Московский авиационный институт (национальный исследовательский университет)

Институт №8 «Информационные технологии и прикладная математика»

Кафедра 806 «Вычислительная математика и программирование»

Лабораторные работы по курсу «Численные методы»

Студент: А. Л. Ядров
Преподаватель: Д. Е. Пивоваров
Группа: М8О-408Б-20

Дата:
Оценка:
Подпись:

Москва, 2024

1 Решение двумерной начально-краевой задачи для дифференциальных уравнений в частных производных параболического типа

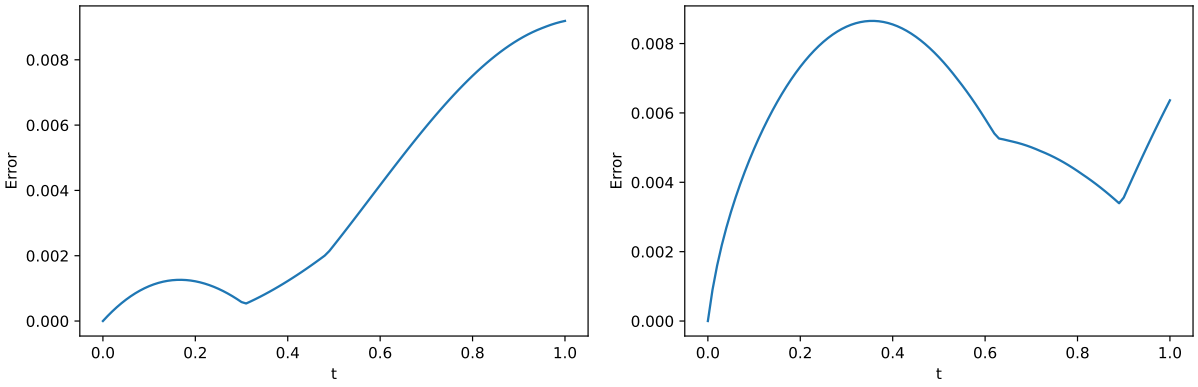
1 Постановка задачи

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 $U(x,t)$. Исследовать зависимость погрешности от сеточных параметров τ, h_x, h_y.

Вариант: 8

$$\frac{\partial u}{\partial t} = a\frac{\partial^2 u}{\partial x^2} + b\frac{\partial^2 u}{\partial y^2} + \sin x \sin y(\mu\cos(\mu t) + (a+b)\sin(\mu t))$$
$$u(0,y,t) = 0$$
$$u(2\pi,y,t) = \sin y \sin(\mu t)$$
$$u(x,0,t) = 0$$
$$u(x,2\pi,t) = \sin x \sin(\mu t)$$
$$u(x,y,0) = 0$$
$$U(x,y) = \sin x \sin y \sin(\mu t)$$
$$a = 1,\ b = 1,\ \mu = 1$$

2 Результаты работы



(a) (b)

Рис. 1: Максимум погрешности решения в зависимости от времени для (a) метода переменных направлений и (b) метода дробных шагов

