Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Методы численного анализа

**ОТЧЁТ**

к лабораторной работе

на тему

Аппроксимация граничных условий второго рода в методе конечных разностей на примере уравнения теплопроводности

Выполнил: студент группы 153501

Тимофеев Кирилл Андреевич

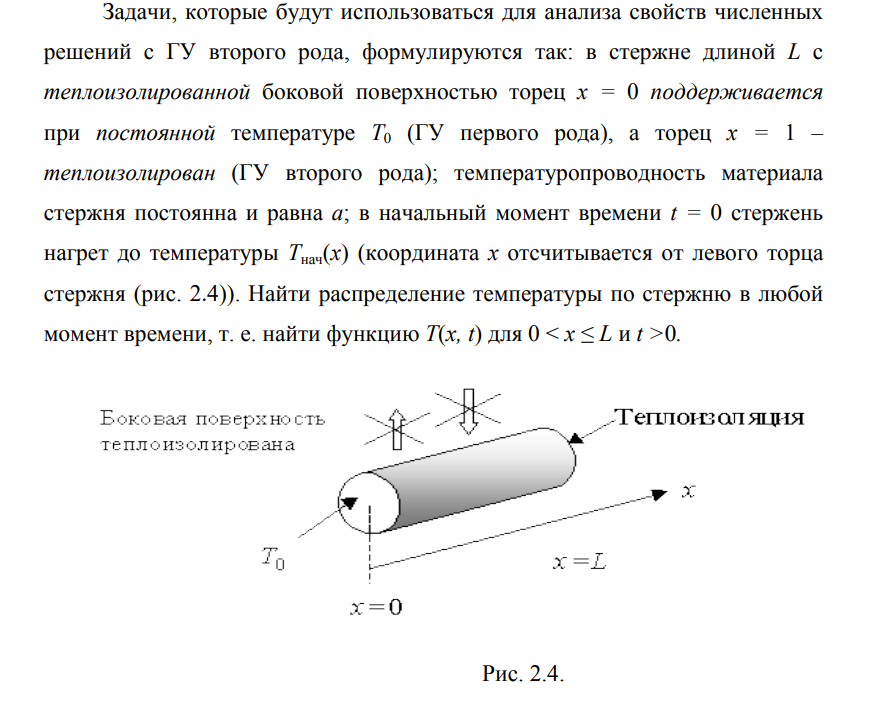
Проверил: Анисимов Владимир Яковлевич

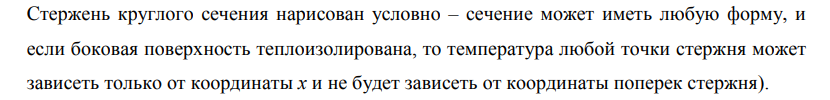
Минск 2023

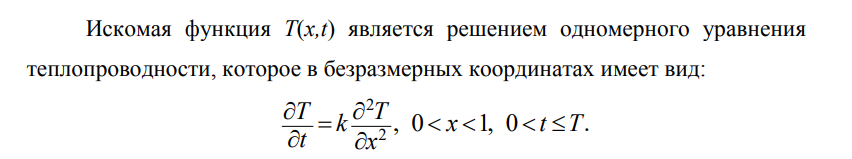
**Цели выполнения задания**

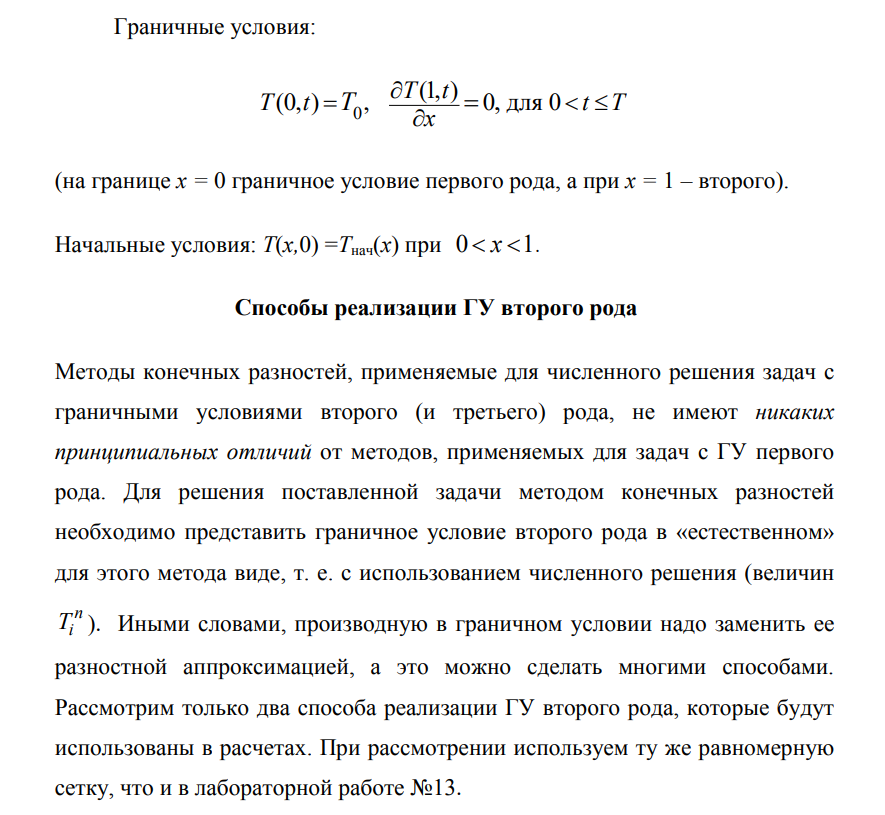
Ознакомиться с наиболее часто применяемыми способами аппроксимации граничных условий второго рода (граничных условий (ГУ) Неймана) в методе конечных разностей (на примере ГУ для одномерного нестационарного уравнения теплопроводности).

