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 краевой задачи для дифференциальных уравнений в частных производных эллиптического типа

1 Постановка задачи

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 результатов с приведенным в задании аналитическим решением $U(x, y)$ Исследовать зависимость погрешности от сеточных параметров h_x, h_y.

Вариант: 8

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = -2\frac{\partial u}{\partial x} - 3u$$
$$u(0, y) = \cos y$$
$$u(\frac{\pi}{2}, y) = 0$$
$$u(x, 0) = \exp(-x)\cos x$$
$$u(x, \frac{\pi}{2}) = 0$$
$$U(x, y) = \exp(-x)\cos x \cos y$$

2 Результаты работы

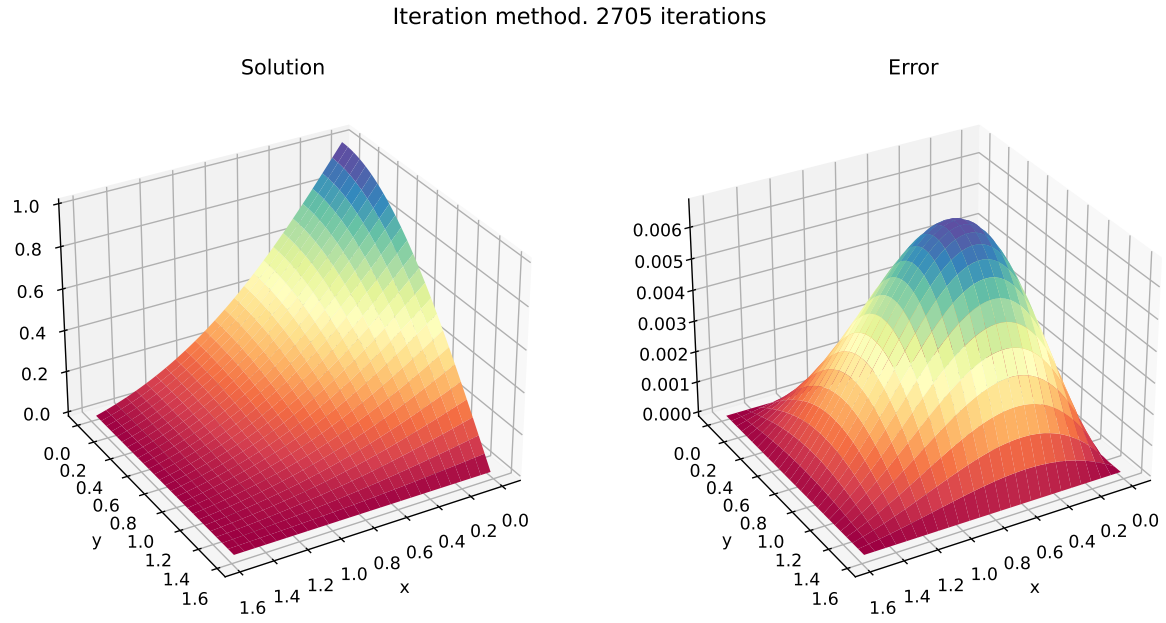

Рис. 1: Метод простых итераций

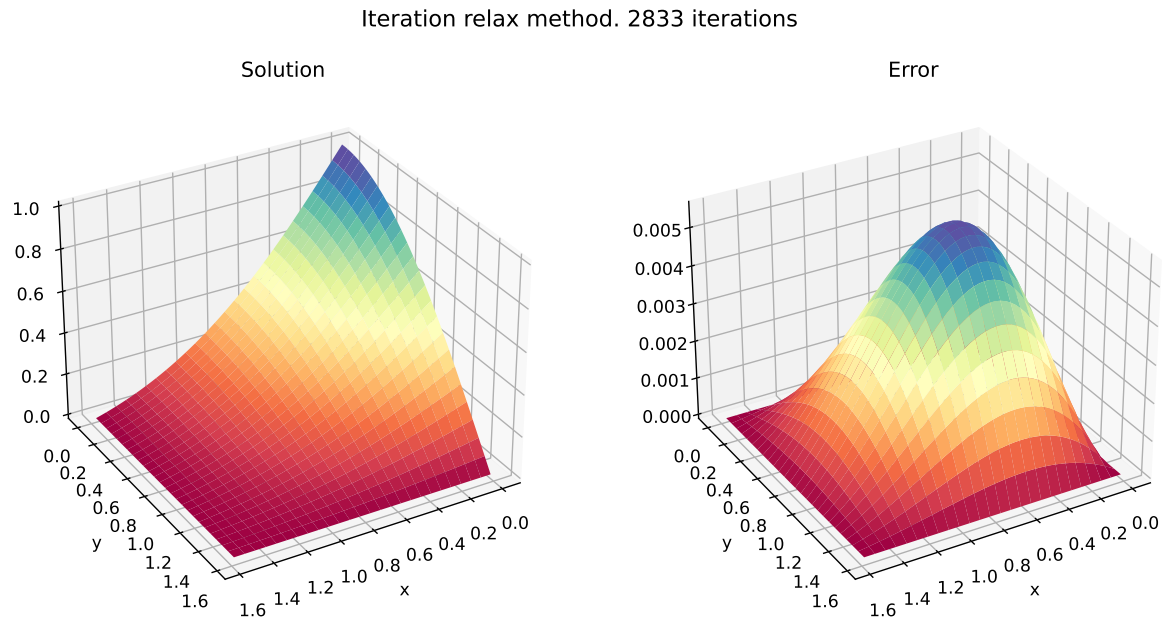

Рис. 2: Метод простых итераций с верхней релаксацией

Рис. 3: Метод Зейделя

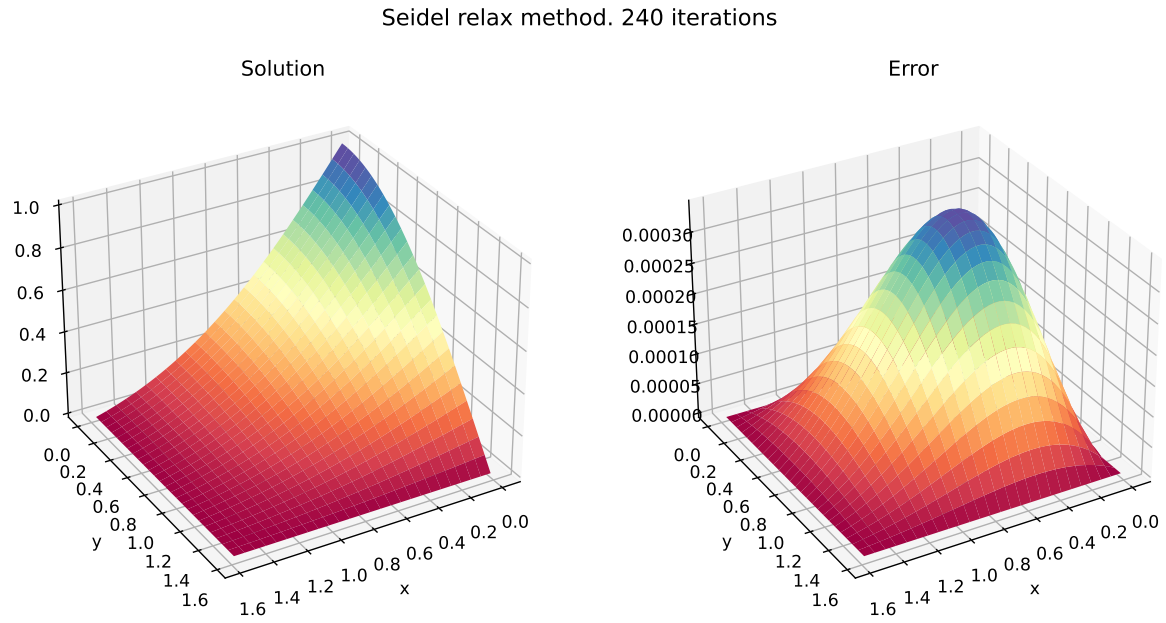

Рис. 4: Метод Зейделя с верхней релаксацией

3 Исходный код

```
#pragma once

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

#include "common.hpp"

namespace EllipticPDE {
  template <class T>
  using grid_t = std::vector<std::vector<T>>;

  template <class T>
  struct PDE {
    using f_x = std::function<T(T)>;
    using f_y = f_x;
    using f_x_y = std::function<T(T, T)>;

    T a, bx, by, c;
    f_x_y f;
    T x0, x1, y0, y1;
    T alpha_x0, beta_x0;
    f_y gamma_x0;
    T alpha_x1, beta_x1;
    f_y gamma_x1;
    T alpha_y0, beta_y0;
    f_x gamma_y0;
    T alpha_y1, beta_y1;
    f_x gamma_y1;
    f_x_y solution;

    PDE() = default;

    PDE(T a, T bx, T by, T c, f_x_y f, T x0, T x1, T y0, T y1,
        T alpha_x0, T beta_x0, f_y gamma_x0, T alpha_x1, T beta_x1, f_y gamma_x1,
        T alpha_y0, T beta_y0, f_x gamma_y0, T alpha_y1, T beta_y1, f_x gamma_y1, f_x_y
             solution) :
        a(a), bx(bx), by(by), c(c), f(f), x0(x0), x1(x1), y0(y0), y1(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), solution(solution) {}

    PDE(T a, T bx, T by, T c, f_x_y f, T x0, T x1, T y0, T y1,
        T alpha_x0, T beta_x0, f_y gamma_x0, T alpha_x1, T beta_x1, f_y gamma_x1,
