**Лабораторная работа №2**

**«Интерполяционный кубический сплайн»**

Башев Ян Дмитриевич 11 группа

Вариант 2



**Алгоритм нахождений интерполяционного кубического сплайна**

Равноотстоящие узлы находились по формуле

Составлено следующее СЛАУ:

В качестве дополнительных условий использовалось

СЛАУ решено методом прогонки

,

Кубический сплайн вычислялся по формуле

**Максимальная погрешность** между значением функции в узлах , ,

**measurement: 0.520795**

**Графики**

График кубического сплайна (синий) и заданной функции (розовый)

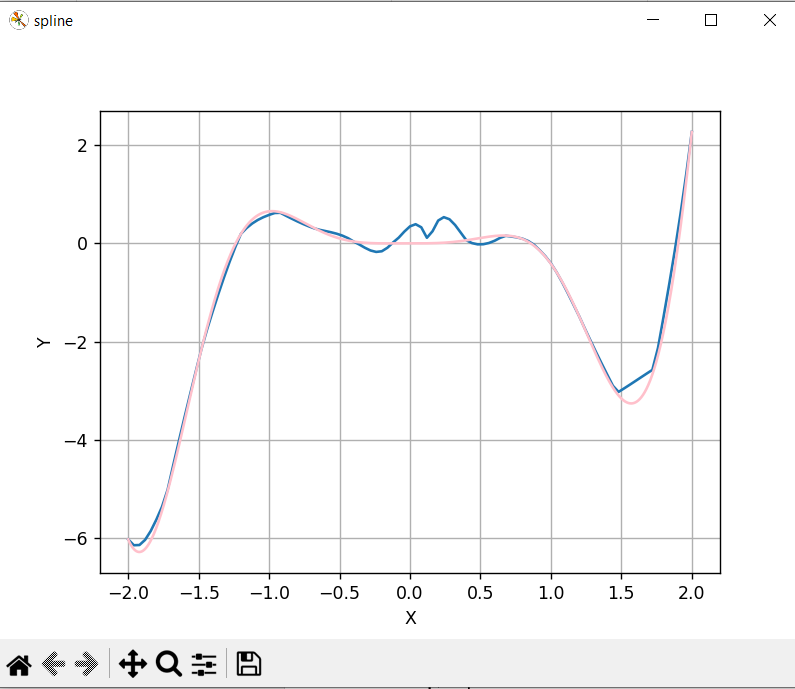
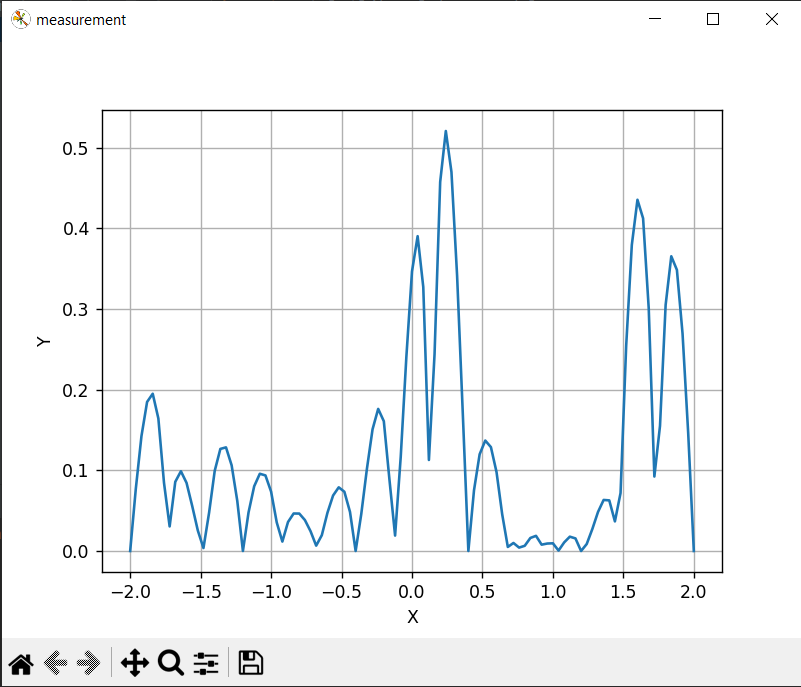


График погрешности интерполирования



**Листинг программы**

#include <iostream>

#include <cmath>

#include <vector>

#include <fstream>

using namespace std;

const double N = 15;

const double a = -2;

const double b = 2;

const double h = (b - a) / N;

double f(double x)

{

return x \* x \* x \* cos(3 \* x - 1);

}

double secondDerivativeF(double x)

{

return 6 \* x \* cos(3 \* x - 1) - 18 \* x \* x \* sin(3 \* x - 1) - 9 \* x \* x \* x \* cos(3 \* x - 1);

}

vector<pair<double, double> > getTableOfEqualNodes(double(\*f)(double))

{

vector<pair<double, double> > table(N + 1);

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

{

table[i] = { a + (double)i \* h, f(a + (double)i \* h) };

}

return table;

}

vector<double> solveProgonka(vector<double> d, vector<double> c, vector<double> e, vector<double> y)

{

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

{

double k = -c[i] / d[i];

c[i] = 0;

d[i + 1] += (e[i] \* k);

y[i + 1] += (y[i] \* k);

}

for (int i = N; i > 0; i--)

{

double k = -e[i - 1] / d[i];

e[i - 1] = 0;

y[i - 1] += (y[i] \* k);

}

return y;

}

double s(double x, int i, vector<pair<double, double> >& table, vector<double>& m)

{

return m[i - 1] \* (table[i].first - x) \* (table[i].first - x) \* (table[i].first - x) / (6. \* h) +

m[i] \* (x - table[i - 1].first) \* (x - table[i - 1].first) \* (x - table[i - 1].first) / (6. \* h) +

(table[i - 1].second - h \* h / 6. \* m[i - 1]) \* (table[i].first - x) / h +

(table[i].second - h \* h / 6. \* m[i]) \* (x - table[i - 1].first) / h;

}

int main()

{

ofstream spline("spline.txt");

ofstream measure("measurement.txt");

vector<double> d(N + 1);

vector<double> u(N);

vector<double> c(N);

vector<double> y(N + 1);

vector<pair<double, double> > table = getTableOfEqualNodes(f);

d[0] = 1;

d[N] = 1;

y[0] = secondDerivativeF(a);

y[N] = secondDerivativeF(b);

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

{

c[i - 1] = h / 6;

d[i] = 2. \* h / 3;

u[i] = h / 6;

y[i] = (table[i + 1].second / - table[i].second) / h - (table[i].second - table[i - 1].second) / h;

}

vector<double> m = solveProgonka(d, c, u, y);

int j = 1;

cout << "spline values with 100 nodes: \n";

double measurement = s(a, 1, table, m);

for (int i = 0; i <= 100; i++)

{

double x = a + (double)i \* (b - a) / 100;

j += (j + 1 <= N && x >= table[j].first);

cout << "S(" << x << "): " << s(x, j, table, m)<<'\n';

spline << x << ' ' << s(x, j, table, m) << '\n';

measurement = max(measurement, abs(s(x, j, table, m) - f(x)));

measure << x << ' ' << abs(s(x, j, table, m) - f(x)) << '\n';

}

cout << "measurement: " << measurement << '\n';

spline.close();

measure.close();

}