Московский авиационный институт
(национальный исследовательский университет)

Институт №8 "Информационные технологии и прикладная математика"
Кафедра 806 «Вычислительная математика и программирование»

Лабораторная работа №5 по
дисциплине:

Численные методы

Вариант №3

Выполнил: студент группы М8О-409Б-20
Чибисов М.Р.
Принял: Пивоваров Е.Д.
Оценка:____

Москва, 2023 г.

1. Задание

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.

Уравнение:

$$\frac{\partial u}{\partial t} = a\frac{\partial^2 u}{\partial x^2}, \quad a > 0,$$
$$u(0,t) = \exp(-at),$$
$$u(\pi,t) = -\exp(-at),$$
$$u(x,0) = \cos x.$$

Аналитическое решение:

$$U(x,t) = \exp(-at)\cos x.$$

2. Решение

- Явная схема:

$$\frac{u_j^{k+1} - u_j^k}{\tau} = \frac{u_{j-1}^k - 2u_j^k + u_{j+1}^k}{h^2} + f\left(x_j, t^k\right)$$

- Неявная схема:

$$\frac{u_j^{k+1} - u_j^k}{\tau} = \frac{u_{j-1}^{k+1} - 2u_j^{k+1} + u_{j+1}^{k+1}}{h^2} + f\left(x_j, t^{k+1}\right)$$

- Гибридная схема (при $\theta = 0.5$, схема Кранка-Николсона)

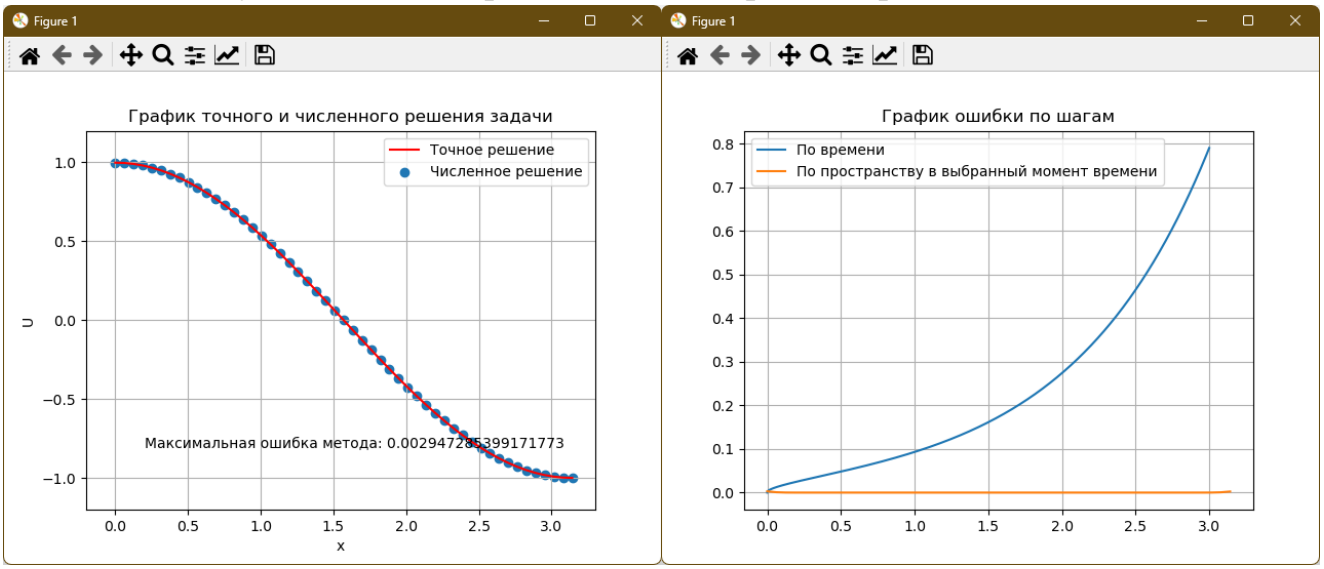
$$\frac{u_j^{k+1} - u_j^k}{\tau} = (\theta)\left(\frac{u_{j-1}^{k+1} - 2u_j^{k+1} + u_{j+1}^{k+1}}{h^2} + f\left(x_j, t^k\right)\right) + (1-\theta)\left(\frac{u_{j-1}^k - 2u_j^k + u_{j+1}^k}{h^2} + f\left(x_j, t^k\right)\right)$$

Погрешность между численным и аналитическим решением рассчитывается как модуль разности.

