Laboratorium 12

Artem Buhera
GĆ01 135678

31.05.2021

W danym laboratorium musieliśmy zaimplementować program, demonstrujący zjawisko Rungego w interpolacji wielomianowej Lagrange'a na przykładzie funkcji

$$f(x) = \frac{1}{1 + 10x^6}, \quad \text{dla } x \in [-1, 1]$$

Program stosuje bazę Newtona do obliczenia wielomianu interpolacyjnego w postaci

$$p_n(x) = \sum_{i=0}^{n} c_i \prod_{j=0}^{i-1} (x - x_j)$$

gdzie współczynniki c_i - ilorazy różnicowe, obliczone na podstawie siatki węzłów funkcji interpolowanej

Używamy dwów rodzajów siatki:

- węzły równoodległe
- węzły Czebyszewa, gdzie

$$x_i = \frac{b+a}{2} + \frac{b-a}{2}\xi_i, \quad \xi_i = \cos\left(\frac{2i+1}{2n+2}\pi\right), \quad i = 0, 1, \dots, n$$

Dla optymalizacji obliczeń $p_n(x)$ używamy modyfikację algorytmu Hornera

$$p_n(x) = \left[\dots\left[\left[c_n(x-x_n) + c_{n-1}\right](x-x_{n-2}) + c_{n-2}\right]\dots\right](x-x_0) + c_0$$

Kod programu:

```cpp
#include <cmath>
#include <iostream>
#include <fstream>
#include <vector.h>

using namespace std;


double exact(double x) {
    return 1.0 / (1.0 + 10.0 * std::pow(x, 6));
}

Vector calculateExact(Vector &args, double (*f)(double)) {
    Vector values(args.n());
    for (int i = 0; i < args.n(); i++)
        values[i] = f(args[i]);
    return values;
}


Vector equidistant(double left_bound, double right_bound, int n) {
    Vector result(n);

    double x = left_bound;
    double step = (right_bound - left_bound) / (n - 1);

    for (int i = 0; i < n; i++) {
        result[i] = x;
        x += step;
    }

    return result;
};

Vector chebyshev(double left_bound, double right_bound, int n) {
    Vector result(n+1, (right_bound + left_bound) / 2.0);
    for (int i = 0; i < n+1; i++) {
        double xi = cos((2.0*i + 1.0) / (2.0*n + 2.0) * M_PI);
        result[i] += (right_bound - left_bound) / 2.0 * xi;
    }
    return result;
};


Vector NewtonBasis(Vector &args, Vector &values) {
    int n = args.n();
    Vector result = values;

    for (int i = 1; i < n; i++) {
        Vector temp = result;

        int count = 0;
        for (int j = i; j < n; j++) {
            temp[j] = (result[j] - result[j-1]) / (args[j] - args[count]);
            count++;
        }

        result = temp;
    }
    return result;
}
```

```cpp
double interpolate(Vector &x, Vector &c, double arg) {
    int n = c.n() - 1;
    double result = c[n] * (arg - x[n-1]) + c[n-1];

    for (n--; n >= 1; n--) {
        result *= arg - x[n-1];
        result += c[n-1];
    }

    return result;
}


void calculate(double left_bound, double right_bound, int n) {
    ofstream out, nodes_equi, nodes_chebyshev;

    stringstream filename;
    filename << "../lab12/n" << to_string(n) << ".csv";
    out.open(filename.str());

    stringstream filename_nodes_equi;
    filename_nodes_equi << "../lab12/n" << to_string(n) << "_nodes_equi.csv";
    nodes_equi.open(filename_nodes_equi.str());

    stringstream filename_nodes_chebyshev;
    filename_nodes_chebyshev << "../lab12/n" << to_string(n) << "_nodes_chebyshev.csv";
    nodes_chebyshev.open(filename_nodes_chebyshev.str());


    Vector args_equi = equidistant(left_bound, right_bound, n);
    Vector values_equi = calculateExact(args_equi, exact);
    Vector coefficients_equi = NewtonBasis(args_equi, values_equi);

    nodes_equi << "x,y" << endl;
    for (int i = 0; i < args_equi.n(); i++)
        nodes_equi << args_equi[i] << "," << values_equi[i] << endl;

    Vector args_chebyshev = chebyshev(left_bound, right_bound, n);
    Vector values_chebyshev = calculateExact(args_chebyshev, exact);
    Vector coefficients_chebyshev = NewtonBasis(args_chebyshev, values_chebyshev);

    nodes_chebyshev << "x,y" << endl;
    for (int i = 0; i < args_chebyshev.n(); i++)
        nodes_chebyshev << args_chebyshev[i] << "," << values_chebyshev[i] << endl;


    out << "x,equidistant,Chebyshev" << endl;
    for (double arg = left_bound; arg < right_bound; arg += 0.01) {
        double interpolated_equidistant =
                interpolate(args_equi, coefficients_equi, arg);
        double interpolated_chebyshev =
                interpolate(args_chebyshev, coefficients_chebyshev, arg);

        out << arg << ","
            << interpolated_equidistant << ","
            << interpolated_chebyshev << endl;
    }

    out.close();
    nodes_equi.close();
    nodes_chebyshev.close();
}
```

