**Отчет по лабораторной работе №5 по курсу «Численные методы»**

Студент группы 8О-406: Полюбин А.И. Работа выполнена: 25.11.2022

Преподаватель: Пивоваров Д.Е. Отчет сдан: 4.01.2023

Итоговая оценка:

Подпись преподавателя:

Москва 2023

**1. Тема работы**

ЧИСЛЕННОЕ РЕШЕНИЕ УРАВНЕНИЙ ПАРАБОЛИЧЕСКОГО ТИПА. ПОНЯТИЕ О МЕТОДЕ КОНЕЧНЫХ РАЗНОСТЕЙ. ОСНОВНЫЕ ОПРЕДЕЛЕНИЯ И КОНЕЧНО-РАЗНОСТНЫЕ СХЕМЫ

**2. Цель работы**

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: .

**3. Ход выполнения работы**

В данной лабораторной реализованы явная, неявная схемы и схема Кранка-Николсона. Также реализована функция подсчета погрешности и функция отрисовки полученного решения. Функции для вычисления производных и метода прогонки были взяты из лабораторных работ прошлого семестра. Графики выводятся при помощи библиотеки matplotlib.

**4. Листинг**

import matplotlib.pyplot as plt

import sys

import warnings

import numpy as np

from functools import reduce

from mpl\_toolkits.mplot3d import Axes3D

def phi\_0(t, a = 1.0):

return np.exp(-a\*t)

def phi\_l(t, a = 1.0):

return -np.exp(-a\*t)

def u\_0(x):

return np.sin(x)

#Истинное решение

def u(x, t, a = 1.0):

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

class Schema:

def \_\_init\_\_(self, a = 1, f0 = phi\_0, fl = phi\_l, u0 = u\_0,

O = 0.5, l0 = 0, l1 = np.pi, T = 5, aprx\_cls = None):

self.fl = lambda t: fl(t, a)

self.f0 = lambda t: f0(t, a)

self.u0 = u0

self.T = T

self.l0 = l0

self.l1 = l1

self.tau = None

self.h = None

self.a = a

self.O = O

self.approx = None

if aprx\_cls is not None:

self.\_init\_approx(aprx\_cls)

self.sigma = None

def \_init\_approx(self, a\_cls):

self.approx = a\_cls(self.f0, self.fl)

def set\_approx(self, aprx\_cls):

self.\_init\_approx(self, aprx\_cls)

def set\_l0\_l1(self, l0, l1):

self.l0 = l0

self.l1 = l1

def set\_T(self, T):

self.T = T

def \_compute\_h(self, N):

self.h = (self.l1 - self.l0) / N

def \_compute\_tau(self, K):

self.tau = self.T / K

def \_compute\_sigma(self):

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

@staticmethod

def nparange(start, end, step = 1):

now = start

e = 0.00000000001

while now - e <= end:

yield now

now += step

def \_compute\_line(self, t, x, last\_line):

pass

def \_\_call\_\_(self, N=30, K=110):

N, K = N-1, K-1

self.\_compute\_tau(K)

self.\_compute\_h(N)

self.\_compute\_sigma()

ans = []

x = list(self.nparange(self.l0, self.l1, self.h))

last\_line = list(map(self.u0, x))

ans.append(list(last\_line))

X = []

Y = []

X.append(x)

Y.append([0.0 for \_ in x])

for t in self.nparange(self.tau, self.T, self.tau):

ans.append(self.\_compute\_line(t, x, last\_line))

X.append(x)

Y.append([t for \_ in x])

last\_line = ans[-1]

return X, Y, ans

#Явная схема

class Explict\_schema(Schema):

def \_compute\_sigma(self):

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

if self.sigma > 0.5:

warnings.warn("Sigma > 0.5")

def \_compute\_line(self, t, x, last\_line):

line = [None for \_ in last\_line]

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

line[i] = self.sigma\*last\_line[i-1]

line[i] += (1 - 2\*self.sigma)\*last\_line[i]

line[i] += self.sigma\*last\_line[i+1]

line[0] = self.approx.explict\_0(t, self.h, self.sigma,

last\_line, line, t - self.tau)

line[-1] = self.approx.explict\_l(t, self.h, self.sigma,

last\_line, line, t - self.tau)

return line

#Схема Кранка\_Николсона

class Explict\_Implict(Schema):

def set\_O(self, O):

self.O = O

@staticmethod

def three\_diagonal(A, b):

