|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **Министерство науки и высшего образования Российской Федерации**  Калужский филиал  федерального государственного бюджетного  образовательного учреждения высшего образования  ***«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»***  ***(КФ МГТУ им. Н.Э. Баумана)*** |

|  |  |
| --- | --- |
| **ФАКУЛЬТЕТ** | **ИУК «Информатика и управление»** |
| **КАФЕДРА** | **ИУК4 «Программное обеспечение ЭВМ,** |
| **информационные технологии»** | |

**Лабораторная работа №3**

**«Метод сеток для решения уравнения гиперболического типа»**

**ДИСЦИПЛИНА: «Моделирование»**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Выполнил: студент гр. ИУК4-62Б | |  |  | ( | Калашников А.С. | ) |
|  |  |  | (подпись) |  | (Ф.И.О.) |  |
| Проверил: | |  |  | ( | Никитенко У.В. | ) |
|  |  |  | (подпись) |  | (Ф.И.О.) |  |

|  |  |
| --- | --- |
| Дата сдачи (защиты):  Результаты сдачи (защиты): | |
|  | - Балльная оценка:  - Оценка: |

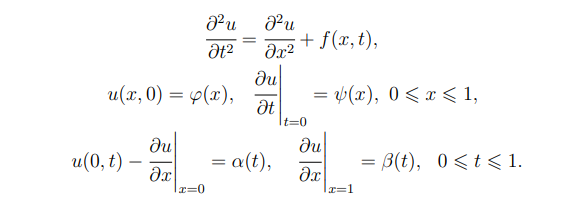
Калуга, 2023

**Цель работы**: сформировать практические навыки анализа возможностей построения и выделения наиболее важных свойств объектов моделей для моделирования и использования специализированных программных пакетов и библиотек для стандартных вычислений и визуализации результатов численного или приближенно-аналитического решения ДУЧП2 гиперболического типа на основе сравнения результатов.

**Постановка задачи**

**Вариант 28**

Найти решение задачи:

****

используя различные разностные схемы:

* явную схему порядка с аппроксимацией производных в граничных условиях с порядком ;
* явную схему порядка с аппроксимацией производных в граничных условиях с порядком ;;
* схему с весами порядка и при (с аппроксимацией производных в граничных условиях с порядком ).

По решению задачи должен быть представлен отчет, содержащий

1) Алгоритм решения задачи.

2) Тестирование алгоритма на решениях, для которых разностная схема точно аппроксимирует дифференциальную задачу.

3) Тестирование алгоритма, например, на решениях , на которых разностная схема неточно аппроксимирует дифференциальную задачу.

4) Таблицы решения на «крупной» сетке независимо от шагов по t и x, с которыми строится решение, следующего вида ().

5) Таблицы, характеризующие точность решения и внутреннюю сходимость.

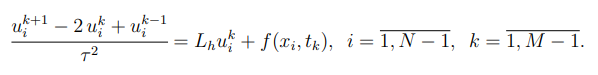
**Ход решения с явной разностной схемой**

Алгоритм решения

1. Находим

2. Находим

3. Находим из



4. Находим

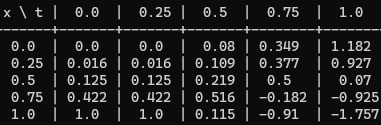
5. Находим

Запишем исходное выражение с использованием узловых значений:

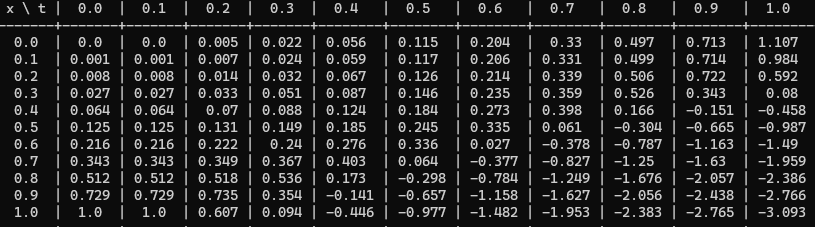
Выразим следующее значение из предыдущих:

Вычислим остальные функции, подставив значение :