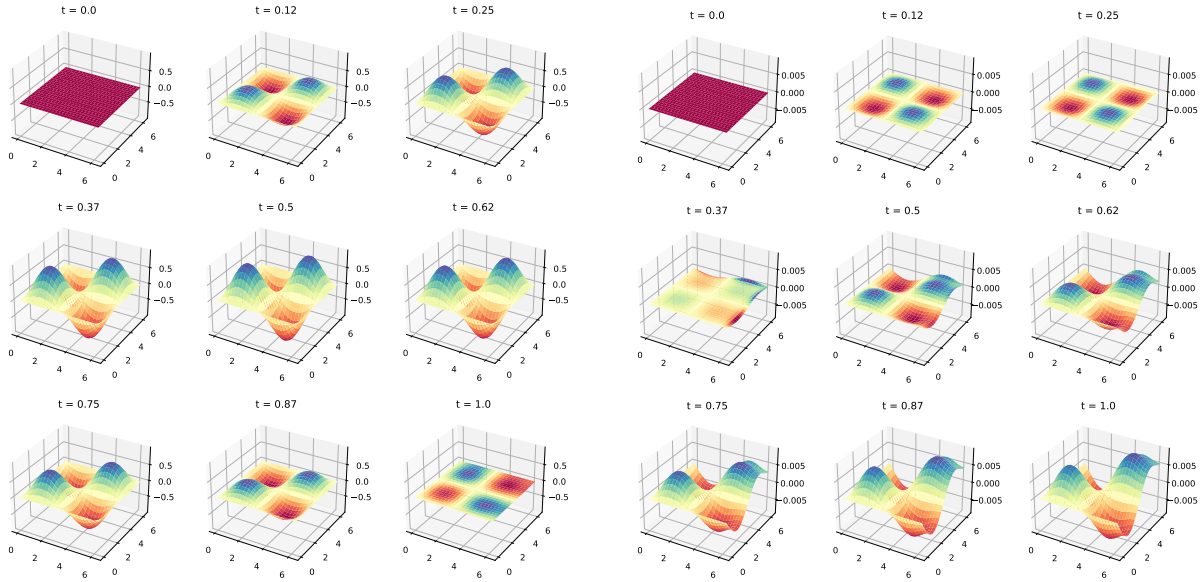


Рис. 2: Эволюция решения и погрешности для метода переменных направлений

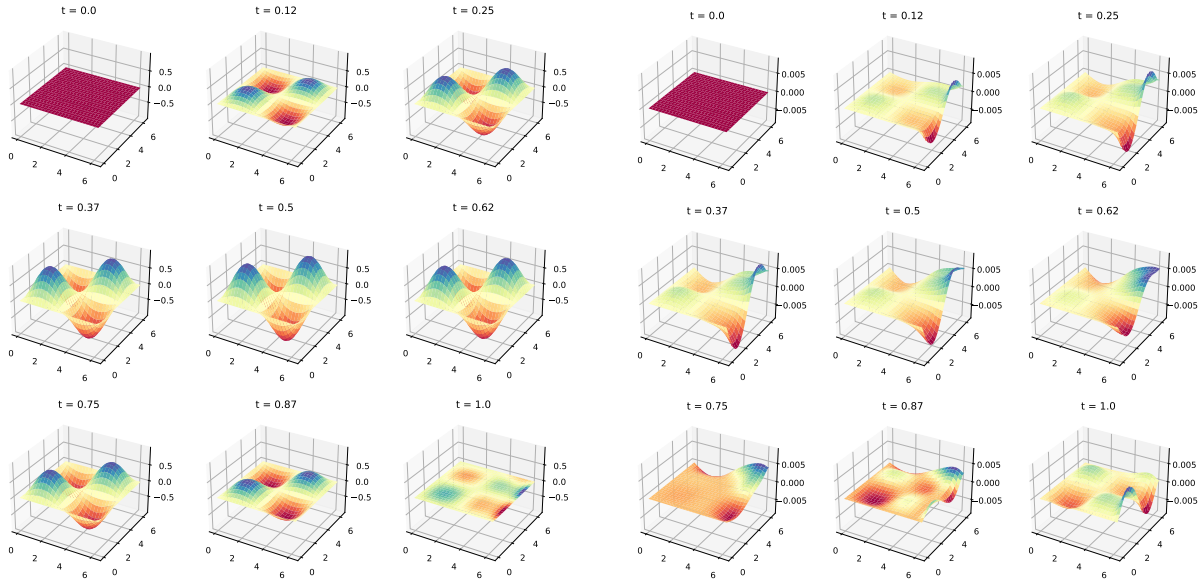


Рис. 3: Эволюция решения и погрешности для метода дробных шагов

3 Исходный код

```
#pragma once

#include <functional>
#include <vector>
#include <tuple>

#include "../linear/tridiagonal_matrix.hpp"
#include "../linear/vector.hpp"
#include "common.hpp"

template <int D, class T>
struct Tensor : public std::vector<Tensor<D - 1, T>> {
  static_assert(D >= 1, "Tensor dimension must be greater than zero");

  template <class... Args>
  Tensor(int size = 0, Args... args) : std::vector<Tensor<D - 1, T>>(size, Tensor<D -
      1, T>(args...)) {}
};

template <class T>
struct Tensor<1, T> : public std::vector<T> {
  Tensor(int size = 0, const T& value = T()): std::vector<T>(size, value) {}
};