P = [-item[2] for item in A]

Q = [item for item in b]

P[0] /= A[0][1]

Q[0] /= A[0][1]

for i in range(1, len(b)):

z = (A[i][1] + A[i][0] \* P[i-1])

P[i] /= z

Q[i] -= A[i][0] \* Q[i-1]

Q[i] /= z

x = [item for item in Q]

for i in range(len(x) - 2, -1, -1):

x[i] += P[i] \* x[i + 1]

return x

def \_compute\_line(self, t, x, last\_line):

a = self.sigma \* self.O

b = -1 - 2 \* self.sigma \* self.O

A = [(a, b, a) for \_ in range(1, len(x)-1)]

w = [

-(last\_line[i] +

(1 - self.O) \* self.sigma\*

(last\_line[i-1] - 2\*last\_line[i] + last\_line[i+1]))

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

]

koeffs = self.approx.nikolson\_0(t, self.h, self.sigma,

last\_line, self.O, t - self.tau)

A.insert(0, koeffs[:-1])

w.insert(0, koeffs[-1])

koeffs = self.approx.nikolson\_l(t, self.h, self.sigma,

last\_line, self.O, t - self.tau)

A.append(koeffs[:-1])

w.append(koeffs[-1])

return self.three\_diagonal(A, w)

Krank\_Nikolson = Explict\_Implict

#Аппроксимация производных

class Derivative\_approx:

def \_\_init\_\_(self, f0, fl):

self.f0 = f0

self.fl = fl

def explict\_0(self, t, h, sigma, l0, l1, t0):

pass

def explict\_l(self, t, h, sigma, l0, l1, t0):

pass

def nikolson\_0(self, t, h, sigma, l0, O, t0):

pass

def nikolson\_l(self, t, h, sigma, l0, O, t0):

pass

#Двухточечная аппроксимация

class Approx\_two\_one(Derivative\_approx):

def explict\_0(self, t, h, sigma, l0, l1, t0):

return -h \* self.f0(t) + l1[1]

def explict\_l(self, t, h, sigma, l0, l1, t0):

return h \* self.fl(t) + l1[-2]

def nikolson\_0(self, t, h, sigma, l0, O, t0):

return 0, -1, 1, h\*self.f0(t)

def nikolson\_l(self, t, h, sigma, l0, O, t0):

return -1, 1, 0, h\*self.fl(t)

#Трехточечная аппрокисмация второго порядка

class Approx\_three\_two(Derivative\_approx):

def explict\_0(self, t, h, sigma, l0, l1, t0):

return (-2\*h\*self.f0(t) + 4\*l1[1] - l1[2]) / 3

def explict\_l(self, t, h, sigma, l0, l1, t0):

return (2\*h\*self.fl(t) + 4\*l1[-2] - l1[-3]) / 3

def nikolson\_0(self, t, h, sigma, l0, O, t0):

d = 2\*sigma\*O\*h\*self.f0(t)

d -= l0[1] + (1 - O)\*sigma\*(l0[0] - 2\*l0[1] + l0[2])

return 0, -2\*sigma\*O, 2\*sigma\*O - 1, d

def nikolson\_l(self, t, h, sigma, l0, O, t0):

d = 2\*sigma\*O\*h\*self.fl(t)

d += l0[-2] + (1 - O)\*sigma\*(l0[-3] - 2\*l0[-2] + l0[-1])

return 1 - 2\*sigma\*O, 2\*sigma\*O, 0, d

#Двухточечная аппрокисмация второго порядка

class Approx\_two\_two(Derivative\_approx):

def explict\_0(self, t, h, sigma, l0, l1, t0):

return -2\*sigma\*h\*self.f0(t0) + \

2\*sigma\*l0[1] + (1 - 2\*sigma)\*l0[0]

def explict\_l(self, t, h, sigma, l0, l1, t0):

return 2\*sigma\*h\*self.fl(t0) + \

2\*sigma\*l0[-2] + (1 - 2\*sigma)\*l0[-1]

def nikolson\_0(self, t, h, sigma, l0, O, t0):

d = 2\*sigma\*O\*h\*self.f0(t) - l0[0]

d -= 2\*(1 - O)\*sigma\*(l0[1] - l0[0] - h\*self.f0(t0))

return 0, -(2\*sigma\*O + 1), 2\*sigma\*O, d

def nikolson\_l(self, t, h, sigma, l0, O, t0):

d = -2\*sigma\*O\*h\*self.fl(t) - l0[-1]

d -= 2\*(1 - O)\*sigma\*(l0[-2] - l0[-1] + h\*self.fl(t0))

return 2\*sigma\*O, -(2\*sigma\*O + 1), 0, d

def real\_z(l0, l1, T, f):

x = np.arange(l0, l1 + 0.005, 0.005)

y = np.arange(0, T + 0.005, 0.005)

