

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:

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

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: $ut = u_{xx} + u_{yy}$

1: $u(0) = \sinh(y) \cdot \exp(-3 \cdot t)$

2: $u(\pi/2) = 0 - \sinh(y) \cdot \exp(-3 \cdot t)$

3: $u_y(0) = \cos(2 \cdot x) \cdot \exp(-3 \cdot t)$

4: $u(\ln(2)) = 3/4 \cdot \cos(2 \cdot x) \cdot \exp(-3 \cdot t)$

5: $u(0) = \cos(2 \cdot x) \cdot \sinh(y)$

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:

```
1  #include "8-1.hpp"
2
3  double Point2Order1 (const std::vector<double> &coeff, double h, double u1, double f,
4      uint64_t i) {
5      double alpha = coeff[0], beta = coeff[1];
6      double ans = 0;
7      if (i == 0) {
8          ans += f - alpha * u1 / h;
9          ans /= beta - alpha / h;
10     } else {
11         ans += f + alpha * u1 / h;
12         ans /= beta + alpha / h;
13     }
14     return ans;
15 }
16
17 double Point2Order2 (const std::vector<double> &ux, const std::vector<double> &coeff,
18     double h, double t, double u0, double u1, double f, uint64_t i) {
19     double alpha = ux[0], beta = ux[1];
20     double a = coeff[0], b = coeff[1], c = coeff[2];
21     double ans = 0;
22     if (i == 0) {
23         ans += h / t * u0 - f * (2 * a - b * h) / alpha + 2 * u1 * a / h;
24         ans /= 2 * a / h + h / t - c * h - (beta / alpha) * (2 * a - b * h);
25     } else {
26         ans += h / t * u0 + f * (2 * a + b * h) / alpha + 2 * u1 * a / h;
27         ans /= 2 * a / h + h / t - c * h + (beta / alpha) * (2 * a + b * h);
28     }
29     return ans;
30 }
31
32 double Point3Order2 (const std::vector<double> &coeff, double h, double u1, double u2,
33     double f, uint64_t i) {
34     double alpha = coeff[0], beta = coeff[1];
35     double ans = 0;
36     if (i == 0) {
37         ans += f - alpha * (4 * u1 - u2) / (2 * h);
38         ans /= beta - 3 * alpha / (2 * h);
39     } else {
40         ans += f + alpha * (4 * u1 - u2) / (2 * h);
41         ans /= beta + 3 * alpha / (2 * h);
42     }
43     return ans;
44 }
```

```

43 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) {
44     double a = coeff[0], b = coeff[2];
45     double xcoeff = a * th / (2 * xh * xh);
46     double ycoeff = b * th / (2 * yh * yh);
47     auto tmp = u[i];
48     {
49         uint64_t n = u[0][0].size() - 2;
50         Matrix<double> matrix(n, n);
51         std::vector<double> ans(n);
52         for (uint64_t j = 1; j < u[0].size() - 1; ++j) {
53             matrix(0, 0) = 1 + 2 * xcoeff;
54             matrix(0, 1) = -xcoeff;
55             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);
56             for (uint64_t k = 1; k < n - 1; ++k) {
57                 matrix(k, k - 1) = -xcoeff;
58                 matrix(k, k) = 1 + 2 * xcoeff;
59                 matrix(k, k + 1) = -xcoeff;
60                 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);
61             }
62             matrix(n - 1, n - 2) = -xcoeff;
63             matrix(n - 1, n - 1) = 1 + 2 * xcoeff;
64             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);
65             ans = RUNsolveSLAE(matrix, ans);
66             for (uint64_t k = 0; k < ans.size(); ++k) {
67                 tmp[j][k + 1] = ans[k];
68             }
69         }
70     }
71     {
72         uint64_t n = u[0].size() - 2;
73         Matrix<double> matrix(n, n);
74         std::vector<double> ans(n);
75         for (uint64_t j = 1; j < u[0][0].size() - 1; ++j) {
76             matrix(0, 0) = 1 + 2 * ycoeff;
77             matrix(0, 1) = -ycoeff;
78             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);
79             for (uint64_t k = 1; k < n - 1; ++k) {
80                 matrix(k, k - 1) = -ycoeff;
81                 matrix(k, k) = 1 + 2 * ycoeff;
82                 matrix(k, k + 1) = -ycoeff;

