# *Лабораторная работа № 2*

# *«Интерполяционный кубический сплайн»*

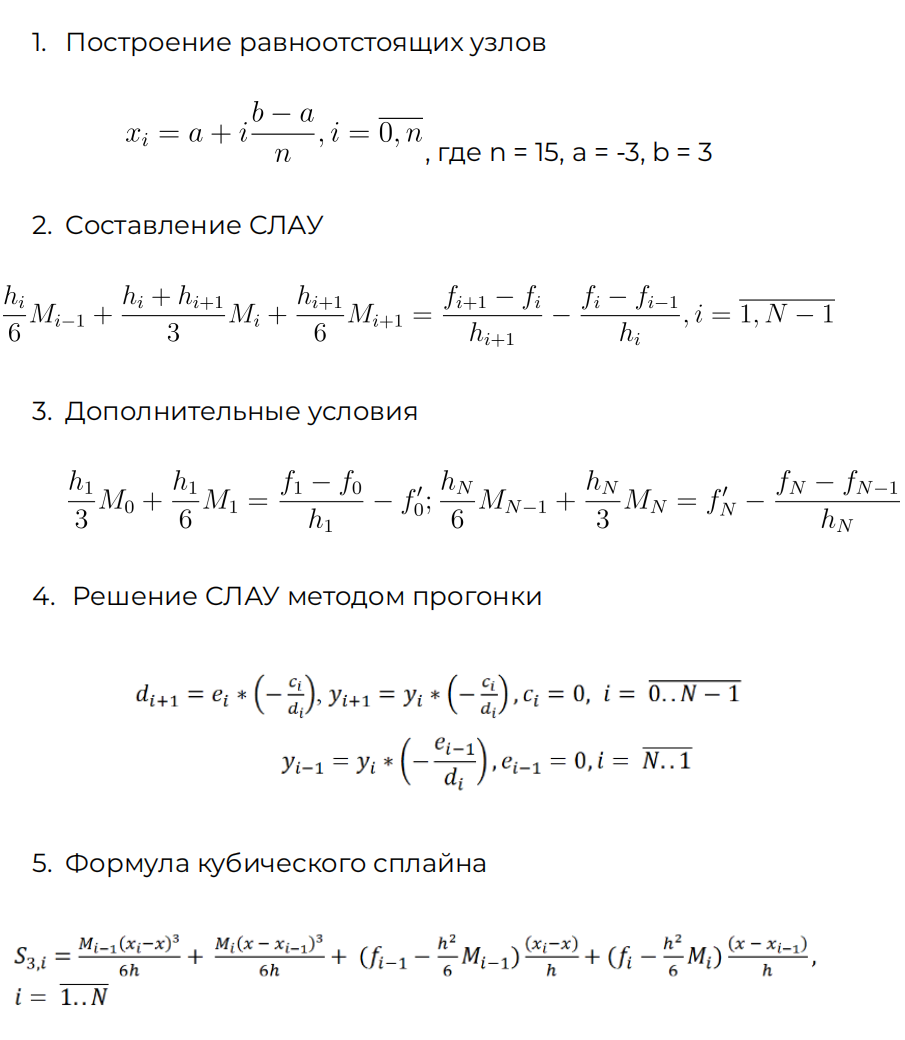
# Вариант 7

# Кохан Даниил 2 курс

7Вариант:

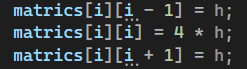
S'(a) = f'(a) и S''(b) = f''(b)

Общие данные и формулы, необходимые для лабораторной работы:

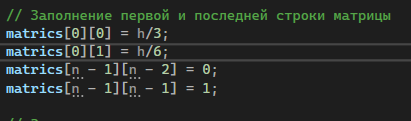


Г-2: MN = F’’(x);

