Лабораторная работа №6

Начально-краевые задачи для дифференциального уравнения гиперболического типа

Сорокин Никита, М8О-403Б-20

Задание

Используя явную схему крест и неявную схему, решить начально-краевую задачу для дифференциального уравнения гиперболического типа. 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 $U(x, t)$.

Вариант 2:

$$\begin{cases} \frac{\partial^2 u}{\partial t^2} = a^2 \frac{\partial^2 u}{\partial x^2}, \;\; a^2 > 0 \\ u_x(0, t) - u(0, t) = 0 \\ u_x(\pi, t) - u(\pi, t) = 0 \\ u(x, 0) = sin(x) + cos(x) \\ u_t(x, 0) = -a(sin(x) + cos(x)) \end{cases}$$

Аналитическое решение:

$$U(x, t) = sin(x - at) + cos(x + at)$$

In [1]:
```
import sys
sys.path

sys.path.insert(0, r"c:\Users\никита\Desktop\учеба\чм\modules")
```

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import time
sns.set()

%matplotlib inline

from LinearAlgebra import *
```

In [3]:
```
a = 1

x_begin = 0
```

```
x_end = np.pi

t_begin = 0
t_end = 1

h = 0.01
tau = h / (2 * a)
sigma = a**2 * tau**2 / h**2
sigma
```

Out[3]: 0.25

In [4]:
```
def check_sigma(sigma):
    res = True if sigma <= 1 else False
    return res

check_sigma(sigma)
```

Out[4]: True

Функция для вычисления погрешности - максимального модуля ошибки

In [5]:
```
def max_abs_error(A, B):
    assert A.shape == B.shape
    return abs(A - B).max()
```

И среднего модуля ошибки:

In [6]:
```
def mean_abs_error(A, B):
    assert A.shape == B.shape
    return abs(A - B).mean()
```

In [7]:
```
def solution(x, t, a=1):
    u = np.sin(x - a * t) + np.cos(x + a * t)
    return u

def phi_0(t, a=1):
    return 0

def phi_1(t, a=1):
    return 1

def psi_1(x, a=1):
    return np.sin(x) + np.cos(x)

def psi_2(x, a=1):
    return -a * (np.sin(x) + np.cos(x))

def dd_psi_1(x, a=1):
    return - (np.sin(x) + np.cos(x))
```

Начальные условия

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

1 порядок

$$\frac{u_j^1 - u_j^0}{\tau} = \psi_2(x_j),$$
$$u_j^1 = \psi_1(x_j) + \psi_2(x_j)\tau, \; j = \overline{0, N}$$

2 порядок

$$u_j^1 = u(x_j, 0 + \tau) = u_j^0 + \frac{\partial u}{\partial t}\Big|_j^0 \tau + \frac{\partial^2 u}{\partial t^2}\Big|_j^0 \frac{\tau^2}{2} + O(\tau^3),$$
$$\frac{\partial^2 u}{\partial t^2}\Big|_j^0 = a^2 \frac{\partial^2 u}{\partial x^2}\Big|_j^0 = a^2 \psi_1''(x_j),$$
$$u_j^1 = \psi_1(x_j) + \psi_2(x_j)\tau + \frac{a^2\tau^2}{2}\psi_1''(x_j)$$

In [8]:
```
def get_initial_values(x_begin, x_end, h, tau, psi_1, psi_2, dd_psi_2, order):

    x = np.arange(x_begin, x_end + h, h)
    u = np.zeros((2, x.size))

    u[0, :] = psi_1(x)

    if order == 1:
        u[1, :] = psi_1(x) + psi_2(x) * tau

    if order == 2:
        u[1, :] = psi_1(x) + psi_2(x) * tau + a**2 * tau**2 * dd_psi_1(x) / 2