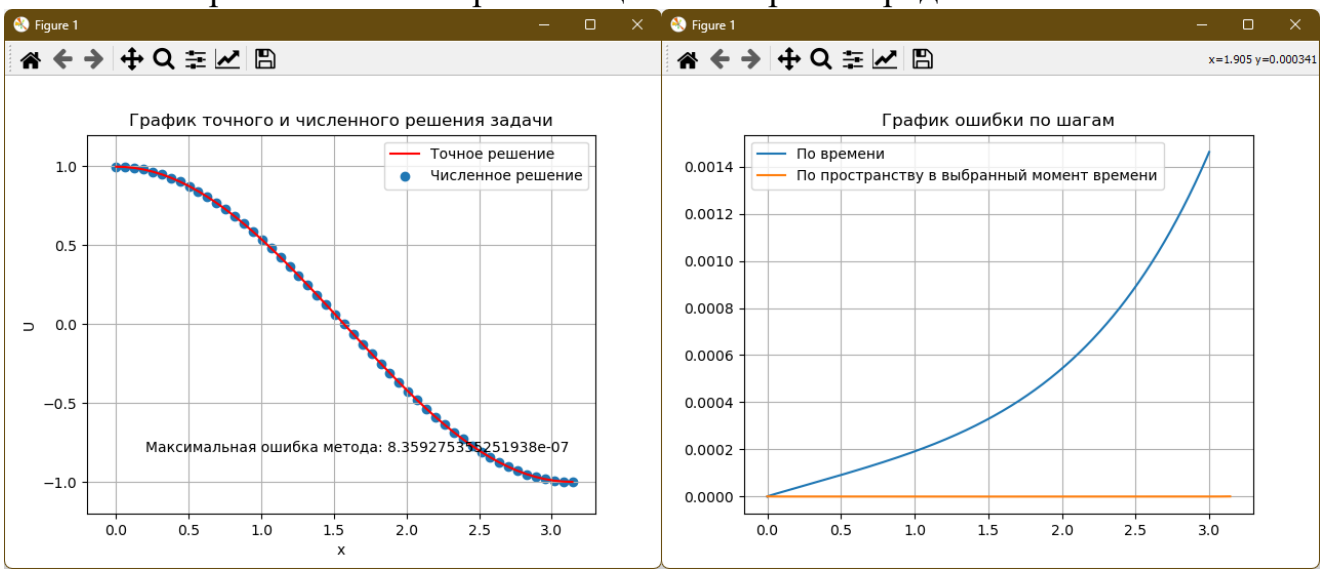
3. Вывод программы

- Явная схема

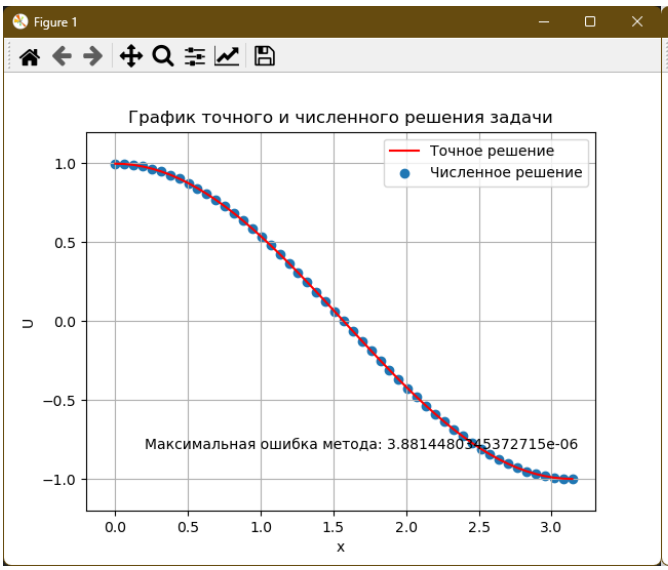
- Двухточечная аппроксимация с первым порядком

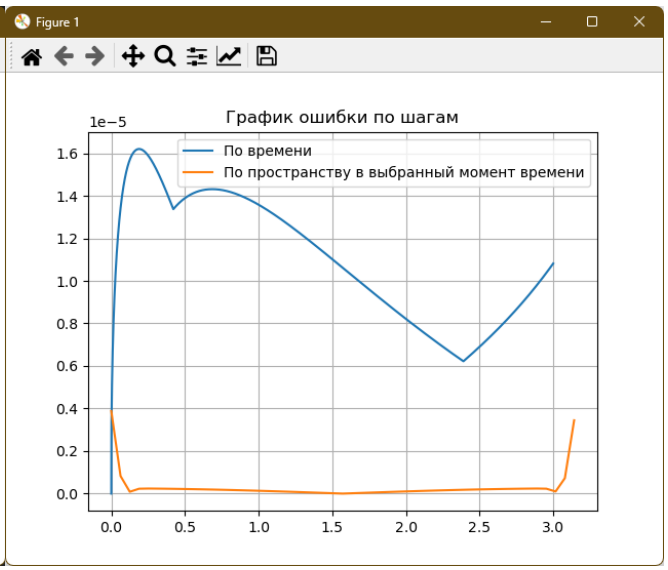


- Трёхточечная аппроксимация со вторым порядком



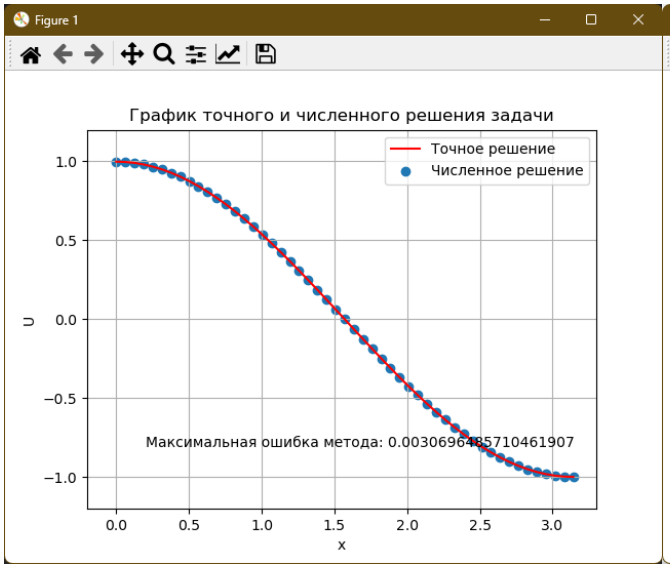
- Двухточечная аппроксимация со вторым порядком