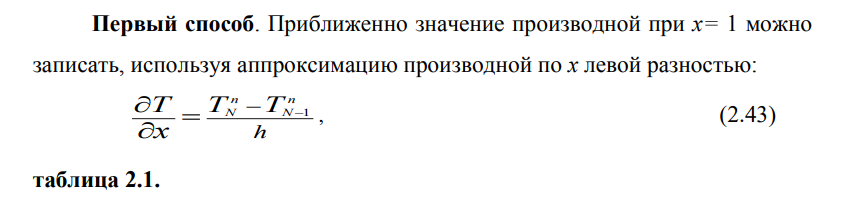
**Краткие теоретические сведения**

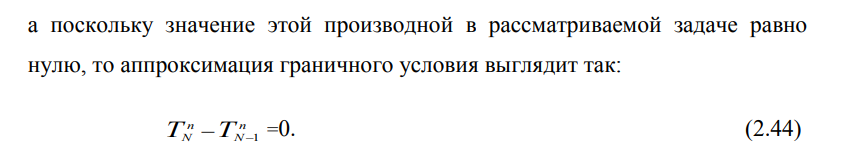


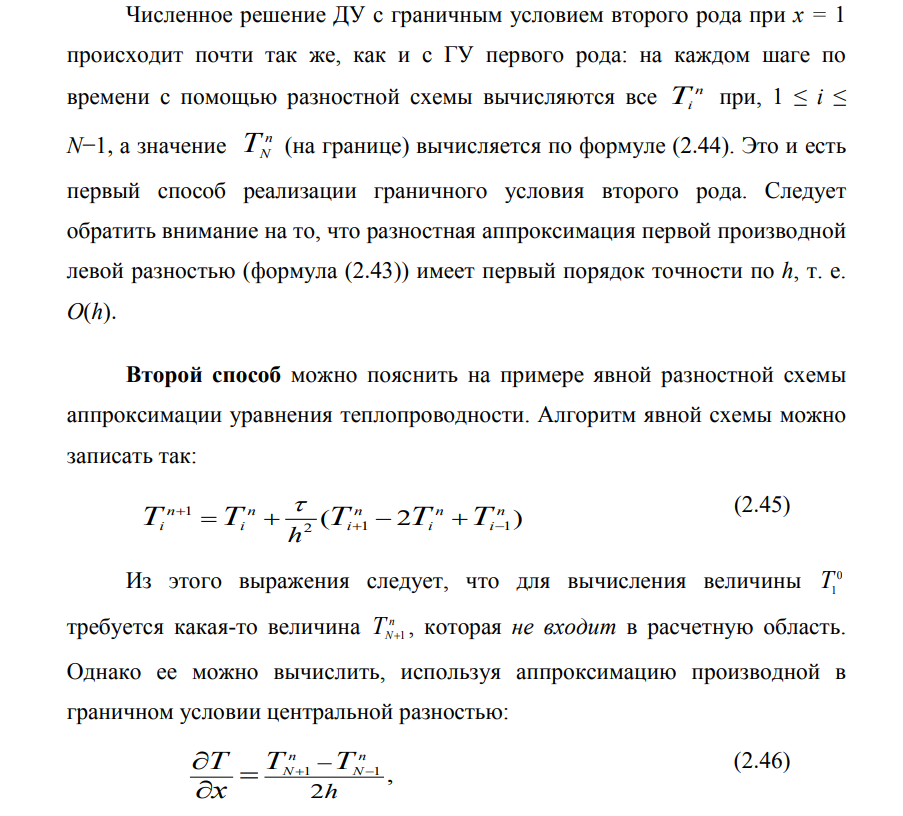


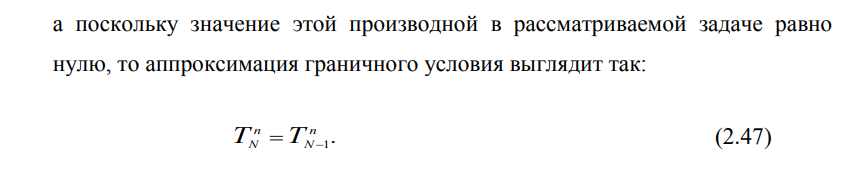


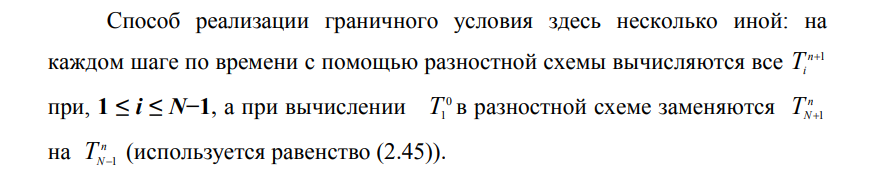


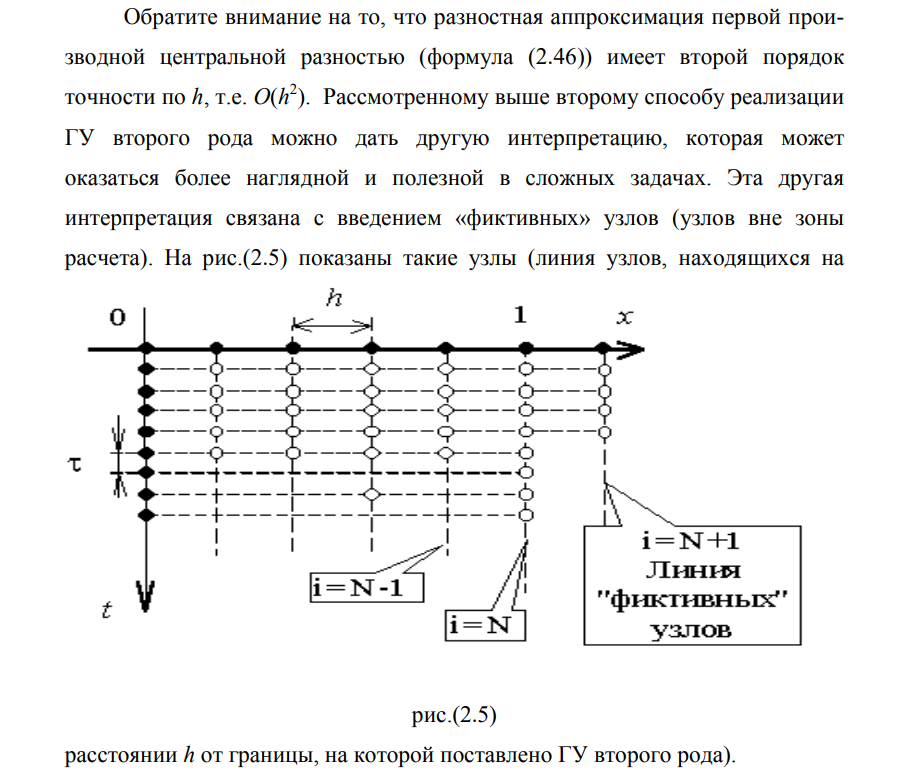


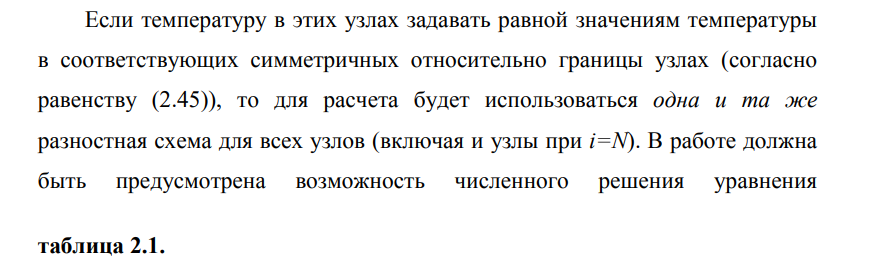


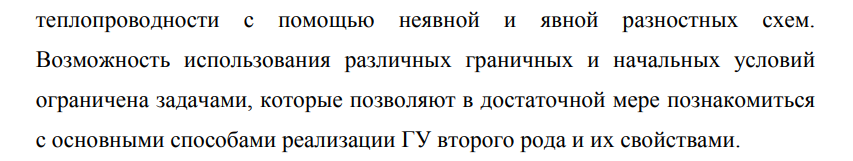


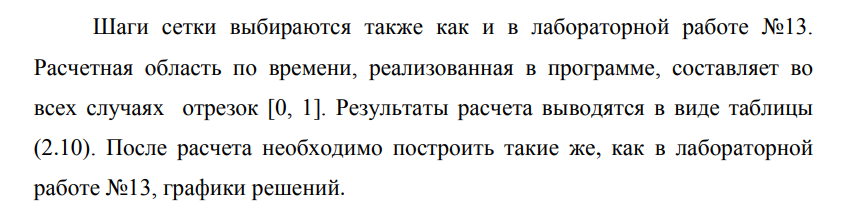












**Программная реализация**

import numpy as np

import math

from matplotlib import pyplot as plt

from scipy import integrate

from mpl\_toolkits.mplot3d import Axes3D

A = -1

B = 1

k = 0.5

phi = lambda x: abs(x)

g1 = lambda t: 1

g2 = lambda t: 1

f = lambda x, t: 0

T = 0.4

def explicit\_method\_1(h, tau, build\_plot=False):

M = int(T / tau) + 1

N = int((B - A) / h) + 1

x\_values = np.linspace(A, B, N)

t\_values = np.linspace(0, tau, M)

matrix = np.zeros((M, N))

matrix[0] = [phi(x) for x in x\_values]

matrix[:, 0] = [g1(t) for t in t\_values]

for i in range(M - 1):

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

matrix[i + 1][j] = k \* tau / h \*\* 2 \* matrix[i][j - 1]

matrix[i + 1][j] += (1 - 2 \* k \* tau / h \*\* 2) \* matrix[i][j]

matrix[i + 1][j] += k \* tau / h \*\* 2 \* matrix[i][j + 1]

matrix[i + 1][j] += tau \* f(x\_values[j], t\_values[i])

matrix[i + 1][-1] = matrix[i + 1][-2] + h \* g2(t\_values[i])

if build\_plot:

fig = plt.figure()

ax = fig.add\_subplot(111, projection='3d')

for i in range(len(t\_values) - 1, -2, -1000):

x = x\_values

y = [i \* tau] \* len(x)

z = matrix[i]

ax.plot(x, y, z)

plt.show()

return matrix

def explicit\_method\_2(h, tau, build\_plot=False):

M = int(T / tau) + 1

N = int((B - A) / h) + 1

x\_values = np.linspace(A, B, N)

t\_values = np.linspace(0, tau, M)

matrix = np.zeros((M, N))

matrix[0] = [phi(x) for x in x\_values]

matrix[:, 0] = [g1(t) for t in t\_values]

for i in range(M - 1):

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

matrix[i + 1][j] = k \* tau / h \*\* 2 \* matrix[i][j - 1]

matrix[i + 1][j] += (1 - 2 \* k \* tau / h \*\* 2) \* matrix[i][j]

matrix[i + 1][j] += k \* tau / h \*\* 2 \* matrix[i][j + 1]

matrix[i + 1][j] += tau \* f(x\_values[j], t\_values[i])

matrix[i + 1][-1] = k \* tau / h \*\* 2 \* matrix[i][-2]

matrix[i + 1][-1] += (1 - 2 \* k \* tau / h \*\* 2) \* matrix[i][-1]

matrix[i + 1][-1] += k \* tau / h \*\* 2 \* (2 \* h \* g2(t\_values[i]) + matrix[i + 1][-2])

matrix[i + 1][-1] += tau \* f(x\_values[j], t\_values[i])

if build\_plot:

fig = plt.figure()

ax = fig.add\_subplot(111, projection='3d')

for i in range(len(t\_values) - 1, -2, -1000):

x = x\_values

y = [i \* tau] \* len(x)

z = matrix[i]

ax.plot(x, y, z)

plt.show()

return matrix

def implicit\_method\_1(h, tau, build\_plot=False):

M = int(T / tau) + 1

N = int((B - A) / h) + 1

x\_values = np.linspace(A, B, N)

t\_values = np.linspace(0, tau, M)

result = np.zeros((M, N))

result[0] = [phi(x) for x in x\_values]

result[:, 0] = [g1(t) for t in t\_values]

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

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

matrix[j][j - 1] = - k \* tau / h \*\* 2

matrix[j][j] = 1 + 2 \* k \* tau / h \*\* 2

matrix[j][j + 1] = - k \* tau / h \*\* 2

matrix[0][0] = 1

matrix[-1][-1] = 1

matrix[-1][-2] = - 1

for i in range(1, M):

b = np.array([tau \* f(x, t\_values[i] + tau) for x in x\_values]) + result[i - 1]

b[0] = g1(t\_values[i])

b[-1] = h \* g2(t\_values[i])

result[i] = np.linalg.solve(matrix, b)

if build\_plot:

fig = plt.figure()

ax = fig.add\_subplot(111, projection='3d')

for i in range(len(t\_values) - 1, -2, -1000):

x = x\_values

y = [i \* tau] \* len(x)

z = result[i]

ax.plot(x, y, z)

plt.show()

return result

def implicit\_method\_2(h, tau, build\_plot=False):

M = int(T / tau) + 1

N = int((B - A) / h) + 1

x\_values = np.linspace(A, B, N)

t\_values = np.linspace(0, tau, M)

result = np.zeros((M, N))

result[0] = [phi(x) for x in x\_values]

result[:, 0] = [g1(t) for t in t\_values]

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

for j in range(1, N):

matrix[j][j - 1] = - k \* tau / h \*\* 2

matrix[j][j] = 1 + 2 \* k \* tau / h \*\* 2

matrix[j][j + 1] = - k \* tau / h \*\* 2

matrix[0][0] = 1

matrix[-1][-1] = 1

matrix[-1][-3] = -1

for i in range(1, M):

b = np.zeros(N + 1)

b[1:-1] = np.array([tau \* f(x, t\_values[i] + tau) for x in x\_values[1:]]) + result[i - 1][1:]

b[0] = g1(t\_values[i])

b[-1] = 2 \* h \* g2(t\_values[i])

result[i] = np.linalg.solve(matrix, b)[:-1]

if build\_plot:

fig = plt.figure()

ax = fig.add\_subplot(111, projection='3d')

for i in range(len(t\_values) - 1, -2, -1000):

x = x\_values

y = [i \* tau] \* len(x)

z = result[i]

ax.plot(x, y, z)

plt.show()

return result

def task():

h = (B - A) / 500

tau = 0.5 \* (h \*\* 2) / k

# explicit\_method\_1(h, tau, build\_plot=True)

# explicit\_method\_2(h, tau, build\_plot=True)

# implicit\_method\_1(h, tau, build\_plot=True)

# implicit\_method\_2(h, tau, build\_plot=True)

from approximation import task

from statistics import stats

if \_\_name\_\_ == '\_\_main\_\_':

task()

stats()

from prettytable import PrettyTable

import matplotlib.pyplot as plt

import numpy as np

from approximation import explicit\_method\_1, explicit\_method\_2, implicit\_method\_1, implicit\_method\_2, A, B

variants\_1 = [

# N, tau, t1, t2

[10, 0.001, 0.25, 0.3],

[10, 0.0005, 0.25, 0.3],

[10, 0.00025, 0.25, 0.3],

[10, 0.000125, 0.25, 0.3],

[10, 0.0000625, 0.25, 0.3]