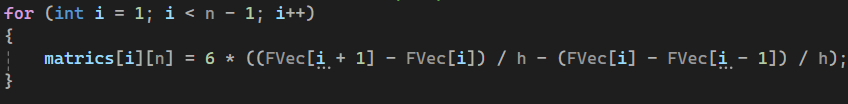
1. Алгоритм построения Кубического сплайна:  
   Построение расширенной трёхдиагональной матрицы по формулам для нахождения моментов (в нашем случае поскольку шаги одинаковые то на главной диагонали 4h на побочках h ). Также стоит учесть дополнительные (в моём случае периодические) ограничения, которые показывают, что что функция и её производные имеют одинаковые значения на обоих концах интервала:



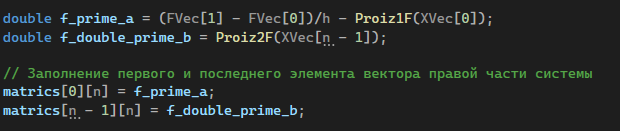
Основные элементы трёх диагоналей

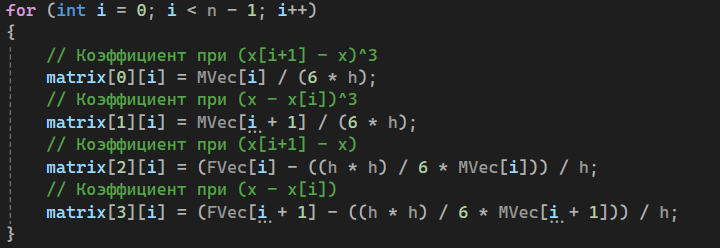


Заполнение первой и последней строки



Заполнение моментов

Заполнение граничных условий



Формула для вывода сплайна

double S = MVec[i] / (6 \* h) \* pow(XVec[i + 1] - x, 3) + MVec[i + 1] / (6 \* h) \* pow(x - XVec[i], 3) + (FVec[i] - ((h \* h) / 6 \* MVec[i])) / h \* (XVec[i + 1] - x) + (FVec[i + 1] - ((h \* h) / 6 \* MVec[i + 1])) / h \* (x - XVec[i]);

Формула для расчета сплайна

vector<double> MomentsCulc(vector<double>& XVec, vector<double>& FVec, int n, double h)

{

// Создание матрицы размером n x (n+1) и заполнение ее нулями

vector<vector<double>> matrics(n, vector<double>(n + 1, 0));

// Заполнение первой и последней строки матрицы

matrics[0][0] = h/3;

matrics[0][1] = h/6;

matrics[n - 1][n - 2] = 0;

matrics[n - 1][n - 1] = 1;

// Заполнение остальных строк матрицы

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

{

matrics[i][i - 1] = h;

matrics[i][i] = 4 \* h;

matrics[i][i + 1] = h;

}

double f\_prime\_a = (FVec[1] - FVec[0])/h - Proiz1F(XVec[0]);

double f\_double\_prime\_b = Proiz2F(XVec[n - 1]);

// Заполнение первого и последнего элемента вектора правой части системы

matrics[0][n] = f\_prime\_a;

matrics[n - 1][n] = f\_double\_prime\_b;

// Заполнение остальных элементов вектора правой части системы

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

{

matrics[i][n] = 6 \* ((FVec[i + 1] - FVec[i]) / h - (FVec[i] - FVec[i - 1]) / h);

}

// Решение системы уравнений методом прогонки и возвращение результата

return Progonka(matrics);

}

Вот вычисления моментов и решение матрицы методом пргогонки(возвращает вектор M)

Дальше, сугубо по формулам выводим кубический сплайн:  
vector<vector<double>> showSplain(vector<double>& XVec, vector<double>& FVec, vector<double>& MVec, int n, double h)

{

vector<vector<double>> matrix(4, vector<double>(n));

