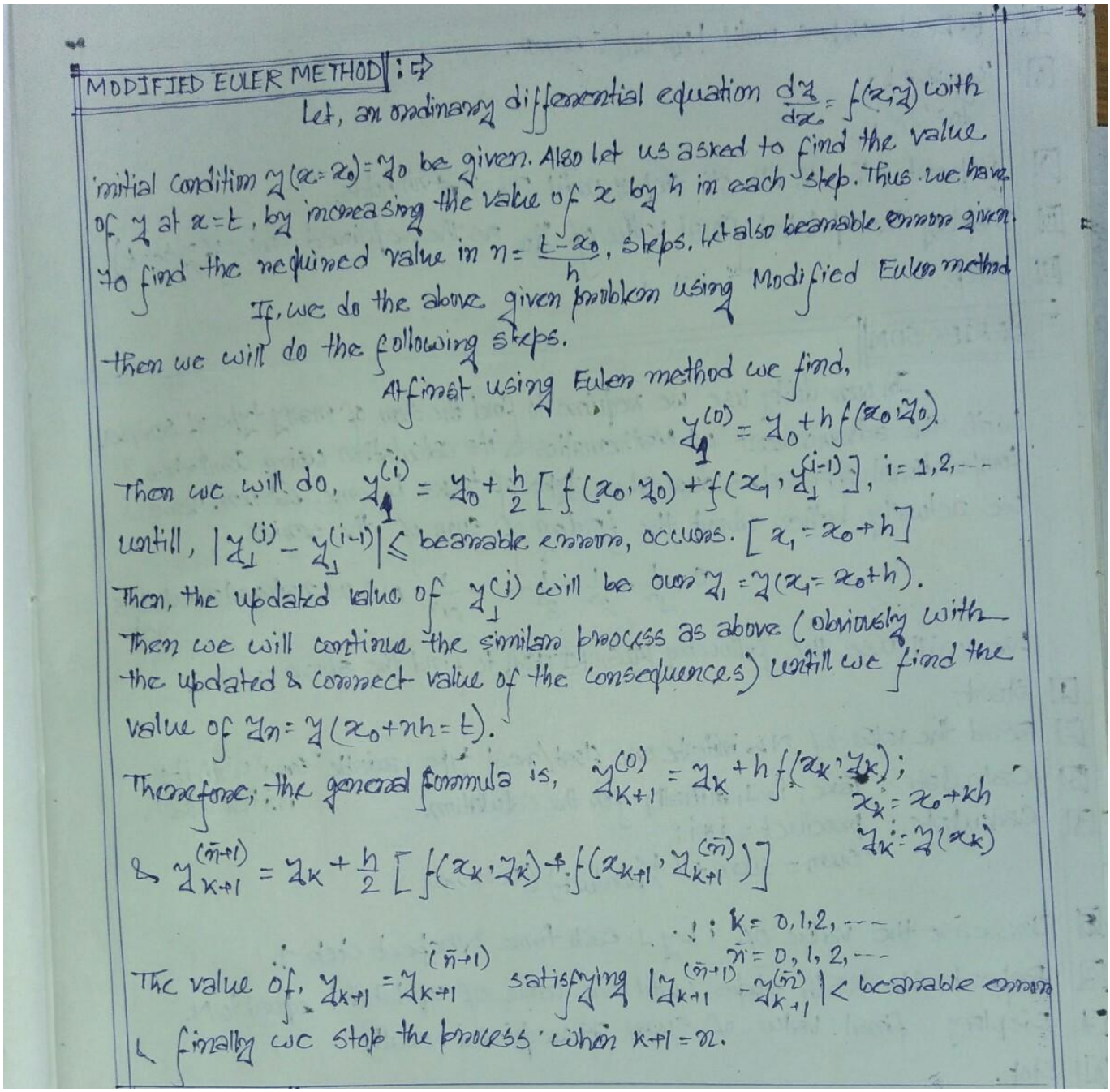
```
{
    float x0,y0,x,h;
    int i=1,n;
    printf("\n Enter initial value of x and initial
value of y: ");
    scanf("%f%f",&x0,&y0);
    printf("\n Enter the final value of x and step
length: ");
    scanf("%f%f",&x,&h);
    n=fabs(x-x0)/h*1.0;
    do
    { for(i=1;i<n+1;i++)
    {
        y0=y0+h*f(x0,y0);
        x0=x0+h;
        printf("\n (%d) The value of
y(%f)=%0.4f",i,x0,y0);
    }
    }while(x0<x);
    printf("\n The value of y(%f)=%0.5f(correct
upto 5 decimal places)",x0,y0);
    return 0;
}
```

- **USER INTERFACE:-**



# (12)MODIFIED EULER METHOD

- **Introduction *and Working Formula* :-**



MODIFIED EULER METHOD : ⇨

Let, an ordinary differential equation $\frac{dy}{dx} = f(x,y)$ with initial condition $y(x=x_0)=y_0$ be given. Also let us asked to find the value of $y$ at $x=t$, by increasing the value of $x$ by $h$ in each step. Thus we have to find the required value in $n = \frac{t-x_0}{h}$, steps. Let also bearable error given.

If, we do the above given problem using Modified Euler method then we will do the following steps.

At first using Euler method we find,

$$y_1^{(0)} = y_0 + h f(x_0, y_0)$$

Then we will do, $y_1^{(i)} = y_0 + \frac{h}{2}\left[f(x_0,y_0) + f(x_1, y_1^{(i-1)})\right]$, $i = 1,2,\cdots$

until, $|y_1^{(i)} - y_1^{(i-1)}| \leq$ bearable error, occurs. $[x_1 = x_0 + h]$

Then, the updated value of $y_1^{(i)}$ will be over $y_1 = y(x_1 = x_0+h)$.

Then we will continue the similar process as above (obviously with the updated & correct value of the consequences) until we find the value of $y_n = y(x_0 + nh = t)$.

Therefore, the general formula is, $y_{k+1}^{(0)} = y_k + h f(x_k, y_k)$;

$x_k = x_0 + kh$

$y_k = y(x_k)$

& $y_{k+1}^{(\bar{n}+1)} = y_k + \frac{h}{2}\left[f(x_k,y_k) + f(x_{k+1}, y_{k+1}^{(\bar{n})})\right]$

