

8

SOME PROGRAMMING IN C

§ 8.1. Newton's forward difference interpolation formula.

Algorithm.

Step 1 : Start the program.

Step 2 : Enter the value of n (no. of given datas -1)

Step 3 : Enter the data $x_i, y_i (i = 0, 1, 2, \dots, n)$

Step 4 : Initialize $s = 0, p = 1$

Step 5 : Enter the interpolating point x

Step 6 : Calculate $h = x_1 - x_0, u = (x - x_0)/h$ and set $s = y_0$

Step 7 : For $j = 0(1)n$

Step 8 : $f_{0,j} = y_j$

Step 9 : For $i = 1(1)n, j = 0(1)n-i$,

Step 10 : $f_{i,j} = f_{i-1,j+1} - f_{i-1,j}$

Step 11 : For $i = 1(1)n$

Step 12 : $p = p(u-i+1)/i$

$$s = s + p f_{i,0}$$

Step 13 : Print the value of s .

Step 14 : Stop the program.

Problem : Find the value of y when $x = 1.142$ using the following data:

x	: 1.140	1.145	1.150	1.155	1.160
$y = f(x)$: 0.13102	0.13540	0.13976	0.14410	0.14842

Program :

```
/*Program for Newton's forward interpolation formula*/
#include<stdio.h>
#include<math.h>
void main()
{
    int i,n,j;
    float X,x[10],y[10],f[10][10],p=1,s=0,u,h;
    printf("\t\t Output \n");
    printf("*****\n");
    printf("Enter the value of n(no. of datas-1):\t");
    scanf("%d",&n);
    printf("Enter the value of fx and y \n");
    printf("x\t y\n");
    for(i=0;i<=n;i++)
        scanf("%f %f",&x[i],&y[i]);
    h=x[1]-x[0];
    printf("Enter the value of X to find y:");
    scanf("%f",&X);
    u=(X-x[0])/h;
    s=y[0];
    for(j=0;j<=n;j++)
        f[0][j]=y[j];
    for(i=1;i<=n;i++)
    {
        for(j=0;j<=n-i;j++)
            f[i][j]=f[i-1][j+1]-f[i-1][j];
    }
    for(i=1;i<=n;i++)
    {
        p=p*(u-i+1)/i;
        s=s+p*f[i][0];
    }
    printf("\ny(%f)=%f\n",X,s);
}
```

/*

Output

```
*****
Enter the value of n(no. of datas-1): 4
Enter the value of x and y
x      y
1.140  0.13102
1.145  0.13540
1.150  0.13976
1.155  0.14410
1.160  0.14842
Enter the value of X to find y:1.142
y(1.142000)=0.132774
```

*/

§ 8.2. Newton's backward difference interpolation formula

Algorithm.

Step 1 : Start the program.

Step 2 : Enter the value of n (no. of given datas -1)

Step 3 : Enter the data $x_i, y_i (i = 0, 1, 2, \dots, n)$

Step 4 : Initialize $s = 0, p = 1$

Step 5 : Enter the interpolating point x

Step 6 : Calculate $h = x_1 - x_0, u = (x - x_0)/h$ and set $s = y_n$

Step 7 : For $j = 0(1)n$

Step 8 : $f_{0,j} = y_j$

Step 9 : For $i = 1(1)n, j = 0(1)n$,

Step 10 : $f_{i,j} = f_{i-1,j} - f_{i-1,j-1}$

Step 11 : For $i = 1(1)n$

Step 12 : $p = p(u+i-1)/i$

$$s = s + p f_{i,n}$$

Step 13 : Print the value of s .

Step 14 : Stop the program.

