Московский Авиационный Институт (национальный исследовательский университет)

**Факультет прикладной математики и физики**

Кафедра Вычислительной математики и программирования

**Лабораторная работа по курсу**

**«Численные методы»**

Студент: Богачев П.С.

Группа: М8О-402Б-20

Руководитель: Пивоваров Д. Е.

Москва 2023

**Лабораторная работа №7**

Решить краевую задачу для дифференциального уравнения эллиптического типа. Аппроксимацию уравнения произвести с использованием центрально-разностной схемы. Для решения дискретного аналога применить следующие методы: метод простых итераций (метод Либмана), метод Зейделя, метод простых итераций с верхней релаксацией. Вычислить погрешность численного решения путем сравнения результатов с приведенным в задании аналитическим решением . Исследовать зависимость погрешности от сеточных параметров .

Вариант №3

,



,

.

Аналитическое решение: .

**Код программы**

#include <iostream>

#include <fstream>

#include <functional>

#include <vector>

#include <cmath>

double phi0(double y) {

return cos(y);

}

double phi1(double y) {

return exp(1.) \* cos(y);

}

double psi0(double x) {

return exp(x);

}

double psi1(double x) {

return 0.;

}

double solution(double x, double y) {

return exp(x) \* cos(y);

}

double L2(std::vector<double>& x, std::vector<double>& y) {

int n = x.size();

double l2 = 0.;

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

l2 += (x[i] - y[i]) \* (x[i] - y[i]);

}

return sqrt(l2);

}

std::vector<double> multiplication(std::vector<std::vector<double>>& A, std::vector<double>& b) {

int n = A.size();

std::vector<double> answer(n, 0.);

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

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

answer[i] += A[i][j] \* b[j];

}

}

return answer;

}

void addition(std::vector<double>& x, std::vector<double>& y) {

int n = x.size();

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

x[i] += y[i];

}

}

std::vector<std::vector<double>> analytical\_solution(double x\_begin, double x\_end, double y\_begin,

double y\_end, double hx, double hy) {

int length\_x = static\_cast<int>((x\_end - x\_begin) / hx) + 1;

int length\_y = static\_cast<int>((y\_end - y\_begin) / hy) + 1;

std::vector<double> x(length\_x), y(length\_y);

x[0] = x\_begin;

for (int i = 1; i < length\_x; ++i) {

x[i] = x[i - 1] + hx;

}

y[0] = y\_begin;

for (int i = 1; i < length\_y; ++i) {

y[i] = y[i - 1] + hy;

}

std::vector<std::vector<double>> result(length\_x, std::vector<double>(length\_y));

for (int i = 0; i < length\_x; ++i) {

for (int j = 0; j < length\_y; ++j) {

result[i][j] = solution(x[i], y[j]);

}

}

return result;

}

std::pair<std::vector<std::vector<double>>, int> finite\_difference\_method(double x\_begin, double x\_end, double y\_begin,

double y\_end, double hx, double hy, double epsilon,

std::function<std::pair<std::vector<double>, int>(std::vector<std::vector<double>>&, std::vector<double>&, double)> method) {

int length\_x = static\_cast<int>((x\_end - x\_begin) / hx) + 1;

int length\_y = static\_cast<int>((y\_end - y\_begin) / hy) + 1;

std::vector<double> x(length\_x), y(length\_y);

x[0] = x\_begin;

for (int i = 1; i < length\_x; ++i) {

x[i] = x[i - 1] + hx;

}

y[0] = y\_begin;

for (int i = 1; i < length\_y; ++i) {

y[i] = y[i - 1] + hy;

}

std::vector<std::vector<double>> result(length\_x, std::vector<double>(length\_y));

for (int i = 0; i < length\_x; ++i) {

result[i][0] = psi0(x[i]);

result[i][length\_y - 1] = psi1(x[i]);

}

for (int i = 0; i < length\_y; ++i) {

result[0][i] = phi0(y[i]);

result[length\_x - 1][i] = phi1(y[i]);

