

Министерство образования и науки Российской Федерации
Санкт-Петербургский политехнический университет Петра
Великого

—
Институт компьютерных наук и технологий
Высшая школа программной инженерии

Лабораторная работа №1
по дисциплине «Вычислительная математика»

Выполнил
Студент группы 5130904/20004

Шелковников Д.С.

Преподаватель

Устинов С.М.

Оглавление

Задание	2
Результаты.....	3
Код программы.....	5
<DIR>/computational_mathematics/first_lab/Langrage.cpp	5
<DIR>/computational_mathematics/first_lab/Langrage.h.....	6
<DIR>/computational_mathematics/first_lab/Quanc8.cpp	6
<DIR>/computational_mathematics/first_lab/Quanc8.h	9
<DIR>/computational_mathematics/first_lab/Spline.cpp.....	10
<DIR>/computational_mathematics/first_lab/Spline.h	12
<DIR>/computational_mathematics/first_lab/main.cpp	12

Задание

Для функции $f(x) = 1/(1+x)$ по узлам $x_k = 0.1k$ ($k=0,1,\dots,10$) построить полином Лагранжа $L(x)$ 10-й степени и сплайн-функцию $S(x)$. Вычислить значения всех трех функций в точках $y_k = 0.05 + 0.1k$ ($k=0,1,\dots,9$). Результаты отобразить графически.

Используя программу **QUANC8**, вычислить два интеграла:

$$\int_2^5 (\text{abs}(x - \text{tg}(x)))^m dx, \text{ для } m = -1 \text{ и для } m = -0.5.$$

Результаты

X	Langrage	spline	F(x)
0.05	0.952381	0.952427	0.952381
0.15	0.869565	0.869549	0.869565
0.25	0.8	0.800002	0.8
0.35	0.740741	0.740738	0.740741
0.45	0.689655	0.689654	0.689655
0.55	0.645161	0.645161	0.645161
0.65	0.606061	0.60606	0.606061
0.75	0.571429	0.571428	0.571429
0.85	0.540541	0.540539	0.540541
0.95	0.51282	0.512824	0.512821

