**Московский авиационный институт**

**(национальный исследовательский университет)**

**Институт информационных технологий и прикладной математики**

**Кафедра вычислительной математики и программирования**

**Лабораторная работа №7**

«Численное решение уравнений с частными производными эллиптического типа»

Вариант №3

Студент: Гордионок Е.А.

Группа: М8О-409Б-19

Руководитель: Пивоваров Д.Е.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: 09.12.2022

**Москва 2022**

**Лабораторная работа №7**

Метод конечных разностей для решения задачи эллиптического типа

**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 .

**Описание метода**

Рассмотрим уравнение Пуассона для третьей краевой задачи в прямоугольнике:

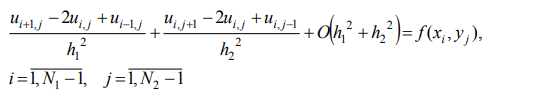
Изображение выглядит как текст

Автоматически созданное описание

Частным случаем уравнение Пуассона является уравнение Лапласа (при f(x, y) = 0).

Для решения такой задачи применяют метод конечных разностей.

Аппроксимируя вторые частные производные, получим следующее выражение для внутренних узлов:



СЛАУ, которая получается при решении данного уравнения, имеет пяти-диагональный вид (каждое уравнение содержит пять неизвестных и при соответствующей нумерации переменных матрица имеет ленточную структуру). Решать ее можно различными методами линейной алгебры, например, итерационными методами, методом матричной прогонки и т.п.

Шаблон данной схемы имеет следующий вид:

Изображение выглядит как антенна

Автоматически созданное описание

Рассмотрим метод простых итераций для решения данной СЛАУ. Для простоты положим h1 = h2 = h, тогда получим (k – номер итерации):

Изображение выглядит как текст

Автоматически созданное описание

Перед решение методом простых итераций необходимо задать начальное приближение.

Условие выхода:



где ε – наперёд заданная точность.

**Аппроксимация граничных условий**

Граничные условия аппроксимируются с первым порядком:

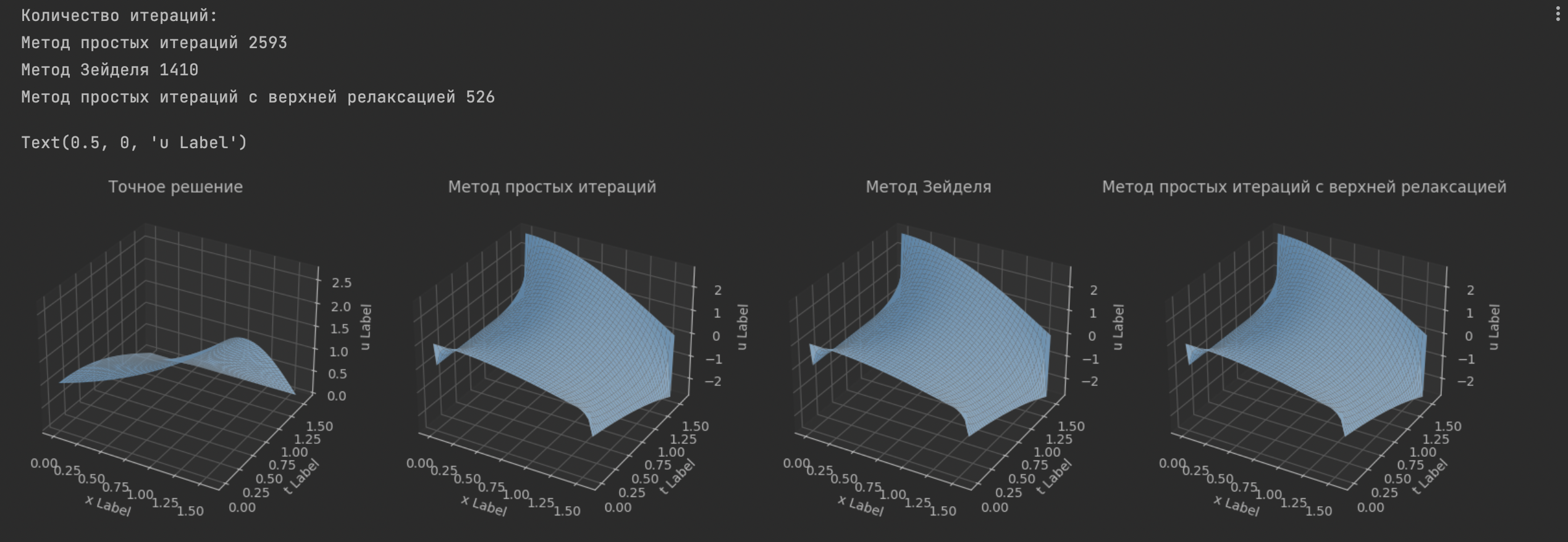
Изображение выглядит как текст

Автоматически созданное описание

**Вариант**

**Text, letter

Description automatically generated**

**Результаты работы программы**

**Выводы**

В данной работе реализована конечно-разностная схемы для решения краевой задачи для дифференциального уравнения эллиптического типа. Для решения дискретного аналога применить следующие методы: метод простых итераций (метод Либмана), метод Зейделя, метод простых итераций с верхней релаксацией. Для сравнения с точным решением вычисляется погрешность как корень квадратный из суммы квадратов погрешностей между точным решением и полученным решением на каждом шаге j для сечения по y (s = 9).

Погрешность составила порядка 10^(-5).

**Приложение. Листинг программы.**

import numpy as np

import matplotlib.pyplot as plt

import math

import copy

# %matplotlib notebook

class Data:

def \_\_init\_\_(self, params):

self.a = params['a']

self.b = params['b']

self.c = params['c']

self.d = params['d']

self.lx = params['lx']

self.ly = params['ly']

self.w = params['w']

self.f = params['f']

self.alpha1 = params['alpha1']

self.alpha2 = params['alpha2']

self.beta1 = params['beta1']

self.beta2 = params['beta2']

self.gamma1 = params['gamma1']

self.gamma2 = params['gamma2']

self.delta1 = params['delta1']

self.delta2 = params['delta2']

self.phi1 = params['phi1']

self.phi2 = params['phi2']

self.phi3 = params['phi3']

self.phi4 = params['phi4']

self.solution = params['solution']

# %%

def diff(L, u, nx, ny):

mx = 0

for i in range(nx):

for j in range(ny):

mx = max(mx, abs(u[i][j] - L[i][j]))

return mx

def compareError(a, b):

err = 0

lst = [abs(i - j) for i, j in zip(a, b)]

for each in lst:

err = max(err, each)

return err

# %%

class ElepticalSolver:

def \_\_init\_\_(self, params, nx, ny):

self.data = Data(params)

iteration = 0

self.eps = 1e-6

self.nx = nx

self.ny = ny

self.hx = self.data.lx / nx

self.hy = self.data.ly / ny

x = np.arange(0, self.data.lx + self.hx, self.hx)

y = np.arange(0, self.data.ly + self.hy, self.hy)

self.x = x

self.y = y

u = self.initalizeU(x, y)

for i in range(1, nx):

for j in range(1, ny):

u[i][j] = u[0][j] + (x[i] - x[0]) \* (u[-1][j] - u[0][j]) / (x[-1] - x[0])

self.u = u

def initalizeU(self, x, y):

u = np.zeros((len(x), len(y)))

for i in range(len(x)):

u[i][0] = self.data.phi3(x[i]) / self.data.gamma1

u[i][-1] = self.data.phi4(x[i]) / self.data.delta1

for j in range(len(y)):

u[0][j] = self.data.phi1(y[j]) / self.data.alpha2

u[-1][j] = self.data.phi2(y[j]) / self.data.beta2

return u

def analyticSolve(self):

nx, ny = self.nx, self.ny

self.hx = self.data.lx / nx

self.hy = self.data.ly / ny

x = np.arange(0, self.data.lx + self.hx, self.hx)

y = np.arange(0, self.data.ly + self.hy, self.hy)

u = []

for yi in y:

u.append([self.data.solution(xi, yi) for xi in x])

# for xi, yi in zip(x, y):

# u.append(self.data.solution(xi, yi))

return u

# метод простых итераций (метод Либмана)

def simpleIterationMethod\_solver(self):

nx, ny = self.nx, self.ny

x, y, u = self.x, self.y, self.u

iteration = 0

cur\_eps = 1e9

while iteration < 10000:

L = copy.deepcopy(u)

u = self.initalizeU(x, y)

for j in range(1, len(y) - 1):

for i in range(1, len(x) - 1):

u[i][j] = (self.hx \* self.hx \* self.data.f(x[i], y[j]) -

(L[i + 1][j] + L[i - 1][j]) - self.data.d \* self.hx \* self.hx \*

(L[i][j + 1] + L[i][j - 1]) /

(self.hy \* self.hy) - self.data.a \* self.hx \* 0.5 \*

(L[i + 1][j] - L[i - 1][j]) - self.data.b \* self.hx \* self.hx \*

(L[i][j + 1] - L[i][j - 1]) /

(2 \* self.hy)) / (self.data.c \* self.hx \* self.hx - 2 \*

(self.hy \* self.hy + self.data.d \* self.hx \* self.hx) /

(self.hy \* self.hy))

last\_eps = cur\_eps

cur\_eps = diff(L, u, nx, ny)

if diff(L, u, nx, ny) <= self.eps or last\_eps < cur\_eps:

break

iteration += 1

return u, iteration

# метод Зейделя

def seidelMethod\_solver(self):

nx, ny = self.nx, self.ny

x, y, u = self.x, self.y, self.u

iteration = 0

cur\_eps = 1e9

while iteration < 10000:

L = copy.deepcopy(u)

u = self.initalizeU(x, y)

for j in range(1, len(y) - 1):

for i in range(1, len(x) - 1):

u[i][j] = ((self.hx \*\* 2) \* self.data.f(x[i], y[j]) -

(L[i + 1][j] + u[i - 1][j]) - self.data.d \* (self.hx \*\* 2) \*

(L[i][j + 1] + u[i][j - 1]) / (self.hy \*\* 2) - self.data.a \* self.hx \* 0.5 \*

(L[i + 1][j] - u[i - 1][j]) - self.data.b \* (self.hx \*\* 2) \*

(L[i][j + 1] - u[i][j - 1]) /

(2 \* self.hy)) / \

(self.data.c \* (self.hx \*\* 2) - 2 \* (self.hy \*\* 2 + self.data.d \* (self.hx \*\* 2)) /

(self.hy \*\* 2))

last\_eps = cur\_eps

cur\_eps = diff(L, u, nx, ny)

if cur\_eps <= self.eps or last\_eps < cur\_eps:

break

iteration += 1

return u, iteration

# метод простых итераций с верхней релаксацией

def simpleIterationMethodRelaxed\_solver(self):

nx, ny = self.nx, self.ny

x, y, u = self.x, self.y, self.u

iteration = 0

cur\_eps = 1e9

while iteration < 10000:

L = copy.deepcopy(u)

u = self.initalizeU(x, y)

for j in range(1, len(y) - 1):

for i in range(1, len(x) - 1):

u[i][j] = (((self.hx \*\* 2) \* self.data.f(x[i], y[j]) -

(L[i + 1][j] + u[i - 1][j]) - self.data.d \* (self.hx \*\* 2) \*

(L[i][j + 1] + u[i][j - 1]) / (self.hy \*\* 2) - self.data.a \* self.hx \* 0.5 \*

(L[i + 1][j] - u[i - 1][j]) - self.data.b \* (self.hx \*\* 2) \*

(L[i][j + 1] - u[i][j - 1]) /

(2 \* self.hy)) / (self.data.c \* (self.hx \*\* 2) - 2 \*

(self.hy \*\* 2 + self.data.d \* (self.hx \*\* 2)) /

(self.hy \*\* 2))) \* self.data.w + (1 - self.data.w) \* L[i][j]

last\_eps = cur\_eps

cur\_eps = diff(L, u, nx, ny)

if diff(L, u, nx, ny) <= self.eps or last\_eps < cur\_eps:

break

iteration += 1

return u, iteration

# %%

nx = 40

ny = 40

# %%

params = {

'a': 0,

'b': 0,

'c': 0,

'd': 1,

'lx': 1,

'ly': np.pi / 2,

'w': 1.5,

'f': lambda x, y: 0,

'alpha1': 0,

'alpha2': 1,

'beta1': 0,

'beta2': 1,

'gamma1': 1,

'gamma2': 0,

'delta1': 1,

'delta2': 0,

'phi1': lambda y: np.cos(y),

'phi2': lambda y: np.exp(1) \* np.cos(y),

'phi3': lambda x: 0,

'phi4': lambda x: -np.exp(x),

'solution': lambda x, y: np.exp(x) \* np.cos(y),

}

# %%

solver = ElepticalSolver(params, nx, ny)

X = np.arange(0, np.pi / 2 + np.pi / 2 / nx, np.pi / 2 / nx)

Y = np.arange(0, np.pi / 2 + np.pi / 2 / ny, np.pi / 2 / ny)

# %%

fig = plt.figure(figsize=(20, 12))

ax = fig.add\_subplot(1, 4, 1, projection='3d')

ax.set\_title('Точное решение')

U = solver.analyticSolve()

W, Q = np.meshgrid(X, Y)

ax.plot\_surface(W, Q, np.array(U))

ax.set\_xlabel('x Label')

ax.set\_ylabel('t Label')

ax.set\_zlabel('u Label')

print('Количество итераций:')

# \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ax = fig.add\_subplot(1, 4, 2, projection='3d')

ax.set\_title('Метод простых итераций')

u, iteration = solver.simpleIterationMethod\_solver()

print('Метод простых итераций', iteration)

W, Q = np.meshgrid(X, Y)

ax.plot\_surface(W, Q, np.array(u))

ax.set\_xlabel('x Label')

ax.set\_ylabel('t Label')

ax.set\_zlabel('u Label')

# \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ax = fig.add\_subplot(1, 4, 3, projection='3d')

ax.set\_title('Метод Зейделя')

u, iteration = solver.seidelMethod\_solver()

print('Метод Зейделя', iteration)

W, Q = np.meshgrid(X, Y)

ax.plot\_surface(W, Q, np.array(u))

ax.set\_xlabel('x Label')

ax.set\_ylabel('t Label')

ax.set\_zlabel('u Label')

# \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ax = fig.add\_subplot(1, 4, 4, projection='3d')

ax.set\_title('Метод простых итераций с верхней релаксацией')

u, iteration = solver.simpleIterationMethodRelaxed\_solver()

print('Метод простых итераций с верхней релаксацией', iteration)

W, Q = np.meshgrid(X, Y)

ax.plot\_surface(W, Q, np.array(u))

ax.set\_xlabel('x Label')

ax.set\_ylabel('t Label')

ax.set\_zlabel('u Label')