Московский авиационный институт
(Национальный исследовательский университет)
Факультет прикладной математики и физики
Кафедра математической кибернетики

Лабораторная работа № 3

по курсу «Численные методы»

Тема: Приближение функций.

Численные дифференцирование и интегрирование.

Студент: Мукин Ю. Д.
Группа: 80-304Б-18
Преподаватель: Гидаспов В.Ю.

Москва, 2021

Задание 1

1) Постановка задачи: Используя таблицу значений Y_i функции $y = f(x)$, вычисленных в точках X_i, $i = 0,...,3$ построить интерполяционные многочлены Лагранжа и Ньютона, проходящие через точки $\{X_i, . Y_i\}$ Вычислить значение погрешности интерполяции в точке X^*.

Вариант 12:

$$\begin{cases} y=\sin(x)+x, \\ \text{а) } X_i=0,\frac{\pi}{6},\frac{2\pi}{6},\frac{3\pi}{6}; \\ \text{б) } X_i=0,\frac{\pi}{6},\frac{\pi}{4},\frac{\pi}{2}; \\ X^*=1.0 \end{cases}$$

2) Теория:

Интерполяционный многочлен, записанный в форме: $L_n(x)=\sum_{i=0}^{n} f_i \prod_{j=0,\ j\neq i}^{n} \frac{(x-x_i)}{(x_i-x_j)}.$,

называется интерполяционным многочленом Лагранжа.

Запись многочлена в формуле:

$$P_n(x)=f(x_0)+(x-x_0)f(x_1,x_0)+(x-x_0)(x-x_1)f(x_0,x_1,x_2)+...+ \\ +(x-x_0)(x-x_1)...(x-x_n)f(x_0,x_1,...,x_n).$$ -

есть так называемый интерполяционный многочлен Ньютона.

По сути это один и тот же полином, но с разной записью.

3) Результат работы программы:

```
enter X: 0 0.5234 1.0472 1.5708
Y: 0.000000; 1.023228; 1.913227; 2.570800;
enter x of point: 1
L(x) = -0.000000(x - 0.523400)(x - 1.047200)(x - 1.570800)+3.563367(x + -0.000000)(x - 1.047200)(x - 1.570800)-6.661494(x + -0.000000)(x - 0.523400)(x - 1.570800)+2.984250(x + -0.000000)(x - 0.523400)(x - 1.047200)
L(x) = 1.841086
y(x) = 1.841471
delta = 0.000385

Newton polynomial

N(x) = 0.000000 + 1.954963(x - 0.000000 ) + -0.244312(x - 0.000000)(x - 0.523400) + -0.113877(x - 0.000000)(x - 0.523400)(x - 1.047200)
N(x) = 1.841086
```

y(x) = 1.841471
delta = 0.000385

enter X: 0 0.5236 0.7854 1.5708
Y: 0.000000; 1.023601; 1.492508; 2.570800;
enter x of point: 1
L(x) = -0.000000(x - 0.523600)(x - 0.785400)(x - 1.570800)+7.130694(x + -0.000000)(x - 0.785400)(x - 1.570800)-9.241985(x + -0.000000)(x - 0.523600)(x - 1.570800)+1.989880(x + -0.000000)(x - 0.523600)(x - 0.785400)
L(x) = 1.843136
y(x) = 1.841471
delta = 0.001665

Newton polynomial

N(x) = 0.000000 + 1.954929(x - 0.000000) + -0.208608(x - 0.000000)(x - 0.523600) + -0.121411(x - 0.000000)(x - 0.523600)(x - 0.785400)
N(x) = 1.843136
y(x) = 1.841471
delta = 0.001665

