Московский авиационный институт

(Национальный исследовательский университет)

Институт №8 «Компьютерные науки и прикладная математика»

**Отчет по лабораторной работе №1**

по курсу «Численные методы»

Вариант - 2

**Выполнила студентка**: Бондарева Е. Е.

**Группа:** М8О-405Б-21

# Преподаватель: Демидова О. Л.

Оценка: !

Дата: !

Москва, 2024

**Лабораторная работа №1**

**Задание:**

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 . Исследовать зависимость погрешности от сеточных параметров .

, ,



.

Аналитическое решение: 

**Теоретические сведения:**

Так как на границах x = 0 и x = l = 1 заданы краевые условия вида u (0, t) = 0;

u (1, t) = 1, т.е. граничные условия первого рода, и, кроме того, заданы начальные условия, то решаемую задачу называют *первой начально-краевой задачей для уравнения теплопроводности*. Подобные задачи решаются методом конечных разностей. Необходимо воспользоваться следующими формулами для решения поставленной ранее задачи:

Явная конечно-разностная схема:



Неявная конечно-разностная схема:



Схема Кранка-Николсона при Ѳ = 0.5:



(1-Ѳ) – вес для явной части, Ѳ – вес неявной части конечно-разностной схемы.

**Код программы:**

import matplotlib.pyplot as plt

import numpy as np

plt.rcParams['figure.figsize'] = [10, 10]

# Задаем начальные краевые условия:

def u\_0\_t(t):

return 0

def u\_l\_t(t):

return 1

def u\_x\_0(x):

return x + np.sin(np.pi \* x)

# Аналитическое решение:

def solution(x, t, a):

return x + np.exp(-(np.pi \* np.pi) \* a \* t) \* np.sin(np.pi \* x)

# Определяем ошибки:

def error(h, tau, K, l, T, a, mesh, solution):

x\_array = np.arange(0, l + h, h)

errors = np.zeros(K + 1)

for t in range(0, K):

u\_correct = solution(x\_array, t \* tau, a)

u\_calculated = mesh[t]

errors[t] = np.amax(np.abs(u\_correct - u\_calculated))

return errors

# График решений:

def graph\_solution(h, tau, K, l, T, a, mesh, solution):

x\_array = np.arange(0, l + h, h)

fig, ax = plt.subplots()

t = [int(K \* 0.05), int(K \* 0.1), int(K \* 0.25)]

colors = ['pink', 'yellow', 'aqua']

for i in range(len(t)):

u\_correct = solution(x\_array, t[i] \* tau, a)

u\_calculated = mesh[t[i]]

ax.plot(x\_array, u\_correct, color=colors[i], label='t=%s' % round(t[i] \* tau, 2))

ax.plot(x\_array, u\_calculated, color=colors[i], linestyle='--')

ax.set\_xlabel('x')

ax.set\_ylabel('U(t, x)')

ax.grid()

ax.legend()

plt.show()

# График ошибок:

def graph\_errors(T, tau, errors):

t\_array = np.arange(0, T + tau, tau)

fig, ax = plt.subplots()

ax.plot(t\_array, errors, color='orange')

ax.set\_xlabel('t')

ax.set\_ylabel('delta(t)')

ax.grid()

ax.legend()

plt.show()

# Метод прогонки:

def sweep\_mehod(A, b):

p = np.zeros(len(b))

q = np.zeros(len(b))

# Прямой ход: поиск прогоночных коэффициентов P и Q

p[0] = -A[0][1] / A[0][0]

q[0] = b[0] / A[0][0]

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

p[i] = -A[i][i + 1] / (A[i][i] + A[i][i - 1] \* p[i - 1])

q[i] = (b[i] - A[i][i - 1] \* q[i - 1]) / (A[i][i] + A[i][i - 1] \* p[i- 1])

p[-1] = 0

q[-1] = (b[-1] - A[-1][-2] \* q[-2]) / (A[-1][-1] + A[-1][-2] \* p[-2])

# Обратный ход: поиск x

x = np.zeros(len(b))

x[-1] = q[-1]

for i in reversed(range(len(b) - 1)):

x[i] = p[i] \* x[i + 1] + q[i]

return x

# Явная схема:

def explicit(a, T, l, N, K, h, tau, sigma):