Problem. Using Newton's backward difference interpolation formula, find the value of y when $x = 0.68$ from the following data :

x	0.1	0.2	0.3	0.4	0.5	0.6	0.7
$y = f(x)$	2.631	3.328	4.097	4.944	5.875	6.896	8.013

Program :

```
/*Program for Newton's backward interpolation formula*/
#include<stdio.h>
#include<math.h>
void main()
{
    int i,n,j;
    float X,x[10],y[10],f[10][10],p=1,s=0,u,h;
    printf("\t\t Output \n");
    printf("*****\n");
    printf("Enter the value of n(no. of datas-1):\t");
    scanf("%d",&n);
    printf("Enter the value of x and y \n");
    printf("x\t y \n");
    for(i=0;i<=n;i++)
        scanf("%f %f",&x[i],&y[i]);
    h=x[1]-x[0];
    printf("Enter the value of X to find y:");
    scanf("%f",&X);
    u=(X-x[n])/h;
    s=y[n];
    for(j=0;j<=n;j++)
        f[0][j]=y[j];
    for(i=1;i<=n;i++)
        for(j=i;j<=n;j++)
            f[i][j]=f[i-1][j-1] - (f[i-1][j-1] - f[i-1][j]) / (x[i] - x[i-1]);
    for(i=1;i<=n;i++)
        s=s+p*f[i][u];
}
```

```

    {
        for(j=i;j<=n;j++)
            f[i][j]=f[i-1][j]-f[i-1][j-1];
    }
    for(i=1;i<=n;i++)
    {
        p=p*(u+i-1)/i;
        s=s+p*f[i][n];
    }
    printf("y(%f)= %f\n",X,s);
}

```

Output

Enter the value of n(no. of datas-1): 6

Enter the value of x and y

x	y
0.1	2.631
0.2	3.328
0.3	4.097
0.4	4.944
0.5	5.875
0.6	6.896
0.7	8.013

Enter the value of X to find y:0.68

y(0.68000)= 7.781632

*/

§ 8.3. Lagrange's Interpolation**Algorithm****Step 1:** Start the program.**Step 2:** Enter the no. of given data (=n) and interpolating point x.**Step 3 :** Enter the data $x_i, y_i, i = 1, 2, \dots, n$ **Step 4 :** Initialize $s = 0, u = 1$ **Step 5 :** For $i = 1(1)n$.**Step 6 :** Initialize $d_i = 1$ **Step 7 :** For $j = 1(1)n$ **Step 8 :** If $i = j, A_{i,j} = x - x_i$, otherwise $A_{i,j} = x_i - x_j$ **Step 9 :** Calculate $d_i = \prod A_{i,j}$.**Step 10 :** Calculate $s = \sum \frac{y_i}{d_i}$ **Step 11 :** Calculate $u = \prod a_{i,i}$ **Step 12 :** Print the value of $u \times s$ **Step 13 :** Stop the program.**Problem :** Using Lagrange's interpolation formula, find the value of $y = f(x)$ when $x = 2.0$ using the following data :

x	: 0.0	1.0	3.0	4.0
$y = f(x)$: 5.0	6.0	50.0	105.0

Program :

```

#include<stdio.h>
#include<math.h>
void main()
{
    int i,n,j;
    float x[20],y[20],d[20],a[20][20],u=1.0,s=0.0;
    printf("\t\t Output\n");
    printf("*****\n");
    printf("\n Total given datas: ");
}

```

```

scanf("%d",&n);
printf("\nEnter x[i],y[i]\n");
for(i=1;i<=n;i++)
scanf("%f,%f",&x[i],&y[i]);
printf("\nEnter the Interpolation Point x=");
scanf("%f",&x[0]);
for(i=1;i<=n;i++)
{
d[i]=1.0;
for(j=1;j<=n;j++)
{
if(j==1)
a[i][j]=x[0]-x[j];
else
a[i][j]=x[i]-x[j];
d[i]=d[i]*a[i][j];
}
s=s+y[i]/d[i];
u=u*a[i][i];
}
printf("\n y(%f)=%f",x[0],u*s);
*/

```

Output

```
*****

```

Total no of given datas : 4

Enter x[i],y[i]

0.000000	5.000000
1.000000	6.000000
3.000000	50.000000
4.000000	105.000000

Enter the interpolating point x=2
 $y(2.000000) = 19.000000$

§ 8.4. Trapezoidal Rule.**Algorithm :**

Step 1 : Start the program.