]

variants\_2 = [

# N, t1, t2

[10, 0.25, 0.3],

[20, 0.25, 0.3],

[30, 0.25, 0.3],

[40, 0.25, 0.3],

[50, 0.25, 0.3]

]

methods = [

[explicit\_method\_1, 'Explicit method #1'],

[explicit\_method\_2, 'Explicit method #2'],

[implicit\_method\_1, 'Implicit method #1'],

[implicit\_method\_2, 'Implicit method #2']

]

table\_1 = PrettyTable()

table\_2 = PrettyTable()

table\_3 = PrettyTable()

table\_4 = PrettyTable()

table\_1.field\_names = ['N', 'tau', 's(t=t1)', 's(t=t2)', 'max(t=t1)', 'max(t=t2)']

table\_2.field\_names = ['N', 'tau', 's(t=t1)', 's(t=t2)', 'max(t=t1)', 'max(t=t2)']

table\_3.field\_names = ['N', 'tau', 's(t=t1)', 's(t=t2)', 'max(t=t1)', 'max(t=t2)']

table\_4.field\_names = ['N', 'tau', 's(t=t1)', 's(t=t2)', 'max(t=t1)', 'max(t=t2)']

errors\_t1 = []

errors\_t2 = []

def stats():

for method, name in methods:

print(name)

print('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_')

for N, tau, t1, t2 in variants\_1:

h = (B - A) / N

solution\_1 = method(h, tau)

solution\_2 = method(h, tau / 2)

err\_t1 = (solution\_1[int(t1 / tau)] - solution\_2[2 \* int(t1 / tau)])

err\_t2 = (solution\_1[int(t2 / tau)] - solution\_2[2 \* int(t2 / tau)])

errors\_t1.append((tau, np.linalg.norm(err\_t1)))

errors\_t2.append((tau, np.linalg.norm(err\_t2)))

table\_1.add\_row([N, tau, err\_t1.std(), err\_t2.std(), max(abs(err\_t1)), max(abs(err\_t2))])

print('h is fixed')

print(table\_1)

table\_1.clear\_rows()

plt.xscale('log')

plt.xlabel('tau', fontsize=18)

plt.ylabel('error', fontsize=18)

plt.plot([tau for tau, err in errors\_t1], [tau for tau, err in errors\_t1], label='t = t1')

plt.legend()

plt.show()

errors\_t1.clear()

plt.xscale('log')

plt.xlabel('tau', fontsize=18)

plt.ylabel('error', fontsize=18)

plt.plot([tau for tau, err in errors\_t2], [tau for tau, err in errors\_t2], label='t = t2')

plt.legend()

plt.show()

errors\_t2.clear()

for N, t1, t2 in variants\_2:

h = (B - A) / N

tau = 0.0001

solution\_1 = method(h, tau)

solution\_2 = method(h / 2, tau)

err\_t1 = (solution\_1[int(t1 / tau)] - solution\_2[int(t1 / tau)][::2])

err\_t2 = (solution\_1[int(t2 / tau)] - solution\_2[int(t2 / tau)][::2])

errors\_t1.append((h, np.linalg.norm(err\_t1)))

errors\_t2.append((h, np.linalg.norm(err\_t2)))

table\_2.add\_row([N, tau, err\_t1.std(), err\_t2.std(), max(abs(err\_t1)), max(abs(err\_t2))])

print('tau is fixed')

print(table\_2)

table\_2.clear\_rows()

plt.xscale('log')

plt.xlabel('h', fontsize=18)

plt.ylabel('error', fontsize=18)

plt.plot([h for h, err in errors\_t1], [h for h, err in errors\_t1], label='t = t1')

plt.legend()

plt.show()

errors\_t1.clear()

plt.xscale('log')

plt.xlabel('h', fontsize=18)

plt.ylabel('error', fontsize=18)

plt.plot([h for h, err in errors\_t2], [h for h, err in errors\_t2], label='t = t2')

plt.legend()

plt.show()

errors\_t2.clear()

print('tau is h ^ 2 / 6 (relative to tau)')

for N, t1, t2 in variants\_2:

h = (B - A) / N

tau = (h \*\* 2) / 6

solution\_1 = method(h, tau)

solution\_2 = method(h, tau / 2)

err\_t1 = (solution\_1[int(t1 / tau)] - solution\_2[2 \* int(t1 / tau)])

err\_t2 = (solution\_1[int(t2 / tau)] - solution\_2[2 \* int(t2 / tau)])

table\_3.add\_row([N, tau, err\_t1.std(), err\_t2.std(), max(abs(err\_t1)), max(abs(err\_t2))])

print(table\_3)

table\_3.clear\_rows()

print('tau is h ^ 2 / 6 (relative to h)')

for N, t1, t2 in variants\_2:

h = (B - A) / N

tau = (h \*\* 2) / 6

solution\_1 = method(h, tau)

solution\_2 = method(h / 2, tau)

err\_t1 = (solution\_1[int(t1 / tau)] - solution\_2[int(t1 / tau)][::2])

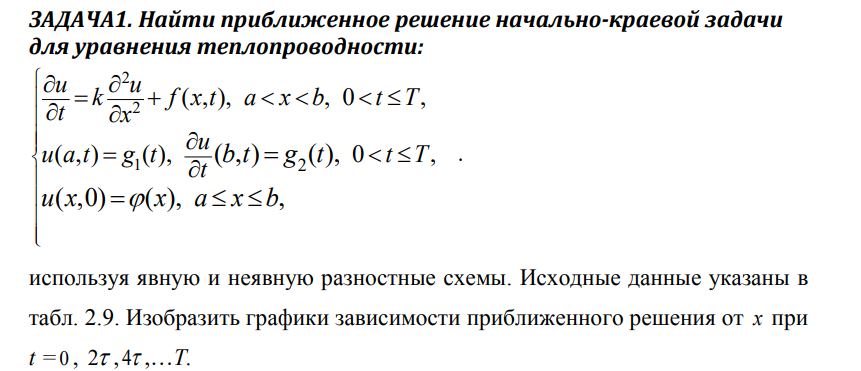
err\_t2 = (solution\_1[int(t2 / tau)] - solution\_2[int(t2 / tau)][::2])

table\_4.add\_row([N, tau, err\_t1.std(), err\_t2.std(), max(abs(err\_t1)), max(abs(err\_t2))])

print(table\_4)

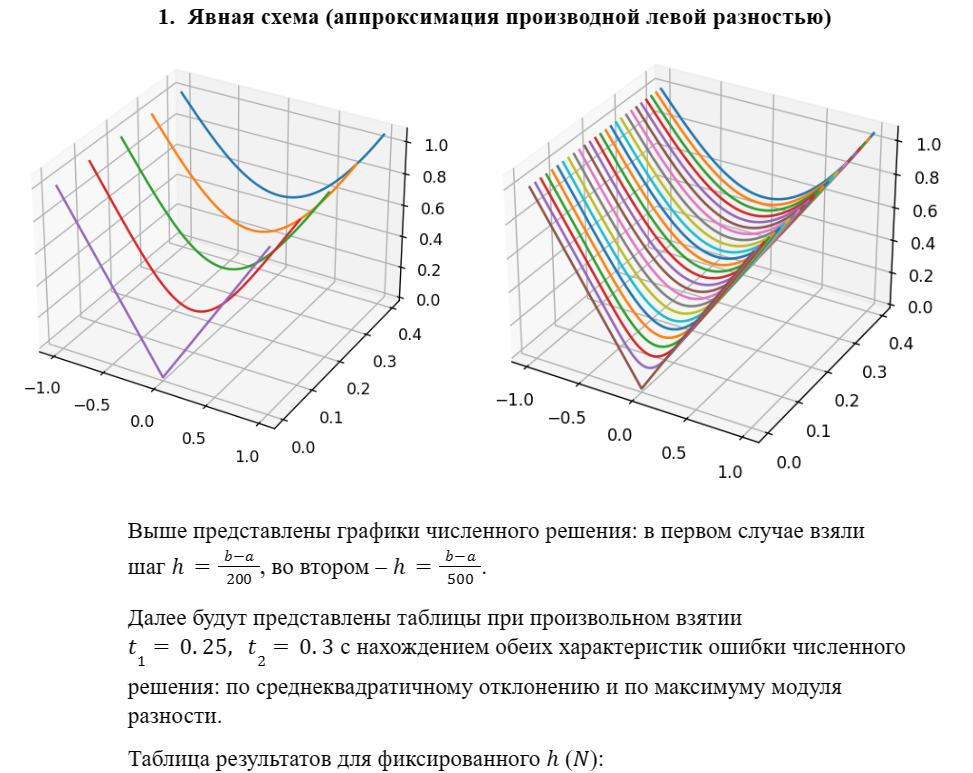
table\_4.clear\_rows()