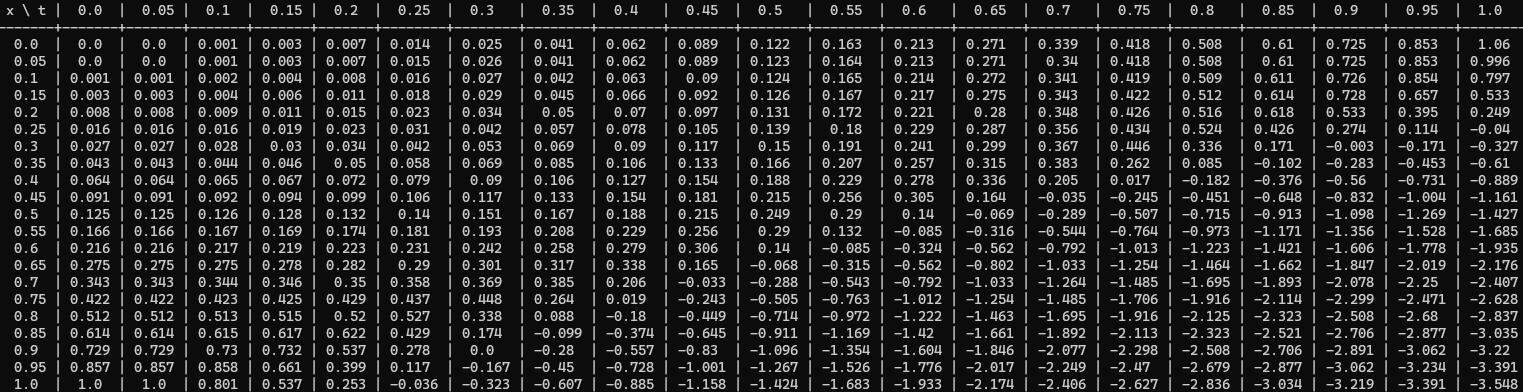
Подставим эти значения в полученный код и получим следующее решение:



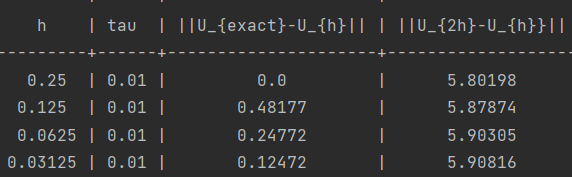
**Рисунок 1 –** Результат аппроксимации при



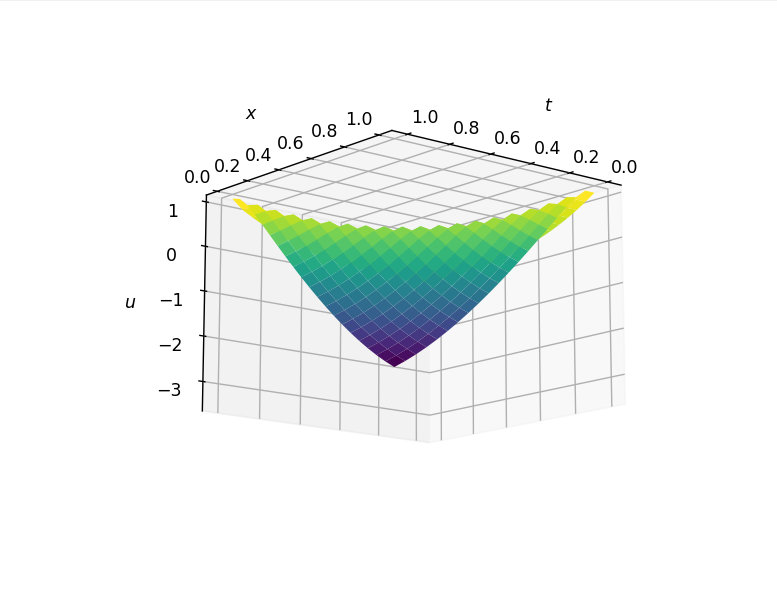
**Рисунок 2 –** Результат аппроксимации при



**Рисунок 3 –** Результат аппроксимации при



**Рисунок 4 –** Таблица, характеризующая точность решения

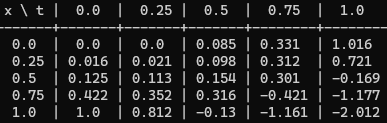


**Рисунок 5 –** График аппроксимированной функции

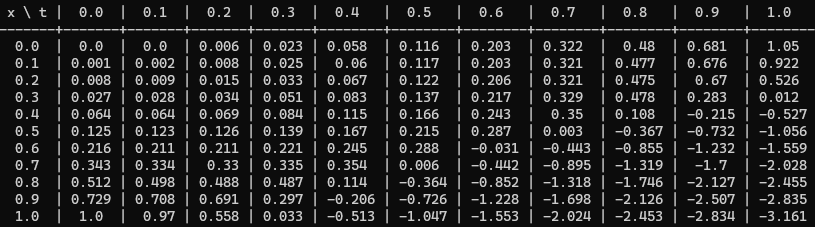
**2.** Составим явную схему порядка с аппроксимацией производных в граничных условиях с порядком .