Step 2 : Define the function $f(x)$

Step 3 : Read a, b, n

Step 4 : Set $x_0 = a, x_n = b$

Step 5 : Take $h = \frac{b-a}{n}$

Step 6 : For $i = 0 (1) n$

Step 7 : $x_i = x_0 + ih, y_i = f(x_i)$

Step 8 : Calculate $S = \sum_{i=1}^{n-1} y_i$
 $t = \frac{h}{2}(y_0 + y_n + 2S)$

Step 9 : Print the value of t .

Step 10 : Stop the program.

Problem : Evaluate $\int_0^1 \frac{dx}{1+x^2}$ using Trapezoidal rule, taking $n = 6$.

Program :

```

#include<stdio.h>
#include<math.h>
float f(float x)
{
    return(1/(1+x*x));
}

```

```

    }

void main()
{
    int i,n;
    float a,b,h,s=0,t;
    printf("\t\t Output \n");
    printf("*****\n Enter the lower limit :");
    scanf("%f",&a);
    printf("\n Enter the upper limit :");
    scanf("%f",&b);
    printf("\n Enter the number of subintervals:");
    scanf("%d",&n);
    h=(b-a)/n;
    for(i=1;i<=n-1;i++)
    {
        s=s+f(a+i*h);
        t=(h/2)*(f(a)+2*s+f(b));
        printf("\n\t The result is %6.4f\n",t);
    }
}

/*
Output
*****
Enter the lower limit :0
Enter the upper limit :1
Enter the number of subintervals:6
The result is 0.7842
*/

```

§ 8.5. Simpson's $\frac{1}{3}$ rd Rule.

Algorithm:

- Step 1 : Start the program.
- Step 2 : Define the function $f(x)$
- Step 3 : Read a, b, n

Step 4 : Set $x_0 = a, x_n = b$

Step 5 : Write $h = \frac{b-a}{n}$

Step 6 : For $i = 0(1)n$

Step 7 : $x_i = x_0 + ih, y_i = f(x_i)$

Step 8 : Calculate $S_1 = y_1 + y_2 + \dots + y_{n-1}$

$$S_2 = y_2 + y_4 + \dots + y_{n-2}$$

Step 9 : Set $S = \frac{h}{3} [f(a) + f(b) + 4S_1 + 2S_2]$

Step 10 : Print the value of S .

Step 11 : Stop the program.

Problem : Evaluate $\int_0^{\pi} \frac{dx}{1+x}$ using Simpson's $\frac{1}{3}$ rd rule taking 12 subintervals.

Program :

```

#include<stdio.h>
#include<math.h>
float f(float x)
{
    return(1/(1+x));
}
void main()
{
    int i,n;
    float a,b,h,s,x,y[20],s1=0,s2=0;

    printf("\t\t Output \n");
    printf("*****\n Enter the lower limit :");
    scanf("%f",&a);
    printf("\n Enter the upper limit :");
    scanf("%f",&b);
    printf("\n Enter the number of subintervals:");
    scanf("%d",&n);
}

```

```

h=(b-a)/n;
x=a;
for(i=0;i<=n;i++)
{
    y[i]=f(x);
    x=x+h;
}
for(i=1;i<n;i+=2)
s1=s1+4*y[i];
for(i=2;i<=n-2;i+=2)
s2=s2+2*y[i];
s=(h/3)*(y[0]+y[n]+s1+s2);
printf("\n\tThe result is %6.4f\n",s);
}

/*
Output
*****
Enter the lower limit :0
Enter the upper limit :0.6
Enter the number of subintervals:12
The result is 0.4700
*/

```

§ 8.6. Weddle's Rule.

Algorithm :

Step 1 : Start the program.