Result for m = -1: 1.37281

Error: 5.45525e-05

NoFun: 145

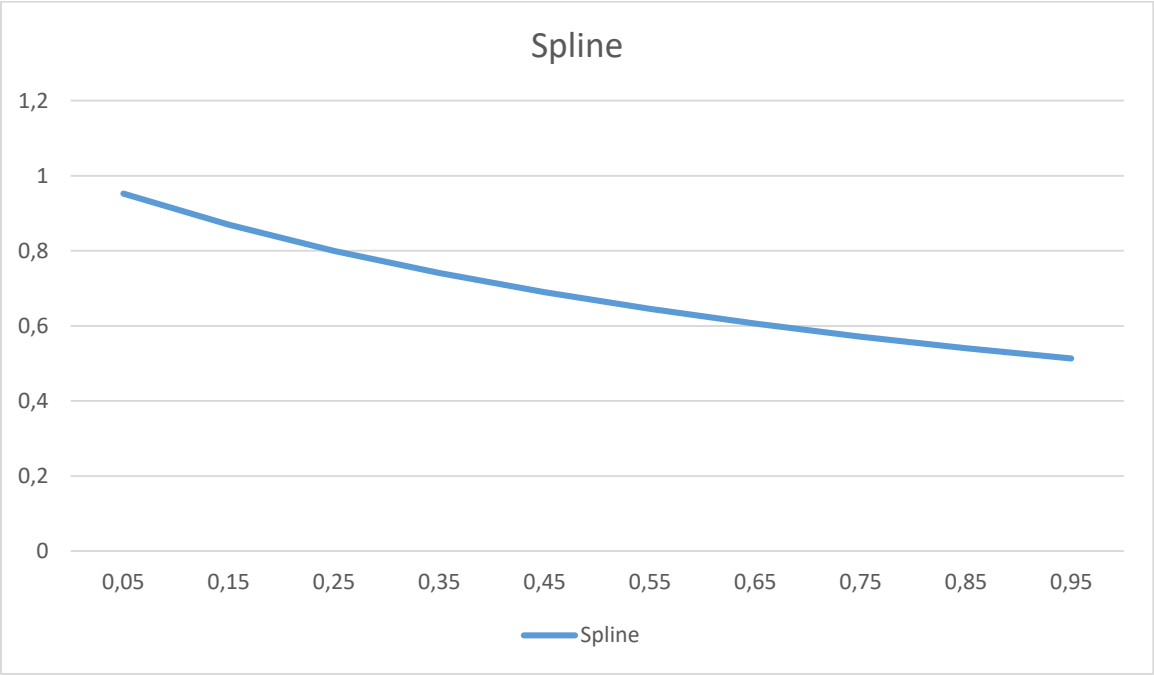
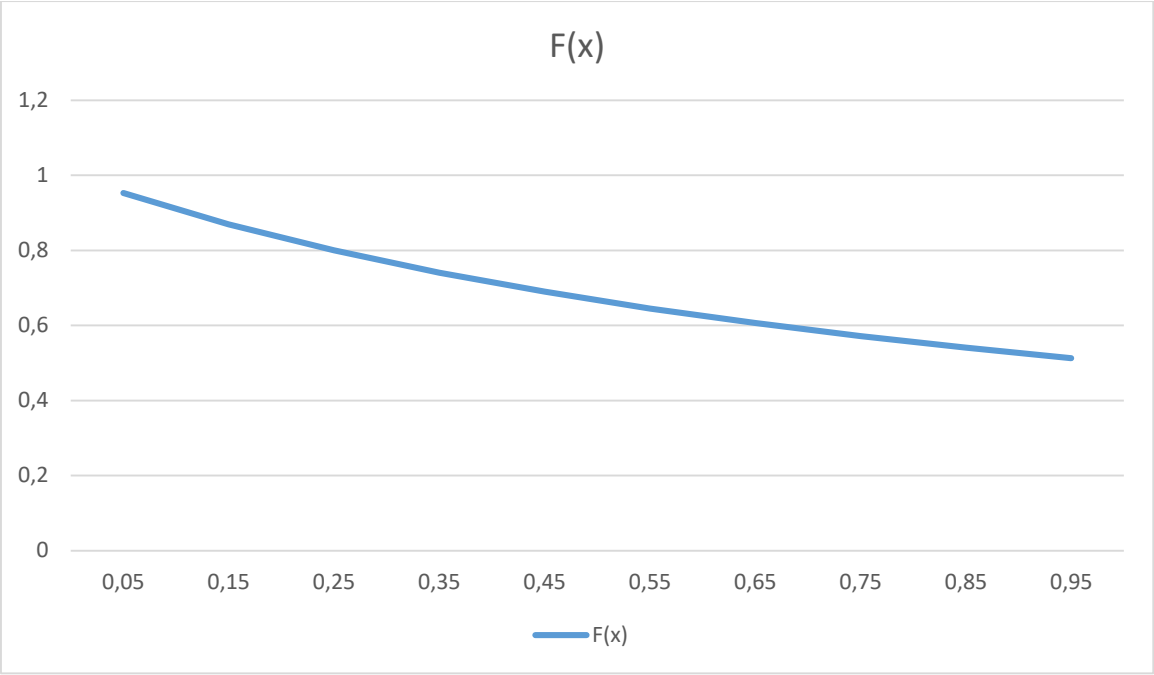
Flag: 0

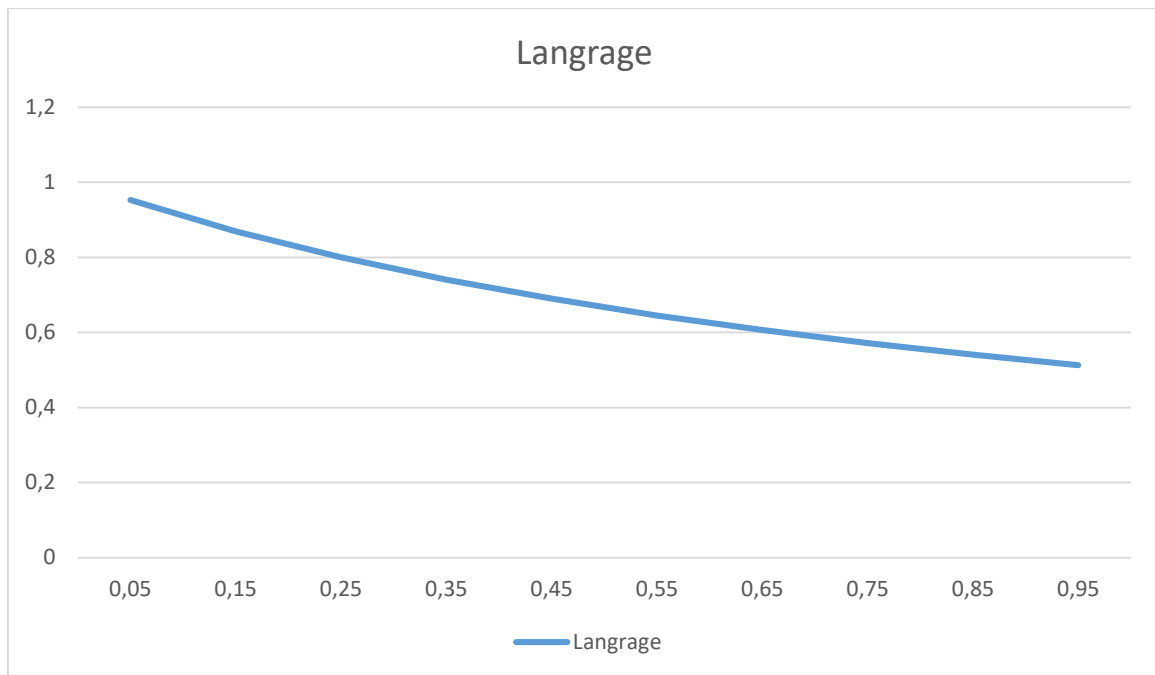
Result for m = -0.5: 1.75731

Error: 6.48105e-06

NoFun: 145

Flag: 0





Код программы

```
<DIR>/computational_mathematics/first_lab/Langrage.cpp
```

```
#include "Langrage.h"
```

```
#include <stdexcept>
```

```
#include <cstdint>
```

```
dimkashelk::Langrage::Langrage(const std::vector<std::pair<double, double> >
&points):
    points_(points)
{}
```

```
double dimkashelk::Langrage::operator()(const double x) const {
    double res = 0.0;
    for (size_t i = 0; i < points_.size(); i++) {
        res += calculateFraction(i, x) * points_[i].second;
    }
    return res;
}
```

```
double dimkashelk::Langrage::calculateFraction(const std::size_t index, const
double x) const {
    double res = 1.0;
    double x_cur = points_[index].first;
    for (size_t i = 0; i < points_.size(); i++) {
        if (i != index) {
            res *= (x - points_[i].first) / (x_cur - points_[i].first);
        }
    }
    return res;
}
```

```
}
```

```
<DIR>/computational_mathematics/first_lab/Langrage.h
#ifndef COMPUTATIONAL_MATHEMATICS_LANGRAGE_H
#define COMPUTATIONAL_MATHEMATICS_LANGRAGE_H

#include <vector>

namespace dimkashelk {
    class Langrage {
    public:
        explicit Langrage(const std::vector<std::pair<double, double>>
&points);

        double operator()(double x) const;

    private:
        const std::vector<std::pair<double, double> > points_;

        double calculateFraction(std::size_t index, double x) const;
    };
}
#endif
```

```
<DIR>/computational_mathematics/first_lab/Quanc8.cpp
#include "Quanc8.h"

#include <algorithm>
#include <cmath>

dimkashelk::Quanc8::Quanc8(const std::function<double(double)> &fun, double
a, double b, double abs_err, double rel_err): fun_(fun),
    a_(a),
    b_(b),
    abs_err_(abs_err),
    rel_err_(rel_err),
    result_(0.0),
    error_(0.0),
    no_fun_(0),
    flag_(0.0)
{
    double QRIGHT[32], F[17], X[17], FSAVE[9][31], XSAVE[9][31];
    int LEVMIN, LEVMAX, LEVOUT, NOMAX, NOFIN, LEV, NIM, J, I;
    double W0, W1, W2, W3, W4, COR11, AREA, X0, F0, STONE, STEP;
    double QLEFT, QNOW, QDIFF, QPREV, TOLERR, ESTERR;

    LEVMIN = 1;
    LEVMAX = 30;
    LEVOUT = 6;
```

```

    NOMAX = 5000;
    NOFIN = NOMAX - (8 * (LEVMAX - LEVOUT + static_cast<int>(std::pow(2,
LEVOUT + 1)))));