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

{

matrix[0][i] = MVec[i] / (6 \* h); //(x[i+1] - x)^3

matrix[1][i] = MVec[i + 1] / (6 \* h); //(x - x[i])^3

matrix[2][i] = (FVec[i] - ((h \* h) / 6 \* MVec[i])) / h; //x[i+1] - x

matrix[3][i] = (FVec[i + 1] - ((h \* h) / 6 \* MVec[i + 1])) / h; //x - x[i]

}

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

{

cout << "S" << i + 1 << "(x) = " << matrix[0][i] << "(x - " << XVec[i + 1] << ")^3 + " << matrix[1][i] << "(x - " << XVec[i] << ")^3 + " << matrix[2][i] << "(x - " << XVec[i + 1] << ") + " << matrix[3][i] << "(x - " << XVec[i] << ")" << endl;

}

return matrix;

}

И я вывел его на консоль

1. Вычисление погрешности вычисления S(x) - f(x)

(Вычисление через формулу кубического сплайна и истинного значения функции в точке x, при 100 узлах)

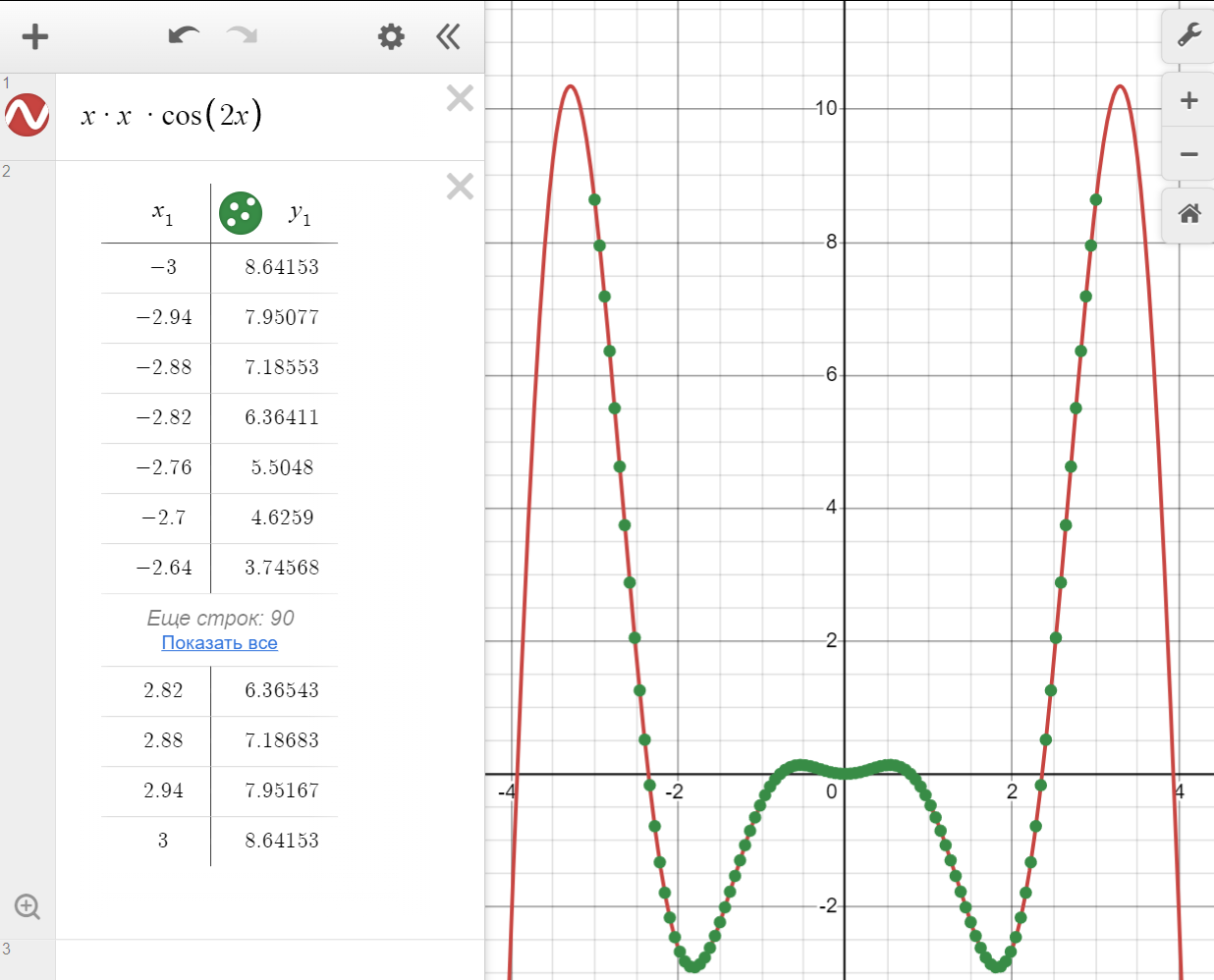
ErrorMax = 0.0139227;