# Проверка на устойчивость:

if (sigma > 0.5):

raise Exception("Измените параметры сетки")

mesh = np.zeros((K + 1, N + 1))

# Начальное условие. Заполнение нижнего слоя:

for j in range(N + 1):

mesh[0][j] = u\_x\_0(j \* h)

# Граничные условия. Заполнение первого и последнего столбца:

for i in range(1, K):

mesh[i][0] = u\_0\_t(i \* tau)

mesh[i][N] = u\_l\_t(i \* tau)

# Заполнение верхних слоев. Явная конечно-разностная схема:

for i in range(1, K):

for j in range(1, N):

mesh[i][j] = sigma \* mesh[i - 1][j + 1] + (1 - 2 \* sigma) \* mesh[i - 1][j] + sigma \* mesh[i - 1][j - 1]

return mesh

# Неявная схема:

def implicit(a, T, l, N, K, h, tau, sigma):

a\_j = sigma

b\_j = -(1 + 2 \* sigma)

c\_j = sigma

mesh = np.zeros((K + 1, N + 1))

# Начальное условие. Заполнение нижнего слоя:

for j in range(N + 1):

mesh[0][j] = u\_x\_0(j \* h)

# Граничные условия. Заполнение первого и последнего столбца:

for i in range(1, K):

mesh[i][0] = u\_0\_t(i \* tau)

mesh[i][N] = u\_l\_t(i \* tau)

# Заполнение верхних слоев. Явная конечно-разностная схема:

for i in range(1, K + 1):

matrix = np.zeros((N - 1, N - 1))

d = np.zeros(N - 1)

matrix[0][0] = b\_j

matrix[0][1] = c\_j

d[0] = -(mesh[i - 1][1] + sigma \* u\_0\_t(i \* tau))

for j in range(1, N - 2):

matrix[j][j - 1] = a\_j

matrix[j][j] = b\_j

matrix[j][j + 1] = c\_j

d[j] = -mesh[i - 1][j + 1]

matrix[N - 2][N - 3] = a\_j

matrix[N - 2][N - 2] = b\_j

d[N - 2] = -(mesh[i - 1][N - 1] + sigma \* u\_l\_t(i \* tau))

# СЛАУ методом прогонки:

solve = sweep\_mehod(matrix, d)

mesh[i][1:N] = solve

return mesh

# Схема Кранка-Николсона:

def CrankNicholson(teta, a, T, l, N, K, h, tau, sigma):

if teta < 0 or teta > 1:

raise Exception("Teta должна быть от 0 до 1")

elif teta == 0:

return explicit(a, T, l, N, K, h, tau)

elif teta == 1:

return implicit(a, T, l, N, K, h, tau)

a\_j = sigma \* teta

b\_j = -(1 + 2 \* sigma \* teta)

c\_j = sigma \* teta

mesh = np.zeros((K + 1, N + 1))

# Начальное условие. Заполнение нижнего слоя:

for j in range(N + 1):

mesh[0][j] = u\_x\_0(j \* h)

# Граничные условия. Заполнение первого и последнего столбца:

for i in range(1, K):

mesh[i][0] = u\_0\_t(i \* tau)

mesh[i][N] = u\_l\_t(i \* tau)

# Заполнение верхних слоев. Явная конечно-разностная схема:

for i in range(1, K + 1):

matrix = np.zeros((N - 1, N - 1))

d = np.zeros(N - 1)

matrix[0][0] = b\_j

matrix[0][1] = c\_j

d[0] = -(mesh[i - 1][1] + sigma \* (1 - teta) \* (mesh[i - 1][2] - 2 \* mesh[i - 1][1] + mesh[i - 1][0]) + sigma \* teta \* u\_0\_t(i \* tau))

for j in range(1, N - 2):

matrix[j][j - 1] = a\_j

matrix[j][j] = b\_j

matrix[j][j + 1] = c\_j

d[j] = -(mesh[i - 1][j + 1] + sigma \* (1 - teta) \* (mesh[i - 1][j + 2] - 2 \* mesh[i - 1][j + 1] + mesh[i- 1][j]))