```

```
24 namespace Parabolic2dPDE {
25   template <class T>
26   struct PDE {
27     using f_x_y = std::function<T(T, T)>;
28     using f_x_t = f_x_y;
29     using f_y_t = f_x_y;
30     using f_x_y_t = std::function<T(T, T, T)>;
31 
32     T a, bx, by, c;
33     f_x_y_t f;
34     T x0, x1, y0, y1;
35     f_x_y psi;
36     T alpha_x0, beta_x0;
37     f_y_t gamma_x0;
38     T alpha_x1, beta_x1;
39     f_y_t gamma_x1;
40     T alpha_y0, beta_y0;
41     f_x_t gamma_y0;
42     T alpha_y1, beta_y1;
43     f_x_t gamma_y1;
44     f_x_y_t solution;
45 
46     PDE() = default;
47 
48     PDE(T a, T bx, T by, T c, f_x_y_t f, T x0, T x1, T y0, T y1, f_x_y psi) :
49         a(a), bx(bx), by(by), c(c), f(f), x0(x0), x1(x1), y0(y0), y1(y1), psi(psi) {}
50 
51     PDE(T a, T bx, T by, T c, f_x_y_t f, T x0, T x1, T y0, T y1, f_x_y psi,
52         T alpha_x0, T beta_x0, f_y_t gamma_x0, T alpha_x1, T beta_x1, f_y_t gamma_x1,
53         T alpha_y0, T beta_y0, f_x_t gamma_y0, T alpha_y1, T beta_y1, f_x_t gamma_y1,
               f_x_y_t solution) :
54         a(a), bx(bx), by(by), c(c), f(f), x0(x0), x1(x1), y0(y0), y1(y1), psi(psi),
55         alpha_x0(alpha_x0), beta_x0(beta_x0), gamma_x0(gamma_x0), alpha_x1(alpha_x1),
               beta_x1(beta_x1), gamma_x1(gamma_x1),
56         alpha_y0(alpha_y0), beta_y0(beta_y0), gamma_y0(gamma_y0), alpha_y1(alpha_y1),
               beta_y1(beta_y1), gamma_y1(gamma_y1), solution(solution) {}
57 
58     PDE(T a, T bx, T by, T c, f_x_y_t f, T x0, T x1, T y0, T y1, f_x_y psi,
59         T alpha_x0, T beta_x0, f_y_t gamma_x0, T alpha_x1, T beta_x1, f_y_t gamma_x1,
60         T alpha_y0, T beta_y0, f_x_t gamma_y0, T alpha_y1, T beta_y1, f_x_t gamma_y1) :
61         a(a), bx(bx), by(by), c(c), f(f), x0(x0), x1(x1), y0(y0), y1(y1), psi(psi),
62         alpha_x0(alpha_x0), beta_x0(beta_x0), gamma_x0(gamma_x0), alpha_x1(alpha_x1),
               beta_x1(beta_x1), gamma_x1(gamma_x1),
63         alpha_y0(alpha_y0), beta_y0(beta_y0), gamma_y0(gamma_y0), alpha_y1(alpha_y1),
               beta_y1(beta_y1), gamma_y1(gamma_y1) {}
64 
65     void SetEquation(T a_, T bx_, T by_, T c_, f_x_y_t f_, T x0_, T x1_, T y0_, T y1_,
           f_x_y psi_) {
66       a = a_;
```

```
    bx = bx_;
    by = by_;
    c = c_;
    f = f_;
    x0 = x0_; x1 = x1_; y0 = y0_; y1 = y1_;
    psi = psi_;
  }

  void SetBoundaries(T alpha_x0_, T beta_x0_, f_y_t gamma_x0_,
                     T alpha_x1_, T beta_x1_, f_y_t gamma_x1_,
                     T alpha_y0_, T beta_y0_, f_x_t gamma_y0_,
