**Отчет по лабораторной работе №6 по курсу «Численные методы»**

Студент группы 8О-406: Полюбин А.И. Работа выполнена: 7.12.2022

Преподаватель: Пивоваров Д.Е. Отчет сдан: 4.01.2023

Итоговая оценка:

Подпись преподавателя:

Москва 2023

**1. Тема работы**

МЕТОД КОНЕЧНЫХ РАЗНОСТЕЙ ДЛЯ РЕШЕНИЯ УРАВНЕНИЙ ГИПЕРБОЛИЧЕСКОГО ТИПА

**2. Цель работы**

Используя явную схему крест и неявную схему, решить начально-краевую задачу для дифференциального уравнения гиперболического типа. Аппроксимацию второго начального условия произвести с первым и со вторым порядком. 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 и неявная конечно-разностная схемы , реализована двухточечная аппроксимация с первым порядком, трехточечная аппроксимация с вторым порядком, двухточечная аппроксимация с вторым порядком. Функции для вычисления производных и метода прогонки были взяты из лабораторных работ прошлого семестра. Графики выводятся при помощи библиотеки matplotlib.

**4. Листинг**

import random

import matplotlib.pyplot as plt

import sys

import warnings

import numpy as np

from functools import reduce

from mpl\_toolkits.mplot3d import Axes3D

def psi\_1(x):

return np.exp(2\*x)

def psi\_2(x):

return 0

def dpsi2\_dx2(x):

return 4\*psi\_1(x)

#Истинное решение

def u(x, t):

return np.exp(2\*x)\*np.cos(t)

class Schema:

def \_\_init\_\_(self, psi1 = psi\_1, psi2 = psi\_2, diffpsi2 = dpsi2\_dx2,

l0 = 0, l1 = 1, T = 5, order2nd = True, aprx\_cls = None):

self.psi1 = psi1

self.diffpsi = diffpsi2

self.psi2 = psi2

self.T = T

self.l0 = l0

self.l1 = l1

self.tau = None

self.h = None

self.approx = None

self.order = order2nd

if aprx\_cls is not None:

self.\_init\_approx(aprx\_cls)

self.sigma = None

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

self.approx = a\_cls()

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.tau\*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\_line1, last\_line2):

pass

def \_\_call\_\_(self, N=30, K=200):

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.psi1, x))

ans.append(list(last\_line))

if self.order:

last\_line = list(map(

lambda a: self.psi1(a) + self.tau\*self.psi2(a) + self.tau\*self.tau\*self.diffpsi(a)/2,

x

))

else:

last\_line = list(map(lambda a: self.psi1(a) + self.tau\*self.psi2(a), x))

ans.append(list(last\_line))

X = [x, x]

Y = [[0.0 for \_ in x]]

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

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

ans.append(self.\_compute\_line(t, x, ans[-1], ans[-2]))

X.append(x)

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

return X, Y, ans

#Явная конечно-разностная схема

class Explict\_Schema(Schema):

def \_compute\_sigma(self):

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

if self.sigma > 1:

warnings.warn("Sigma > 1")

def \_compute\_line(self, t, x, last\_line1, last\_line2):

line = [None for \_ in last\_line1]

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

line[i] = self.sigma\*(last\_line1[i-1] - 2\*last\_line1[i] + last\_line1[i+1])

line[i] -= 5\*self.tau\*self.tau\*last\_line1[i]

line[i] += 2\*last\_line1[i]

line[i] -= last\_line2[i]

line[0] = self.approx.explict\_0(self.h, self.sigma, line, last\_line1, last\_line2, self.tau)

line[-1] = self.approx.explict\_l(self.h, self.sigma, line, last\_line1, last\_line2, self.tau)

return line

#Неявная конечно-разностная схема

class Implict\_Schema(Schema):

@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\_line1, last\_line2):

a = 1

b = -(2 + 5\*self.h\*self.h + 1/self.sigma)

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

w = [

(last\_line2[i] - 2\*last\_line1[i]) / self.sigma

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

]

koeffs = self.approx.implict\_0(self.h, self.sigma, last\_line1, last\_line2)

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

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

koeffs = self.approx.implict\_l(self.h, self.sigma, last\_line1, last\_line2)

A.append(koeffs[:-1])

w.append(koeffs[-1])

return self.three\_diagonal(A, w)

#Аппроксимация производных

class Derivative\_approx:

