Министерство образования и науки Российской Федерации

Санкт-Петербургский политехнический университет Петра Великого

—

Институт компьютерных наук и технологий

Высшая школа программной инженерии

**Лабораторная работа №2**

**по дисциплине «Вычислительная математика»**

Выполнил

Студент группы 5130904/20004 Шелковников Д.С.

Преподаватель Устинов С.М.

Оглавление

[Задание 2](#_Toc160059443)

[Результаты 3](#_Toc160059444)

[Выводы 5](#_Toc160059445)

[Код программы 5](#_Toc160059446)

[<DIR>/computational\_mathematics/second\_lab/Decomp.cpp 5](#_Toc160059447)

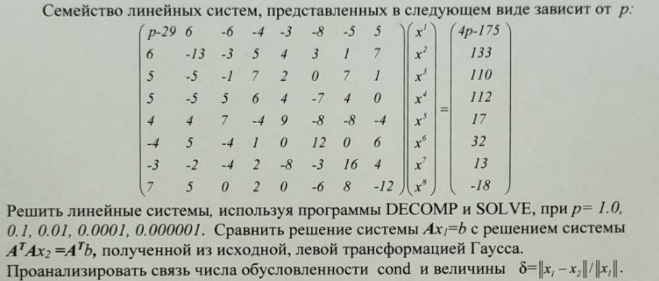
[<DIR>/computational\_mathematics/second\_lab/Decomp.h 6](#_Toc160059448)

[<DIR>/computational\_mathematics/second\_lab/Solve.cpp 7](#_Toc160059449)

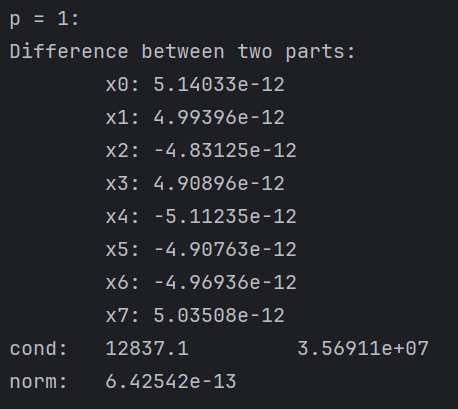
[<DIR>/computational\_mathematics/second\_lab/Solve.h 8](#_Toc160059450)

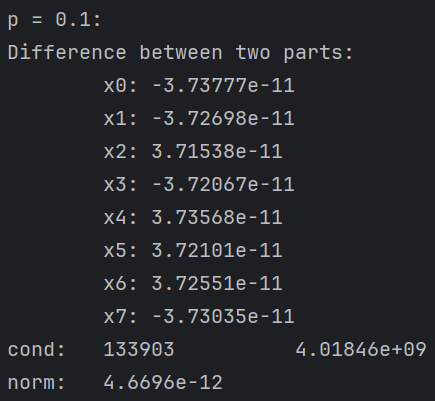
[<DIR>/computational\_mathematics/second\_lab/main.cpp 9](#_Toc160059451)