                     T alpha_y1_, T beta_y1_, f_x_t gamma_y1_) {
    alpha_x0 = alpha_x0_; beta_x0 = beta_x0_; gamma_x0 = gamma_x0_;
    alpha_x1 = alpha_x1_; beta_x1 = beta_x1_; gamma_x1 = gamma_x1_;
    alpha_y0 = alpha_y0_; beta_y0 = beta_y0_; gamma_y0 = gamma_y0_;
    alpha_y1 = alpha_y1_; beta_y1 = beta_y1_; gamma_y1 = gamma_y1_;
  }

  void SetSolution(f_x_y_t solution_) {
    solution = solution_;
  }
};

template <class T>
std::tuple<Tensor<3, T>, std::vector<T>, std::vector<T>, std::vector<T>>
AlternatingDirectionMethod(const PDE<T>& pde, int nx, int ny, int nt, T t_end) {
  std::vector<T> x(nx + 1), y(ny + 1), t(nt + 1);
  Tensor<3, T> u(nt + 1, nx + 1, ny + 1);
  Tensor<2, T> p(nx + 1, ny + 1);

  T hx = (pde.x1 - pde.x0) / nx;
  T hy = (pde.y1 - pde.y0) / ny;
  T tau = t_end / nt;

  for (int i = 0; i <= nx; ++i) {
    x[i] = pde.x0 + i * hx;
  }
  for (int i = 0; i <= ny; ++i) {
    y[i] = pde.y0 + i * hy;
  }
  for (int i = 0; i <= nt; ++i) {
    t[i] = i * tau;
  }

  for (size_t i = 0; i < x.size(); ++i) {
    for (size_t j = 0; j < y.size(); ++j) {
      u[0][i][j] = pde.psi(x[i], y[j]);
    }
  }
```

```
116
117     Vector<T> vx(nx+1), vy(ny+1);
118     TDMatrix<T> mx(nx+1), my(ny+1);
119
120     T alpha = (- pde.a / hx + pde.bx / 2) / hx,
121       beta = 2 * pde.a / hx / hx + 2 / tau - pde.c,
122       gamma = (- pde.a / hx - pde.bx / 2) / hx;
123     for (int i = 1; i < nx; ++i) {
124       mx.a[i] = alpha;
125       mx.b[i] = beta;
126       mx.c[i] = gamma;
127     }
128
129     alpha = (- pde.a / hy + pde.by / 2) / hy,
130     beta = 2 * pde.a / hy / hy + 2 / tau - pde.c,
131     gamma = (- pde.a / hy - pde.by / 2) / hy;
132     for (int j = 1; j < ny; ++j) {
133       my.a[j] = alpha;
134       my.b[j] = beta;
135       my.c[j] = gamma;
136     }
137
138     for (int k = 0; k < nt; ++k) {
139       for (int j = 1; j < ny; ++j) {
140         for (int i = 1; i < nx; ++i) {
141           vx[i] = u[k][i][j-1] * (pde.a / hy - pde.by / 2) / hy +
142                   2 * u[k][i][j] * (1 / tau - pde.a / hy / hy) +
143                   u[k][i][j+1] * (pde.a / hy + pde.by / 2) / hy +
144                   pde.f(x[i], y[j], t[k] + tau / 2);
145         }
146         vx[0] = pde.gamma_x0(y[j], t[k] + tau / 2);
147         vx[nx] = pde.gamma_x1(y[j], t[k] + tau / 2);
148
149         mx.b[0] = -pde.alpha_x0 / hx + pde.beta_x0;
150         mx.c[0] = pde.alpha_x0 / hx;
151
152         mx.a[nx] = -pde.alpha_x1 / hx;
153         mx.b[nx] = pde.alpha_x1 / hx + pde.beta_x1;
154
155         vx = mx.Solve(vx);
156
