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 №8

Задача: Используя схемы переменных направлений и дробных шагов, решить двумерную начально-краевую задачу для дифференциального уравнения параболического типа. В различные моменты времени вычислить погрешность численного решения путем сравнения результатов с приведенным в задании аналитическим решением $U(x, y, t)$. Исследовать зависимость погрешности от сеточных параметров τ, h_x, h_y.

Вариант: 5

Двумерная начально-краевая задача:

$$\begin{cases} \dfrac{\partial u}{\partial t} = a\dfrac{\partial^2 u}{\partial x^2} + a\dfrac{\partial^2 u}{\partial y^2}, a > 0 \\ u(0, y, t) = \sinh(x)\exp(-3at), \\ u(\dfrac{\pi}{2}, y, t) = -\sinh(x)\exp(-3at), \\ u_y(x, 0, t) = \cos(2x)\exp(-3at), \\ u(x, \ln(2), t) = \dfrac{3}{4}\cos(2x)\exp(-3at), \\ u(x, y, 0) = \cos(2x)\sinh(y) \end{cases}$$

Точное решение:

$$U(x, y, t) = \cos(2x)\sinh(y)\exp(-3at)$$

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

=====8.1=====

Введите начально-краевую задачу:

0: ut = uxx + uyy

1: u(0) = sinh(y)*exp(-3*t)

2: u(pi/2) = 0-sinh(y)*exp(-3*t)

3: uy(0) = cos(2*x)*exp(-3*t)

4: u(ln(2)) = 3/4 * cos(2*x)*exp(-3*t)

5: u(0) = cos(2*x) * sinh(y)

Введите размер шага для "x "y"и "t":

Введите функцию для сравнения:

Размер таблицы функции U: 157x69x500

500 69 157

Средняя погрешность метода переменных направлений: 0.02

Средняя погрешность метода дробных шагов: 0.08

Исходный код

8-1.cpp:

```
#include "8-1.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;
}