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

pass

def explict\_0(self, h, sigma, line, last\_line1, last\_line2, tau):

pass

def explict\_l(self, h, sigma, line, last\_line1, last\_line2, tau):

pass

def implict\_0(self, h, sigma, l0, l1):

pass

def implict\_l(self, h, sigma, l0, l1):

pass

#Двухточечная аппроксимация

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

def explict\_0(self, h, sigma, line, last\_line1, last\_line2, tau):

return line[1] / (1 + 2\*h)

def explict\_l(self, h, sigma, line, last\_line1, last\_line2, tau):

return line[-2] / (1 - 2\*h)

def implict\_0(self, h, sigma, l0, l1):

return 0, (1 + 2\*h), -1, 0

def implict\_l(self, h, sigma, l0, l1):

return -1, (1 - 2\*h), 0, 0

#Трехточечная аппрокисмация второго порядка

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

def explict\_0(self, h, sigma, line, last\_line1, last\_line2, tau):

return (4\*line[1] - line[2]) / (3 + 4\*h)

def explict\_l(self, h, sigma, line, last\_line1, last\_line2, tau):

return (4\*line[-2] - line[-3]) / (3 - 4\*h)

def implict\_0(self, h, sigma, l0, l1):

return 0, -(2 + 4\*h), -(5\*h\*h + 1/sigma - 2), (-2\*l0[1] + l1[1])/sigma

def implict\_l(self, h, sigma, l0, l1):

return -(5\*h\*h + 1/sigma - 2), -(2 - 4\*h), 0, (-2\*l0[-2] + l1[-2])/sigma

#Двухточечная аппрокисмация второго порядка

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

def explict\_0(self, h, sigma, line, last\_line1, last\_line2, tau):

ans = sigma\*(2\*last\_line1[1] - (2 + 4\*h)\*last\_line1[0])

ans += (2 - 5\*tau\*tau)\*last\_line1[0] - last\_line2[0]

return ans

def explict\_l(self, h, sigma, line, last\_line1, last\_line2, tau):

ans = sigma\*(2\*last\_line1[-2] + (4\*h - 2)\*last\_line1[-1])

ans += (2 - 5\*tau\*tau)\*last\_line1[-1] - last\_line2[-1]

return ans

def implict\_0(self, h, sigma, l0, l1):

return 0, -(2 + 5\*h\*h + 4\*h + 1/sigma), 2, (-2\*l0[0] + l1[0])/sigma

def implict\_l(self, h, sigma, l0, l1):

return 2, -(2 + 5\*h\*h - 4\*h + 1/sigma), 0, (-2\*l0[-1] + l1[-1])/sigma

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

x = np.arange(l0, l1 + 0.002, 0.002)

y = np.arange(0, T + 0.002, 0.002)

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(n = 5, k=10, t=1):

schema = Explict\_Schema(T = t, aprx\_cls=Approx\_two\_two)

x, y, z = schema(N = n, K = k)

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, 1, t, u), 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()

plt.show()

Implicit\_plot()

None

def Explicit\_plot(n = 5, k=10, t=1):

schema = Implict\_Schema(T = t, aprx\_cls=Approx\_two\_two)

x, y, z = schema(N = n, K = k)

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, 1, t, u), 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()

plt.show()

Explicit\_plot()

None

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

ans = 0.0

for i in range(len(z)):

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

temp = abs(z[i][j] - f(x[i][j], y[i][j]))

ans = temp if temp > ans else ans

return ans

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

h = []

e = []

for N in range(4, 50, 1):

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

plt.plot([-3.5, -1], [-3, 2], label="Зависимость $O(h^2)$")

plt.xlabel("log h")

plt.ylabel("log e")

plt.legend()

plt.grid()

implict = Implict\_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(implict, u)

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

plt.xlabel("h")

plt.ylabel("e")

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="Неявная")

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

plt.plot([-3.5, -1], [-3, 2], label="Зависимость $O(h^2)$")

plt.xlabel("log h")

plt.ylabel("log e")

plt.legend()

plt.grid()

def 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 = 1, 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 = 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)

#График в логарифмической шкале

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 = Implict\_Schema(T = 1, aprx\_cls=Approx\_two\_two, order2nd=True)

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

plt.subplot(2, 1, 1)

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

tau, e = graphic\_tau(implict, u)

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

plt.xlabel("t")

plt.ylabel("e")

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([-4, 0], [-0.5, 3.5], label="Зависимость $O(t)$")

plt.plot([-4, 0], [0, 2], label="Зависимость $O(\sqrt{t})$")