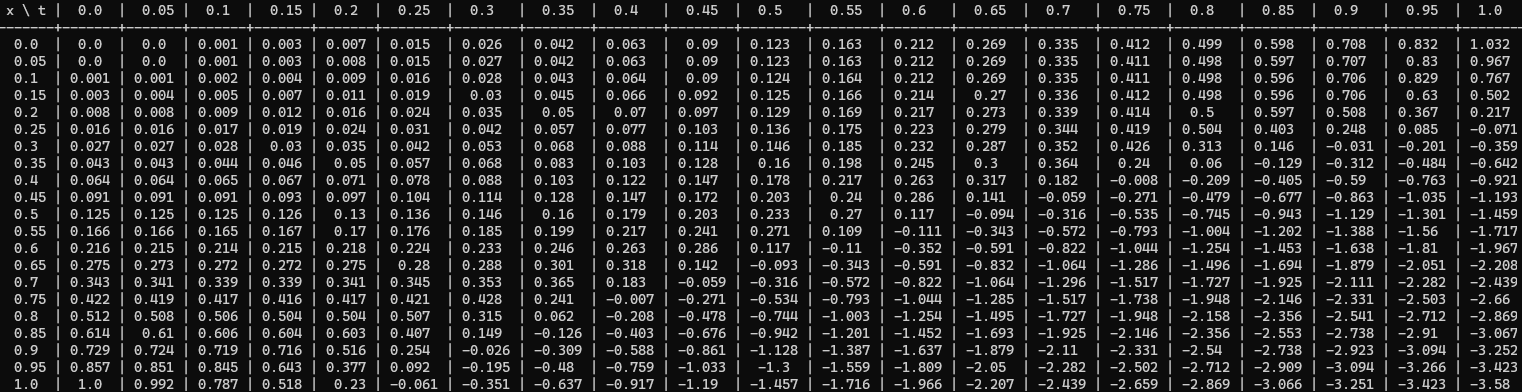
Для решения поставленной задачи, используя ряд Тейлора, получим:

Подставим эти значения в полученный код и получим следующее решение:

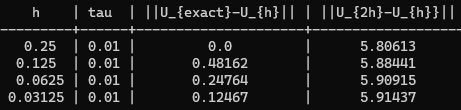


**Рисунок 6 –** Результат аппроксимации при

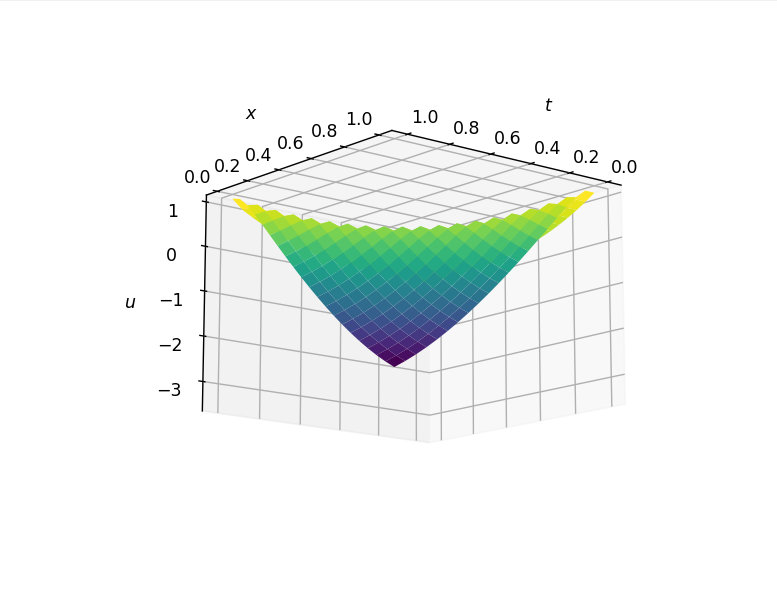
** Рисунок 7 –** Результат аппроксимации при

**

**Рисунок 8 –** Результат аппроксимации при



**Рисунок 9 –** Таблица, характеризующая точность решения



**Рисунок 10 –** График аппроксимированной функции

**3.** Составим схему с весами порядка и при (с аппроксимацией производных в граничных условиях с порядком ).

Алгоритм решения

1. Находим

2. Находим

3. Находим

4. Находим

Так как к моменту определения решения на (k + 1)-ом слое решение на предыдущих слоях (k − 1) и k уже известно, систему (12) перепишем следующим образом:

Для решения на каждом последующем слое необходимо решить систему уравнений. Составим коэффициенты для этой системы.

