Московский авиационный институт

(национальный исследовательский университет)

Институт №8 «Информационные технологии и прикладная математика»

**Лабораторная работа №4 по численным методам**

**7 семестр**

Студент: Сухотин Игорь

Группа: М8O-402Б

Москва, 2020

# Постановка задачи

Пункт 4.1

Решить начально-краевую задачу для ДифУрЧаП гиперболического типа.

Использовать схемы:

* ЯВНУЮ со вторым порядком точности по и для аппроксимации самого уравнения, с 1 порядком точности по для вычисления первого внутреннего слоя

# Метод решения

Рассмотрим волновое уравнение в общем виде:

1. Сделаем *дискретизацию* пространственной области с малым шагом :

а также – временной области (для этого нужно выбрать конечный момент времени ) с малым шагом

Теперь для решения задачи (1)-(5) необходимо определить значение искомой функции в узлах дискретизации .

Далее, для краткости будем писать

1. *Явная схема*.

Аппроксимируем уравнение (1) конечными разностями, получим:

Разрешим (6) относительно и получим, пренебрегая :

Значения функции на нулевом слое мы знаем из начального условия . Значения на первом временном слое можно найти двумя способами:

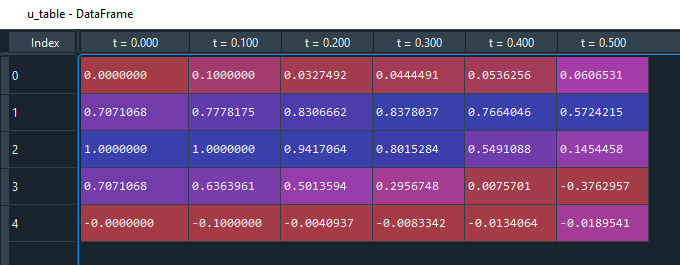
* Аппроксимация начальной скорости (3) с первым порядком:

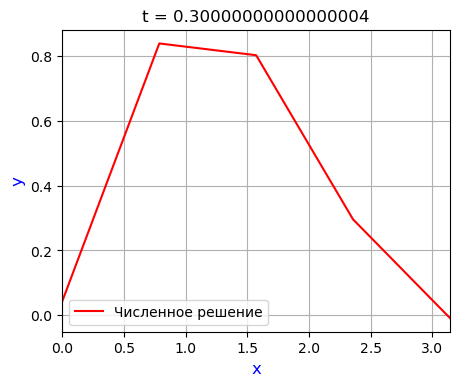
Работа с граничными условиями (4), (5) происходит аналогично предыдущей ЛР.

Порядок аппроксимации схемы без учёта аппроксимации начальной скорости , условная устойчивость .

# Результат решения

Написана программа на языке *Python* с применением фреймворков *numpy* и *matplotlib*. Среда программирования: *Spyder* 3.





# Выводы

Решена задача уравнения математической физики, а именно одномерное гиперболическое уравнение с краевыми условиями второго рода. Физический смысл этого уравнения и искомой функции заключается в распределении амплитуд волн тонкой струны.

# Листинг

***main.py***

*# -\*- coding: utf-8 -\*-*

*#%% Либы*

**import** numpy **as** np

**import** matplotlib.pyplot **as** plt

**from** num\_methods **import** wave\_eq\_solve

**from** pandas **import** DataFrame

*#%% Ввод условий*

a = 4

b = -1

c = -1

d = -4

f = **lambda** x, t: x \* t / (1 + x \*\* 2 + 2 \* t \*\* 2)

L = 3.1415926536

N = 5

T = 0.5

K = 6

phi0 = **lambda** t: -t / np.exp(t)

phil = **lambda** t: t \*\* 2 / np.exp(t)

alpha = [[0, -5], [0, -8]]

psi1 = **lambda** x: np.sin(x)

psi2 = **lambda** x: np.cos(x)

psi1\_d = **lambda** x: np.cos(x)

psi1\_dd = **lambda** x: -np.sin(x)

method = "explicit"

*#%% Решение*

x, t, u = wave\_eq\_solve(a, psi1, psi2, phi0, phil,

L, T,

N=N, K=K,

b=b, c=c, d=d, f=f,

alpha=alpha,

method=method, order1=1, order2=1)

h = x[1] - x[0]

tau = t[1] - t[0]

u\_table = DataFrame(u, columns=["t = {:.3f}".format(t\_) **for** t\_ **in** t ])