157         for (int i = 0; i <= nx; ++i) {
158           p[i][j] = vx[i];
159         }
160       }
161
162       for (int i = 0; i <= nx; ++i) {
163         p[i][0] = (pde.gamma_y0(x[i], t[k] + tau/2) - pde.alpha_y0 / hy * p[i][1]) / (-
                pde.alpha_y0 / hy + pde.beta_y0);
```

```
164       p[i][ny] = (pde.gamma_y1(x[i], t[k] + tau/2) + pde.alpha_y1 / hy * p[i][ny-1])
              / (pde.alpha_y1 / hy + pde.beta_y1);
165     }
166
167     for (int i = 1; i < nx; ++i) {
168       for (int j = 1; j < ny; ++j) {
169         vy[j] = p[i-1][j] * (pde.a / hx - pde.bx / 2) / hx +
170                 2 * p[i][j] * (1 / tau - pde.a / hx / hx) +
171                 p[i+1][j] * (pde.a / hx + pde.bx / 2) / hx +
172                 pde.f(x[i], y[j], t[k+1]);
173       }
174       vy[0] = pde.gamma_y0(x[i], t[k+1]);
175       vy[ny] = pde.gamma_y1(x[i], t[k+1]);
176
177       my.b[0] = -pde.alpha_y0 / hy + pde.beta_y0;
178       my.c[0] = pde.alpha_y0 / hy;
179
180       my.a[ny] = -pde.alpha_y1 / hy;
181       my.b[ny] = pde.alpha_y1 / hy + pde.beta_y1;
182
183       vy = my.Solve(vy);
184
185       for (int j = 0; j <= ny; ++j) {
186         u[k+1][i][j] = vy[j];
187       }
188     }
189
190     for (int j = 0; j <= ny; ++j) {
191       u[k+1][0][j] = (pde.gamma_x0(y[j], t[k+1]) - pde.alpha_x0 / hx * u[k+1][1][j])
              / (-pde.alpha_x0 / hx + pde.beta_x0);
192       u[k+1][nx][j] = (pde.gamma_x1(y[j], t[k+1]) + pde.alpha_x1 / hx * u[k+1][nx-1][
              j]) / (pde.alpha_x1 / hx + pde.beta_x1);
193     }
194   }
195
196   return {u, x, y, t};
197 }
198
199 template <class T>
200 std::tuple<Tensor<3, T>, std::vector<T>, std::vector<T>, std::vector<T>>
201 FractionalStepMethod(const PDE<T>& pde, int nx, int ny, int nt, T t_end) {
202   std::vector<T> x(nx + 1), y(ny + 1), t(nt + 1);
203   Tensor<3, T> u(nt + 1, nx + 1, ny + 1);
204   Tensor<2, T> p(nx + 1, ny + 1);
205
206   T hx = (pde.x1 - pde.x0) / nx;
207   T hy = (pde.y1 - pde.y0) / ny;
208   T tau = t_end / nt;
209
```

```
210     for (int i = 0; i <= nx; ++i) {
211       x[i] = pde.x0 + i * hx;
212     }
213     for (int i = 0; i <= ny; ++i) {
214       y[i] = pde.y0 + i * hy;
215     }
216     for (int i = 0; i <= nt; ++i) {
217       t[i] = i * tau;
218     }
219
220     for (size_t i = 0; i < x.size(); ++i) {
221       for (size_t j = 0; j < y.size(); ++j) {
222         u[0][i][j] = pde.psi(x[i], y[j]);
223       }
224     }
225
226     Vector<T> vx(nx+1), vy(ny+1);
227     TDMatrix<T> mx(nx+1), my(ny+1);
228
229     T alpha = (- pde.a / hx + pde.bx / 2) / hx,
230       beta = 2 * pde.a / hx / hx + 1 / tau - pde.c,
231       gamma = (- pde.a / hx - pde.bx / 2) / hx;
232     for (int i = 1; i < nx; ++i) {
233       mx.a[i] = alpha;
234       mx.b[i] = beta;
235       mx.c[i] = gamma;
236     }
237
238     alpha = (- pde.a / hy + pde.by / 2) / hy,
239     beta = 2 * pde.a / hy / hy + 1 / tau - pde.c,
240     gamma = (- pde.a / hy - pde.by / 2) / hy;
241     for (int j = 1; j < ny; ++j) {
242       my.a[j] = alpha;
243       my.b[j] = beta;
244       my.c[j] = gamma;
245     }
246
247     for (int k = 0; k < nt; ++k) {
248       for (int j = 1; j < ny; ++j) {
249         for (int i = 1; i < nx; ++i) {
250           vx[i] = u[k][i][j] / tau + pde.f(x[i], y[j], t[k]) / 2;
251         }
252         vx[0] = pde.gamma_x0(y[j], t[k+1]);
253         vx[nx] = pde.gamma_x1(y[j], t[k+1]);
254
255         mx.b[0] = -pde.alpha_x0 / hx + pde.beta_x0;
256         mx.c[0] = pde.alpha_x0 / hx;
257
258         mx.a[nx] = -pde.alpha_x1 / hx;
```

```
259         mx.b[nx] = pde.alpha_x1 / hx + pde.beta_x1;
260
261         vx = mx.Solve(vx);
262
263         for (int i = 0; i <= nx; ++i) {
264           p[i][j] = vx[i];
265         }
266       }
267
268       for (int i = 0; i <= nx; ++i) {
269         p[i][0] = (pde.gamma_y0(x[i], t[k+1]) - pde.alpha_y0 / hy * p[i][1]) / (-pde.
              alpha_y0 / hy + pde.beta_y0);
270         p[i][ny] = (pde.gamma_y1(x[i], t[k+1]) + pde.alpha_y1 / hy * p[i][ny-1]) / (pde
              .alpha_y1 / hy + pde.beta_y1);
271       }
272
273       for (int i = 1; i < nx; ++i) {
274         for (int j = 1; j < ny; ++j) {
275           vy[j] = p[i][j] / tau + pde.f(x[i], y[j], t[k+1]) / 2;
276         }
277         vy[0] = pde.gamma_y0(x[i], t[k+1]);
278         vy[ny] = pde.gamma_y1(x[i], t[k+1]);
279
280         my.b[0] = -pde.alpha_y0 / hy + pde.beta_y0;
281         my.c[0] = pde.alpha_y0 / hy;
282
283         my.a[ny] = -pde.alpha_y1 / hy;
284         my.b[ny] = pde.alpha_y1 / hy + pde.beta_y1;
285
286         vy = my.Solve(vy);
287
288         for (int j = 0; j <= ny; ++j) {
289           u[k+1][i][j] = vy[j];
290         }
291       }
292
293       for (int j = 0; j <= ny; ++j) {
294         u[k+1][0][j] = (pde.gamma_x0(y[j], t[k+1]) - pde.alpha_x0 / hx * u[k+1][1][j])
              / (-pde.alpha_x0 / hx + pde.beta_x0);
295         u[k+1][nx][j] = (pde.gamma_x1(y[j], t[k+1]) + pde.alpha_x1 / hx * u[k+1][nx-1][
              j]) / (pde.alpha_x1 / hx + pde.beta_x1);
296       }
297     }
298
299     return {u, x, y, t};
300   }
301 }
```