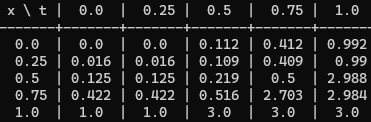
Подставим значение из прошлого случая:

Исходя из полученного уравнения и граничных условий, можно составить следующую систему:

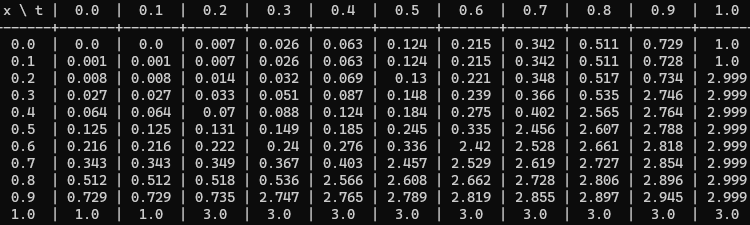
Для того, чтобы получить необходимые значения , нужно решить эту систему. Так как матрица будет являться трёхдиагональной, воспользуемся методом прогонки.

В зависимости от значения разностная схема будет менять свой вид.

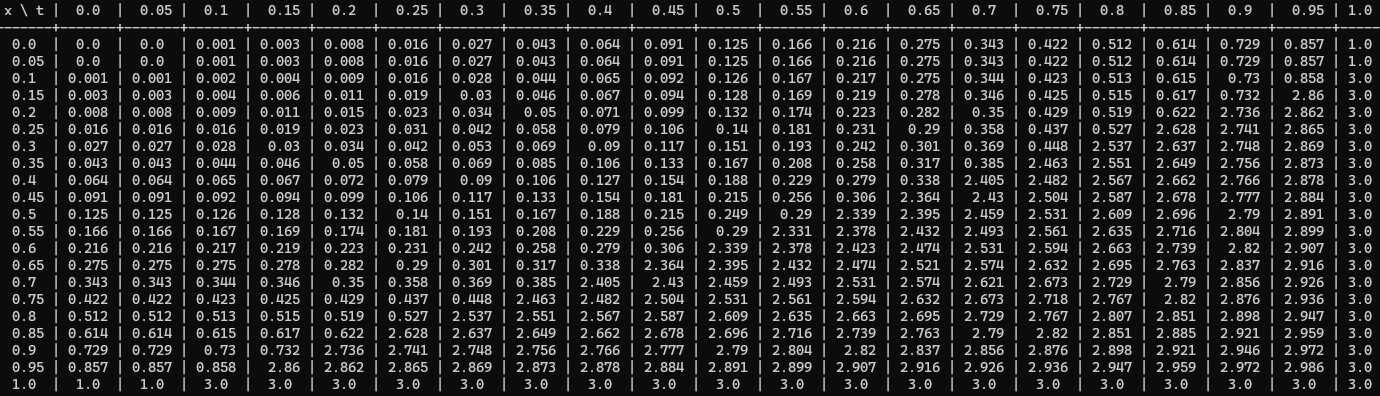
Вычислим значения при .



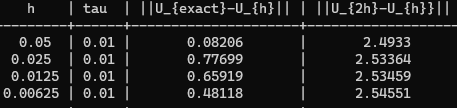
**Рисунок 11 –** Результат аппроксимации при



**Рисунок 12 –** Результат аппроксимации при



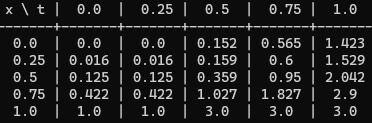
**Рисунок 13 –** Результат аппроксимации при



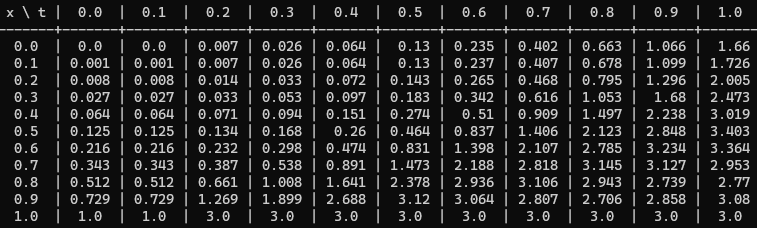
**Рисунок 14 –** Таблица, характеризующая точность решения

При разностная схема приобретает вид такой же, как и в пункте 1 (за исключением начальных условий, аппроксимируемых на порядок ниже).

