

МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

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

Отчет по лабораторным работам по курсу «Численные методы»

Студент: Махмудов О. С.

Группа: М80-405Б-18

Преподаватель: Сластушевский Ю.В.

Дата:

Оценка:

Москва, 2021

Лабораторная работа 1

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 . Исследовать зависимость погрешности от сеточных параметров.

Реализованный метод	Решаемая задача
<ul style="list-style-type: none"> Явная конечно-разностная схема Неявная конечно-разностная схема Разностная схема Кранка-Николсона 	Начально-краевая задача для дифференциального уравнения параболического типа

10.

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

$$u_x(0, t) + u(0, t) = \exp((c - a)t)(\cos(bt) + \sin(bt)),$$

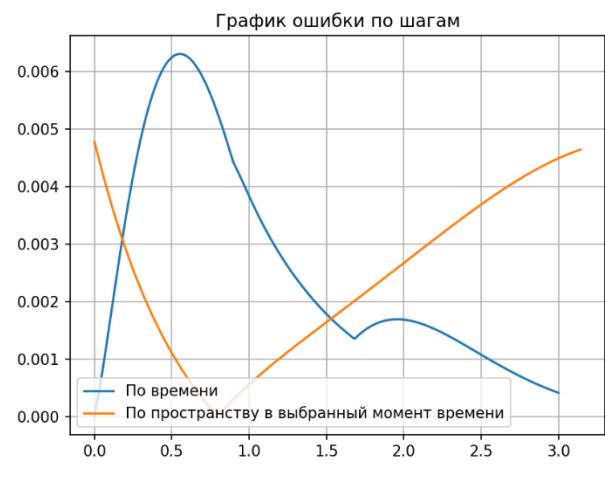
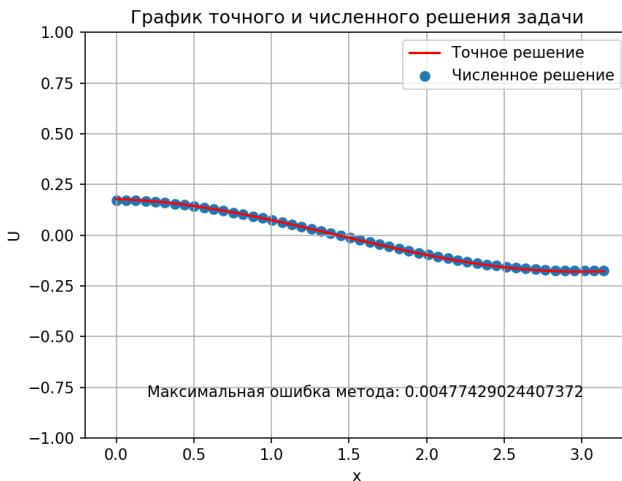
$$u_x(\pi, t) + u(\pi, t) = -\exp((c - a)t)(\cos(bt) + \sin(bt)),$$

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

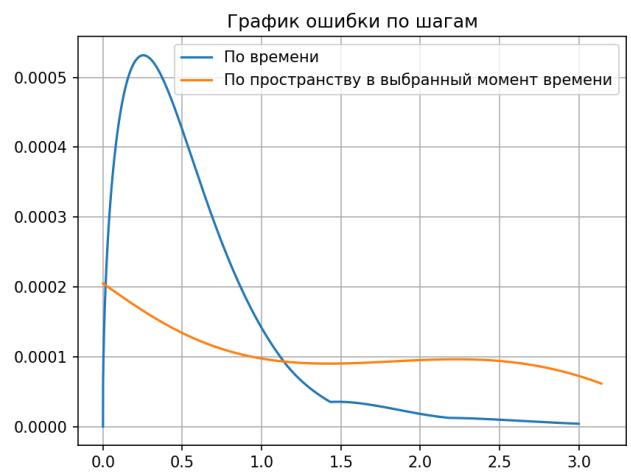
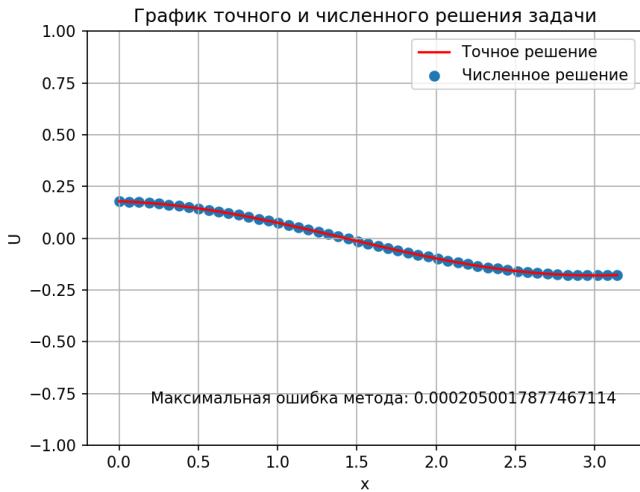
Аналитическое решение: $U(x, t) = \exp((c - a)t)\sin(x + bt)$.

1. Явный метод:

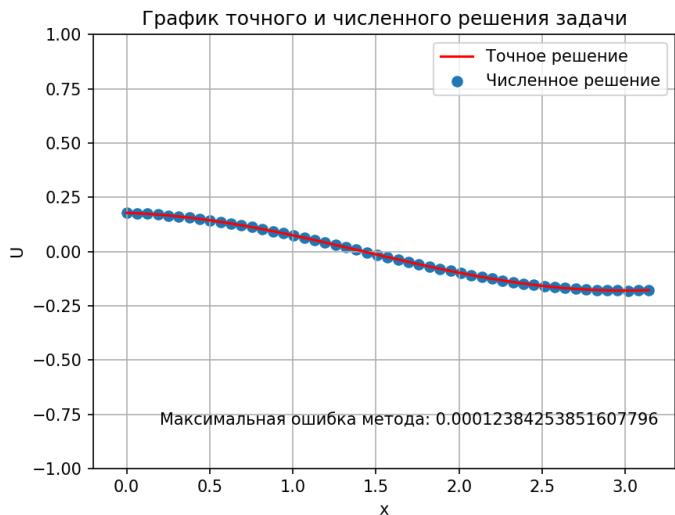
1) Двухточечная аппроксимация с первым порядком



2) Трёхточечная аппроксимация со вторым порядком

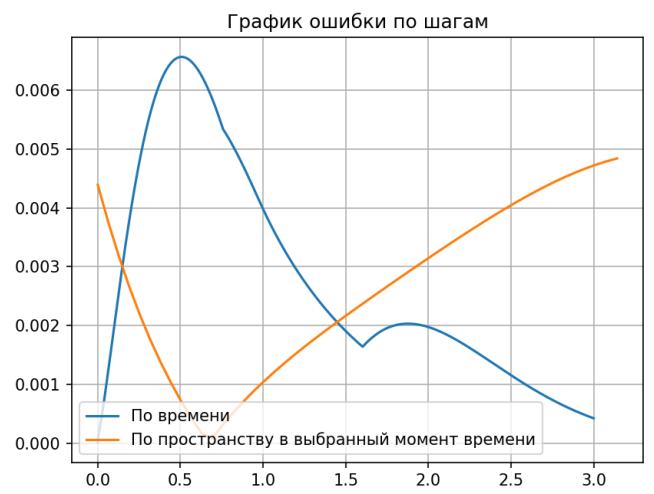
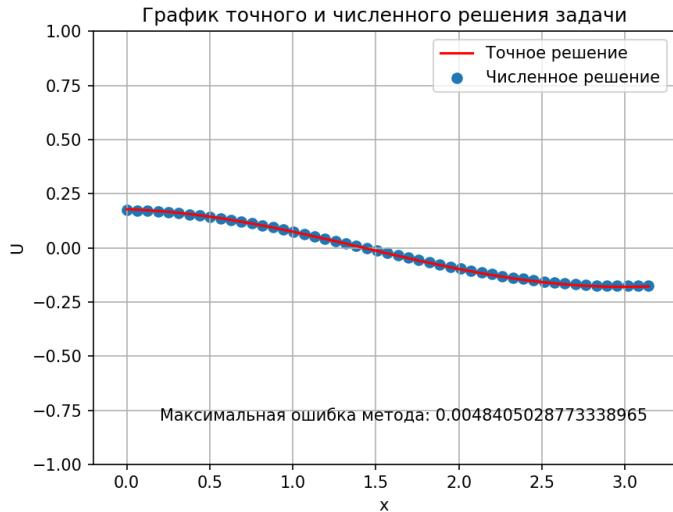


