Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Курсовой проект
по курсу «Численные методы»

Студент: А. А. Садаков
Группа: М8О-406Б-19

Москва, 2023

Лабораторная работа №7

Задача: 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 $U(x, y)$. Исследовать зависимость погрешности от сеточных параметров h_x, h_y.

Вариант: 5

Краевая задача:

$$\begin{cases} \dfrac{\partial^2 u}{\partial x^2} + \dfrac{\partial^2 u}{\partial y^2} = -u, \\ u_x(0, y) = \cos(y), \\ u_x(1, y) - u(1, y) = 0, \\ u(x, 0) = x, \\ u(x, \dfrac{\pi}{2}) = 0 \end{cases}$$

Точное решение:

$$U(x, y) = x \cos(y)$$

Вывод программы для шагов $h_x = 0.01$ и $h_y = 0.01$:

=====7.1=====

Введите начально-краевую задачу:

0: uxx + uyy = 0 - u

1: ux(0) = cos(y)

2: ux(1) - u(1) = 0

3: u(0) = x

4: u(1.57) = 0

Введите размер шага для "x"и для "y":

Введите функцию для сравнения:

Размер таблицы функции U: 100x157

Средняя погрешность метода простых итераций: 0.23

Средняя погрешность метода Зейделя: 0.20

Средняя погрешность метода Либмана: 0.25

Исходный код

7-1.cpp:

```cpp
#include "7-1.hpp"

double Point2Order1 (const std::vector<double> &coeff, double h, double u1, double f,
    uint64_t i) {
    double alpha = coeff[0], beta = coeff[1];
    double ans = 0;
    if (i == 0) {
        ans += f - alpha * u1 / h;
        ans /= beta - alpha / h;
    } else {
        ans += f + alpha * u1 / h;
        ans /= beta + alpha / h;
    }
    return ans;
}

double Point2Order2 (const std::vector<double> &ux, const std::vector<double> &coeff,
    double h, double t, double u0, double u1, double f, uint64_t i) {
    double alpha = ux[0], beta = ux[1];
    double a = coeff[0], b = coeff[1], c = coeff[2];
    double ans = 0;
    if (i == 0) {
        ans += h / t * u0 - f * (2 * a - b * h) / alpha + 2 * u1 * a / h;
        ans /= 2 * a / h + h / t - c * h - (beta / alpha) * (2 * a - b * h);
    } else {
        ans += h / t * u0 + f * (2 * a + b * h) / alpha + 2 * u1 * a / h;
        ans /= 2 * a / h + h / t - c * h + (beta / alpha) * (2 * a + b * h);
    }
    return ans;
}

double Point3Order2 (const std::vector<double> &coeff, double h, double u1, double u2,
    double f, uint64_t i) {
    double alpha = coeff[0], beta = coeff[1];
    double ans = 0;
    if (i == 0) {
        ans += f - alpha * (4 * u1 - u2) / (2 * h);
        ans /= beta - 3 * alpha / (2 * h);
    } else {
        ans += f + alpha * (4 * u1 - u2) / (2 * h);
        ans /= beta + 3 * alpha / (2 * h);
    }
    return ans;
}

