Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет

телекоммуникаций и информатики»

Кафедра ПМиК

Курсовая работа по дисциплине

Вычислительная математика

Вариант 14

Выполнил: студент 2 курса

Ф. ИВТ, группа: ИП-711

Мартасов И. О.

Проверила: доцент кафедры ПМиК

Рубан А. А.

Новосибирск, 2019

**Содержание**

1. Постановка задачи …..........................................................................................3

2. Основные идеи используемых методов…… ...................................................4

3. Код программы..............…................................................................................12

4.Результаты работы……………………………………………………………29

**Постановка задачи**

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

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

**Основные идеи используемых методов**

**1.Простейшая модификация метода Эйлера – метод Рунге-Кутта 2-го порядка.**

Заменим приращение функции на первом шаге  не на , как делали в методе Эйлера, а на более точное значение - на значение производной в середине интервала . А для того, чтобы найти .

Заменим 

Окончательно получаем следующую формулу:



Формула Рунге-Кутта 2го порядка с усреднением по

времени.

**2.Краевые задачи для дифференциальных уравнений**

Для ДУ высших порядков часто бывает необходимо решить не задачу Коши, а так называемую краевую задачу, т.е. начальные условия, которые заданы в разных точках.

Рассмотрим простейшую краевую задачу для ДУ 2го порядка:

 (6.10)

А мы умеем решать:

(6.11)

В (6.11) нам известно , поэтому для решения задачи (6.10) мы будем подбирать  в (6.11), с тем, чтобы у(b) = у1

**2.1.Метод стрельб**

После пристрелки и определения интервала [a,b],

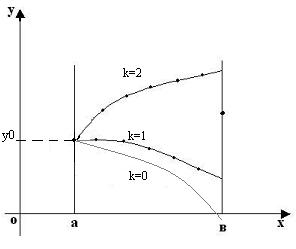
где идёт смена знака, запускаем МПД или МХ.

На практике это выглядит так, как будто мы

решаем уравнение , где возвращает

решение задачи Коши (6.11) в точке b при

заданном k.



**2.2.Что делать, если ДУ не может быть разрешено относительно старшей производной?**

Так как ДУ не может быть решено относительно старшей производной, то тогда на каждом шаге решаем нелинейное уравнение относительно y(n)(все остальные неизвестные y,y’,y”,…, y(n-1)-к этому моменту уже известны).

Решать уравнение относительно старшей производной любым методом(Хорд, МПД, Ньютона).

**3.Интерполяция кубическими сплайнами.**

**3.1. Определение кубического Сплайна.**

Кубическим сплайном на сетке x0,x1,…xn называется функция S(х), которая обладает следующими свойствами:

1. на каждом интервале [хi-1, хi], где 1 ≤ i ≤ n, функция S(х) является кубическим многочленом (на каждом интервале свой многочлен).
2. на всем интервале [х0, хп] S(х) – дважды непрерывно дифференцируемая функция
3. на краях интервала вторая производная обращается в ноль (краевое условие).

S΄΄(x0)=S΄΄(xn)=0

3’. для периодических кубических сплайнов.

S΄΄(x0)=S΄΄(xn)=0 ; S΄(x0)=S΄(xn)=0

Исследуем вопрос: любую ли функцию можно проинтерполировать кубическим сплайном и всегда ли это можно сделать единственным образом?

Имеем n участков интерполяции, на каждом – свой кубический многочлен, который задается четырьмя коэффициентами. Итого, имеем 4n коэффициентов, которые нам необходимо найти, для этого нам потребуется столько же уравнений (т.е. 4n. уравнений).

Исходя из условий кубического сплайна:

(подсчет уравнений, которых нам дают условия кубического сплайна)

n участков [хi-1, хi], на границах должны выполнятся условия интерполяции  ;  - на каждом участке 2 условия, итого получаем 2n условий.

Вспомним второе условие кубического сплайна, т.е. наша функция дважды непрерывно дифференцируема. Внутри участков это, очевидно, выполняется (т.к. - кубический многочлен). Необходимо проверить непрерывность S, S’ и S” только лишь на границах интервалов, т.е. рассмотрим точку  - в ней стыкуются два интервала: [хi-1, хi] и [хi, хi+1]

соответственно кубические сплайны:  и 

Предел слева должен быть равен пределу справа для S, S’ и S”, т.е.