        T alpha_y0, T beta_y0, f_x gamma_y0, T alpha_y1, T beta_y1, f_x gamma_y1) :
```

```
45        a(a), bx(bx), by(by), c(c), f(f), x0(x0), x1(x1), y0(y0), y1(y1),
46        alpha_x0(alpha_x0), beta_x0(beta_x0), gamma_x0(gamma_x0), alpha_x1(alpha_x1),
              beta_x1(beta_x1), gamma_x1(gamma_x1),
47        alpha_y0(alpha_y0), beta_y0(beta_y0), gamma_y0(gamma_y0), alpha_y1(alpha_y1),
              beta_y1(beta_y1), gamma_y1(gamma_y1) {}
48
49    void SetEquation(T a_, T bx_, T by_, T c_, f_x_y f_) {
50      a = a_;
51      bx = bx_;
52      by = by_;
53      c = c_;
54      f = f_;
55    }
56
57    void SetBoundaries(T x0_, T x1_, T y0_, T y1_, T alpha_x0_, T beta_x0_, f_y
          gamma_x0_,
58                       T alpha_x1_, T beta_x1_, f_y gamma_x1_, T alpha_y0_, T beta_y0_,
                             f_x gamma_y0_,
59                       T alpha_y1_, T beta_y1_, f_x gamma_y1_) {
60      x0 = x0_; x1 = x1_; y0 = y0_; y1 = y1_;
61      alpha_x0 = alpha_x0_; beta_x0 = beta_x0_; gamma_x0 = gamma_x0_;
62      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_;
64      alpha_y1 = alpha_y1_; beta_y1 = beta_y1_; gamma_y1 = gamma_y1_;
65    }
66
67    void SetSolution(f_x_y solution_) {
68      solution = solution_;
69    }
70  };
71
72  template <class T>
73  inline T Relax(T old_value, T new_value, T relax) {
74    return old_value + relax * (new_value - old_value);
75  }
76
77  template <class T>
78  std::tuple<std::vector<T>, std::vector<T>, grid_t<T>, int>
79  IterationSolver(const PDE<T>& pde, int nx, int ny, T eps, double relax = 1) {
80    std::vector<T> x(nx + 1), y(ny + 1);
81    grid_t<T> u(nx + 1, std::vector<T>(ny + 1, 0)), next_u(nx + 1, std::vector<T>(ny +
          1, 0));
82
83    T hx = (pde.x1 - pde.x0) / nx;
84    T hy = (pde.y1 - pde.y0) / ny;
85
86    for (int i = 0; i <= ny; ++i) {
87      y[i] = pde.y0 + i * hy;
88    }
```

```
89      for (int i = 0; i <= nx; ++i) {
90        x[i] = pde.x0 + i * hx;
91      }
92
93      for (int i = 0; i <= nx; ++i) {
94        u[i][0] = pde.gamma_y0(x[i]) / (pde.beta_y0 - pde.alpha_y0 / hy);