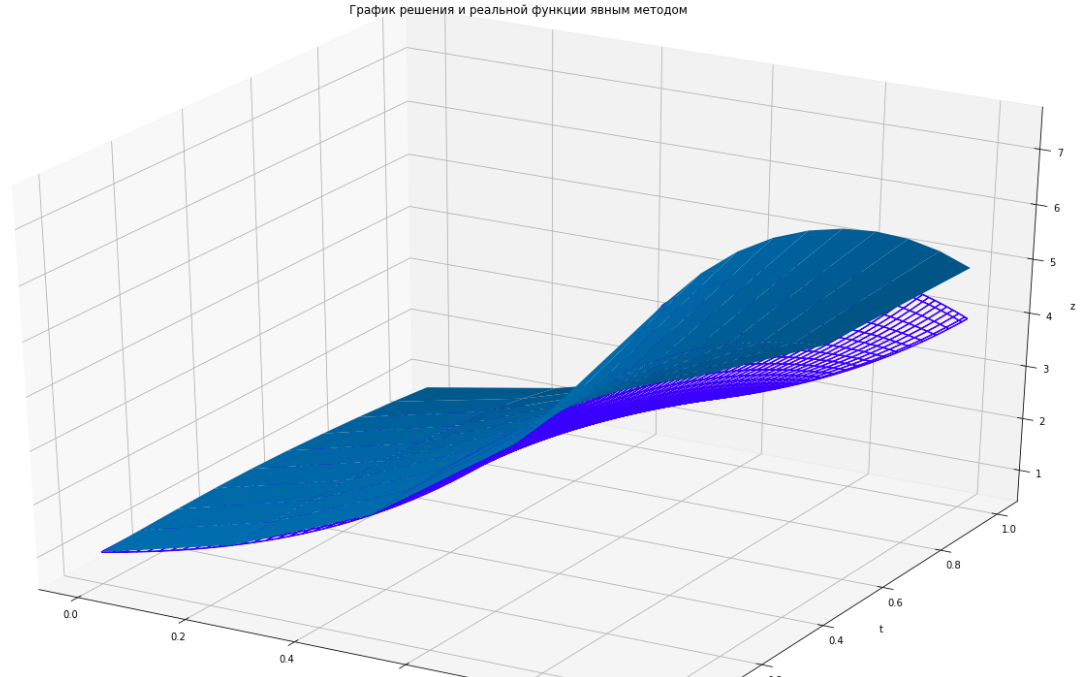
plt.xlabel("log t")

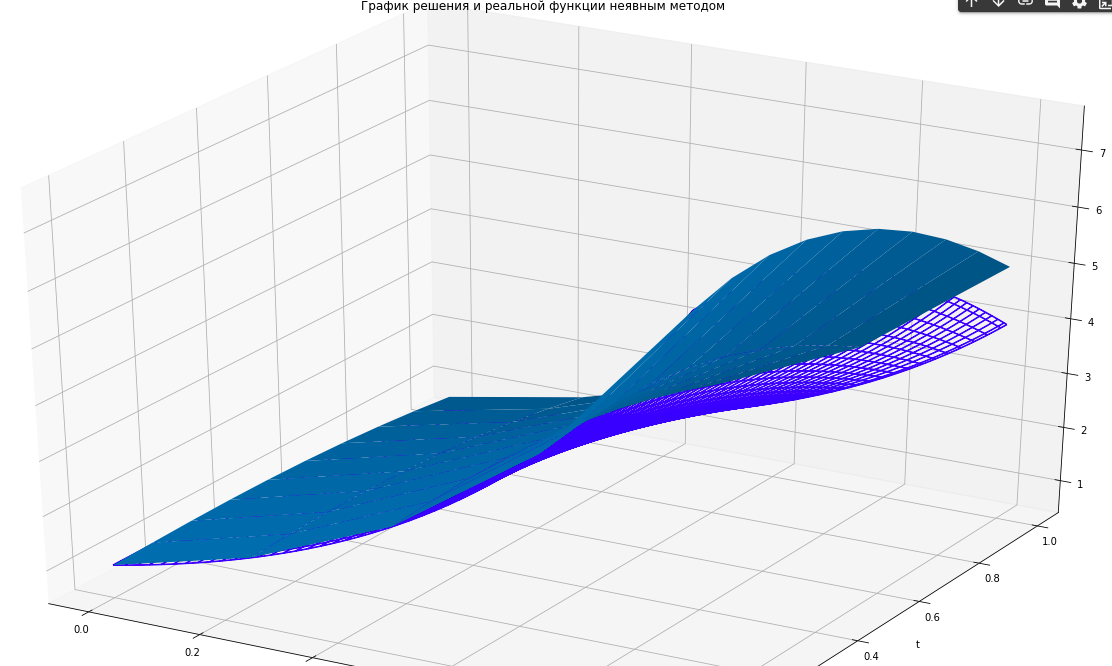
plt.ylabel("log e")

plt.legend()

plt.grid()

**5. Результаты**

****

****

**6. Выводы**

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