}

std::vector<std::vector<int>> mapping(length\_x, std::vector<int>(length\_y));

int current\_equation = 0;

for (int i = 1; i < length\_x - 1; ++i) {

for (int j = 1; j < length\_y - 1; ++j) {

mapping[i][j] = current\_equation++;

}

}

int number\_of\_equations = (length\_x - 2) \* (length\_y - 2);

std::vector<std::vector<double>> A(number\_of\_equations, std::vector<double>(number\_of\_equations, 0.));

std::vector<double> b(number\_of\_equations, 0.);

for (int i = 1; i < length\_x - 1; ++i) {

for (int j = 1; j < length\_y - 1; ++j) {

current\_equation = mapping[i][j];

A[current\_equation][mapping[i][j]] = 1;

if (j == 1) {

b[current\_equation] += psi0(x[i]) \* hx \* hx / ((hx \* hx + hy \* hy) \* 2.);

}

else {

A[current\_equation][mapping[i][j - 1]] = -hx \* hx / ((hx \* hx + hy \* hy) \* 2.);

}

if (j == length\_y - 2) {

b[current\_equation] += psi1(x[i]) \* hx \* hx / ((hx \* hx + hy \* hy) \* 2.);

}

else {

A[current\_equation][mapping[i][j + 1]] = -hx \* hx / ((hx \* hx + hy \* hy) \* 2.);

}

if (i == 1) {

b[current\_equation] += phi0(y[j]) \* hy \* hy / ((hx \* hx + hy \* hy) \* 2.);

}

else {

A[current\_equation][mapping[i - 1][j]] = -hy \* hy / ((hx \* hx + hy \* hy) \* 2.);

}

if (i == length\_x - 2) {

b[current\_equation] += phi1(y[j]) \* hy \* hy / ((hx \* hx + hy \* hy) \* 2.);

}

else {

A[current\_equation][mapping[i + 1][j]] = -hy \* hy / ((hx \* hx + hy \* hy) \* 2.);

}

}

}

auto [answer, iterations] = method(A, b, epsilon);

for (int i = 1; i < length\_x - 1; ++i) {

for (int j = 1; j < length\_y - 1; ++j) {

result[i][j] = answer[mapping[i][j]];

}

}

return std::make\_pair(result, iterations);

}

std::pair<std::vector<double>, int> iterative(std::vector<std::vector<double>>& A, std::vector<double>& b, double epsilon) {

int n = A.size();

std::vector<std::vector<double>> alpha(n, std::vector<double>(n, 0.));

std::vector<double> beta(n, 0.);

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

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

if (i != j) {

alpha[i][j] -= A[i][j] / A[i][i];

}

}

beta[i] = b[i] / A[i][i];

}

int iterations = 0;

std::vector<double> current\_x(beta);

bool converge = false;

while (!converge) {

std::vector<double> previous\_x(current\_x);

current\_x = multiplication(alpha, previous\_x);

addition(current\_x, beta);

++iterations;

double l2 = L2(previous\_x, current\_x);

converge = l2 <= epsilon;

std::cout << "Iterative method. Iteration: " << iterations << ", l2: " << l2 << "\n";

}

return std::make\_pair(current\_x, iterations);

}

std::vector<double> seidel\_multiplication(std::vector<std::vector<double>>& alpha, std::vector<double>& x, std::vector<double>& beta) {

int n = alpha.size(), m = alpha[0].size();

std::vector<double> result(x);

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

result[i] = beta[i];

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

result[i] += alpha[i][j] \* result[j];

}

}

return result;

}

std::pair<std::vector<double>, int> seidel(std::vector<std::vector<double>>& A, std::vector<double>& b, double epsilon) {

int n = A.size();

std::vector<std::vector<double>> alpha(n, std::vector<double>(n, 0.));

std::vector<double> beta(n, 0.);

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

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

if (i != j) {

alpha[i][j] -= A[i][j] / A[i][i];

}

}

beta[i] = b[i] / A[i][i];