**print**(u\_table)

*#%% График*

t\_ = 0.3

ti = int(np.round(t\_ \* (K - 1) / T))

plt.figure(figsize=(5, 4), dpi=100)

plt.plot(x, u[:, ti], "r", markersize = 1, label="Численное решение")

plt.xlim((x[0], x[-1]))

plt.xlabel('x', fontsize=12, color='blue')

plt.ylabel('y', fontsize=12, color='blue')

plt.title("t = {0}".format(t[ti]))

plt.legend()

plt.grid(True)

plt.savefig("plot.png")

***num\_methods.py***

*# -\*- coding: utf-8 -\*-*

**import** numpy **as** np

**def** prog\_solve(el, d, eu, b):

"""

Метод прогонки решения системы с трёхдиагональной матрицей

"""

n = len(d)

eu = np.concatenate((eu, [0]))

P = np.zeros(n)

Q = np.zeros(n)

P[0] = -eu[0] / d[0]

Q[0] = b[0] / d[0]

**for** i **in** range(1, n):

P[i] = -eu[i] / (d[i] + el[i - 1] \* P[i - 1])

Q[i] = (b[i] - el[i - 1] \* Q[i - 1]) / (d[i] + el[i - 1] \* P[i - 1])

x = np.zeros(n)

x[-1] = Q[-1]

**for** i **in** range(n - 2, -1, -1):

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

**return** x

**def** wave\_eq\_solve(a, psi1, psi2, phi0, phil,

l, T,

N=11, K=101, sigma=None,

b=0, c=0, d=0, f=**lambda** x, t: np.zeros\_like(x) \* np.zeros\_like(t),

alpha=[[0, 1], [0, 1]],

order1=2, order2=3,

method="explicit",

psi1\_d=None, psi1\_dd=None):

"""

Parameters

----------

a : double

коэффициент теплопроводности, a > 0.

psi1 : указатель на функцию одной переменной

начальное условие u(x, 0)

psi2 : указатель на функцию одной переменной

начальная скорость u\_t(x, 0)

phi0 : указатель на функцию одной переменной

значение на левом конце

phil : указатель на функцию одной переменной

значение на правом конце

l : double

длина стержня

T : double

конечным момент времени

N : double, optional

число точек разбиения [0, l] (h = l / (N - 1)). The default is 11.

K : double, optional

число точек разбиения [0, T] (tau = T / (K - 1)). The default is 101.

sigma : double, optional

число Куррента, если указано, то K выбирается автоматически, так чтобы

выполнялось sigma = a \* tau \*\* 2 / h \*\* 2. The default is None.

b : double, optional

коэффициент перед u\_x. The default is 0.

c : double, optional

коэффициент перед u. The default is 0.

d : double, optional

коэффициент перед u\_t. The default is 0.

f : указатель на функцию двух переменных, optional

f(x, t). The default is lambda x, t: np.zeros\_like(x) \* np.zeros\_like(t).

alpha : матрица 2x2, optional

матрица краевых условий. The default is [[0, 1], [0, 1]].

order1 : 1 or 2, optional

порядок аппроксимации значений на 1 слое. The default is 2.

order2 : 1, 2 or 3, optional

порядок аппроксимации краевых условий. The default is 3.

method : "explicit" or "implicit", optional

явная или неявная схема. The default is "explicit".

psi1\_d : указатель на функцию одной переменной, optional

первая производная начального условия, если не указана, то рассчитывается численно со вторым

порядком. The default is None.

psi1\_dd : указатель на функцию одной переменной, optional

вторая производная начального условия, если не указана, то рассчитывается численно со вторым

порядком. The default is None.

Returns

-------

x : vector, double

разбиение пространственного отрезка.

t : vector, double

разбиение временного отрезка.

u : matrix, double

значение функции u[i, k] в точке x\_i, t\_k.

