C-Programming Problems

Mathematics Honours, J. K. College, Purulia

1. Using **TRAPEZOIDAL RULE**, evaluate the integral

$$\int_1^2 \frac{x^2+\sqrt{x^2+2x+5}}{x+\log_{10}(x^2+12x+2R)}dx$$

correct up to six places of decimal, taking 50 equal subintervals, where R is your Roll Number. The output should contain the limits of integration, number of subintervals, length of each sub-interval, and the required value of the integral.

2. Using **SIMPSON'S 1/3rd RULE**, evaluate the integral

$$\int_1^2 \frac{x^2+\sqrt{x^2+2x+5}}{x+\log_{10}(x^2+12x+2R)}dx$$

correct up to six places of decimal, taking 50 equal subintervals, where R is your Roll Number. The output should contain the limits of integration, number of subintervals, length of each sub-interval, and the required value of the integral.

3. Using **NEWTON-RAPHSON'S METHOD**, find a real root of the equation

$$x^3+7x^2-5\sin\left(\frac{x}{8}+\frac{3R}{100}\right)=0$$

correct up to six places of decimal, taking initial value $x_0=1.0$, where R is your Roll Number. The output should contain the initial approximation, tolerance, maximum number of iterations, the actual number of iterations taken and the required real root of the equation.

4. Using **FIXED POINT ITERATION METHOD**, find a real root of the equation

$$x^3+7x^2-5\sin\left(\frac{x}{8}+\frac{3R}{100}\right)=0$$

correct up to six places of decimal, taking initial value $x_0=1.0$, where R is your Roll Number. The output should contain the initial approximation, tolerance, maximum number of iterations, the actual number of iterations taken and the required real root of the equation.

5. Using **4TH-ORDER RUNGE-KUTTA METHOD**, find the value of y at $x=1.5$ from the initial value problem :

$$\frac{dy}{dx}=\frac{\frac{R}{10}+y-3}{4+\sin(x+y)} \qquad \text{with } y(1)=1,$$

taking step length $h=0.05$ correct up to six places of decimal, where R is your Roll Number. The output should contain the initial values of x, y, step length h, the values of y at $x=1.05, 1.10, \cdots, 1.50$ and the required result.

6. Using **MODIFIED EULER'S METHOD**, find the value of y at $x = 1.5$ from the initial value problem :

$$\frac{dy}{dx} = \frac{\frac{R}{10} + y - 3}{4 + \sin(x + y)} \quad \text{with } y(1) = 1,$$

taking step length $h = 0.05$ correct up to six places of decimal, where R is your Roll Number. The output should contain the initial values of x, y, step length h, the values of y at $x = 1.05, 1.10, \cdots, 1.50$ and the required result.

7. Using **LAGRANGE'S INTERPOLATION FORMULA**, find the value of y for $x = 1.0 + 0.0001R$ from the set of values :

x	y
1.00	3.61 23 599
1.01	3.62 75 156
1.02	3.64 25 829
1.03	3.65 75 630
1.04	3.67 24 569
1.05	3.68 72 658

correct up to six places of decimal, where R is your Roll Number.
The output should contain the number of points, the tabular values of x and y, the value of x for which y is to be computed and the required result.

8. Using **NEWTON'S FORWARD INTERPOLATION FORMULA**, find the value of y for $x = 1.0 + 0.0001R$ from the set of values :

x	y
1.00	3.61 23 599
1.01	3.62 75 156
1.02	3.64 25 829
1.03	3.65 75 630
1.04	3.67 24 569
1.05	3.68 72 658

correct up to six places of decimal, where R is your Roll Number.
The output should contain the number of points, the tabular values of x and y, the value of x for which y is to be computed and the required result.

9. Using **NEWTON'S BACKWARD INTERPOLATION FORMULA**, find the value of y for $x = 1.05 - 0.0001R$ from the set of values :

x	y
1.00	3.61 23 599
1.01	3.62 75 156
1.02	3.64 25 829
1.03	3.65 75 630
1.04	3.67 24 569
1.05	3.68 72 658

correct up to six places of decimal, where R is your Roll Number.
The output should contain the number of points, the tabular values of x and y, the value of x for which y is to be computed and the required result.