```

```cpp
void VariableDirectionsIteration (std::vector<std::vector<std::vector<double>>> &u, const std::vector<std::vector<double>> &ux, const std::vector<double> &coeff, double xh, double yh, double th, const std::function<double(double, double, double)> &f, uint64_t i) {
    double a = coeff[0], b = coeff[2];
    double xcoeff = a * th / (2 * xh * xh);
    double ycoeff = b * th / (2 * yh * yh);
    auto tmp = u[i];
    {
        uint64_t n = u[0][0].size() - 2;
        Matrix<double> matrix(n, n);
        std::vector<double> ans(n);
        for (uint64_t j = 1; j < u[0].size() - 1; ++j) {
            matrix(0, 0) = 1 + 2 * xcoeff;
            matrix(0, 1) = -xcoeff;
            ans[0] = u[i][j][0] + xcoeff*u[i + 1][j][0] + ycoeff*(u[i][j + 1][0] - 2*u[i][j][0] + u[i][j - 1][0]) + (th/2)*f(0, j * yh, i*th + th / 2);
            for (uint64_t k = 1; k < n - 1; ++k) {
                matrix(k, k - 1) = -xcoeff;
                matrix(k, k) = 1 + 2 * xcoeff;
                matrix(k, k + 1) = -xcoeff;
                ans[k] = u[i][j][k] + ycoeff*(u[i][j + 1][k] - 2*u[i][j][k] + u[i][j - 1][k]) + (th/2)*f(k*xh, j*yh, i * th + th/2);
            }
            matrix(n - 1, n - 2) = -xcoeff;
            matrix(n - 1, n - 1) = 1 + 2 * xcoeff;
            ans[n - 1] = u[i][j][n - 1] + xcoeff*u[i + 1][j][n - 1] + ycoeff*(u[i][j + 1][n - 1] - 2*u[i][j][n - 1] + u[i][j - 1][n - 1]) + (th/2)*f(n*xh, j*yh, i*th + th/2);
            ans = RUNsolveSLAE(matrix, ans);
            for (uint64_t k = 0; k < ans.size(); ++k) {
                tmp[j][k + 1] = ans[k];
            }
        }
    }
    {
        uint64_t n = u[0].size() - 2;
        Matrix<double> matrix(n, n);
        std::vector<double> ans(n);
        for (uint64_t j = 1; j < u[0][0].size() - 1; ++j) {
            matrix(0, 0) = 1 + 2 * ycoeff;
            matrix(0, 1) = -ycoeff;
            ans[0] = u[i][0][j] + ycoeff*u[i + 1][0][j] + xcoeff*(u[i][0][j + 1] - 2*u[i][0][j] + u[i][0][j - 1]) + (th/2)*f(j*xh, 0, i*th + th / 2);
            for (uint64_t k = 1; k < n - 1; ++k) {
                matrix(k, k - 1) = -ycoeff;
                matrix(k, k) = 1 + 2 * ycoeff;
                matrix(k, k + 1) = -ycoeff;
```

```
                ans[k] = u[i][k][j] + xcoeff*(u[i][k][j + 1] - 2*u[i][k][j] + u[i][k][j - 1]) + (th/2)*f(j*xh, k*yh, i*th + th/2);
            }
            matrix(n - 1, n - 2) = -ycoeff;
            matrix(n - 1, n - 1) = 1 + 2 * ycoeff;
            ans[n - 1] = u[i][n - 1][j] + ycoeff*u[i + 1][n - 1][j] + xcoeff*(u[i][n - 1][j + 1] - 2*u[i][n - 1][j] + u[i][n - 1][j - 1]) + (th/2)*f(j*xh, n*yh, i*th + th/2);
            ans = RUNsolveSLAE(matrix, ans);
            for (uint64_t k = 0; k < ans.size(); ++k) {
                u[0][k + 1][j] = ans[k];
            }
        }
    }
}

void VariableDirections (std::vector<std::vector<std::vector<double>>> &u, const std::vector<std::vector<double>> &ux, const std::vector<double> &coeff, double xh, double yh, double th, const std::function<double(double, double, double)> &f) {
    for (uint64_t i = 0; i < u.size() - 1; ++i) {
        VariableDirectionsIteration(u, ux, coeff, xh, yh, th, f, i);
    }
}

void FractionalStepsIteration (std::vector<std::vector<std::vector<double>>> &u, const std::vector<std::vector<double>> &ux, const std::vector<double> &coeff, double xh, double yh, double th, const std::function<double(double, double, double)> &f, uint64_t i) {
    double a = coeff[0], b = coeff[2];
    double xcoeff = a * th / (xh * xh);
    double ycoeff = b * th / (yh * yh);
    auto tmp = u[i];
    {
        uint64_t n = u[0][0].size() - 2;
        Matrix<double> matrix(n, n);
        std::vector<double> ans(n);
        for (uint64_t j = 1; j < u[0].size() - 1; ++j) {
            matrix(0, 0) = 1 + 2 * xcoeff;
            matrix(0, 1) = -xcoeff;
            ans[0] = u[i][j][0] + xcoeff*u[i + 1][j][0] + (th/2)*f(0, j * yh, i*th + th / 2);
            for (uint64_t k = 1; k < n - 1; ++k) {
                matrix(k, k - 1) = -xcoeff;
                matrix(k, k) = 1 + 2 * xcoeff;
                matrix(k, k + 1) = -xcoeff;