Погрешность интерполяции кубическим сплайном может зависеть от нескольких факторов:

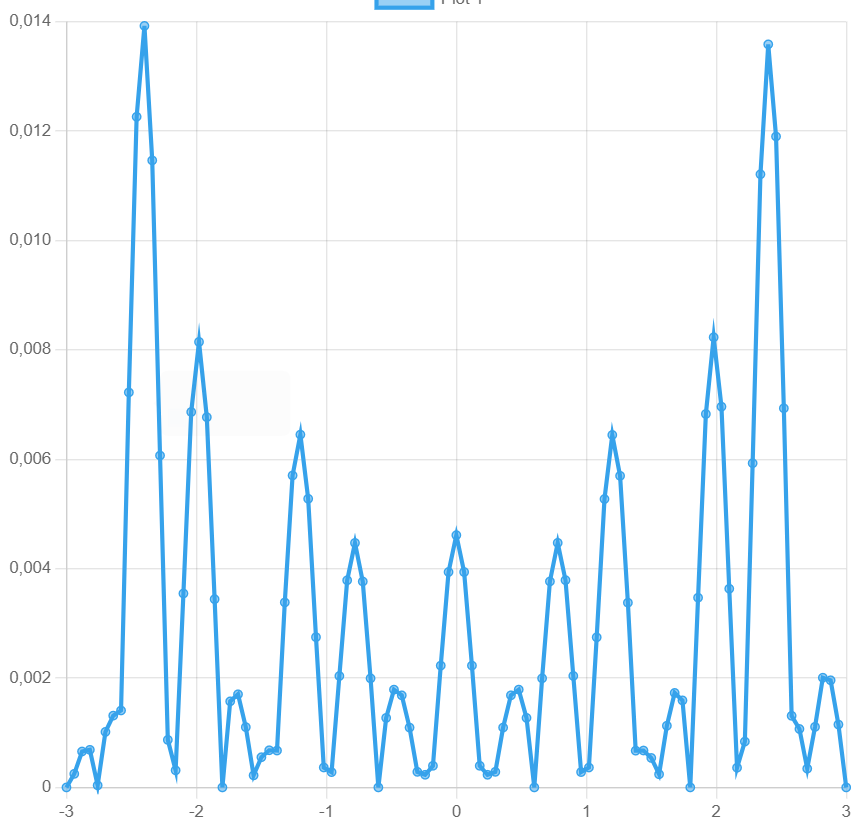
**Сложность функции**: Если функция, которую вы пытаетесь интерполировать, имеет высокую степень сложности (например, она имеет резкие изгибы или быстро меняется), то кубический сплайн может не смочь точно приблизить ее, особенно если точки, по которым строится сплайн, расположены далеко друг от друга.

**Распределение точек**: Если точки, по которым строится сплайн, расположены неравномерно или сгруппированы в определенных областях, это может привести к большим погрешностям в других областях.

**Ошибки округления**: При вычислении коэффициентов сплайна и значений сплайна в конкретных точках могут возникать ошибки округления, которые увеличивают погрешность.