    return u
```

Граничные условия

2 точки, 1 порядок

$$\phi_0(t^{k+1}) = \frac{u_1^{k+1} - u_0^{k+1}}{h} - u_0^{k+1} = 0$$
$$u_0^{k+1} = \frac{u_1^{k+1}}{1+h}$$

Аналогично получается:

$$u_N^{k+1} = \frac{u_{N-1}^{k+1}}{1-h}$$

3 точки, 2 порядок

$$\phi_0(t^{k+1}) = \frac{-3u_0^{k+1} + 4u_1^{k+1} - u_2^{k+1}}{h} - u_0^{k+1} = 0$$

$$u_0^{k+1} = \frac{-4u_1^{k+1} + u_2^{k+1}}{-3 - 2h}$$

Аналогично получается:

$$u_N^{k+1} = \frac{-u_{N-2}^{k+1} + 4u_{N-1}^{k+1}}{3 - 2h}$$

2 точки, 2 порядок

$$u_1^{k+1} = u(0 + h, t^{k+1}) = u_0^{k+1} + \frac{\partial u}{\partial x}\Big|_0^{k+1} h + \frac{\partial^2 u}{\partial x^2}\Big|_0^{k+1} \frac{h^2}{2} + O(h^3)$$

$$\frac{\partial^2 u}{\partial x^2}\Big|_{0,\,N}^{k+1} = \frac{1}{a^2}\frac{\partial^2 u}{\partial t^2}\Big|_{0,\,N}^{k+1}$$

$$\frac{\partial u}{\partial x}\Big|_0^{k+1} = \frac{u_1^{k+1} - u_0^{k+1}}{h} - \frac{h}{2a^2}\frac{u_0^{k+1} - 2u_0^k + u_0^{k-1}}{\tau^2}$$

$$u_0^{k+1} = \frac{u_1^{k+1} + \frac{1}{\sigma}u_0^k - \frac{1}{2\sigma}u_0^{k-1}}{1 + h + 1/2\sigma}$$

Аналогично получается:

$$u_N^{k+1} = \frac{u_{N-1}^{k+1} + \frac{1}{\sigma}u_N^k - \frac{1}{2\sigma}u_N^{k-1}}{1 - h + 1/2\sigma}$$

In [10]:
```
def get_boundary_values(u, h, sigma, type):

    if type == (2, 1):
        u_0 = u[-1, 1] / (1 + h)
        u_N = u[-1, -2] / (1 - h)

    if type == (3, 2):
        u_0 = (-4 * u[-1, 1] + u[-1, 2]) / (-3 - 2 * h)
        u_N = (-u[-1, -3] + 4 * u[-1, -2]) / (3 - 2 * h)

    if type == (2, 2):
        u_0 = (u[-1, 1] + u[-2, 0] / sigma - u[-3, 0] / (2 * sigma)) / (1 + h + 1 /
        u_N = (u[-1, -2] + u[-2, -1] / sigma - u[-3, -1] / (2 * sigma)) / (1 - h +

    return u_0, u_N
```

In [11]:
```
def get_boundary_coefficients(u, h, sigma, type):

    A = np.zeros((2, 2))
```

```python
    b = np.zeros(2)

    if type == (2, 1):

        A[0, 0] = (1 + 2 * sigma - sigma / (1 + h))
        A[0, 1] = -sigma
        A[-1, -2] = -sigma
        A[-1, -1] = (1 + 2 * sigma - sigma / (1 - h))
        b[0] = 2 * u[-1, 1] - u[-2, 1]
        b[-1] = 2 * u[-1, -2] - u[-2, -2]

    if type == (3, 2):

        A[0, 0] = (-3 - 2 * h) * (1 + 2 * sigma) + 4 * sigma
        A[0, 1] = -sigma + (3 + 2 * h) * sigma
        A[-1, -2] = -(3 - 2 * h) * sigma + sigma
        A[-1, -1] = (3 - 2 * h) * (1 + 2 * sigma) - 4 * sigma
        b[0] = (-3 - 2 * h) * (2 * u[-1, 1] - u[-2, 1])
        b[-1] = (3 - 2 * h) * (2 * u[-1, -2] - u[-2, -2])

    if type == (2, 2):

        A[0, 0] = (1 + 2 * sigma) * (1 + h + 1 / (2 * sigma)) - sigma
        A[0, 1] = -sigma * (1 + h + 1 / (2 * sigma))
        A[-1, -2] = -sigma * (1 - h + 1 / (2 * sigma))
        A[-1, -1] = (1 + 2 * sigma) * (1 - h + 1 / (2 * sigma)) - sigma
        b[0] = (1 + h + 1 / (2 * sigma)) * (2 * u[-1, 1] - u[-2, 1]) + u[-1, 0] - u
        b[-1] = (1 - h + 1 / (2 * sigma)) * (2 * u[-1, -2] - u[-2, -2]) + u[-1, -1]


    return A[0, 0], A[0, 1], A[-1, -2], A[-1, -1], b[0], b[-1]
```

Точное решение

Дано по условию:

In [12]:
```python
def get_analytical_solution(x_begin, x_end, t_begin, t_end, h, tau, a):

    x = np.arange(x_begin, x_end + h, h)
    t = np.arange(t_begin, t_end + tau, tau)