3) Двухточечная аппроксимация со вторым порядком

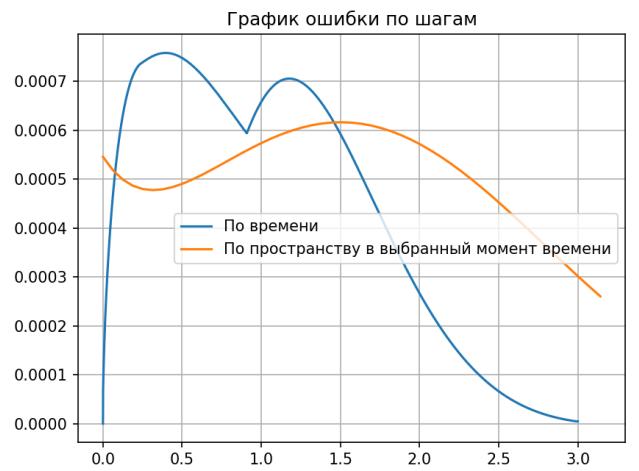
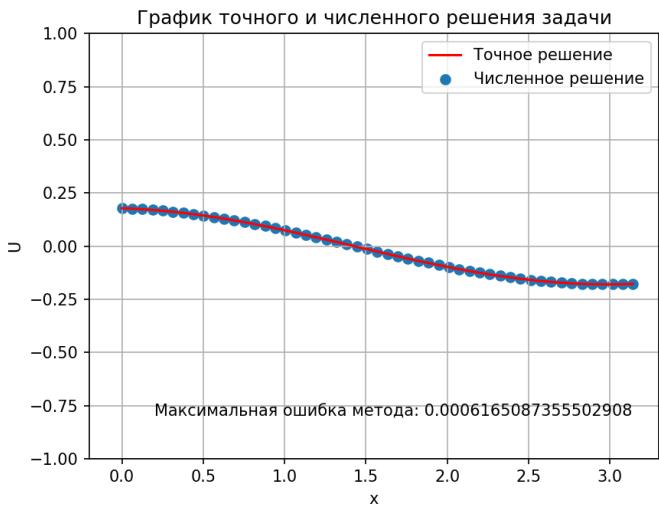


2. Неявный метод:

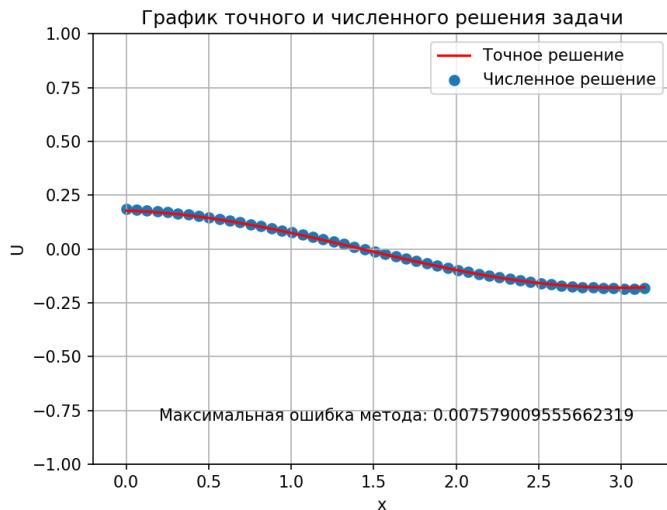
1) Двухточечная аппроксимация с первым порядком



2) Трёхточечная аппроксимация со вторым порядком

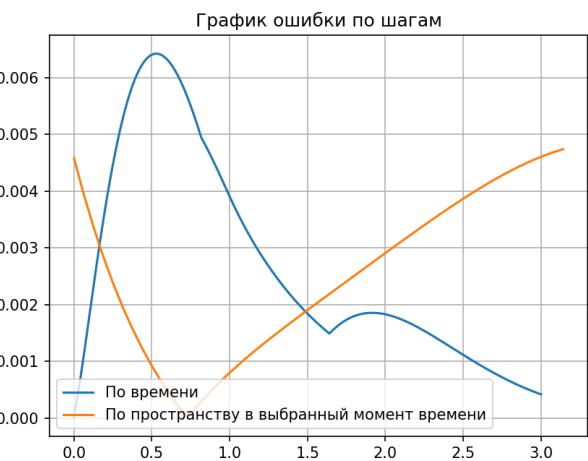
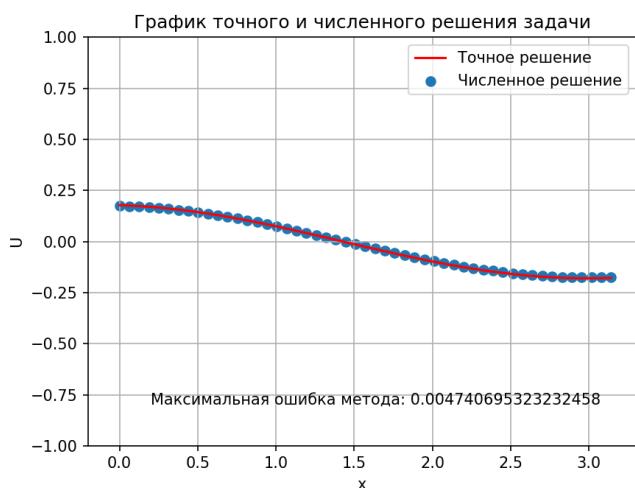


3) Двухточечная аппроксимация со вторым порядком

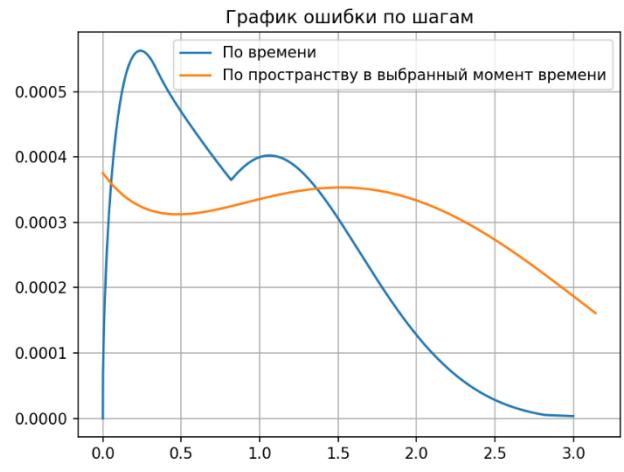
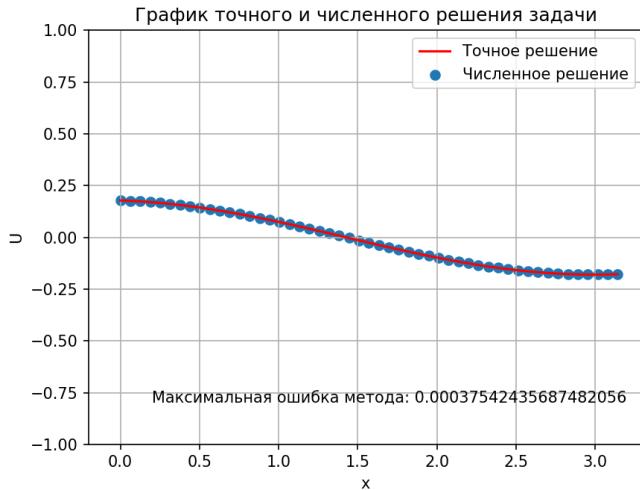


3. Метод Кранка-Никольсона:

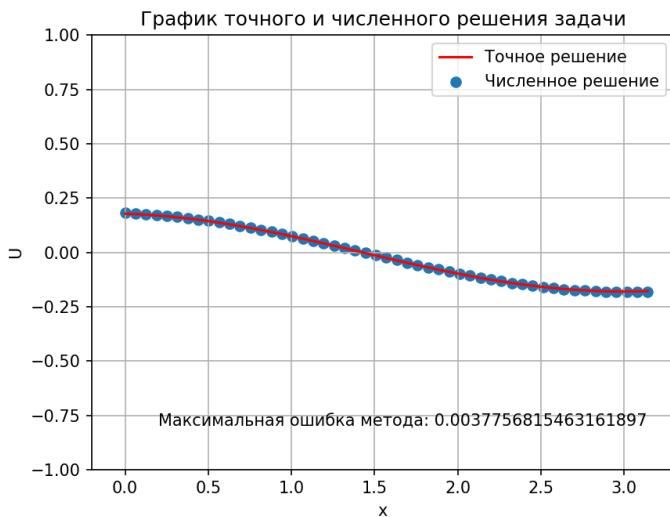
1) Двухточечная аппроксимация с первым порядком



2) Трёхточечная аппроксимация со вторым порядком



3) Двухточечная аппроксимация со вторым порядком



Листинг программы:

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

def analyt_func(x, a, b, c, t):
    return np.exp((c - a)* t) * np.sin(x + b * t)

def func_border1(a, b, c, t):
    return np.exp((c - a) * t) * (np.cos(b * t) + np.sin(b * t))

def func_border2(a, b, c, t):
    return -np.exp((c - a) * t) * (np.cos(b * t) + np.sin(b * t))

def run_through(a, b, c, d, s):
    P = np.zeros(s + 1)
    Q = np.zeros(s + 1)

    P[0] = -c[0] / b[0]
    Q[0] = d[0] / b[0]

    k = s - 1
    for i in range(1, s):
        P[i] = -c[i] / (b[i] + a[i] * P[i - 1])
        Q[i] = (d[i] - a[i] * Q[i - 1]) / (b[i] + a[i] * P[i - 1])
    P[k] = 0
    Q[k] = (d[k] - a[k] * Q[k - 1]) / (b[k] + a[k] * P[k - 1])

    x = np.zeros(s)
    x[k] = Q[k]

    for i in range(s - 2, -1, -1):
        x[i] = P[i] * x[i + 1] + Q[i]

    return x

def explicit(K, t, tau, h, a, b, c, x, approx):
    N = len(x)
    U = np.zeros((K, N))
    for j in range(N):
        U[0, j] = np.sin(x[j])

    for k in range(K - 1):
        t += tau
        for j in range(1, N - 1):
            U[k + 1, j] = tau * (a * (U[k, j - 1] - 2 * U[k, j] + U[k, j + 1]) / h**2 + b * (U[k, j + 1] - U[k, j - 1]) / (2 * h) \
```