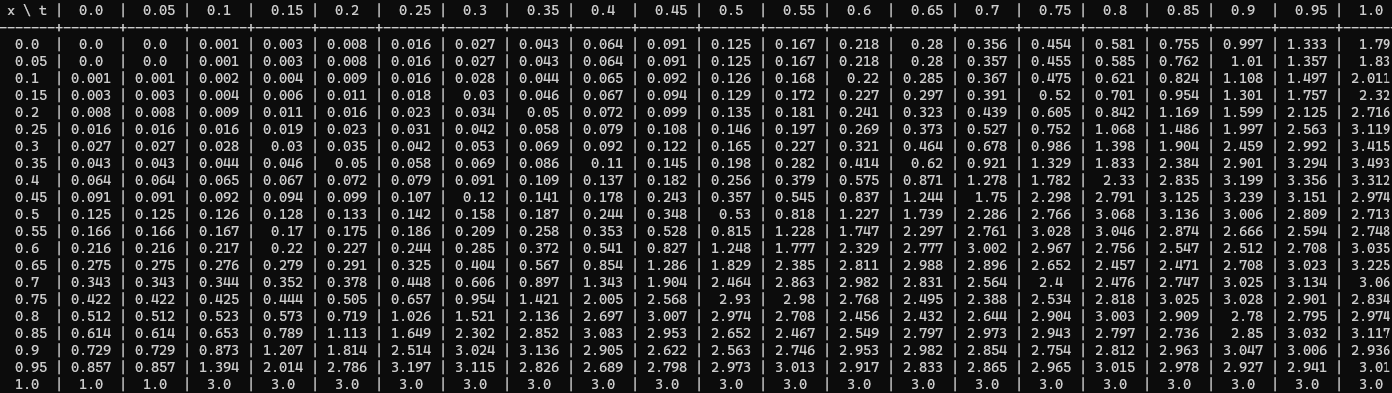
Вычислим значения при .



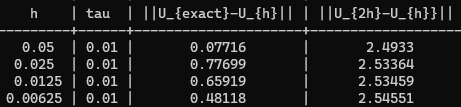
**Рисунок 15 –** Результат аппроксимации при



**Рисунок 16 –** Результат аппроксимации при



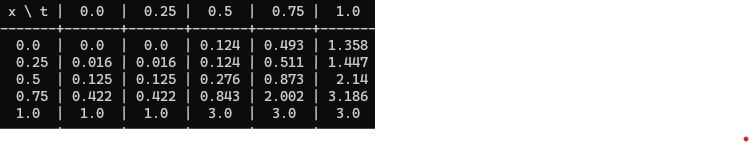
**Рисунок 17 –** Результат аппроксимации при



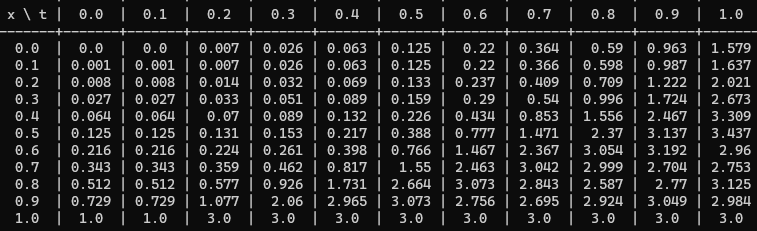
**Рисунок 18 –** Таблица, характеризующая точность решения

При получаем неявную разностную схему с семиточечным шаблоном и точность, превосходящую .

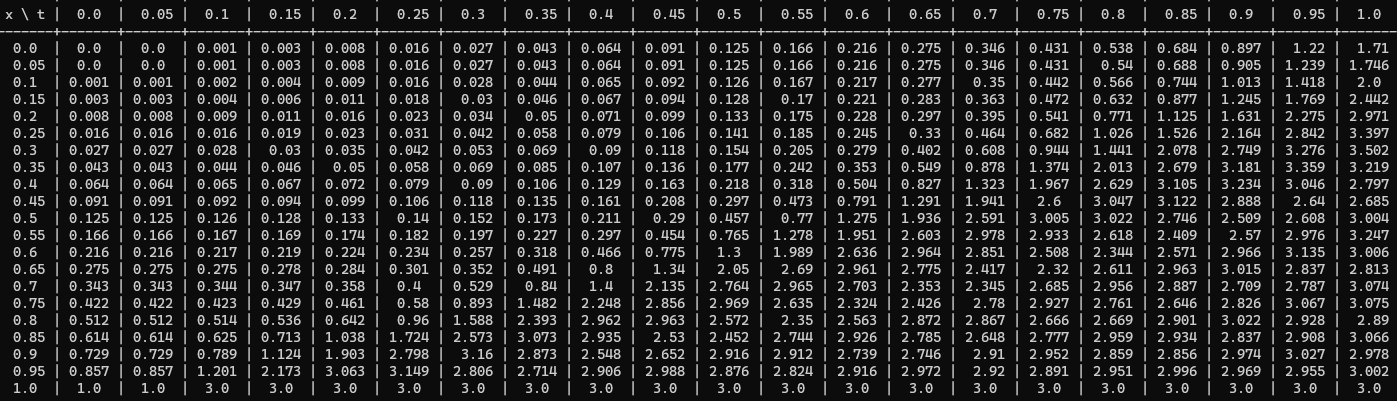
Вычислим значения при .



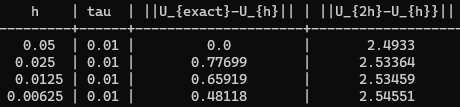
**Рисунок 19 –** Результат аппроксимации при



**Рисунок 20 –** Результат аппроксимации при



**Рисунок 21 –** Результат аппроксимации при



**Рисунок 22 –** Таблица, характеризующая точность решения

При получаем неявную разностную схему с девятиточечным шаблоном и точность, превосходящую .

**Вывод**: в ходе выполнения работы были сформированы практические навыки анализа возможностей построения и выделения наиболее важных свойств объектов моделей для моделирования и использования специализированных программных пакетов и библиотек для стандартных вычислений и визуализации результатов численного или приближенно-аналитического решения ДУЧП2 гиперболического типа на основе сравнения результатов.

**ПРИЛОЖЕНИЯ**

**Листинг решения пункта 1**

import numpy as np  
import matplotlib.pyplot as plt  
from prettytable import PrettyTable  
  
def f(x, t):  
 return 6 \* t - 6 \* x  
  
  
def lu(u: np.array, x: np.linspace, t: np.linspace,i, k, h):  
 return (u[i + 1, k] - 2 \* u[i, k] + u[i - 1, k]) / pow(h, 2)  
  
  
def solve(h, tau):  
 x\_min = 0  
 x\_max = 1  
 xs = np.arange(x\_min, x\_max + h, h)  
 n\_x = len(xs)  
  
 t\_min = 0  
 t\_max = 1  
 ts = np.arange(t\_min, t\_max + tau, tau)  
 n\_t = len(ts)  
  
 phi = lambda x: x \*\* 3  
 psi = lambda x: 0  
 alpha = lambda t: t \*\* 3  
 beta = lambda x: 3  
  
 U = np.zeros((n\_x, n\_t))  
 U[:, 0] = [phi(x) for x in xs]  
 U[:, 1] = [tau \* psi(x) + phi(x) for x in xs]  
  
 for k in range(1, n\_t - 1):  
 for i in range(1, n\_x - 1):  
 U[i, k + 1] = 2 \* U[i, k] - U[i, k - 1] + tau \*\* 2 \* (lu(U, xs, ts, i, k, h) + f(xs[i], ts[k]))  
 U[0, k + 1] = (2 \* h \* alpha(ts[k + 1]) - U[2, k + 1] + 4 \* U[1, k + 1]) / (2 \* h + 3)  
 U[-1, k + 1] = (2 \* h \* beta(ts[k + 1]) - 4 \* U[-2, k + 1] + U[-3, k + 1]) / -3  
  
 return [xs, ts, U]  
  
  
def makeTableFromResult(xs, ts, U):  
 table = PrettyTable()  
 ts = ts.round(3)  
 xs = xs.round(3)  
 U = U.round(3)  
 table.add\_column("x \ t", xs)  
 for k in range(len(ts)):  
 table.add\_column(f"{ts[k]}", U[:, k])  
 return table  
  
  
[xs, ts, U] = solve(0.05, 0.05) #Измение шага  
  
print("Результат:")  
print(makeTableFromResult(xs, ts, U))  
  
fig = plt.figure()  
ax = plt.axes(projection='3d')  
X, Y = np.meshgrid(ts, xs)  
ax.plot\_surface(X, Y, U, rstride=1, cstride=1,cmap='viridis', edgecolor='none')  
ax.set\_xlabel('$t$')  
ax.set\_ylabel('$x$')  
ax.set\_zlabel('$u$')  
plt.show()  
  
  
def makeTableFromStep(hs, taus, diff, exact\_diff):  
 table = PrettyTable()  
 table.add\_column("h", hs)  
 table.add\_column("tau", taus)  
 table.add\_column("||U\_{exact}-U\_{h}||", diff)  
 table.add\_column("||U\_{2h}-U\_{h}}||", exact\_diff)  
 return table  
  
  
h = 0.25  
tau = 0.01  
  
hs = []  
taus = []  
exact\_diff = []  
last\_diff = []  
  
[\_, \_, last\_u] = solve(h, tau)  
last\_u = last\_u[0::2]  
  
for i in range(4):  
 [xs, ts, U] = solve(h, tau)  
  
 u = lambda t, x: x \*\* 3 + t \*\* 3  
 U\_exact = np.array([[u(t, x) for t in ts] for x in xs])  
  
 hs.append(h)  
 taus.append(tau)  
 exact\_diff.append(np.amax(np.abs(U - U\_exact)))  
 last\_diff.append(np.amax(np.abs(last\_u - U[0::2])))  
  
 h /= 2  
 last\_u = U  
  
hs = np.array(hs).round(5)  
taus = np.array(taus).round(5)  
exact\_diff = np.array(exact\_diff).round(5)  
last\_diff = np.array(last\_diff).round(5)  
  
print(makeTableFromStep(hs, taus, last\_diff, exact\_diff))