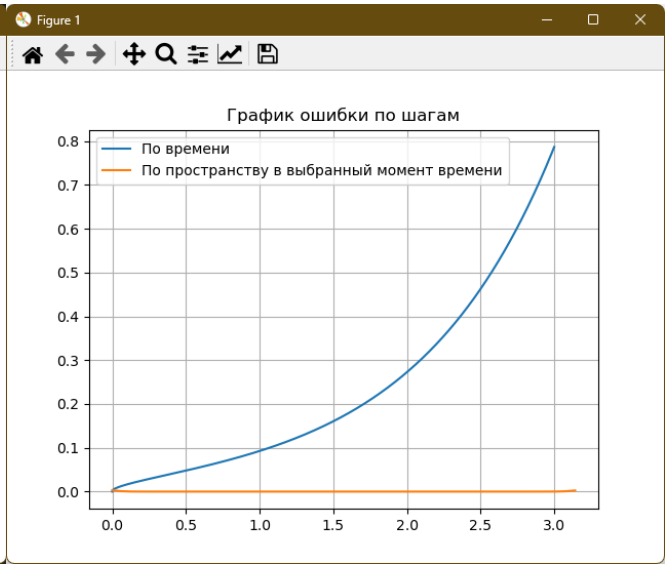




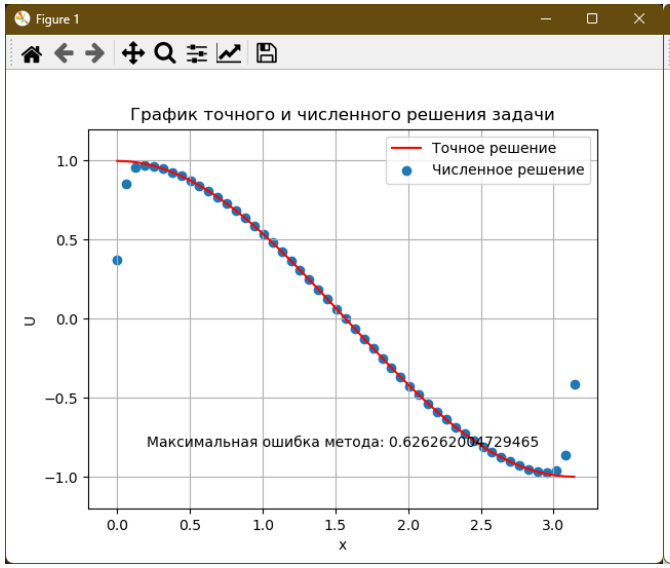
- Неявная схема

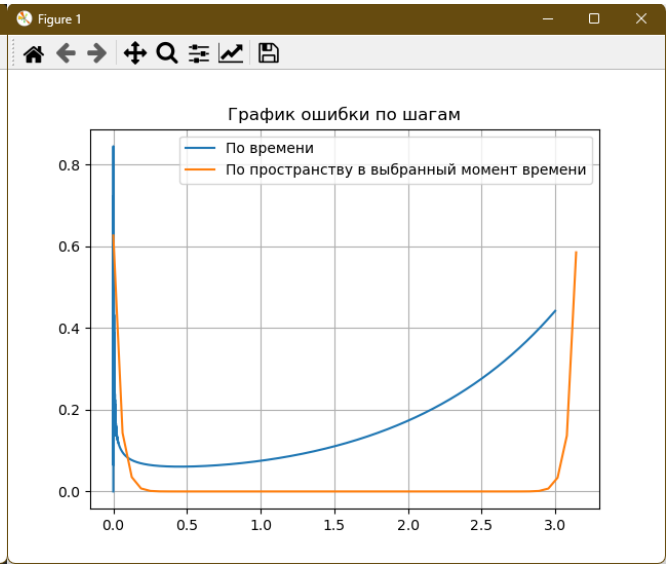
- Двухточечная аппроксимация с первым порядком