}

int iterations = 0;

std::vector<double> current\_x(beta);

bool converge = false;

while (!converge) {

std::vector<double> previous\_x(current\_x);

current\_x = seidel\_multiplication(alpha, previous\_x, beta);

++iterations;

double l2 = L2(previous\_x, current\_x);

converge = l2 <= epsilon;

std::cout << "Seidel method. Iteration: " << iterations << ", l2: " << l2 << "\n";

}

return std::make\_pair(current\_x, iterations);

}

double w = 1.5;

void relaxations\_additon(std::vector<double>& x, std::vector<double>& y) {

int n = x.size();

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

x[i] = x[i] \* w + y[i] \* (1. - w);

}

}

std::pair<std::vector<double>, int> relaxations(std::vector<std::vector<double>>& A, std::vector<double>& b, double epsilon) {

int n = A.size();

std::vector<std::vector<double>> alpha(n, std::vector<double>(n, 0.));

std::vector<double> beta(n, 0.);

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

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

if (i != j) {

alpha[i][j] -= A[i][j] / A[i][i];

}

}

beta[i] = b[i] / A[i][i];

}

int iterations = 0;

std::vector<double> current\_x(beta);

bool converge = false;

while (!converge) {

std::vector<double> previous\_x(current\_x);

current\_x = seidel\_multiplication(alpha, previous\_x, beta);

relaxations\_additon(current\_x, previous\_x);

++iterations;

double l2 = L2(previous\_x, current\_x);

converge = l2 <= epsilon;

std::cout << "Relaxations method. Iteration: " << iterations << ", l2: " << l2 << "\n";

}

return std::make\_pair(current\_x, iterations);

}

double max\_abs\_error(std::vector<std::vector<double>>& A, std::vector<std::vector<double>>& B) {

int n = A.size(), m = A[0].size();

double max = 0.;

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

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

max = std::max(max, std::abs(A[i][j] - B[i][j]));

}

}

return max;

}

double mean\_abs\_error(std::vector<std::vector<double>>& A, std::vector<std::vector<double>>& B) {

int n = A.size(), m = A[0].size();

double mean = 0., prod = static\_cast<double>(n \* m);

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

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

mean += std::abs(A[i][j] - B[i][j]) / prod;

}

}

return mean;

}

void output\_to\_file(std::string filepath, const std::vector<std::vector<double>>& arr) {

std::ofstream fout(filepath);

int n = arr.size(), m = arr[0].size();

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

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

fout << arr[i][j] << " ";

}

fout << "\n";

}

fout.close();

}

int main()

{

double x\_begin = 0., x\_end = 1., y\_begin = 0., y\_end = acos(-1) / 2., hx = 0.01, hy = 0.01, epsilon = 1e-3;

std::vector<std::vector<double>> as = analytical\_solution(x\_begin, x\_end, y\_begin, y\_end, hx, hy);

output\_to\_file("analytical\_solution.txt", as);

auto [is, ii] = finite\_difference\_method(x\_begin, x\_end, y\_begin, y\_end, hx, hy, epsilon, iterative);

std::cout << "Iterative method took " << ii << " iterations until convergence\n";

std::cout << "Max abs error between analytical solution and iterative method solution: " << max\_abs\_error(as, is) << "\n";

std::cout << "Mean abs error between analytical solution and iterative method solution: " << mean\_abs\_error(as, is) << "\n";

output\_to\_file("iterative\_method.txt", is);

auto [ss, si] = finite\_difference\_method(x\_begin, x\_end, y\_begin, y\_end, hx, hy, epsilon, seidel);

std::cout << "Seidel method took " << si << " iterations until convergence\n";

std::cout << "Max abs error between analytical solution and Seidel method solution: " << max\_abs\_error(as, ss) << "\n";

std::cout << "Mean abs error between analytical solution and Seidel method solution: " << mean\_abs\_error(as, ss) << "\n";

output\_to\_file("seidel\_method.txt", ss);

auto [rs, ri] = finite\_difference\_method(x\_begin, x\_end, y\_begin, y\_end, hx, hy, epsilon, relaxations);

std::cout << "Relaxations method took " << ri << " iterations until convergence\n";

std::cout << "Max abs error between analytical solution and relaxations method solution: " << max\_abs\_error(as, rs) << "\n";

std::cout << "Mean abs error between analytical solution and relaxations method solution: " << mean\_abs\_error(as, rs) << "\n";

output\_to\_file("relaxations\_method.txt", rs);

return 0;