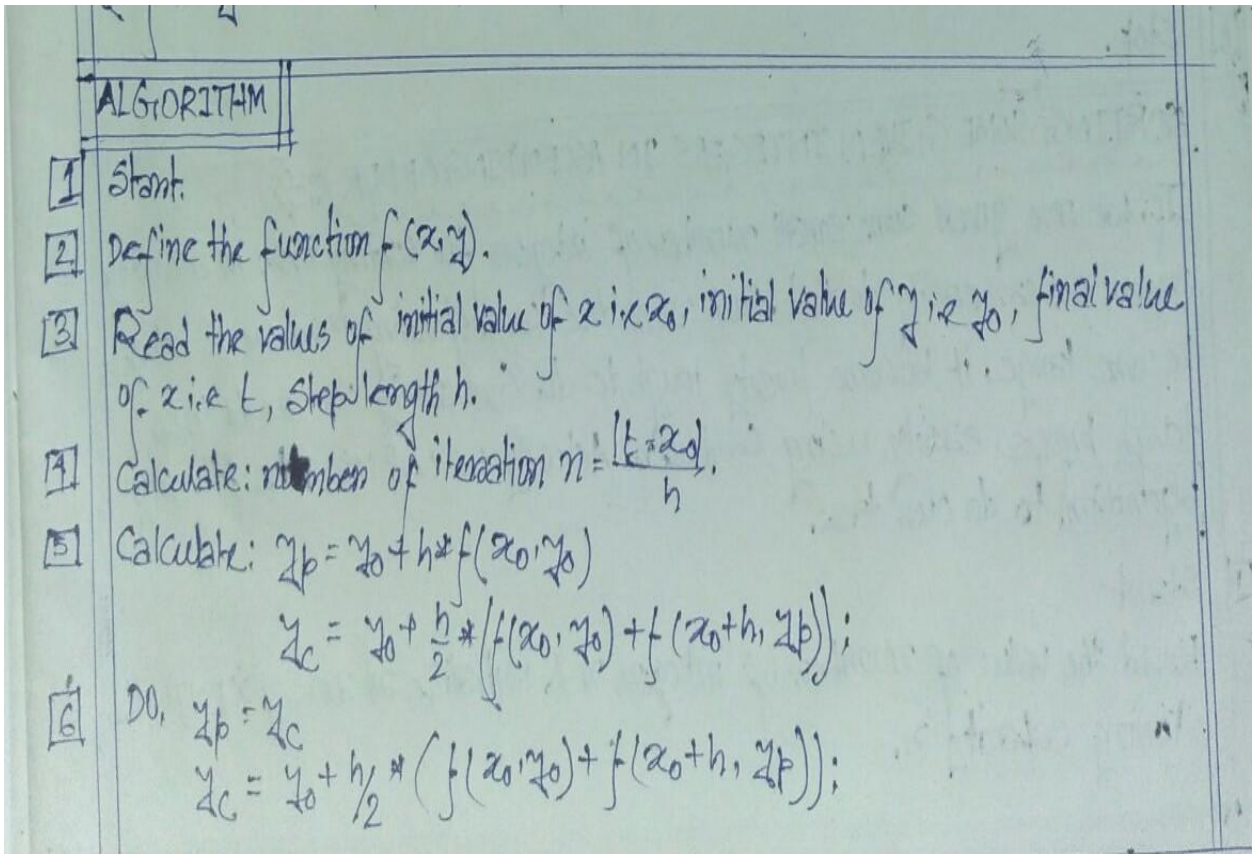
$\therefore$ $k = 0,1,2,\cdots$

$\bar{n} = 0,1,2,\cdots$

The value of $y_{k+1} = y_{k+1}^{(\bar{n}+1)}$ satisfying $|y_{k+1}^{(\bar{n}+1)} - y_{k+1}^{(\bar{n})}| <$ bearable error

& finally we stop the process when $k+1 = n$.

- ***Algorithm:-***

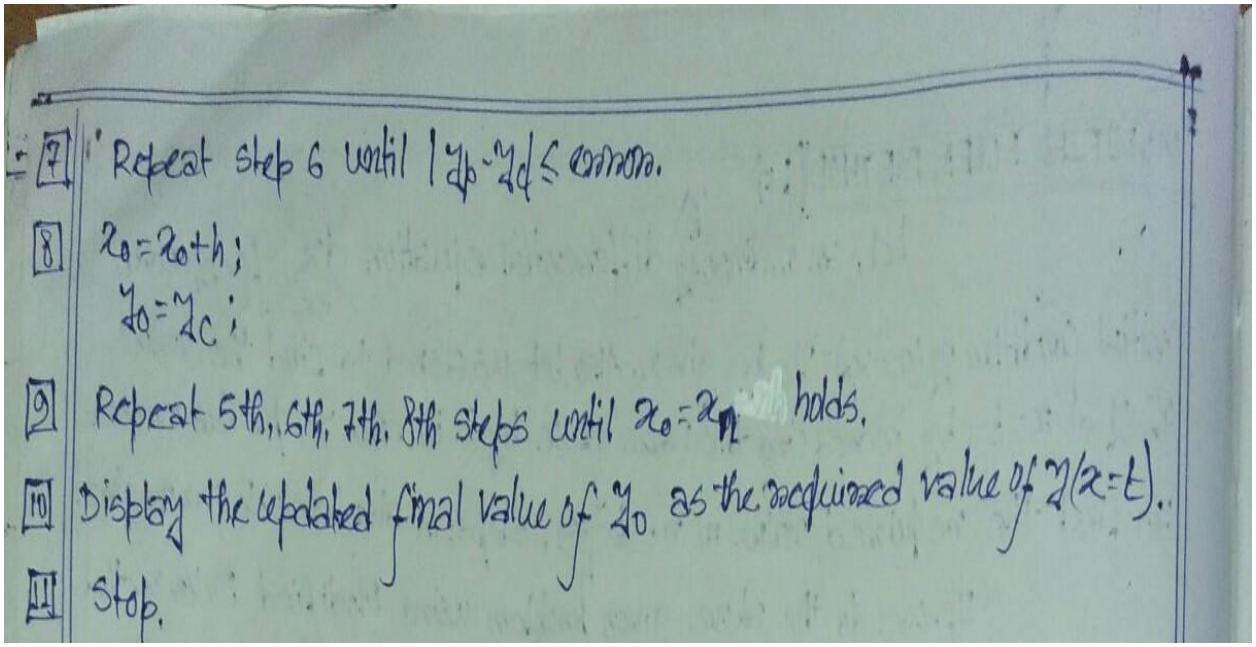


ALGORITHM

1. Start.
2. Define the function $f(x,y)$.
3. Read the values of initial value of $x$ i.e $x_0$, initial value of $y$ i.e $y_0$, final value of $x$ i.e $t$, Step length $h$.
4. Calculate: number of iteration $n = \frac{|t-x_0|}{h}$.
5. Calculate: $y_p = y_0 + h * f(x_0, y_0)$

   $y_c = y_0 + \frac{h}{2} * (f(x_0, y_0) + f(x_0+h, y_p))$;
6. DO, $y_p = y_c$

   $y_c = y_0 + h/2 * (f(x_0, y_0) + f(x_0+h, y_p))$;



7. Repeat step 6 until $|y_p - y_c| \leq$ error.
8. $x_0 = x_0 + h$;

   $y_0 = y_c$;
9. Repeat 5th, 6th, 7th, 8th steps until $x_0 = x_n$ holds.
10. Display the updated final value of $y_0$ as the required value of $y(x=t)$.
11. Stop.

- ***Question:-***

**Write a C program to solve the following differential equation for x=0(0.1)0.5 by modified Euler method correct up to six decimal places.**

$$\frac{dy}{dx} = \frac{1+1.1x^2+0.3y^2}{3+0.9x^2y^3}, y(0) = 1.1 + \frac{R}{100}$$

**R is the last of the University Roll Number.**

Sol:

**Input:**

```
//Modified Euler Method//
#include<stdio.h>
#include<math.h>
float f(float x,float y);
int main()
{
float x0,xn,y0,h,yc,yp,R,error=1.e-8;
int n;
x0=0;
xn=0.5;
h=0.1;
n=(xn-x0)/h;
printf("enter the value of R\n");
```

```
scanf("%f",&R);
y0=1.1+(R/100);
printf("\n The number of steps will be %d",n);
while(x0<xn)
{
yp=y0+h*f(x0,y0);
yc=y0+(h/2)*(f(x0,y0)+f(x0+h,yp));
while(fabs(yp-yc)>error)
{
yp=yc;
yc=y0+(h/2)*(f(x0,y0)+f(x0+h,yp));
}
x0=x0+h;
y0=yc;
printf("y(%3.1f)=%7.6f\n",x0,y0);
}
printf("(correct upto six decimal places)");
}
float f(float x,float y)
{
float z;
z=(1.0+1.1*pow(x,2.0)+0.3*pow(y,2.0))/(3.0+0.9*pow(x,2.0)*pow(y,3.0));
return(z);}
```