95        u[i][ny] = pde.gamma_y1(x[i]) / (pde.beta_y1 + pde.alpha_y1 / hy);
96      }
97      for (int i = 0; i <= ny; ++i) {
98        u[0][i] = pde.gamma_x0(y[i]) / (pde.beta_x0 - pde.alpha_x0 / hx);
99        u[nx][i] = pde.gamma_x1(y[i]) / (pde.beta_x1 + pde.alpha_x1 / hx);
100     }
101
102     T eps_k;
103     int iter_count = 0;
104     T coeff = (2 * pde.a * (1.0 / hx / hx + 1.0 / hy / hy) + pde.c);
105     do {
106       eps_k = 0;
107       for (int i = 1; i < nx; ++i) {
108         for (int j = 1; j < ny; ++j) {
109           next_u[i][j] = Relax(u[i][j],
110                                (pde.a * ((u[i+1][j] + u[i-1][j]) / hx / hx + (u[i][j+1] + u
                                     [i][j-1]) / hy / hy) -
111                                 (pde.bx * (u[i+1][j] - u[i-1][j]) / hx + pde.by * (u[i][j
                                     +1] - u[i][j-1]) / hy) / 2 - pde.f(x[i], y[j])) / coeff
                                     ,
112                                 relax);
113
114           eps_k = std::max(eps_k, std::abs(next_u[i][j] - u[i][j]));
115         }
116       }
117
118       for (int i = 1; i < nx; ++i) {
119         next_u[i][0] = Relax(u[i][0], (pde.gamma_y0(x[i]) - pde.alpha_y0 / hy * next_u[
              i][1]) / (pde.beta_y0 - pde.alpha_y0 / hy), relax);
120         next_u[i][ny] = Relax(u[i][ny], (pde.gamma_y1(x[i]) + pde.alpha_y1 / hy *
              next_u[i][ny-1]) / (pde.beta_y1 + pde.alpha_y1 / hy), relax);
121
122         eps_k = std::max(eps_k, std::abs(next_u[i][0] - u[i][0]));
123         eps_k = std::max(eps_k, std::abs(next_u[i][ny] - u[i][ny]));
124       }
125       for (int i = 1; i < ny; ++i) {
126         next_u[0][i] = Relax(u[0][i], (pde.gamma_x0(y[i]) - pde.alpha_x0 / hx * next_u
              [1][i]) / (pde.beta_x0 - pde.alpha_x0 / hx), relax);
127         next_u[nx][i] = Relax(u[nx][i], (pde.gamma_x1(y[i]) + pde.alpha_x1 / hx *
              next_u[nx-1][i]) / (pde.beta_x1 + pde.alpha_x1 / hx), relax);
128
129         eps_k = std::max(eps_k, std::abs(next_u[0][i] - u[0][i]));