 - не пишем т.к. оно уже было посчитано в условии интерполяции.





+ два условия из пункта 3. Итого, 4n условий.

**3.1. Свойства кубического Сплайна**

**Теорема 4.5:**

Среди всех функций, интерполирующих функцию f в точках хi, где  именно кубический сплайн обладает наименьшей энергией изгиба, т.е. для него достигается минимум интеграла энергии. - интеграл энергии.

***Следствие 4.6***

Из математического анализа известно, что радиус кривизны функции у(х):  (k(x) – кривизна изгиба). Как известно из физики, энергия изгиба гибкой линейки, принявшей очертание графика функций y(x), вычисляется по формуле: 

- коэффициент жесткости линейки (предположим y’0)

Таким образом, энергия изгиба линейки .

Как мы знаем из физики, любая физическая система, в том числе и линейка, стремится минимизировать свою энергию, следовательно, гибкая линейка, пропущенная через точки (хi, уi) , (теорема 4.5) примет очертание кубического сплайна. Отсюда и происходит само слово сплайн (spline – рейка, которую используют чертежники).

Очевидно, что кривизна линейки есть функция непрерывная, следовательно, S, S’ и S” непрерывны – это условие 2 из определения кубического сплайна. Также понятно, что на краях кривизна линейки будет нулевая – отсюда берется условие 3.

**3.2. Формулы для вычисления кубического сплайна.**

С одной стороны мы можем составить 4n уравнений для 4n коэффициента кубического многочлена (см. пункт 3.1).

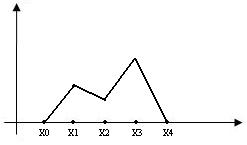
На практике подобный подход не используется, выгоднее идти другим путём (уравнений и неизвестных будет меньше).

**Другой вариант вычисления кубического сплайна.**

Введём следующие моменты: S΄΄(xi)=Mi , с помощью их и будем вычислять кубический сплайн.

 из 3-го условия.

Т.к. S(х) кусочно-кубический многочлен, то S”(х) – кусочно-линейная функция, которая при этом непрерывна.

  
Очевидно, что на i-ом участке 

 (4.14)

 - длина i-ого интервала.

Чтобы получить Si(x) проинтегрируем Si’’(x) дважды:



Осталось только подставить константы интегрирования -  и . Для этого необходимо вспомнить условия интерполяции на краях участков.



Подставив эти условия в формулу для , получим 2 уравнения для констант  и , решив систему, подставим в формулу и получим:

 (4.15)

Теперь для расчета кубического сплайна необходимо найти неизвестные моменты Mi.

Мы уже знаем , остается найти моменты . Для этого необходимо ограничения 1,2,3, налагаемые на кубический сплайн.

Условие интерполяции использовали при нахождении констант  и (оно же непрерывность S). Непрерывность S” мы тоже уже использовали, когда писали формулы для кусочно-линейной функции S”. Остается использовать условие непрерывности S’, т.е.

  (4.16)

Получим (n-1) недостающее уравнение для (n-1) неизвестного (для ).

Как нетрудно убедиться, эти условия превращаются в СЛАУ (4.17) для нахождения М:

CM=d (4.17), где



- столбец неизвестных.

- квадратная трёх диагональная матрица.

Элементы матрицы С вычисляются по формуле:



(главная диагональ)

(верхняя диагональ)

(нижняя диагональ)

 , () - вектор правых частей.

Таким образом, для вычисления кубического сплайна необходимо:

1. Составить СЛАУ по формуле (4.17) (размером (n-1)x(n-1))

2. Решить эту СЛАУ, находя моменты , добавить к ним .

3. Найдя моменты , подставить их в формулу (4.15) для нахождения кубического сплайна (перед этим нужно найти i-номер интервала, в котором лежит точка х, т.е. ).