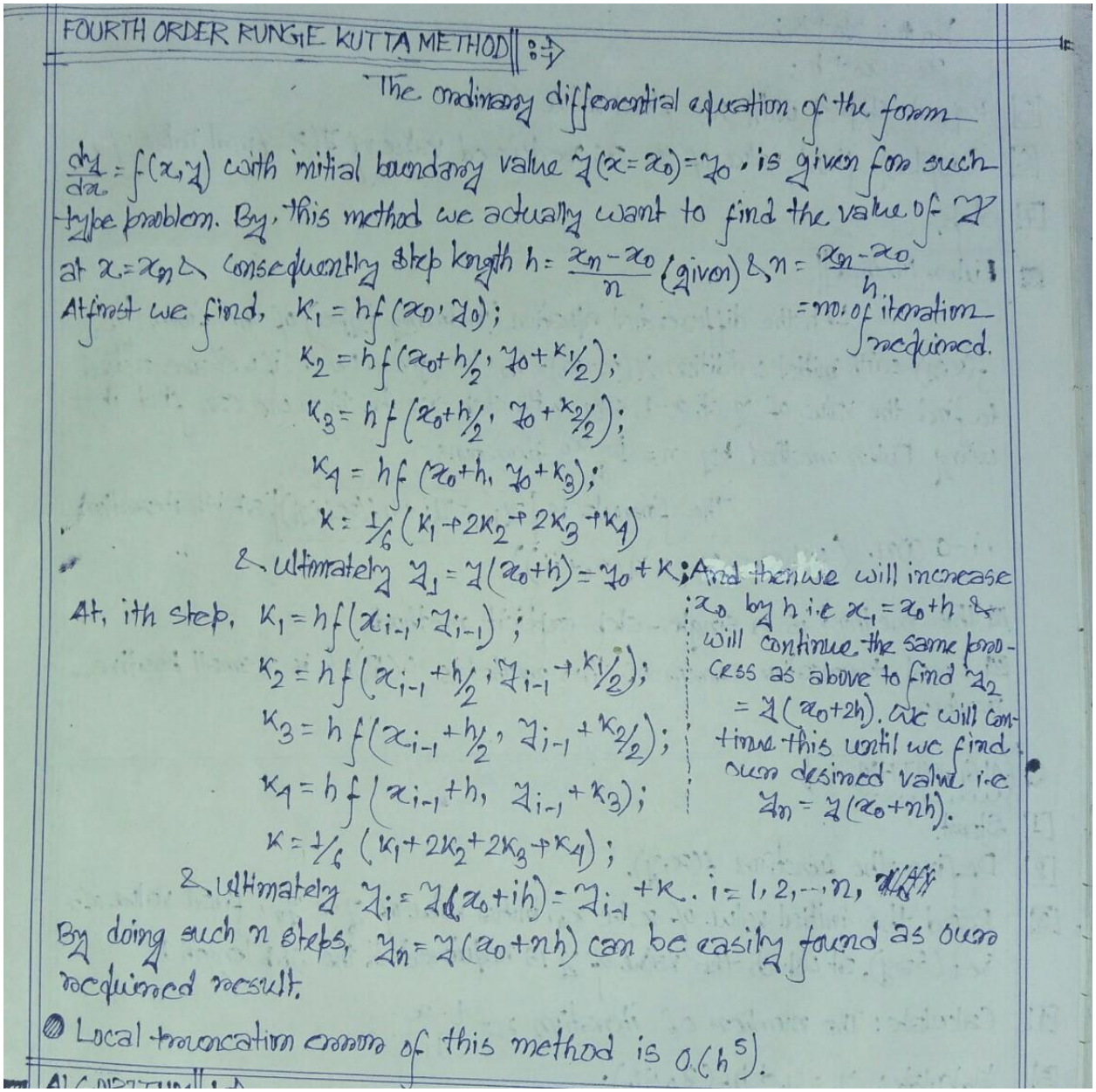
## USER INTERFACE

```
enter the value of R
6

 The number of step will be 5
   y(0.1)=1.207407

   y(0.2)=1.256121

   y(0.3)=1.306146

   y(0.4)=1.357192

   y(0.5)=1.408685
(correct upto six decimal places)
[Program finished]
```

**(13)**

## FOURTH ORDER RUNGE KUTTA METHOD

- ***Introduction and Working Formula:***



FOURTH ORDER RUNGE KUTTA METHOD :→

The ordinary differential equation of the form $\frac{dy}{dx} = f(x,y)$ with initial boundary value $y(x=x_0)=y_0$, is given for such type problem. By this method we actually want to find the value of $y$ at $x=x_n$. Consequently step length $h = \frac{x_n - x_0}{n}$ (given) & $n = \frac{x_n - x_0}{h}$ = no. of iteration required.

At first we find, $k_1 = hf(x_0, y_0)$;

$k_2 = hf(x_0 + h/2, y_0 + k_1/2)$;

$k_3 = hf(x_0 + h/2, y_0 + k_2/2)$;

$k_4 = hf(x_0 + h, y_0 + k_3)$;

$k = \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$

& ultimately $y_1 = y(x_0 + h) = y_0 + k$; And then we will increase $x_0$ by $h$ i.e $x_1 = x_0 + h$ & will continue the same process as above to find $y_2 = y(x_0 + 2h)$. We will continue this until we find our desired value i.e $y_n = y(x_0 + nh)$.

At, ith step, $k_1 = hf(x_{i-1}, y_{i-1})$;

$k_2 = hf(x_{i-1} + h/2, y_{i-1} + k_1/2)$;

$k_3 = hf(x_{i-1} + h/2, y_{i-1} + k_2/2)$;

$k_4 = hf(x_{i-1} + h, y_{i-1} + k_3)$;

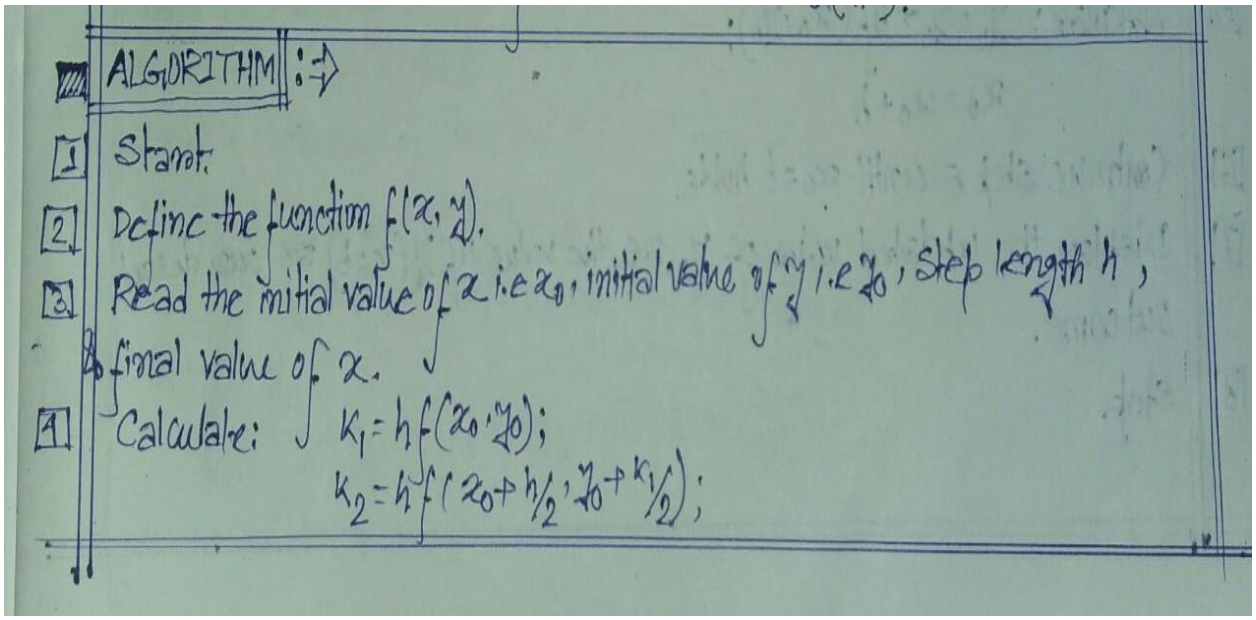
$k = \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$;

& ultimately $y_i = y(x_0 + ih) = y_{i-1} + k$. $i = 1, 2, \dots, n$,

By doing such $n$ steps, $y_n = y(x_0 + nh)$ can be easily found as our required result.