    u = np.zeros((t.size, x.size))
    for j in range(x.size):
        for k in range(t.size):
            u[k, j] = solution(x[j], t[k], a)

    return u
```

In [13]:
```python
u_exact = get_analytical_solution(x_begin, x_end, t_begin, t_end, h, tau, a)
```

Явная схема

$$\frac{u_j^{k+1} - 2u_j^k + u_j^{k-1}}{\tau^2} = a^2 \frac{u_{j+1}^k - 2u_j^k + u_{j-1}^k}{h^2} + O(h^2 + \tau^2)$$

Выразим u_j^{k+1} и получим:

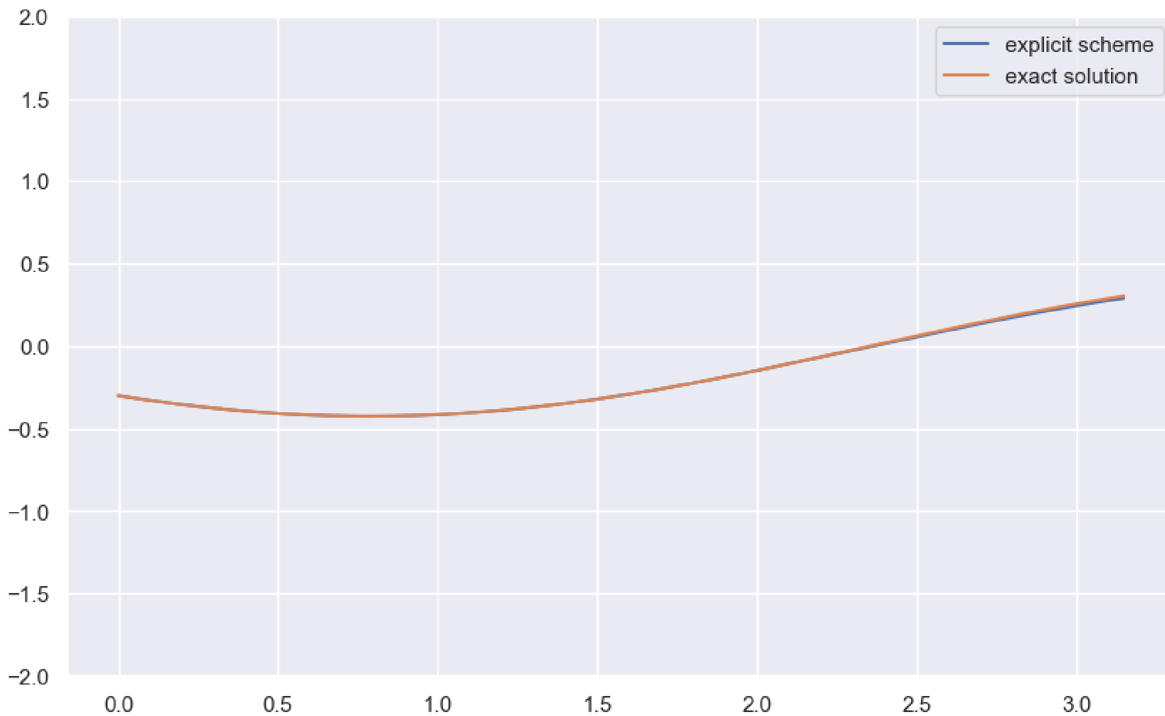
$$u_j^{k+1} = \sigma u_{j-1}^k + 2(1-\sigma)u_j^k + \sigma u_{j+1}^k - u_j^{k-1}$$

где $\sigma = \frac{a\tau^2}{h^2}$

In [14]:
```
def explicit_scheme(x_begin, x_end, t_begin, t_end, h, tau, a, psi_1, psi_2, dd_psi

    sigma = a**2 * tau**2 / h**2
    x = np.arange(x_begin, x_end + h, h)
    t = np.arange(t_begin, t_end + tau, tau)
    u = np.zeros((t.size, x.size))

    u[0:2, :] = get_initial_values(x_begin, x_end, h, tau, psi_1, psi_2, dd_psi_2,

    for k in range(1, t.size - 1):

        u[k + 1, 1:-1] = sigma * u[k, 0:-2] + 2 * (1 - sigma) * u[k, 1:-1] + sigma
        u[k + 1, 0], u[k + 1, -1] = get_boundary_values(u[k - 1:k + 2, :], h, sigma

    return u
```

In [16]:
```
order = 2
type = (2, 2)

u_explicit = explicit_scheme(x_begin, x_end, t_begin, t_end, h, tau, a, psi_1, psi_

x = np.arange(x_begin, x_end + h, h)
t = np.arange(t_begin, t_end + tau, tau)

fig, axs = plt.subplots(figsize=(10, 6))

i = -1
line1, = axs.plot(x, u_explicit[i, :], label="explicit scheme")
line3, = axs.plot(x, u_exact[i, :], label="exact solution")

plt.ylim(-2, 2)
plt.legend()
```

Out[16]: <matplotlib.legend.Legend at 0x200a3105290>



Неявная схема

$$\frac{u_j^{k+1} - 2u_j^k + u_j^{k-1}}{\tau^2} = a^2 \frac{u_{j+1}^{k+1} - 2u_j^{k+1} + u_{j-1}^{k+1}}{h^2} + O(h^2 + \tau^2)$$

Выражаем u_j^{k+1} и получаем СЛАУ для трехдиагональной матрицы, которую можно решать методом прогонки написанным ранее:

$$\begin{cases} b_1 u_1^{k+1} + c_1 u_2^{k+1} = d_1, \quad j = 1, \\ a_j u_{j-1}^{k+1} + b_j u_j^{k+1} + c_j u_{j+1}^{k+1} = d_j, \quad j = 2 \ldots N-2, \\ a_{N-1} u_{N-2}^{k+1} + b_{N-1} u_{N-1}^{k+1} = d_{N-1}, \quad j = N-1. \end{cases}$$

$$a_j = c_j = -\sigma \tag{1}$$
$$b_j = 1 + 2\sigma \tag{2}$$
$$d_j = 2u_j^k - u_j^{k-1}, \; j = 2 \ldots N-2 \tag{3}$$

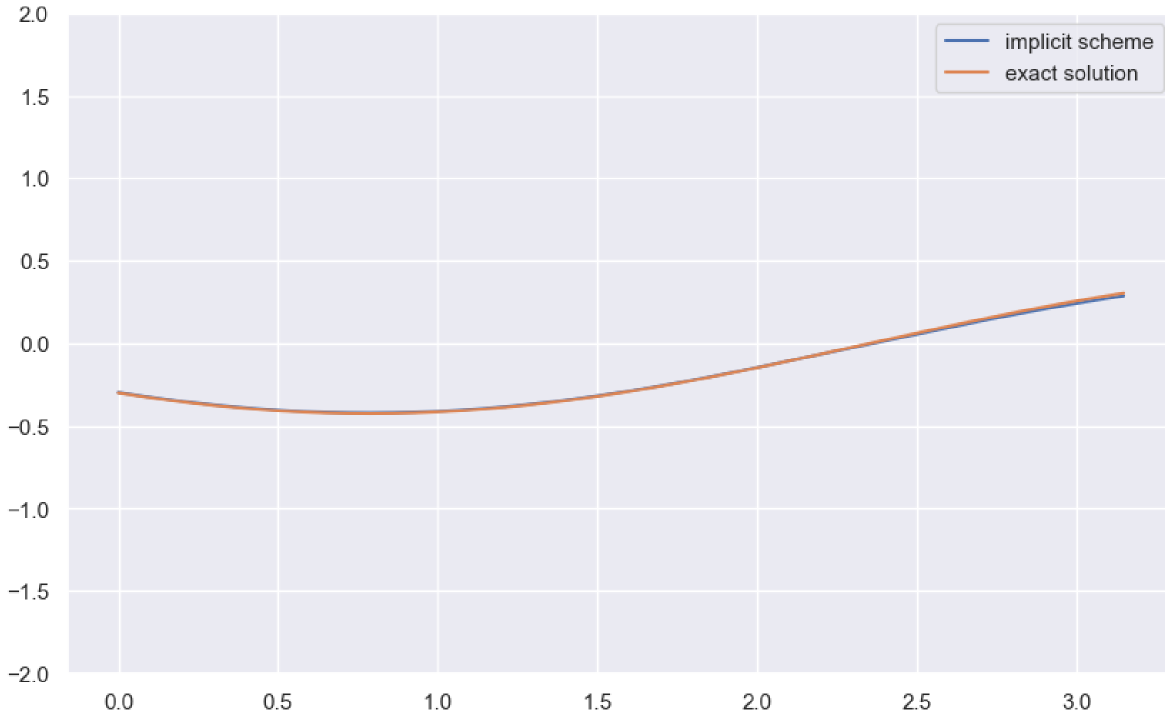
In [17]:
```
def implicit_scheme(x_begin, x_end, t_begin, t_end, h, tau, a, psi_1, psi_2, dd_psi