```

        + c * U[k, j]) + U[k, j]

    if approx == 1:
        U[k + 1, 0] = (h * func_border1(a, b, c, t) - U[k + 1, 1]) / (h - 1)
        U[k + 1, N - 1] = (h * func_border2(a, b, c, t) + U[k + 1, N - 2]) / (h + 1)
    elif approx == 2:
        U[k + 1, 0] = (2 * h * func_border1(a, b, c, t) - 4 * U[k + 1, 1] + U[k + 1, 2]) / (2 * h
- 3)
        U[k + 1, N - 1] = (2 * h * func_border2(a, b, c, t) + 4 * U[k + 1, N - 2] - U[k + 1, N -
3]) / (2 * h + 3)
    elif approx == 3:
        U[k + 1, 0] = (func_border1(a, b, c, t) * h * tau * (2 - b * h) - U[k + 1, 1] * (2 * tau)
- U[k, 0] * h**2) / \
(-2 * tau - h**2 + c * tau * h**2 + h * tau * (2 - b * h))
        U[k + 1, N - 1] = (func_border2(a, b, c, t) * h * tau * (2 + b * h) + U[k + 1, N - 2] * (2
* tau) + U[k, N - 1] * h**2) / \
(2 * tau + h**2 - c * tau * h**2 + h * tau * (2 + b * h))

    return U


def implicit(K, t, tau, h, a1, b1, c1, x, approx):
    N = len(x)
    U = np.zeros((K, N))
    for j in range(N):
        U[0, j] = np.sin(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 * (a1 / h**2 - b1 / (2 * h))
            b[j] = tau * ((-2 * a1) / h**2 + c1) - 1
            c[j] = tau * (a1 / h**2 + b1 / (2 * h))
            d[j] = -U[k][j]

        if approx == 1:
            b[0] = 1 - 1 / h
            c[0] = 1 / h
            d[0] = func_border1(a1, b1, c1, t)

            a[N - 1] = -1 / h
            b[N - 1] = 1 + 1 / h
            d[N - 1] = func_border2(a1, b1, c1, t)
        elif approx == 2:
            k0 = 1 / (2 * h) / c[1]

```

```

        b[0] = (-3 / (2 * h)) + 1 + a[1] * k0
        c[0] = 2 / h + b[1] * k0
        d[0] = func_border1(a1, b1, c1, 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] = func_border2(a1, b1, c1, t) + d[N - 2] * k1
    elif approx == 3:
        b[0] = 2 * a1**2 / h + h / tau - c1 * h - (2 - b1 * h)
        c[0] = - 2 * a1**2 / h
        d[0] = (h / tau) * U[k - 1][0] - func_border1(a1, b1, c1, t) * (2 - b1 * h)

        a[N - 1] = -2 * a1**2 / h
        b[N - 1] = 2 * a1**2 / h + h / tau - c1 * h + (2 + b1 * h)
        d[N - 1] = (h / tau) * U[k - 1][N - 1] + func_border2(a1, b1, c1, t) * (2 + b1 * h)

    u_new = run_through(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, a1, b1, c1, x, approx, theta):
    N = len(x)
    if theta == 0:
        U = explicit(K, t, tau, h, a1, b1, c1, x, approx)
    elif theta == 1:
        U = implicit(K, t, tau, h, a1, b1, c1, x, approx)
    else:
        U_ex = explicit(K, t, tau, h, a1, b1, c1, x, approx)
        U_im = implicit(K, t, tau, h, a1, b1, c1, 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

def main(N, K, time):
    h = (np.pi - 0) / N
    tau = time / K
    x = np.arange(0, np.pi + h / 2 - 1e-4, h)
    T = np.arange(0, time, tau)
    a = 1
    b = 2
    c = -1

```

```

t = 0

while (1):
    print("Выберите метод:\n"
          "1 - явная конечно-разностная схема\n"
          "2 - неявная конечно-разностная схема\n"
          "3 - схема Кранка-Николсона\n"
          "0 - выход из программы")
    method = int(input())
    if method == 0:
        break
    else:
        print("Выберите уровень аппроксимации:\n"
              "1 - двухточечная аппроксимация с первым порядком\n"
              "2 - трехточечная аппроксимация со вторым порядком\n"
              "3 - двухточечная аппроксимация со вторым порядком")
        approx = int(input())

    if method == 1:
        if a * tau / h**2 <= 0.5:
            print("Условие Куррента выполнено:", a * tau / h**2, "<= 0.5\n")
            U = explicit(K, t, tau, h, a, b, c, x, approx)
        else:
            print("Условие Куррента не выполнено:", a * tau / h**2, "> 0.5")
            break
    elif method == 2:
        U = implicit(K, t, tau, h, a, b, c, x, approx)
    elif method == 3:
        theta = float(input("Введите параметр theta от 0 до 1:"))
        U = Krank_Nikolson(K, t, tau, h, a, b, c, x, approx, theta)

    dt = int(input("Введите момент времени:"))
    U_analytic = analyt_func(x, a, b, c, T[dt])
    error = abs(U_analytic - U[dt, :])
    plt.title("График точного и численного решения задачи")
    plt.plot(x, U_analytic, label = "Точное решение", color = "red")
    plt.scatter(x, U[dt, :], label = "Численное решение")
    plt.xlabel("x")
    plt.ylabel("U")
    plt.text(0.2, -0.8, "Максимальная ошибка метода: " + str(max(error)))
    plt.axis([-0.2, 3.3, -1, 1])
    plt.grid()
    plt.legend()
    plt.show()

    plt.title("График ошибки по шагам")
    error_time = np.zeros(len(T))
    for i in range(len(T)):

```

```

        error_time[i] = max(abs(analyt_func(x, a, b, c, T[i]) - U[i, :]))
    plt.plot(T, error_time, label = "По времени")
    plt.plot(x, error, label = "По пространству в выбранный момент времени")
    plt.legend()
    plt.grid()
    plt.show()