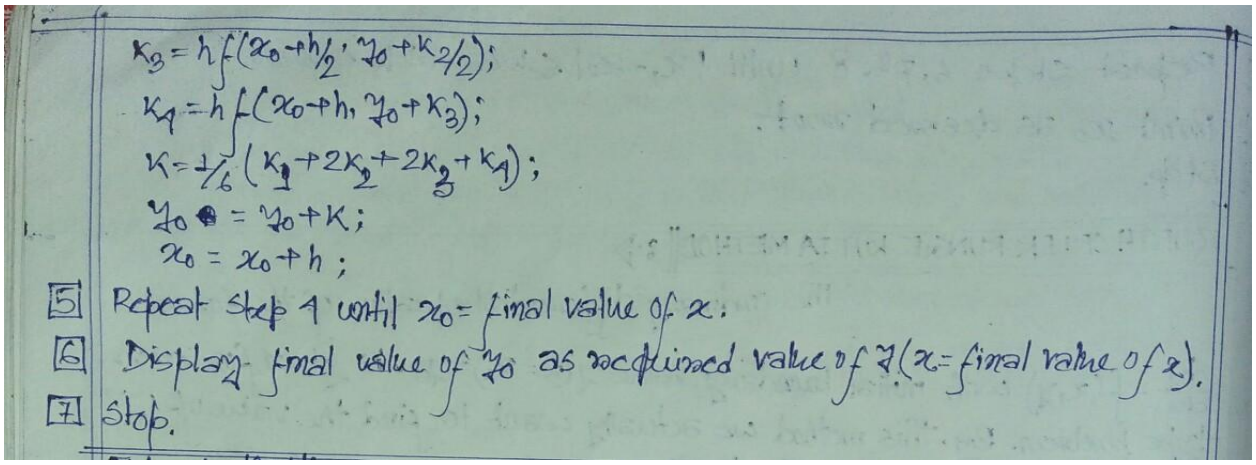
● Local truncation error of this method is $O(h^5)$.

## Algorithm:-



ALGORITHM :⇒

1. Start.
2. Define the function $f(x, y)$.
3. Read the initial value of $x$ i.e $x_0$, initial value of $y$ i.e $y_0$, step length $h$, final value of $x$.
4. Calculate: $k_1 = hf(x_0, y_0)$;
   $k_2 = hf(x_0 + h/2, y_0 + k_1/2)$;



   $k_3 = hf(x_0 + h/2, y_0 + k_2/2)$;
   $k_4 = hf(x_0 + h, y_0 + k_3)$;
   $k = 1/6(k_1 + 2k_2 + 2k_3 + k_4)$;
   $y_0 = y_0 + k$;
   $x_0 = x_0 + h$;
5. Repeat step 4 until $x_0$ = final value of $x$.
6. Display final value of $y_0$ as required value of $y$ ($x$ = final value of $x$).
7. Stop.

- ***Question:-***

**Write a C program to solve the following differential equation equation for x=0(0.1)1.0 by fourth order Runge Kutta method correct up to six decimal places.**

$$\frac{dy}{dx} = \frac{\sqrt[3]{1+x^3y^3}}{\sqrt[2]{1+x^2y^2}}, y(0) = \frac{R+1}{20}$$

**R is the last of the University Roll Number.**

Sol:

**Intput:**

```
//Fourth Order Range Kutta Method//
#include<stdio.h>
#include<math.h>
float f(float x,float y);
int main()
{
float x0,xn,y0,h,k,k1,k2,k3,k4,R;
printf("\n Enter Initial value of x: ");
scanf("%f",&x0);
printf("\n Enter final value of x and step length: ");
scanf("%f%f",&xn, &h);
printf("\nEnter the value of R:");
```

```
scanf("%f",&R);
y0=(R+1)/20;
while(x0<xn)
{
k1=h*f(x0,y0);
k2=h*f(x0+h/2.,y0+k1/2.);
k3=h*f(x0+h/2.,y0+k2/2.);
k4=h*f(x0+h,y0+k3);
k=(1./6.)*(k1+2.*k2+2.*k3+k4);
y0=y0+k;
x0=x0+h;
printf("y(%3.1f)=%8.6f\n",x0,y0);
}
printf("\n The value of y(%f) =%0.6f(Correct up to six decimal places)",xn, y0);
}
float f(float x,float y)
{
float z;
z=cbrt(1+pow(x,3.0)*pow(y,3.0))/sqrt(1+pow(x,2.0)*pow(y,2.0));
return(z);
}
```

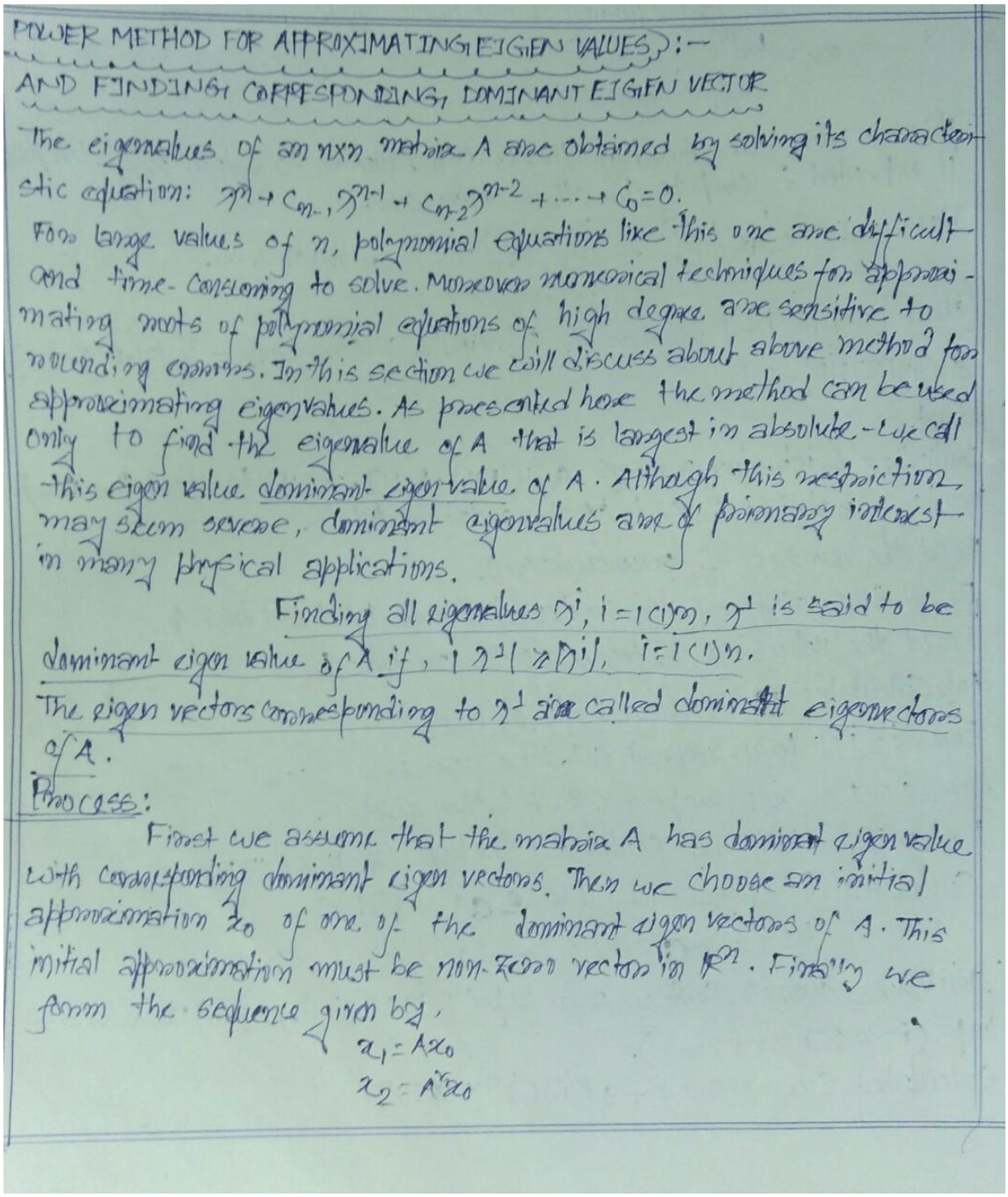
- ***User Interface:-***

```
 Enter initial value of x: 0

 Enter final value of x and step length: 1
 0.1

Enter the value of R:6
y(0.1)=0.449970
y(0.2)=0.549684
y(0.3)=0.648663
y(0.4)=0.746250
y(0.5)=0.841813
y(0.6)=0.935033
y(0.7)=1.026117
y(0.8)=1.115763
y(0.9)=1.204890
y(1.0)=1.294323

 The value of y(1.000000)=1.294323(Correct up to six decimal
 places)
[Program finished]
```