130         eps_k = std::max(eps_k, std::abs(next_u[nx][i] - u[nx][i]));
```

```
131       }
132
133       std::swap(next_u, u);
134
135       ++iter_count;
136
137     } while (eps_k > eps);
138
139     u[0][0] = (pde.gamma_y0(x[0]) - pde.alpha_y0 / hx * u[0][1]) / (pde.beta_y0 - pde.
          alpha_y0 / hx);
140     u[nx][0] = (pde.gamma_y0(x[nx]) - pde.alpha_y0 / hx * u[nx][1]) / (pde.beta_y0 -
          pde.alpha_y0 / hx);
141     u[0][ny] = (pde.gamma_y1(x[0]) + pde.alpha_y1 / hx * u[0][ny-1]) / (pde.beta_y1 +
          pde.alpha_y1 / hx);
142     u[nx][ny] = (pde.gamma_y1(x[nx]) + pde.alpha_y1 / hx * u[nx][ny-1]) / (pde.beta_y1+
           pde.alpha_y1 / hx);
143
144     return {x, y, u, iter_count};
145   }
146
147   template <class T>
148   std::tuple<std::vector<T>, std::vector<T>, grid_t<T>, int>
149   SeidelSolver(const PDE<T>& pde, int nx, int ny, T eps, double relax = 1) {
150     std::vector<T> x(nx + 1), y(ny + 1);
151     grid_t<T> u(nx + 1, std::vector<T>(ny + 1, 0));
152
153     T hx = (pde.x1 - pde.x0) / nx;
154     T hy = (pde.y1 - pde.y0) / ny;
155
156     for (int i = 0; i <= ny; ++i) {
157       y[i] = pde.y0 + i * hy;
158     }
159     for (int i = 0; i <= nx; ++i) {
160       x[i] = pde.x0 + i * hx;
161     }
162
163     for (int i = 0; i <= nx; ++i) {
164       u[i][0] = pde.gamma_y0(x[i]) / (pde.beta_y0 - pde.alpha_y0 / hy);
165       u[i][ny] = pde.gamma_y1(x[i]) / (pde.beta_y1 + pde.alpha_y1 / hy);
166     }
167     for (int i = 0; i <= ny; ++i) {
168       u[0][i] = pde.gamma_x0(y[i]) / (pde.beta_x0 - pde.alpha_x0 / hx);
169       u[nx][i] = pde.gamma_x1(y[i]) / (pde.beta_x1 + pde.alpha_x1 / hx);
170     }
171
172     T eps_k, next_u;
173     int iter_count = 0;
174     T coeff = (2 * pde.a * (1.0 / hx / hx + 1.0 / hy / hy) + pde.c);
175     do {
```

```
176       eps_k = 0;
177       for (int i = 1; i < nx; ++i) {
178         for (int j = 1; j < ny; ++j) {
179           next_u = Relax(u[i][j],
180                          (pde.a * ((u[i+1][j] + u[i-1][j]) / hx / hx + (u[i][j+1] + u[i][j
                                 -1]) / hy / hy) -