1. График функции и Кубического сплайна:  
     
   

4)График погрешности интерполирования кубическим сплайном.



5)Вывод:

Я использовал кубические сплайны, чтобы создать гладкую кривую, которая проходит через набор точек. Сначала я вычислил коэффициенты сплайна, а затем использовал их, чтобы найти значения сплайна в новых точках, и я получил кривую, которая хорошо приближала к исходной функции.

Но есть небольшие различия между истинными значениями функции и значениями, которые я получил с помощью сплайна. Я думаю, что это могло быть из-за того, что функция была сложной, или из-за того, как расположены точки, или из-за маленьких ошибок, которые происходят, когда мы делаем вычисления на компьютере.

Так что, хотя кубические сплайны - это отличный инструмент, я понял, что они могут давать погрешности, зависящие от некоторых факторов. И я понял, что всегда полезно проверять, насколько большие эти ошибки, чтобы знать, насколько я могу доверять результатам.

1. Листенинг программы:

#include <vector>

#include <iostream>

#include <vector>

#include<algorithm>

#include<fstream>

#include <iomanip>

using namespace std;

ofstream fout("outputPoints.txt");

ofstream fout2("ErrorSize.txt");

//решение трехдиагональной расширенной матрицы методом прогонки

vector<double> Progonka(vector<vector<double>>& matrics)

{

int n = matrics.size();

vector<double> P(n);

vector<double> Q(n);

vector<double> X(n);

P[0] = -matrics[0][1] / matrics[0][0];

Q[0] = matrics[0][n] / matrics[0][0];

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

{

P[i] = -matrics[i][i + 1] / (matrics[i][i] + matrics[i][i - 1] \* P[i - 1]);

Q[i] = (matrics[i][n] - matrics[i][i - 1] \* Q[i - 1]) / (matrics[i][i] + matrics[i][i - 1] \* P[i - 1]);

}

X[n - 1] = Q[n - 1];

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

{

X[i] = P[i] \* X[i + 1] + Q[i];

}

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

{

cout << X[i] << " ";

}

cout << endl;

return X;

}

double Proiz1F(double x) {

return 2\*x\*cos(2\*x) - 2\*x\*x\*sin(2\*x);

}

double Proiz2F(double x) {

return -8\*x\*sin(2\*x) - 4\*x\*x\*cos(2\*x) + 2\*cos(2\*x);

}

// Функция для вычисления моментов сплайна с учетом ограничений S'(a) = f'(a) и S''(b) = f''(b)

vector<double> MomentsCulc(vector<double>& XVec, vector<double>& FVec, int n, double h)

{

// Создание матрицы размером n x (n+1) и заполнение ее нулями

vector<vector<double>> matrics(n, vector<double>(n + 1, 0));

// Заполнение первой и последней строки матрицы

matrics[0][0] = h/3;

matrics[0][1] = h/6;

matrics[n - 1][n - 2] = 0;

matrics[n - 1][n - 1] = 1;

// Заполнение остальных строк матрицы

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

{

matrics[i][i - 1] = h;

matrics[i][i] = 4 \* h;

matrics[i][i + 1] = h;

}

double f\_prime\_a = (FVec[1] - FVec[0])/h - Proiz1F(XVec[0]);

double f\_double\_prime\_b = Proiz2F(XVec[n - 1]);

// Заполнение первого и последнего элемента вектора правой части системы

matrics[0][n] = f\_prime\_a;

matrics[n - 1][n] = f\_double\_prime\_b;

// Заполнение остальных элементов вектора правой части системы

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

{

matrics[i][n] = 6 \* ((FVec[i + 1] - FVec[i]) / h - (FVec[i] - FVec[i - 1]) / h);

}

// Решение системы уравнений методом прогонки и возвращение результата

return Progonka(matrics);

}

// Функция для отображения сплайнов

vector<vector<double>> showSplain(vector<double>& XVec, vector<double>& FVec, vector<double>& MVec, int n, double h)