4. **Общие формулы трапеции численного интегрирования**

Если требуется найти на большом промежутке, то мы разбиваем этот интервал на множество меньших интервалов, на каждом из которых применяем соответствующую формулу Ньютона – Котеса (трапеции или Симпсона).

**Общая формула трапеций.**



**Код программы**

#include <iostream>

#include <math.h>

#include <graphics.h>

#include <iomanip>

#define EPS 1e-9

#define A 0

#define B 1

#define YA 4

#define YB 10

#define H 0.05

using namespace std;

double f(double y0, double y1, double y2, double x)

{

return (pow(y2, 3) + 8 \* cos(x) \* y2 - 4 \* x - 7 \* pow(M\_E, x) \* y1 + 5 \* y0 \* (x + 8));

}

double MPD(double y0, double y1, double x)

{

double L = A;

double R = B;

double C = 0;

while(fabs(((R - L) / 2)) >= EPS)

{

C = (L + R) / 2;

if(f(y0, y1, C, x) \* f(y0, y1, L, x) < 0)

{

R = C;

}

else if(f(y0, y1, C, x) \* f(y0, y1, R, x) < 0)

{

L = C;

}

else

{

break;

}

}

return C;

}

double \* rk(double x0, double xn, double y0, double y1, double h)

{

double \* y = new double[2];

double \* yz = new double[2];

double \* Y = new double[2];

y[0] = y0;

y[1] = y1;

while(fabs(xn - x0) >= EPS)

{

Y[0] = y[1];

Y[1] = MPD(y[0], y[1], x0);

//Y[1] = f(y[0], y[1], x0);

for(int i = 0; i < 2; i++)

{

yz[i] = y[i] + Y[i] \* h / 2;

}

Y[0] = yz[1];

Y[1] = MPD(yz[0], yz[1], x0 + h / 2);

//Y[1] = f(yz[0], yz[1], x0 + h / 2);

for(int i = 0; i < 2; i++)

{

y[i] += Y[i] \* h;

}

x0 += h;

}

return y;

}

double \* count(double x0, double xn, double y0, double y1)

{

double \* Y1 = new double[2];

double \* Y2 = new double[2];

int k = 1;

double sum1, sum2;

do

{

Y1 = rk(x0, xn, y0, y1, H / k);

Y2 = rk(x0, xn, y0, y1, H / (k \* 2));

sum1 = Y1[0] - Y2[0];

sum2 = Y1[1] - Y2[1];

k \*= 2;

}while((fabs(sum1) > 3 \* EPS) || (fabs(sum2) > 3 \* EPS));

return Y2;

}

double shooting()

{

double L = 0;

double R = 10;

double C = 0;

double \* y1 = new double[2];

double \* y2 = new double[2];

double \* y3 = new double[2];

/\*int i = 0;

double \* z1 = new double[2];

int isLeft = 0;

int isRight = 0;

do

{

z1 = count(A, B, YA, i);

if(z1[0] > YB)

{

R = z1[0];

isRight++;

}

else if(z1[0] < YB)

{

L = z1[0];

isLeft++;

}

cout << L << endl << R << endl;

i++;

}while((isLeft == 0) || (isRight == 0));\*/

do

{

C = (L + R) / 2;

y1 = count(A, B, YA, L);

y2 = count(A, B, YA, R);

y3 = count(A, B, YA, C);

if((y3[0] - YB) \* (y1[0] - YB) < 0)

{

R = C;

}

else if((y3[0] - YB) \* (y2[0] - YB) < 0)

{

L = C;

}

else

{

break;

}

}while(fabs((y1[0] - y2[0])/ 2) >= EPS);

return C;

}

void swap(double\*\* a, int n, int p1, int p2)

{

double tmp;

for(int i = p2; i <= n; i++)

{

tmp = a[p1][i];

a[p1][i] = a[p2][i];

a[p2][i] = tmp;

}

}

int gauss(double\*\* a, int n)

{

double max;

double buf;

double buf2;

int max\_row;

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

{

max = abs(a[i][i]);

max\_row = i;

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

{

if(max < abs(a[j][i]))

{

max = abs(a[j][i]);

max\_row = j;

}

}

swap(a, n, max\_row, i);

if(a[i][i] == 0)