```

```
43 void borderApprox (std::vector<std::vector<double>> &u, const std::vector<std::vector<
       double>> &ux, const std::vector<double> &coeff, double xh, double yh, const std::
       vector<ApproxLevel> &border) {
44     uint64_t left = 0, right = u[0].size() - 1, down = 0, up = u.size() - 1;
45     for (uint64_t j = 1; j < u.size(); ++j) {
46         switch (border[0]) {
47             case ApproxLevel::POINT2_ORDER1:
48                 u[j][left] = Point2Order1(ux[0], xh, u[j][left + 1], u[j][left], 0);
49                 break;
50             case ApproxLevel::POINT2_ORDER2:
51                 u[j][left] = Point2Order2(ux[0], coeff, xh, yh, u[j - 1][left], u[j][
                       left + 1], u[j][left], 0);
52                 break;
53             case ApproxLevel::POINT3_ORDER2:
54                 u[j][left] = Point3Order2(ux[0], xh, u[j][left + 1], u[j][left + 2], u[j
                       ][left], 0);
55                 break;
56             default:
57                 break;
58         }
59     }
60     for (uint64_t j = 1; j < u.size(); ++j) {
61         switch (border[1]) {
62             case ApproxLevel::POINT2_ORDER1:
63                 u[j][right] = Point2Order1(ux[1], xh, u[j][right - 1], u[j][right], 0);
64                 break;
65             case ApproxLevel::POINT2_ORDER2:
66                 u[j][right] = Point2Order2(ux[1], coeff, xh, yh, u[j - 1][right], u[j][
                       right - 1], u[j][right], 0);
67                 break;
68             case ApproxLevel::POINT3_ORDER2:
69                 u[j][right] = Point3Order2(ux[1], xh, u[j][right - 1], u[j][right - 2],
                       u[j][right], 0);
70                 break;
71             default:
72                 break;
73         }
74     }
75     for (uint64_t j = 1; j < u[0].size(); ++j) {
76         switch (border[2]) {
77             case ApproxLevel::POINT2_ORDER1:
78                 u[down][j] = Point2Order1(ux[2], xh, u[down + 1][j], u[down][j], 0);
79                 break;
80             case ApproxLevel::POINT2_ORDER2:
81                 u[down][j] = Point2Order2(ux[2], coeff, xh, yh, u[down][j - 1], u[down +
                        1][j], u[down][j], 0);
82                 break;
83             case ApproxLevel::POINT3_ORDER2:
```

```
84                      u[down][j] = Point3Order2(ux[2], xh, u[down + 1][j], u[down + 2][j], u[
                            down][j], 0);
85                      break;
86                  default:
87                      break;
88              }
89          }
90          for (uint64_t j = 1; j < u[0].size(); ++j) {
91              switch (border[3]) {
92                  case ApproxLevel::POINT2_ORDER1:
93                      u[up][j] = Point2Order1(ux[3], xh, u[up - 1][j], u[up][j], 0);
94                      break;
95                  case ApproxLevel::POINT2_ORDER2:
96                      u[up][j] = Point2Order2(ux[3], coeff, xh, yh, u[up][j - 1], u[up - 1][j
                            ], u[up][j], 0);
97                      break;
98                  case ApproxLevel::POINT3_ORDER2:
99                      u[up][j] = Point3Order2(ux[3], xh, u[up - 1][j], u[up - 2][j], u[up][j],
                             0);
100                     break;
101                 default:
102                     break;
103             }
104         }
105 }
106
107 double norma (const std::vector<std::vector<double>> &v1, const std::vector<std::vector<
        double>> &v2) {
108     double ans = 0;
109     uint64_t size1 = std::min(v1.size(), v2.size());
110     uint64_t size2 = std::min(v1[0].size(), v2[0].size());
111     for (uint64_t i = 0; i < size1; ++i) {
112         for (uint64_t j = 0; j < size2; ++j) {
113             ans = std::max(ans, std::abs(v1[i][j] - v2[i][j]));
114         }
115     }
116     return ans;
117 }
118
119 void UInit (std::vector<std::vector<double>> &u) {
120     uint64_t height = u.size(), width = u[0].size();
121     for (uint64_t i = 1; i < height - 1; ++i) {
122         double left = u[i].front(), right = u[i].back();
123         double step = (right - left) / width;
124         for (uint64_t j = 1; j < width - 1; ++j) {
125             u[i][j] = left + step * j;
126         }
127     }
```

```
}

void SIIteration (std::vector<std::vector<double>> &u, double xh, double yh, const std::
    vector<double> &coeff, const std::function<double(double, double)> &f, double eps,
    SIMethod method) {
    uint64_t n = u[0].size() - 2;
    Matrix<double> matrix(n, n);
    std::vector<double> ans(n);
    double a = coeff[0], b = coeff[1], c = coeff[2];
        for (uint64_t i = 1; i < u.size() - 1; ++i) {

            matrix(0, 0) = a*xh*yh*yh + b*xh*xh*yh - c*xh*xh*yh*yh - 2*yh*yh - 2*xh*xh;
            matrix(0, 1) = yh*yh - a*xh*yh*yh;
            ans[0] = -yh*yh*u[i][0] + (b*xh*xh*yh - xh*xh)*u[i + 1][1] - xh*xh*u[i - 1][1] +
                 xh*xh*yh*yh*f(0, i*yh);