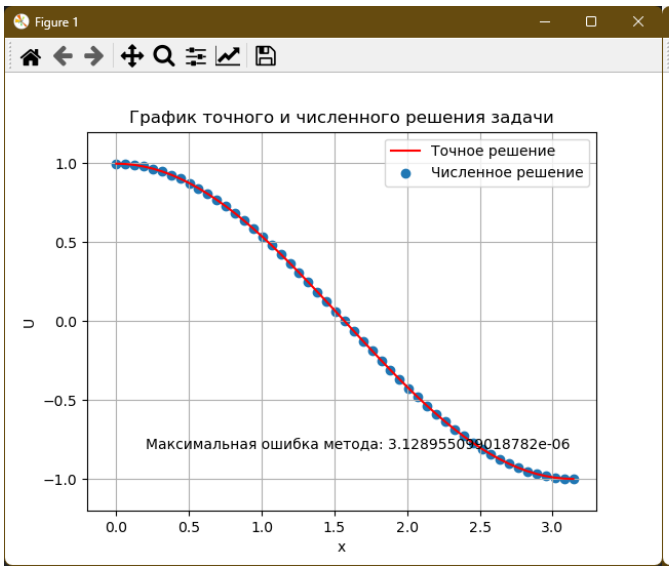


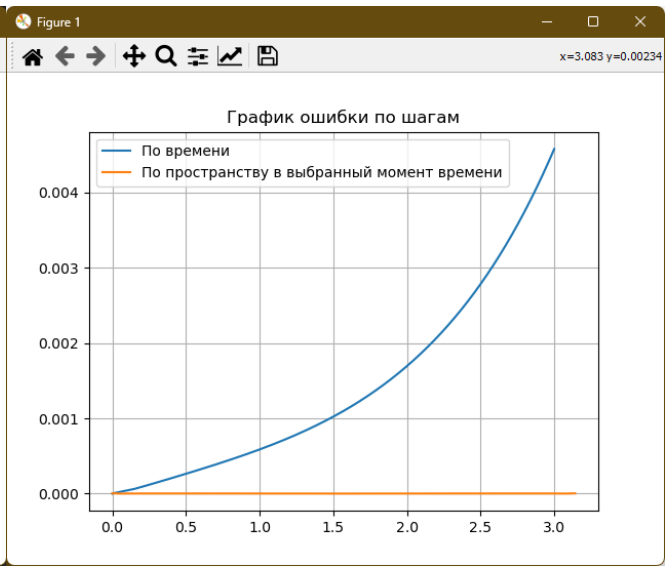
- Двухточечная аппроксимация со вторым порядком