    sigma = a**2 * tau**2 / h**2
    x = np.arange(x_begin, x_end + h, h)
    t = np.arange(t_begin, t_end + tau, tau)
    u = np.zeros((t.size, x.size))

    u[0:2, :] = get_initial_values(x_begin, x_end, h, tau, psi_1, psi_2, dd_psi_1,

    for k in range(1, t.size - 1):
```

```python
        A = np.zeros((x.size - 2, x.size - 2))

        for i in range(1, (x.size - 2) - 1):
            A[i, i - 1] = -sigma
            A[i, i] = 1 + 2 * sigma
            A[i, i + 1] = -sigma

        b = np.zeros(x.size - 2)
        b[1:-1] = 2 * u[k, 2:-2]  - u[k - 1, 2:-2]

        A[0, 0], A[0, 1], A[-1, -2], A[-1, -1], b[0], b[-1] = get_boundary_coeffici

        u[k + 1, 1:-1] = sweep_method(A, b)
        u[k + 1, 0], u[k + 1, -1] = get_boundary_values(u[k - 1:k + 2, :], h, sigma

    return u
```

In [18]:
```python
order = 2
type = (2, 2)

u_implicit = implicit_scheme(x_begin, x_end, t_begin, t_end, h, tau, a, psi_1, psi_
```

In [19]:
```python
x = np.arange(x_begin, x_end + h, h)
t = np.arange(t_begin, t_end + tau, tau)

fig, axs = plt.subplots(figsize=(10, 6))

i = -1
line1, = axs.plot(x, u_implicit[i, :], label="implicit scheme")
line3, = axs.plot(x, u_exact[i, :], label="exact solution")

plt.ylim(-2, 2)
plt.legend()
```

Out[19]:
```
<matplotlib.legend.Legend at 0x200a3112450>
```



Полученные результаты

In [20]:
```
def print_errors(method):

    params = {
        'order': [1, 2],
        'type': [(2, 1), (3, 2), (2, 2)]
    }

    for order in params['order']:
        for i, type in zip(range(3), params['type']):
            if method == 'explicit':
                u = explicit_scheme(x_begin, x_end, t_begin, t_end, h, tau, a, psi_
            if method == 'implicit':
                u = implicit_scheme(x_begin, x_end, t_begin, t_end, h, tau, a, psi_
            print(f"order: {order}, type: {params['type'][(i + 1) % 3]}")
            print(f"mean abs error = {mean_abs_error(u_exact, u)}")
            print(f"max abs error = {max_abs_error(u_exact, u)} \n")
```

In [21]:
```
print_errors('explicit')
```

```
order: 1, type: (3, 2)
mean abs error = 0.0014848953460146353
max abs error = 0.011939871435103666

order: 1, type: (2, 2)
mean abs error = 0.0018608466006349474
max abs error = 0.015929381202457682

order: 1, type: (2, 1)
mean abs error = 0.0018650209818401368
max abs error = 0.0160241682779958

order: 2, type: (3, 2)
mean abs error = 0.0007386862590550672
max abs error = 0.009798512522318725

order: 2, type: (2, 2)
mean abs error = 0.0008288608888675565
max abs error = 0.013780041605109539

order: 2, type: (2, 1)
mean abs error = 0.0008330797177552054
max abs error = 0.013874844087588822
```

In [23]:
```
print_errors('implicit')
```

```
order: 1, type: (3, 2)
mean abs error = 0.0023362301402059553
max abs error = 0.014878776429625207

order: 1, type: (2, 2)
mean abs error = 0.0027197723046961343
max abs error = 0.018897905427790018

order: 1, type: (2, 1)
mean abs error = 0.002720501663077587
max abs error = 0.018938232601557747

order: 2, type: (3, 2)
mean abs error = 0.001333766152520951
max abs error = 0.012742526207755234

order: 2, type: (2, 2)
mean abs error = 0.0016879524734827798
max abs error = 0.01675361224208144

order: 2, type: (2, 1)
mean abs error = 0.0016886703486353447
max abs error = 0.016793864953648596
```

Вывод

В данной работе я научился решать начально-краевые задачи для ДУ гиперболического типа двумя способами:

- с помощью явной конечно-разностной схемы
- с помощью неявной конечно-разностной схемы

С помощью каждого метода получилось решить заданное ДУ с приемлемой точностью.

Явная конечно-разностная схема легко считается, но она условно устойчива, следовательно результат получится правдивый не при любых параметрах сетки.

Неявная конечно-разностная схема считается более сложным образом, но зато она абсолютно устойчива