**(14)**

## POWER METHOD

- **Introduction and Working Formula**



POWER METHOD FOR APPROXIMATING EIGEN VALUES :-
AND FINDING CORRESPONDING DOMINANT EIGEN VECTOR

The eigenvalues of an nxn matrix A are obtained by solving its characteristic equation: $\lambda^n + c_{n-1}\lambda^{n-1} + c_{n-2}\lambda^{n-2} + \cdots + c_0 = 0$.
For large values of n, polynomial equations like this one are difficult and time-consuming to solve. Moreover numerical techniques for approximating roots of polynomial equations of high degree are sensitive to rounding errors. In this section we will discuss about above method for approximating eigenvalues. As presented here the method can be used only to find the eigenvalue of A that is largest in absolute - we call this eigen value dominant eigenvalue of A. Although this restriction may seem severe, dominant eigenvalues are of primary interest in many physical applications.

Finding all eigenvalues $\lambda_i$, $i = 1(1)n$, $\lambda^1$ is said to be dominant eigen value of A if, $|\lambda^1| > |\lambda_i|$, $i = 1(1)n$.

The eigen vectors corresponding to $\lambda^1$ are called dominant eigenvectors of A.

Process:

First we assume that the matrix A has dominant eigen value with corresponding dominant eigen vectors. Then we choose an initial approximation $x_0$ of one of the dominant eigen vectors of A. This initial approximation must be non-zero vector in $R^n$. Finally we form the sequence given by,

$x_1 = Ax_0$

$x_2 = Ax_0$

$x_3 = A^3 x_0$

$\vdots$

$x_k = A^k x_0$

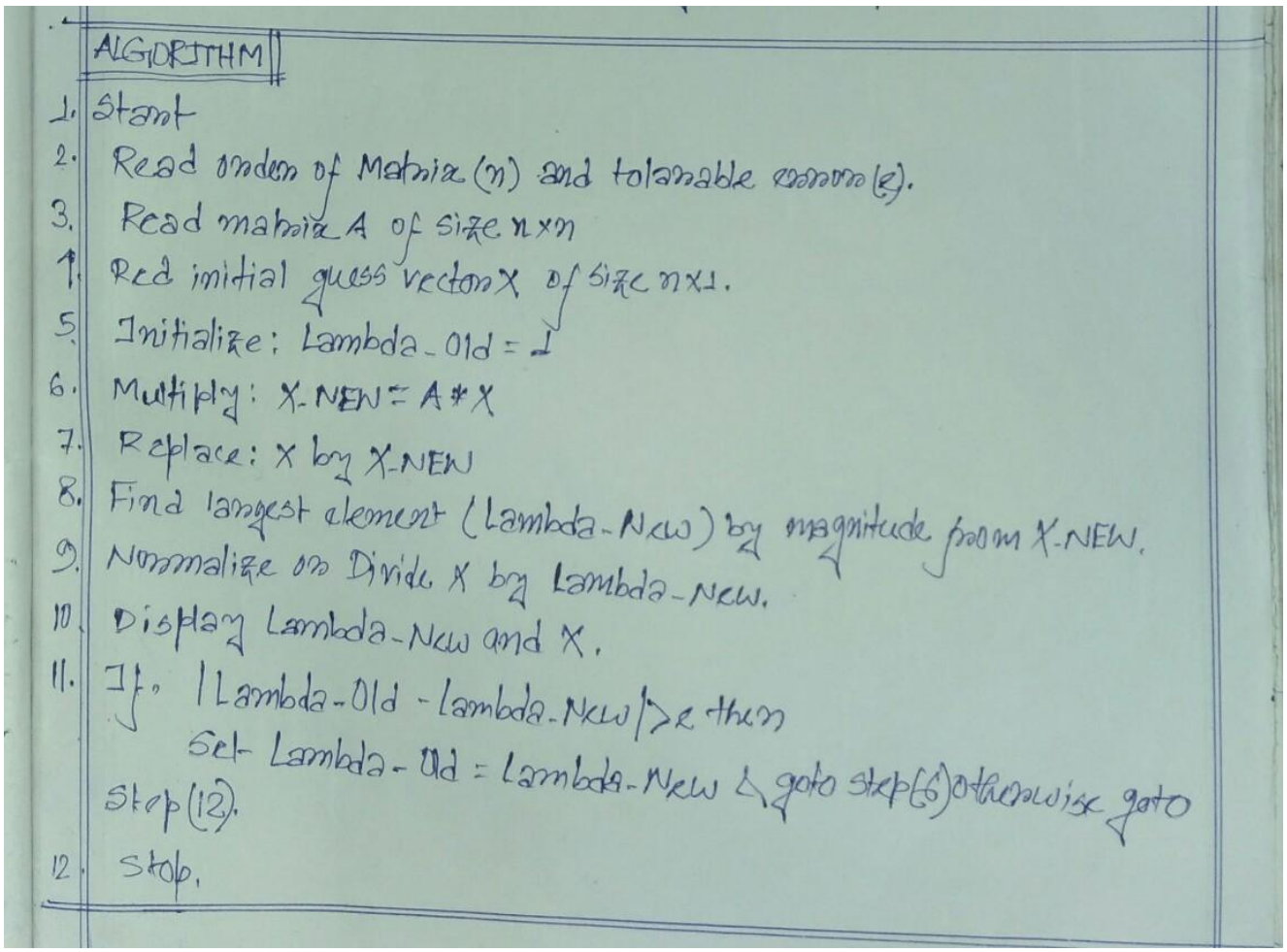
For large powers of $k$, and by properly scaling this sequence, we will see that we obtain a good approximation of dominant eigen vectors of $A$. This procedure is iterative.

Corresponding dominant eigen value $\lambda_1 = \lim_{n \to \infty} \frac{(A^{n+1}x)_k}{(A^n x)_k}$.

Where, $(A^n x)_k = (\lambda_1^n)^n \left( C_1 x_{1k} + C_2 \left(\frac{\lambda_2}{\lambda_1}\right)^n x_{2k} + \cdots + C_n \left(\frac{\lambda_n}{\lambda_1}\right)^n x_{nk} \right)$.

$x_i = (x_{i1} \; x_{i2} \cdots x_{in})^t$ (the eigen vector representation).

- **Algorithm**:-



ALGORITHM

1. Start
2. Read order of Matrix (n) and tolerable error (e).
3. Read matrix A of size n x n
4. Red initial guess vector X of size n x 1.
5. Initialize: Lambda_Old = 1
6. Multiply: X_NEW = A * X
7. Replace: X by X_NEW
8. Find largest element (Lambda_New) by magnitude from X_NEW.
9. Normalize or Divide X by Lambda_New.
10. Display Lambda_New and X.
11. If, |Lambda_Old - lambda_New| > e then
    Set Lambda_Old = Lambda_New & goto step(6) otherwise goto Step(12).
12. Stop.

- **Question:-**

***Write a C program to find dominant eigen pair of the following matrix using Power Method.***

| | | | |
|---|---|---|---|
| *9.12+p* | *-1.21* | *1.65* | *-3.11* |
| *-1.21* | *15.23+p* | *-3.14* | *2.11* |
| *1.65* | *-3.14* | *8.55+p* | *-1.18* |
| *-3.11* | *2.11* | *-1.18* | *9.25+p* |

***Where p=0.1*R,R being last digit of university roll number.***

***Answer:-***

**[The last digit of my university roll number being 6,I will use R=6 and consequently the value of p=0.6 and I will use this value for run time input as the value of p]**

***Input:-***

```
//Power Method For Finding Dominant Eigen Pair//
#include<stdio.h>
#include<conio.h>
#include<math.h>

#define SIZE 10

int main()
```

```
{
    float a[SIZE][SIZE], x[SIZE],x_new[SIZE];
    float temp, lambda_new, lambda_old, error;
    int i,j,n, step=1;
    clrscr();
    /* Inputs */
    printf("Enter Order of Matrix: ");
    scanf("%d", &n);
    printf("Enter Tolerable Error: ");
    scanf("%f", &error);
    /* Reading Matrix */
    printf("Enter Coefficient of Matrix:\n");
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=n;j++)
        {
            printf("a[%d][%d]=",i,j);
            scanf("%f", &a[i][j]);
        }
    }
    /* Reading Intial Guess Vector */
    printf("Enter Initial Guess Vector:\n");
    for(i=1;i<=n;i++)
    {
        printf("x[%d]=",i);
```

```
        scanf("%f", &x[i]);
}
/* Initializing Lambda_Old */
lambda_old = 1;
/* Multiplication */
up:
for(i=1;i<=n;i++)
{
        temp = 0.0;
        for(j=1;j<=n;j++)
        {
            temp = temp + a[i][j]*x[j];
        }
        x_new[i] = temp;
}
/* Replacing */
for(i=1;i<=n;i++)
{
     x[i] = x_new[i];
}
/* Finding Largest */
lambda_new = fabs(x[1]);
for(i=2;i<=n;i++)
{
        if(fabs(x[i])>lambda_new)
```

```
            {
                lambda_new = fabs(x[i]);
            }
        }
        /* Normalization */
        for(i=1;i<=n;i++)
        {
            x[i] = x[i]/lambda_new;
        }
        /* Display */
        printf("\n\nSTEP-%d:\n", step);
        printf("Eigen Value = %0.5f(correct upto 5
decimal places)\n", lambda_new);
        printf("Eigen Vector:\n");
        for(i=1;i<=n;i++)
        {
            printf("%0.5f\t", x[i]);
        }
        /* Checking Accuracy */
        if(fabs(lambda_new-lambda_old)>error)
        {
            lambda_old=lambda_new;
            step++;
            goto up;
        }
```

```
        getch();
        return(0);
}
```

- ***USER INTERFACE:-***

```
Enter Order of Matrix: 4
Enter Tolerable Error: 0.0000001
Enter Coefficient of Matrix:
a[1][1]=9.72
a[1][2]=-1.21
a[1][3]=1.65
a[1][4]=-3.11
a[2][1]=-1.21
a[2][2]=15.83
a[2][3]=-3.14
a[2][4]=2.11
a[3][1]=1.65
a[3][2]=-3.14
a[3][3]=9.15
a[3][4]=-1.18
a[4][1]=-3.11
a[4][2]=2.11
a[4][3]=-1.18
a[4][4]=9.85
Enter Initial Guess Vector:
x[1]=0.5
x[2]=0.5
x[3]=0.5
x[4]=0.5


STEP-1:
Eigen Value = 6.79500(correct upto 5 decimal places)
Eigen Vector:
0.51876 1.00000 0.47682 0.56439

STEP-2:
Eigen Value = 14.89593(correct upto 5 decimal places)
Eigen Vector:
0.19226 1.00000 0.09485 0.36877

STEP-3:
Eigen Value = 16.07764(correct upto 5 decimal places)
Eigen Vector:
-0.02063         1.00000 -0.14866         0.31302

STEP-4:
Eigen Value = 16.98220(correct upto 5 decimal places)
Eigen Vector:
-0.15482         1.00000 -0.28875         0.31991
```

```
STEP-5:
Eigen Value = 17.59901(correct upto 5 decimal places)
Eigen Vector:
-0.23787         1.00000 -0.36451         0.34566

STEP-6:
Eigen Value = 17.99173(correct upto 5 decimal places)
Eigen Vector:
-0.28894         1.00000 -0.40439         0.37154

STEP-7:
Eigen Value = 18.23335(correct upto 5 decimal places)
Eigen Vector:
-0.32036         1.00000 -0.42534         0.39189

STEP-8:
Eigen Value = 18.38008(correct upto 5 decimal places)
Eigen Vector:
-0.33974         1.00000 -0.43650         0.40633

STEP-9:
Eigen Value = 18.46904(correct upto 5 decimal places)
Eigen Vector:
-0.35173         1.00000 -0.44258         0.41605

STEP-10:
Eigen Value = 18.52315(correct upto 5 decimal places)
Eigen Vector:
-0.35917         1.00000 -0.44598         0.42240

STEP-11:
Eigen Value = 18.55623(correct upto 5 decimal places)
Eigen Vector:
-0.36380         1.00000 -0.44792         0.42648

STEP-12:
Eigen Value = 18.57655(correct upto 5 decimal places)
Eigen Vector:
-0.36667         1.00000 -0.44906         0.42908
```

```
STEP-13:
Eigen Value = 18.58908(correct upto 5 decimal places)
Eigen Vector:
-0.36847         1.00000 -0.44974         0.43072

STEP-14:
Eigen Value = 18.59684(correct upto 5 decimal places)
Eigen Vector:
-0.36958         1.00000 -0.45015         0.43175

STEP-15:
Eigen Value = 18.60166(correct upto 5 decimal places)
Eigen Vector:
-0.37028         1.00000 -0.45040         0.43240

STEP-16:
Eigen Value = 18.60465(correct upto 5 decimal places)
Eigen Vector:
-0.37072         1.00000 -0.45055         0.43280

STEP-17:
Eigen Value = 18.60651(correct upto 5 decimal places)
Eigen Vector:
-0.37099         1.00000 -0.45064         0.43306

STEP-18:
Eigen Value = 18.60767(correct upto 5 decimal places)
Eigen Vector:
-0.37116         1.00000 -0.45070         0.43322

STEP-19:
Eigen Value = 18.60840(correct upto 5 decimal places)
Eigen Vector:
-0.37126         1.00000 -0.45074         0.43332

STEP-20:
Eigen Value = 18.60885(correct upto 5 decimal places)
Eigen Vector:
-0.37133         1.00000 -0.45076         0.43338

STEP-21:
Eigen Value = 18.60913(correct upto 5 decimal places)
Eigen Vector:
-0.37137         1.00000 -0.45078         0.43342
```