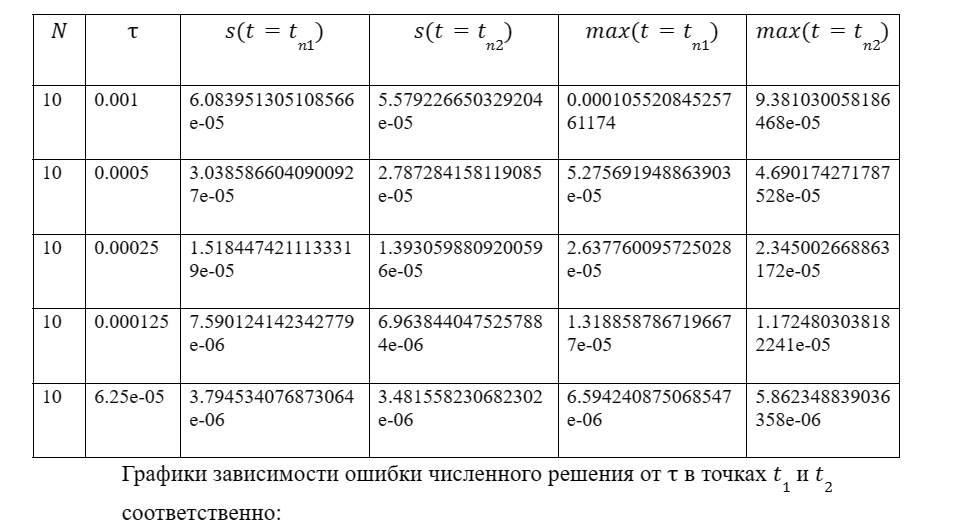
**Полученные результаты**

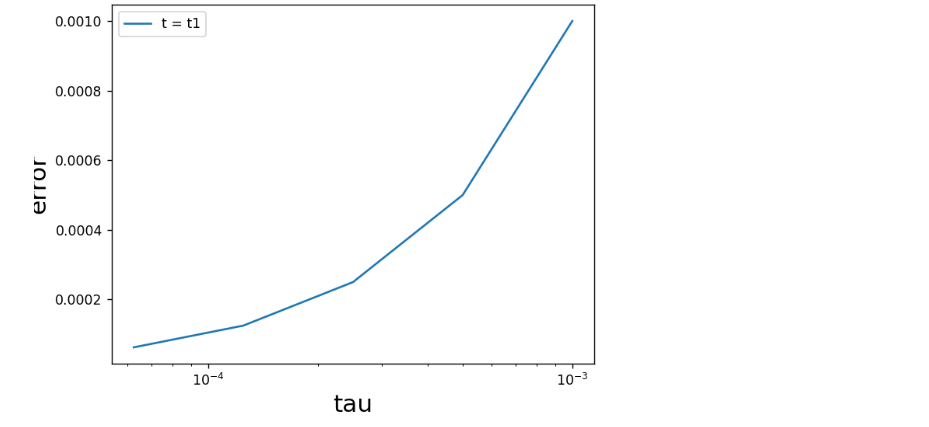


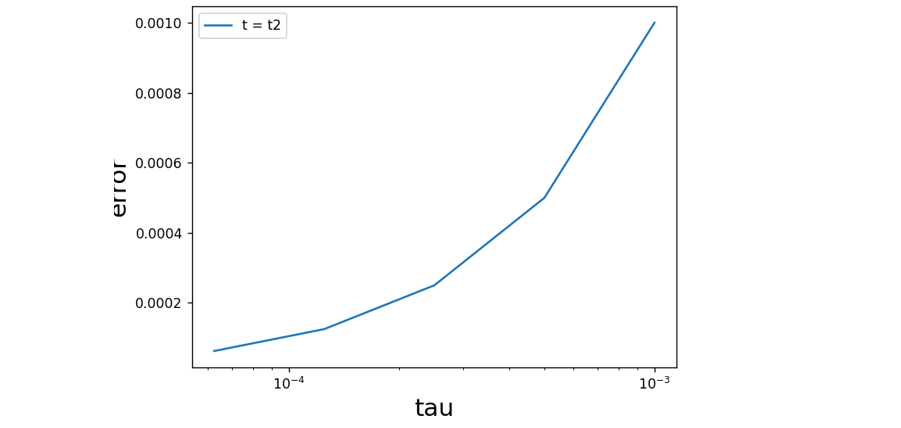


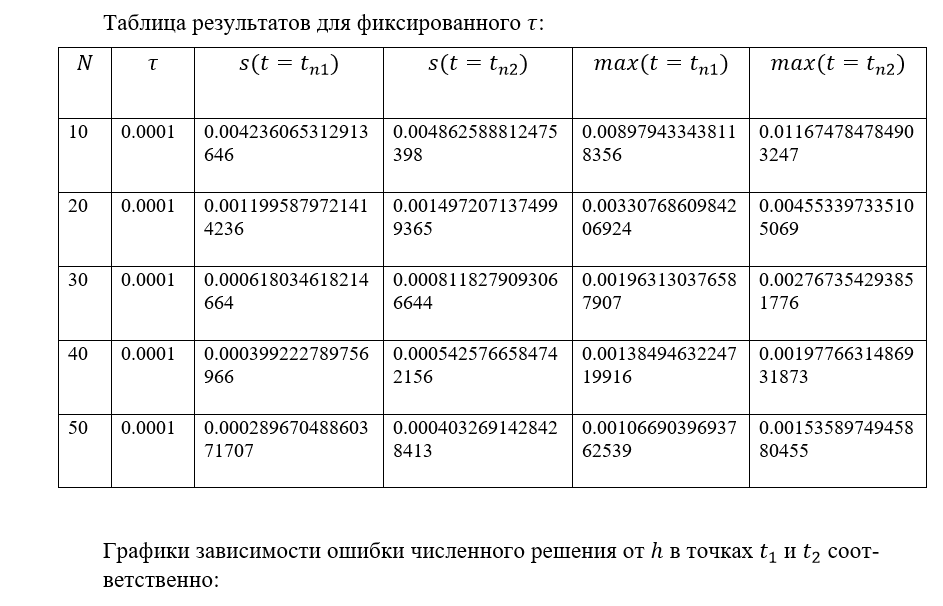


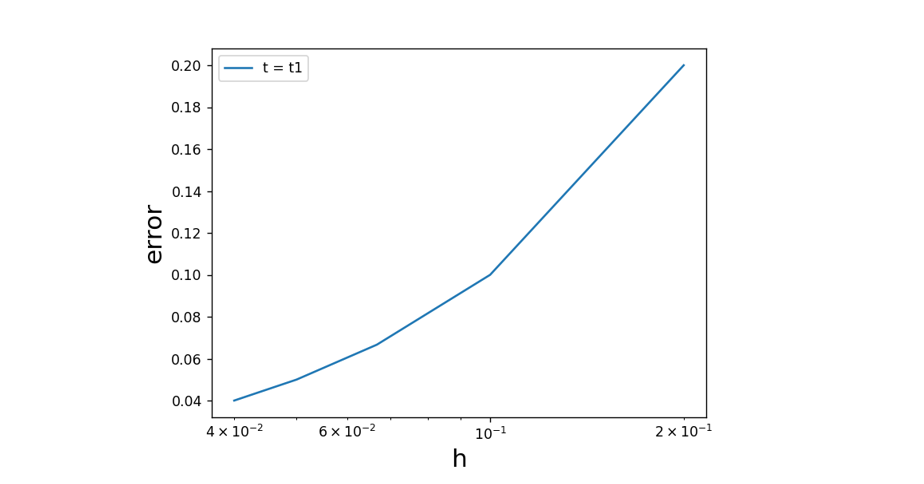


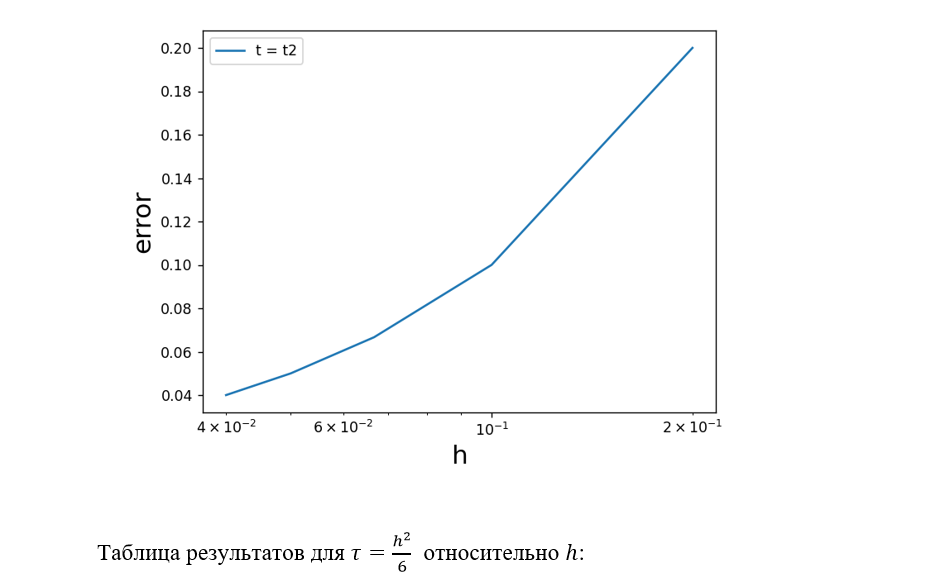