            for (uint64_t j = 1; j < n - 1; ++j) {
                matrix(j, j - 1) = yh*yh;
                matrix(j, j) = a*xh*yh*yh + b*xh*xh*yh - c*xh*xh*yh*yh - 2*yh*yh - 2*xh*xh;
                matrix(j, j + 1) = yh*yh - a*xh*yh*yh;
                ans[j] = (b*xh*xh*yh - xh*xh)*u[i + 1][j + 1] - xh*xh*u[i - 1][j + 1] + xh*
                    xh*yh*yh*f(j*xh, i*yh);
            }
            matrix(n - 1, n - 2) = yh*yh;
            matrix(n - 1, n - 1) = a*xh*yh*yh + b*xh*xh*yh - c*xh*xh*yh*yh - 2*yh*yh - 2*xh*
                xh;
            ans[n - 1] = -(yh*yh - a*xh*yh*yh)*u[i][n + 1] + (b*xh*xh*yh - xh*xh)*u[i + 1][n
                ] - xh*xh*u[i - 1][n] + xh*xh*yh*yh*f(n*xh, i*yh);
            ans = SISolveSLAE(matrix, ans, eps, method);
            for (uint64_t j = 0; j < ans.size(); ++j) {
                u[i][j + 1] = ans[j];
            }
        }
}

void SI (std::vector<std::vector<double>> &u, const std::vector<std::vector<double>> &ux,
    const std::vector<double> &coeff, double xh, double yh, const std::function<double(
    double, double)> &f, const std::vector<ApproxLevel> &border, double eps, SIMethod
    method) {
    UInit(u);
    SIIteration(u, xh, yh, coeff, f, eps, method);
    borderApprox(u, ux, coeff, xh, yh, border);
    auto prev = u;
    do {
        prev = u;
        SIIteration(u, xh, yh, coeff, f, eps, method);
        borderApprox(u, ux, coeff, xh, yh, border);
    } while (norma(u, prev) > eps);
}
```

```
167
168 double LibmanIteration (std::vector<std::vector<double>> &u, double xh, double yh, const
        std::vector<double> &coeff, const std::function<double(double, double)> &f) {
169     double maxAbs = 0;
170     double a = coeff[0], b = coeff[1], c = coeff[2];
171     uint64_t height = u.size(), width = u[0].size();
172     for (uint64_t i = 1; i < height - 1; ++i) {
173         for (uint64_t j = 1; j < width - 1; ++j) {
174             double currU = 0;
175             currU += u[i + 1][j] * (a*xh*yh*yh - yh*yh);
176             currU -= u[i - 1][j] * yh*yh;
177             currU += u[i][j + 1] * (b*xh*xh*yh - xh*xh);
178             currU -= u[i][j - 1] * xh*xh;
179             currU += xh*xh*yh*yh*f(i*xh, j*yh);
180             currU /= a*xh*yh*yh + b*xh*xh*yh - c*xh*xh*yh*yh - 2*yh*yh - 2*xh*xh;
181             maxAbs = std::max(maxAbs, std::abs(u[i][j] - currU));
182             u[i][j] = currU;
183         }
184     }
185     return maxAbs;
186 }
187
188 void Libman (std::vector<std::vector<double>> &u, const std::vector<std::vector<double>> &
        ux, const std::vector<double> &coeff, double xh, double yh, const std::function<double
        (double, double)> &f, const std::vector<ApproxLevel> &border, double eps) {
189     UInit(u);
190     LibmanIteration(u, xh, yh, coeff, f);
191     borderApprox(u, ux, coeff, xh, yh, border);
192     auto prev = u;
193     do {
194         prev = u;
195         LibmanIteration(u, xh, yh, coeff, f);
196         borderApprox(u, ux, coeff, xh, yh, border);
197     } while (norma(u, prev) > eps);
198 }
199
200 std::vector<std::vector<double>> SolveIBVP (const Task &task, double xh, double yh, Method
         method) {
201     double l1 = task.X[1], l2 = task.Y[1];
202     std::vector<std::vector<double>> u(uint64_t(l2 / yh), std::vector<double>(uint64_t(l1 /
          xh), 0));
203     std::vector<ApproxLevel> border(4, ApproxLevel::NONE);
204     for (uint64_t i = 0; i < 4; ++i) {
205         if (task.ux[i][0] != 0) {
206             border[i] = ApproxLevel::POINT2_ORDER2;
207         }
208     }
209     auto f = [&] (double x, double y) -> double {
```

```
        return task.trees[0]({0, 0, 0, x, y});
    };
    auto fx0 = [&] (double y) -> double {
        return task.trees[1](y);
    };
    auto fxl = [&] (double y) -> double {
        return task.trees[2](y);
    };
    auto fy0 = [&] (double x) -> double {
        return task.trees[3](x);
    };
    auto fyl = [&] (double x) -> double {
        return task.trees[4](x);
    };
    for (uint64_t i = 0; i < u[0].size(); ++i) {
        u.front()[i] = fy0(i * xh);
        u.back()[i] = fyl(i * xh);
    }
    for (uint64_t i = 1; i < u.size() - 1; ++i) {
        u[i].front() = fx0(i * yh);
        u[i].back() = fxl(i * yh);
    }
    switch (method) {
        case Method::YAKOBI:
            SI(u, task.ux, task.coeff, xh, yh, f, border, 0.01, SIMethod::SI_YAKOBI_METHOD);
            break;
        case Method::ZEIDEL:
            SI(u, task.ux, task.coeff, xh, yh, f, border, 0.01, SIMethod::SI_ZEIDEL_METHOD);
            break;
        case Method::LIBMAN:
            Libman(u, task.ux, task.coeff, xh, yh, f, border, 0.01);
            break;
        default:
            break;
    }
    return u;
}
```

Выводы

Точность решения падает с уменьшением шага h_x и мало зависит от размера шага h_y. При размерах сетки более 200 для h_x время выполнения сильно падает, при этом размер сетки для h_y не сильно влияет на производительность.