|  |  |
| --- | --- |
|  | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ Информатики и систем управления

КАФЕДРА Теоретической информатики и компьютерных технологий

**Лабораторная работа № 1**

«Приближение функции кубическими сплайнами»

по курсу «Численные методы»

Выполнил:

студент группы ИУ9-62Б

Марченко Андрей

Проверила:

Домрачева А. Б.

Москва, 2024

1. **Цель**

Целью данной работы является изучение приближения заданной функции путем сплайн-интерполяции, построение сплайна третьего порядка на основе заданной функции и вычисление значения сплайна третьего порядка в серединах частичных отрезков между узлами интерполяции.

1. **Постановка задачи**

**Дано:** функция задана явно уравнением.

**Найти:** интерполяционную функцию (т.е. функцию, совпадающую со значениями в узлах интерполяции ):

1. Для заданных узлов построить кубический сплайн (распечатать массивы a, b, c, d).

2. Вычислить значения в серединах частичных отрезков между узлами интерполяции, т.е. в точках .

3. Посчитать погрешность по сравнению с данной функцией в узлах и между ними.

**Индивидуальный вариант:**

1. **Основные теоретические сведения**

Интерполяционной функцией называется функция , проходящая через заданные точки, называемые узлами интерполяции:

. При этом в промежуточных точках равенство выполняется с некоторой погрешностью . Задача интерполяции заключается в поиске такой функции

Приближение функции кубическим сплайном — пример задачи интерполяции.

Сплайн k-го порядка — функция, проходящая через все узлы , являющаяся многочленом -ой степени на каждом частичном отрезке разбиения и имеющая первые непрерывных на производных.

– дефект сплайна. Чем выше дефект, тем грубее сплайн.

Наиболее употребительны сплайны третьего порядка с дефектом (кубические сплайны).

На каждом частичном отрезке разбиения кубический сплайн описывается

На частные многочлены накладываются условия:

* 1. Сплайн проходит через все узлы
  2. Условие гладкости на краях

; 0

* 1. Непрерывность сплайна и его первых двух производных в промежуточных узлах

Эти условия позволяют выразить коэффициенты и приводят к трехдиагональной СЛАУ относительно коэффициента

СЛАУ с трехдиагональной матрицей относительно коэффициента :

,

где

1. **Реализация**

Листинг 1. Сплайн-интерполяция

#include <iostream>

#include <vector>

#include <tuple>

#include <cmath>

#include <iomanip>

double func(double x) {

     return std::log(x) \* std::log(x) / x;

}

std::vector<double> find\_c(int n, double h, const std::vector<double>& y) {

    std::vector<double> a(n, 1), c(n, 1), b(n, 4), d(n, 0);

    c[n - 1] = 0;

    a[0] = 0;

    for (int i = 0; i < n; ++i) {

        d[i] = 3 \* (y[i + 2] - 2 \* y[i + 1] + y[i]) / (h \* h);

    }

    std::vector<double> alpha(n, 0), beta(n, 0);

    alpha[0] = -c[0] / b[0];

    beta[0] = d[0] / b[0];

    for (int i = 1; i < n; ++i) {

        alpha[i] = -c[i] / (alpha[i - 1] \* a[i - 1] + b[i]);

        beta[i] = (d[i] - a[i] \* beta[i - 1]) / (alpha[i - 1] \* a[i - 1] + b[i]);

    }

    std::vector<double> x(n, 0);

    x[n - 1] = beta[n - 1];

    for (int i = n - 2; i >= 0; --i) {

        x[i] = alpha[i] \* x[i + 1] + beta[i];

    }

    std::vector<double> result = {0};

    result.insert(result.end(), x.begin(), x.end());

    result.push\_back(0);

    return result;

}

std::tuple <std::vector<double>, std::vector<double>, std::vector<double> > find\_a\_b\_d(int n, const std::vector<double>& c, const std::vector<double>& y, double h) {

    std::vector<double> a(y.begin(), y.end() - 1), b(n, 0), d(n, 0);

    for (int i = 0; i < n - 1; ++i) {

        b[i] = (y[i + 1] - y[i]) / h - h \* (c[i + 1] + 2 \* c[i]) / 3;

        d[i] = (c[i + 1] - c[i]) / (3 \* h);

    }

    return {a, b, d};

}

int main() {

    int n = 31;

    std::vector <double> x, y;

    auto a\_dot = 1 / std::numbers::e;

    auto b\_dot = std::numbers::e;

    auto step = (b\_dot - a\_dot) / n;

    for (int i = 0; i <= n; i++){

        x.push\_back(a\_dot + step \* i);

        y.push\_back(func(a\_dot + step \* i));

    }

    for (int i = 0; i < x.size(); i++){

        std::cout << i << ": " << x[i] << " " << y[i] << std::endl;

    }

    std::cout << std::endl;

    double h = (x[n] - x[0]) / n;

    std::vector<double> c = find\_c(n - 1, h, y);

    auto res = find\_a\_b\_d(n, c, y, h);

    auto a = std::get<0>(res);

    auto b = std::get<1>(res);

    auto d = std::get<2>(res);

    std::vector<double> x\_ext;

    std::vector<double> splines;

    for (int i = 0; i < n; ++i) {

        x\_ext.push\_back(x[i]);

        x\_ext.push\_back((x[i] + x[i + 1]) / 2);

    }

    x\_ext.push\_back(x[n]);

    std::cout << "a: ";

    for (const auto& val : a) std::cout << val << " ";

    std::cout << std::endl;

    std::cout << "b: ";

    for (const auto& val : b) std::cout << val << " ";

    std::cout << std::endl;

    std::cout << "c: ";

    for (const auto& val : c) std::cout << val << " ";

    std::cout << std::endl;

    std::cout << "d: ";

    for (const auto& val : d) std::cout << val << " ";

    std::cout << std::endl;

    for (int i = 0; i < x\_ext.size(); ++i) {

        double x\_star = x[0] + 0.5 \* h \* i;

        int j = i / 2;

        if (j == n) {

             j = n - 1;

        }

        splines.push\_back(a[j] + b[j] \* (x\_star - x[j]) + c[j] \* (x\_star - x[j]) \* (x\_star - x[j]) + d[j] \* (x\_star - x[j]) \* (x\_star - x[j]) \* (x\_star - x[j]));

    }

    std::cout << "x\_ext: ";

    for (const auto& val : x\_ext) std::cout << val << " ";

    std::cout << std::endl << std::endl;

    int j = 0;

    for (int i = 0; i < x\_ext.size(); ++i) {

        std::cout << std::fixed << std::setprecision(6) << x\_ext[i] << "\t" << splines[i];

        if (i % 2 == 0) {

            std::cout << "\t" << func(x\_ext[i]) << "\t" << std::abs(splines[i] - func(x\_ext[i])) << "\t -- node";

            j++;

        }

        else {

            std::cout << "\t" << func(x\_ext[i]) << "\t" << std::abs(splines[i] - func(x\_ext[i]));

        }

        std::cout << std::endl;

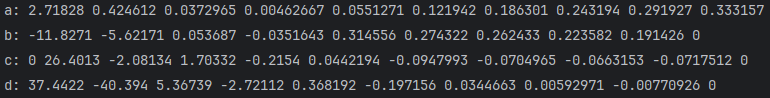
    }

    return 0;

}

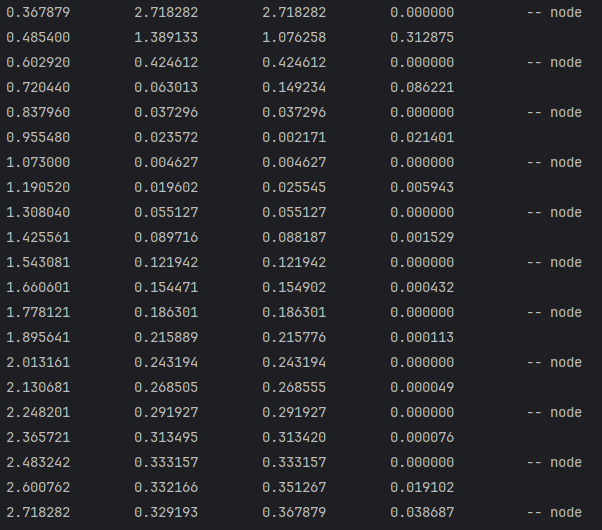
1. **Результаты**

Для заданной функции получены следующие коэффициенты интерполяции и разбиения отрезка на 10 частей:



Далее, в таблице представлены результаты в формате:

x – точка, в которой считается значение – узел или точка между узлами, S(x) – значение интерполяции в этой точке, f(x) – значение исходной функции в этой точке, |f(x)-S(x)| - погрешность вычислений в данной точке.



Значением “—node” отмечены исходные узлы разбиения.

1. **Вывод**

В ходе выполнения лабораторной работы был изучен метод приближения функции с помощью интерполяции кубическим сплайном, был построен сплайн третьего порядка на основе заданных узлов интерполяции и найдены значения функции в серединах частичных отрезков между узлами интерполяции. В результате тестирования был сделан вывод, что даже при 10 разбиениях, имеем хорошую точность приближения, а в узлах интерполяции значения совпадают.