"""

x = np.linspace(0, l, N)

h = x[1] - x[0]

*# пользователь указал сигму*

**if** sigma != None:

tau = h \* np.sqrt(sigma / a)

t = np.arange(0, T+1e-9, tau)

K = len(t)

**else**:

t = np.linspace(0, T, K)

tau = t[1] - t[0]

a1 = alpha[0][0]

b1 = alpha[0][1]

a2 = alpha[1][0]

b2 = alpha[1][1]

psi1 = psi1(x)

psi2 = psi2(x)

phi0 = phi0(t)

phil = phil(t)

f = np.array([f(x\_i, t) **for** x\_i **in** x])

u = np.zeros((N, K))

u[:, 0] = psi1 *# 1 слой*

"""

Просчёт значений на втором слое

"""

**if** order1 == 1:

u[:, 1] = tau \* psi2 + psi1

**elif** order1 == 2:

*# 1-я производная psi1*

**if** psi1\_d == None: *# берём численно*

psi1\_d = np.zeros\_like(psi1)

psi1\_d[1:N-1] = (psi1[2:N] - psi1[0:N-2]) / (2 \* h)

psi1\_d[0] = (-3 \* psi1[0] + 4 \* psi1[1] - psi1[2]) / (2 \* h)

psi1\_d[N-1] = (psi1[-3] - 4 \* psi1[-2] + 3 \* psi1[-1]) / (2 \* h)

**else**: *# берём аналитически*

psi1\_d = psi1\_d(x)

*# 2-я производная psi1*

**if** psi1\_dd == None: *# берём численно*

psi1\_dd = np.concatenate(([(2 \* psi1[0] - 5 \* psi1[1] + 4 \* psi1[2] - psi1[3]) / h \*\* 2],

(psi1[0:N-2] - 2 \* psi1[1:N-1] + psi1[2:N]) / h \*\* 2,

[(-psi1[-4] + 4 \* psi1[-3] - 5 \* psi1[-2] + 2 \* psi1[-1]) / h \*\* 2]))

**else**: *# берём аналитически*

psi1\_dd = psi1\_dd(x)

u[:, 1] = psi1 \* (1 + c \* tau \*\* 2 / 2) + \

psi1\_d \* (tau \*\* 2 \* b / 2) + \

psi1\_dd \* (tau \*\* 2 \* a / 2) + \

psi2 \* (tau - tau \*\* 2 \* d / 2) + \

f[:, 0] \* (tau \*\* 2 / 2)

"""

Основной цикл

"""

**if** method == "explicit":

"""

Явная схема

"""

**for** k **in** range(2, K):

u[1:N-1, k] = (u[1:N-1, k-1] \* (2 / tau \*\* 2 - 2 \* a / h \*\* 2 + c) + \

u[1:N-1, k-2] \* (-(tau \*\* -2) + d / (2 \* tau)) + \

u[2:N , k-1] \* (a / h \*\* 2 + b / (2 \* h)) + \

u[0:N-2, k-1] \* (a / h \*\* 2 - b / (2 \* h)) + \

f[1:N-1, k-1]) / \

(tau \*\* -2 + d / (2 \* tau))

"""

Различные варианты аппроксимации граничных условий

"""

**if** order2 == 1: *# Двухточечная аппроксимация граничных условий с 1-м порядком*

u[0, k] = (a1 \* u[1, k] - h \* phi0[k]) / (a1 - h \* b1)

u[N-1, k] = (a2 \* u[N-2, k] + h \* phil[k]) / (a2 + h \* b2)

**elif** order2 == 2 **or** order2 == 3: *# Трёхточечная аппроксимация граничных условий со 2-м порядком*

mul0 = - 3 \* a1 / (2 \* h) + b1

mull = 3 \* a2 / (2 \* h) + b2

u[0, k] = (- 4 \* a1 \* u[1, k] / (2 \* h) + a1 \* u[2, k] / (2 \* h) + phi0[k]) / mul0

u[N-1, k] = (4 \* a2 \* u[N-2, k] / (2 \* h) - a2 \* u[N-3, k] / (2 \* h) + phil[k]) / mull

**elif** method == "implicit":

"""

Неявная схема

"""

"""

Различные варианты аппроксимации граничных условий

"""

**if** order2 == 1: *# Двухточечная аппроксимация граничных условий с 1-м порядком*

**for** k **in** range(2, K):

el = np.concatenate((np.full(N-2, -a / h \*\* 2 + b / (2 \* h)),

[-a2 / h]))

ld = np.concatenate(([-a1 / h + b1],

np.full(N-2, tau \*\* -2 + d / (2 \* tau) + 2 \* a / h \*\* 2 - c),

[a2 / h + b2]))

eu = np.concatenate(([a1 / h],

np.full(N-2, -a / h \*\* 2 - b / (2 \* h))))

rb = np.concatenate(([phi0[k]],

u[1:N-1, k-1] \* (2 / tau \*\* 2) + u[1:N-1, k-2] \* (-(tau \*\* -2) + d / (2 \* tau)) + f[1:N-1, k],