return 0

N = 50
K = 7000
time = 3
main(N, K, time)

```

Лабораторная работа 2

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 . Исследовать зависимость погрешности от сеточных параметров .

Реализованный метод	Решаемая задача
<ul style="list-style-type: none"> • Явная конечно-разностная схема («крест») • Неявная конечно-разностная схема • Неустойчивая конечно-разностная схема 	Начально-краевая задача для дифференциального уравнения гиперболического типа

10.

$$\frac{\partial^2 u}{\partial t^2} + 3 \frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial u}{\partial x} - u - \cos x \exp(-t),$$

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

$$u_x(\pi, t) = -\exp(-t),$$

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

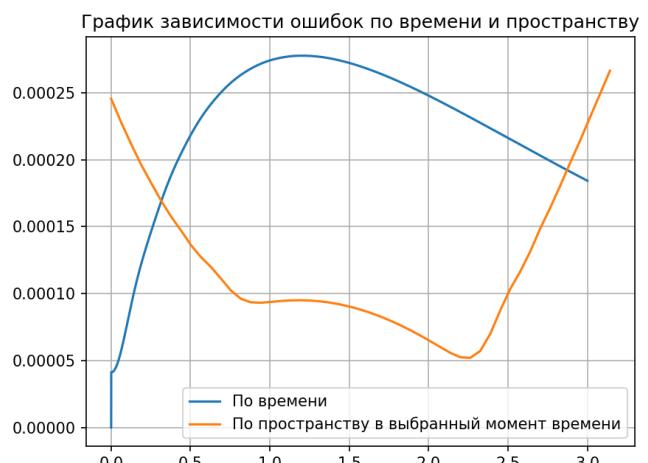
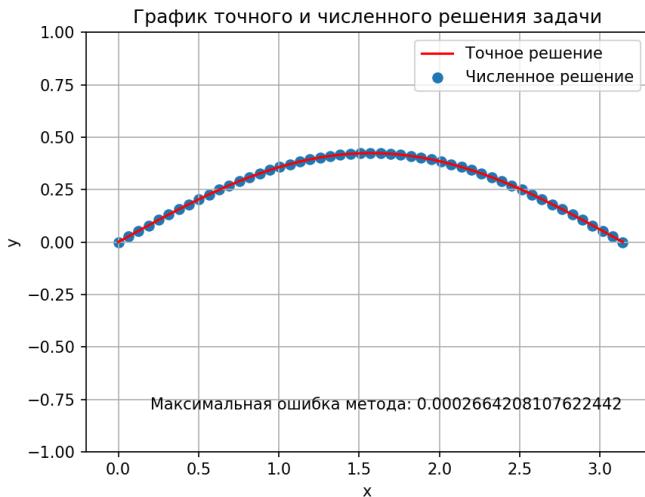
$$u_t(x, 0) = -\sin x.$$

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

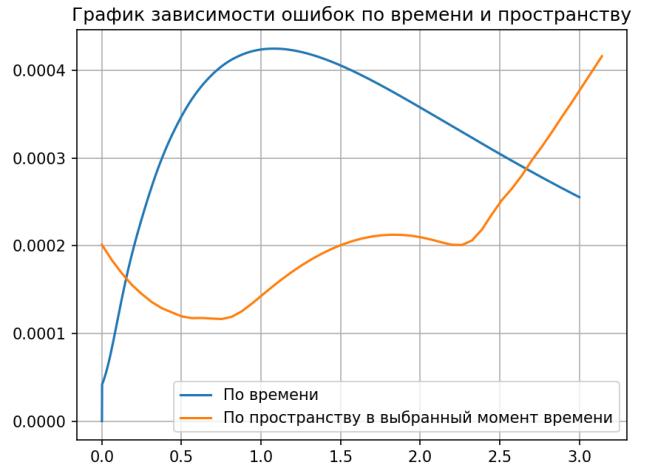
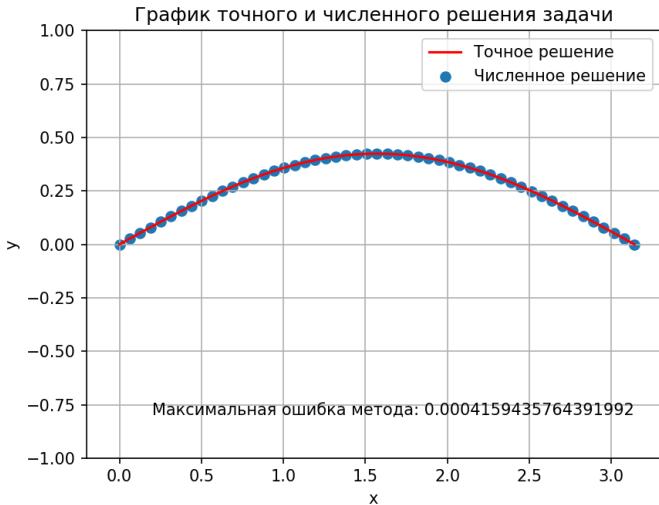
1. Явный метод

1) Двухточечная аппроксимация с первым порядком

1.1) Аппроксимация начальных условий с первым порядком

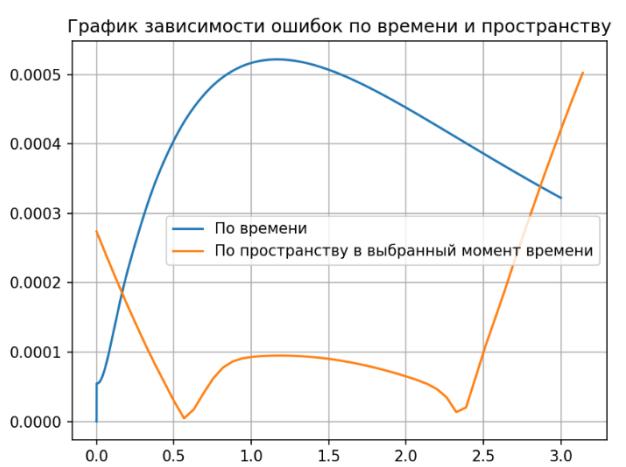
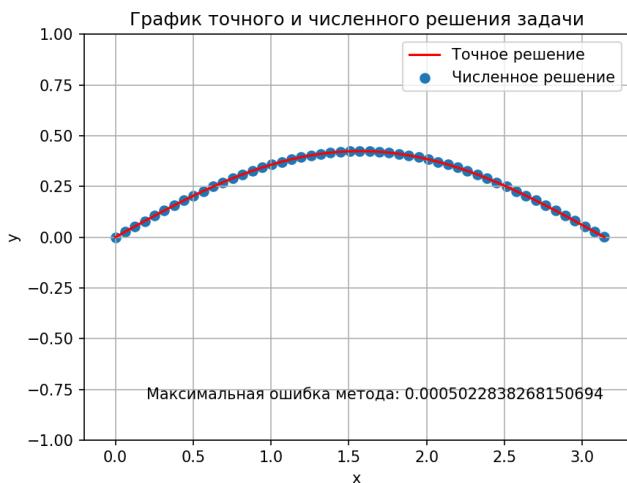


1.2) Аппроксимация начальных условий со вторым порядком

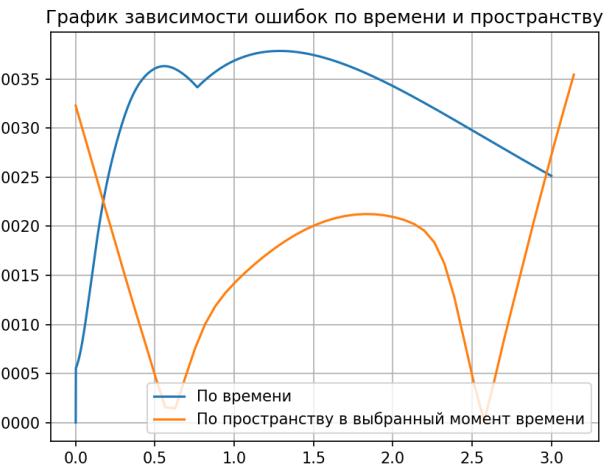
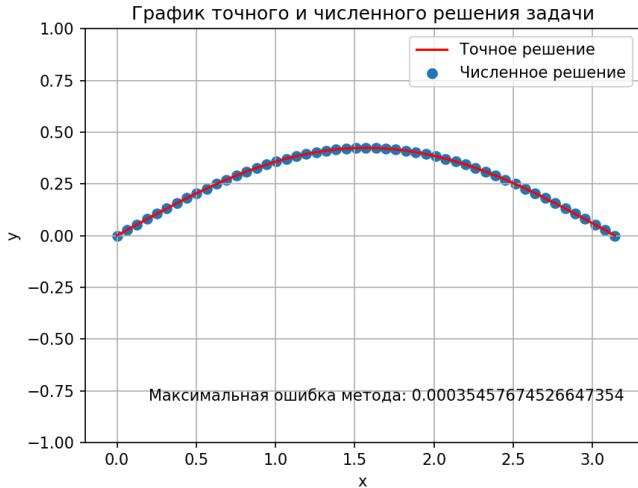


2) Трёхточечная аппроксимация со вторым порядком

2.1) Аппроксимация начальных условий с первым порядком

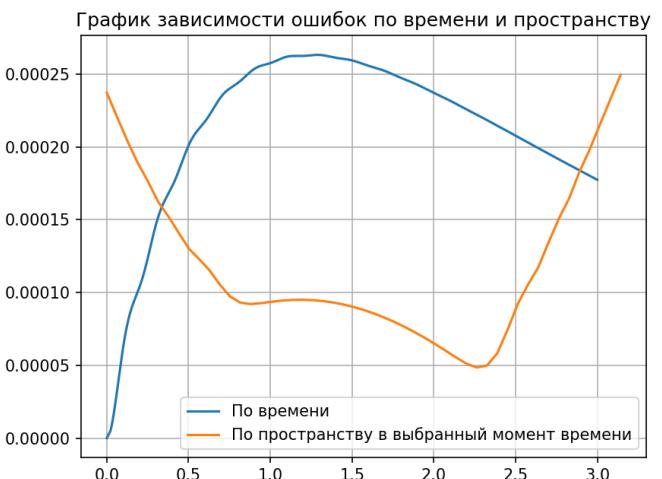
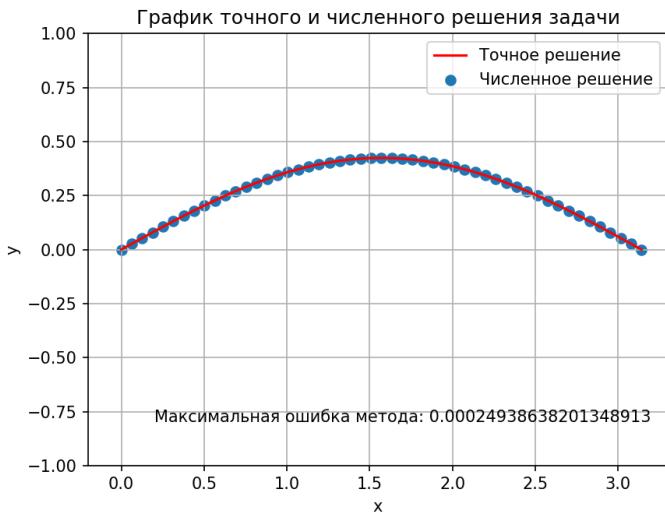


2.2) Аппроксимация начальных условий со вторым порядком

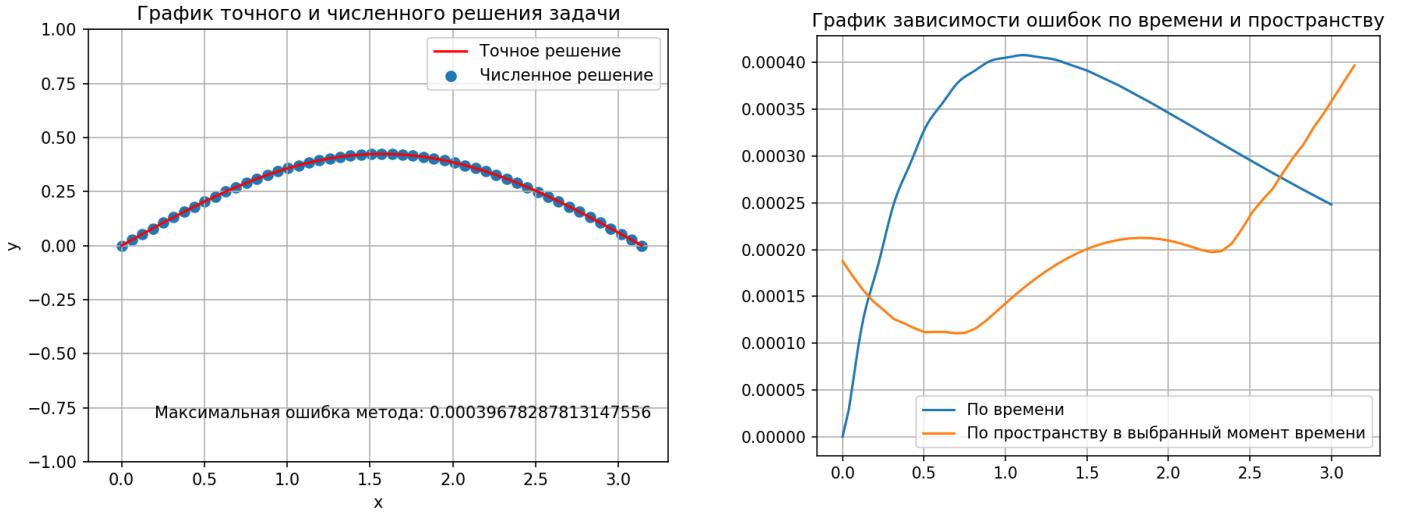


3) Двухточечная аппроксимация со вторым порядком

3.1) Аппроксимация начальных условий с первым порядком



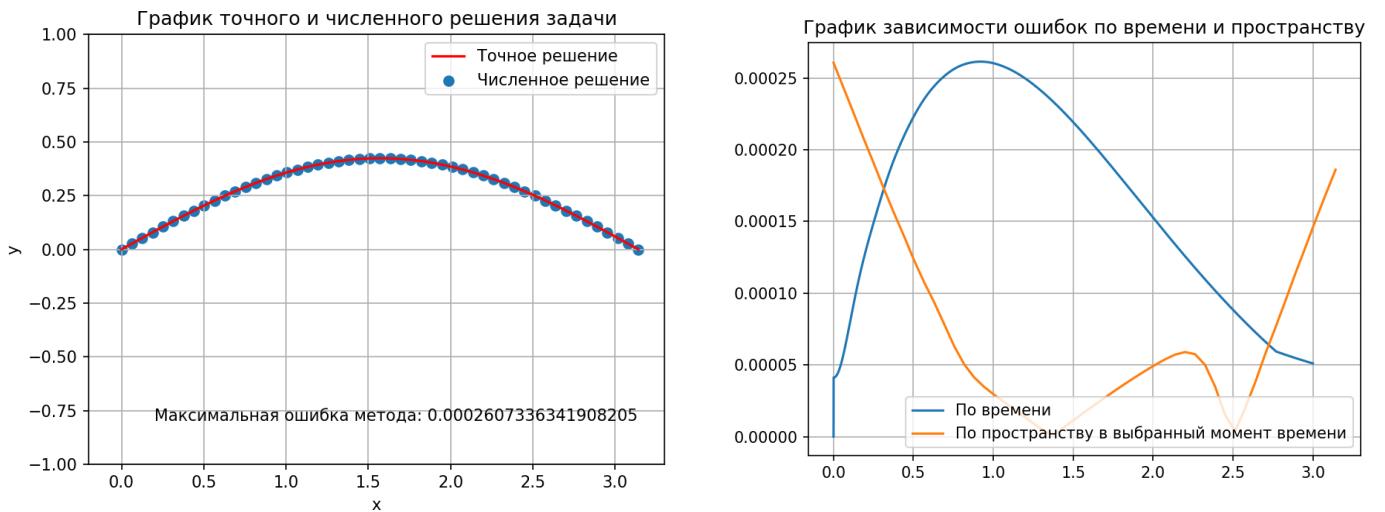
3.2) Аппроксимация начальных условий со вторым порядком



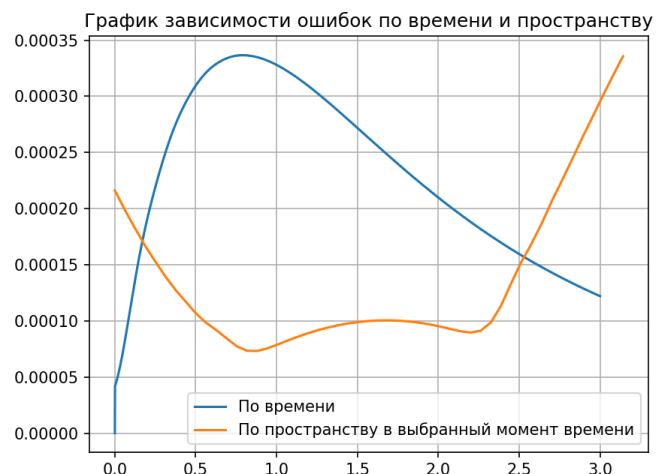
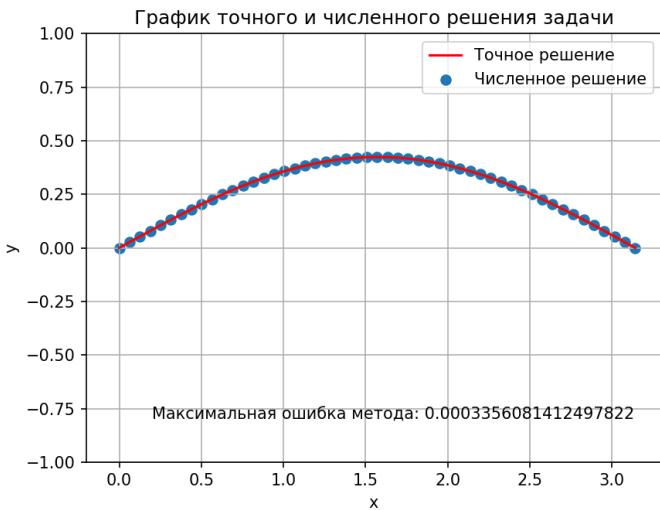
1. Неявный метод

1) Двухточечная аппроксимация с первым порядком

1.1) Аппроксимация начальных условий с первым порядком

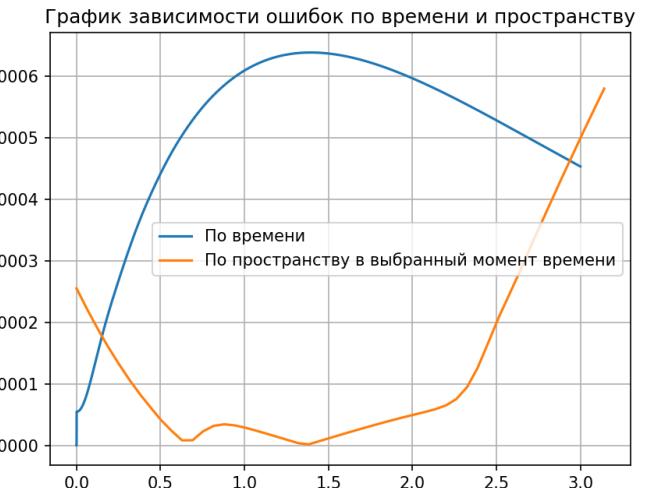
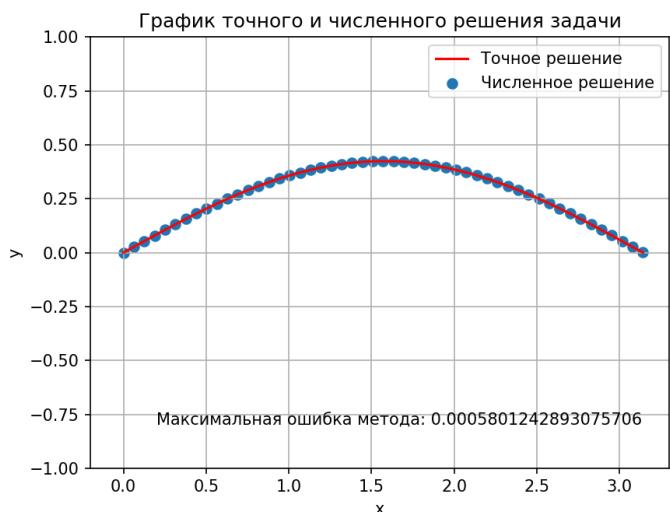


1.2) Аппроксимация начальных условий со вторым порядком

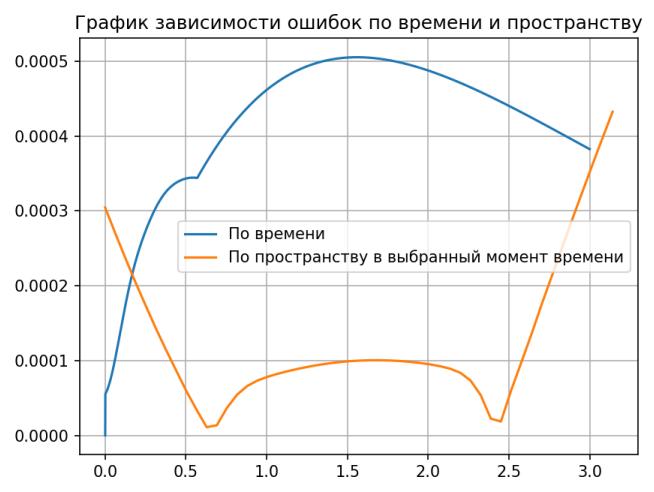
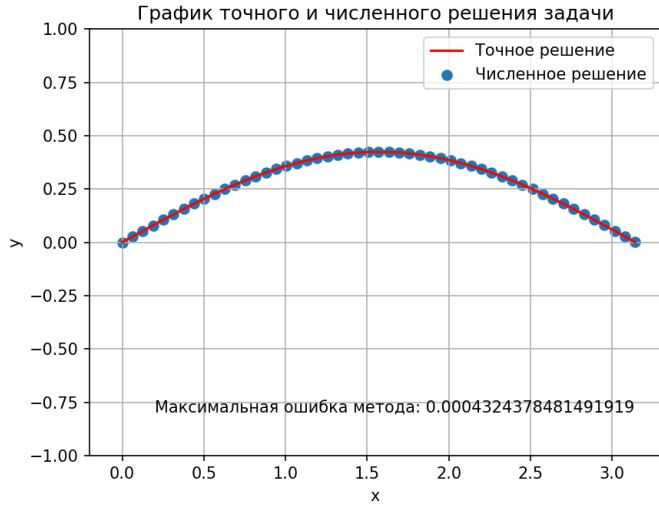


2) Трёхточечная аппроксимация со вторым порядком

2.1) Аппроксимация начальных условий с первым порядком

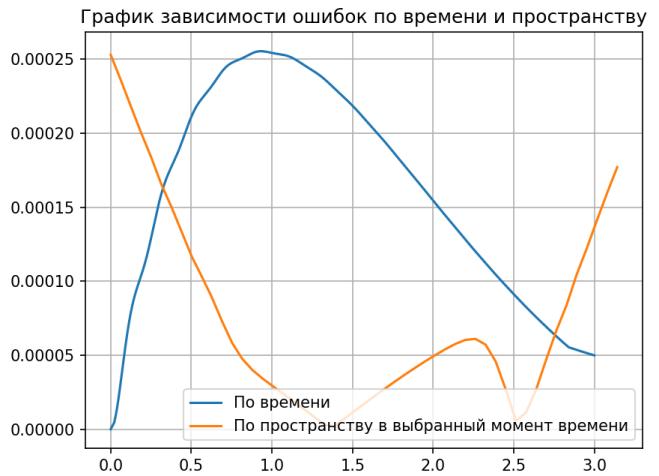
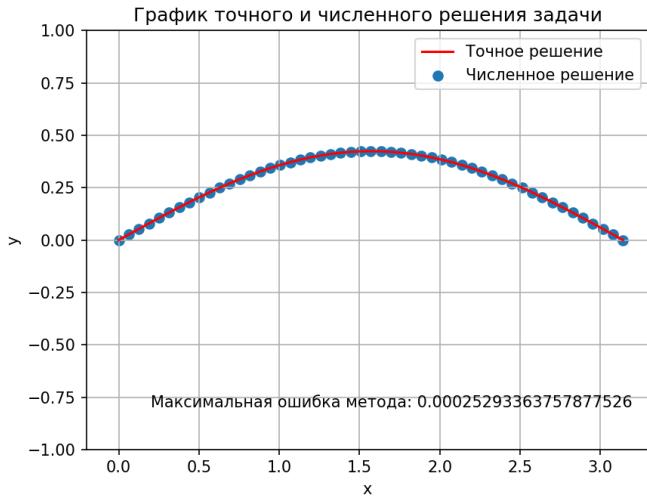


2.2) Аппроксимация начальных условий со вторым порядком

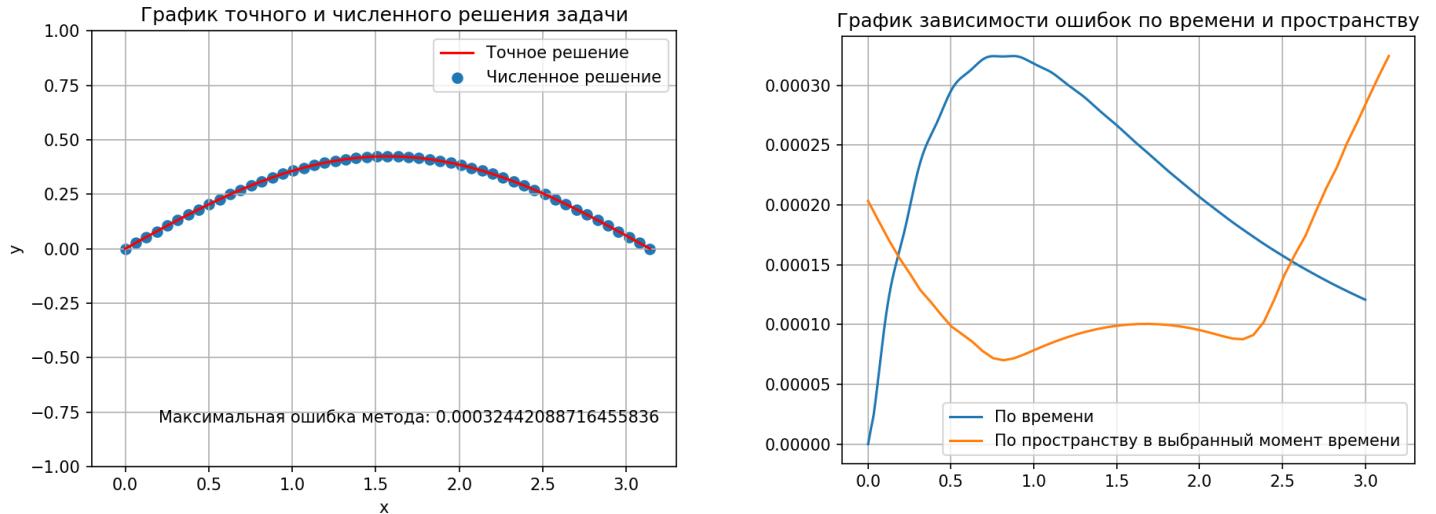


3) Двухточечная аппроксимация начальных условий со вторым порядком