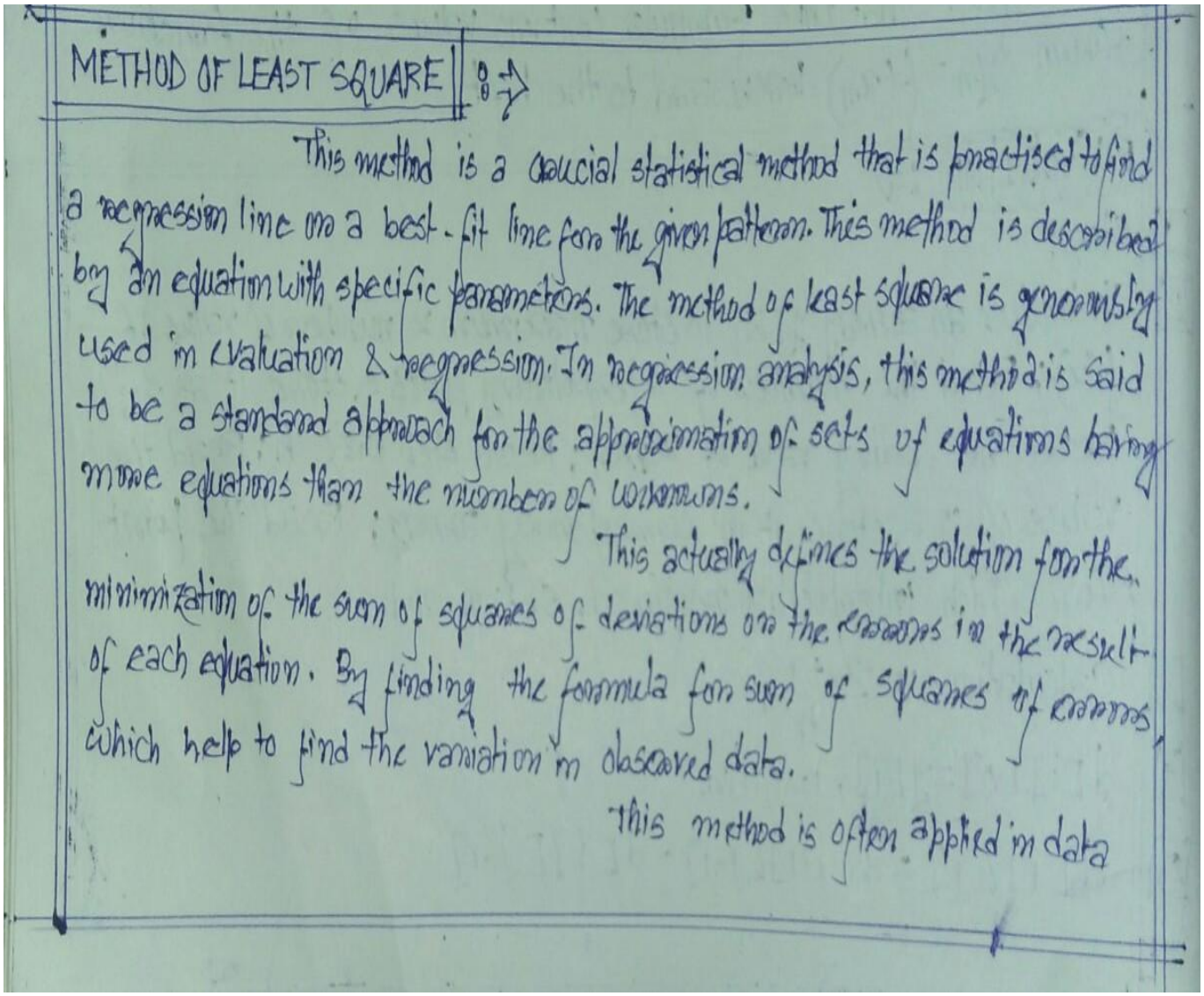
```
STEP-22:
Eigen Value = 18.60930(correct upto 5 decimal places)
Eigen Vector:
-0.37140         1.00000 -0.45079         0.43344

STEP-23:
Eigen Value = 18.60942(correct upto 5 decimal places)
Eigen Vector:
-0.37141         1.00000 -0.45079         0.43346

STEP-24:
Eigen Value = 18.60948(correct upto 5 decimal places)
Eigen Vector:
-0.37142         1.00000 -0.45079         0.43347

STEP-25:
Eigen Value = 18.60953(correct upto 5 decimal places)
Eigen Vector:
-0.37143         1.00000 -0.45080         0.43347

STEP-26:
Eigen Value = 18.60955(correct upto 5 decimal places)
Eigen Vector:
-0.37143         1.00000 -0.45080         0.43347

STEP-27:
Eigen Value = 18.60957(correct upto 5 decimal places)
Eigen Vector:
-0.37144         1.00000 -0.45080         0.43348

STEP-28:
Eigen Value = 18.60958(correct upto 5 decimal places)
Eigen Vector:
-0.37144         1.00000 -0.45080         0.43348

STEP-29:
Eigen Value = 18.60959(correct upto 5 decimal places)
Eigen Vector:
-0.37144         1.00000 -0.45080         0.43348

STEP-30:
Eigen Value = 18.60959(correct upto 5 decimal places)
Eigen Vector:
-0.37144         1.00000 -0.45080         0.43348
```

```
STEP-31:
Eigen Value = 18.60959(correct upto 5 decimal places)
Eigen Vector:
-0.37144        1.00000 -0.45080        0.43348

STEP-32:
Eigen Value = 18.60959(correct upto 5 decimal places)
Eigen Vector:
-0.37144        1.00000 -0.45080        0.43348

STEP-33:
Eigen Value = 18.60960(correct upto 5 decimal places)
Eigen Vector:
-0.37144        1.00000 -0.45080        0.43348

STEP-34:
Eigen Value = 18.60960(correct upto 5 decimal places)
Eigen Vector:
-0.37144        1.00000 -0.45080        0.43348
[Program finished]
```

**(15)**

## CURVE FITTING

- ***Introduction and Working Formula :-***



METHOD OF LEAST SQUARE

This method is a crucial statistical method that is practised to find a regression line or a best-fit line for the given pattern. This method is described by an equation with specific parameters. The method of least square is commonly used in evaluation & regression. In regression analysis, this method is said to be a standard approach for the approximation of sets of equations having more equations than the number of unknowns.

This actually defines the solution for the minimization of the sum of squares of deviations or the errors in the result of each equation. By finding the formula for sum of squares of errors which help to find the variation in observed data.

This method is often applied in data

fitting. The best fit result is assumed to reduce the sum of squared errors or residuals which are stated to be the differences between the observed or experimental value & corresponding fitted value given in the model.

There are two basic categories least-squares problems:

i) Ordinary or linear least squares.

ii) Nonlinear least squares.

These depend upon linearity or nonlinearity of the residuals. The linear problems are often seen in regression analysis in statistics. On the other hand, the non-linear problems generally used in the iterative method of refinement in which the model is approximated to the linear one with each iteration.

## Working Method for Fitting a curve of the Form $y = ax + b$

We have given, $Y = aX + b$ -- (i)

**Form Normal Equation:-**

$$\Sigma Y = a\Sigma X + nb \text{ -- (ii)}$$

$$\Sigma XY = a\Sigma X^2 + b\Sigma X \text{ -- (iii)}$$

Now, we will solve it for a & b.

