Лабораторная работа №6 учебного года 2023-2024 по курсу «Численные методы»

Выполнил: Борисов Я. А
Группа: М8О-408Б-20
Преподаватель: Пивоваров Д.Е.
Вариант по списку группы: 4

Условие лабораторной 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 $U(x, t)$. Исследовать зависимость погрешности от сеточных параметров τ, h .

Вариант 4

$\frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2} - 5u$,

$u_x(0,t) - 2u(0,t) = 0,$

$u_x(1,t) - 2u(1,t) = 0,$

$u(x,0) = \exp(2x)$,

$u_t(x,0) = 0$.

Аналитическое решение: $U(x,t) = \exp(2x)\cos t$

Программа

```
import numpy as np

from functions import tma

class Data:
    def __init__(self, params):
        self.a = params['a']
        self.b = params['b']
        self.c = params['c']
        self.d = params['d']
        self.l = params['l']
```

```
        self.f = params['f']
        self.alpha = params['alpha']
        self.beta = params['beta']
        self.gamma = params['gamma']
        self.delta = params['delta']
        self.psi1 = params['psi1']
        self.psi2 = params['psi2']
        self.psi1_dir1 = params['psi1_dir1']
        self.psi1_dir2 = params['psi1_dir2']
        self.phi0 = params['phi0']
        self.phi1 = params['phi1']
        self.bound_type = params['bound_type']
        self.approximation = params['approximation']
        self.solution = params['solution']

class HyperbolicSolver:
    def __init__(self, params, equation_type):
        self.data = Data(params)
        self.h = 0
        self.tau = 0
        self.sigma = 0
        try:
            self.solve_func = getattr(self, f'{equation_type}_solver')
        except:
              raise Exception("Такого типа не существует!")

    def solve(self, N, K, T):
        self.h = self.data.l / N
        self.tau = T / K
        self.sigma = (self.tau ** 2) / (self.h ** 2)
        return self.solve_func(N, K, T)

    def analyticSolve(self, N, K, T):
        self.h = self.data.l / N
        self.tau = T / K
        self.sigma = (self.tau ** 2) / (self.h ** 2)
        u = np.zeros((K, N))
        for k in range(K):
            for j in range(N):
                u[k][j] = self.data.solution(j * self.h, k * self.tau)
        return u

    def calculate(self, N, K):
        u = np.zeros((K, N))

        for j in range(0, N - 1):
            x = j * self.h
            u[0][j] = self.data.psi1(x)
```