}

Вывод программы

График функций:

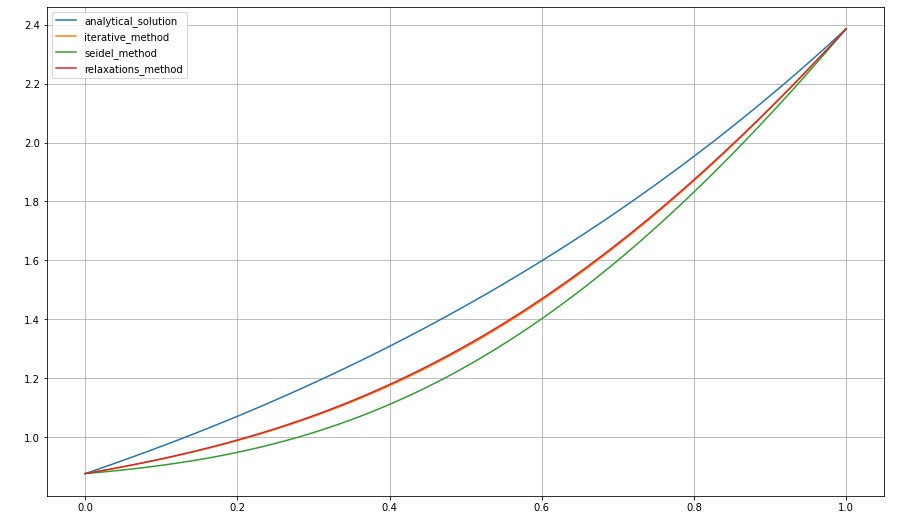
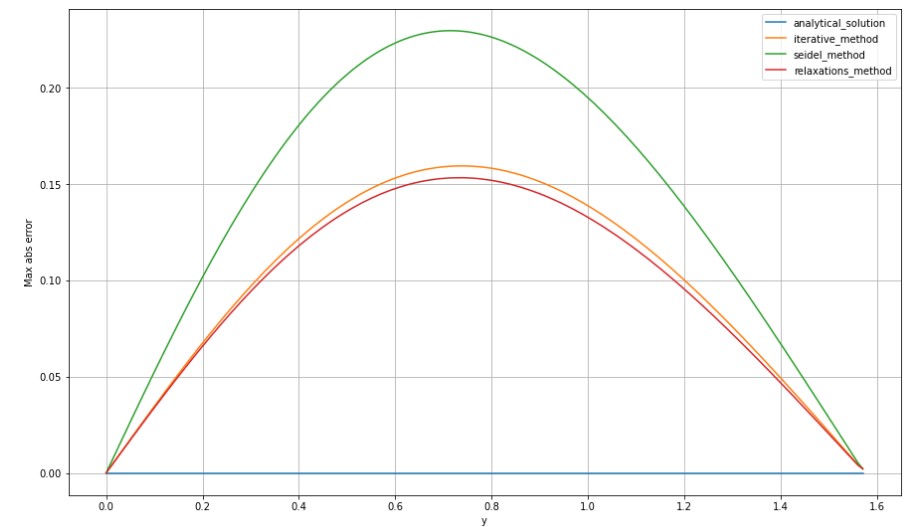


График ошибок:



**Вывод**

В лабораторной работе решена краевая задача для дифференциального уравнение эллиптического типа. Произведена аппроксимация с использованием центрально-разностной схемы. Реализованы методы дискретного аналога: метод простых итераций (метод Либмана), метод Зейделя, метод простых итераций с верхней релаксацией. Вычислена погрешность решения.