ОТЧЁТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ № 8

УРАВНЕНИЕ ГИПЕРБОЛИЧЕСКОГО ТИПА
(Вариант 15)

Выполнил студент 3 курса ПМ
Ушаков Никита

Задача: усвоить сущность и методы решения ***линейного дифференциального уравнения 2-го порядка гиперболического типа***.

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

$x_j \in [a, b]$, $t_i \in [c, d]$

Численное решение таких дифференциальных уравнений возможно методами конечных разностей.

Погрешность решения, найденного этими методами, оценивается величиной $O(\tau^p, h^q)$, где p, q - порядок метода.

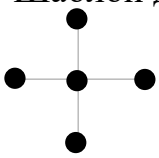
Задание.

Решить волновое уравнение

$$\frac{\partial^2 U}{\partial t^2} = D^2 \frac{\partial^2 U}{\partial x^2} + f(t,x)$$

явным методом и неявными методами второго порядка точности

Шаблон для явного метода:



Шаблон для неявного метода:

Вывести результаты в виде графиков U(x) для разных значений t от 1 до 10 с шагом 1

Неявные методы решать с помощью прогонки.

Для всех вариантов $[a, b] = [0; 1]$, $[c, d] = [0; 10]$, $f(x,t)=0$

Погрешность решения 0,01.

15	$U(0,x) = x, \frac{\partial U}{\partial t}(0,x) = 2$	$U(t,0) = 0, \quad U(t,1) = 1$	1

Решение:

Явный метод:

$$u_i^{j+1} = 2(1-\lambda)\,u_i^j + \lambda\left(u_{i+1}^j + u_{i-1}^j\right) - u_i^{j-1}$$

$$\lambda = D * \frac{\tau^2}{h^2}, i \in [1, I-1], j \in [1, J-1]$$

Условие устойчивости:

$$\sqrt{D}\frac{\tau}{h} < 1$$

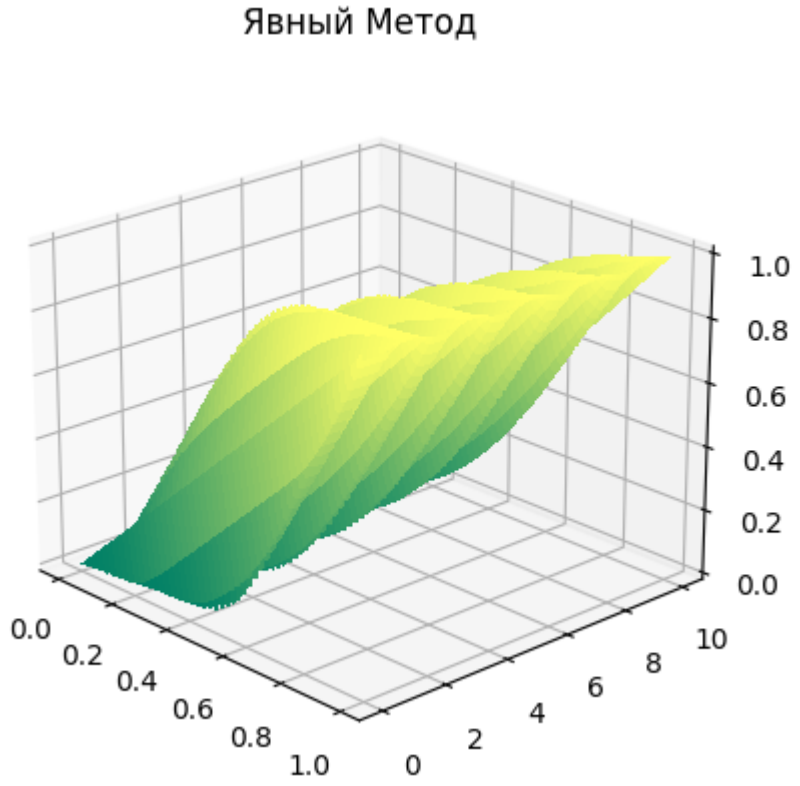
Берём h = 0.1, τ = 0.01

Решение начинается со 2-го слоя:

$$u_i^0 = \varphi(x_i) = 1 - x_i$$

$$u_i^1 = u_i^0 + \tau\psi(x_i) + \frac{D^2\tau^2}{2}\varphi''(x_i) = 1 - x_i - 1\cdot\tau + \frac{1\cdot\tau^2}{2}\cdot 0 = 1 - x_i - \tau$$

Результат:



Первый неявный метод:

$$\lambda u_{i-1}^{j+1} - (1+2\lambda)\, u_i^{j+1} + \lambda u_{i+1}^{j+1} = (1+2\lambda)\, u_i^{j-1} - \lambda\left(u_{i+1}^{j-1} + u_{i-1}^{j-1}\right) - 2u_i^j$$

$$\lambda = D\frac{\tau^2}{2h^2}, i \in [1, I-1]$$

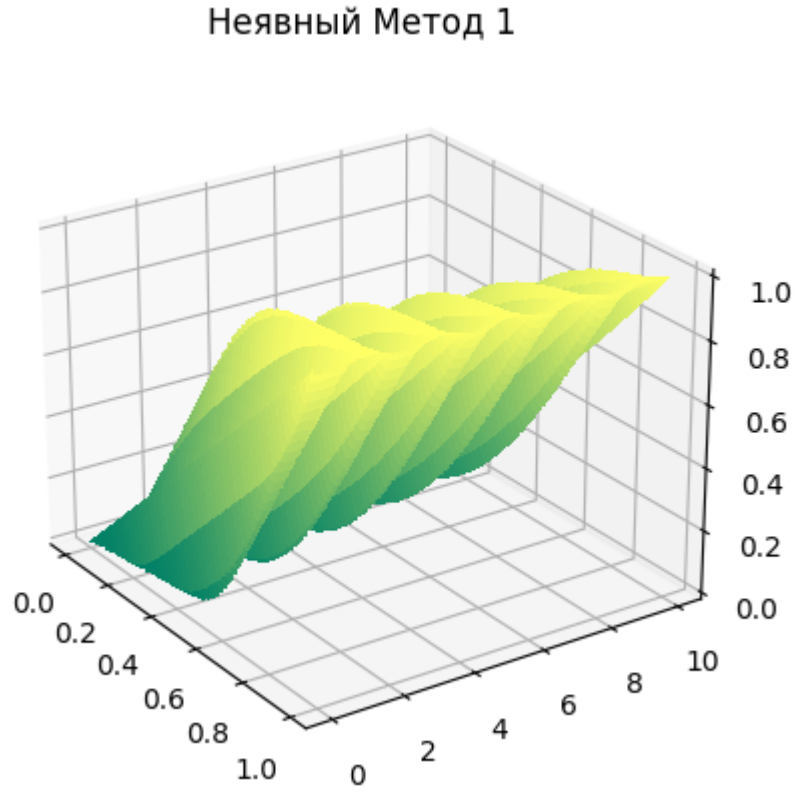
Решается методом прогонки:

$$A*u[i\text{-}1] - B*u[i] + A*u[i+1] = F[i]$$

$$u_{i+1}^{j+1} = A[i]\cdot u_i^j + B[i]$$

$$i \in [1, I], j \in [1, J\text{-}1]$$

Результат:



$$\lambda u_{i-1}^{j+1}$$

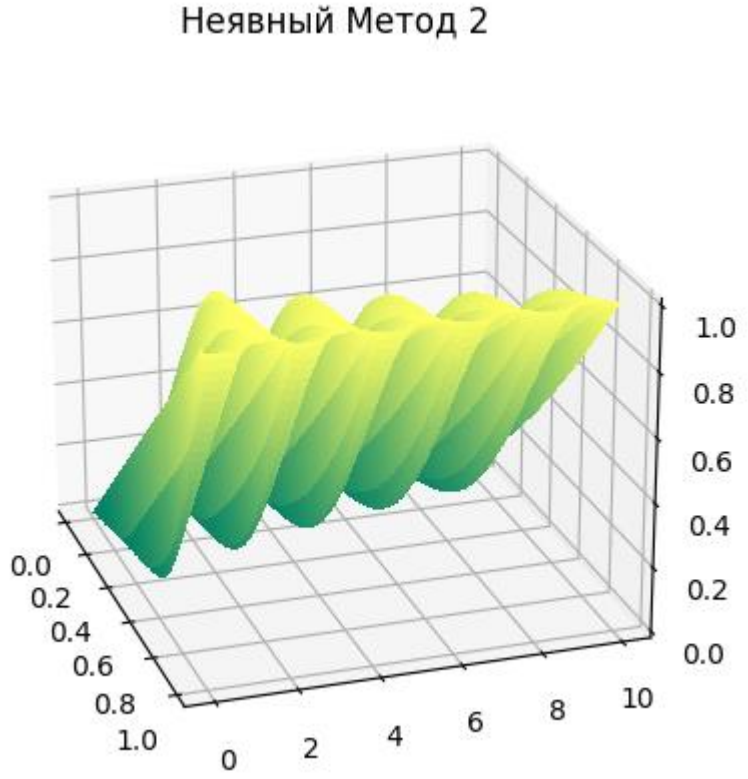
Второй неявный метод:

$$\lambda u_{i-1}^{j+1} - (1 + 2\lambda)u_i^{j+1} + \lambda u_i^{j-1} = (1 + 2\lambda)u_i^{j-1} - \lambda(u_{i+1}^{j-1} + u_{i-1}^{j-1}) - 2u_i^j$$

Решается методом прогонки:

$$Au[i - 1] - Bu[i] + Au[i + 1] = F[i]$$
$$u_{i+1}^{j+1} = A[i]u_i^j + B[i],\ i \in [1, I],\ j \in [1, J - 1]$$

Результат:



ПРИЛОЖЕНИЕ

```
import numpy as np
from matplotlib import cm
import matplotlib.pyplot as plt


1 usage  new *
class Solver:

    new *
    def __init__(self, a, b, c, d, eps, h, tau, U0_x, der_U_t, Ut_0, Ut_1):
        self.a = a
        self.b = b
        self.c = c
        self.d = d
        self.eps = eps
        self.h = h
        self.tau = tau
        self.x_count = int((b - a) / h)
        self.t_count = int((d - c) / tau)
        self.l_ambda = (D * tau / h)
        self.A = self.l_ambda
        self.B = 2 * self.l_ambda + 1
        self.U0_x = U0_x
        self.der_U_t = der_U_t
        self.Ut_0 = Ut_0
        self.Ut_1 = Ut_1
```

```
def explicit(self):
    u = np.zeros((self.t_count, self.x_count))
    for i in range(self.x_count):
        u[0][i] = self.U0_x(0, i * h)
        u[1][i] = u[0][i] + (-1) * tau
    for i in range(self.t_count):
        u[i][0] = self.Ut_0(i * tau, 0)
        u[i][self.x_count - 1] = self.Ut_1(i * tau, 1)
    for j in range(1, self.t_count - 1):
        for i in range(1, self.x_count - 1):
            u[j + 1][i] = 2 * (1 - self.l_ambda) * u[j][i] + self.l_ambda * (u[j][i + 1] +
             u[j][i - 1] - u[j - 1][i])
    return u

1 usage  new *
def not_explicit_1(self):
    u = np.zeros((self.t_count, self.x_count))
    for i in range(self.x_count):
        u[0][i] = self.U0_x(0, i * h)
        u[1][i] = u[0][i] + (-1) * tau
    for i in range(self.t_count):
        u[i][0] = self.Ut_0(i * tau, 0)
        u[i][self.x_count - 1] = self.Ut_1(i * tau, 1)
    a, b, c = np.zeros(self.x_count), np.zeros(self.x_count), np.zeros(self.x_count)
    for j in range(1, self.t_count - 1):
        a[self.x_count - 1] = 0
        b[self.x_count - 1] = u[j + 1][self.x_count - 1]
        c[self.x_count - 1] = 1 / (self.B - self.A * a[self.x_count - 1])
        for i in range(self.x_count - 1, 0, -1):
            a[i - 1] = c[i] * self.A
            b[i - 1] = c[i] * (self.A * b[i] - (u[j - 1][i] - 2 * u[j][i]))
            c[i - 1] = 1 / (self.B - self.A * a[i - 1])
        for i in range(1, self.x_count - 1):
            u[j + 1][i + 1] = a[i] * u[j + 1][i] + b[i]
    return u
```

```python
def not_explicit_2(self):
    u = np.zeros((self.t_count, self.x_count))
    for i in range(self.x_count):
        u[0][i] = self.U0_x(0, i * h)
        u[1][i] = u[0][i] + (-1) * tau
    for i in range(self.t_count):
        u[i][0] = self.Ut_0(i * tau, 0)
        u[i][self.x_count - 1] = self.Ut_1(i * tau, 1)
    a, b, c = np.zeros(self.x_count), np.zeros(self.x_count), np.zeros(self.x_count)
    for j in range(1, self.t_count - 1):
        a[self.x_count - 1] = 0
        b[self.x_count - 1] = u[j + 1][self.x_count - 1]
        c[self.x_count - 1] = 1 / (self.B - self.A * a[self.x_count - 1])
        for i in range(self.x_count - 1, 0, -1):
            a[i - 1] = c[i] * self.A
            b[i - 1] = c[i] * (self.A * b[i] - (u[j - 1][i] - 2 * u[j][i]))
            c[i - 1] = 1 / (self.B - self.A * a[i - 1])
        for i in range(1, self.x_count - 1):
            u[j + 1][i + 1] = a[i] * u[j + 1][i] + b[i]
    return u

3 usages  new *
def draw(self, u, method):
    x = np.arange(a, b, h)
    t = np.arange(c, d, tau)
    x, t = np.meshgrid( *xi: x, t)
    fig = plt.figure()
    ax = fig.add_subplot(projection='3d')
    ax.set_title(method.title())
    ax.plot_surface(x, t, u, cmap=cm.summer, antialiased=False)
    plt.show()
```

```python
D = 1
a, b = [0, 1]
c, d = [0, 10]
eps, h, tau = 0.01, 0.01, 0.01
U0_x = lambda t, x: x
der_U_t = lambda x: 2
Ut_0 = lambda t, x: 0
Ut_1 = lambda t, x: 1

solver = Solver(a, b, c, d, eps, h, tau, U0_x, der_U_t, Ut_0, Ut_1)
u1 = solver.explicit()
u2 = solver.not_explicit_1()
u3 = solver.not_explicit_2()

solver.draw(u1, method: "Явный метод")
solver.draw(u2, method: "Неявный метод 1")
solver.draw(u3, method: "Неявный метод 2")
```