matrix[N - 2][N - 3] = a\_j

matrix[N - 2][N - 2] = b\_j

d[N - 2] = -(mesh[i - 1][N - 1] + sigma \* (1 - teta) \* (mesh[i - 1][N] - 2 \* mesh[i - 1][N - 1] + mesh[i - 1][N -2]) + sigma \* teta \* u\_l\_t(i \* tau))

solve = sweep\_mehod(matrix, d)

mesh[i][1:N] = solve

return mesh

def main():

a = 1

T = 1

l = 1

N = 10

sigma = 0.4

h = l / N

tau = sigma \* h \*\* 2 / a

K = int(round(T / tau))

print("a =", a)

print("T =", T)

print("l =", l)

print("N =", N)

print("K =", K)

print("h =", h)

print("tau =", tau)

print("sigma =", sigma)

mesh1 = explicit(a, T, l, N, K, h, tau, sigma)

graph\_solution(h, tau, K, l, T, a, mesh1, solution)

errors1 = error(h, tau, K, l, T, a, mesh1, solution)

graph\_errors(T, tau, errors1)

mesh2 = implicit(a, T, l, N, K, h, tau, sigma)

graph\_solution(h, tau, K, l, T, a, mesh2, solution)

errors2 = error(h, tau, K, l, T, a, mesh2, solution)

graph\_errors(T, tau, errors2)

mesh3 = CrankNicholson(0.5, a, T, l, N, K, h, tau, sigma)

graph\_solution(h, tau, K, l, T, a, mesh3, solution)

errors3 = error(h, tau, K, l, T, a, mesh3, solution)

graph\_errors(T, tau, errors3)

main()

**Результат работы программы:**

a = 1

T = 1

l = 1

N = 10

K = 250

h = 0.1

tau = 0.004000000000000001

sigma = 0.4

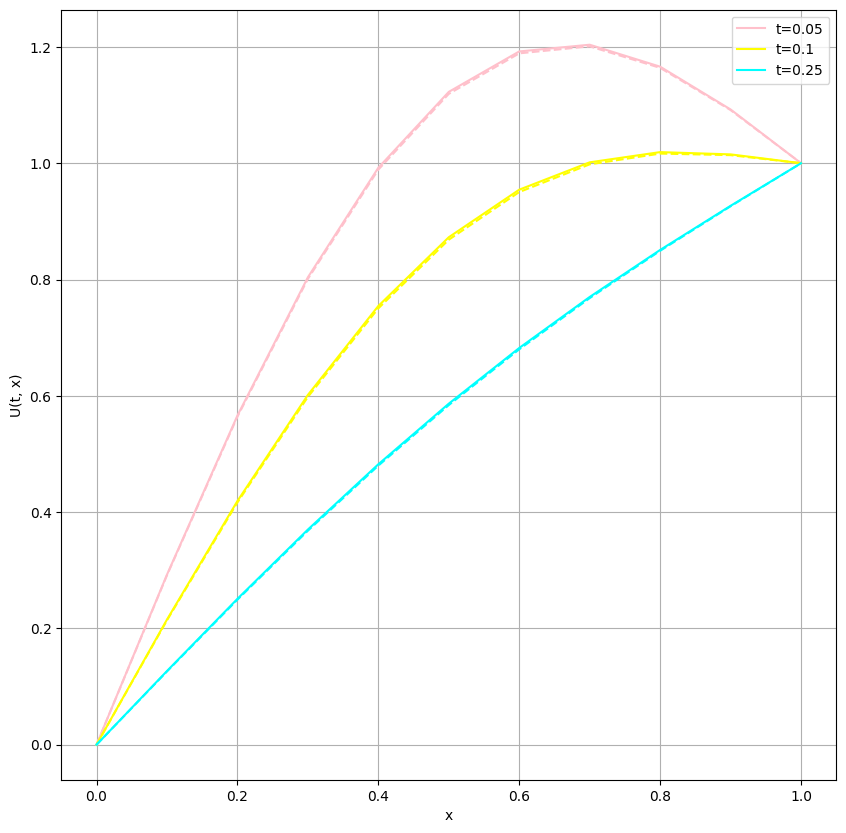
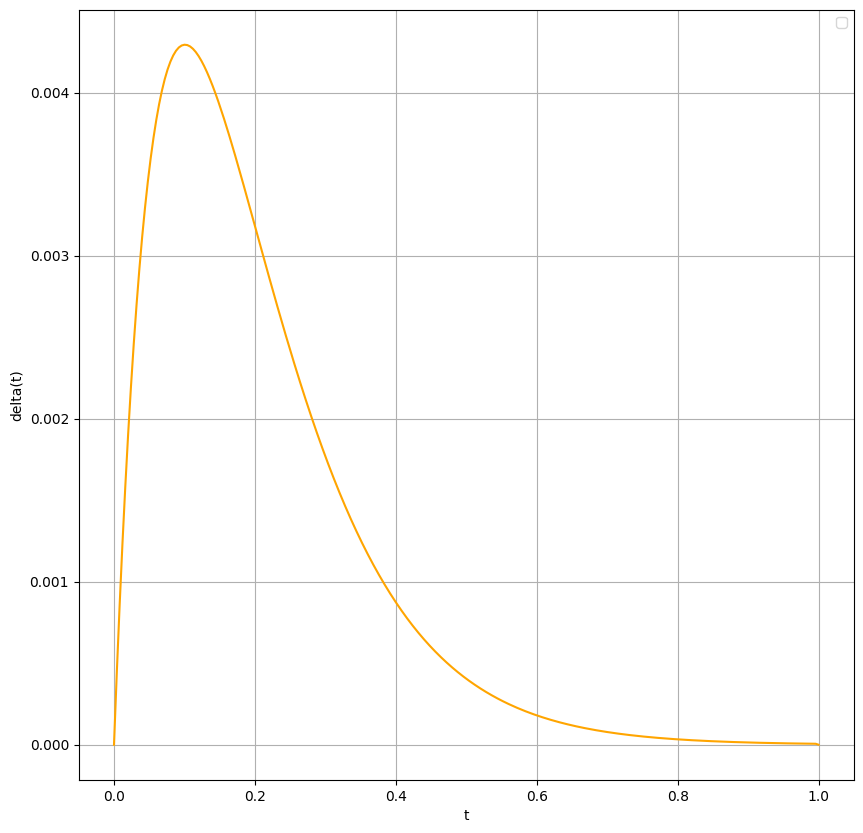
****