3.1) Аппроксимация начальных условий с первым порядком



3.2) Аппроксимация начальных условий со вторым порядком



Листинг программы:

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

def analyt_func(x, t):
    return np.exp(-t) * np.sin(x)

def func(x, t):
    return np.cos(x) * np.exp(-t)

def func_border1(t):
    return np.exp(-t)

def func_border2(t):
    return -np.exp(-t)

def run_through(a, b, c, d, s):
    P = np.zeros(s + 1)
    Q = np.zeros(s + 1)

    P[0] = -c[0] / b[0]
    Q[0] = d[0] / b[0]

    k = s - 1
    for i in range(1, s):
        P[i] = -c[i] / (b[i] + a[i] * P[i - 1])
```

```

        Q[i] = (d[i] - a[i] * Q[i - 1]) / (b[i] + a[i] * P[i - 1])
        P[k] = 0
        Q[k] = (d[k] - a[k] * Q[k - 1]) / (b[k] + a[k] * P[k - 1])

    x = np.zeros(s)
    x[k] = Q[k]

    for i in range(s - 2, -1, -1):
        x[i] = P[i] * x[i + 1] + Q[i]

    return x


def explicit(K, t, tau, h, x, approx_st, approx_bo):
    N = len(x)
    U = np.zeros((K, N))
    t += tau

    for j in range(N):
        U[0, j] = np.sin(x[j])
        if approx_st == 1:
            U[1][j] = np.sin(x[j]) + -np.sin(x[j]) * tau
        if approx_st == 2:
            U[1][j] = np.sin(x[j]) + -np.sin(x[j]) * tau + (-2 * np.sin(x[j]) + np.cos(x[j]) +
func(x[j], t)) * tau**2 / 2

    for k in range(1, K - 1):
        t += tau
        for j in range(1, N - 1):
            U[k + 1, j] = ((2 + 3 * tau) * U[k, j] - U[k - 1, j] + \
tau**2 * ((U[k, j + 1] - 2 * U[k, j] + U[k, j - 1]) / h**2 + (U[k, j + 1] - U[k, j - 1]) / \
(2 * h) - U[k, j] - func(x[j], t))) \
            / (1 + 3 * tau)
            if approx_bo == 1:
                U[k + 1, 0] = -h * func_border1(t) + U[k + 1, 1]
                U[k + 1, N - 1] = h * func_border2(t) + U[k + 1, N - 2]
            elif approx_bo == 2:
                U[k + 1, 0] = (2 * h * func_border1(t) - 4 * U[k + 1, 1] + U[k + 1, 2]) / -3
                U[k + 1, N - 1] = (2 * h * func_border2(t) + 4 * U[k + 1, N - 2] - U[k + 1, N - 3]) / 3
            elif approx_bo == 3:
                U[k + 1, 0] = (2 * tau**2 * (h - h**2 / 2) * func_border1(t) - 2 * tau**2 * U[k + 1, 1] + \
tau**2 * h**2 * func(x[0], t) \
                - h**2 * (2 + 3 * tau) * U[k, 0] + h**2 * U[k - 1, 0]) / (-2 * tau**2 - tau**2 * h**2 - \
h**2 - 3 * h**2 * tau)
                U[k + 1, N - 1] = (2 * tau**2 * (h + h**2 / 2) * func_border2(t) + 2 * tau**2 * U[k + 1, N - 2] - \
tau**2 * h**2 * func(x[N - 1], t) \
                + h**2 * (2 + 3 * tau) * U[k, N - 1] - h**2 * U[k - 1, N - 1]) / (2 * tau**2 + tau**2 * \
h**2 + h**2 + 3 * h**2 * tau)

    return U

```

```

def implicit(K, t, tau, h, x, approx_st, approx_bo):
    N = len(x)
    U = np.zeros((K, N))
    t += tau
    for j in range(N):
        U[0, j] = np.sin(x[j])
        if approx_st == 1:
            U[1][j] = np.sin(x[j]) + -np.sin(x[j]) * tau
        if approx_st == 2:
            U[1][j] = np.sin(x[j]) + -np.sin(x[j]) * tau + (-2 * np.sin(x[j]) + np.cos(x[j]) +
func(x[j], t)) * tau**2 / 2

    for k in range(1, 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] = 1 / h**2 - 1 / (2 * h)
            b[j] = -2 / h**2 - 1 - 1 / tau**2 - 3 / tau
            c[j] = 1 / h**2 + 1 / (2 * h)
            d[j] = func(x[j], t) - (2 * U[k, j]) / tau**2 - (3 * U[k, j]) / tau + U[k - 1, j] / tau**2
        if approx_bo == 1:
            b[0] = -1 / h
            c[0] = 1 / h
            d[0] = func_border1(t)

            a[N - 1] = -1 / h
            b[N - 1] = 1 / h
            d[N - 1] = func_border2(t)
        elif approx_bo == 2:
            k0 = 1 / (2 * h) / c[1]
            b[0] = (-3 / (2 * h)) + a[1] * k0
            c[0] = 2 / h + b[1] * k0
            d[0] = func_border1(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)) + c[N - 2] * k1
            d[N - 1] = func_border2(t) + d[N - 2] * k1
        elif approx_bo == 3:
            b[0] = -1 - h**2 / 2 - h**2 / (2 * tau**2) - (3 * h**2) / (2 * tau)
            c[0] = 1
            d[0] = func_border1(t) * (h - h**2 / 2) + (func(x[0], t) * h**2) / 2 - (U[k, 0] * h**2) /
tau**2 \