{

cout << "Система не имеет решений" << endl;

return 0;

}

buf2 = a[i][i];

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

{

buf = -1 \* (a[j][i] / a[i][i]);

for(int k = i; k <= n; k++)

{

if(i == k)

{

a[j][k] = 0;

}

else

{

a[j][k] += buf \* a[i][k];

}

}

}

for(int j = i; j <= n; j++)

{

a[i][j] /= buf2;

}

}

}

int gauss2(double\*\* a, double\* b, int n)

{

for (int i = n - 1; i >= 0; i--)

{

if(a[i][i] == 0)

{

cout << "Система не имеет решений" << endl;

return 0;

}

b[i + 1] = a[i][n] / a[i][i];

for (int k = i - 1; k >= 0; k--)

{

a[k][n] -= a[k][i] \* b[i + 1];

}

}

}

double Spline(double xi, double \* m, double \* x, double \* y, double \* h, int n)

{

double sum = 0;

double sum1, sum2, sum3, sum4;

int k = 0;

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

{

if((xi > x[i]) && (xi < x[i + 1]))

{

k = i + 1;

}

}

if(xi == x[0])

{

k = 1;

}

if(xi == x[1])

{

k = 1;

}

if(xi == x[2])

{

k = 2;

}

if(xi == x[3])

{

k = 3;

}

if(xi == x[4])

{

k = 4;

}

if(xi == x[5])

{

k = 5;

}

sum1 = m[k - 1] \* (pow((x[k] - xi), 3) / (6 \* h[k - 1]));

sum2 = m[k] \* (pow((xi - x[k - 1]), 3) / (6 \* h[k - 1]));

sum3 = (y[k - 1] - ((m[k - 1] \* pow(h[k - 1], 2)) / 6)) \* ((x[k] - xi) / h[k - 1]);

sum4 = (y[k] - ((m[k] \* pow(h[k - 1], 2)) / 6)) \* ((xi - x[k - 1]) / h[k - 1]);

sum += sum1 + sum2 + sum3 + sum4;

return sum;

}

void output(double\*\* a, int n)

{

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

{

for(int j = 0; j <= n; j++)

{

cout << setw(12) << a[i][j];

}

cout << endl;

}

cout << endl;

}

void output2(double\* b, int n)

{

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

{

cout << "M" << i + 1 << " = " << b[i] << endl;

}

cout << endl;

}

double triangle(double K, double h)

{

double sum = 0;

double sum2 = 0;

double n1 = 0;

int n = 0;

n1 = (B - A) / h + 1;

n = n1;

double \* y = new double[n];

double \* y2 = new double[2];

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

{

y2 = count(A, A + i \* h, YA, K);

y[i] = pow(y2[0], 2);

}

for(int i = 1; i < n - 1; i++)

{

sum += y[i];

}

sum2 = 0.5 \* (y[0] + y[n - 1]);

return h \* (sum + sum2);

}

int main()

{

double K = shooting();

double \* y2 = new double[2];

int n = 6;

double hi = 0.2;

int k = 1;

double \* m = new double[n];

double \* x = new double[n];

double \* y = new double[n];

double \* h = new double[n - 1];

double \* d = new double[n - 2];

double \*\* c = new double \* [n - 2];

m[0] = m[n - 1] = 0;

for(int i = 0; i < n - 2; i++)

{

c[i] = new double[n - 1];

}

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

{

x[i] = A + i \* hi;

y2 = count(A, x[i], YA, K);

y[i] = y2[0];

}

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

{

h[i] = x[i + 1] - x[i];

}

for(int i = 0; i < n - 2; i++)

{

d[i] = ((y[i + 2] - y[i + 1]) / h[i + 1]) - ((y[i + 1] - y[i]) / h[i]);

}

for(int i = 0; i < n - 2; i++)

{

c[i][n - 2] = d[i];

for(int j = 0; j <= n - 2; j++)

{

if(i == j)

{

c[i][j] = (h[i] + h[i + 1]) / 3;

}

if(j == i + 1)

{

c[i][j] = h[i + 1] / 6;

}

if(j == i - 1)

{

c[i][j] = h[i] / 6;

}

if(abs(i - j) > 1)

{

c[i][j] = 0;

}

c[i][n - 2] = d[i];

}

}

cout << endl;

gauss(c, n - 2);

gauss2(c, m, n - 2);

for(double i = 0.0; i < 1; i += H)

{

y2 = count(A, i, YA, K);

printf("X=%3.2f Y=%5.8f\tY^=%5.8f\tINT=%5.8f\n", i, y2[0], y2[1], Spline(i, m, x, y, h, n));

}

y2 = count(A, B, YA, K);

printf("X=%3.d Y=%5.8f\tY^=%5.8f\tINT=%5.8f\n", B, y2[0], y2[1], Spline(B, m, x, y, h, n));

cout << "Интеграл - " << triangle(K, hi) << endl;

int height = 1366;

int weight = 768;

int scale = 50;

initwindow(height, weight);

setcolor(GREEN);

for(double i = 0.0; i <= 1; i += 0.2)

{

y2 = count(A, i, YA, K);

circle(height / 2 + i \* scale, weight - y2[0] \* scale, 4);

}

moveto(height / 2, weight / 2);

setcolor(BLACK);

for(double i = 0.0; i < 2 + H; i += H)

{

y2 = count(A, i, YA, K);

lineto(height / 2 + i \* scale, weight - y2[0] \* scale);

setcolor(WHITE);

}

moveto(height / 2, weight / 2);

setcolor(BLACK);

for(double i = 0.00; i < x[n - 1]; i += 0.01)

{

lineto(height / 2 + i \* scale, weight - Spline(i, m, x, y, h, n) \* scale);

setcolor(RED);

}

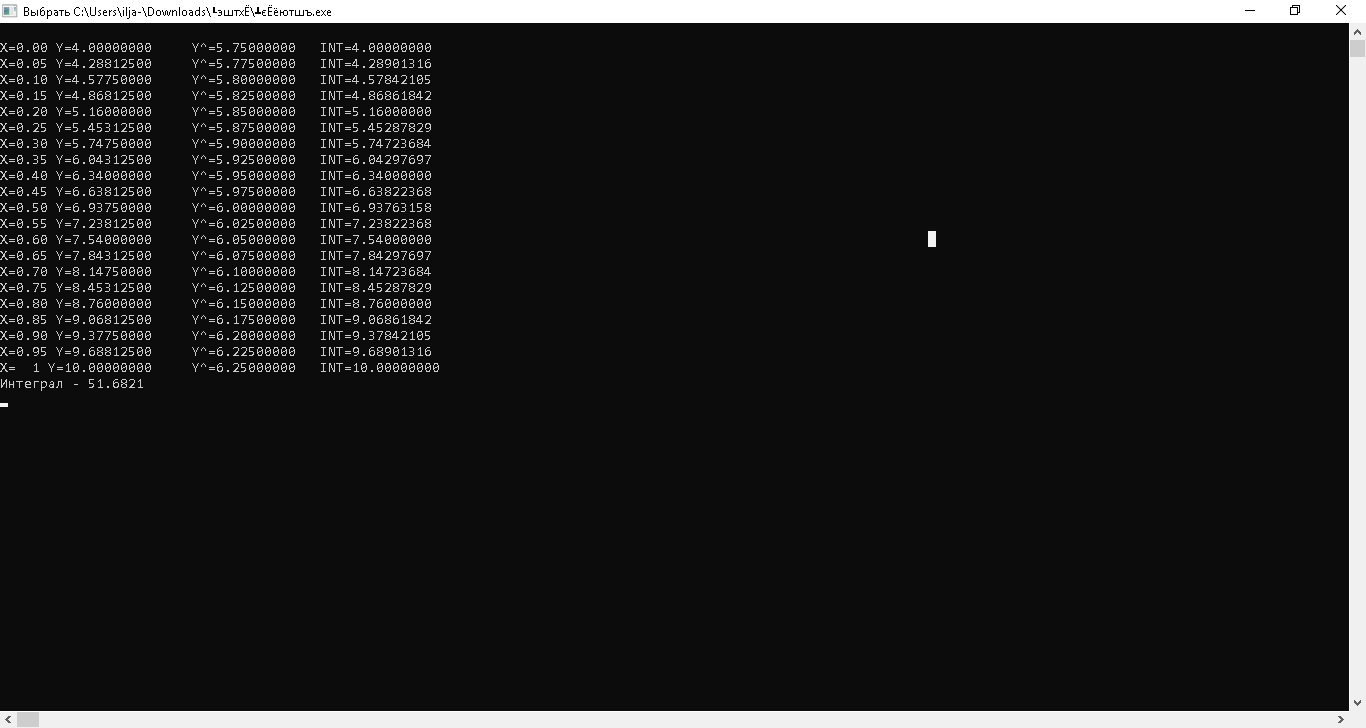
lineto(height / 2 + 1 \* scale, weight - Spline(B, m, x, y, h, n) \* scale);