Step 2 : Define the function $f(x)$

Step 3 : Read a, b, n

Step 4 : Set $x_0 = a, x_n = b$

Step 5 : Write $h = \frac{b-a}{n}$

Step 6 : For $i = 0$ to n

Step 7 : $x_i = x_0 + ih, y_i = f(x_i)$

Step 8 : Calculate $S_1 = y_2 + y_4 + y_8 + y_{10} + \dots + y_{n-2}$

$$S_2 = y_1 + y_5 + y_7 + y_{11} + \dots + y_{n-1}$$

$$S_3 = y_3 + y_9 + y_{15} + \dots + y_{n-3}$$

$$S_4 = y_6 + y_{12} + y_{18} + \dots + y_{n-6}$$

Step 9 : Set $S = \frac{3h}{10} [f(a) + f(b) + S_1 + 5S_2 + 6S_3 + 2S_4]$

Step 10 : Print the value of S .

Step 11 : Stop the program.

Problem : Evaluate $\int_0^6 \frac{dx}{1+x^2}$ using weddle's rule, taking

$n=18$.

Program :

```

#include<stdio.h>
#include<math.h>
float f(float x)
{
    return(1/(1+x*x));
}
void main()
{
    int i,n;
    float a,b,h,s,x,y[20],s1=0,s2=0,s3=0,s4=0;
    printf("\t\tOutput\n");
    printf("*****\n");
    printf("\n Enter the lower limit :");
    scanf("%f",&a);
    printf("\n Enter the upper limit :");
    scanf("%f",&b);
    printf("\n Enter the number of subintervals:");
    scanf("%d",&n);
    h=(b-a)/n;
    x=a;
    for(i=0;i<=n;i++)
    {

```

```

y[i]=f(x);
x=x+h;
}
for(i=0;i<=n-6;i+=6)
{
    s1=s1+y[i+2]+y[i+4];
    s2=s2+y[i+1]+y[i+5];
    s3=s3+y[i+3];
}
for(i=0;i<=n-12;i+=6)
s4=s4+y[i+6];
s=(3*h/10)*(y[0]+y[n]+s1+5*s2+6*s3+2*s4);
printf("\n\tThe result is %6.6f\n",s);
}

```

/*

Output

Enter the lower limit :0

Enter the upper limit :6

Enter the number of subintervals:12

The result is 1.406975

*/

§ 8.7. Gauss Elimination Method.**Algorithm.****Step 1 :** Start the program.**Step 2 :** Read the augmented matrix

$$(A:b) = [a_{ij}], i=1, 2, \dots, n; j=1, 2, \dots, (n+1)$$

Step 3 : Compute $m_{ij} = \frac{a_{ij}}{a_{11}}$, $j = 2, 3, \dots, n+1$, provided $a_{11} \neq 0$ (for eliminate x_1 from last $(n-1)$ equations).**Step 4 :** Compute $a_{ij}^{(1)} = a_{ij} - m_{1j}a_{1j}$, $i = 2, 3, \dots, n$; $j = 2, 3, \dots, (n+1)$ **Step 5 :** Next compute $m_{2j} = \frac{a_{j2}^{(1)}}{a_{22}^{(1)}}$, $j = 3, 4, \dots, (n+1)$, provided $a_{22} \neq 0$ (for eliminating x_2 from the last $(n-2)$ equation with the help of the 2nd equation).**Step 6 :** Compute $a_{ij}^{(2)} = a_{ij}^{(1)} - m_{2j}a_{2j}^{(1)}$, $i = 3, 4, \dots, n$; $j = 3, 4, \dots, (n+1)$ **Step 7 :** Repeat the process until we get a triangular system of equations.**Step 8 :** Compute $x_n = a_{n(n+1)}^{(n-1)} / a_{nn}^{(n-1)}$

$$x_i = \frac{1}{a_{ii}^{(i-1)}} \left[a_{i,n+1}^{(i-1)} - \sum_{j=i+1}^n a_{ij}^{(i-1)} x_j \right] \text{ for } i = n-1, n-2, \dots, 2, 1$$

Step 9 : Print the value of x_i ($i = 1, 2, \dots, n$).**Step 10 :** Stop the program.**Problem :** Apply Gauss-Elimination method to solve the following system of equations.

