**Федеральное государственное бюджетное образовательное учреждение «Сибирский государственный университет телекоммуникаций и информатики»**

Кафедра ПМ и К

**КУРСОВАЯ РАБОТА**

По дисциплине «Вычислительная математика»

Вариант 6

**Выполнил**:

студент гр. ИВ-621

Дьяченко Д.В.

**Проверил**:

Чирихин К. С.

Новосибирск, 2018

1. Постановка задачи 3
2. Описание алгоритма 4
3. Результат работы программы 5
4. Заключение 7
5. Листинг 7

**Постановка задачи**

Решить краевую задачу методом Рунге-Кутта II порядка с усреднением по производной.

Построить графики функции y(x) и кубического сплайна S(x) (интерполяция по точкам x=0; 0.2;0.4; 0.6; 0.8; 1.0). Найти интеграл

**Описание алгоритма**

Этапы решения краевой задачи:

1. С помощью метода стрельбы находим значение первой производной.

2. Решаем задачу Коши методом Рунге-Кутта II порядка с усреднением по времени.

Метод стрельбы:

Выбираем параметры: a ─ y(0) из краевой задачи, а b ─ произвольно; для того, чтобы найти отрезок, в котором будет искомое значение; корректируем исходные параметры в зависимости от перелёта или недолёта (y(a) – y1 > 0 или y(a) – y1 < 0 соответственно). Как только по a перелет, а по b недолет, останавливаем корректировку, на данном этапе отрезок найден. Для нахождения первой производной остается решить нелинейное уравнение любым известным способом, в частности методом бисекции: y(b) = y2, где y(b) – решение задачи Коши.

Задача Коши:

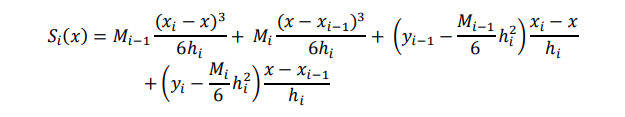
Применяем метод Рунге-Кутта:



Так как по условию дано уравнение, которое не может быть разрешено относительно старшей производной, то каждый раз решаем нелинейное уравнение относительно старшей производной.

Интерполяция:

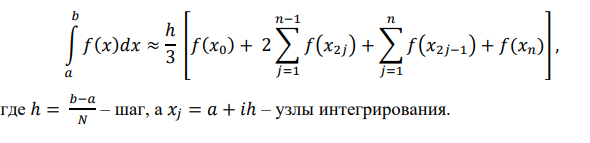
Для вычисления кубического сплайна на заданной сетке будем использовать формулу:



В данной задаче подразумевается интерполяция естественным кубическим сплайном, т. е. 𝑀0 = 𝑀𝑛 . Шаг для сетки: h = const. Чтобы найти другие 𝑀𝑖 составим СЛАУ; получим трехдиагональную матрицу, решаем систему методом прогонки и вычисляем значения сплайна в текущей точке.

Вычисление интеграла:

Численное интегрирование по формуле Симпсона; отрезок [a, b] разбивается на N = 2n частей:



**Результат работы программы**

Решение краевой задачи:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| i | 0 | 1 | 2 | 3 | 4 | 5 |
|  | 0.0 | 0.2 | 0.4 | 0.6 | 0.8 | 1.0 |
|  | 3.000 | 2.660 | 2.395 | 2.199 | 2.069 | 2.003 |
|  | -1.907 | -1.507 | -1.149 | -0.813 | -0.490 | -0.176 |

Интерполяция:

Рисунок Интерполяция Кубическим Сплайном

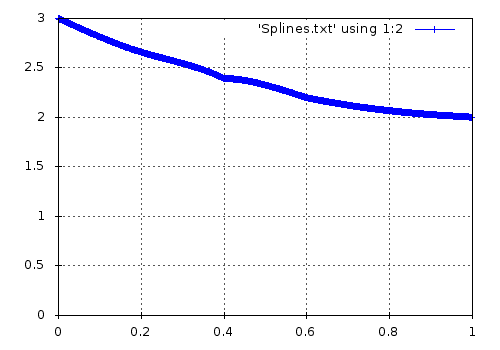


Рисунок Результат функции Рунге-Кутты для y(x)