TRAPEZOIDAL RULE

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

float f(float x)
{
   float fun;
   fun = (x * x + sqrt(x * x + 2 * x + 5.0)) / (x + log(x * x + 12 * x + 2 * 2.0) / log(10));
   return fun;
}

int main()
{
   float a, b, h, x0, sum = 0, trap;
   int i, n;

   printf("Enter the lower limit, Upper limit, Number of sub–intervals:\n");
   scanf("%f%f%d", &a, &b, &n);
   h = (b – a) / n;
   printf("lower limit=%.1f      upper limit=%.1f\n", a, b);
   printf("Length of subinterval=%.2f      no of subinterval=%d\n", h, n);
   x0 = a;

   for (i = 1; i <= n; i++)
   {
      sum += f(x0) + f(x0 + h);
      x0 = x0 + h;
   }

   trap = h / 2.0 * sum;

   printf("The value of the Integral = %.6f", trap);
   return 0;
}
```

SIMPSON'S 1/3rd RULE

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

   float f(float x)
{
   float fun;
   fun = (x * x + sqrt(x * x + 2 * x + 5.0)) / (x + log(x * x + 12 * x + 2 * 2.0) / log(10));
   return fun;
}

int main()
{
   float a, b, h, x0, sum = 0, simp;
   int i, n;

   printf("Enter the lower limit, Upper limit, Number of sub–intervals:\n");
   scanf("%f%f%d", &a, &b, &n);
   h = (b – a) / n;

   printf("lower limit=%.1f      upper limit=%.1f\n", a, b);
   printf("Length of subinterval=%.2f      no of subinterval=%d\n", h, n);
   x0 = a;

   for (i = 1; i <= n / 2; i++)
   {
      sum += f(x0) + 4 * f(x0 + h) + f(x0 + 2 * h);
      x0 = x0 + 2.0 * h;
   }

   simp = h / 3.0 * sum;
   printf("the value of the Integral = %.6f", simp);

   return 0;
}
```

NEWTON–RAPHSON METHOD

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

   float f(float x)
{
   float fun;
   fun = pow(x, 3) + 7 * x * x – 5 * sin(x / 8 + 3.0 * 2 / 100);
   return fun;
}
float df(float x)
{
   float dfun;
   dfun = 3 * x * x + 14 * x – 5.0 / 8 * cos(x / 8 + 3.0 * 2 / 100);
   return dfun;
}

int main()
{
   float x0, x1, tol = 0.0000005, esp = 0.01;
   int maxit = 100, i = 0;
   printf("Enter the value of x0\n");
   scanf("%f", &x0);

   if (fabs(df(x0)) < esp)
   {
      printf("derivative is small \n");
   }
   printf("x0 = %.2f      Tol = %.7f      maxit = %d\n", x0, tol, maxit);

step2:
   i = i + 1;
   if (i > maxit)
   {
      printf("Iteration is not sufficient");
      goto end;
   }

   x1 = x0 – f(x0) / df(x0);
   printf("I = %d          x = %.7f\n", i, x1);

   if (fabs(x1 – x0) < tol)
      goto step1;

   x0 = x1;
   goto step2;

step1:
   printf("no of iteration = %d       real root = %.6f\n", i, x1);

end:;
   return 0;
}
```

FIXED POINT ITERATION METHOD

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

float f(float x)
{
   float fun;
   fun = pow(x, 3) + 7 * x * x – 5 * sin(x / 8 + 3.0 * 2 / 100);
   return fun;
}
float fi(float x)
{
   float fifun;
   fifun = sqrt(5.0 * sin(x / 8.0 + 3.0 * 2 / 100) / (x + 7.0));
   return fifun;
}
float dfi(float x)
{
   float dfifun;
   dfifun = 2.5 * sqrt((x + 7.0) / (5.0 * sin(x / 8.0 + 3.0 * 2 / 100))) * (cos(x / 8 + 3.0 * 2 / 100) * (x + 7.0) – 8 * sin(x / 8
+ 3.0 * 2 / 100)) / (8.0 * (x + 7.0) * (x + 7.0));
   return dfifun;
}

int main()
{
   float x0, x1, tol = 0.0000005;
   int maxit = 100, i = 0;

   printf("Supply x0\n");
   scanf("%f", &x0);

   if (fabs(dfi(x0)) >= 1)
   {
      printf("it is not convergent");
      goto end;
   }

   printf("x0 = %.2f      tolerance = %.7f     maxit = %d\n", x0, tol, maxit);

step1:
   i = i + 1;
   if (i > maxit)
   {
      printf("iteration is not sufficient");
      goto end;
   }

   x1 = fi(x0);
   printf("I = %d         x = %.7f\n", i, x1);

   if (fabs(x1 – x0) < tol)
      goto step2;
```

```
    x0 = x1;
    goto step1;

step2:
    printf("no of iteration = %d        real root = %.6f\n", i, x1);

end:;
    return 0;
}
```