```

```

+ (U[k - 1, 0] * h**2) / (2 * tau**2) - (U[k , 0] * 3 * h**2) / (2 * tau)

a[N - 1] = -1
b[N - 1] = 1 + h**2 / 2 + h**2 / (2 * tau**2) + (3 * h**2) / (2 * tau)
d[N - 1] = func_border2(t) * (h + h**2 / 2) - (func(x[N - 1], t) * h**2) / 2 + (U[k, N -
1] * h**2) / tau**2 \
- (U[k - 1, N - 1] * h**2) / (2 * tau**2) + (U[k , N - 1] * 3 * h**2) / (2 * tau)

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

return U


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

    while (1):
        print("Выберите метод:\n"
              "1 - явная конечно-разностная схема\n"
              "2 - неявная конечно-разностная схема\n"
              "0 - выход из программы")
        method = int(input())
        if method == 0:
            break
        else:
            print("Выберите уровень аппроксимации начальных условий:\n"
                  "1 - первого порядка\n"
                  "2 - второго порядка")
            approx_st = int(input())

            print("Выберите уровень аппроксимации краевых условий:\n"
                  "1 - двухточечная аппроксимация с первым порядком\n"
                  "2 - трехточечная аппроксимация со вторым порядком\n"
                  "3 - двухточечная аппроксимация со вторым порядком")
            approx_bo = int(input())

            if method == 1:
                if tau / h**2 <= 1:
                    print("Условие Куррента выполнено:", tau / h**2, "<= 1\n")
                    U = explicit(K, t, tau, h, x, approx_st, approx_bo)
                else:
                    print("Условие Куррента не выполнено:", tau / h**2, "> 1")
                    break
            else:
                print("Условие Куррента выполнено:", tau / h**2, "<= 1\n")
                U = implicit(K, t, tau, h, x, approx_st, approx_bo)

```

```

    elif method == 2:
        U = implicit(K, t, tau, h, x, approx_st, approx_bo)

        dt = int(input("Введите момент времени:"))
        U_analytic = analyt_func(x, T[dt])
        error = abs(U_analytic - U[dt, :])
        plt.title("График точного и численного решения задачи")
        plt.plot(x, U_analytic, label = "Точное решение", color = "red")
        plt.scatter(x, U[dt, :], label = "Численное решение")
        plt.xlabel("x")
        plt.ylabel("y")
        plt.text(0.2, -0.8, "Максимальная ошибка метода: " + str(max(error)))
        plt.axis([-0.2, 3.3, -1, 1])
        plt.grid()
        plt.legend()
        plt.show()

        plt.title("График зависимости ошибок по времени и пространству")
        error_time = np.zeros(len(T))
        for i in range(len(T)):
            error_time[i] = max(abs(analyt_func(x, T[i]) - U[i, :]))
        plt.plot(T, error_time, label = "По времени")
        plt.plot(x, error, label = "По пространству в выбранный момент времени")
        plt.legend()
        plt.grid()
        plt.show()

    return 0

N = 50
K = 7000
time = 3
main(N, K, time)

```

Лабораторная работа 3

Решить краевую задачу для дифференциального уравнения эллиптического типа. Аппроксимацию уравнения произвести с использованием центрально-разностной схемы. Для решения дискретного аналога применить следующие методы: метод простых итераций (метод Либмана), метод Зейделя, метод простых итераций с верхней релаксацией. Вычислить погрешность численного решения путем сравнения результатов с приведенным в задании аналитическим решением . Исследовать зависимость погрешности от сеточных параметров .

Реализованный метод	Решаемая задача
<ul style="list-style-type: none">• Метод простых итераций (метод Либмана)• Метод Зейделя	Краевая задача для дифференциального уравнения эллиптического типа

10.

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = -2 \frac{\partial u}{\partial x} - 2 \frac{\partial u}{\partial y} - 4u,$$

$$u(0, y) = \exp(-y) \cos y,$$

$$u\left(\frac{\pi}{2}, y\right) = 0,$$

$$u(x, 0) = \exp(-x) \cos x,$$

$$u\left(x, \frac{\pi}{2}\right) = 0.$$

Аналитическое решение: $U(x, y) = \exp(-x - y) \cos x \cos y$.

1. Метод Либмана

График точного и численного решения задачи

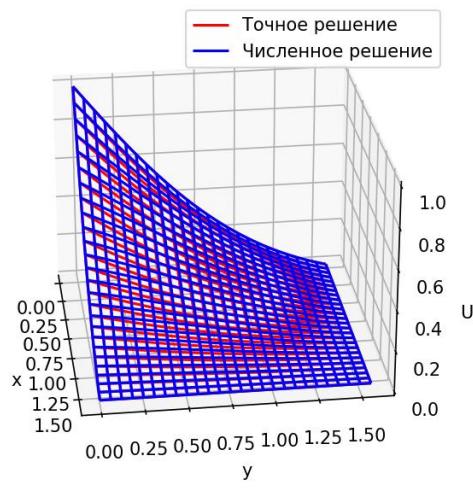
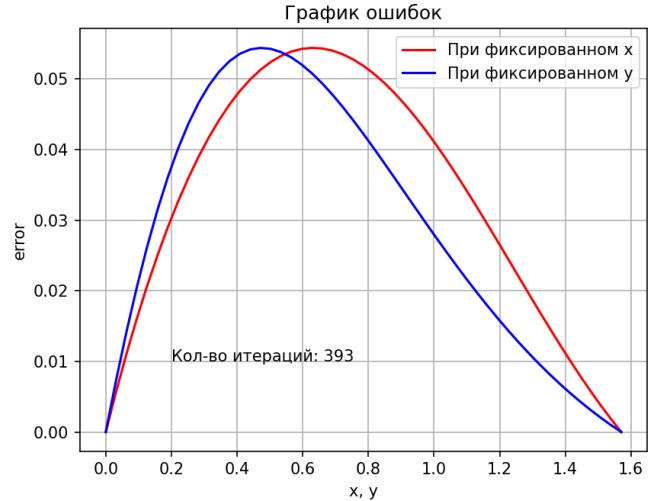


График ошибок



2. Метод Зейделя

График точного и численного решения задачи

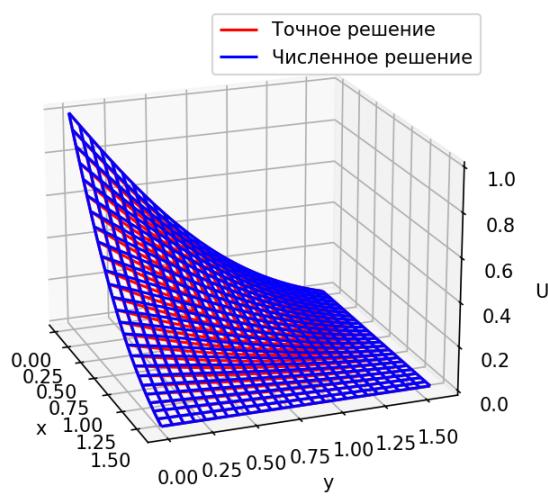
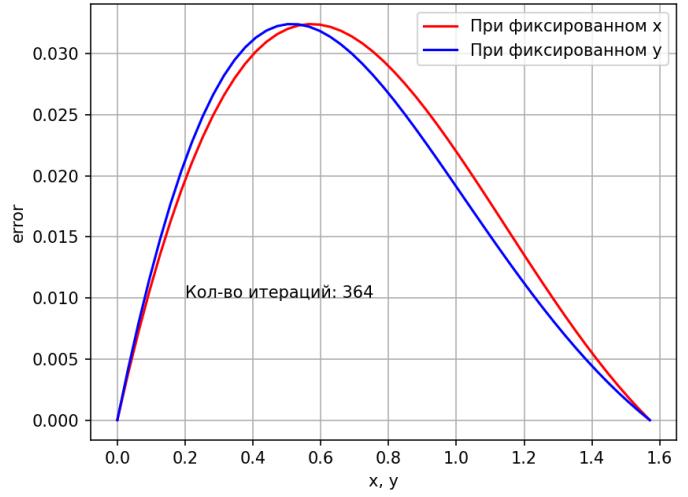
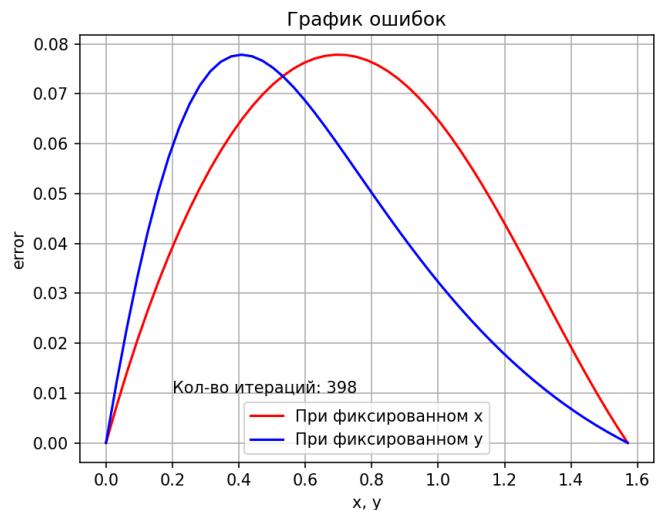
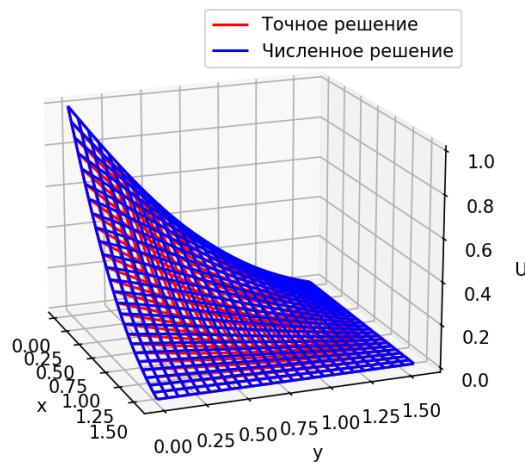