    W0 = 3956.0 / 14175.0;
    W1 = 23552.0 / 14175.0;
    W2 = -3712.0 / 14175.0;
    W3 = 41984.0 / 14175.0;
    W4 = -18160.0 / 14175.0;

    flag_ = 0.0;
    result_ = 0.0;
    COR11 = 0.0;
    error_ = 0.0;
    AREA = 0.0;
    no_fun_ = 0;
    if (a_ == b_) { return; }
    LEV = 0;
    NIM = 1;
    X0 = a_;
    X[16] = b_;
    QPREV = 0.0;
    F0 = fun_(X0);
    STONE = (b_ - a_) / 16.0;
    X[8] = (X0 + X[16]) / 2.0;
    X[4] = (X0 + X[8]) / 2.0;
    X[12] = (X[8] + X[16]) / 2.0;
    X[2] = (X0 + X[4]) / 2.0;
    X[6] = (X[4] + X[8]) / 2.0;
    X[10] = (X[8] + X[12]) / 2.0;
    X[14] = (X[12] + X[16]) / 2.0;
    for (J = 2; J <= 16; J = J + 2) {
        F[J] = fun_(X[J]);
    }
    no_fun_ = 9;

trenta:
    X[1] = (X0 + X[2]) / 2.0;
    F[1] = fun_(X[1]);
    for (J = 3; J <= 15; J = J + 2) {
        X[J] = (X[J - 1] + X[J + 1]) / 2.0;
        F[J] = fun_(X[J]);
    }
    no_fun_ = no_fun_ + 8;
    STEP = (X[16] - X0) / 16.0;
    QLEFT = (W0 * (F0 + F[8]) + W1 * (F[1] + F[7]) + W2 * (F[2] + F[6]) + W3
* (F[3] + F[5])
        + W4 * F[4]) * STEP;
    QRIGHT[LEV + 1] = (W0 * (F[8] + F[16]) + W1 * (F[9] + F[15]) + W2 *
(F[10] + F[14])
        + W3 * (F[11] + F[13]) + W4 * F[12]) * STEP;
    QNOW = QLEFT + QRIGHT[LEV + 1];

```

Санкт-Петербург

2024

```

QDIFF = QNOW - QPREV;
AREA = AREA + QDIFF;

ESTERR = fabs(QDIFF) / 1023.0;
if (abs_err > (rel_err * fabs(AREA)) * (STEP / STONE))
    TOLERR = abs_err;
else
    TOLERR = (rel_err * fabs(AREA)) * (STEP / STONE);
if (LEV < LEVMIN)
    goto cinquanta;
if (LEV >= LEVMAX)
    goto sessantadue;
if (no_fun_ > NOFIN)
    goto sessanta;
if (ESTERR <= TOLERR)
    goto settanta;

cinquanta:
    NIM = 2 * NIM;
    LEV = LEV + 1;

    for (I = 1; I <= 8; I++) {
        FSAVE[I][LEV] = F[I + 8];
        XSAVE[I][LEV] = X[I + 8];
    }

    QPREV = QLEFT;
    for (I = 1; I <= 8; I++) {
        J = -I;
        F[2 * J + 18] = F[J + 9];
        X[2 * J + 18] = X[J + 9];
    }
    goto trenta;

sessanta:
    NOFIN = 2 * NOFIN;
    LEVMAX = LEVOUT;
    flag_ = flag_ + ((b_ - x0) / (b_ - a_));
    goto settanta;

sessantadue:
    flag_ = flag_ + 1.0;

settanta:
    result_ = result_ + QNOW;
    error_ = error_ + ESTERR;
    COR11 = COR11 + QDIFF / 1023.0;

    while (NIM % 2 != 0) {
        NIM = NIM / 2;
        LEV = LEV - 1;
    }

