**Московский авиационный институт**

**(национальный исследовательский университет)**

**Институт информационных технологий и прикладной математики**

**Кафедра вычислительной математики и программирования**

Лабораторная работа №5

«Численное решение уравнений с частными производными параболического типа»

Вариант №3

Студент: Гордионок Е.А.

Группа: М8О-409Б-19

Руководитель: Пивоваров Д.Е.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: 28.10.2022

**Москва 2022**

**Лабораторная работа №5**

Численное решение уравнений с частными производными параболического типа. Понятие о методе конечных разностей. Конечно-разностные схемы.

**Задача**

Используя явную и неявную конечно-разностные схемы, а также схему Кранка - Николсона, решить начально-краевую задачу для дифференциального уравнения параболического типа. Осуществить реализацию трех вариантов аппроксимации граничных условий, содержащих производные: двухточечная аппроксимация с первым порядком, трехточечная аппроксимация со вторым порядком, двухточечная аппроксимация со вторым порядком. 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.

**Описание метода**

Рассматривается решение уравнения теплопроводности с граничными условиями второго рода на обоих концах интервала, т.е. рассматривается следующая задача:

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание



Для решения такой задачи применяют метод конченых разностей. Для этого вводят понятие разностной сетки



с пространственным шагом h и шагом по времени τ.

Введём понятие сеточной функции. Сеточной функцией называют следующее отображение целых аргументов j и k:

Затем происходит аппроксимация производной по времени и второй производной по пространству.

Если аппроксимировать вторую производную по пространству на нижнем временном слое, то получим **явную** конечно-разностную схему.

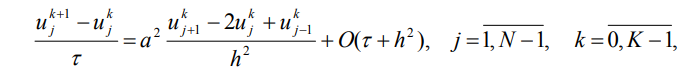
Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Явная конечно-разностная схема:



где для каждого j-го уравнения неизвестна только одна величина , которая может быть явно выражена из этого уравнения:



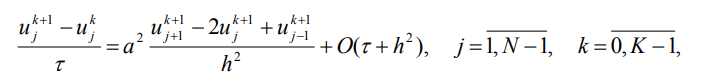
Для устойчивости данной схемы накладывается ограничение на σ: σ <= 1/2.

Если аппроксимировать производную по пространству на верхнем временном слое, то получим **неявную** конечно-разностную схему

Изображение выглядит как текст, антенна

Автоматически созданное описание

Неявная схема:



где для нахождения сеточной функции на верхнем временном слое необходимо решать СЛАУ с трёхдиагональной матрицей:

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

**Явно-неявная** конечно-разностная схема имеет следующий вид:

Изображение выглядит как текст, антенна

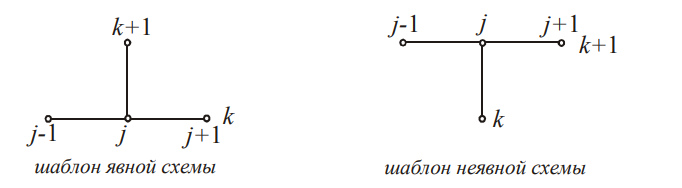
Автоматически созданное описание

где 0ϴ1.

При ϴ = 0 получается явная схема, при ϴ = 1 – неявная, при ϴ = 1/2 - схема Кранка-Николсона.

Здесь так же, как и в неявной схеме для нахождения на каждом шаге необходимо решать СЛАУ с трёхдиагональной матрицей.

Шаблоны трёх схем:



Шаблон схемы Кранка-Николсона:

Изображение выглядит как часы, датчик

Автоматически созданное описание

**Аппроксимация граничных условий**

Рассмотрим 3 способа аппроксимации граничных условий 2Т1П, 3Т2П, 2Т2П.

Двухточечная, с первым порядком:



Трёхточечная, со вторым порядком:

Изображение выглядит как текст

Автоматически созданное описание

При использовании неявной и явно-неявной схемы СЛАУ теряет трёхдиагональность, поэтому сначала необходимо привести СЛАУ к трёхдиагональному виду линейной комбинацией первой строки со второй, предпоследней с последней.

Двухточечная, со вторым порядком:

Для того, чтобы получить эти формулы для начала раскладывают в ряд Тейлора в окрестности x = 0 и в окрестности x = l:

Изображение выглядит как текст

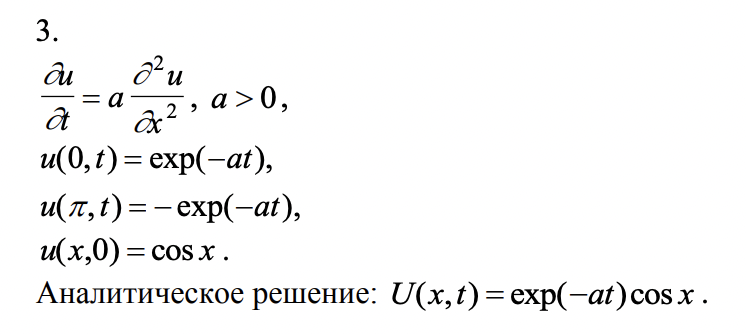
Автоматически созданное описание

Далее берут информацию из исходного уравнения, выражая оттуда вторую производную и подставляя это выражение, получают:

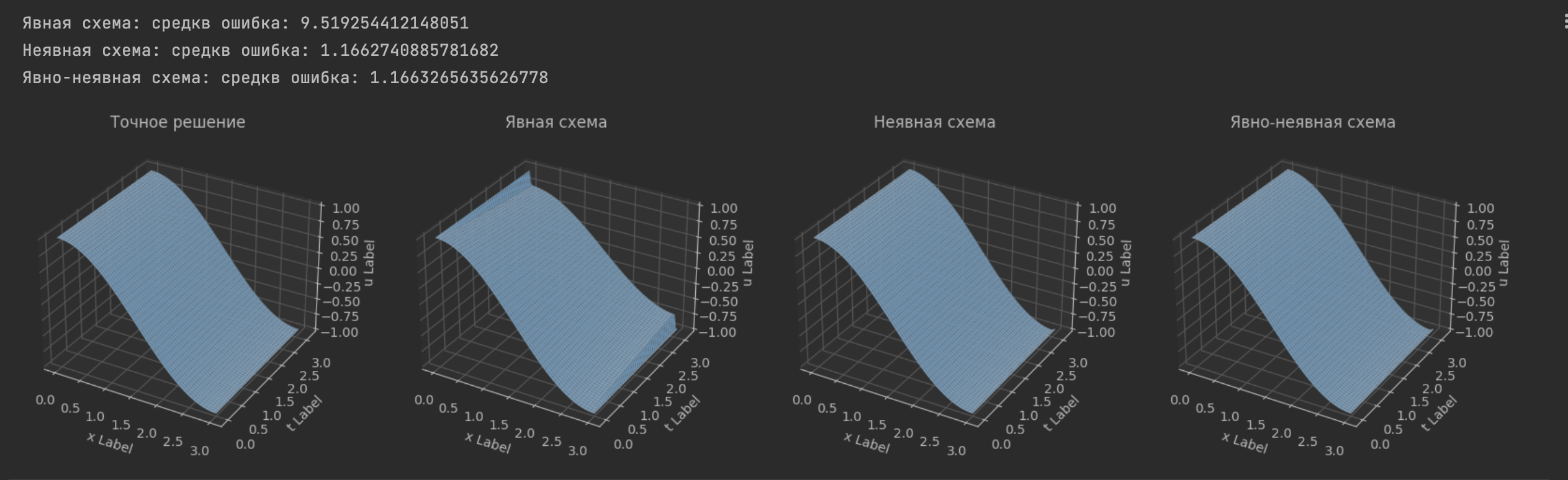
Изображение выглядит как текст, стол

Автоматически созданное описание

**Вариант**

****

**Результаты работы программы**



**Выводы**

В данной работе реализована явная и неявная конечно-разностные схемы, а также схема Кранка – Николсона для решения начально-краевую задачу для дифференциального уравнения параболического типа. Осуществлена реализацию трех вариантов аппроксимации граничных условий, содержащих производные: двухточечная аппроксимация с первым порядком, трехточечная аппроксимация со вторым порядком, двухточечная аппроксимация со вторым порядком.

**Приложение. Листинг программы.**

import numpy as np

import matplotlib.pyplot as plt

import math

class ParabolicTypeEquations:

def \_\_init\_\_(self, func, phi\_0, phi\_l, xi, x, t, a, N=100):

self.func = func

self.phi\_0 = phi\_0

self.phi\_l = phi\_l

self.xi = xi

self.x = x

self.t = t

self.N = N

self.h = (x[-1] - x[1]) / N

self.tau = (t[-1] - t[1]) / N

self.sigma = a \* a \* self.tau / self.h / self.h

# Точное решение

def exact\_solution(self):

return np.array([[self.func(x, t) for t in self.t] for x in self.x])

# Явная конечно-разностная схема

def explicit\_finite\_difference\_scheme(self):

u = [[0 for j in range(len(self.t))] for i in range(len(self.x))]

for j in range(len(self.x)):

u[j][0] = self.xi(self.x[j])

for k in range(len(self.t) - 1):

u[0][k + 1] = self.phi\_0(self.t[k + 1])

for k in range(len(self.t) - 1):

u[-1][k + 1] = self.phi\_l(self.t[k + 1])

for j in range(1, len(self.x) - 1):

for k in range(len(self.t) - 1):

u[j][k + 1] = self.sigma \* u[j + 1][k] + (1 - 2 \* self.sigma) \* u[j][k] + self.sigma \* u[j - 1][k]

return u

# Неявная схема

def implicit\_finite\_difference\_scheme(self):

u = [[0 for j in range(len(self.t))] for i in range(len(self.x))]

for k in range(len(self.t) - 1):

u[0][k + 1] = self.phi\_0(self.t[k + 1])

for k in range(len(self.t) - 1):

u[-1][k + 1] = self.phi\_l(self.t[k + 1])

for j in range(len(self.x)):

u[j][0] = self.xi(self.x[j])

for k in range(len(self.t) - 1):

A = [self.sigma if i != 0 else 0 for i in range(self.N - 2)]

B = [-(1 + 2 \* self.sigma) for i in range(self.N - 2)]

C = [self.sigma if i != self.N - 3 else 0 for i in range(self.N - 2)]

D = [-u[j][k] for j in range(2, self.N - 2)]

D.insert(0, -(u[1][k] + self.sigma \* self.phi\_0(self.t[k + 1])))

D.append(-(u[self.N - 1][k] + self.sigma \* self.phi\_l(self.t[k + 1])))

uk = self.progonka(A, B, C, D)

for j in range(1, self.N - 1):

u[j][k + 1] = uk[j - 1]

return u

# Явно-неявня схема

def explicit\_implicit\_finite\_difference\_scheme(self, teta):

u = [[0 for j in range(len(self.t))] for i in range(len(self.x))]

for k in range(len(self.t) - 1):

u[0][k + 1] = self.phi\_0(self.t[k + 1])

for k in range(len(self.t) - 1):

u[-1][k + 1] = self.phi\_l(self.t[k + 1])

for j in range(len(self.x)):

u[j][0] = self.xi(self.x[j])

for k in range(len(self.t) - 1):

A = [teta \* self.sigma if i != 0 else 0 for i in range(self.N - 2)]

B = [-(1 + 2 \* teta \* self.sigma) for i in range(self.N - 2)]

C = [teta \* self.sigma if i != self.N - 3 else 0 for i in range(self.N - 2)]