```

```

83         ans[k] = u[i][k][j] + xcoeff*(u[i][k][j + 1] - 2*u[i][k][j] + u[i][k][j -
84             1]) + (th/2)*f(j*xh, k*yh, i*th + th/2);
85     }
86     matrix(n - 1, n - 2) = -ycoeff;
87     matrix(n - 1, n - 1) = 1 + 2 * ycoeff;
88     ans[n - 1] = u[i][n - 1][j] + ycoeff*u[i + 1][n - 1][j] + xcoeff*(u[i][n - 1][j
89         + 1] - 2*u[i][n - 1][j] + u[i][n - 1][j - 1]) + (th/2)*f(j*xh, n*yh, i*th +
90         th/2);
91     ans = RUNsolveSLAE(matrix, ans);
92     for (uint64_t k = 0; k < ans.size(); ++k) {
93         u[0][k + 1][j] = ans[k];
94     }
95 }
96 void VariableDirections (std::vector<std::vector<std::vector<double>>> &u, const std::
97     vector<std::vector<double>> &ux, const std::vector<double> &coeff, double xh, double
98     yh, double th, const std::function<double(double, double, double)> &f) {
99     for (uint64_t i = 0; i < u.size() - 1; ++i) {
100         VariableDirectionsIteration(u, ux, coeff, xh, yh, th, f, i);
101     }
102 }
103 void FractionalStepsIteration (std::vector<std::vector<std::vector<double>>> &u, const std
104     ::vector<std::vector<double>> &ux, const std::vector<double> &coeff, double xh, double
105     yh, double th, const std::function<double(double, double, double)> &f, uint64_t i) {
106     double a = coeff[0], b = coeff[2];
107     double xcoeff = a * th / (xh * xh);
108     double ycoeff = b * th / (yh * yh);
109     auto tmp = u[i];
110     {
111         uint64_t n = u[0][0].size() - 2;
112         Matrix<double> matrix(n, n);
113         std::vector<double> ans(n);
114         for (uint64_t j = 1; j < u[0].size() - 1; ++j) {
115             matrix(0, 0) = 1 + 2 * xcoeff;
116             matrix(0, 1) = -xcoeff;
117             ans[0] = u[i][j][0] + xcoeff*u[i + 1][j][0] + (th/2)*f(0, j * yh, i*th + th / 2)
118                 ;
119             for (uint64_t k = 1; k < n - 1; ++k) {
120                 matrix(k, k - 1) = -xcoeff;
121                 matrix(k, k) = 1 + 2 * xcoeff;
122                 matrix(k, k + 1) = -xcoeff;
123                 ans[k] = u[i][j][k] + (th/2)*f(k*xh, j*yh, i * th + th/2);
124             }
125             matrix(n - 1, n - 2) = -xcoeff;
126             matrix(n - 1, n - 1) = 1 + 2 * xcoeff;