$$10x_1 + 8x_2 - 3x_3 + x_4 = 16$$

$$2x_1 + 10x_2 + x_3 - 4x_4 = 9$$

$$3x_1 - 4x_2 + 10x_3 + x_4 = 10$$

$$2x_1 + 2x_2 - 3x_3 + 10x_4 = 11$$

Program :

/*Program for Gauss Elimination Method*/

#include<stdio.h>

void main()

{

int n,i,j,k;

float a[20][20],x[20],m,s=0.0;

```

printf("\t\t Output \n");
printf("*****\n");
printf("\n Enter the rank of the coefficient matrix :");
scanf("%d", &n);
printf("\n Enter the element of the augmented matrix
rowwise:\n");
for(i=1;i<=n;i++)
{
    for(j=1;j<=n+1;j++)
        scanf("%f",&a[i][j]);
}
for(i=1;i<=n-1;i++)
{
    for(j=i+1;j<=n;j++)
    {
        m=a[j][i]/a[i][i];
        for(k=1;k<=n+1;k++)
            a[j][k]=a[j][k]-m*a[i][k];
    }
}
printf("\n The upper trianguler matrix is :\n");
for(i=1;i<=n;i++)
{
    for(j=1;j<=n+1;j++)
        printf("%4.3f \t",a[i][j]);
    printf("\n");
}
x[n]=a[n][n+1]/a[n][n];
for(i=n-1;i>=1;i--)
{
    s=0.0;
    for(j=i+1;j<=n;j++)
        s=s+a[j][i]*x[j];
    x[i]=(a[i][n+1]-s)/a[i][i];
}
printf("\n\n the required soln is :\n");
for(i=1;i<=n;i++)

```

```

printf("x[%d]= %4.3f\n",i,x[i]);
}

```

```

/*
          Output
*****

```

Enter the rank of the coefficient matrix :4

Enter the element of the augmented matrix rowwise:

10	8	-3	1	16
2	10	1	-4	9
3	-4	10	1	10
2	2	-3	10	11

The upper trianguler matrix is :

10.000	8.000	-3.000	1.000	16.000
-0.000	8.400	1.600	-4.200	5.800
-0.000	0.000	12.119	-2.500	9.619
-0.000	0.000	0.000	9.489	9.489

The required soln is :

```

x[1]= 1.000
x[2]= 1.000
x[3]= 1.000
x[4]= 1.000
*/

```

§ 8.8. Gauss-Seidel iterative Method.

Algorithm.

Step 1 : Start the program.

Step 2 : Read the augmented matrix (a_{ij}) , $i=1$ to n ; $j=1$ to $(n+1)$

Step 3 : Enter the initial approximation $x_i = 0, i = 1 \text{ to } n$.

Step 4 : For $i = 1(1)n$.

Step 5 : Set $s = a_{i,n+1}$

Step 6 : For $j = 1(1)n$.

Step 7 : If $i \neq j$, set $s = s - a_{ij} x_j$;

Step 8 : Else, next j

Step 9 : $x_i = s/a_{ii}$

Step 10 : Print the value of $x_i (i = 1, 2, \dots, n)$.

Step 11 : Stop the program.

Problem : Solve the following system of equations using Gauss-Seidal iterative method.

$$8x_1 + 2x_2 - 2x_3 = 8$$

$$x_1 - 8x_2 + 8x_3 = -4$$

$$2x_1 + x_2 + 9x_3 = 12$$

Program :

```
#include<stdio.h>
void main()
{
    int n,i,j,k;
    float a[20][20],x[20],s;
    printf("\t\t Output \n");
    printf("*****\n");
    printf("\n Enter the order of the coefficient matrix :");
    scanf("%d", &n);
    printf("\n Enter the element of the augmented matrix
rowwise:\n");
```

```
for(j=1;j<=n;j++)
{
    for(i=1;i<=n+1;j++)
        scanf("%f",&a[i][j]);
}
printf("Enter the initial approximation:\n");
for(i=1;i<=n;i++)
    scanf("%f",&x[i]);
for(k=1;k<=15;k++)
{
    for(i=1;i<=n;i++)
    {
        s=a[i][n+1];
        for(j=1;j<=n;j++)
        {
            if(j!=i)
                s=s-a[i][j]*x[j];
        }
        x[i]=s/a[i][i];
    }
}
printf("\n\n the required soln is :\n");
for(i=1;i<=n;i++)
    printf("x[%d] = %6.4f\n",i,x[i]);
}
```

*


```

printf("\n %2d %f %f",i,a,m);
if(fabs(a-m)<=.000001) goto ans;
else {
    a=m;
    i=i+1;
    goto loop;
}
ans:
printf("\n*****");
printf("\n The required root is : %f",a);
getch();
} /* End of main()
*/

```

Output

Enter the initial gauss of the root a=1

n x[n] x[n+1]

0 1.000000 1.500000

1 1.500000 1.431813

2 1.431818 1.430501

3 1.430501 1.430501

The required root is=1.430501

*/

§ 8.10. Regula-Falsi Method.**Algorithm.**

Step 1 : Start the program.

Step 2 : Define the function $f(x)$.

Step 3 : Select the interval (a, b) in which the root lies.

Step 4 : Calculate $x = b - \frac{b-a}{f(b)-f(a)} f(b)$.

Step 5 : If $f(b)f(x) < 0$, set $b = x$, otherwise $a = x$.

Step 6 : If $|a-b| < \epsilon$, ϵ being the prescribed accuracy, then go to step 7 else step 4.

Step 7 : Print the value of x .

Step 8 : Stop the program.

Problem : Find a real root of the equation $x^3 - 5x + 1 = 0$ in $[0, 1]$ using Regula-Falsi Method.

Program :

```

/*Program for Finding a Real Root of a Nonlinear Equation
x^3-5x+1=0 by Regula-falsi Method*/
#include<stdio.h>
#include<conio.h>
#include<math.h>
float f(float x)
{
    return(x*x*x-5*x+1);
}
void main()
{
    clrscr();
    float x,a,b;
    printf("\t\tOutput\n");
    printf("Enter a,b : ");
    scanf("%f, %f",&a,&b);
}
```

```

printf("\n Table");
printf("\n*****");
printf("\n a[i]    b[i]    x[i+1]");
printf("\n*****");
line : x=((b-a)/(f(b)-f(a)))*f(b);
if(f(b)*f(x)<0)
    b=x;
else
    a=x;
printf("\n %f    %f    %f",a,b,x);
if(fabs(b-a)>=.000001) goto line;
printf("\n*****");
printf("\n The required root is : %f",x);
getch();
}
/*
Output
Enter a,b: 0,1
Table
*****
a[i]    b[i]    x[i+1]
*****
0.250000  1.000000  0.250000
0.250000  0.186441  0.186441
0.250000  0.201736  0.201736
0.250000  0.201639  0.201639

```

0.250000	0.201640	0.201640
0.251640	0.201640	0.201640

The required root is=0.201640

*/

§ 8.11. Euler's method.

Algorithm :

Step 1 : Start the program.

Step 2 : Define the function $f(x)$

Step 3 : Read x_0, y_0, h, x

Step 4 : Compute $n = \frac{x - x_0}{h}$

Step 5 : Set $x = x_0, y = y_0$

Step 6 : For $i = 0(1)n$

Step 7 : $y = y + h \times f(x, y)$

$x = x + h$

Step 8 : Print the value of y .

Step 9 : Stop the program.

Problem : Using Euler's method determine $y(0.6)$ given

$$\frac{dy}{dx} = x - y \text{ with } y(0) = 1, \text{ taking } h = 0.2.$$

Program :

/*Program of Euler's methode*/

```

#include<stdio.h>
#include<math.h>
float f(float x, float y)
{
    return(x-y);
}
int main()
{
    int i,n;
    float x0,y0,x,y,h;

```

```

printf("\t\t Output \n");
printf("*****");
printf("\n Enter the values of x0,y0,h,x:\n");
scanf("%f %f %f %f",&x0,&y0,&h,&x);
n=(x-x0)/h;
x=x0;y=y0;
for(i=0;i<=n;i++)
{
    y=y+h*f(x,y);
    x=x+h;
}
printf("\n The required value is %f\n",y);
}

/*
Output
*****
Enter the values of x0,y0,h,x:
0      1      0.2      0.6
The required value is 0.619200
*/

```

§ 8.12. Runge-Kutta Method of order 2.