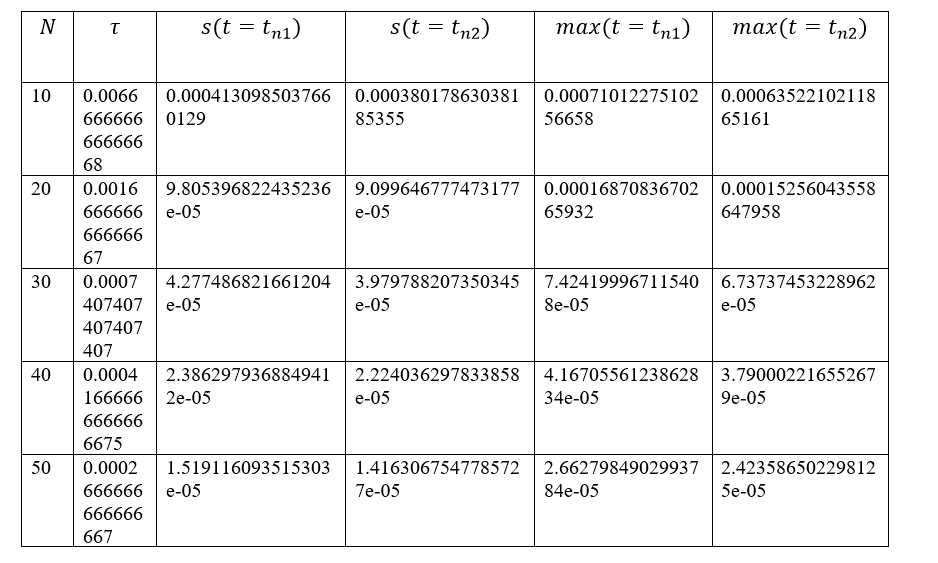




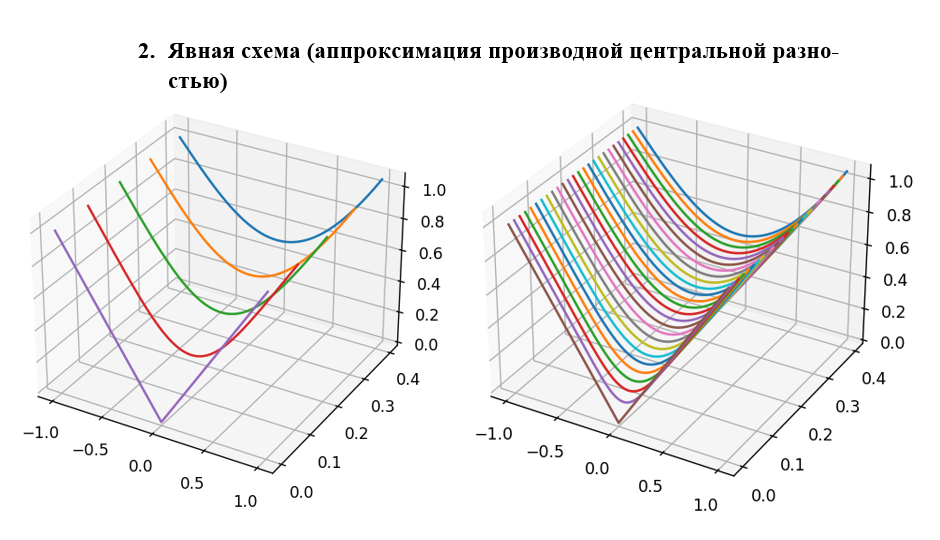


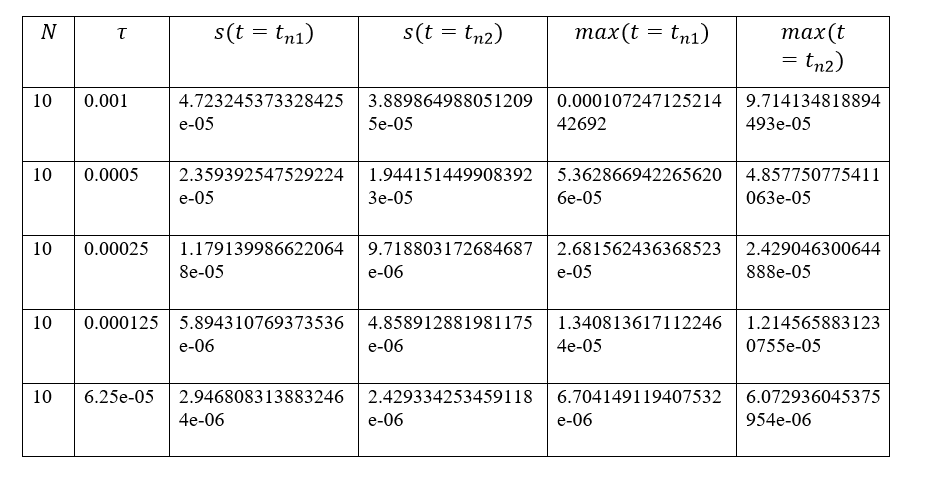


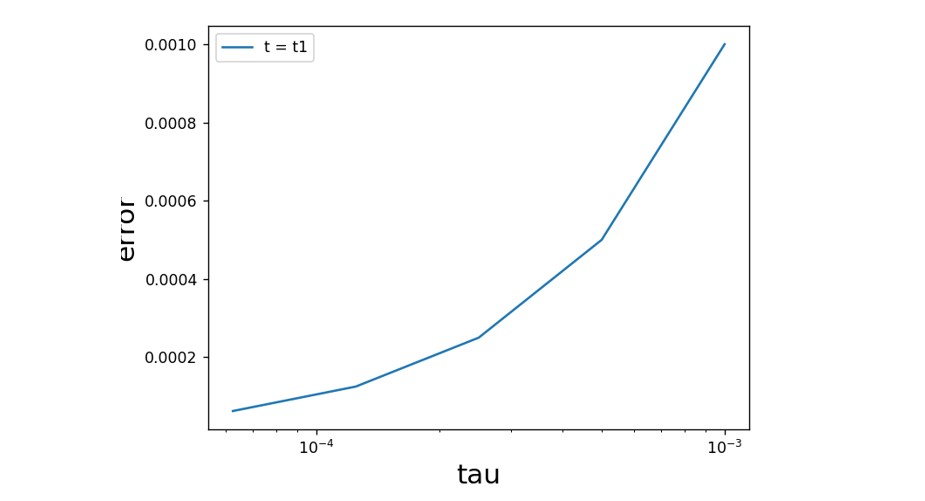


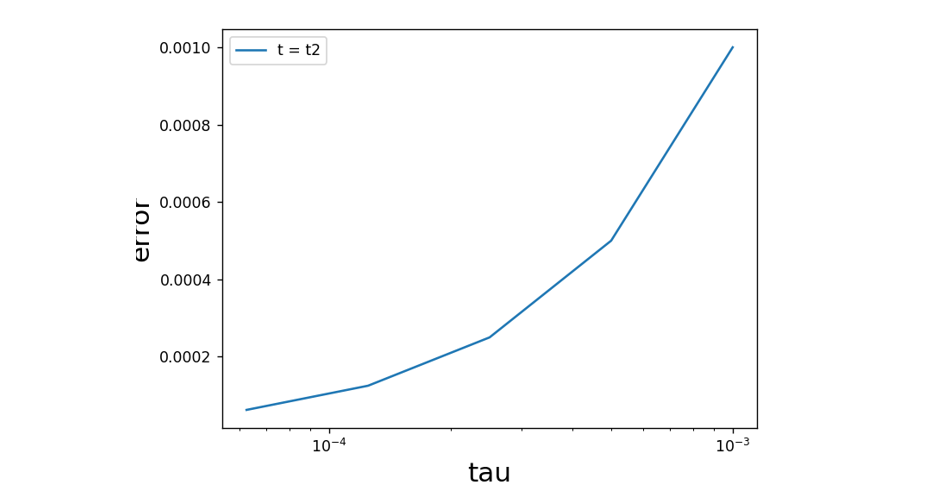


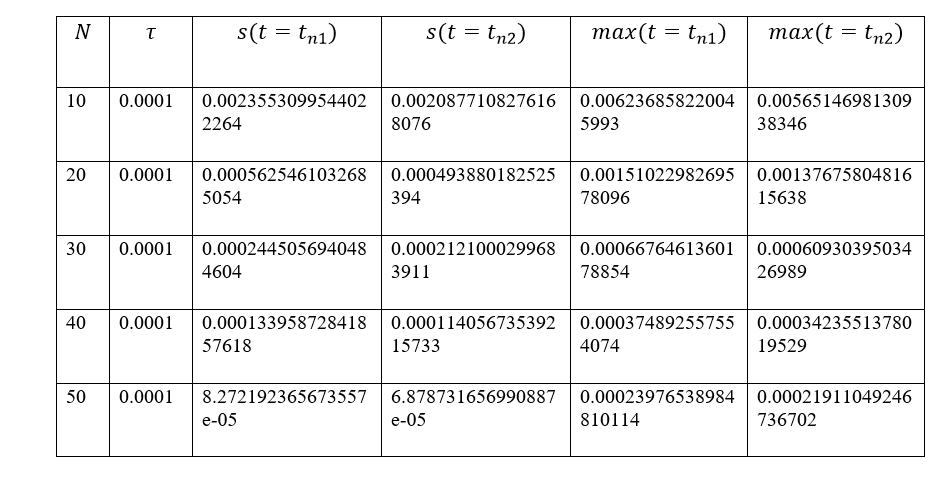


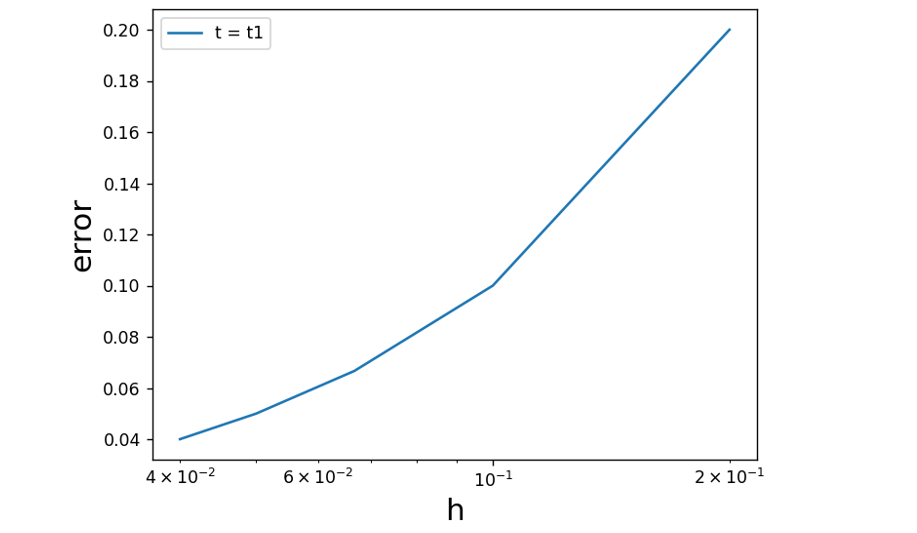


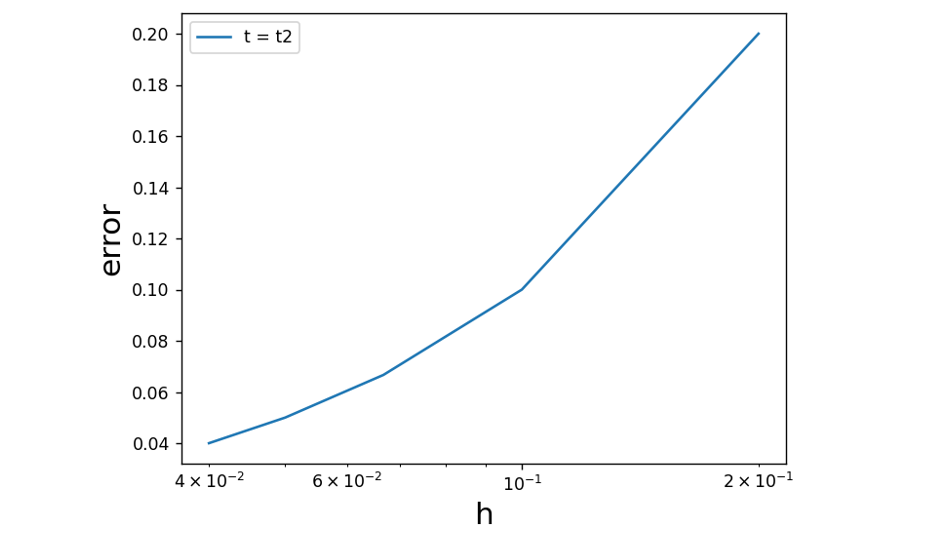


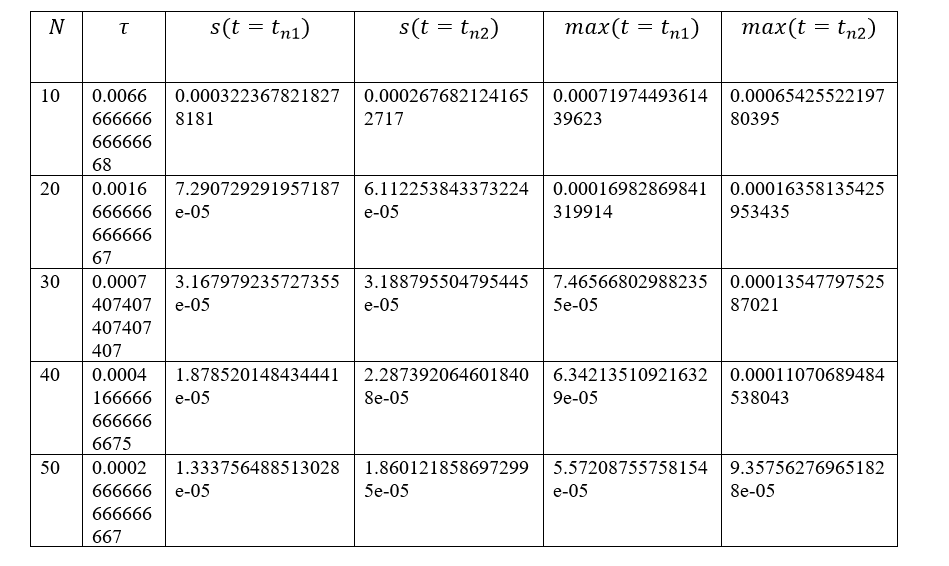


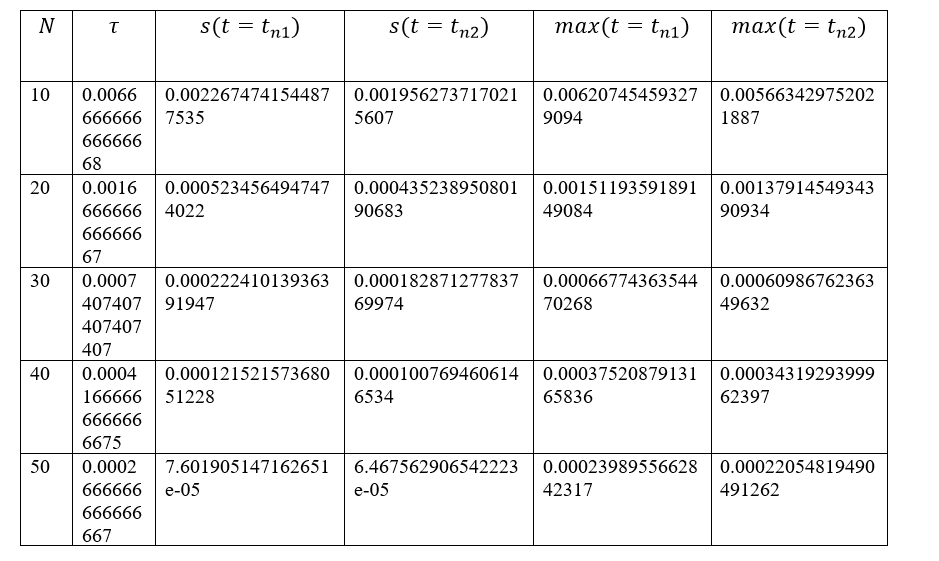


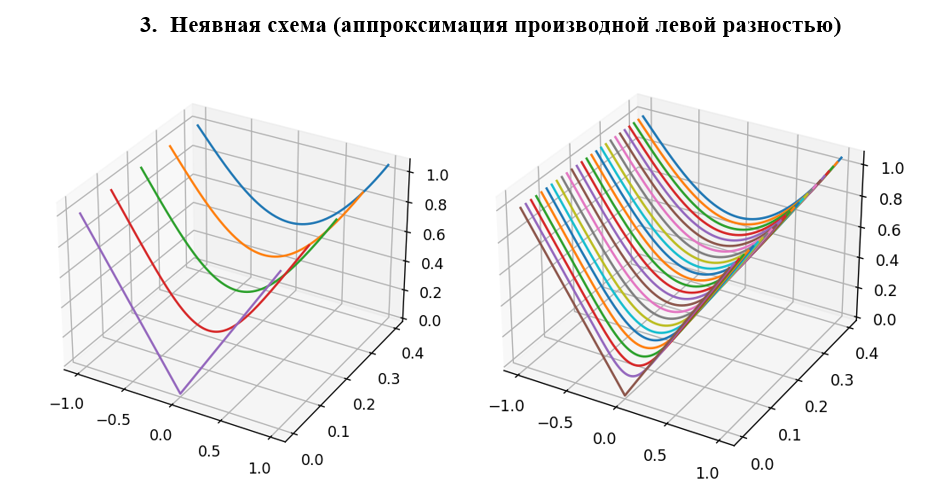


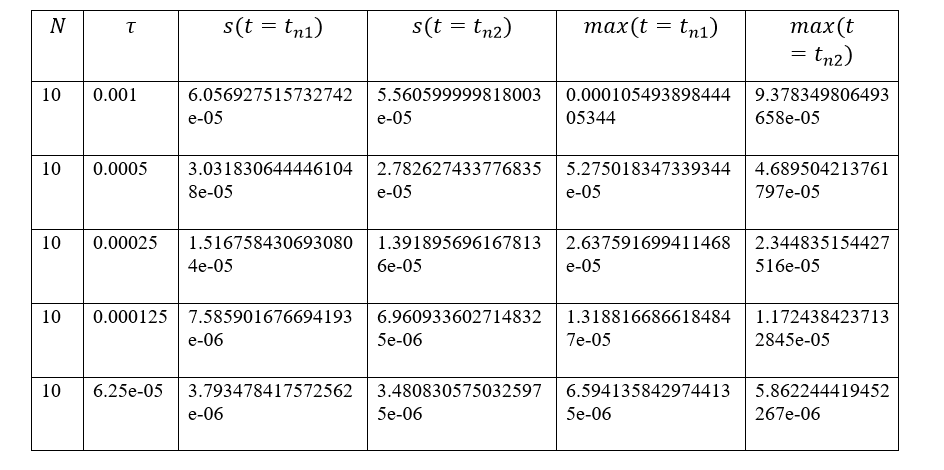


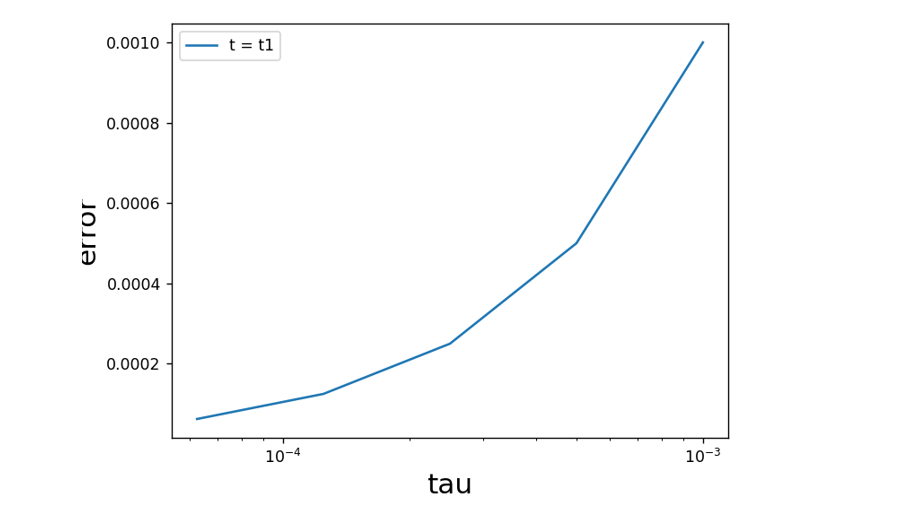


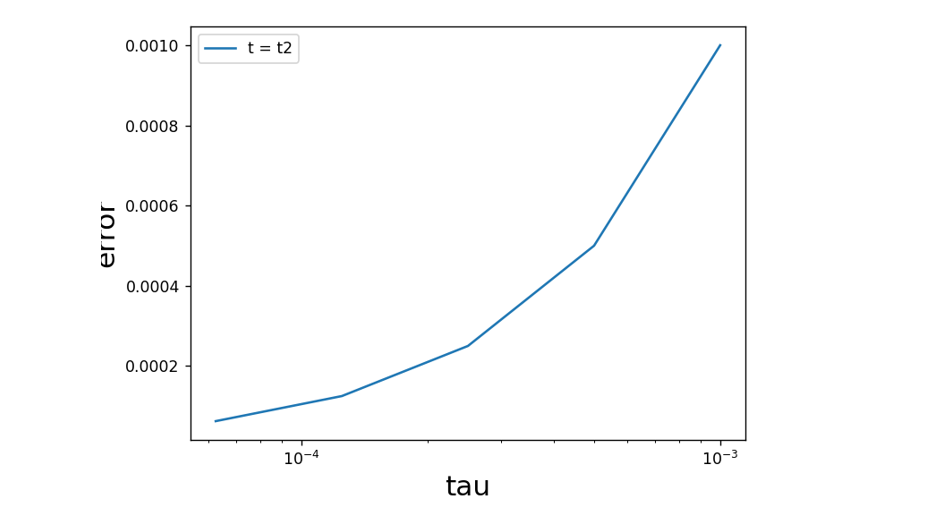


