Московский Авиационный Институт

(национальный исследовательский университет)

Институт №8 “Информационные технологии и прикладная математика”

Кафедра 806 “ Вычислительная математика и программирование”

Лабораторная работа № 6 по курсу “Численные методы”

на тему

“ Численные методы решения ДУЧП гиперболического типа”

Студент: Четвергов А. О.

Группа: М8О-409Б-19

Вариант: 3

Преподаватель: Пивоваров Д. Е.

Москва

2022

1. Задание

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.

2. Вариант

3.

$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2} - 3u,$$

$$u(0,t) = \sin(2t)),$$

$$u(\pi,t) = -\sin(2t),$$

$$u(x,0) = 0,$$

$$u_t(x,0) = 2\cos x.$$

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

3. Теория:

Третья начально-краевая задача для волнового уравнения:

$$\begin{cases} \dfrac{\partial^2 u}{\partial t^2} = a^2 \dfrac{\partial^2 u}{\partial x^2}, & 0 < x < l, \quad t > 0; \\ \alpha \dfrac{\partial u(0,t)}{\partial x} + \beta u(0,t) = \varphi_0(t), & x = 0, \quad t > 0; \\ \gamma \dfrac{\partial u(l,t)}{\partial x} + \delta u(l,t) = \varphi_l(t), & x = l, \quad t > 0; \\ u(x,0) = \psi_1(x), & 0 \le x \le l, \quad t = 0; \\ \dfrac{\partial u(x,0)}{\partial t} = \psi_2(x), & 0 \le x \le l, \quad t = 0. \end{cases}$$

На пространственно-временной сетке

$$\omega_{h\tau} = \{x_j = jh, \ \ j = \overline{0,N}; \ \ t^k = k\tau, \ \ k = \overline{0,K}\}$$

Будем аппроксимировать дифференциальное уравнение одной из следующих конечно-разностных схем:

$$\frac{u_j^{k+1}-2u_j^k+u_j^{k-1}}{\tau^2}=a^2\frac{u_{j+1}^k-2u_j^k+u_{j-1}^k}{h^2}+O\left(\tau^2+h^2\right),\quad j=\overline{1,N-1};\quad k=1,2,\ldots \qquad (5.38)$$

И

$$\frac{u_j^{k+1}-2u_j^k+u_j^{k-1}}{\tau^2}=a^2\frac{u_{j+1}^{k+1}-2u_j^{k+1}+u_{j-1}^{k+1}}{h^2}+O\left(\tau+h^2\right),\quad j=\overline{1,N-1};\quad k=1,2,\ldots \qquad (5.39)$$

При этом схема (5.38) является явной. С её помощью решение $u_j^{k+1}, j=\overline{1,N-1}, k=1,2,\ldots$ определяется сразу, поскольку значения сеточных функций u_j^{k-1}, u_j^k на нижних временных слоях должны быть известны. Порядок аппроксимации равен двум, как по пространственной, так и по временной переменной. При этом явная конечно-разностная схема (5.38) для волнового уравнения условно устойчива с условием $\sigma=\frac{a^2\tau^2}{h^2}<1$, накладываемым на сеточные характеристики τ и h.

Схема (5.39) является неявной схемой и обладает абсолютной устойчивостью. Ее можно свести к СЛАУ с трехдиагональной матрицей, решаемой методом прогонки.

В обеих схемах необходимо знать значения $u_j^{k-1}, u_j^k,\ j=\overline{1,N-1}, k=1,2,\ldots$ на нижних временных слоях. Для k=1 это делается следующим образом:

$$u_j^0=\psi_1(x_j),\ j=\overline{0,N},$$

Где $\psi_1(x)$ – функция из начального условия.

Для определения u_j^1 можно воспользоваться простейшей аппроксимацией второго начального условия:

$$\frac{u_j^1-u_j^0}{\tau}=\psi_2(x_j).$$

Откуда для искомых значений u_j^1 получаем следующее выражение:

$$u_j^1=\psi_1(x_j)+\psi_2(x_j)\tau.$$

Сеточная функция u_j^1 со вторым порядком точности:

$$u_j^1=\psi_1(x_j)+\psi_2(x_j)\tau+a^2\psi_1''(x_j)\frac{\tau^2}{2}.$$

4. **Программа:**

Начальные данные:

```
a = 1
c = -3
l = np.pi
N = 10
K = 1000
T = 1
h = l/N
tau = T/K
sgm = h**2 / tau**2
```

Сетка:

```
tk = np.zeros(K)
xn = np.zeros(N)

for i in range(N):
    xn[i] = h * i
for i in range(K):
    tk[i] = tau * i
```

Дано:

```
def phi0(t):
    return np.sin(2*t)

def phi1(t):
    return np.sin(2*t) * (-1)

def psi1():
    return 0

def psi2(x):
    return 2 * np.cos(x)

def exact(x,t):
    return np.cos(x) * np.sin(2*t)
```