**Листинг решения пункта 2**

import numpy as np  
import matplotlib.pyplot as plt  
from scipy.interpolate import approximate\_taylor\_polynomial  
from prettytable import PrettyTable  
  
def f(x, t):  
 return 6 \* t - 6 \* x  
  
def lu(u: np.array, x: np.linspace, t: np.linspace,i, k, h):  
 return (u[i + 1, k] - 2 \* u[i, k] + u[i - 1, k]) / pow(h, 2)  
  
def lphi(phi, x):  
 return approximate\_taylor\_polynomial(phi, 0, 2, 1)(x)  
  
def solve(h, tau):  
 x\_min = 0  
 x\_max = 1  
 xs = np.arange(x\_min, x\_max + h, h)  
 n\_x = len(xs)  
  
 t\_min = 0  
 t\_max = 1  
 ts = np.arange(t\_min, t\_max + tau, tau)  
 n\_t = len(ts)  
  
 phi = lambda x: x \*\* 3  
 psi = lambda x: 0  
 alpha = lambda t: t \*\* 3  
 beta = lambda x: 3  
  
 U = np.zeros((n\_x, n\_t))  
 U[:, 0] = [phi(x) for x in xs]  
 U[:, 1] = [tau \* psi(x) + phi(x) + tau \*\* 2 / 2 \* (lphi(phi, x) \* f(x, 0)) for x in xs]  
  
 for k in range(1, n\_t - 1):  
 for i in range(1, n\_x - 1):  
 U[i, k + 1] = 2 \* U[i, k] - U[i, k - 1] + \  
 tau \*\* 2 \* (lu(U, xs, ts, i, k, h) + f(xs[i], ts[k]))  
 U[0, k + 1] = (2 \* h \* alpha(ts[k + 1]) - U[2, k + 1] + 4 \* U[1, k + 1]) \  
 / (2 \* h + 3)  
 U[-1, k + 1] = (2 \* h \* beta(ts[k + 1]) - 4 \* U[-2, k + 1] + U[-3, k + 1]) / -3  
  
 return [xs, ts, U]  
  
def makeTableFromResult(xs, ts, U):  
 table = PrettyTable()  
 ts = ts.round(3)  
 xs = xs.round(3)  
 U = U.round(3)  
 table.add\_column("x \ t", xs)  
 for k in range(len(ts)):  
 table.add\_column(f"{ts[k]}", U[:, k])  
 return table  
  
  
[xs, ts, U] = solve(0.05, 0.05)  
  
print("Результат:")  
print(makeTableFromResult(xs, ts, U))  
  
fig = plt.figure()  
ax = plt.axes(projection='3d')  
X, Y = np.meshgrid(ts, xs)  
ax.plot\_surface(X, Y, U, rstride=1, cstride=1,  
 cmap='viridis', edgecolor='none')  
ax.set\_xlabel('$t$')  
ax.set\_ylabel('$x$')  
ax.set\_zlabel('$u$')  
plt.show()  
  
  
def makeTableFromStep(hs, taus, diff, exact\_diff):  
 table = PrettyTable()  
 table.add\_column("h", hs)  
 table.add\_column("tau", taus)  
 table.add\_column("||U\_{exact}-U\_{h}||", diff)  
 table.add\_column("||U\_{2h}-U\_{h}}||", exact\_diff)  
 return table  
  
  
h = 0.25  
tau = 0.01  
  
hs = []  
taus = []  
exact\_diff = []  
last\_diff = []  
  
[\_, \_, last\_u] = solve(h, tau)  
last\_u = last\_u[0::2]  
  
for i in range(4):  
 [xs, ts, U] = solve(h, tau)  
  
 u = lambda t, x: x \*\* 3 + t \*\* 3  
 U\_exact = np.array([[u(t, x) for t in ts] for x in xs])  
  
 hs.append(h)  
 taus.append(tau)  
 exact\_diff.append(np.amax(np.abs(U - U\_exact)))  
 last\_diff.append(np.amax(np.abs(last\_u - U[0::2])))  
  
 h /= 2  
 last\_u = U  
  
hs = np.array(hs).round(5)  
taus = np.array(taus).round(5)  
exact\_diff = np.array(exact\_diff).round(5)  
last\_diff = np.array(last\_diff).round(5)  
  
print(makeTableFromStep(hs, taus, last\_diff, exact\_diff))

**Листинг решения пункта 3**

import numpy as np  
import matplotlib.pyplot as plt  
from mpl\_toolkits import mplot3d  
import scipy.linalg as la  
from prettytable import PrettyTable  
  
def f(x, t):  
 return 6 \* t - 6 \* x

def lu(u: np.array, x: np.linspace, t: np.linspace,i, k, h):  
 return (u[i + 1, k] - 2 \* u[i, k] + u[i - 1, k]) / pow(h, 2)  
  
def solve(h, tau, sigma):  
 x\_min = 0  
 x\_max = 1  
 xs = np.arange(x\_min, x\_max + h, h)  
 n\_x = len(xs)  
  
 t\_min = 0  
 t\_max = 1  
 ts = np.arange(t\_min, t\_max + tau, tau)  
 n\_t = len(ts)  
  
 phi = lambda x: x \*\* 3  
 psi = lambda x: 0  
 alpha = lambda t: t \*\* 3  
 beta = lambda x: 3  
  
 U = np.zeros((n\_x, n\_t))  
 G = np.zeros((n\_x, n\_t))  
 U[:, 0] = [phi(x) for x in xs]  
 U[:, 1] = [tau \* psi(x) + phi(x) for x in xs]  
 A = np.zeros((n\_x - 1))  
 B = np.zeros((n\_x))  
 C = np.zeros((n\_x - 1))  
  
 for k in range(1, n\_t - 1):  
 for i in range(1, n\_x - 1):  
 G[i, k + 1] = (-2 \* U[i, k] + U[i, k - 1]) / tau \*\* 2 \  
 - (1 - 2 \* sigma) \* lu(U, xs, ts, i, k, h) \  
 - sigma \* lu(U, xs, ts, i, k - 1, h) \  
 - f(xs[i], ts[k])  
 A[i - 1] = (sigma / h \*\* 2)  
 B[i] = (-2 / h \*\* 2) \* sigma - 1 / tau \*\* 2  
 C[i] = (sigma / h \*\* 2)  
  
 B[0] = (1 + 1 / h)  
 C[0] = -1 / h  
 A[-1] = 0  
 B[-1] = 1  
  
 G[0, k + 1] = alpha(ts[k + 1])  
 G[-1, k + 1] = beta(ts[k + 1])  
  
 matrix = np.array([[0, \*C], B, [\*A, 0]])  
  
 U[:, k + 1] = la.solve\_banded((1, 1), matrix, G[:, k + 1])  
  
 return [xs, ts, U]  
  
  
def makeTableFromResult(xs, ts, U):  
 table = PrettyTable()  
 ts = ts.round(4)  
 xs = xs.round(4)  
 U = U.round(5)  
 table.add\_column("x \ t", xs)  
 for k in range(len(ts)):  
 table.add\_column(f"{ts[k]}", U[:, k])  
 return table  
  
  
[xs, ts, U] = solve(0.25, 0.25, 0.25)  
  
fig = plt.figure()  
ax = plt.axes(projection='3d')  
X, Y = np.meshgrid(ts, xs)  
ax.plot\_surface(X, Y, U, rstride=1, cstride=1,  
 cmap='viridis', edgecolor='none')  
ax.set\_xlabel('$t$')  
ax.set\_ylabel('$x$')  
ax.set\_zlabel('$u$')  
plt.show()  
  
def makeTableFromStep(hs, taus, diff, exact\_diff):  
 table = PrettyTable()  
 table.add\_column("h", hs)  
 table.add\_column("tau", taus)  
 table.add\_column("||U\_{exact}-U\_{h}||", diff)  
 table.add\_column("||U\_{2h}-U\_{h}}||", exact\_diff)  
 return table  
  
def makeTableFromResult(xs, ts, U):  
 table = PrettyTable()  
 ts = ts.round(3)  
 xs = xs.round(3)  
 U = U.round(3)  
 table.add\_column("x \ t", xs)  
 for k in range(len(ts)):  
 table.add\_column(f"{ts[k]}", U[:, k])  
 return table  
  
  
[xs, ts, U] = solve(0.25, 0.25, 0.25)  
  
print("Результат:")  
print(makeTableFromResult(xs, ts, U))  
  
h = 0.25  
tau = 0.01  
  
hs = []  
taus = []  
exact\_diff = []  
last\_diff = []  
  
[\_, \_, last\_u] = solve(h, tau, 0.25)  
last\_u = last\_u[0::2]  
  
for i in range(4):  
 [xs, ts, U] = solve(h, tau, 0.25)  
  
 u = lambda t, x: x \*\* 3 + t \*\* 3  
 U\_exact = np.array([[u(t, x) for t in ts] for x in xs])  
  
 hs.append(h)  
 taus.append(tau)  
 exact\_diff.append(np.amax(np.abs(U - U\_exact)))  
 last\_diff.append(np.amax(np.abs(last\_u - U[0::2])))  
  
 h /= 2  
 last\_u = U  
  
hs = np.array(hs).round(5)  
taus = np.array(taus).round(5)  
exact\_diff = np.array(exact\_diff).round(5)  
last\_diff = np.array(last\_diff).round(5)  
  
print(makeTableFromStep(hs, taus, last\_diff, exact\_diff))