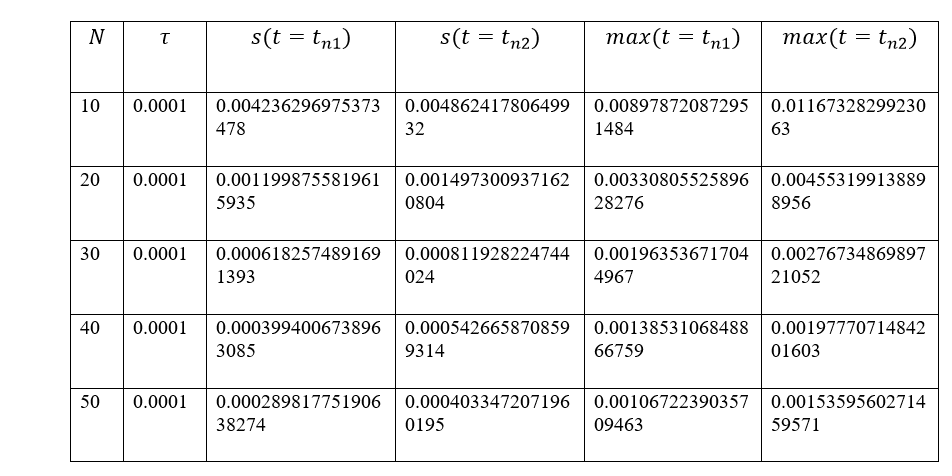


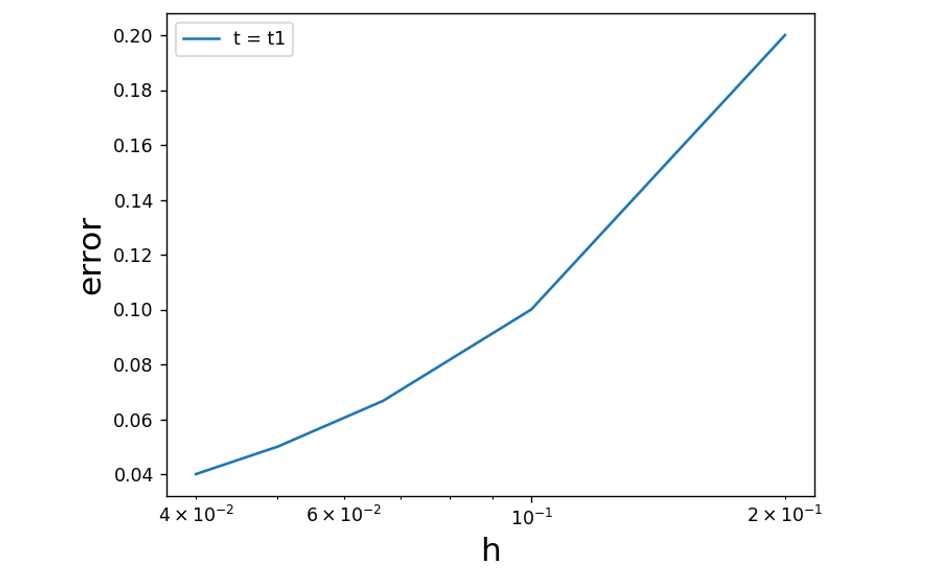


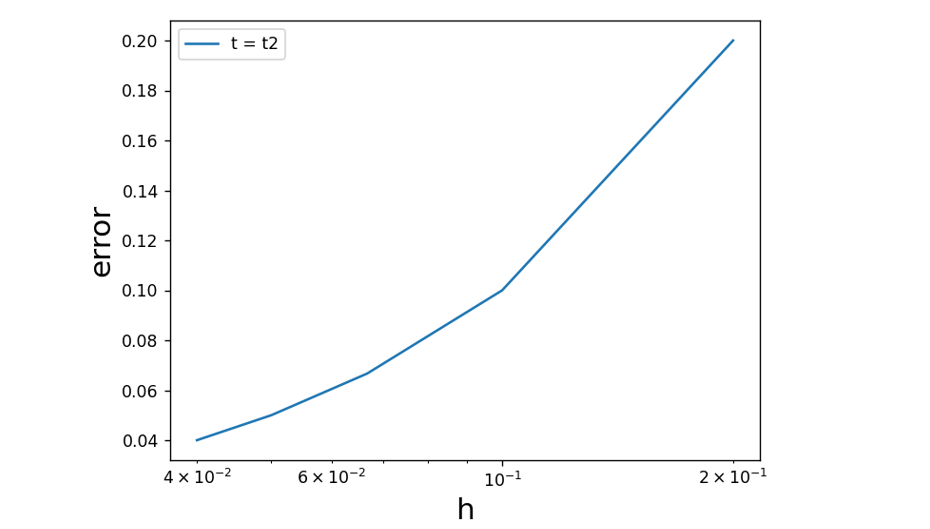


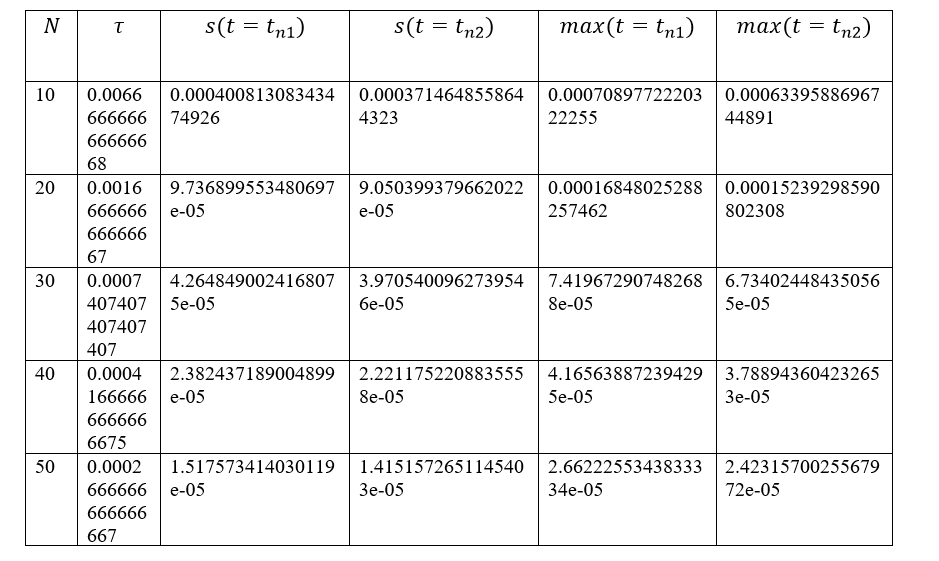


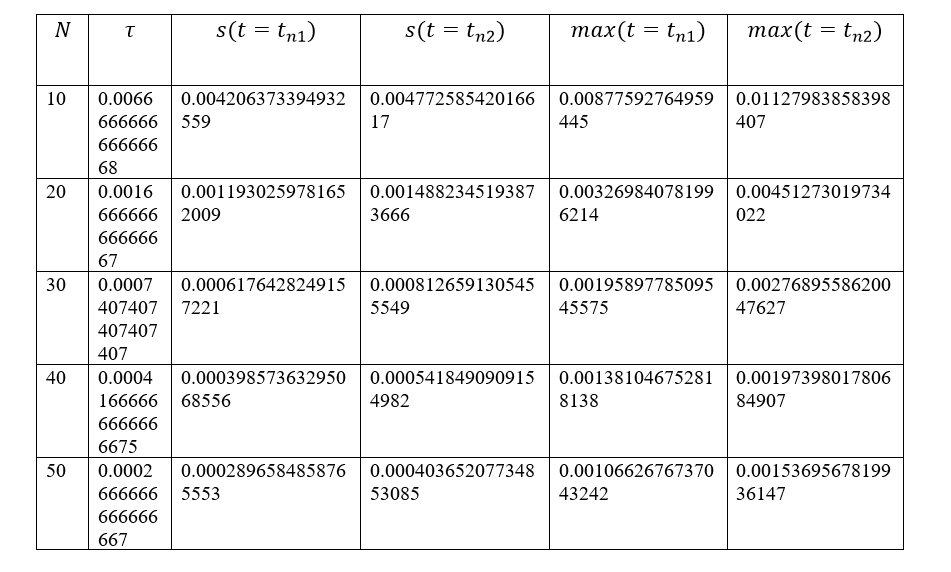


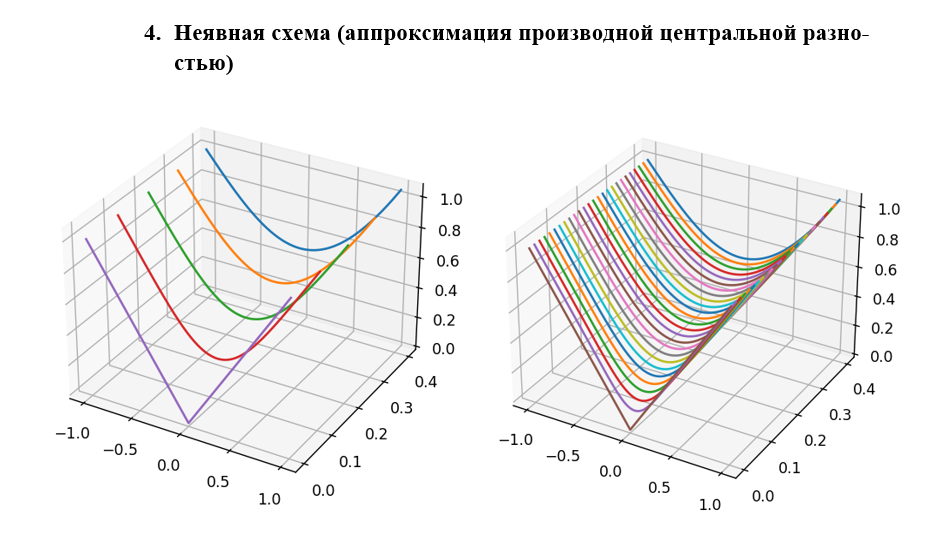


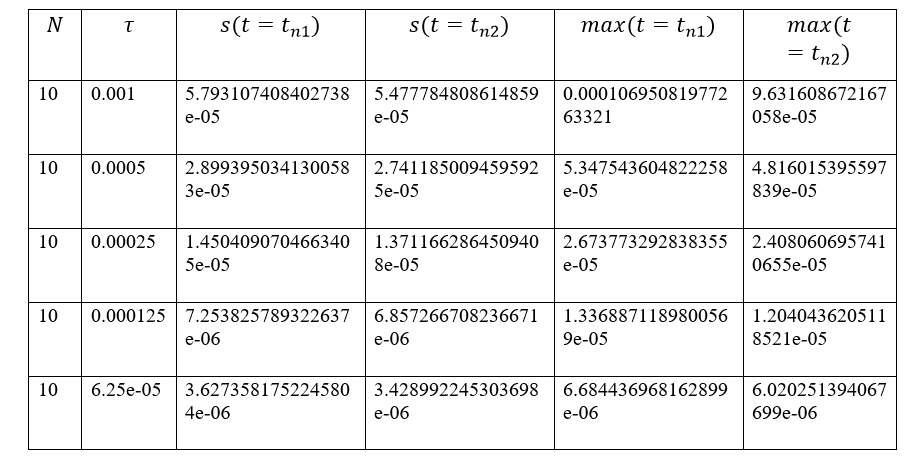