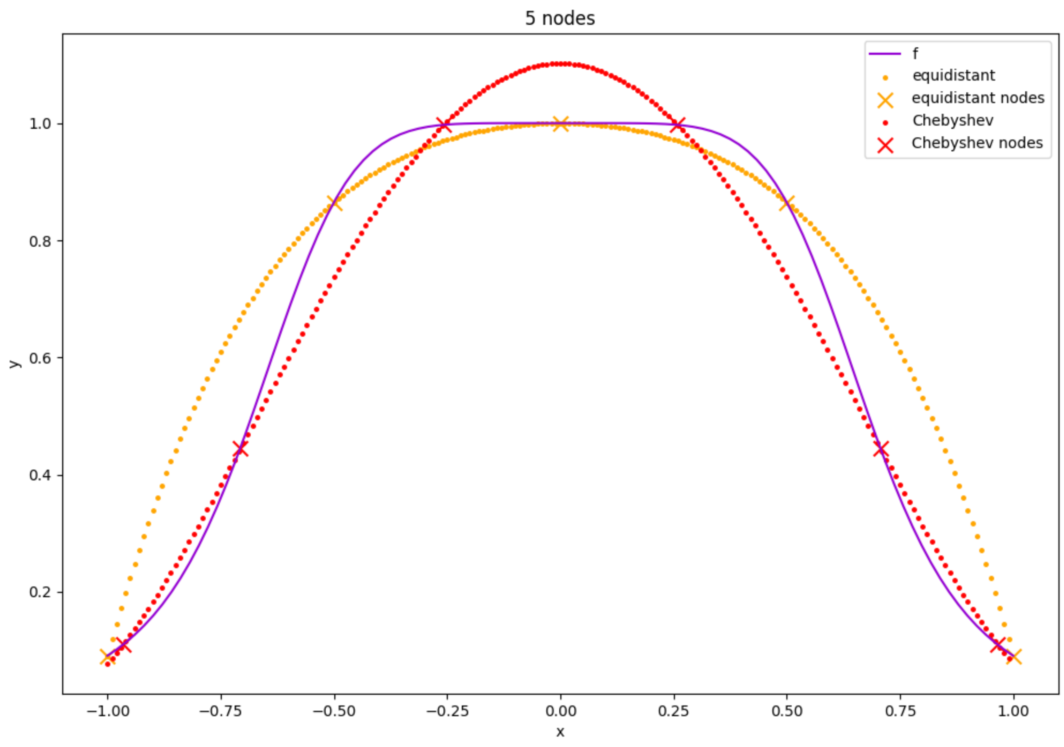
```
int main() {
    double left_bound = -1.0, right_bound = 1.0;
    calculate(left_bound, right_bound, 5);
    calculate(left_bound, right_bound, 10);
    calculate(left_bound, right_bound, 12);
    calculate(left_bound, right_bound, 15);
}
```

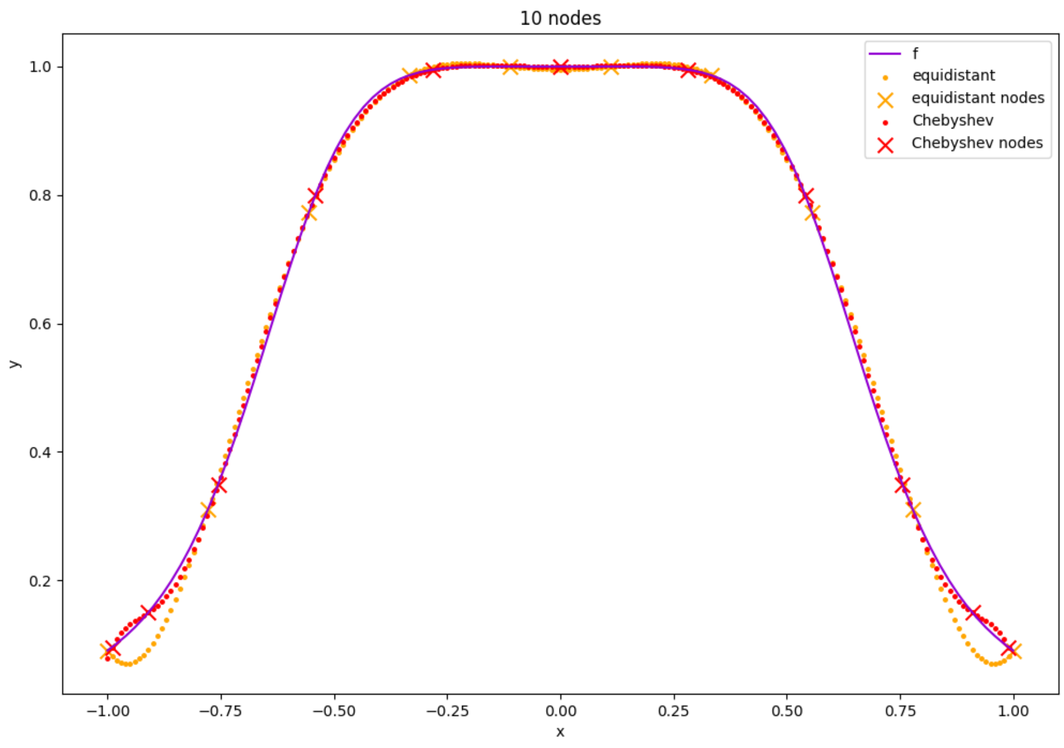
Vector.h:

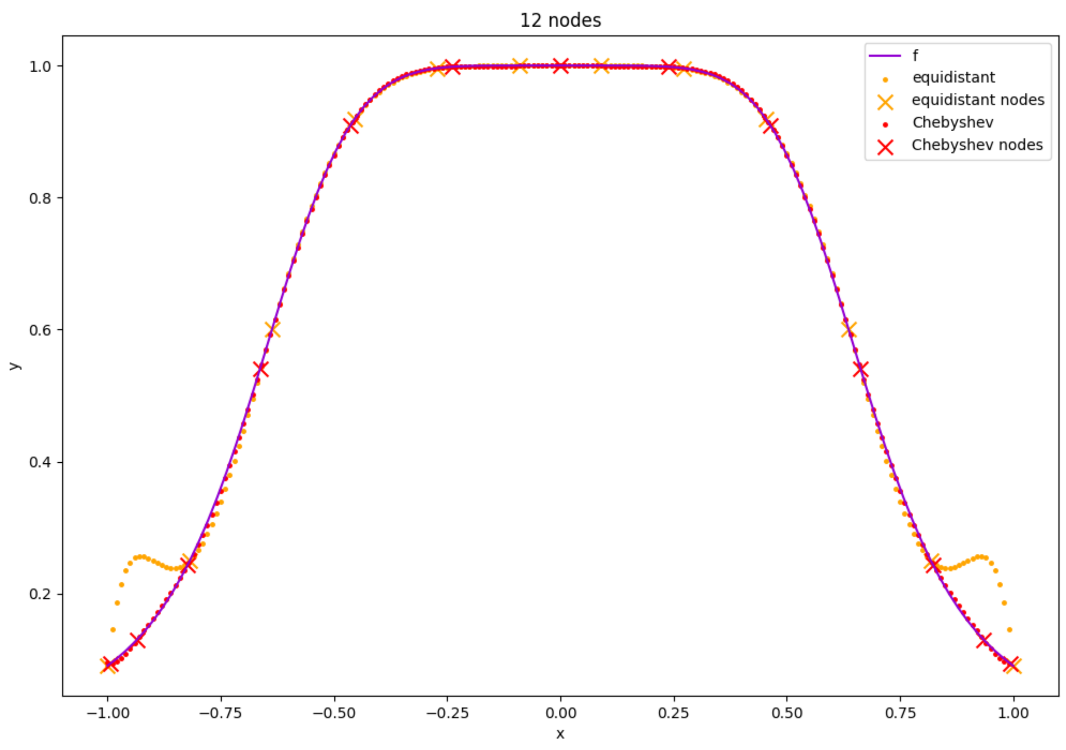
```
#include <vector>
#include <algorithm>
#include <ostream>
#include <iostream>
#include <iomanip>

#ifndef VECTOR_H
#define VECTOR_H

using namespace std;


class Vector {
    int size;
    double* m_points;

public:
    Vector(int size, double default_value) : size(size) {
        m_points = new double[size];
        for (int i = 0; i < size; i++) {
            m_points[i] = default_value;
        }
    }

    Vector(int size) : size(size) {
        m_points = new double[size];
    }

    Vector(std::initializer_list<double> input) : size(input.size()) {
        m_points = new double [size];
        auto begin = input.begin();
        for (int i = 0; i < size; i++) {
            m_points[i] = *begin;
            begin++;
        }
    }

    Vector(const Vector &vect) : size(vect.size) {
        m_points = new double[size];
        for (int i = 0; i < size; i++) {
            m_points[i] = vect.m_points[i];
        }
    }

    ~Vector() {
        delete[] m_points;
        m_points = nullptr;
    }
```