4) Код программы

```
void count_y(double(*fun)(double x),double Y[], double X[], int n)
{
   printf("Y: ");
   for(int i = 0; i<n; i++)
   {
      Y[i] = fun(X[i]);
      printf("%lf; ", Y[i]);
   }
}

double omega_function_derivative(int k, int n, double X[])
{
   double res =1;
   for(int i=0; i<n; i++)
      if(i!=k) res*= X[k]-X[i];
   return res;
}

double omega_function_Newton(double x, int n, double X[])
{
   double res = 1;
   for(int i=0; i<=n; i++)
      res*= x-X[i];
   return res;
}

double f(int n, int i, int j, double X[], double Y[])
{
   if(n==0)
      return (Y[i] - Y[j]) / (X[i] - X[j]);
   else
      return (f(n-1, i, j-1, X, Y) - f(n-1, i+1, j, X, Y)) / (X[i] - X[j]);
}

double newton_polynom(double(*fun)(double x), double x, int n, double X[], double Y[])
{
   double N = fun(X[0]) + (x- X[0])*f(0, 1, 0, X, Y);
   printf("\nN(x) = %lf + %lf(x - %lf )", fun(X[0]), f(0, 1, 0, X, Y), X[0]);
   for(int i=1; i<n-1; i++)
   {
      double tmp = f(i, 0, i+1, X, Y);
```

```
        printf(" + %lf", tmp);
        for(int j=0; j<=i; j++)
            printf("(x - %lf)", X[j]);
        N += omega_function_Newton(x, i, X)*tmp;
    }
    return N;
}

double lagrange_polinom(double x, int n, double X[], double Y[])
{
    double L = 0;
    printf("L(x) = ");
    for(int i=0; i<n; i++)
    {
        double tmp = Y[i]/omega_function_derivative(i, n, X);
        if(tmp>0 && i>0) printf("+");
        printf("%lf", tmp);
        for(int j=0; j<n; j++)
        {
            if(i!=j)
                X[j]>0 ? printf("(x - %lf)",X[j]):printf("(x + %lf)",-X[j]);
        }
        L += (omega_function_Lagrange(x, n, X) * Y[i])/((x - X[i]) * omega_function_derivative(i, n, X));
    }
    return L;
}

void Lagrange_and_Newton_polynomial()
{
    int n;
    double x, L, N;
    printf("\n\nLagrange polynomial\n");
    printf("\nenter quantity of x: ");
    scanf("%d",&n);
    double *X = malloc(n*sizeof(double)); double *Y = malloc(n*sizeof(double));
    printf("\nenter X: ");
    for(int i = 0; i<n; i++)
        scanf("%lf", &X[i]);
    count_y(function_v12_1, Y, X, n);
    printf("\nenter x of point: ");
    scanf("%lf",&x);
    L = lagrange_polinom(x, n, X, Y);
    printf("\nL(x) = %lf", L);
    printf("\ny(x) = %lf", function_v12_1(x));
    printf("\ndelta = %lf\n",fabs(L - function_v12_1(x)));

    printf("\nNewton polynomial\n");
    N = newton_polynom(function_v12_1, x, n, X, Y);
    printf("\nN(x) = %lf", N);
    printf("\ny(x) = %lf", function_v12_1(x));
    printf("\ndelta = %lf",fabs(N - function_v12_1(x)));
    free(X); free(Y);
}
```

Задание 2

1) Постановка задачи: Построить кубический сплайн для функции, заданной в узлах интерполяции, предполагая, что сплайн имеет нулевую кривизну при $x = x_0$ и $x = x_4$. Вычислить значение функции в точке $x = X^*$.

Вариант 12:

$X^* = 0.8$

i	0	1	2	3	4
x_i	0.0	0.5	1.0	1.5	2.0
f_i	0.0	0.97943	1.8415	2.4975	2.9093

2) Теория: Для построения кубического сплайна необходимо построить n многочленов третьей степени, т.е. определить 4n неизвестных . Эти коэффициенты ищутся из условий в узлах сетки

$$
\begin{aligned}
&S(x_{i-1}) = a_i = a_{i-1} + b_{i-1}(x_{i-1} - x_{i-2}) + c_{i-1}(x_{i-1} - x_{i-2})^2 + d_{i-1}(x_{i-1} - x_{i-2})^3 = f_{i-1} \\
&S'(x_{i-1}) = b_i = b_{i-1} + 2c_{i-1}(x_{i-1} - x_{i-2}) + 3d_{i-1}(x_{i-1} - x_{i-2})^2, \\
&S''(x_{i-1}) = 2c_i = 2c_{i-1} + 6d_{i-1}(x_{i-1} - x_{i-2}), \qquad\qquad i = 2,3,\ldots,n \\
&S(x_0) = a_1 = f_0 \qquad\qquad , \\
&S''(x_0) = c_1 = 0 \\
&S(x_n) = a_n + b_n(x_n - x_{n-1}) + c_n(x_n - x_{n-1})^2 + d_n(x_n - x_{n-1})^3 = f_n \\
&S''(x_n) = c_n + 3d_n(x_n - x_{n-1}) = 0
\end{aligned}
$$