[phil[k]]))

u[:, k] = prog\_solve(el, ld, eu, rb)

**elif** order2 == 2: *# Трёхточечная аппроксимация граничных условий со 2-м порядком*

**for** k **in** range(2, K):

mul0 = -a1 \* h / ( 2 \* a + b \* h)

mull = a2 \* h / ( 2 \* a - b \* h)

el = np.concatenate((np.full(N-2, -a / h \*\* 2 + b / (2 \* h)),

[-2 \* a2 / h + mull \* (tau \*\* -2 + d / (2 \* tau) + 2 \* a / h \*\* 2 - c)]))

ld = np.concatenate(([-3 \* a1 / (2 \* h) + b1 + mul0 \* (-a / h \*\* 2 + b / (2 \* h))],

np.full(N-2, tau \*\* -2 + d / (2 \* tau) + 2 \* a / h \*\* 2 - c),

[3 \* a2 / (2 \* h) + b2 + mull \* (-a / h \*\* 2 - b / (2 \* h))]))

eu = np.concatenate(([2 \* a1 / h + mul0 \* (tau \*\* -2 + d / (2 \* tau) + 2 \* a / h \*\* 2 - c)],

np.full(N-2, -a / h \*\* 2 - b / (2 \* h))))

rb = np.concatenate(([phi0[k] + mul0 \* (u[1, k-1] \* (2 / tau \*\* 2) + u[1, k-2] \* (-(tau \*\* -2) + d / (2 \* tau)) + f[1, k])],

u[1:N-1, k-1] \* (2 / tau \*\* 2) + u[1:N-1, k-2] \* (-(tau \*\* -2) + d / (2 \* tau)) + f[1:N-1, k],

[phil[k] + mull \* (u[N-2, k-1] \* (2 / tau \*\* 2) + u[N-2, k-2] \* (-(tau \*\* -2) + d / (2 \* tau)) + f[N-2, k])]))

u[:, k] = prog\_solve(el, ld, eu, rb)

**elif** order2 == 3: *# Двухточечная аппроксимация граничных условий со 2-м порядком*

**for** k **in** range(2, K):

mul0 = 2 \* a / (2 \* a - b \* h)

mull = 2 \* a / (2 \* a + b \* h)

el = np.concatenate((np.full(N-2, -a / h \*\* 2 + b / (2 \* h)),

[-a2 / h \* mull]))

ld = np.concatenate(([(-a1 / h - a1 \* h / (2 \* a \* tau \*\* 2) - a1 \* d \* h / (4 \* a \* tau) + a1 \* h \* c / (2 \* a)) \* mul0 + b1],

np.full(N-2, tau \*\* -2 + d / (2 \* tau) + 2 \* a / h \*\* 2 - c),

[( a2 / h + a2 \* h / (2 \* a \* tau \*\* 2) + a2 \* d \* h / (4 \* a \* tau) - a2 \* h \* c / (2 \* a)) \* mull + b2]))

eu = np.concatenate(([ a1 / h \* mul0],

np.full(N-2, -a / h \*\* 2 - b / (2 \* h))))

rb = np.concatenate(([phi0[k] + u[0 , k-1] \* (-a1 \* h / (a \* tau \*\* 2)) \* mul0 + u[0 , k-2] \* ( a1 \* h / (2 \* a \* tau \*\* 2) - a1 \* d \* h / (4 \* a \* tau)) \* mul0 - f[0 , k] \* a1 \* h / (2 \* a - b \* h)],

u[1:N-1, k-1] \* (2 / tau \*\* 2) + u[1:N-1, k-2] \* (-(tau \*\* -2) + d / (2 \* tau)) + f[1:N-1, k],

[phil[k] + u[N-1, k-1] \* ( a2 \* h / (a \* tau \*\* 2)) \* mull + u[N-1, k-2] \* (-a2 \* h / (2 \* a \* tau \*\* 2) + a2 \* d \* h / (4 \* a \* tau)) \* mull + f[N-1, k] \* a2 \* h / (2 \* a + b \* h)]))

u[:, k] = prog\_solve(el, ld, eu, rb)

**return** x, t, u