**ОТЧЁТ**

**ПО ЛАБОРАТОРНОЙ РАБОТЕ № 9**

**УРАВНЕНИЯ ПАРАБОЛИЧЕСКОГО ТИПА**

**(Вариант 5)**

*Выполнил студент 3 курса ПМ*

*Бут Дмитрий*

***Постановка задачи:*** усвоить сущность и методы решения ***линейного дифференциального уравнения 2-го порядка параболического типа***.

Численное решение дифференциального уравнения в частных производных предполагает получение двумерной числовой таблицы приближенных значений *Uij* искомой функции *U*(*t,x)* с заданной точностью для некоторых значений аргументов

*xj ∈* [*a*, *b*], *ti ∈* [*c*, *d*]

Численное решение таких дифференциальных уравнений возможно методами конечных разностей.

Погрешность решения, найденного этими методами, оценивается величиной O(*τ p, h q*)*,* где *p*, *q* - порядок метода.

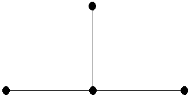
***Задание.***

Решить параболическое уравнение

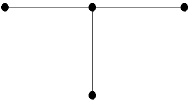


явным методом и неявным методом.

Шаблон для явного метода:



Шаблон для неявного метода:



Вывести результаты в виде двумерных графиков U(x,t).

Неявные схемы решать с помощью прогонки.

**Метод прогонки РАСПИСАТЬ подробно!**

Входные данные: [*a*, *b*] = [0; 1], [*c*, *d*] = [0; 10], D=1. Погрешность решения 0,01

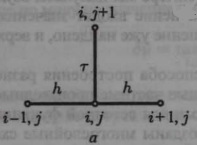




***Результаты расчетов***

**Явный метод.**

Используемый шаблон:

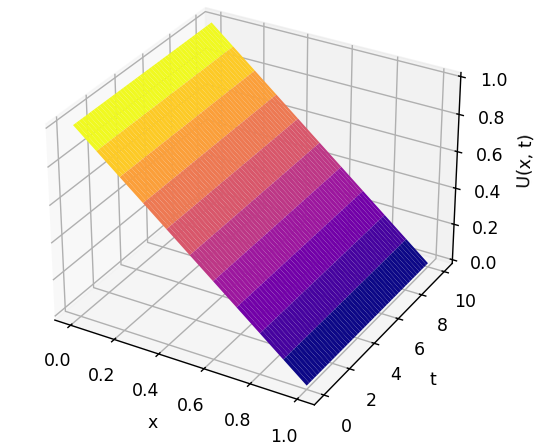


Заменяя в исходном уравнении частные производные искомой функции с помощью отношений конечных разностей, получаем разностную схему:

Отсюда можно найти явное выражение для значения сеточной функции на (j+ 1)-м слое:

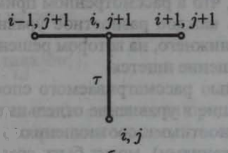
где

Получив точки, построим трёхмерный график:



**Неявный метод.**

Используемый шаблон:



Заменяя в исходном уравнении частные производные искомой функции с помощью отношений конечных разностей, получаем разностную схему:

Введём параметр

Приведём к стандартному трёхдиагональному виду:

(1)

Тогда , , ,

Применим метод прогонки:

Пусть , тогда

Используя это соотношение, получим выражение через :

Подставим в уравнение (1):

где  — правая часть *i*-го уравнения. Это соотношение будет выполняться независимо от решения, если потребовать:

Отсюда следует:

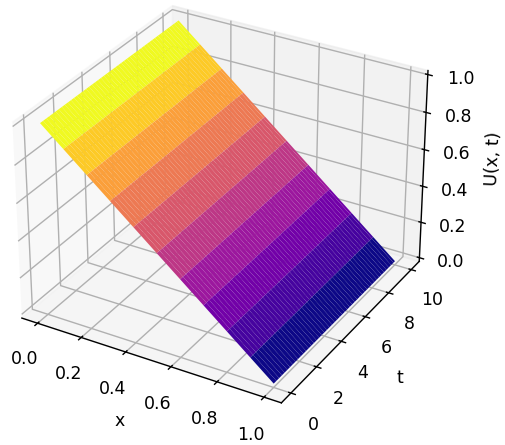
Здесь для определения неизвестных значений на (i + 1)-м слое нужно знать решения на i-м слое. Поэтому начать счет по формулам можно лишь для первого слоя, а решения на нулевом слое должны быть известны. Для этого используем начальные условия

Имеем:

Тогда

Также имеем граничное условие, задающее значение функции в последнем узле:

Получив точки, построим трёхмерный график:

**

**ПРИЛОЖЕНИЕ**

***Программа реализации явного метода***

import matplotlib.pyplot as plt  
import numpy as np  
  
  
def urmat(ux0, ut0, ut1, D, h, tau):  
 x = np.arange(0, 1 + h, h)  
 t = np.arange(0, 10 + tau, tau)  
 u = np.zeros((len(x), len(t)))  
 for i in range(len(x)):  
 u[i][0] = ux0(x[i])  
 for j in range(len(t)):  
 u[0][j] = ut0(t[j])  
 u[-1][j] = ut1(t[j])  
 l = D \* tau / h\*\*2  
 for j in range(0, len(t) - 1):  
 for i in range(1, len(x) - 1):  
 u[i][j + 1] = l \* u[i + 1][j] + (1 - 2 \* l) \* u[i][j] + l \* u[i - 1][j]  
 X, T = np.meshgrid(x, t)  
 fig = plt.figure()  
 ax = fig.add\_subplot(111, projection='3d')  
 ax.plot\_surface(X, T, u.T, cmap='plasma')  
 ax.set\_xlabel('x')  
 ax.set\_ylabel('t')  
 ax.set\_zlabel('U(x, t)')  
 plt.show()  
  
  
ux0 = lambda x: 2 - x - x \* (1 - x)  
ut0 = lambda t: 2  
ut1 = lambda t: 1  
D = 1  
h = 0.1  
tau = 0.001  
urmat(ux0, ut0, ut1, D, h, tau)

***Программа реализации неявного метода***

import matplotlib.pyplot as plt  
import numpy as np  
  
  
def urmat(ux0, ut0, ut1, d, h, tau):  
 x = np.arange(0, 1 + h, h)  
 t = np.arange(0, 10 + tau, tau)  
 u = np.zeros((len(x), len(t)))  
 for i in range(len(x)):  
 u[i][0] = ux0(x[i])  
 for j in range(len(t)):  
 u[0][j] = ut0(t[j])  
 u[-1][j] = ut1(t[j])  
 l = d \* tau / h\*\*2  
 for j in range(0, len(t) - 1):  
 D = [u[0][j + 1]]  
 for i in range(1, len(x) - 1):  
 D.append(-u[i][j])  
 a = [0]  
 b = [2]  
 for i in range(1, len(x) - 1):  
 a.append(-l / (l \* a[i - 1] - (2 \* l + 1)))  
 b.append((D[i] - l \* b[i - 1]) / (l \* a[i - 1] - (2 \* l + 1)))  
 for i in range(len(x) - 2, 0, -1):  
 u[i][j + 1] = a[i] \* u[i + 1][j + 1] + b[i]  
 X, T = np.meshgrid(x, t)  
 fig = plt.figure()  
 ax = fig.add\_subplot(111, projection='3d')  
 ax.plot\_surface(X, T, u.T, cmap='plasma')  
 ax.set\_xlabel('x')  
 ax.set\_ylabel('t')  
 ax.set\_zlabel('U(x, t)')  
 plt.show()  
  
  
ux0 = lambda x: 2 - x - x \* (1 - x)  
ut0 = lambda t: 2  
ut1 = lambda t: 1  
D = 1  
h = 0.1  
tau = 0.01  
urmat(ux0, ut0, ut1, D, h, tau)