Явная:

```
def explicit():

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

    for k in range(0,K):
        u[k][0] = phi0(tau * k)
        u[k][-1] = phi1(tau * k)

    for j in range(0,N):
        u[0][j] = psi1()
        u[1][j] = psi2(j * h) * tau

    for k in range(1,K-1):
        for j in range(1,N-1):
            u[k+1][j] = u[k][j + 1] * (a ** 2 * tau ** 2) / h ** 2 + u[k][j]
* (-2 * (a ** 2 * tau ** 2) / h ** 2 + 2 + c * tau ** 2) + u[k][j - 1] * (a
** 2 * tau ** 2) / h ** 2 - u[k - 1][j]

        u[k][0] = phi0(k * tau)
        u[k][-1] = phi1(k * tau)

    return u
```

Неявная:

```
def implicit():

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

    for k in range(0,K):
        u[k][0] = phi0(tau * k)
        u[k][-1] = phi1(tau * k)

    for j in range(0,N):
        u[0][j] = psi1()
        u[1][j] = psi2(j * h) * tau

    ap = np.zeros(N)
    bp = np.zeros(N)
    cp = np.zeros(N)
    dp = np.zeros(N)

    for k in range(2,K):

        ap[0] = 0
        bp[0] = 1
        cp[0] = 0
        dp[0] = phi0(tau * k)

        for j in range(1,N-1):
            ap[j] = 2 * a
            bp[j] = -2 * sgm + 2*(h**2)*c - 4 * a
            cp[j] = 2 * a
            dp[j] = -4 * sgm * u[k-1][j] + (2 * sgm * u[k-2][j])

        ap[-1] = 0
        bp[-1] = 1
        cp[-1] = 0
        dp[-1] = phi1(tau * k)

        u[k] = swp(ap,bp,cp,dp)

    return u
```

Погрешность:

```
def accuracy_error(tk,xn,u):
    res = np.zeros(K)
    for i in range(K):
        res[i] = np.max(np.abs(u[i] - np.array([exact(x,tk[i]) for x in
xn])))

    plt.figure(figsize=(16, 8))
    plt.plot(tk[1:], res[1:])
    plt.xlabel('t')
    plt.ylabel('error')
    plt.grid(True)
    plt.show()
```

Погрешность в различные моменты времени:

```
def result(ti, xi, u1, u2):
    fig,ax = plt.subplots(4)
    fig.set_figheight(10)
    fig.set_figwidth(15)
    t = 0
    for i in range(4):
        ax[i].plot(xi, u1[t, :], label='explicit')
        ax[i].plot(xi, u2[t, :], label='implicit')
```

```
        ax[i].plot(xi, [exact(x, ti[t]) for x in xi], label='Analytic')
        ax[i].grid(True)
        ax[i].set_xlabel('x')
        ax[i].set_ylabel('u')
        ax[i].set_title(f'Решения при t = {t / K}')
        t = K - (i+2)*100

    plt.legend()
    plt.show()
```