X = np.ones((y.shape[0], x.shape[0]))

Y = np.ones((x.shape[0], y.shape[0]))

Z = np.ones((y.shape[0], x.shape[0]))

for i in range(Y.shape[0]):

Y[i] = y

Y = Y.T

for i in range(X.shape[0]):

X[i] = x

for i in range(Z.shape[0]):

for j in range(Z.shape[1]):

Z[i, j] = f(X[i, j], Y[i, j])

return X, Y, Z

def Implicit\_plot(a = 1):

schema = Explict\_schema(T = 1, aprx\_cls=Approx\_two\_two, a=a)

x, y, z = schema(N = 8, K = 40)

fig = plt.figure(num=1, figsize=(19, 12), clear=True)

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

ax.plot\_wireframe(\*real\_z(0, np.pi, 1, lambda i, j: u(i, j, a)), color="blue")

ax.plot\_surface(np.array(x), np.array(y), np.array(z))

ax.set(xlabel='x', ylabel='t', zlabel='z', title='График решения и реальной функции явным методом')

fig.tight\_layout()

Implicit\_plot()

None

def Explicit\_plot(a = 1):

schema = Krank\_Nikolson(T = 1, aprx\_cls=Approx\_two\_two, a=a, O = 1)

x, y, z = schema(N = 8, K = 40)

fig = plt.figure(num=1, figsize=(19, 12), clear=True)

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

ax.plot\_wireframe(\*real\_z(0, np.pi, 1, lambda i, j: u(i, j, a)), color="blue")

ax.plot\_surface(np.array(x), np.array(y), np.array(z))

ax.set(xlabel='x', ylabel='t', zlabel='z', title='График решения и реальной функции неявным методом')

fig.tight\_layout()

Explicit\_plot()

None

def Krank\_Nikolson\_plot(a = 1):

schema = Krank\_Nikolson(T = 1, aprx\_cls=Approx\_two\_two, a=a, O = 0.5)

x, y, z = schema(N = 8, K = 40)

fig = plt.figure(num=1, figsize=(19, 12), clear=True)

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

ax.plot\_wireframe(\*real\_z(0, np.pi, 1, lambda i, j: u(i, j, a)), color="blue")

ax.plot\_surface(np.array(x), np.array(y), np.array(z))

ax.set(xlabel='x', ylabel='t', zlabel='z', title='График решения и реальной функции методом Кранка-Николсона')

fig.tight\_layout()

plt.show()

Krank\_Nikolson\_plot()

None

def epsilon(x, y, z, f):

ans = 0.0

for i in range(len(z)):

for j in range(len(z[i])):

ans += (z[i][j] - f(x[i][j], y[i][j]))\*\*2

return ans\*\*0.5

def graphic\_h(solver, real\_f):

h = []

e = []

for N in range(3, 50):

x, y, z = solver(N)

h.append(solver.h)

e.append(epsilon(x, y, z, real\_f))

return h, e

explict = Explict\_schema(T = 1, aprx\_cls=Approx\_two\_two)

plt.figure(figsize = (16, 10))

plt.subplot(2, 1, 1)

plt.title("Зависимость погрешности от длины шага")

h, e = graphic\_h(explict, u)

plt.plot(h, e, label="Явная", color = "blue")

plt.xlabel("h")

plt.ylabel("e")

plt.ylim([0, 2.1])

plt.xticks(list(explict.nparange(0, 1.6, 0.1)))

plt.yticks(list(explict.nparange(0, 2.1, 0.2)))

plt.legend()

plt.grid()

plt.subplot(2, 1, 2)

#График в логарифмической шкале

plt.plot(list(map(np.log, h)), list(map(np.log, e)), label="Явная")

plt.plot([-2, 0.5], [-3, -0.5], label="Зависимость $O(h)$")

plt.plot([-2, 0.5], [-3, 2], label="Зависимость $O(h^2)$")

plt.xlabel("log h")

plt.ylabel("log e")

plt.ylim([-3, 1])

plt.xlim([-2.1, 0.5])

plt.xticks(list(explict.nparange(-2, 0.5, 0.1)))

plt.yticks(list(explict.nparange(-3, 1, 0.2)))

plt.legend()

plt.grid()

implict = Krank\_Nikolson(T = 1, aprx\_cls=Approx\_two\_two, O=1)

plt.figure(figsize = (16, 10))

plt.subplot(2, 1, 1)

plt.title("Зависимость погрешности от длины шага")

h, e = graphic\_h(implict, u)

plt.plot(h, e, label="Неявная", color = "blue")

plt.xlabel("h")

plt.ylabel("e")

plt.ylim([0, 2.1])

plt.xticks(list(implict.nparange(0, 1.6, 0.1)))

plt.yticks(list(implict.nparange(0, 2.1, 0.2)))

plt.legend()

plt.grid()

plt.subplot(2, 1, 2)

plt.plot(list(map(np.log, h)), list(map(np.log, e)), label="Неявная")

plt.plot([-2, 0.5], [-3, -0.5], label="Зависимость $O(h)$")

plt.plot([-2, 0.5], [-3, 2], label="Зависимость $O(h^2)$")

plt.xlabel("log h")

plt.ylabel("log e")

plt.ylim([-3, 1])

plt.xlim([-2, 0.5])

plt.xticks(list(implict.nparange(-2, 0.5, 0.1)))

plt.yticks(list(implict.nparange(-3, 1, 0.2)))

plt.legend()

plt.grid()

krank = Krank\_Nikolson(T = 1, aprx\_cls=Approx\_two\_two)

plt.figure(figsize = (16, 10))

plt.subplot(2, 1, 1)

plt.title("Зависимость погрешности от длины шага")

h, e = graphic\_h(krank, u)

plt.plot(h, e, label="Кранка-Николсона", color = "blue")

plt.xlabel("h")

plt.ylabel("e")

plt.ylim([0, 2.1])

plt.xticks(list(krank.nparange(0, 1.6, 0.1)))

plt.yticks(list(krank.nparange(0, 2.1, 0.2)))

plt.legend()

plt.grid()

plt.subplot(2, 1, 2)

plt.plot(list(map(np.log, h)), list(map(np.log, e)), label="Кранк-Николсон")

plt.plot([-2, 0.5], [-3, -0.5], label="Зависимость $O(h)$")

plt.plot([-2, 0.5], [-3, 2], label="Зависимость $O(h^2)$")

plt.xlabel("log h")

plt.ylabel("log e")

plt.ylim([-3, 1])

plt.xlim([-2, 0.5])

plt.xticks(list(krank.nparange(-2, 0.5, 0.1)))

plt.yticks(list(krank.nparange(-3, 1, 0.2)))

plt.legend()

plt.grid()

def get\_graphic\_tau(solver, real\_f):

tau = []

e = []

for K in range(3, 90):

x, y, z = solver(K = K)

tau.append(solver.tau)

e.append(epsilon(x, y, z, real\_f))

return tau, e

explict = Explict\_schema(T = 5, aprx\_cls=Approx\_two\_two)

plt.figure(figsize = (16, 10))

plt.subplot(2, 1, 1)

plt.title("Зависимость погрешности от мелкости разбиения по времени")

tau, e = get\_graphic\_tau(explict, u)

plt.plot(tau, e, label="Явная", color = "blue")

plt.xlabel("t")

plt.ylabel("e")

plt.xticks(list(explict.nparange(0, 2.5, 0.1)))

plt.legend()

plt.grid()

plt.subplot(2, 1, 2)

plt.plot(list(map(np.log, tau)), list(map(np.log, e)), label="Явная")

plt.xlabel("log t")

plt.ylabel("log e")

plt.xticks(list(explict.nparange(-3, 1, 0.2)))

plt.legend()

plt.grid()

implict = Krank\_Nikolson(T = 5, aprx\_cls=Approx\_two\_two, O=1)

plt.figure(figsize = (16, 10))

plt.subplot(2, 1, 1)

plt.title("Зависимость погрешности от мелкости разбиения по времени")

tau, e = get\_graphic\_tau(implict, u)

plt.plot(tau, e, label="Неявный", color = "blue")

plt.xlabel("t")

plt.ylabel("e")

plt.xticks(list(explict.nparange(0, 2.5, 0.1)))

plt.legend()

plt.grid()

plt.subplot(2, 1, 2)

plt.plot(list(map(np.log, tau)), list(map(np.log, e)), label="Неявный")

plt.plot([-3, 1], [-0.5, 3.5], label="Зависимость $O(t)$")

plt.plot([-3, 1], [0, 2], label="Зависимость $O(\sqrt{t})$")

plt.xlabel("log t")

plt.ylabel("log e")

plt.xticks(list(explict.nparange(-3, 1, 0.2)))

plt.legend()

plt.grid()

krank = Krank\_Nikolson(T = 5, aprx\_cls=Approx\_two\_two)

plt.figure(figsize = (16, 10))

plt.subplot(2, 1, 1)

plt.title("Зависимость погрешности от мелкости разбиения по времени")

tau, e = get\_graphic\_tau(krank, u)

plt.plot(tau, e, label="Кранк-Николсон", color = "red")

plt.xlabel("t")

plt.ylabel("e")

plt.xticks(list(krank.nparange(0, 2.5, 0.1)))

plt.legend()

plt.grid()

plt.subplot(2, 1, 2)

plt.plot(list(map(np.log, tau)), list(map(np.log, e)), label="Кранк-Николсон")

plt.plot([-3, 1], [-5, 3], label="Зависимость $O(t^2)$")

plt.plot([-3, 1], [-3.5, -0.5], label="Зависимость $O(t)$")

plt.xlabel("log t")

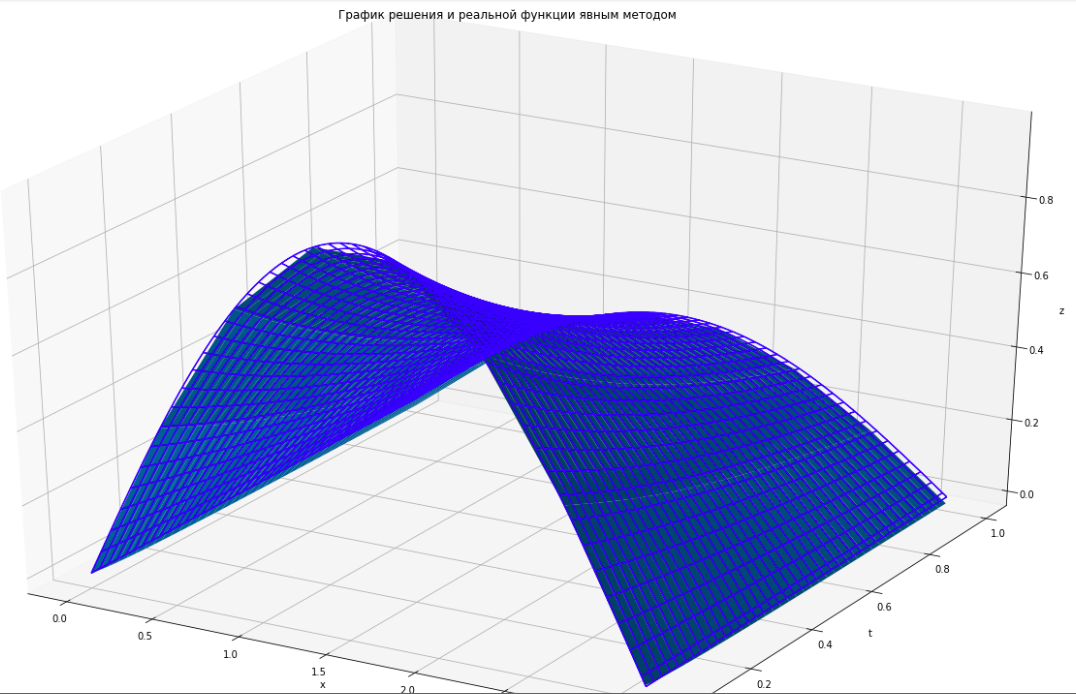
plt.ylabel("log e")