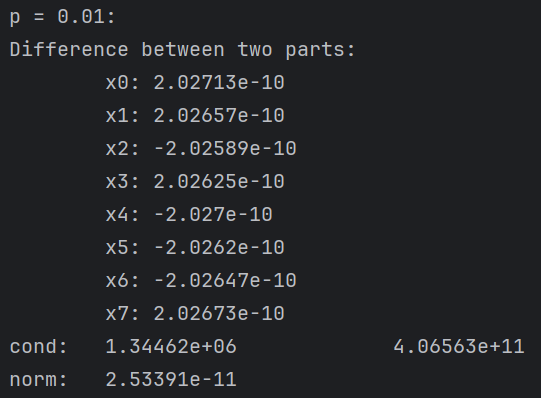
# Задание

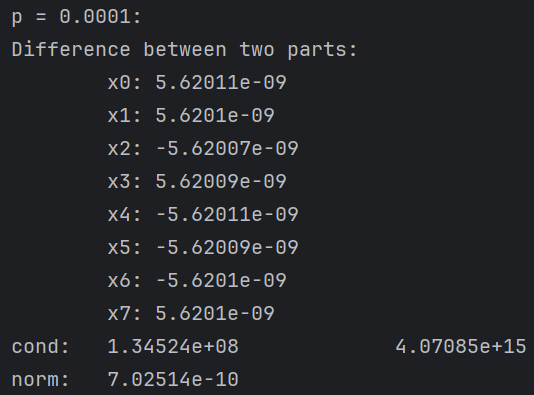


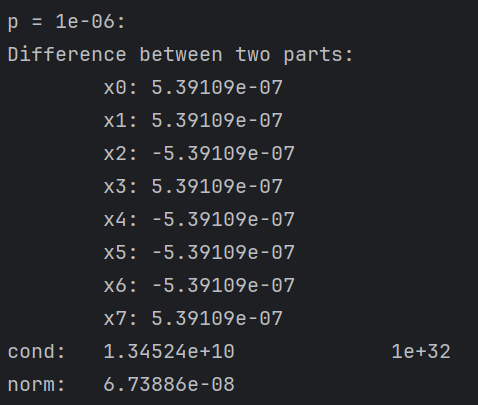
# Результаты











# Выводы

Решения двух систем максимально схожи, что видно по разности между соответствующими значениями x. Причем с уменьшением значения параметра P увеличивается разница между ними. Так же с уменьшением параметра Р увеличивается число обусловленности и уменьшается подсчитываемая величина. Это происходит потому что:

* Матрица близка к вырождению (определитель стремится к 0 с изменением параметра)
* Различия между элементами растет

# Код программы

## <DIR>/computational\_mathematics/second\_lab/Decomp.cpp

#include "Decomp.h"

#include <stdexcept>

#include <cmath>

#include "Solve.h"

dimkashelk::Decomp::Decomp(): cond\_(0.0),

size\_(0),

data\_(nullptr),

pivot\_(nullptr),

flag\_(0) {

}

void dimkashelk::Decomp::operator()(const std::vector<std::vector<double> > &matrix) {

if (matrix.empty()) {

throw std::logic\_error("Check matrix");

}

if (matrix.size() != matrix[0].size()) {

throw std::logic\_error("Check size of matrix");

}

free();

size\_ = static\_cast<int>(matrix.size());

data\_ = new double[size\_ \* size\_];

try {

pivot\_ = new int[matrix.size() \* matrix.size()];

} catch (...) {

delete[] data\_;

throw;

}

int ind = 0;

for (auto &i: matrix) {

for (double j: i) {

data\_[ind] = j;

ind++;

}

}

details::decomp(size\_, size\_, data\_, std::addressof(cond\_), pivot\_, std::addressof(flag\_));

}

dimkashelk::Decomp::~Decomp() {

free();

}

void dimkashelk::Decomp::free() const {

if (data\_ != nullptr) {

delete[] data\_;

}

if (pivot\_ != nullptr) {

delete[] pivot\_;

}

}

## <DIR>/computational\_mathematics/second\_lab/Decomp.h

#ifndef DECOMP\_H

#define DECOMP\_H

#include <vector>

namespace dimkashelk {

namespace details {

int decomp(int n, int ndim,

double \*a, double \*cond,

int pivot[], int \*flag);

}

class Solve;

class Decomp {

friend class Solve;

public:

Decomp();

void operator()(const std::vector<std::vector<double> > &matrix);

~Decomp();

private:

double cond\_;

int size\_;

double \*data\_;

int \*pivot\_;

int flag\_;

void free() const;

};

}

#endif

## <DIR>/computational\_mathematics/second\_lab/Solve.cpp

#include "Solve.h"

#include <stdexcept>

#include "Decomp.h"

dimkashelk::Solve::Solve(): size\_(0),

data\_right\_(nullptr),

cond\_(0.0) {

}

void dimkashelk::Solve::operator()(const std::vector<std::vector<double> > &matrix\_left,

const std::vector<double> &matrix\_right) {

if (matrix\_left.size() != matrix\_right.size()) {

throw std::logic\_error("Check data");

}

free();

size\_ = static\_cast<int>(matrix\_right.size());

data\_right\_ = new double[size\_];

for (int i = 0; i < size\_; i++) {

data\_right\_[i] = matrix\_right[i];

}

Decomp dec;

dec(matrix\_left);

cond\_ = dec.cond\_;

const int size = static\_cast<int>(matrix\_left.size());

details::solve(size, size, dec.data\_, data\_right\_, dec.pivot\_);

}

std::vector<double> dimkashelk::Solve::get\_result() const {

std::vector<double> res(size\_);

for (int i = 0; i < size\_; i++) {

res[i] = data\_right\_[i];

}

return res;

}

double dimkashelk::Solve::get\_cond() const {

return cond\_;

}

dimkashelk::Solve::~Solve() {

free();

}

void dimkashelk::Solve::free() const {

if (data\_right\_ != nullptr) {

delete[] data\_right\_;

}

}

## <DIR>/computational\_mathematics/second\_lab/Solve.h

#ifndef SOLVE\_H

#define SOLVE\_H

#include <vector>

namespace dimkashelk {

namespace details {

int solve(int n, int ndim,

double \*a, double b[],

int pivot[]);

}

class Decomp;

class Solve {

friend class Decomp;

public:

Solve();

void operator()(const std::vector<std::vector<double> > &matrix\_left, const std::vector<double> &matrix\_right);