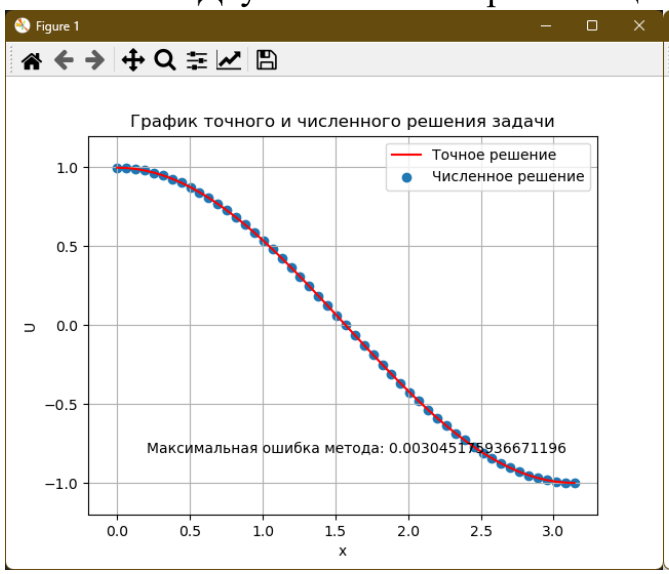


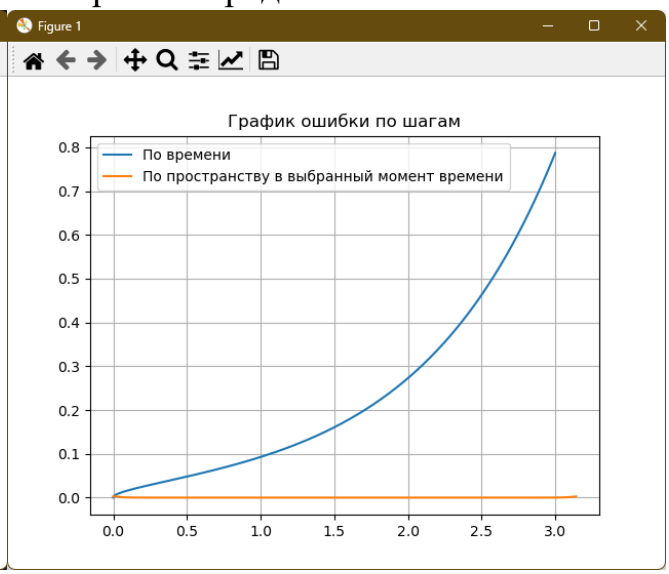
- Трёхточечная аппроксимация со вторым порядком



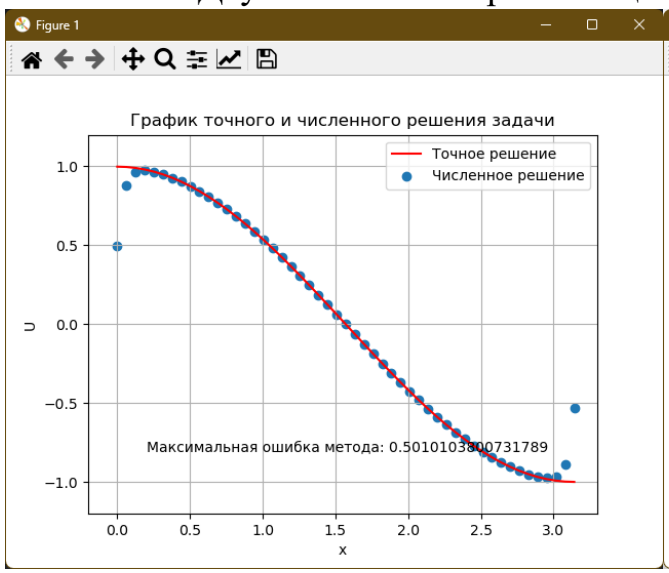


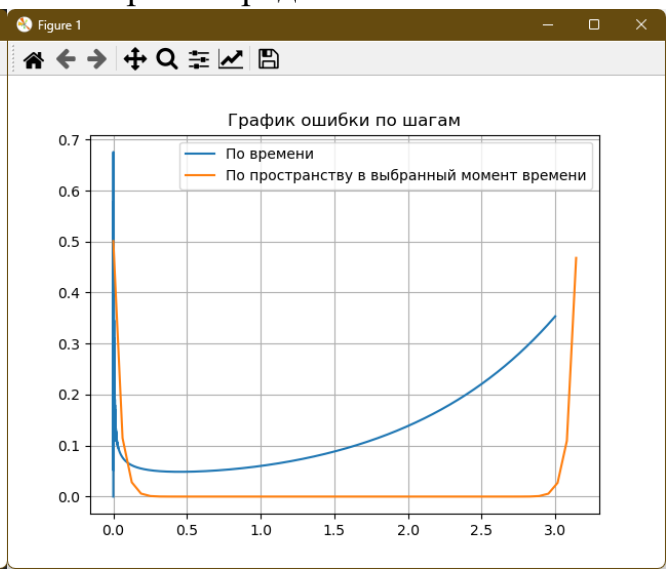
- Схема Кранка-Николсона ($\theta = 0.8$):
  - Двухточечная аппроксимация с первым порядком