Рис. 1. Явная схема

****  
Рис. 2. График погрешности явной схемы

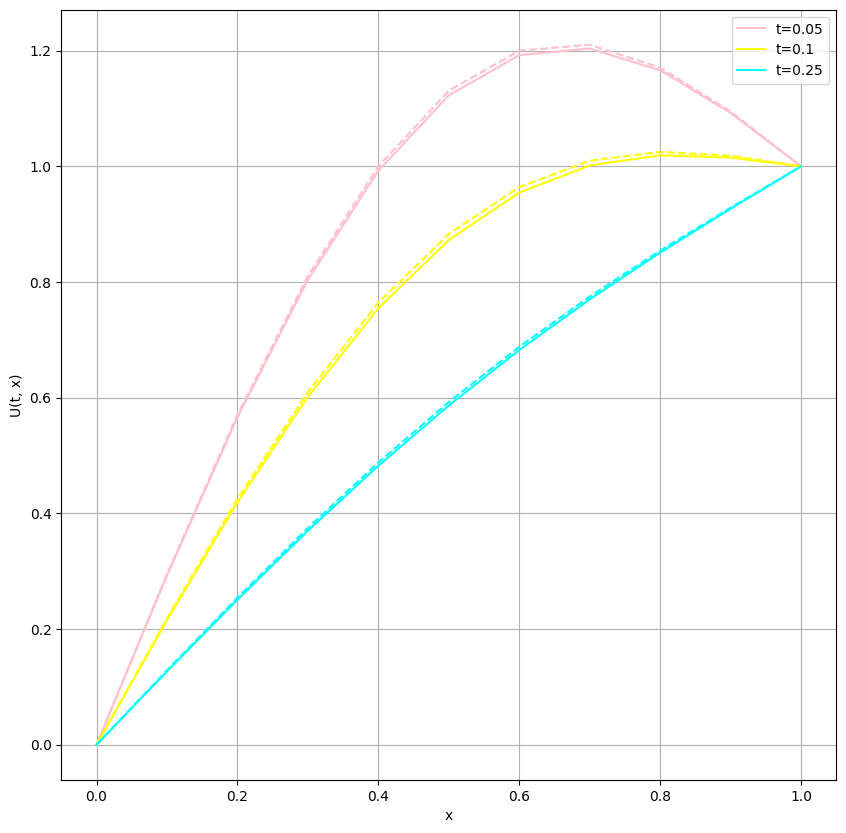
****

Рис. 3. Неявная схема

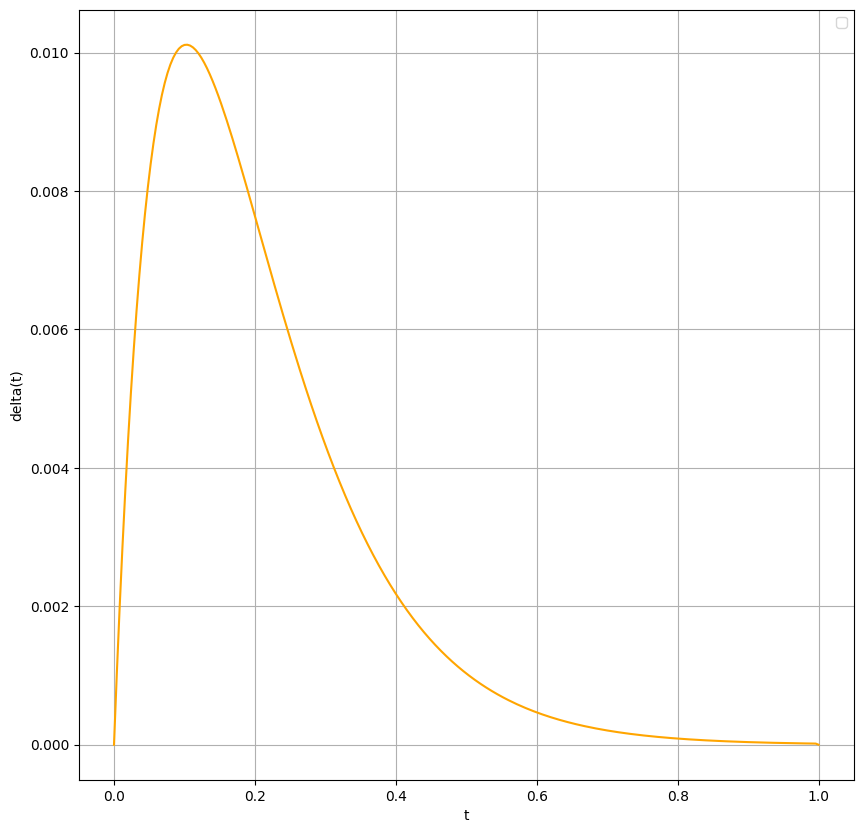
****

Рис. 4. График погрешности неявной схемы

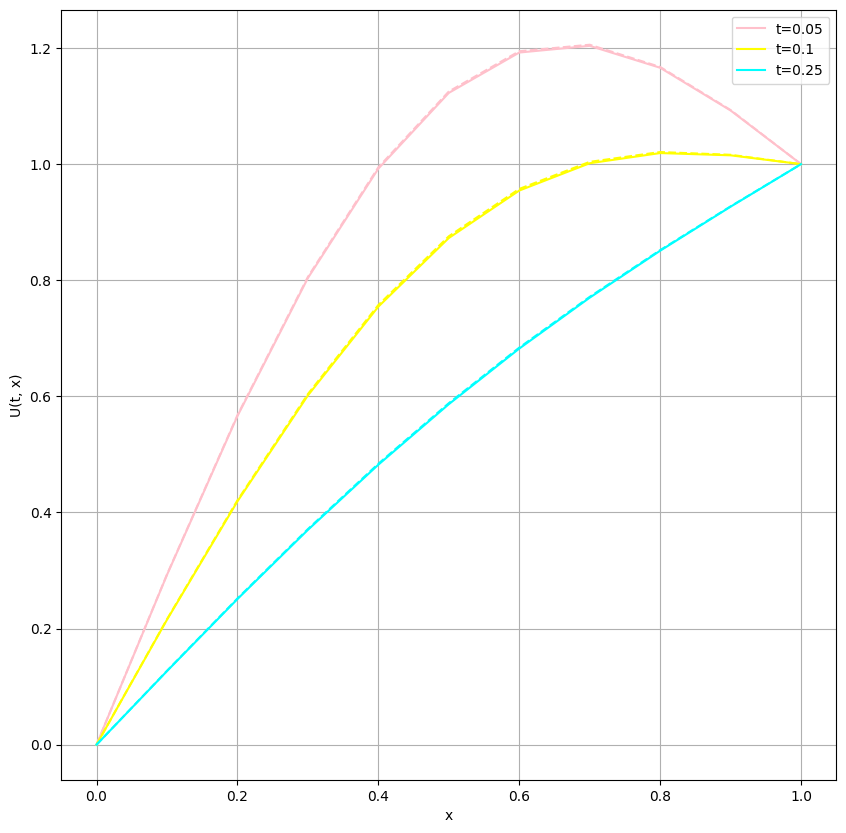
****

Рис. 5. Схема Кранка-Николсона

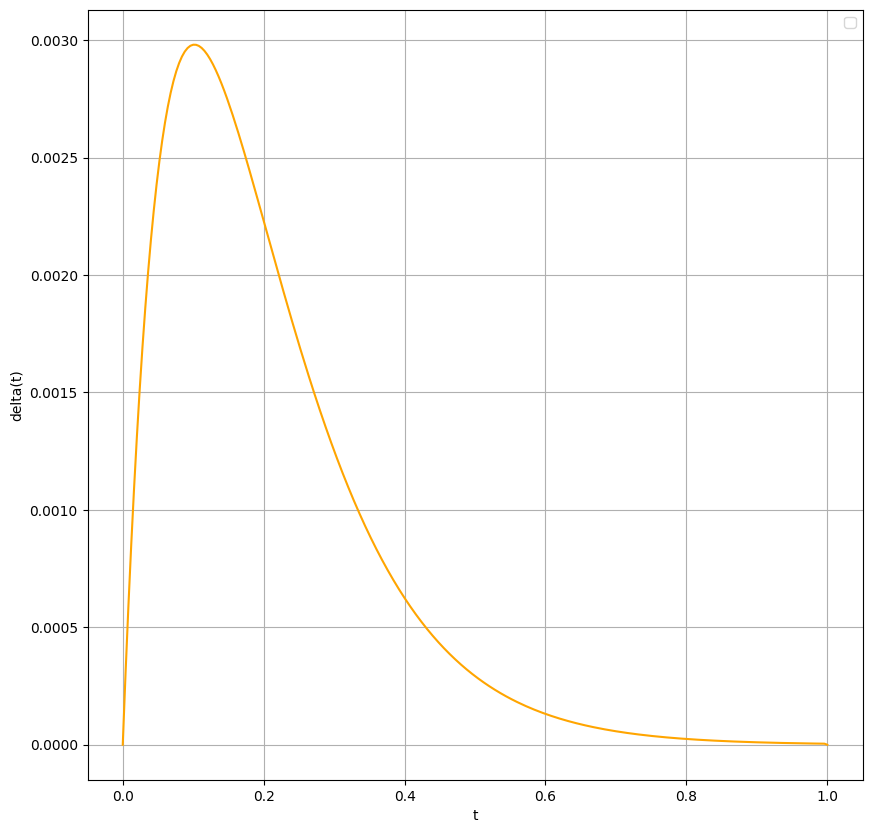
****

Рис. 6. График погрешности схемы Кранка-Николсона

**Вывод:** в результате выполнения лабораторной работы была решена начально-краевая задача для дифференциального уравнения параболического типа; в различные моменты времени вычислена погрешность численного решения путем сравнения результатов с аналитическим решением  и исследована зависимость погрешности от сеточных параметров .

Проанализировав полученные результаты, можно сделать вывод, что метод Кранка-Николсона лучше сработал. При увеличении длины шага по координате 𝑥 погрешность вычислений закономерно увеличивается.