[[nodiscard]] std::vector<double> get\_result() const;

[[nodiscard]] double get\_cond() const;

~Solve();

private:

int size\_;

double \*data\_right\_;

double cond\_;

void free() const;

};

}

#endif

## <DIR>/computational\_mathematics/second\_lab/main.cpp

#include <algorithm>

#include <iostream>

#include "Solve.h"

std::vector<double> get\_difference\_of\_matrix(const std::vector<double> &left,

const std::vector<double> &right) {

std::vector result = left;

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

result[i] -= right[i];

}

return result;

}

double get\_matrix\_norm(const std::vector<double> &matrix) {

return \*std::max\_element(matrix.begin(), matrix.end());

}

std::vector<std::vector<double> > get\_gaussian\_elimination(std::vector<std::vector<double> > &m) {

auto matrix = m;

const int n = static\_cast<int>(matrix.size());

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

int maxRow = i;

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

if (std::abs(matrix[k][i]) > std::abs(matrix[maxRow][i])) {

maxRow = k;

}

}

if (maxRow != i) {

std::swap(matrix[i], matrix[maxRow]);

}

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

const double factor = matrix[k][i] / matrix[i][i];

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

matrix[k][j] -= factor \* matrix[i][j];

}

}

}

for (int i = n - 1; i >= 0; --i) {

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

matrix[i][n] -= matrix[i][j] \* matrix[j][n];

}

matrix[i][n] /= matrix[i][i];

}

return matrix;

}

std::vector<std::vector<double> > multiply\_matrices(const std::vector<std::vector<double> > &matrix1,

const std::vector<std::vector<double> > &matrix2) {

const int rows1 = static\_cast<int>(matrix1.size());

const int cols1 = static\_cast<int>(matrix1[0].size());

const int cols2 = static\_cast<int>(matrix2[0].size());

std::vector result(rows1, std::vector(cols2, 0.0));

for (int i = 0; i < rows1; ++i) {

for (int j = 0; j < cols2; ++j) {

for (int k = 0; k < cols1; ++k) {

result[i][j] += matrix1[i][k] \* matrix2[k][j];

}

}

}

return result;

}

std::vector<double> multiply\_matrices(const std::vector<std::vector<double> > &matrix1,

const std::vector<double> &matrix2) {

const int rows1 = static\_cast<int>(matrix1.size());

const int cols1 = static\_cast<int>(matrix1.size());

const int cols2 = static\_cast<int>(matrix2.size());

std::vector result(rows1, 0.0);

for (int i = 0; i < cols2; i++) {

for (int j = 0; j < rows1; j++) {

result[i] += matrix1[i][j] \* matrix2[j];

}

}

return result;

}

std::vector<std::vector<double> > get\_left\_matrix(double p) {

return {

{p - 29, 6, -6, -4, -3, -8, -5, 5},

{6, -13, -3, 5, 4, 3, 1, 7},

{5, -5, -1, 7, 2, 0, 7, 1},

{5, -5, 5, 6, 4, -7, 4, 0},

{4, 4, 7, -4, 9, -8, -8, -4},

{-4, 5, -4, 1, 0, 12, 0, 6},

{-3, -2, -4, 2, -8, -3, 16, 4},

{7, 5, 0, 2, 0, -6, 8, -12}

};

}

std::vector<double> get\_right\_matrix(double p) {

return {

4 \* p - 175,

133,

110,

112,

17,

32,

13,

-18

};

}

int main() {

dimkashelk::Solve solve;

const auto numbers = {1.0, 0.1, 0.01, 0.0001, 0.000001};

for (const auto number: numbers) {

std::cout << "p = " << number << ": \n";

std::cout << "Part one\t\t\tPart two\n";

auto left = get\_left\_matrix(number);

auto right = get\_right\_matrix(number);

solve(left, right);

auto res1 = solve.get\_result();

const auto cond1 = solve.get\_cond();

auto gaussian = get\_gaussian\_elimination(left);

auto new\_left = multiply\_matrices(gaussian, left);

auto new\_right = multiply\_matrices(gaussian, right);

solve(new\_left, new\_right);

auto res2 = solve.get\_result();

const auto cond2 = solve.get\_cond();

for (int i = 0; i < res1.size(); i++) {

std::cout << "\tx" << i << ": " << res1[i] << "\t\t\t" << res2[i] << '\n';

}

std::cout << "cond: \t" << cond1 << "\t\t" << cond2 << "\n";

const auto res3 = get\_matrix\_norm(get\_difference\_of\_matrix(res1, res2)) / get\_matrix\_norm(res1);

std::cout << "norm:\t" << res3 << "\n";

std::cout << "\n";

}

return 0;

}