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](#_Toc159931522)

[Результаты 3](#_Toc159931523)

[Код программы 5](#_Toc159931524)

[<DIR>/computational\_mathematics/first\_lab/Langrage.cpp 5](#_Toc159931525)

[<DIR>/computational\_mathematics/first\_lab/Langrage.h 6](#_Toc159931526)

[<DIR>/computational\_mathematics/first\_lab/Quanc8.cpp 6](#_Toc159931527)

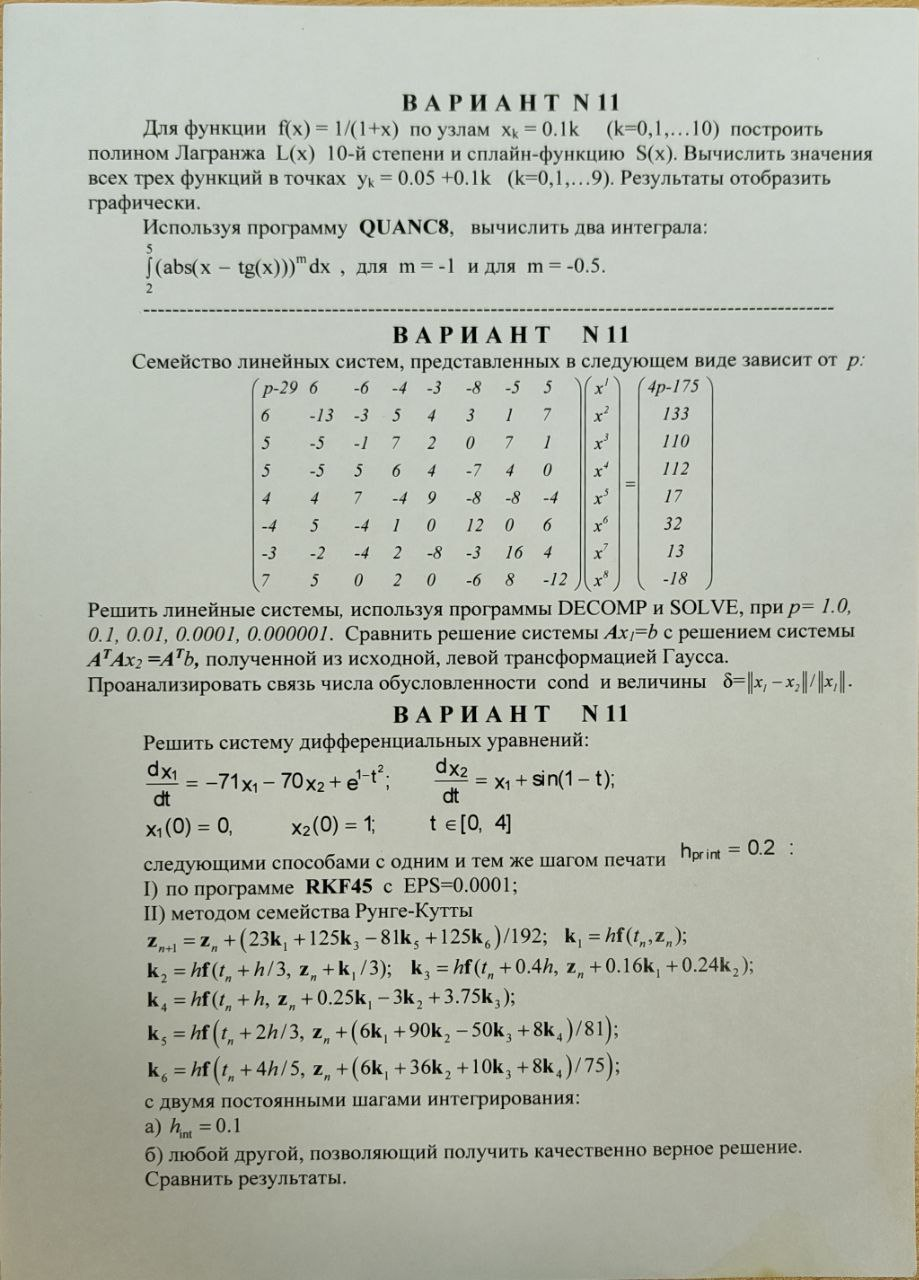
[<DIR>/computational\_mathematics/first\_lab/Quanc8.h 9](#_Toc159931528)

[<DIR>/computational\_mathematics/first\_lab/Spline.cpp 10](#_Toc159931529)

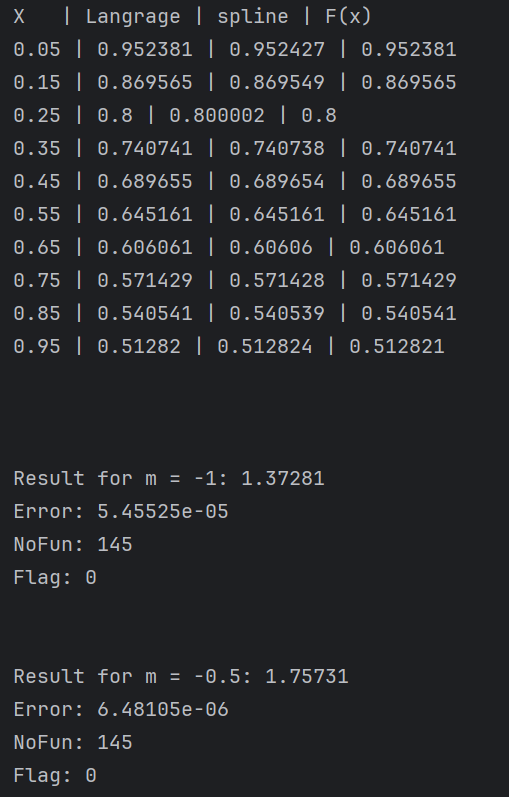
[<DIR>/computational\_mathematics/first\_lab/Spline.h 12](#_Toc159931530)

[<DIR>/computational\_mathematics/first\_lab/main.cpp 12](#_Toc159931531)

# Задание



# Результаты



# Код программы

## <DIR>/computational\_mathematics/first\_lab/Langrage.cpp

#include "Langrage.h"

#include <stdexcept>

#include <cstddef>

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];

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;

}