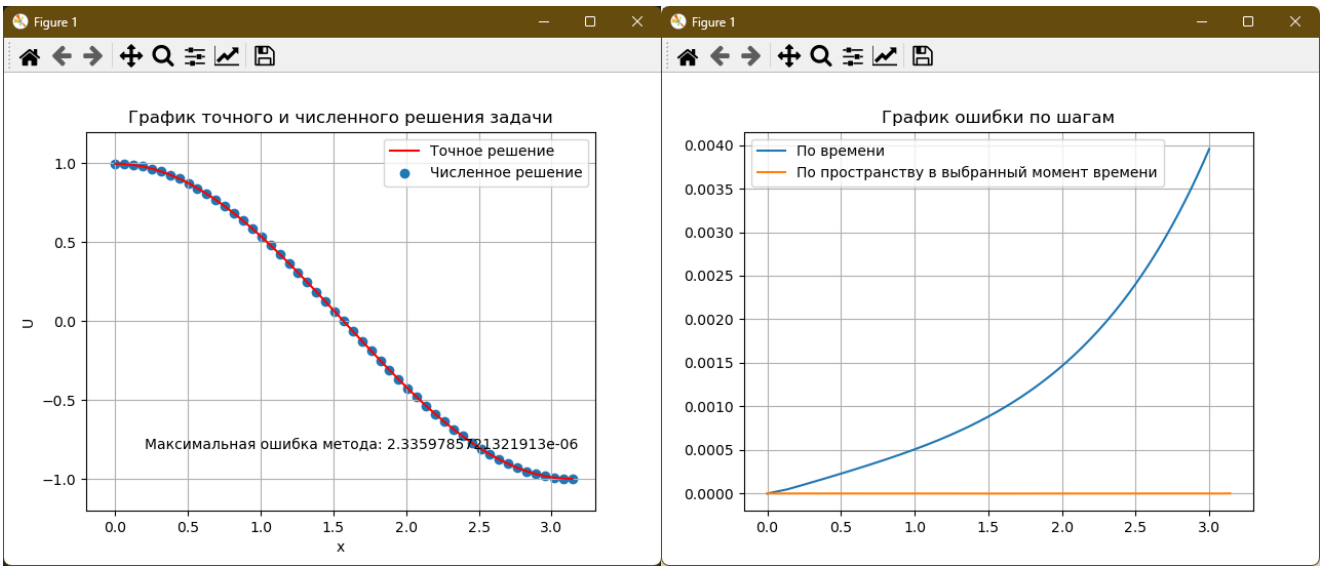


  - Двухточечная аппроксимация со вторым порядком





  - Трёхточечная аппроксимация со вторым порядком



Листинг

```
import matplotlib.pyplot as plt
import numpy as np
def analyt_func(x, alpha, t):
    return np.exp(-alpha * t) * np.cos(x)
def left_border(alpha, t):
    return np.exp(- alpha * t)
def right_border(alpha, t):
    return -np.exp(- alpha * t)
def tma(a, b, c, d, shape):
    p = [-c[0] / b[0]]
    q = [d[0] / b[0]]
    x = [0] * (shape + 1)
    for i in range(1, shape):
        p.append(-c[i] / (b[i] + a[i] * p[i - 1]))
        q.append((d[i] - a[i] * q[i - 1]) / (b[i] + a[i] * p[i - 1]))
    for i in reversed(range(shape)):
        x[i] = p[i] * x[i + 1] + q[i]
    return x[:-1]
def explicit(K, t, tau, h, alpha, x, approx):
    N = len(x)
    U = np.zeros((K, N))
    for j in range(N):
        U[0, j] = np.cos(x[j])

    for k in range(K - 1):
        t += tau
        for j in range(1, N - 1):
            U[k + 1, j] = tau * (alpha * (U[k, j - 1] - 2 * U[k, j] +
U[k, j + 1]) / h ** 2) + U[k, j]
        if approx == 1:
            U[k + 1, 0] = (h * left_border(alpha, t) - U[k + 1, 1]) /
```

```
(h - 1)
            U[k + 1, N - 1] = (h * right_border(alpha, t) + U[k + 1, N
- 2]) / (h + 1)
        elif approx == 2:
            U[k + 1, 0] = (2 * h * left_border(alpha, t) - 4 * U[k +
1, 1] + U[k + 1, 2]) / (2 * h - 3)
            U[k + 1, N - 1] = (2 * h * right_border(alpha, t) + 4 *
U[k + 1, N - 2] - U[k + 1, N - 3]) / (2 * h + 3)
        elif approx == 3:
            U[k + 1, 0] = ((left_border(alpha, t) * h * tau * 2 - U[k
+ 1, 1] * (2 * tau) - U[k, 0] * h ** 2) /
                           (-2 * tau - h ** 2 + h * tau * 2))
            U[k + 1, N - 1] = (
                    (right_border(alpha, t) * h * tau * 2 + U[k + 1, N
- 2] * (2 * tau) + U[k, N - 1] * h ** 2) /
                    (2 * tau + h ** 2 + h * tau * 2))

    return U

def implicit(K, t, tau, h, alpha, x, approx):
    N = len(x)
    U = np.zeros((K, N))
    for j in range(N):
        U[0, j] = np.cos(x[j])

    for k in range(0, K - 1):
        a = np.zeros(N)
        b = np.zeros(N)
        c = np.zeros(N)
        d = np.zeros(N)
        t += tau

        for j in range(1, N - 1):
            a[j] = tau * alpha / h ** 2
