International IT University
Faculty of Computer technologies and cyber security
Department: MCM

Report

In the discipline «Numerical Analysis»

Executed: Taldybayev B.A.

Group: IT3-2203

Lecturer: Шахан Н.Ш.

Almaty, 2025

Task 5: 1D Burgers Equation

1. $$\frac{\partial U}{\partial t} + a\frac{\partial U}{\partial x} = \mu\frac{\partial^2 U}{\partial x^2},$$ where $t > 0$, $x \in [0, L]$ and a – transport coefficient, μ – viscosity coefficient
2. Derivative by time:

$$\frac{\partial U}{\partial t} \approx \frac{U_i^{n+1} - U_i^n}{\Delta t}$$

Derivative in space:

$$\frac{\partial U}{\partial x} \approx \frac{U_i^{n+1} - U_{i-1}^{n+1}}{h}$$

Second derivative of x:

$$\frac{\partial^2 U}{\partial x^2} \approx \frac{U_{i+1}^{n+1} - 2U_i^{n+1} + U_{i-1}^{n+1}}{h^2}$$

3. Substitution in the formula:

$$\frac{U_i^{n+1} - U_i^n}{\Delta t} + a\frac{U_i^{n+1} - U_{i-1}^{n+1}}{h} = q\frac{U_{i+1}^{n+1} - 2U_i^{n+1} + U_{i-1}^{n+1}}{h^2}$$

4. Multiply to delta t:

$$U_i^{n+1} - U_i^n + a\frac{\Delta t}{h}(U_i^{n+1} - U_{i-1}^{n+1}) = q\frac{\Delta t}{h^2}(U_{i+1}^{n+1} - 2U_i^{n+1} + U_{i-1}^{n+1})$$

5. Replacing Variables:

$$r = \frac{q\Delta t}{h^2}, \quad s = \frac{a\Delta t}{h}$$

6. In the end:

$$-rU_{i-1}^{n+1} + (1 + 2r + s)U_i^{n+1} - rU_{i+1}^{n+1} = U_i^n$$

Code and graph:

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


def thomas_algorithm(a, b, c, d):
    n = len(d)
    c_ = np.zeros(n - 1)
    d_ = np.zeros(n)

    c_[0] = c[0] / b[0]
    d_[0] = d[0] / b[0]

    for i in range(1, n - 1):
        c_[i] = c[i] / (b[i] - a[i - 1] * c_[i - 1])

    for i in range(1, n):
        d_[i] = (d[i] - a[i - 1] * d_[i - 1]) / (b[i] - a[i - 1] * c_[i - 1])

    u = np.zeros(n)
    u[-1] = d_[-1]
    for i in range(n - 2, -1, -1):
        u[i] = d_[i] - c_[i] * u[i + 1]

    return u


L = 2.0
T = 1.0
N = 100
M = 200
dx = L / (N - 1)
dt = T / M
a = 1.0
mu = 0.1

r = mu * dt / dx ** 2
s = a * dt / (2 * dx)

x = np.linspace(0, L, N)
u = np.sin(np.pi * x)

time_steps = [0.2, 0.4, 0.6, 0.8, 1.0]
solutions = []

for n in range(M):
    a_diag = -r * np.ones(N - 2)
    b_diag = (1 + 2 * r) * np.ones(N - 1)
    c_diag = -r * np.ones(N - 2)
    d = u[1:-1] - s * (u[2:] - u[:-2])

    u_new = np.zeros(N)
    u_new[0] = 0
    u_new[-1] = 0
    d[0] += r * u_new[0]
    d[-1] += r * u_new[-1]

    u_new[1:-1] = thomas_algorithm(a_diag, b_diag, c_diag, d)

    u = u_new.copy()

    if (n + 1) * dt in time_steps:
```

```
        solutions.append(u.copy())

plt.figure(figsize=(8, 6))
for i, t in enumerate(time_steps):
    plt.plot(x, solutions[i], label=f't = {t}')

plt.xlabel("x")
plt.ylabel("u(x, t)")
plt.title("Burger's Solution")
plt.legend()
plt.grid()
plt.show()
```