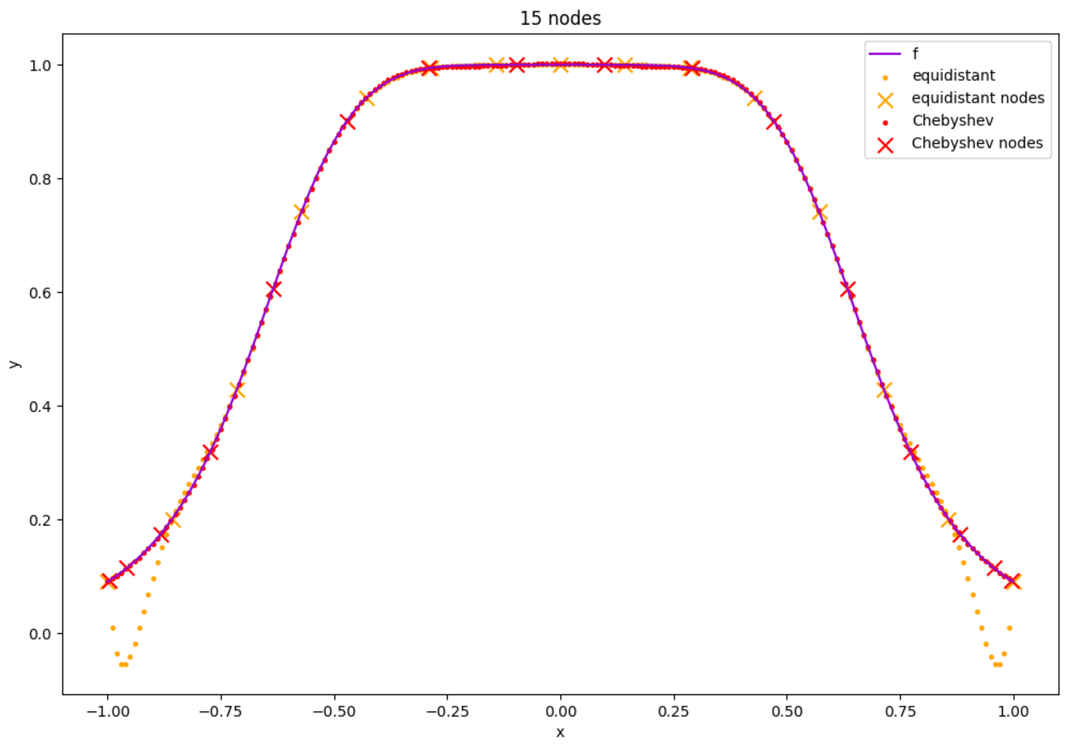
```cpp
    double &operator[](int i) const {
        return m_points[i];
    }

    double &operator[](int i) {
        return m_points[i];
    }

    void operator=(Vector &vect) {
        delete[] m_points;

        size = vect.size;
        m_points = new double[vect.size];
        for (int i = 0; i < size; i++) {
            m_points[i] = vect.m_points[i];
        }
    }

    void print(int width = 8) {
        cout << left;
        for (int i = 0; i < size; i++) {
            cout << setw(width) << m_points[i] << endl;
        }
        cout << endl << endl;
    }

    const int n() {
        return int(size);
    }
};

#endif
```

Otrzymane wykresy funkcji interpolowanej oraz interpolujących na węzłach równoodległych i Czebyszewa w zależności od ilości węzłów:









Dla węzłów równoodległych możemy łatwo zaobserwować zjawisko Rungego na końcach wykresów.

Natomiast dla węzłów Czebyszewa wartości pozostają zbliżone do faktycznych na całym zakresie.