Algorithm.

Step 1: Start the program.

Step 2: Define the function $f(x, y)$

Step 3: Read x_0, y_0, h, x

Step 4: Compute $n = \frac{x - x_0}{h}$

Step 5: Set $x = x_0, y = y_0$

Step 6: For $i = 0(1)n$

Step 7: $k_1 = hf(x, y)$

$$k_2 = hf(x + h, y + k_1)$$

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

$$y = y + k, x = x + h$$

Step 8: Print the value of y .

Step 9: Stop the program.

Problem : Find $y(1.3)$ using Runge-kutta method of order 2 given $\frac{dy}{dx} = x^2 + y^2, y(1) = 0$.

Program :

```

/*Program for Runge-kutta method of order 2*/
#include<stdio.h>
#include<math.h>
float f(float x,float y)
{
    return(x*x+y*y);
}
void main()
{
    int i,n;
    float x0,y0,x,y,k1,k2,h;
    printf("\t\t Output \n");
    printf("*****");
    printf("\n Enter the values of x0,y0,h,x:\n");
    scanf("%f %f %f %f",&x0,&y0,&h,&x);
    n=(x-x0)/h;
    x=x0;y=y0;
    for(i=0;i<=n;i++)
    {
        k1=h*f(x,y);
        k2=h*f(x+h,y+k1);
        k=(k1+k2)/2;
        x=x+h; y=y+k;
    }
    printf("\n The required value is %f\n",y);
}

/*
Output
*****
Enter the values of x0, y0, h, x:
1      0      0.1      1.3

```

The required value is 0.414260

*/

§ 8.13. Runge-Kutta Method of order 4.

Algorithm :

Step 1 : Start the program.

Step 2 : Define the function $f(x, y)$

Step 3 : Read x_0, y_0, h, x

Step 4 : Compute $n = \frac{x - x_0}{h}$

Step 5 : Set $x = x_0, y = y_0$

Step 6 : For $i = 0(1)n$

Step 7 : $k_1 = hf(x, y)$

$$k_2 = hf\left(x + \frac{h}{2}, y + \frac{k_1}{2}\right)$$

$$k_3 = hf\left(x + \frac{h}{2}, y + \frac{k_2}{2}\right)$$

$$k_4 = hf(x + h, y + k_3)$$

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

$$y = y + k, x = x + h.$$

Step 8 : Print the value of y .

Step 9 : Stop the program.

Problem : Find $y(1.4)$ using Runge-Kutta Method of order

4 given $\frac{dy}{dx} = x^2 + y^2 x, y(1) = 0$, taking $h = 0.1$.

Program :

/*Program of Runge-Kutta method of order 4*/

#include<stdio.h>

#include<math.h>

```
float f(float x, float y)
{
    return(x*x+y*y);
}

void main()
{
    int i, n;
    float x0, y0, x, y, k1, k2, k3, k4, k, h;

    printf("\t\t Output \n");
    printf("*****");
    printf("\n Enter the values of x0, y0, h, x:\n");
    scanf("%f %f %f %f", &x0, &y0, &h, &x);
    n = (x - x0)/h;
    x = x0; y = y0;
    for(i = 0; i <= n; i++)
    {
        k1 = h * f(x, y);
        k2 = h * f(x + h/2, y + k1/2);
        k3 = h * f(x + h/2, y + k2/2);
        k4 = h * f(x + h, y + k3);
        k = (k1 + (k2 + k3)*2 + k4)/6;
        x = x + h; y = y + k;
    }
    printf("\n The required value is %f\n", y);
}
/*

```

Output

Enter the values of x0,y0,h,x:

1 0 0.1 1.3

The required value is 0.413570

*/