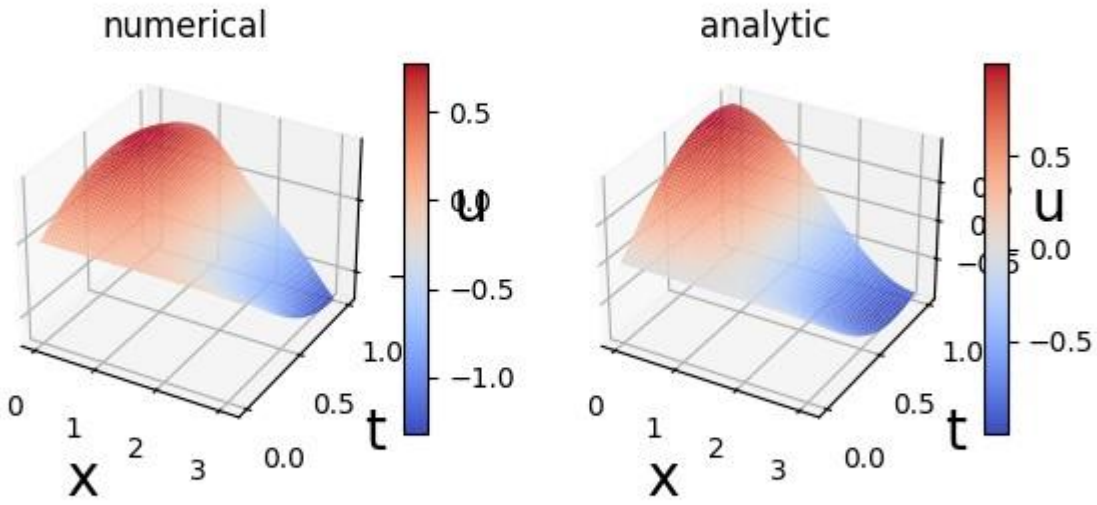
```
            if self.data.approximation == 'p1':
                u[1][j] = self.data.psi1(x) + self.data.psi2(x) * self.tau +
self.data.psi1_dir2(x) * (self.tau ** 2 / 2)
            elif self.data.approximation == 'p2':
                u[1][j] = self.data.psi1(x) + self.data.psi2(x) * self.tau + \
                          (self.data.psi1_dir2(x) + self.data.b * self.data.psi1_dir1(x) +
                           self.data.c * self.data.psi1(x) + self.data.f()) * (self.tau ** 2 /
2)

        return u

    def implicit_solver(self, N, K, T):
        u = self.calculate(N, K)

        a = np.zeros(N)
        b = np.zeros(N)
        c = np.zeros(N)
        d = np.zeros(N)

        for k in range(2, K):
            for j in range(1, N):
                a[j] = self.sigma
                b[j] = -(1 + 2 * self.sigma)
                c[j] = self.sigma
                d[j] = -2 * u[k - 1][j] + u[k - 2][j]

            if self.data.bound_type == 'a1p2':
                b[0] = self.data.alpha / self.h / (self.data.beta - self.data.alpha / self.h)
                c[0] = 1
                d[0] = 1 / (self.data.beta - self.data.alpha / self.h) * self.data.phi0(k *
self.tau)
                a[-1] = -self.data.gamma / self.h / (self.data.delta + self.data.gamma /
self.h)
                d[-1] = 1 / (self.data.delta + self.data.gamma / self.h) * self.data.phi1(k *
self.tau)

            elif self.data.bound_type == 'a2p3':
                k1 = 2 * self.h * self.data.beta - 3 * self.data.alpha
                omega = self.tau ** 2 * self.data.b / (2 * self.h)
                xi = self.data.d * self.tau / 2

                b[0] = 4 * self.data.alpha - self.data.alpha / (self.sigma + omega) * \
                       (1 + xi + 2 * self.sigma - self.data.c * self.tau ** 2)
                c[0] = k1 - self.data.alpha * (omega - self.sigma) / (omega + self.sigma)
                d[0] = 2 * self.h * self.data.phi0(k * self.tau) + self.data.alpha * d[1] / (-
self.sigma - omega)
                a[-1] = -self.data.gamma / (omega - self.sigma) * \
                        (1 + xi + 2 * self.sigma - self.data.c * self.tau ** 2) - 4 *
self.data.gamma
```

```
                d[-1] = 2 * self.h * self.data.phi1(k * self.tau) - self.data.gamma * d[-2] /
(omega - self.sigma)

            elif self.data.bound_type == 'a2p2':
                b[0] = 2 * self.data.a / self.h
                c[0] = -2 * self.data.a / self.h + self.h / self.tau ** 2 - self.data.c *
self.h + \
                       -self.data.d * self.h / (2 * self.tau) + \
                       self.data.beta / self.data.alpha * (2 * self.data.a + self.data.b *
self.h)
                d[0] = self.h / self.tau ** 2 * (u[k - 2][0] - 2 * u[k - 1][0]) - self.h *
self.data.f() + \
                       -self.data.d * self.h / (2 * self.tau) * u[k - 2][0] + \
                       (2 * self.data.a - self.data.b * self.h) / self.data.alpha *
self.data.phi0(k * self.tau)
                a[-1] = -b[0]
                d[-1] = self.h / self.tau ** 2 * (-u[k - 2][0] + 2 * u[k - 1][0]) + self.h *
self.data.f() + \
                        self.data.d * self.h / (2 * self.tau) * u[k - 2][0] + \
                        (2 * self.data.a + self.data.b * self.h) / self.data.alpha *
self.data.phi1(k * self.tau)

            u[k] = tma(a, b, c, d)

        return u

    def _left_bound_a1p2(self, u, k, t):
        coeff = self.data.alpha / self.h
        return (-coeff * u[k - 1][1] + self.data.phi0(t)) / (self.data.beta - coeff)