delay(9999999);

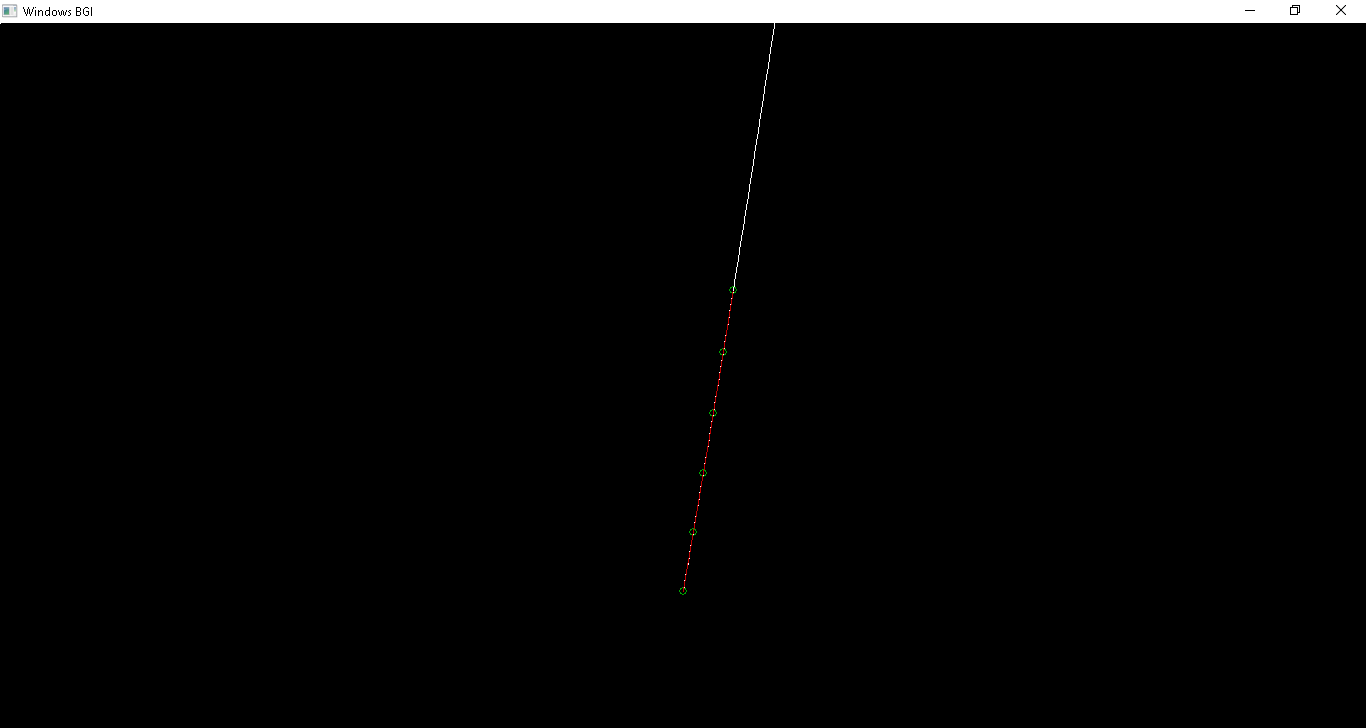
return 0;

}

**Результаты работы**

****

**График кубического сплайна**

****