                ans[k] = u[i][j][k] + (th/2)*f(k*xh, j*yh, i * th + th/2);
            }
            matrix(n - 1, n - 2) = -xcoeff;
            matrix(n - 1, n - 1) = 1 + 2 * xcoeff;
```

```
            ans[n - 1] = u[i][j][n - 1] + xcoeff*u[i + 1][j][n - 1] + (th/2)*f(n*xh, j*yh, i
                *th + th/2);
            ans = RUNsolveSLAE(matrix, ans);
            for (uint64_t k = 0; k < ans.size(); ++k) {
                tmp[j][k + 1] = ans[k];
            }
        }
    }
    {
        uint64_t n = u[0].size() - 2;
        Matrix<double> matrix(n, n);
        std::vector<double> ans(n);
        for (uint64_t j = 1; j < u[0][0].size() - 1; ++j) {
            matrix(0, 0) = 1 + 2 * ycoeff;
            matrix(0, 1) = -ycoeff;
            ans[0] = u[i][0][j] + ycoeff*u[i + 1][0][j] + (th/2)*f(j*xh, 0, i*th + th / 2);
            for (uint64_t k = 1; k < n - 1; ++k) {
                matrix(k, k - 1) = -ycoeff;
                matrix(k, k) = 1 + 2 * ycoeff;
                matrix(k, k + 1) = -ycoeff;
                ans[k] = u[i][k][j] + (th/2)*f(j*xh, k*yh, i*th + th/2);
            }
            matrix(n - 1, n - 2) = -ycoeff;
            matrix(n - 1, n - 1) = 1 + 2 * ycoeff;
            ans[n - 1] = u[i][n - 1][j] + ycoeff*u[i + 1][n - 1][j] + (th/2)*f(n*xh, j*yh, i
                *th + th/2);
            ans = RUNsolveSLAE(matrix, ans);
            for (uint64_t k = 0; k < ans.size(); ++k) {
                u[i + 1][k + 1][j] = ans[k];
            }
        }
    }
}

void FractionalSteps (std::vector<std::vector<std::vector<double>>> &u, const std::vector<
    std::vector<double>> &ux, const std::vector<double> &coeff, double xh, double yh,
    double th, const std::function<double(double, double, double)> &f) {
    for (uint64_t i = 0; i < u.size() - 1; ++i) {
        FractionalStepsIteration(u, ux, coeff, xh, yh, th, f, i);
    }
}

std::vector<std::vector<std::vector<double>>> SolveIBVP (const Task &task, double xh,
    double yh, double th, Method method) {
    double l1 = task.X[1], l2 = task.Y[1], l3 = task.T[1];
    std::vector<std::vector<std::vector<double>>> u(uint64_t(l3 / th), std::vector<std::
        vector<double>>(uint64_t(l2 / yh), std::vector<double>(uint64_t(l1 / xh), 0)));
    std::vector<ApproxLevel> border(4, ApproxLevel::NONE);
```

```
    for (uint64_t i = 0; i < 4; ++i) {
        if (task.ux[i][0] != 0) {
            border[i] = ApproxLevel::POINT2_ORDER2;
        }
    }
    auto f = [&] (double x, double y, double t) -> double {
        return task.trees[0]({0, 0, 0, 0, 0, x, y, t});
    };
    auto fx0 = [&] (double y, double t) -> double {
        return task.trees[1]({y, t});
    };
    auto fxl = [&] (double y, double t) -> double {
        return task.trees[2]({y, t});
    };
    auto fy0 = [&] (double x, double t) -> double {
        return task.trees[3]({x, t});
    };
    auto fyl = [&] (double x, double t) -> double {
        return task.trees[4]({x, t});
    };
    auto ft0 = [&] (double x, double y) -> double {
        return task.trees[5]({x, y});
    };
    for (uint64_t i = 0; i < u[0].size(); ++i) {
        for (uint64_t j = 0; j < u[0][0].size(); ++j) {
            u.front()[i][j] = ft0(j * xh, i * yh);
        }
    }
    for (uint64_t i = 0; i < u.size(); ++i) {
        for (uint64_t j = 0; j < u[0][0].size(); ++j) {
            u[i].front()[j] = fy0(i * xh, j * th);
            u[i].back()[j] = fyl(i * xh, j * th);
        }
    }
    for (uint64_t i = 0; i < u.size(); ++i) {
        for (uint64_t j = 0; j < u[0].size(); ++j) {
            u[i][j].front() = fx0(j * yh, i * th);
            u[i][j].back() = fxl(j * yh, i * th);
        }
    }
    switch (method) {
        case Method::VARIABLE_DIRECTIONS:
            VariableDirections(u, task.ux, task.coeff, xh, yh, th, f);
            break;
        case Method::FRACTIONAL_STEPS:
            FractionalSteps(u, task.ux, task.coeff, xh, yh, th, f);
            break;
        default:
```

```
            break;
    }
    return u;
}
```

Выводы

Погрешность решения зависит от параметров h_x и h_y и слабо зависит от h_t.