{

// Создание матрицы 4xN

vector<vector<double>> matrix(4, vector<double>(n));

// Заполнение матрицы коэффициентами сплайнов

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

{

// Коэффициент при (x[i+1] - x)^3

matrix[0][i] = MVec[i] / (6 \* h);

// Коэффициент при (x - x[i])^3

matrix[1][i] = MVec[i + 1] / (6 \* h);

// Коэффициент при (x[i+1] - x)

matrix[2][i] = (FVec[i] - ((h \* h) / 6 \* MVec[i])) / h;

// Коэффициент при (x - x[i])

matrix[3][i] = (FVec[i + 1] - ((h \* h) / 6 \* MVec[i + 1])) / h;

}

// Вывод сплайнов на экран

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

{

cout << "S" << i + 1 << "(x) = " << matrix[0][i] << "(x - " << XVec[i + 1] << ")^3 + " << matrix[1][i] << "(x - " << XVec[i] << ")^3 + " << matrix[2][i] << "(x - " << XVec[i + 1] << ") + " << matrix[3][i] << "(x - " << XVec[i] << ")" << endl;

}

// Возвращение матрицы коэффициентов

return matrix;

}

// Функция для вычисления значения функции f(x) = e^sin(x)

double func(double x)

{

return x\*x\*cos(2\*x);

}

// Функция для вычисления значения сплайна в точке x, учитывая, что x принадлежит отрезку [XVec[i],XVec[i+1]]

double SplainValue(vector<double>& XVec, vector<double>& FVec, vector<double>& MVec, int i, double x, double h, vector<double>& ErrorSize)

{

// Вычисление значения сплайна

double S = MVec[i] / (6 \* h) \* pow(XVec[i + 1] - x, 3) + MVec[i + 1] / (6 \* h) \* pow(x - XVec[i], 3) + (FVec[i] - ((h \* h) / 6 \* MVec[i])) / h \* (XVec[i + 1] - x) + (FVec[i + 1] - ((h \* h) / 6 \* MVec[i + 1])) / h \* (x - XVec[i]);

// Запись значения сплайна в файл

fout << x << "\t" << S << endl;

// Вычисление ошибки между значением функции и сплайна и добавление ее в вектор ошибок

ErrorSize.push\_back(abs(func(x) - S));

// Возвращение значения сплайна

return S;

}

int main()

{

// Определение размера вектора и границ интервала

int size = 15 + 1;

double a = -3, b = 3;

vector<double> XVec(size);

// Вычисление шага

double h = (b - a) / (size - 1);

// Создание вектора для значений функции

vector<double> FVec(size);

// Заполнение векторов XVec и FVec

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

{

XVec[i] = a + i \* h;

FVec[i] = func(XVec[i]);

}

// Вычисление моментов сплайна

vector<double> moments = MomentsCulc(XVec, FVec, size, h);

// Отображение сплайнов

showSplain(XVec, FVec, moments, size, h);

// Создание вектора для точек, в которых будет вычислено значение сплайна

vector<double> points(101);

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

{

points[i] = a + i \* (b - a) / 100;

}

// Создание вектора для хранения ошибок

vector<double> ErrorSize;

// Вычисление значения сплайна в каждой точке из вектора points

for (int j = 0; j < points.size(); j++)

{

double x = points[j];

int i;

for (i = 0; i < size - 1; i++)

{

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

break;

}

SplainValue(XVec, FVec, moments, i, x, h, ErrorSize);

}

// Вычисление максимальной ошибки

double maxErr = \*max\_element(ErrorSize.begin(), ErrorSize.end());

fout2 << maxErr << std::endl;

//setprecision(x)

// Запись ошибок в файл

for (int i = 0; i < 101; i++)

{

fout2 << points[i] << "\t" << ErrorSize[i] << endl;

}

// Закрытие файлов

fout.close();

fout2.close();

return 0;

}