```

```

123         ans[n - 1] = u[i][j][n - 1] + xcoeff*u[i + 1][j][n - 1] + (th/2)*f(n*xh, j*yh, i
124             *th + th/2);
125         ans = RUNsolveSLAE(matrix, ans);
126         for (uint64_t k = 0; k < ans.size(); ++k) {
127             tmp[j][k + 1] = ans[k];
128         }
129     }
130     {
131         uint64_t n = u[0].size() - 2;
132         Matrix<double> matrix(n, n);
133         std::vector<double> ans(n);
134         for (uint64_t j = 1; j < u[0][0].size() - 1; ++j) {
135             matrix(0, 0) = 1 + 2 * ycoeff;
136             matrix(0, 1) = -ycoeff;
137             ans[0] = u[i][0][j] + ycoeff*u[i + 1][0][j] + (th/2)*f(j*xh, 0, i*th + th / 2);
138             for (uint64_t k = 1; k < n - 1; ++k) {
139                 matrix(k, k - 1) = -ycoeff;
140                 matrix(k, k) = 1 + 2 * ycoeff;
141                 matrix(k, k + 1) = -ycoeff;
142                 ans[k] = u[i][k][j] + (th/2)*f(j*xh, k*yh, i*th + th/2);
143             }
144             matrix(n - 1, n - 2) = -ycoeff;
145             matrix(n - 1, n - 1) = 1 + 2 * ycoeff;
146             ans[n - 1] = u[i][n - 1][j] + ycoeff*u[i + 1][n - 1][j] + (th/2)*f(n*xh, j*yh, i
147                 *th + th/2);
148             ans = RUNsolveSLAE(matrix, ans);
149             for (uint64_t k = 0; k < ans.size(); ++k) {
150                 u[i + 1][k + 1][j] = ans[k];
151             }
152         }
153     }
154 }
155 void FractionalSteps (std::vector<std::vector<std::vector<double>>> &u, const std::vector<
156     std::vector<double>> &ux, const std::vector<double> &coeff, double xh, double yh,
157     double th, const std::function<double(double, double, double)> &f) {
158     for (uint64_t i = 0; i < u.size() - 1; ++i) {
159         FractionalStepsIteration(u, ux, coeff, xh, yh, th, f, i);
160     }
161 }
162 std::vector<std::vector<std::vector<double>>> SolveIBVP (const Task &task, double xh,
163     double yh, double th, Method method) {
164     double l1 = task.X[1], l2 = task.Y[1], l3 = task.T[1];
165     std::vector<std::vector<std::vector<double>>> u(uint64_t(l3 / th), std::vector<std:::
166         vector<double>>(uint64_t(l2 / yh), std::vector<double>(uint64_t(l1 / xh), 0)));
167     std::vector<ApproxLevel> border(4, ApproxLevel::NONE);

```

```

165     for (uint64_t i = 0; i < 4; ++i) {
166         if (task.ux[i][0] != 0) {
167             border[i] = ApproxLevel::POINT2_ORDER2;
168         }
169     }
170     auto f = [&] (double x, double y, double t) -> double {
171         return task.trees[0]({0, 0, 0, 0, 0, x, y, t});
172     };
173     auto fx0 = [&] (double y, double t) -> double {
174         return task.trees[1]({y, t});
175     };
176     auto fx1 = [&] (double y, double t) -> double {
177         return task.trees[2]({y, t});
178     };
179     auto fy0 = [&] (double x, double t) -> double {
180         return task.trees[3]({x, t});
181     };
182     auto fyl = [&] (double x, double t) -> double {
183         return task.trees[4]({x, t});
184     };
185     auto ft0 = [&] (double x, double y) -> double {
186         return task.trees[5]({x, y});
187     };
188     for (uint64_t i = 0; i < u[0].size(); ++i) {
189         for (uint64_t j = 0; j < u[0][0].size(); ++j) {
190             u.front()[i][j] = ft0(j * xh, i * yh);
191         }
192     }
193     for (uint64_t i = 0; i < u.size(); ++i) {
194         for (uint64_t j = 0; j < u[0][0].size(); ++j) {
195             u[i].front()[j] = fy0(i * xh, j * th);
196             u[i].back()[j] = fyl(i * xh, j * th);
197         }
198     }
199     for (uint64_t i = 0; i < u.size(); ++i) {
200         for (uint64_t j = 0; j < u[0].size(); ++j) {
201             u[i][j].front() = fx0(j * yh, i * th);
202             u[i][j].back() = fx1(j * yh, i * th);
203         }
204     }
205     switch (method) {
206         case Method::VARIABLE_DIRECTIONS:
207             VariableDirections(u, task.ux, task.coeff, xh, yh, th, f);
208             break;
209         case Method::FRACTIONAL_STEPS:
210             FractionalSteps(u, task.ux, task.coeff, xh, yh, th, f);
211             break;
212         default:

```



```
213 |         break;
214 |     }
215 |     return u;
216 | }
```

Выводы

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