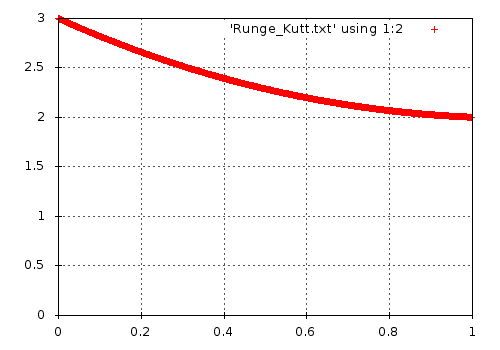
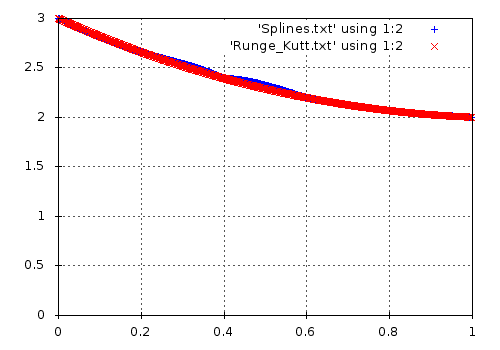


Рисунок Сравнение результата функции Рунге-Кутты для y(x) и интерполяции Кубическим Сплайном



Численное интегрирование:

**Заключение**

В рамках курсовой работы была решена краевая задача, результаты которой удовлетворяют заданным граничным условиям в концах интервала. Проведена интерполяция кубическими сплайнами, построен график сеточной функции, который иллюстрирует решение дифференциального уравнения. По формуле Симпсона вычислено приближенное значение интеграла для заданной подынтегральной функции.