            b[j] = tau * (-2 * alpha) / h ** 2 - 1
            c[j] = tau * alpha / h ** 2
            d[j] = -U[k][j]

        if approx == 1:
            b[0] = 1 - 1 / h
            c[0] = 1 / h
            d[0] = left_border(alpha, t)

            a[N - 1] = -1 / h
            b[N - 1] = 1 + 1 / h
            d[N - 1] = right_border(alpha, t)
```

```
        elif approx == 2:
            b[0] = 2 * alpha ** 2 / h + h / tau - 2
            c[0] = - 2 * alpha ** 2 / h
            d[0] = (h / tau) * U[k - 1][0]- left_border(alpha, t) * 2

            a[N - 1] = -2 * alpha ** 2 / h
            b[N - 1] = 2 * alpha ** 2 / h + h / tau + 2
            d[N - 1] = (h / tau) * U[k - 1][N - 1] +
right_border(alpha, t) * 2

        elif approx == 3:
            k0 = 1 / (2 * h) / c[1]
            b[0] = (-3 / (2 * h)) + 1 + a[1] * k0
            c[0] = 2 / h + b[1] * k0
            d[0] = left_border(alpha, t) + d[1] * k0

            k1 = -(1 / (h * 2)) / a[N - 2]
            a[N - 1] = (-2 / h) + b[N - 2] * k1
            b[N - 1] = (3 / (h * 2)) + 1 + c[N - 2] * k1
            d[N - 1] = right_border(alpha, t) + d[N - 2] * k1

        u_new = tma(a, b, c, d, N)
        for i in range(N):
            U[k + 1, i] = u_new[i]

    return U


def Krank_Nikolson(K, t, tau, h, alpha, x, approx, theta):
    N = len(x)
    if theta == 0:
        U = explicit(K, t, tau, h, alpha, x, approx)
    elif theta == 1:
        U = implicit(K, t, tau, h, alpha, x, approx)
    else:
        U_ex = explicit(K, t, tau, h, alpha, x, approx)
        U_im = implicit(K, t, tau, h, alpha, x, approx)
        U = np.zeros((K, N))
        for i in range(K):
            for j in range(N):
                U[i, j] = theta * U_im[i][j] + (1-theta) * U_ex[i][j]

    return U
```

```
N = 50
K = 7000
time = 3

h = np.pi / N
tau = time / K
x = np.arange(0, np.pi + h / 2 - 1e-4, h)
T = np.arange(0, time, tau)
alpha = 1
t = 0
```

4. Вывод

В ходе выполнения лабораторной работы были изучены явная, неявная и гибридная схемы решений начально-краевой задачи для дифференциального уравнения параболического типа. Также были изучены три варианта аппроксимации граничных условий. Были получены результаты в графическом представлении и подсчитаны погрешности для каждого варианта решения.