```



```

    NIM = NIM + 1;
    if (LEV <= 0)
        goto ottanta;

    QPREV = QRIGHT[LEV];
    X0 = X[16];
    F0 = F[16];
    for (I = 1; I <= 8; I++) {
        F[2 * I] = FSAVE[I][LEV];
        X[2 * I] = XSAVE[I][LEV];
    }
    goto trenta;

ottanta:
    result_ = result_ + COR11;
    if (error_ == 0.0)
        return;
    while (std::fabs(result_) + (error_) == std::fabs(result_))
        error_ = 2.0 * (error_);
}

double dimkashelk::Quanc8::getResult() const {
    return result_;
}

double dimkashelk::Quanc8::getError() const {
    return error_;
}

int dimkashelk::Quanc8::getNoFun() const {
    return no_fun_;
}

double dimkashelk::Quanc8::getFlag() const {
    return flag_;
}

```

```

<DIR>/computational_mathematics/first_lab/Quanc8.h
#ifndef QUANC8_H
#define QUANC8_H
#include <functional>

```

```

namespace dimkashelk {
    class Quanc8 {
    public:
        /**
         * \brief
         * \param fun user functions with one double argument
         * \param a lower bound of integration
         * \param b upper bound of integration

```

```

        * \param abs_err absolute error
        * \param rel_err intermediate error
        */
    Quanc8(const std::function<double (double)> &fun, double a, double b,
double abs_err, double rel_err);
    double getResult() const;
    double getError() const;
    int getNoFun() const;
    double getFlag() const;
private:
    std::function<double (double)> fun_;
    const double a_;
    const double b_;
    const double abs_err_;
    const double rel_err_;
    double result_;
    double error_;
    int no_fun_;
    double flag_;
};
}
#endif

```

<DIR>/computational_mathematics/first_lab/Spline.cpp

```
#include "Spline.h"
```

```
#include <stdexcept>
```

```

dimkashelk::Spline::Spline(const std::vector<std::pair<double, double> >
&points) {
    std::vector<double> B(points.size());
    std::vector<double> C(points.size());
    std::vector<double> D(points.size());

    const size_t count_minus_1 = points.size() - 1;
    if (points.size() < 2) { throw std::logic_error("Check points"); }
    if (points.size() > 2) {
        D[0] = points[1].first - points[0].first;
        C[1] = (points[1].second - points[0].second) / D[0];
        for (size_t i = 2; i <= count_minus_1; i++) {
            D[i - 1] = points[i].first - points[i - 1].first;
            B[i - 1] = 2.0 * (D[i - 2] + D[i - 1]);
            C[i] = (points[i].second - points[i - 1].second) / D[i - 1];
            C[i - 1] = C[i] - C[i - 1];
        }
        B[0] = -D[0];
        B[points.size() - 1] = -D[points.size() - 2];
        C[0] = 0.0;
        C[points.size() - 1] = 0.0;
    }
}