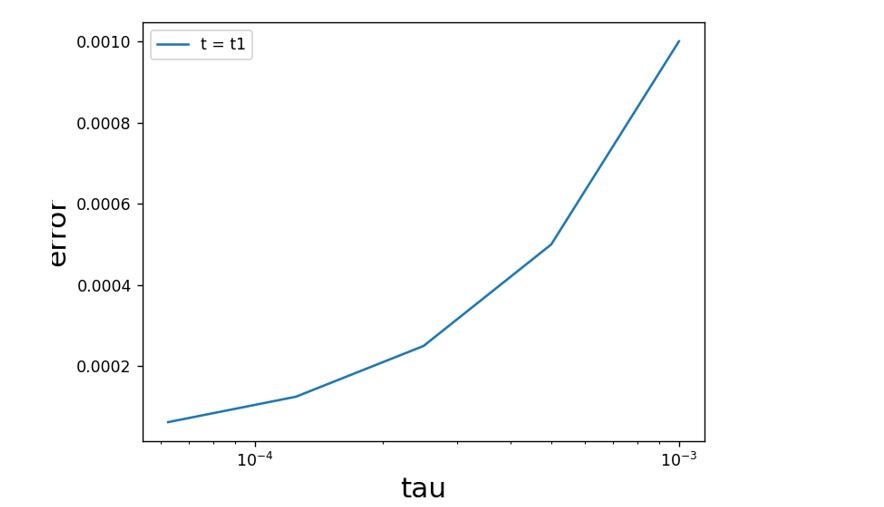


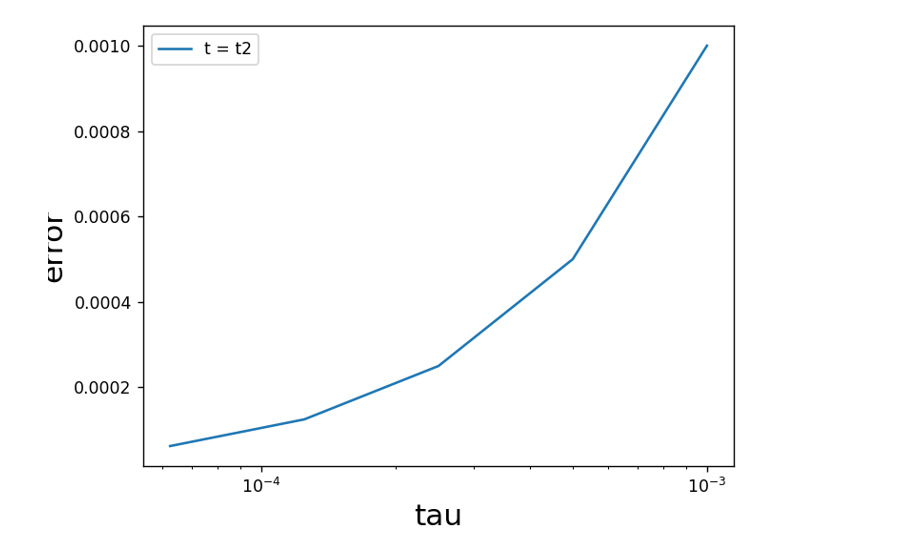


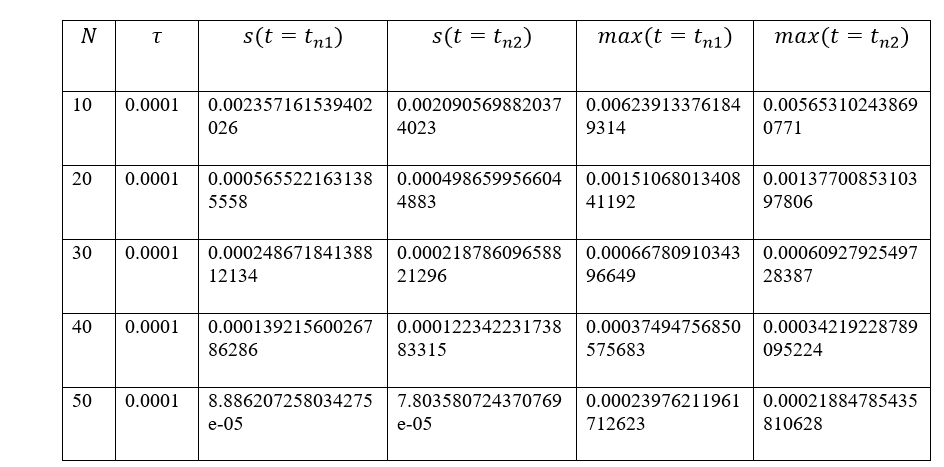


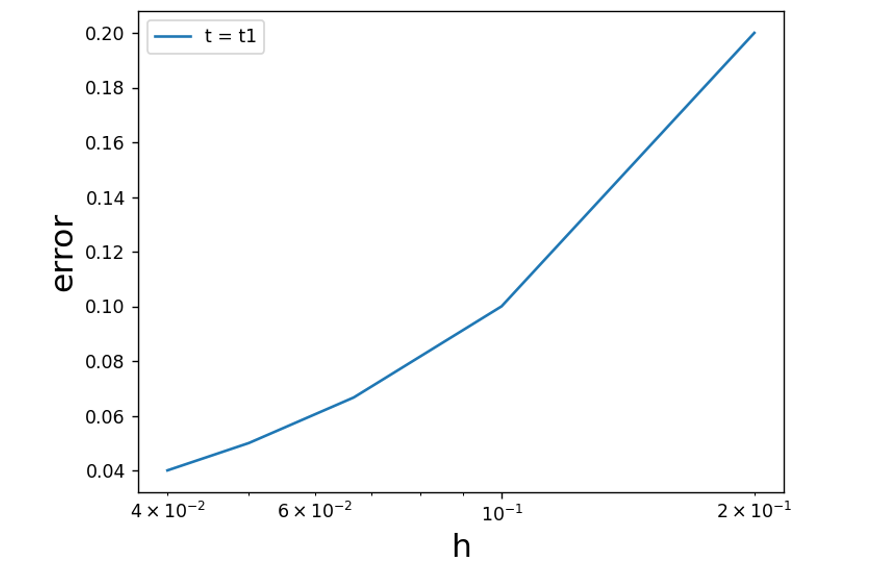


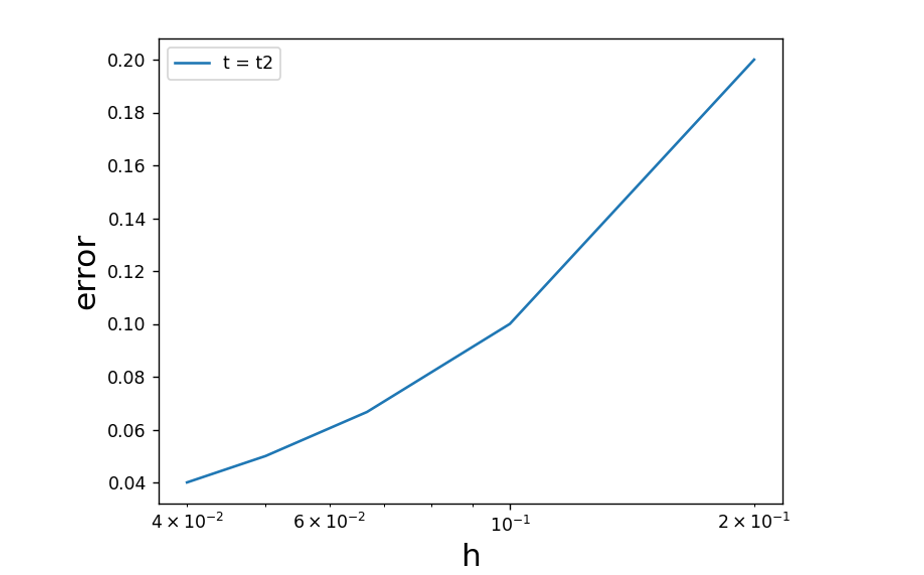


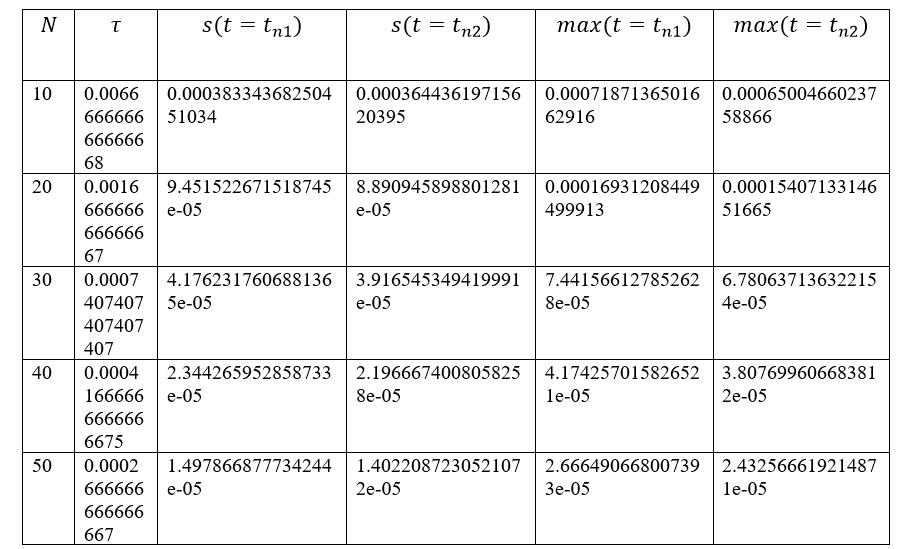


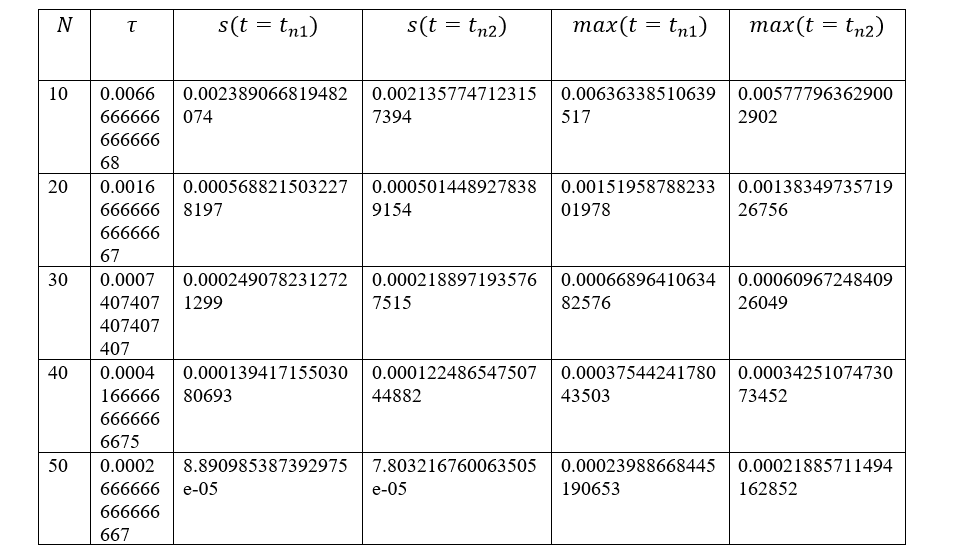












**Выводы**