From (i) we get, $a = \frac{\Sigma Y - nb}{\Sigma X}$ .. (iv)

Now, using the above in equation (ii) we get,

$$\Sigma XY = \frac{\Sigma Y - nb}{\Sigma X} * \Sigma X^2 + b\Sigma X$$

$$\Rightarrow \Sigma X * \Sigma XY = \Sigma X^2 * \Sigma Y - nb\Sigma X^2 + b\Sigma X * \Sigma X$$

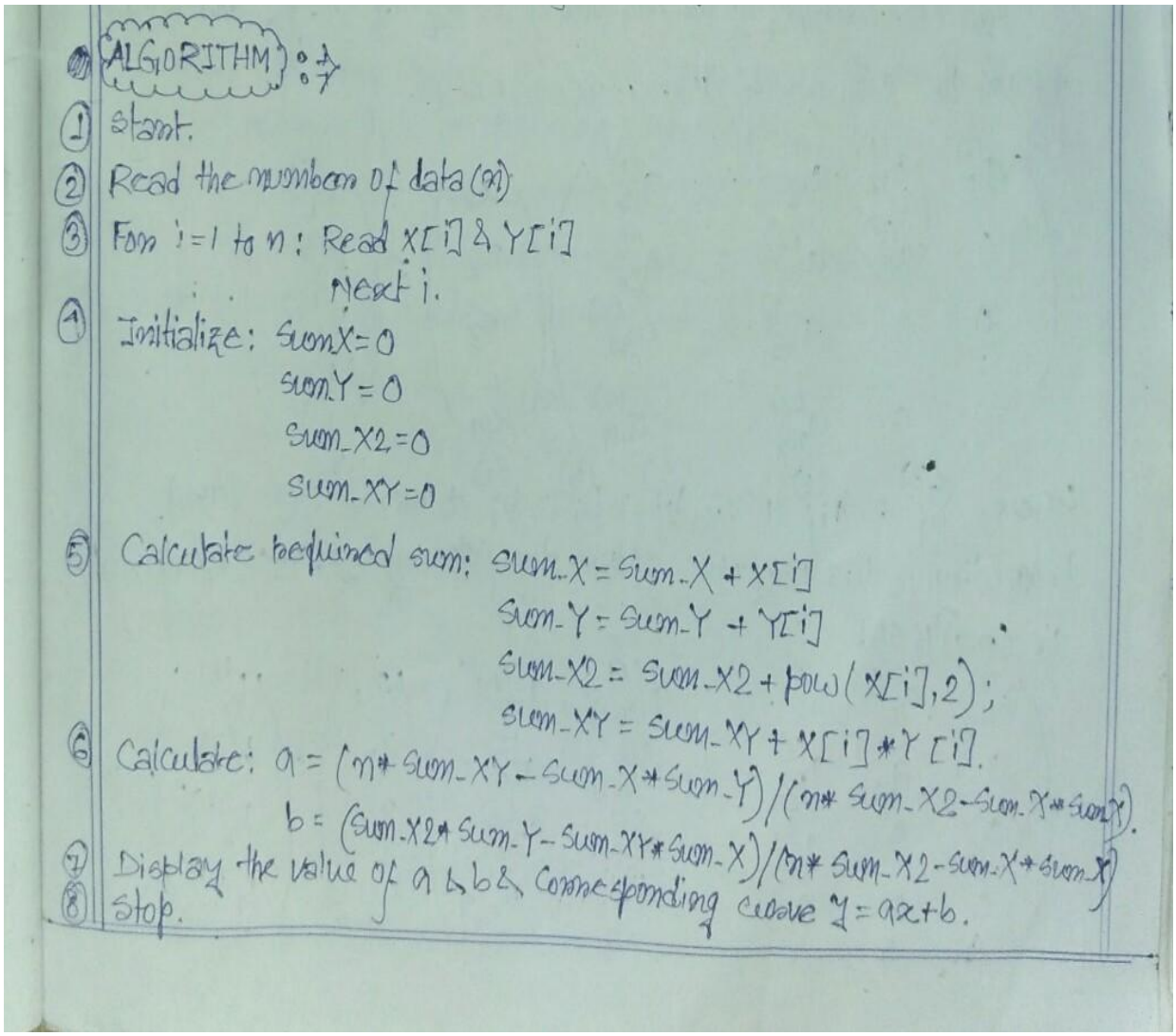
$$\Rightarrow \boxed{b = \frac{\Sigma X^2 * \Sigma XY - \Sigma X * \Sigma XY}{n\Sigma X^2 - \Sigma X * \Sigma X}}$$

Now, substituting above representation of b in (iv), we can easily calculate a.

**Best Fit curve substitution:**

Now, by using above a & b we can easily substitute them in $y = ax + b$ to find best fit curve.

- ***Algorithm:-***



ALGORITHM :-

1. Start.
2. Read the number of data (n)
3. For i=1 to n: Read X[i] & Y[i]
   Next i.
4. Initialize: Sum_X=0
   Sum_Y=0
   Sum_X2=0
   Sum_XY=0
5. Calculate required sum: Sum_X = Sum_X + X[i]
   Sum_Y = Sum_Y + Y[i]
   Sum_X2 = Sum_X2 + pow(X[i],2);
   Sum_XY = Sum_XY + X[i]*Y[i].
6. Calculate: a = (n* Sum_XY - Sum_X * Sum_Y)/(n* Sum_X2 - Sum_X * Sum_X).
   b = (Sum_X2 * Sum_Y - Sum_XY * Sum_X)/(n* Sum_X2 - Sum_X * Sum_X)
7. Display the value of a & b & Corresponding curve Y = aX+b.
8. Stop.

- ***Question***

**Write a C program to fit a curve of the form $y = ax + b$ to the following data:**

| $X$ | 1.9 | 2.3 | 2.7 | 3.1 | 3.5 | 3.9 | 4.3 | 4.7 |
|---|---|---|---|---|---|---|---|---|
| $y$ | 32 | 35 | 38 | $42 + k$ | $47 + k$ | $50 + k$ | $54 + k$ | $60 + k$ |

**Where $k = \frac{R}{10}$. R is the last of the University Roll Number.**

Sol:

**Input:**

**[The last digit of my University Roll Number being R=6,I will use as k=R/10=0.6 wherever the value of k is required in this problem and corresponding solution ]**

```
//Curve fitting of a straight line of the form y = a + bx//
#include<stdio.h>
#include<math.h>
int main()
{
   float x[11],y[11],sum_x=0,sum_y=0,sum_x2=0,sum_xy=0,a,b;
        int n,i;
        printf("Enter the number of data\n");
        scanf("%d",&n);
        printf("Enter the x data\n");
        for(i=1;i<=n;i++)
        scanf("%f",&x[i]);
        printf("Enter the y data\n");
```

```
        for(i=1;i<=n;i++)
        scanf("%f",&y[i]);
        for(i=1;i<=n;i++)
        {
                sum_x=sum_x+x[i];
                sum_y=sum_y+y[i];
                sum_x2=sum_x2+pow(x[i],2);
                sum_xy=sum_xy+pow(x[i],2);
        }
        a=(n*sum_xy-sum_x*sum_y/(n*sum_x2-sum_x*sum_x));
        b=(sum_x2*sum_y-sum_xy*sum_x)/(n*sum_x2-sum_x*sum_x);
        printf("\n The curve is y=%6.4fx+%6.4f\n",a,b);
        printf("correct up to four decimal places\n");
return 0;
}
```

- ***User Interface :-***

```
Enter the number of data
8
Enter the x data
1.9
2.3
2.7
3.1
3.5
3.9
4.3
4.7
Enter the y data
32
35
38
42.6
47.6
50.6
54.6
60.6

 The curve is y=573.4428*x+584.0576
correct up to four decimal places

[Program finished]
```

## ***BIBLIOGRAPHY:-***


1. ***Numerical Method With Basic Concept in C Programming(Author(s):Rahul Banerjee); TECHNO WORLD PUBLISHERS(April 2021).***
2. ***Introduction To Numerical Analysis (Author(s):A. Gupta, S. C. Bose); Academic Publishers(July 2019)***
3. ***Website:-CodeSansar.***