```

```

        if (points.size() != 3) {
            C[0] = C[2] / (points[3].first - points[1].first) - C[1] /
(points[2].first - points[0].first);
            C[points.size() - 1] = C[points.size() - 2] /
(points[points.size() - 1].first -
points[points.size() - 3].first) -
C[points.size() - 3] /
(points[points.size() - 2].first -
points[points.size() - 4].first);
            C[0] = C[0] * D[0] * D[0] / (points[3].first - points[0].first);
            C[points.size() - 1] = -C[points.size() - 1] * D[points.size() -
2] * D[points.size() - 2] /
(points[points.size() - 1].first -
points[points.size() - 4].first);
        }
        for (size_t i = 2; i <= points.size(); i++) {
            const double current = D[i - 2] / B[i - 2];
            B[i - 1] = B[i - 1] - current * D[i - 2];
            C[i - 1] = C[i - 1] - current * C[i - 2];
        }
        C[points.size() - 1] = C[points.size() - 1] / B[points.size() - 1];
        for (size_t i = 1; i <= count_minus_1; i++) {
            const size_t d = points.size() - i;
            C[d - 1] = (C[d - 1] - D[d - 1] * C[d]) / B[d - 1];
        }
        B[points.size() - 1] = (points[points.size() - 1].second -
points[count_minus_1 - 1].second) /
D[count_minus_1 - 1] + D[count_minus_1 - 1] *
(C[count_minus_1 - 1] + 2. * C[points.size() -
1]);
        for (size_t i = 1; i <= count_minus_1; i++) {
            B[i - 1] = (points[i].second - points[i - 1].second) / D[i - 1] -
D[i - 1] * (C[i] + 2. * C[i - 1]);
            D[i - 1] = (C[i] - C[i - 1]) / D[i - 1];
            C[i - 1] = 3.0 * C[i - 1];
        }
        C[points.size() - 1] = 3. * C[points.size() - 1];
        D[points.size() - 1] = D[points.size() - 2];
    } else {
        B[0] = (points[1].second - points[0].second) / (points[1].first -
points[0].first);
        C[0] = 0.0;
        D[0] = 0.0;
        B[1] = B[0];
        C[1] = 0.0;
        D[1] = 0.0;
    }
    points_ = points;
    for (const auto &point: points) {
        a.push_back(point.second);
    }
    b = B;

```

```

        c = C;
        d = D;
    }

double dimkashelk::Spline::operator()(const double number) const {
    size_t i = 1;
    size_t j = points_.size() + 1;
    do {
        const size_t k = (i + j) / 2;
        if (number < points_[k - 1].first) {
            j = k;
        }
        if (number >= points_[k - 1].first) {
            i = k;
        }
    } while (j > i + 1);
    const double DX = number - points_[i - 1].first;
    return a[i - 1] + DX * b[i - 1] + DX * DX * c[i - 1] + DX * DX * DX * d[i
- 1];
}

```

```

<DIR>/computational_mathematics/first_lab/Spline.h
#ifndef SPLINE_H
#define SPLINE_H

#include <vector>

namespace dimkashelk {
    class Spline {
    public:
        explicit Spline(const std::vector<std::pair<double, double> >
&points);

        double operator()(double number) const;

    private:
        std::vector<std::pair<double, double>> points_;
        std::vector<double> a_;
        std::vector<double> b_;
        std::vector<double> c_;
        std::vector<double> d_;
    };
}
#endif

```

```

<DIR>/computational_mathematics/first_lab/main.cpp
#include <cmath>
#include <iostream>
#include <vector>
#include <functional>

```

```

#include "Langrage.h"
#include "Quanc8.h"
#include "Spline.h"

double f(const double x) {
    return 1 / (1 + x);
}

double func(const double x, const double m) {
    return std::pow(std::abs(x - std::tan(x)), m);
}

int main() {
    std::vector<std::pair<double, double>> function;
    function.reserve(11);
    for (int i = 0; i < 11; i++) {
        function.emplace_back(0.1 * i, f(0.1 * i));
    }
    const dimkashelk::Langrage langrage(function);
    const dimkashelk::Spline spline(function);
    std::cout << "X   | Langrage | spline | F(x)\n";
    for (int i = 0; i < 10; i++) {
        const double x = 0.05 + 0.1 * i;
        std::cout << x << " | " << langrage(x) << " | " << spline(x) << " | "
<< f(x) << "\n";
    }

    std::cout << "\n\n\n";

    auto f1 = std::bind(func, std::placeholders::_1, -1);
    const dimkashelk::Quanc8 func1(f1, 2, 5, 0.0001, 0.00001);
    std::cout << "Result for m = -1: " << func1.getResult() << "\n"
        << "Error: " << func1.getError() << "\n"
        << "NoFun: " << func1.getNoFun() << "\n"
        << "Flag: " << func1.getFlag() << "\n\n\n";

    auto f2 = std::bind(func, std::placeholders::_1, -0.5);
    const dimkashelk::Quanc8 func2(f2, 2, 5, 0.00001, 0.00001);
    std::cout << "Result for m = -0.5: " << func2.getResult() << "\n"
        << "Error: " << func2.getError() << "\n"
        << "NoFun: " << func2.getNoFun() << "\n"
        << "Flag: " << func2.getFlag() << "\n";
    return 0;
}

```