    def _right_bound_a1p2(self, u, k, t):
        coeff = self.data.gamma / self.h
        return (coeff * u[k - 1][-2] + self.data.phi1(t)) / (self.data.delta + coeff)

    def _left_bound_a2p2(self, u, k, t):
        n = self.data.c * self.h - 2 * self.data.a / self.h - self.h / self.tau ** 2 -
self.data.d * self.h / \
            (2 * self.tau) + self.data.beta / self.data.alpha * (2 * self.data.a - self.data.b
* self.h)
        return 1 / n * (- 2 * self.data.a / self.h * u[k][1] +
                        self.h / self.tau ** 2 * (u[k - 2][0] - 2 * u[k - 1][0]) +
                        -self.data.d * self.h / (2 * self.tau) * u[k - 2][0] + -self.h *
self.data.f() +
                        (2 * self.data.a - self.data.b * self.h) / self.data.alpha *
self.data.phi0(t))

    def _left_bound_a2p3(self, u, k, t):
        denom = 2 * self.h * self.data.beta - 3 * self.data.alpha
```

```
        return self.data.alpha / denom * u[k - 1][2] - 4 * self.data.alpha / denom * u[k -
1][1] + \
               2 * self.h / denom * self.data.phi0(t)

    def _right_bound_a2p2(self, u, k, t):
        n = -self.data.c * self.h + 2 * self.data.a / self.h + self.h / self.tau ** 2 +
self.data.d * self.h / \
            (2 * self.tau) + self.data.delta / self.data.gamma * (2 * self.data.a +
self.data.b * self.h)
        return 1 / n * (2 * self.data.a / self.h * u[k][-2] +
                        self.h / self.tau ** 2 * (2 * u[k - 1][-1] - u[k - 2][-1]) +
                        self.data.d * self.h / (2 * self.tau) * u[k - 2][-1] + self.h *
self.data.f() +
                        (2 * self.data.a + self.data.b * self.h) / self.data.gamma *
self.data.phi1(t))

    def _right_bound_a2p3(self, u, k, t):
        denom = 2 * self.h * self.data.delta + 3 * self.data.gamma
        return 4 * self.data.gamma / denom * u[k - 1][-2] - self.data.gamma / denom * u[k -
1][-3] + \
               2 * self.h / denom * self.data.phi1(t)

    def explicit_solver(self, N, K):
        global left_bound, right_bound
        u = self.calculate(N, K)

        if self.data.bound_type == 'a1p2':
            left_bound = self._left_bound_a1p2
            right_bound = self._right_bound_a1p2

        elif self.data.bound_type == 'a2p2':
            left_bound = self._left_bound_a2p2
            right_bound = self._right_bound_a2p2

        elif self.data.bound_type == 'a2p3':
            left_bound = self._left_bound_a2p3
            right_bound = self._right_bound_a2p3

        for k in range(2, K):
            t = k * self.tau
            for j in range(1, N - 1):
                quadr = self.tau ** 2
                tmp1 = self.sigma + self.data.b * quadr / (2 * self.h)
                tmp2 = self.sigma - self.data.b * quadr / (2 * self.h)
                u[k][j] = u[k - 1][j + 1] * tmp1 + \
                    u[k - 1][j] * (-2 * self.sigma + 2 + self.data.c * quadr) + \
                    u[k - 1][j - 1] * tmp2 - u[k - 2][j] + quadr * self.data.f()

            u[k][0] = left_bound(u, k, t)
```

```
            u[k][-1] = right_bound(u, k, t)

        return u
```

Результаты:



Вывод по лабораторной работе

Благодаря данной лабораторной работе, я приобрел знания в области численных методов для решения дифференциальных уравнений гиперболического типа: были исследованы различные методы решения начально-краевой задачи для дифференциального уравнения гиперболического типа, а также была оценена точность и эффективность каждого метода, построен график зависимости ошибки от времени и график U(x).