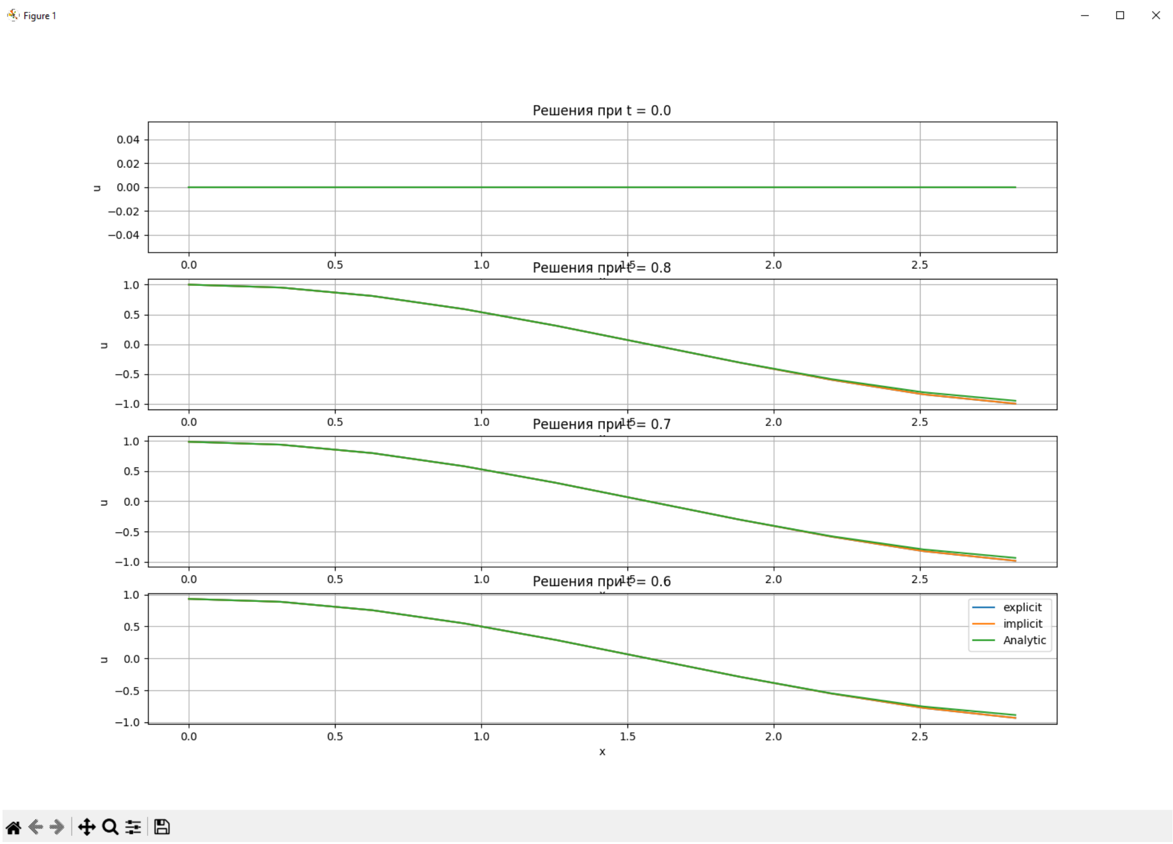
Метод прогонки:

```
def swp(a,b,c,d):

    i = np.shape(d)[0]

    def searchP():
        P = np.zeros(i)
        P[0] = -c[0] / b[0]
        for j in range(1, len(P)):
            P[j] = -c[j] / (b[j] + a[j] * P[j - 1])
        return P

    def searchQ():
        Q = np.zeros(i)
        Q[0] = d[0] / b[0]
        for j in range(1, len(Q)):
            Q[j] = (d[j] - a[j] * Q[j - 1]) / (b[j] + a[j] * P[j - 1])
        return Q

    def searchX():
        X = np.zeros(i)
        X[i - 1] = Q[i - 1]
        for j in range(len(X) - 2, -1, -1):
            X[j] = P[j] * X[j + 1] + Q[j]
        return X

    P = searchP()
    Q = searchQ()
    X = searchX()

    return X
```

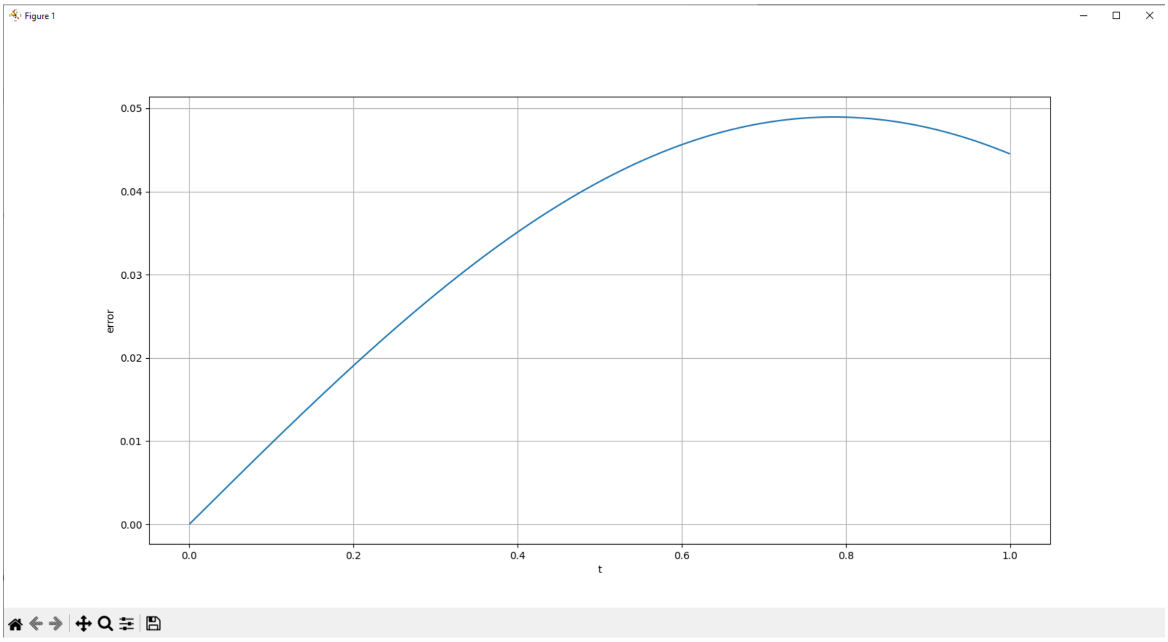
Main:

```
u1 = explicit()
u2 = implicit()

result(tk,xn,u1,u2)

accuracy_error(tk,xn,u1)
accuracy_error(tk,xn,u2)
```

5. Результат:



Явная схема:



Неявная схема: