

Лабораторная работа №5 учебного года 2023-2024 по курсу «Численные методы»

Выполнил: Москвин А. А.

Группа: М8О-408Б-20

Преподаватель: Пивоваров Д.Е.

Вариант по списку группы: 18

Условие лабораторной работы

Используя явную и неявную конечно-разностные схемы, а также схему Кранка - Николсона, решить начально-краевую задачу для дифференциального уравнения параболического 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 $U(x, t)$. Исследовать зависимость погрешности от сеточных параметров τ, h .

Вариант 8

$$\frac{\partial u}{\partial t} = a \frac{\partial^2 u}{\partial x^2} + cu, \quad a > 0, \quad c < 0.$$

$$u_x(0, t) = \exp((c - a)t),$$

$$u\left(\frac{\pi}{2}, t\right) = \exp((c - a)t),$$

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

Аналитическое решение: $U(x, t) = \exp((c - a)t) \sin x$.

Программа

main.py

```
import numpy as np
```

```
from functions import phi0, phi1, psi
from show import show_inaccuracy, show_result
from task import a, c, count_t, count_x, h, sigma, tau
```

```

def explicit_scheme(bound_condition):
    u = np.zeros((count_t, count_x))

    for i in range(1, count_x):
        u[0][i] = psi(i * h)

    for k in range(1, count_t):
        for i in range(1, count_x - 1):
            u[k][i] = (
                sigma * u[k - 1][i + 1]
                + (1 - 2 * sigma) * u[k - 1][i]
                + sigma * u[k - 1][i - 1]
                + c * tau * u[k - 1][i]
            )

        if bound_condition == 1:
            u[k][0] = u[k][1] - h * phi0(k * tau)
            u[k][-1] = phi1(k * tau)
        elif bound_condition == 2:
            u[k][0] = (phi0(k * tau) + u[k][2] / (2 * h) - 2 * u[k][1] / h) *
2 * h / -3
            u[k][-1] = phi1(k * tau)
        elif bound_condition == 3:
            u[k][0] = (
                u[k][1] - h * phi0(k * tau) + (h**2 / (2 * tau) * u[k -
1][0])
            ) / (1 + h**2 / (2 * tau))
            u[k][-1] = phi1(k * tau)
        else:
            print("Условие не найдено")
    return u

```

```

def implicit_scheme(bound_condition):
    u = np.zeros((count_t, count_x))

    ai = np.zeros(count_x)
    bi = np.zeros(count_x)
    ci = np.zeros(count_x)
    di = np.zeros(count_x)

```

```

    for i in range(1, count_x):
        u[0][i] = psi(i * h)

```

```

for k in range(1, count_t):
    for i in range(1, count_x - 1):
        ai[i] = sigma
        bi[i] = -2 * sigma + c * tau - 1
        ci[i] = sigma
        di[i] = -u[k - 1][i]

    if bound_condition == 1:
        bi[0] = -(1 + 2 * sigma - c * tau)
        ci[0] = 2 * sigma
        di[0] = -(u[k - 1][0] - 2 * a * tau * phi0(k * tau) / h)
        ai[-1] = 2 * sigma
        bi[-1] = -(1 + 2 * sigma - c * tau)
        di[-1] = -phi1(k * tau)
    elif bound_condition == 2:
        bi[0] = -(1 + 2 * sigma - c * tau)
        ci[0] = 2 * sigma
        di[0] = -(u[k - 1][0] - 2 * a * tau * phi0(k * tau) / h)
        ai[-1] = 2 * sigma
        bi[-1] = -(1 + 2 * sigma - c * tau)
        di[-1] = -phi1(k * tau)
    elif bound_condition == 3:
        bi[0] = -(1 + 2 * sigma - c * tau)
        ci[0] = 2 * sigma
        di[0] = -(
            (1 - sigma) * u[k - 1][1] + sigma / 2 * u[k - 1][0]
        ) - sigma * phi0(k * tau)
        ai[-1] = 2 * sigma
        bi[-1] = -(1 + 2 * sigma - c * tau)
        di[-1] = -phi1(k * tau)
    else:
        print("Условие не найдено")
    u[k] = thomas_algorithm(ai, bi, ci, di)
return u

```

```

def thomas_algorithm(a, b, c, d):

```

```

    size = len(a)
    p = np.zeros(size)
    q = np.zeros(size)
    p[0] = -c[0] / b[0]
    q[0] = d[0] / b[0]

```

```

    for i in range(1, size):
        s = b[i] + a[i] * p[i - 1]

```

```
p[i] = -c[i] / s
q[i] = (d[i] - a[i] * q[i - 1]) / s
```

```
result = np.zeros(size)
result[-1] = q[-1]
```

```
for i in range(size - 2, -1, -1):
    result[i] = p[i] * result[i + 1] + q[i]
```

```
return result
```

```
def explicit_implicit_scheme(omega, bound_condition):
    u = np.zeros((count_t, count_x))
```

```
    imp = implicit_scheme(bound_condition)
    exp = explicit_scheme(bound_condition)
```

```
    for k in range(count_t):
        for i in range(count_x):
            u[k][i] = imp[k][i] * omega + exp[k][i] * (1 - omega)
```

```
    return u
```

```
def get_axis_np(count, mul):
```

```
    axis = np.zeros(count)
    for i in range(count):
        axis[i] = mul * i
    return axis
```

```
def solve():
```

```
    res1 = explicit_scheme(1)
    res2 = implicit_scheme(1)
    res3 = explicit_implicit_scheme(0.5, 1)
```

```
    t_axis = get_axis_np(count_t, tau)
    x_axis = get_axis_np(count_x, h)
```

```
    show_result(t_axis, x_axis, res1, res2, res3)
    show_inaccuracy(t_axis, x_axis, res1)
```

```
if __name__ == "__main__":  
    solve()
```

show.py

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

```
from functions import analytic_solution  
from task import count_t
```

```
def show_result(t_axis, x_axis, u1, u2, u3):  
    fig, ax = plt.subplots(2)  
    fig.suptitle("Сравнение численных решений ДУ с  
аналитическим")  
    fig.set_figheight(15)  
    fig.set_figwidth(16)  
    time = 0  
    for i in range(2):  
        ax[i].plot(x_axis, u1[time, :], label="Explicit scheme")  
        ax[i].plot(x_axis, u2[time, :], label="Implicit scheme")  
        ax[i].plot(x_axis, u3[time, :], label="Crank-Nicholson  
scheme")  
        ax[i].plot(  
            x_axis,  
            [analytic_solution(x, t_axis[time]) for x in x_axis],  
            label="Analytic",  
        )  
        ax[i].grid(True)  
        ax[i].set_xlabel("x")  
        ax[i].set_ylabel("u")  
        ax[i].set_title(f"Решения при t = {time / count_t}")  
        time += count_t - 1  
  
    plt.legend(bbox_to_anchor=(1.05, 2), loc="upper right",  
borderaxespad=0)  
    plt.show()  
  
fig = plt.figure(num=1, figsize=(19, 12), clear=True)  
ax = fig.add_subplot(1, 1, 1, projection="3d")  
fig.suptitle("Аналитическое решение")  
xgrid, tgrid = np.meshgrid(x_axis, t_axis)
```

```

ax.plot_surface(xgrid, tgrid, analytic_solution(xgrid, tgrid))
ax.set(xlabel="x", ylabel="t", zlabel="u")
fig.tight_layout()
plt.show()

```

```

def show_inaccuracy(t_axis, x_axis, u):
    inaccuracy = np.zeros(count_t)
    for i in range(count_t):
        inaccuracy[i] = np.max(
            np.abs(u[i] - np.array([analytic_solution(x, t_axis[i]) for x in
x_axis])))
    )

    plt.figure(figsize=(14, 8))
    plt.plot(t_axis[1:], inaccuracy[1:], "violet", label="Ошибка")
    plt.legend(bbox_to_anchor=(1.05, 1), loc="upper right",
borderaxespad=0.0)
    plt.title("График изменения ошибки во времени")
    plt.xlabel("t")
    plt.ylabel("error")
    plt.grid(True)

    plt.show()

```

task.py

```

import numpy as np

```

```

t_max = 1

```

```

count_x = 60
count_t = 1000

```

```

r_coord = np.pi / 2

```

```

a = 0.1
c = -1

```

```

h = r_coord / count_x
tau = t_max / count_t

```

```

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

```

functions.py

```
import numpy as np
```

```
from task import a, c
```

```
def analytic_solution(x, t):  
    return np.exp((c - a) * t) * np.sin(x)
```

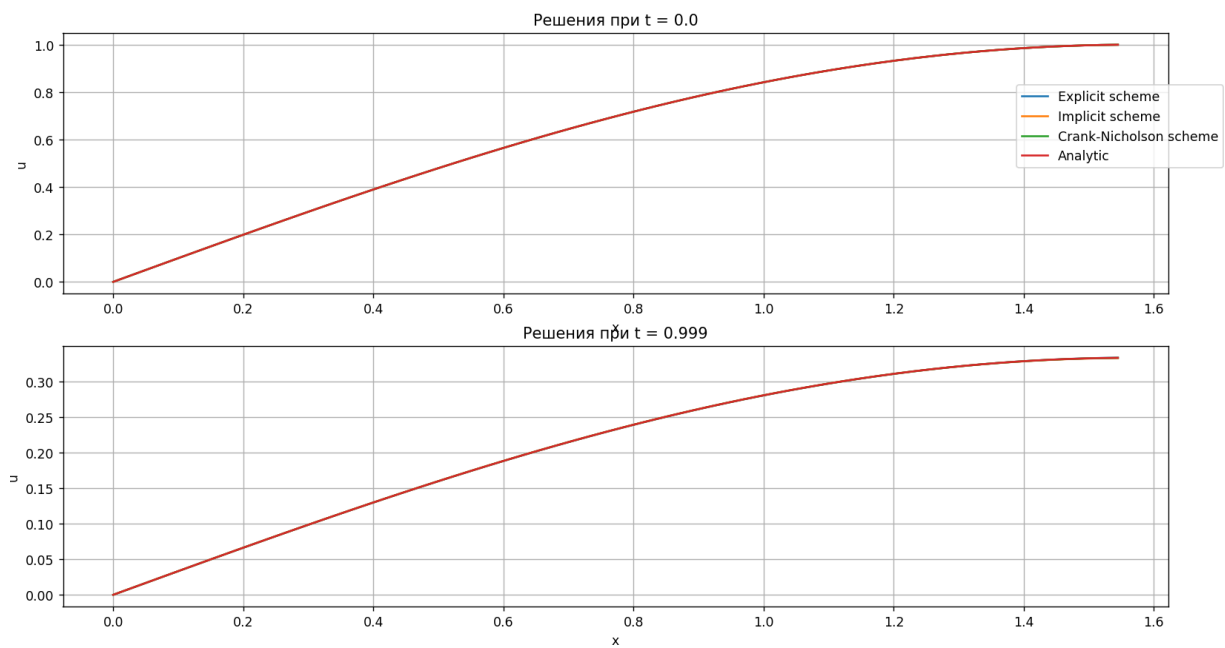
```
def psi(x):  
    return np.sin(x)
```

```
def phi0(t):  
    return np.exp((c - a) * t)
```

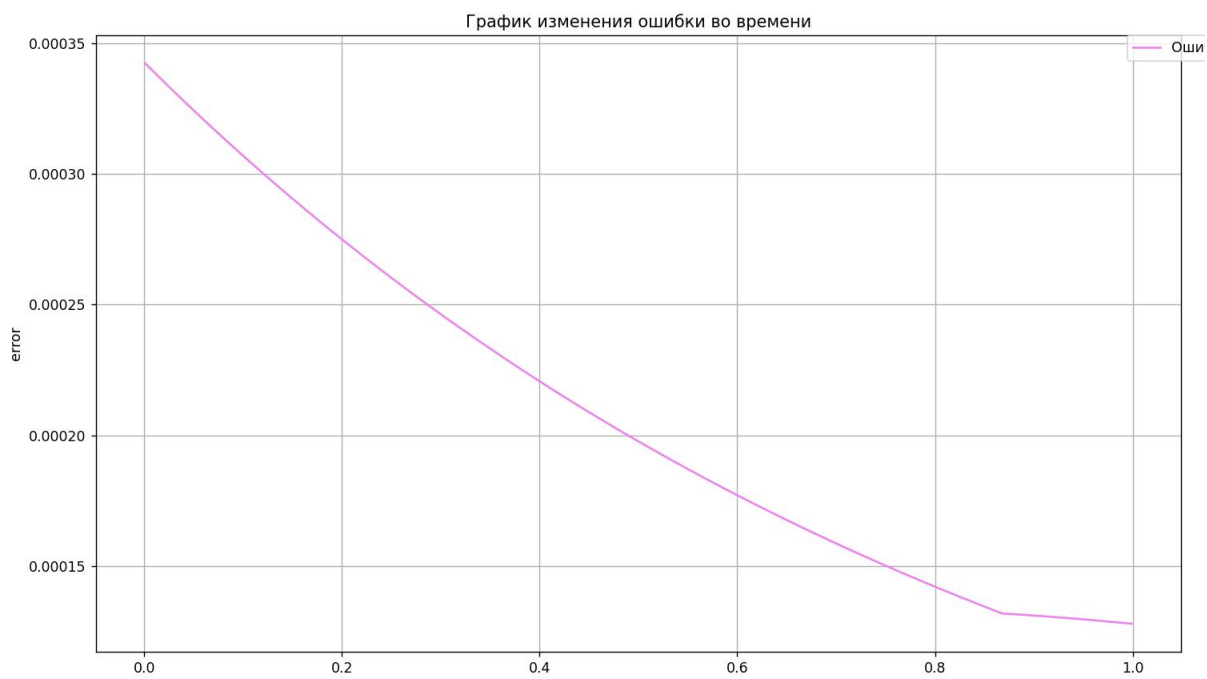
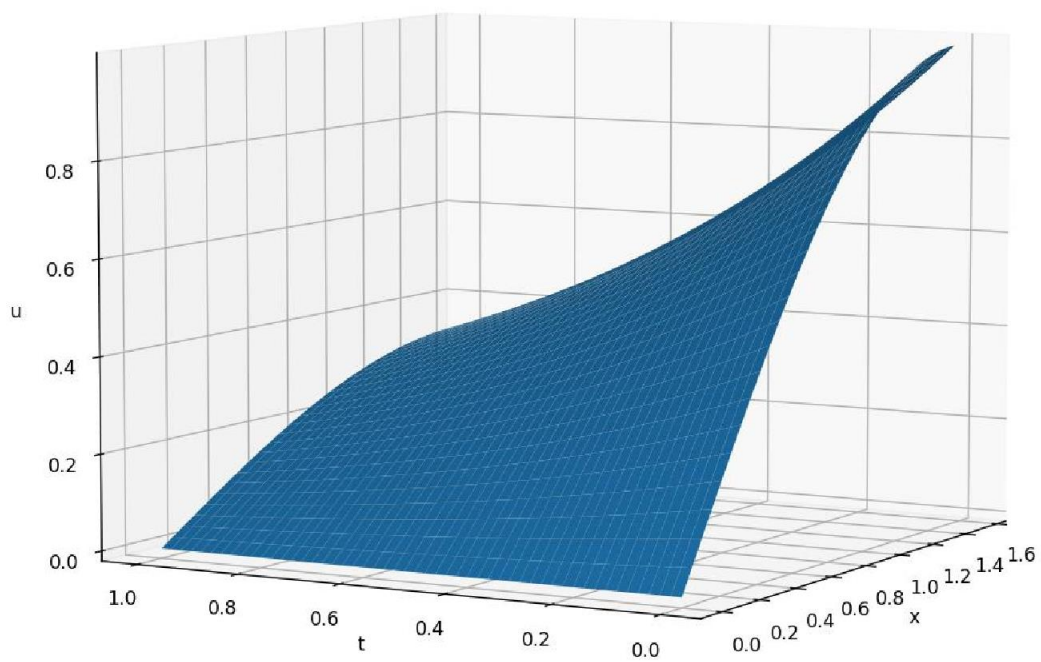
```
def phi1(t):  
    return np.exp((c - a) * t)
```

Результаты работы

Сравнение численных решений ДУ с аналитическим



Аналитическое решение



Вывод по лабораторной работе

Благодаря данной лабораторной работе, я приобрел знания в области численных методов для решения дифференциальных уравнений параболического типа: были исследованы различные методы решения начально-краевой задачи для дифференциального уравнения параболического типа, включая схему Кранка-Николсона, неявную и явную конечно-разностные методы, а также использование аналитического решения. Эксперименты позволили оценить точность и эффективность каждого метода.