RUNGE–KUTTA METHOD

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

   float f(float x, float y)
{
   float fun;
   fun = (0.2 + y – 3) / (4 + sin(x + y));
   return fun;
}

int main()
{
   float x0, y0, xn, h, d, d1, d2, d3, d4, x, y, n;
   int i;

   printf("intput x0,xn,y0,h\n");
   scanf("%f%f%f%f", &x0, &xn, &y0, &h);

   n = (xn – x0) / h;
   printf("x0 = %.2f     xn = %.2f     y0 = %.2f     h = %.2f      n = %.f\n", x0, xn, y0, h, n);

   x = x0, y = y0;
   for (i = 1; i <= n; i++)
   {
      d1 = h * f(x, y);
      d2 = h * f(x + h / 2, y + d1 / 2);
      d3 = h * f(x + h / 2, y + d2 / 2);
      d4 = h * f(x + h, y + d3);
      d = (d1 + 2 * d2 + 2 * d3 + d4) / 6;
      y = y + d;
      x = x + h;
      printf("x = %.2f          y = %.8f\n", x, y);
   }

   printf("At x = %.2f       y= %.6f", x, y);

   return 0;
}
```

MODIFIED EULER'S METHOD

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

   float f(float x, float y)
{
   float fun;
   fun = (0.2 + y – 3) / (4.0 + sin(x + y));
   return (fun);
}

int main()
{
   float x0, y0, xn, x, y, yp, yc, h, tol = 0.000005, n;
   int maxit = 100, i = 0, j;

   printf("enter x0,y0,xn,h\n");
   scanf("%f%f%f%f", &x0, &y0, &xn, &h);

   n = (xn – x0) / h;
   printf("x0=%.2f     y0=%.2f     xn=%.2f     n=%.f      h=%.2f\n", x0, y0, xn, n, h);

   x = x0, y = y0;
   for (j = 1; j <= n; j++)
   {
      yp = y + h * f(x, y);

   step1:
      i = i + 1;
      if (i > maxit)
         printf("iteration is not sufficient");

      yc = y + h / 2 * (f(x, y) + f(x + h, yp));
      if (fabs(yp – yc) > tol)
      {
         yp = yc;
         goto step1;
      }
      y = yc;
      x = x + h;

      printf("x=%.2f      y=%.8f\n", x, y);
   }

   printf("At x=%.2f      y=%.6f\n", x, y);

   return 0;
}
```

LAGRANGE'S INTERPOLATION

```c
#include <stdio.h>
#include <math.h>
int main()
{
   float x[6] = {1.00, 1.01, 1.02, 1.03, 1.04, 1.05};
   double y[6] = {3.6123599, 3.6275156, 3.6425829, 3.6575630, 3.6724569, 3.6872658};
   float z, coef, sum = 0;
   int n, i, j, k;
   z = 1.0 + 0.0001 * 2;

   printf("enter no of points\n");
   scanf("%d", &n);
   printf("the no of points=%d \n", n);

   for (k = 0; k < n; k++)
      printf("x = %.2f         y = %.7f\n", x[k], y[k]);

   for (i = 0; i < n; i++)
   {
      coef = 1;
      for (j = 0; j < n; j++)
      {
         if (i != j)
            coef = coef * (z – x[j]) / (x[i] – x[j]);
      }
      sum = sum + coef * y[i];
   }

   printf("At x = %.4f        y = %.6f\n", z, sum);

   return 0;
}
```

NEWTON'S FORWARD INTERPOLATION

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

   int main()
{
   float x[6] = {1.00, 1.01, 1.02, 1.03, 1.04, 1.05};
   double y[6] = {3.6123599, 3.6275156, 3.6425829, 3.6575630, 3.6724569, 3.6872658};
   float d[10][10], coef, sum, h, z, u;
   int i, j, n;

   printf("enter no of points\n");
   scanf("%d", &n);

   z = 1.0 + 0.0001 * 2;
   h = x[1] – x[0];
   u = (z – x[0]) / h;
   printf("no of points=%d     h=%.2f\n", n, h);

   for (i = 0; i < n; i++)
      printf("x = %.2f         y = %.7f\n", x[i], y[i]);

   for (i = 0; i < n; i++)
      d[0][i] = y[i];

   for (i = 1; i < n; i++)
   {
      for (j = 0; j < n; j++)
         d[i][j] = d[i – 1][j + 1] – d[i – 1][j];
   }

   sum = y[0];
   coef = 1;

   for (i = 1; i < n; i++)
   {
      coef = coef * (u + 1 – i) / i;
      sum = sum + coef * d[i][0];
   }

   printf("At x = %.4f         y = %.6f\n", z, sum);

   return 0;
}
```