**Листинг**

*main.c*

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <math.h>  double eps = 1e-4;  double diff(double x, double y, double D1, double D2)  {  if (x == 0) {  x = 0.0001;  }  return pow(D2, 5) - cos(x) \* D2 - sin(x) - 5 \* log(x) \* D1 - y \* (x + 3);  }  double \*addition\_of\_vectors(double \*v1, double \*v2, int n)  {  double \*v3 = malloc(sizeof(double) \* n);  for (int i = 0; i < n; i++) {  v3[i] = v1[i] + v2[i];  }  return v3;  }  double \*multiple\_dig\_by\_vector(double a, double \*v, int n)  {  double \*v2 = malloc(sizeof(double) \* n);  for (int i = 0; i < n; i++) {  v2[i] = a \* v[i];  }  return v2;  }  double \*f(double x, double \*y)  {  double a = 0, b = 2;  double fa = 0, fb = 0;  do {  fa = diff(x, y[0], y[1], a--);  fb = diff(x, y[0], y[1], b++);  } while (fa \* fb > 0);  double c = 0;  while(fabs(b - a) >= eps) {  c = (a + b) / 2;  if(diff(x, y[0], y[1], a) \* diff(x, y[0], y[1], c) < 0)  b = c;  else if (diff(x, y[0], y[1], c) \* diff(x, y[0], y[1], b) < 0)  a = c;  }  double \*tmp\_y = malloc(sizeof(double) \* 2);  tmp\_y[0] = y[1];  tmp\_y[1] = (a + b) / 2;  return tmp\_y;  }  double \*Runge\_Kutt(double a, double b, double h, double \*y0)  {  double \*tmp\_y;  double \*y = y0;  for (double i = a + h; i <= b; i += h) {  tmp\_y = addition\_of\_vectors(y, multiple\_dig\_by\_vector(h, f(i, y), 2), 2);  y = addition\_of\_vectors(y, multiple\_dig\_by\_vector(h / 2, addition\_of\_vectors(f(i, y), f(i + h, tmp\_y), 2), 2), 2);  }  return y;  }  double MethodShooting(double x0, double x1, double y0, double y1, double h)  {  double al = 1.0;  double bt = 0.0;  double fa = 0.0;  double fb = 0.0;  double tmp[2];  double \*vt;  do {  tmp[0] = y0;  tmp[1] = al;  vt = Runge\_Kutt(x0, x1, h, tmp);  fa = vt[0] - y1;  tmp[1] = bt;  vt = Runge\_Kutt(x0, x1, h, tmp);  fb = vt[0] - y1;  al -= h;  bt += h;  } while (fa \* fb > 0);  double c = 0.0;  while (fabs(bt - al) >= eps) {  c = (al + bt) / 2;  tmp[1] = al;  double \*tmp1 = Runge\_Kutt(x0, x1, h, tmp);  tmp[1] = c;  double \*tmp2 = Runge\_Kutt(x0, x1, h, tmp);  tmp[1] = bt;  double \*tmp4 = Runge\_Kutt(x0, x1, h, tmp);  if ((((tmp1[0] - y1) \* (tmp2[0] - y1)) < 0)) {  bt = c;  } else if ((((tmp2[0] - y1) \* (tmp4[0] - y1)) < 0)) {  al = c;  }  }  return (al + bt) / 2;  }  void set\_h(double \*h, double \*X, int n)  {  for (int i = 1; i < n; i++)  h[i] = X[i] - X[i - 1];  }  void set\_d(double \*d, double \*h, double \*Y, int n)  {  for (int i = 1; i < n - 1; i++)  d[i] = ((Y[i + 1] - Y[i]) / h[i + 1]) - ((Y[i] - Y[i - 1]) / h[i]);  }  void set\_C(double \*C, double \*h, int n)  {  for (int i = 1; i < n - 1; i++) {  for (int j = 1; j < n - 1; j++) {  if (i == j) {  C[i \* n + j] = (h[i] + h[i + 1]) / 3;  } else if (j == i + 1) {  C[i \* n + j] = h[i + 1] / 6;  } else if (j == i - 1) {  C[i \* n + j] = h[i] / 6;  } else {  C[i \* n + j] = 0;  }  }  }  }  int Matrix\_max\_first\_elem(double \*a, int j, int n)  {  double num = 0;  int num\_ind = 0;  for (int z = j; z < n; z++) {  if (fabsf(a[z \* (n + 1) + j]) > num) {  num = a[z \* (n + 1) + j];  num\_ind = z;  }  }  return num\_ind;  }  void Matrix\_swap\_line(double \*a, int j, int k, int n)  {  double buf;  for (int i = 1; i < n + 1; i++) {  buf = a[j \* (n + 1) + i];  a[j \* (n + 1) + i] = a[k \* (n + 1) + i];  a[k \* (n + 1) + i] = buf;  }  }  void Matrix\_answer(double \*arr\_arg, double \*a, int n)  {  for (int i = n - 1; i > 0; i--) {  for (int j = n - 1; j != i; j--) {  a[i \* (n + 1) + n] = a[i \* (n + 1) + n] - (a[i \* (n + 1) + j] \* arr\_arg[j]);  a[i \* (n + 1) + j] = 0;  }  arr\_arg[i] = a[i \* (n + 1) + n] / a[i \* (n + 1) + i];  }  }  void set\_M(double \*M, double \*C, double \*d, int n)  {  double \*arr = malloc(sizeof(double) \* n \* (n + 1));  for (int i = 1; i < n - 1; i++) {  for (int j = 1; j < n; j++) {  arr[i \* n + j] = C[i \* n + j];  }  arr[i \* n + (n - 1)] = d[i];  }  double mult;  for (int j = 1; j < n - 2; j++) {  int num\_ind = Matrix\_max\_first\_elem(arr, j, n - 1);  Matrix\_swap\_line(arr, j, num\_ind, n - 1);  for (int i = j + 1; i < (n - 1); i++) {  if (arr[i \* (n - 1) + j] != 0) {  mult = -(arr[i \* (n - 1) + j] / arr[j \* (n - 1) + i]);  } else {  break;  }  for (int k = j; k < n; k++) {  arr[i \* (n - 1) + k] += mult \* arr[j \* (n - 1) + k];  }  }  }  Matrix\_answer(M, arr, (n - 1));  }  int set\_i(double \*X, double x, int n)  {  int i = 0;  for (int j = 1; j < n; j++) {  if (X[j - 1] <= x && x <= X[j])  i = j;  }  return i;  }  double set\_s(double \*X, double \*Y, double x, double \*h, double \*M, int i)  {  double s = M[i - 1] \* (pow((X[i] - x), 3) / (6 \* h[i]));  s += M[i] \* (pow((x - X[i - 1]), 3) / (6 \* h[i]));  s += (Y[i - 1] - ((M[i - 1] \* pow(h[i], 2)) / 6)) \* ((X[i] - x) / h[i]);  s += (Y[i] - ((M[i] \* pow(h[i], 2)) / 6)) \* ((x - X[i - 1]) / h[i]);  return s;  }  double Splines(double \*X, double \*Y, double x, int n)  {  double \*h = malloc(sizeof(double) \* n);  double \*d = malloc(sizeof(double) \* (n - 1));  double \*C = malloc(sizeof(double) \* n \* n);  double \*M = malloc(sizeof(double) \* n);  set\_h(h, X, n);  set\_d(d, h, Y, n);  set\_C(C, h, n);  set\_M(M, C, d, n);  int i = set\_i(X, x, n);  double s = set\_s(X, Y, x, h, M, i);  free(M);  free(C);  free(d);  free(h);  return s;  }  double Form\_of\_Simpson(double a, double b, double h, double \*y0)  {  double res = 0.0;  double j = a;  double \*tmp;  for (int i = 1; j <= b - h; i++, j += h) {  tmp = Runge\_Kutt(a, j, h, y0);  res += (i % 2 ? 4 : 2) \* tmp[0];  }  tmp = Runge\_Kutt(a, 0, h, y0);  res += tmp[0];  tmp = Runge\_Kutt(a, b, h, y0);  res += tmp[0];  res = (res \* h) / 3;  return res;  }  double double\_counting(double (\*method)(double, double, double, double \*), double a, double b, double h, double Eps, double \*y)  {  h = b - a;  double prev = method(a, b, h, y);  h /= 2;  double cur = method(a, b, h, y);  int count = 0;  while (fabs(prev - cur) >= Eps) {  prev = cur;  h /= 2;  cur = method(a, b, h, y);  count++;  }  printf("Count iteration = %d\n", count);  return cur;  }  double \*\*dbl\_counting\_Runge(double a, double b, double h, double Eps, double \*y, int \*count, double \*h\_)  {  double \*\*y\_prev;  double \*\*y\_cur;  double max;  int count\_elem;  do {  count\_elem = (b - a) / h;  y\_prev = malloc(sizeof(double\*) \* count\_elem);  double t = a;  for (int i = 0; fabs(t - b) >= 1e-8; i++, t += h) {  y\_prev[i] = malloc(sizeof(double) \* 2);  y\_prev[i] = Runge\_Kutt(a, t, h, y);  }  h /= 2;  count\_elem = (b - a) / h;  t = a;  y\_cur = malloc(sizeof(double\*) \* count\_elem);  for (int i = 0; fabs(t - b) >= 1e-8; i++, t += h) {  y\_cur[i] = malloc(sizeof(double) \* 2);  y\_cur[i] = Runge\_Kutt(a, t, h, y);  }  double mod = 0.0;  max = 0;  int j = 0;  int del = count\_elem;  int i;  for (i = 0; j < del + (del % 2 ? 1 : 0); i++, j += 2) {  mod = fabs(y\_prev[i][0] - y\_cur[j][0]);  if (mod > max) {  max = mod;  }  }  } while (fabs(max) >= Eps);  \*count = count\_elem;  \*h\_ = h;  for (int i = 0, j = 0; i < count\_elem; i += 2, j++) {  y\_cur[i][0] -= (y\_cur[i][0] - y\_prev[j][0]) / 3;  y\_cur[i][1] -= (y\_cur[i][1] - y\_prev[j][1]) / 3;  }  return y\_cur;  }  double dbl\_count\_D1(double x0, double x1, double y0, double y1, double h, double Eps)  {  double prev;  double cur;  do {  prev = MethodShooting(x0, x1, y0, y1, h);  h /= 2;  cur = MethodShooting(x0, x1, y0, y1, h);  } while (fabs(prev - cur) >= Eps);  return cur;  }  int main()  {  double a = 0.0;  double b = 1.0;  double h = 0.2;  int size = (a + b) / h;  double y[size + 1];  double x0 = 0.0;  double y0 = 3.0;  double x1 = 1.0;  double y1 = 2.0;  double D1 = dbl\_count\_D1(x0, x1, y0, y1, h, 1e-3);  printf("D1 = %.3lf\n", D1);  FILE \*out = fopen("Runge\_Kutt.txt", "w");  printf("x\ty(x)\ty\'(x)\n");  double Eps = 1e-3;  double tmp[2] = { y0, D1 };  int count\_elem;  double h\_;  double \*\*yt = dbl\_counting\_Runge(a, b, h, Eps, tmp, &count\_elem, &h\_);  int i\_count[6];  i\_count[0] = 0;  double x[6] = { 0.0, 0.2, 0.4, 0.6, 0.8, 1.0 };  int t = count\_elem / 5;  for (int j = 1; j < 6; j++) {  i\_count[j] = t \* j;  }  i\_count[5]--;  double ta = a;  for (int i = 0; i < count\_elem; i++, ta += h\_) {  fprintf(out, "%.4lf %lf\n", ta, yt[i][0]);  }  double m = 0.0;  for (int i = 0; i < 6; i++, m += h) {  printf("%.2lf\t", m);  printf("%.3lf\t", yt[i\_count[i]][0]);  printf("%.3lf\n", yt[i\_count[i]][1]);  y[i] = yt[i\_count[i]][0];  }  printf("\n");  fclose(out);  printf("Spleins interpolation:\n");  FILE \*splines\_out = fopen("Splines.txt", "w");  for (double i = a; i <= b; i += h\_) {  double tmp = Splines(x, y, i, 6);  fprintf(splines\_out, "%lf %lf\n", i, tmp);  }  printf("\n");  fclose(splines\_out);  Eps = 1e-2;  printf("Integration = %.10lf\n", double\_counting(Form\_of\_Simpson, a, b, h, Eps, tmp));  return 0;  } |