181                           (pde.bx * (u[i+1][j] - u[i-1][j]) / hx + pde.by * (u[i][j+1] - u
                                 [i][j-1]) / hy) / 2 - pde.f(x[i], y[j])) / coeff,
182                           relax);
183
184           eps_k = std::max(eps_k, std::abs(next_u - u[i][j]));
185           u[i][j] = next_u;
186         }
187       }
188
189       for (int i = 1; i < nx; ++i) {
190         next_u = Relax(u[i][0], (pde.gamma_y0(x[i]) - pde.alpha_y0 / hy * u[i][1]) / (
                 pde.beta_y0 - pde.alpha_y0 / hy), relax);
191         eps_k = std::max(eps_k, std::abs(next_u - u[i][0]));
192         u[i][0] = next_u;
193
194         next_u = Relax(u[i][ny], (pde.gamma_y1(x[i]) + pde.alpha_y1 / hy * u[i][ny-1])
                 / (pde.beta_y1 + pde.alpha_y1 / hy), relax);
195         eps_k = std::max(eps_k, std::abs(next_u - u[i][ny]));
196         u[i][ny] = next_u;
197       }
198       for (int i = 1; i < ny; ++i) {
199         next_u = Relax(u[0][i], (pde.gamma_x0(y[i]) - pde.alpha_x0 / hx * u[1][i]) / (
                 pde.beta_x0 - pde.alpha_x0 / hx), relax);
200         eps_k = std::max(eps_k, std::abs(next_u - u[0][i]));
201         u[0][i] = next_u;
202
203         next_u = Relax(u[nx][i], (pde.gamma_x1(y[i]) + pde.alpha_x1 / hx * u[nx-1][i])
                 / (pde.beta_x1 + pde.alpha_x1 / hx), relax);
204         eps_k = std::max(eps_k, std::abs(next_u - u[nx][i]));
205         u[nx][i] = next_u;
206       }
207
208       ++iter_count;
209
210     } while (eps_k > eps);
211
212     u[0][0] = (pde.gamma_y0(x[0]) - pde.alpha_y0 / hx * u[0][1]) / (pde.beta_y0 - pde.
            alpha_y0 / hx);
213     u[nx][0] = (pde.gamma_y0(x[nx]) - pde.alpha_y0 / hx * u[nx][1]) / (pde.beta_y0 -
            pde.alpha_y0 / hx);
214     u[0][ny] = (pde.gamma_y1(x[0]) + pde.alpha_y1 / hx * u[0][ny-1]) / (pde.beta_y1 +
            pde.alpha_y1 / hx);
```

```
215     u[nx][ny] = (pde.gamma_y1(x[nx]) + pde.alpha_y1 / hx * u[nx][ny-1]) / (pde.beta_y1+
            pde.alpha_y1 / hx);
216
217     return {x, y, u, iter_count};
218   }
219 }
```