3) Результат работы программы:

```
Spline: 0.000000 + 2.001014(x - 0.000000) + 0.000000(x - 0.000000)^2 + -0.168617(x - 0.000000)^3
Spline: 0.979430 + 1.874551(x - 0.500000) + -0.252926(x - 0.500000)^2 + -0.095794(x – 0.500000)^3 #
Spline: 1.841500 + 1.549780(x - 1.000000) + -0.396617(x - 1.000000)^2 + -0.157886(x - 1.000000)^3
Spline: 2.497500 + 1.034749(x - 1.500000) + -0.633446(x - 1.500000)^2 + 0.422297(x - 1.500000)^3
f(x) = 1.514479
```

4) Код программы

```
void get_c_value(double x[], double y[], int n, double t[])
{
   t[0] = 0;
   matrix A = create_matrix(n-2,n-2);
   matrix B = create_matrix(1, n-2);
   *get_element(&A, 0,0) = 2*(h(1, x)+h(2, x));
   *get_element(&A, 0,1) = h(2, x);
   *get_element(&B, 0,0) = 3*(((y[2] - y[1]) / h(2, x)) - ((y[1] - y[0]) / h(1, x)));
   int j=0;
   for(int i=3; i<n-1; i++)
   {
      *get_element(&A, i-2,j) = h(i-1,x);
      *get_element(&A, i-2,j+1) = 2*(h(i-1, x)+h(i, x));
      *get_element(&A, i-2,j+2) = h(i,x);
      *get_element(&B, 0,i-2) = 3*(((y[i] - y[i-1]) / h(i, x)) - ((y[i-1] - y[i-2]) / h(i-1, x)));
      j++;
   }
```

```
    *get_element(&A, n-3,j) = h(n-2, x);
    *get_element(&A, n-3,j+1) = 2*(h(n-2, x)+h(n-1, x));
    *get_element(&B, 0,n-3) = 3*(((y[n-1] - y[n-2]) / h(n-1, x)) - ((y[n-2] - y[n-3]) / h(n-2, x)));
    matrix X = run_method(A, B, n-2);
    for (int j=0; j<n-2; j++)
        t[j+1] = X.body[j];
    destroy_matrix(&A); destroy_matrix(&B); destroy_matrix(&X);
}

double get_spline(int n, double X, double a[], double b[], double c[], double d[], double x[])
{
    int j;
    double k;
    for(k=x[0], j=0; k<=x[n-1]; j++, k++)
    {
        if ((k<=X)&&(k>=X-1)) break;
    }
    for(int i=0; i<n-1;i++)
    {
        printf("\nSpline: %lf + %lf(x - %lf) + %lf(x - %lf)^2 + %lf(x - %lf)^3", a[i], b[i], x[i], c[i], x[i], d[i], x[i]);
        if (i==j)
            printf(" #");
    }
    double t = X-x[j];
    return a[j] + b[j]*t + c[j]*t*t + d[j]*t*t*t;
}

void get_a_b_d(int n, double a[], double b[], double d[], double x[], double y[], double c[])
{
    for(int i=1; i<n; i++)
    {
        a[i-1] =  y[i-1];
        b[i-1] = ((y[i] - y[i-1])/h(i,x)) - ((h(i,x)*(c[i] + 2*c[i-1]))/3);
        d[i-1] = (c[i] - c[i-1]) / (3*h(i,x));
    }
    a[n-1] = y[n-1];
    b[n-1] = ( ((y[n] - y[n-1]) / h(n-1,x)) - 2*h(n-1,x)*c[n-1]/3 );
    d[n-1] = (-c[n-1]/(3*h(n-1, x)));
}

void cubic_spline()
{
    int n;
    printf("specify the number of records: ");
    scanf("%d", &n);
    double **table  = malloc(2*sizeof(double*));
    printf("enter values from table: \n");
    for(int i=0; i<2; i++)
    {
        table[i] = malloc(n*sizeof(double));
        for (int j=0; j<n; j++)
            scanf("%lf", &table[i][j]);
    }
    double *c =  malloc(n*sizeof(double));
    double *a = malloc(n*sizeof(double));
    double *b = malloc(n*sizeof(double));
    double *d = malloc(n*sizeof(double));
    get_c_value(table[0], table[1], n, c);
    get_a_b_d(n-1,a,b,d,table[0], table[1], c);
    double X;
    printf("enter the point at which you want to find the value: ");
    scanf("%lf", &X);
    double r = get_spline(n, X, a, b, c, d, table[0]);
```

```
    printf("\nf(x) = %lf ", r);
    free(table); free(a); free(b); free(c);free(d);
}
```