D = [- (1 - teta) \* self.sigma \* u[j + 1][k] - (1 - 2 \* (1 - teta) \* self.sigma) \* u[j][k] - (

1 - teta) \* self.sigma \* u[j - 1][k] for j in range(2, self.N - 2)]

D.insert(0, -(teta \* self.sigma \* self.phi\_0(self.t[k + 1]) + (1 - teta) \* self.sigma \* u[2][k] + (

1 - 2 \* (1 - teta) \* self.sigma) \* u[1][k] + (1 - teta) \* self.sigma \* self.phi\_0(self.t[k])))

D.append(-(teta \* self.sigma \* self.phi\_l(self.t[k + 1]) + (1 - teta) \* self.sigma \* self.phi\_l(self.t[k]) + (

1 - 2 \* (1 - teta) \* self.sigma) \* u[self.N - 1][k] + (1 - teta) \* self.sigma \* u[self.N - 2][

k]))

uk = self.progonka(A, B, C, D)

for j in range(1, self.N - 1):

u[j][k + 1] = uk[j - 1]

return u

def progonka(self, a, b, c, d):

n = len(a)

for i in range(n):

if math.fabs(b[i]) < math.fabs(a[i]) + math.fabs(c[i]):

raise Exception

# Формирование массивов P, Q (Расчет значений) ((Прямой ход))

P, Q = [-c[0] / b[0]], [d[0] / b[0]]

for i in range(1, n):

P.append(-c[i] / (b[i] + a[i] \* P[i - 1]))

Q.append((d[i] - a[i] \* Q[i - 1]) / (b[i] + a[i] \* P[i - 1]))

# Вычисление решения системы (Обратный ход)

x = [Q[n - 1]]

for i in range(1, n):

x.append(P[n - 1 - i] \* x[i - 1] + Q[n - 1 - i])

return list(reversed(x))

def main():

a = 0.01

U = lambda x, t: np.exp(-a \* t) \* np.cos(x)

phi\_0 = lambda t: math.exp(-a \* t)

phi\_l = lambda t: -math.exp(-a \* t)

xi = lambda x: math.cos(x)

N = 100

X = np.linspace(0, 3, N)

T = np.linspace(0, math.pi, N)

equations = ParabolicTypeEquations(U, phi\_0, phi\_l, xi, X, T, a)

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

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

ax.set\_title('Точное решение')

u = equations.exact\_solution()

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

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, 2, projection='3d')

ax.set\_title('Явная схема')

u = equations.explicit\_finite\_difference\_scheme()

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

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

ax.set\_xlabel('x Label')

ax.set\_ylabel('t Label')

ax.set\_zlabel('u Label')

print('Явная схема: средкв ошибка:',

math.sqrt(sum([sum([(U(X[i], T[j]) - u[i][j]) \*\* 2 for j in range(len(T))]) for i in range(len(X))])))

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

ax.set\_title('Неявная схема')

u = equations.implicit\_finite\_difference\_scheme()

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

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

ax.set\_xlabel('x Label')

ax.set\_ylabel('t Label')

ax.set\_zlabel('u Label')

print('Неявная схема: средкв ошибка:',

math.sqrt(sum([sum([(U(X[i], T[j]) - u[i][j]) \*\* 2 for j in range(len(T))]) for i in range(len(X))])))

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

ax.set\_title('Явно-неявная схема')

u = equations.explicit\_implicit\_finite\_difference\_scheme(1 / 2)

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

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

ax.set\_xlabel('x Label')

ax.set\_ylabel('t Label')

ax.set\_zlabel('u Label')

print('Явно-неявная схема: средкв ошибка:',

math.sqrt(sum([sum([(U(X[i], T[j]) - u[i][j]) \*\* 2 for j in range(len(T))]) for i in range(len(X))])))

if \_\_name\_\_ == "\_\_main\_\_":

main()