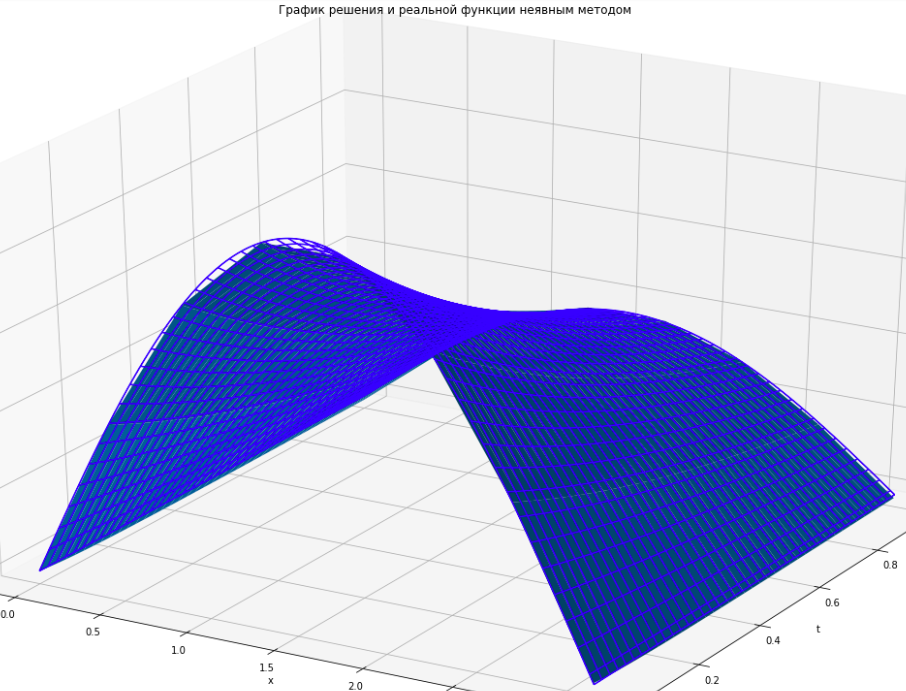
plt.xticks(list(krank.nparange(-3, 1, 0.2)))

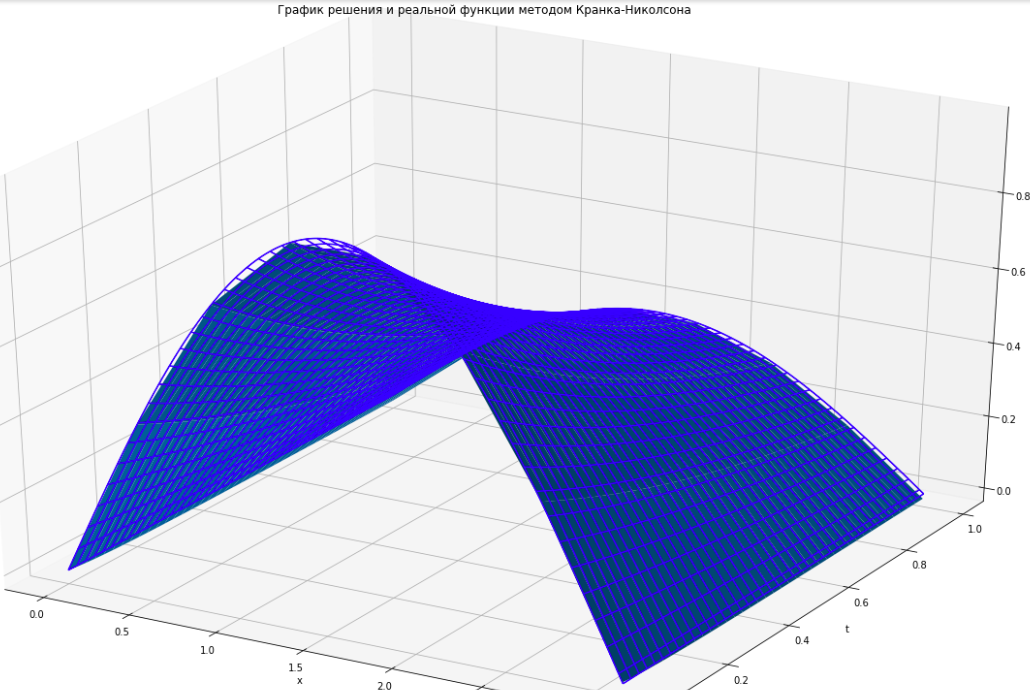
plt.legend()

plt.grid()

**5. Результаты**



****

****

**6. Выводы**

При выполнении данной лабораторной работы я научился находить численное решение уравнений параболического типа методом конечных разностей, а также аппроксимировать граничные значения разными способами.