Задание 3

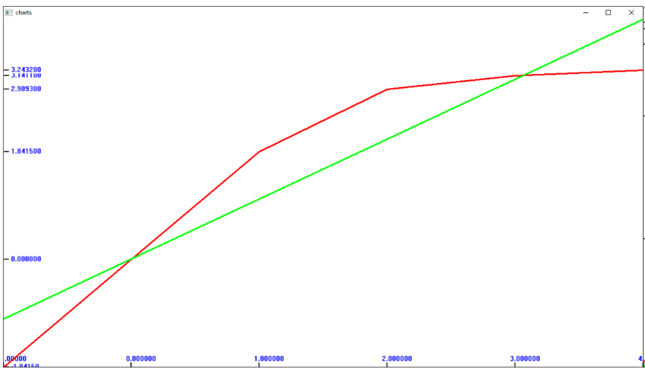
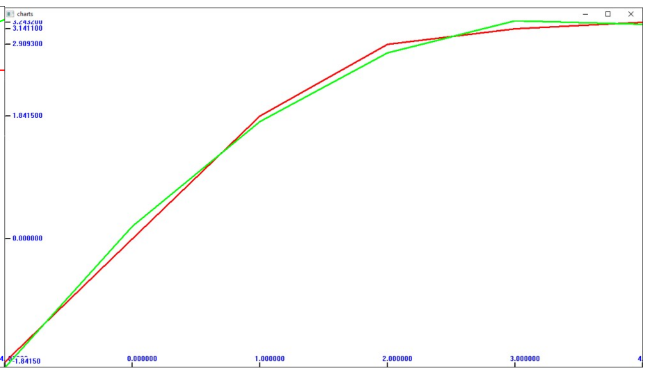
1) Для таблично заданной функции путем решения нормальной системы МНК найти приближающие многочлены а) 1-ой и б) 2-ой степени. Для каждого из приближающих многочленов вычислить сумму квадратов ошибок. Построить графики приближаемой функции и приближающих многочленов.

Вариант 12:

i	0	1	2	3	4	5
x_i	-1.0	0.0	1.0	2.0	3.0	4.0
y_i	-1.8415	0.0	1.8415	2.9093	3.1411	3.2432

2) Теория : Система $\sum_{i=0}^{n} a_i \sum_{j=0}^{N} x_j^{k+i} = \sum_{j=0}^{N} y_j x_j^k,\ k = 0, 1, \ldots, n.$ называется нормальной системой метода наименьших квадратов (МНК) представляет собой систему линейных алгебраических уравнений относительно коэффициентов . Решив систему, построим многочлен , приближающий функцию и минимизирующий квадратичное отклонение.

3) Результат работы программы:

-1 0 1 2 3 4
-1.8415 0.0 1.8415 2.9093 3.1411 3.2432
enter the degree of the polynomial: 2
a0*6.000000 + a1*9.000000 + a2*31.000000 = 9.293600
a0*9.000000 + a1*31.000000 + a2*99.000000 = 31.897700
a0*31.000000 + a1*99.000000 + a2*355.000000 = 91.798300

F(x) = 0.189924 + 1.836978x^1 - 0.270282x^2
-1.917336 0.189924 1.756620 2.782751 3.268319 3.213321
(y-F)^2 = 0.082119

-1 0 1 2 3 4
-1.8415 0.0 1.8415 2.9093 3.1411 3.2432
enter the degree of the polynomial: 1
a0*6.000000 + a1*9.000000 = 9.293600
a0*9.000000 + a1*31.000000 = 31.897700

F(x) = 0.009736 + 1.026131x^1
-1.016395 0.009736 1.035868 2.061999 3.088130 4.114262
(y-F)^2 = 2.809410

4) Код программы

```
void find_a_vector(int n, int N, double x[], double y[], double A[])
{
  double tmp_x, tmp_y;
  matrix a, b;
  a = create_matrix(n,n); b = create_matrix(1,n);
  for(int i=0; i<n; i++)
  {
    for(int j=0; j<n; j++)
    {
      tmp_x = 0;
      tmp_y = 0;
      for(int k=0; k<N; k++)
      {
        tmp_x += pow(x[k], i+j);
        tmp_y += y[k]*pow(x[k], i);
      }
      *get_element(&a,i,j) = tmp_x;
      printf("a%d*%lf + ", j, tmp_x);
    }
    printf("\b\b= %lf\n", tmp_y);
    *get_element(&b,0,i) = tmp_y;
  }
  matrix tmp = LU_method(a, b, n);
  for(int i=0; i<n; i++)
    A[i] = tmp.body[i];
  printf("\nF(x) = %lf",A[0]);
  for(int i=1; i<n; i++)
    if(A[i]>0)
```

```
            printf(" + %lfx^%d",A[i], i);
        else
            printf(" - %lfx^%d",-A[i], i);
    destroy_matrix(&a); destroy_matrix(&b); destroy_matrix(&tmp);
}

double get_value(double A[], double x, int n)
{
    double tmp=0;
    for(int i=0; i<n; i++)
        tmp+=A[i]*pow(x, i);
    return tmp;
}

void approximating_polynomials()
{
    int n, N;
    printf("specify the number of records: ");
    scanf("%d", &n);
    double **table  = malloc(2*sizeof(double*));
    double *A = malloc(5*sizeof(double));
    printf("enter values from table: \n");
    for(int i=0; i<2; i++)
    {
        table[i] = malloc(n*sizeof(double));
        for (int j=0; j<n; j++)
            scanf("%lf", &table[i][j]);
    }
    printf("enter the degree of the polynomial: ");
    scanf("%d", &N);
    N++;
    find_a_vector(N, n, table[0], table[1], A);
    double tmp = 0;
    double tmp1=0;
    printf("\n");
    for(int i=0; i<n; i++)
    {
        tmp1  = get_value(A, table[0][i], N);
        tmp += pow((table[1][i]-tmp1),2);
```

```
        printf("%lf ", tmp1);
    }
    printf("\n(y-F)^2 = %lf", tmp);
    n_point = n; n_graf = 2;
    xfd = malloc(n_point*sizeof(double));
    yfd = malloc(n_graf*sizeof(double*));
    for(int i=0; i<n_graf; i++)
        yfd[i] = malloc(n_point*sizeof(double));
    yfd[0] = table[1];
    for(int i=0; i<n; i++)
        yfd[1][i] = get_value(A, table[0][i], N);
    xfd = table[0];
    drow_graf();
    free(table); free(A); free(xfd); free(yfd);
}
```

Задание 4

1) Постановка задачи: Вычислить первую и вторую производную от таблично заданной функции y_i= f(x), i = 0,1,2,3,4 в точке x = X^*.

Вариант 12:

X^*=0.2

i	0	1	2	3	4
x_i	-1.0	-0.4	0.2	0.6	1.0
y_i	-1.4142	-0.55838	0.27870	0.84008	1.4142

2) Теория: В первом приближении, таблично заданная функция может быть аппроксимирована отрезками прямой $y(x) \approx \varphi(x) = y_i + \frac{y_{i+1} - y_i}{x_{i+1} - x_i}(x - x_i), x \in [x_i, x_{i+1}]$. В этом случае: $y'(x) \approx \varphi'(x) = \frac{y_{i+1} - y_i}{x_{i+1} - x_i} = const, x \in [x_i, x_{i+1}],$.

При использовании для аппроксимации таблично заданной функции интерполяционного многочлена второй степени имеем:

$$y(x) \approx \varphi(x) = y_i + \frac{y_{i+1} - y_i}{x_{i+1} - x_i}(x - x_i) + \frac{\frac{y_{i+2} - y_{i+1}}{x_{i+2} - x_{i+1}} - \frac{y_{i+1} - y_i}{x_{i+1} - x_i}}{x_{i+2} - x_i}(x - x_i)(x - x_{i+1}),\ x \in [x_i, x_{i+1}]$$

$$y'(x) \approx \varphi'(x) = \frac{y_{i+1} - y_i}{x_{i+1} - x_i} + \frac{\frac{y_{i+2} - y_{i+1}}{x_{i+2} - x_{i+1}} - \frac{y_{i+1} - y_i}{x_{i+1} - x_i}}{x_{i+2} - x_i}(2x - x_i - x_{i+1}),\ x \in [x_i, x_{i+1}]$$

Для вычисления второй производной, необходимо использовать интерполяционный многочлен, как минимум второй степени. После дифференцирования многочлена получаем:

$$y''(x) \approx \varphi''(x) = 2\frac{\frac{y_{i+2} - y_{i+1}}{x_{i+2} - x_{i+1}} - \frac{y_{i+1} - y_i}{x_{i+1} - x_i}}{x_{i+2} - x_i},\ \ x \in [x_i, x_{i+1}]$$

3) Результат работы программы:

```
-1.0 -0.4 0.2 0.6 1.0
-1.4142 -0.55838 0.27870 0.84008 1.4142
specify point of derivative: 0.2

y' = 1.395133
y'' = 0.016633
```

4) Код программы

```
double first_derivative(int i, double x[], double y[])
{
    return (y[i+1] - y[i]) / (x[i+1] - x[i]);
}

double second_derivative(int i, double x[], double y[])
{
    double a = (y[i+2] - y[i+1]) / (x[i+2] - x[i+1]);
    double b = (y[i+1] - y[i]) / (x[i+1] - x[i]);
    double c = x[i+2] - x[i];
    return 2 * (a - b) /c;
}

void derivatives()
{
    int n, i;
    double x0;
    printf("specify the number of records: ");
    scanf("%d", &n);
    double **table  = malloc(2*sizeof(double*));
```

```
    printf("enter values from table: \n");
    for(int i=0; i<2; i++)
    {
        table[i] = malloc(n*sizeof(double));
        for (int j=0; j<n; j++)
            scanf("%lf", &table[i][j]);
    }
    printf("specify point of derivative: ");
    scanf("%lf", &x0);
    for(i=0; i<n; i++)
        if ((table[0][i]<=x0) && (x0 <=table[0][i+1]))
            break;
    printf("\ny' = %lf", first_derivative(i, table[0], table[1]));
    printf("\ny" = %lf", second_derivative(i, table[0], table[1]));
    free(table);
}
```

Задание 5

1) Постановка задачи: Вычислить определенный интеграл $F=\int_{X_0}^{X_1} ydx$, методами прямоугольников, трапеций, Симпсона с шагами h_1, h_2. Оценить погрешность вычислений, используя Метод Рунге-Ромберга:

Вариант 12:

$y = \dfrac{x}{x^3+8}$, $\quad X_0 = -1, \quad X_k = 1, \quad h_1 = 0.5, \quad h_2 = 0.25;$

2) Теория: Формула прямоугольников: $\int_a^b f(x)\,dx \approx \sum_{i=1}^{N} h_i\, f\left(\frac{x_{i-1}+x_i}{2}\right)$

Формула трапеций: $F = \int_a^b f(x)\,dx \approx \frac{1}{2}\sum_{i=1}^{N}(f_i + f_{i-1})h_i$

Формула Симпсона: $F = \int_a^b f(x)\,dx \approx \frac{1}{3}\sum_{i=1}^{N}(f_{i-1} + 4f_{i-\frac{1}{2}} + f_i)h_i$

Метод Рунге-Ромберга-Ричардсона позволяет получать более высокий порядок точности вычисления. Если имеются результаты вычисления определенного интеграла на сетке с шагом h - $F = F_h + O(h^p)$ и на сетке с шагом kh - $F = F_{kh} + O((kh)^p)$, то $F = \int_a^b f(x)\,dx = F_h + \frac{F_h - F_{kh}}{k^p - 1} + O(h^{p+1})$

3) Результат работы программы:

indicate step sizes: 0.5 0.25

h = 0.500000: rectangle = -0.005019; trapeze = -0.008913; sympson = -0.006593
h = 0.250000: rectangle = -0.005961; trapeze = -0.006966; sympson = -0.006317

refined rectangle = -0.006276; refined trapeze = -0.006317; refined sympson = -0.006298; betwe h0=0.500000 and h1=0.250000
error: rectangle = 0.000314; trapeze = 0.000649; sympson = 0.000018;

4) Код программы

```
double function_v12_5(double x)
{
   return x/(pow(x,3)+8);
}

double rectangle_method(double(*func)(double x), double h, double x[], int n)
{
   double sum=0;
   for(int i=1; i<n; i++)
      sum += func((x[i]+x[i-1])/2);
   return h*sum;
}

double trapeze_method(double(*func)(double x), double h, double x[], int n)
{
   double sum = (func(x[0]) + func(x[n-1])) / 2;
   for(int i=1; i<n-1; i++)
   {
      sum += func(x[i]);
   }
   return h*sum;
}

double sympson_method(double(*func)(double x), double h, double x[], int n)
{
   double sum = func(x[0]) + func(x[n-1]);
   for(int i=1; i<n-1; i++)
   {
      if(i%2==0)
         sum += 2*func(x[i]);
      else
         sum += 4*func(x[i]);
   }
   return h*sum/3;
}

double runge_romberg_method(double Fh, double Fkh, double p)
{
   return Fh + ((Fh - Fkh) / (pow(2,p) - 1));
}

void rectangle_trapeze_sympson_rrm()
{
   int n;
   double k0, k1;
   printf("\nenter the number of steps to consider: ");
   scanf("%d", &n);
   printf("\nindicate the limits of integration: ");
   scanf("%lf %lf", &k0, &k1);
   double *h = malloc(n*sizeof(double));
   double *rectangle = malloc(n*sizeof(double));
   double *trapeze = malloc(n*sizeof(double));
```

```
    double *sympson = malloc(n*sizeof(double));
    double **rrm = malloc((n-1)*sizeof(double));
    printf("\nindicate step sizes: ");
    for(int i=0; i<n; i++)
        scanf("%lf", &h[i]);
    for(int i=0; i<n; i++)
    {
        int N = (int)((k1-k0)/h[i]+1);
        double *x = malloc(N*sizeof(double));
        x[0] = k0;
        for(int j = 1; j<N; j++)
            x[j] = x[j-1] + h[i];
        rectangle[i] = rectangle_method(function_v12_5, h[i], x, N);
        trapeze[i] = trapeze_method(function_v12_5, h[i], x, N);
        sympson[i] = sympson_method(function_v12_5, h[i], x, N);
        printf("\nh = %lf: rectangle = %lf; trapeze = %lf; sympson = %lf",h[i], rectangle[i], trapeze[i], sympson[i]);
    }
    printf("\n");
    for(int i=1; i<n; i++)
    {
        rrm[i-1] = malloc(3*sizeof(double));
        rrm[i-1][0] = runge_romberg_method(rectangle[i], rectangle[i-1], 2);
        rrm[i-1][1] = runge_romberg_method(trapeze[i], trapeze[i-1], 2);
        rrm[i-1][2] = runge_romberg_method(sympson[i], sympson[i-1], 4);
         printf("\nrefined rectangle = %lf; refined trapeze = %lf; refined sympson = %lf; betwe h%d=%lf and h%d=%lf",
rrm[i-1][0], rrm[i-1][1], rrm[i-1][2], i-1, h[i-1], i, h[i]);
         printf("\nerror: rectangle = %lf; trapeze = %lf;  sympson = %lf;\n", fabs(rectangle[i] - rrm[i-1][0]), fabs(trapeze[i]
- rrm[i-1][1]), fabs(sympson[i] - rrm[i-1][2]));
    }
    free(h); free(rectangle); free(trapeze); free(sympson); free(rrm);
}
```