График ошибок



3. Метод простых итераций с верхней релаксацией

График точного и численного решения задачи



Листинг программы:

```
import numpy as np, matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d

def analyt_func(x, y):
    return np.exp(-x - y) * np.cos(x) * np.cos(y)

def func_border1(x, y):
    return np.exp(-y) * np.cos(y)

def func_border2(x, y):
    return 0

def func_border3(x, y):
    return np.exp(-x) * np.cos(x)

def func_border4(x, y):
    return 0

def norm(cur_u, prev_u):
    max = 0
    for i in range(cur_u.shape[0]):
```

```

        for j in range(cur_u.shape[1]):
            if abs(cur_u[i, j] - prev_u[i, j]) > max:
                max = abs(cur_u[i, j] - prev_u[i, j])

    return max


def liebman( x, y, h, eps):
    N = len(x)
    count = 0
    prev_u = np.zeros((N, N))
    cur_u = np.zeros((N, N))

    for j in range(len(y)):
        coeff = (func_border2(x[j], y[j]) - func_border1(x[j], y[j])) / (len(x) - 1)
        addition = func_border1(x[j], y[j])
        for i in range(len(x)):
            cur_u[i][j] = coeff * i + addition
    for i in range(1, N - 1):
        cur_u[i, 0] = func_border3(x[i], y[i])
        cur_u[i, -1] = func_border4(x[i], y[i])

    while norm(cur_u, prev_u) > eps:
        count += 1
        prev_u = np.copy(cur_u)
        for i in range(1, N - 1):
            for j in range(1, N - 1):
                cur_u[i, j] = ((1 + h) * (prev_u[i + 1, j] + prev_u[i, j + 1]) + (1 - h) * (prev_u[i - 1, j] + prev_u[i, j - 1]) + 4 * h**2 * prev_u[i, j]) / 4
        U = np.copy(cur_u)

    return U, count


def relaxation(x, y, h, eps, tau):
    N = len(x)
    count = 0
    prev_u = np.zeros((N, N))
    cur_u = np.zeros((N, N))
    for j in range(len(y)):
        coeff = (func_border2(x[j], y[j]) - func_border1(x[j], y[j])) / (len(x) - 1)
        addition = func_border1(x[j], y[j])
        for i in range(len(x)):
            cur_u[i][j] = coeff * i + addition
    for i in range(1, N - 1):
        cur_u[i, 0] = func_border3(x[i], y[i])

```

```

    cur_u[i, -1] = func_border4(x[i], y[i])

    while norm(cur_u, prev_u) > eps:
        count += 1
        prev_u = np.copy(cur_u)
        for i in range(1, N - 1):
            for j in range(1, N - 1):
                cur_u[i, j] = (1 - tau) * prev_u[i, j] + tau * (((1 + h) *
(prev_u[i + 1, j] + prev_u[i, j + 1]) + (1 - h) \
* (prev_u[i - 1, j] + prev_u[i, j - 1]) + 4 * h**2 * prev_u[i,
j]) / 4)
        U = np.copy(cur_u)

    return U, count

def Zeidel(x, y, h, eps, tau):
    N = len(x)
    count = 0
    prev_u = np.zeros((N, N))
    cur_u = np.zeros((N, N))
    for j in range(len(y)):
        coeff = (func_border2(x[j], y[j]) - func_border1(x[j], y[j])) /
(len(x) - 1)
        addition = func_border1(x[j], y[j])
        for i in range(len(x)):
            cur_u[i][j] = coeff * i + addition
    for i in range(1, N - 1):
        cur_u[i, 0] = func_border3(x[i], y[i])
        cur_u[i, -1] = func_border4(x[i], y[i])

    while norm(cur_u, prev_u) > eps:
        count += 1
        prev_u = np.copy(cur_u)

        for i in range(1, N - 1):
            for j in range(1, N - 1):
                cur_u[i, j] = (1 - tau) * prev_u[i, j] + tau * (((1 + h) *
(prev_u[i + 1, j] + prev_u[i, j + 1]) + (1 - h) \
* (cur_u[i - 1, j] + cur_u[i, j - 1]) + 4 * h**2 * prev_u[i,
j]) / 4)
        U = np.copy(cur_u)

    return U, count

def main(N, eps):
    h = (np.pi / 2 - 0) / N
    x = np.arange(0, np.pi / 2 + h / 2 - 1e-4, h)

```

```

y = np.arange(0, np.pi / 2 + h / 2 - 1e-4, h)

while (1):
    print("Выберите метод:\n"
          "1 - метод Либмана\n"
          "2 - метод Зейделя\n"
          "3 - метод простых итераций с верхней релаксацией\n"
          "0 - выход из программы")
    method = int(input())
    if method == 0:
        break
    if method == 1:
        U, count = liebman(x, y, h, eps)
    if method == 2:
        tau = float(input("Введите параметр tau от 0 до 1:"))
        U, count = Zeidel(x, y, h, eps, tau)
    if method == 3:
        tau = float(input("Введите параметр tau от 0 до 1:"))
        U, count = relaxation(x, y, h, eps, tau)

X, Y = np.meshgrid(x, y)
U_analytic = analyt_func(X, Y)
error_x = []
error_y = []
for i in range(len(x)):
    error_x.append(max(abs(U_analytic[:, i] - U[:, i])))
    error_y.append(max(abs(U_analytic[i, :] - U[i, :])))
plt.title("График ошибок")
plt.plot(x, error_y, label = "При фиксированном x", color = "red")
plt.plot(x, error_x, label = "При фиксированном y", color = "blue")
plt.text(0.2, 0.01, "Кол-во итераций: " + str(count))
plt.xlabel("x, y")
plt.ylabel("error")
plt.grid()
plt.legend()
fig = plt.figure()
ax = fig.add_subplot(projection='3d')
ax.set_title("График точного и численного решения задачи")
ax.plot_wireframe(X, Y, U_analytic, color = "red", label = "Точное
решение")
ax.plot_wireframe(X, Y, U, color = "blue", label = "Численное
решение")
ax.set_xlabel("x")
ax.set_ylabel("y")
ax.set_zlabel("U")
ax.legend()
plt.show()

```

```

    return 0

eps = 0.0001
N = 50
main(N, eps)

```

Лабораторная работа 4

Используя схемы переменных направлений и дробных шагов, решить двумерную начально-краевую задачу для дифференциального уравнения параболического типа. В различные моменты времени вычислить погрешность численного решения путем сравнения результатов с приведенным в задании аналитическим решением. Исследовать зависимость погрешности от сеточных параметров.

Реализованный метод	Решаемая задача
<ul style="list-style-type: none"> • Метод переменных направлений • Метод дробных шагов 	Двумерная начально-краевая задача для дифференциального уравнения параболического типа

10.

$$\frac{\partial u}{\partial t} = a \frac{\partial^2 u}{\partial x^2} + b \frac{\partial^2 u}{\partial y^2} + \sin x \sin y (\mu \cos \mu t + (a+b) \sin \mu t),$$

$$u(0, y, t) = 0,$$

$$u_x(\pi, y, t) = -\sin y \sin(\mu t),$$

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

$$u_y(x, \pi, t) = -\sin x \sin(\mu t),$$

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

Аналитическое решение: $U(x, y, t) = \sin x \sin y \sin(\mu t)$.

1). $a = 1, b = 1, \mu = 1$.

2). $a = 2, b = 1, \mu = 1$.

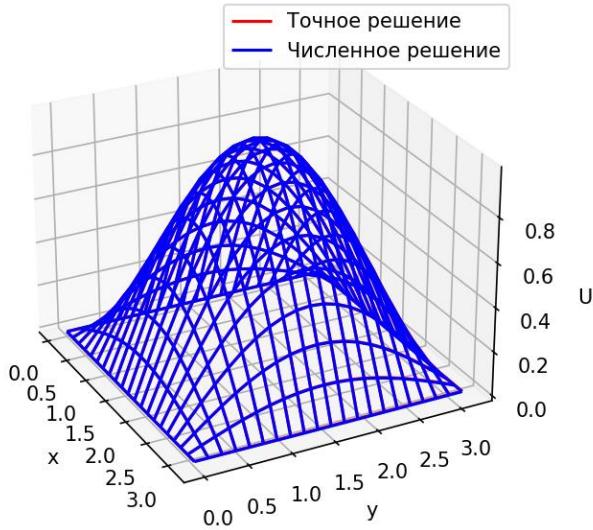
3). $a = 1, b = 2, \mu = 1$.

4). $a = 1, b = 1, \mu = 2$.

1. Метод переменных направлений

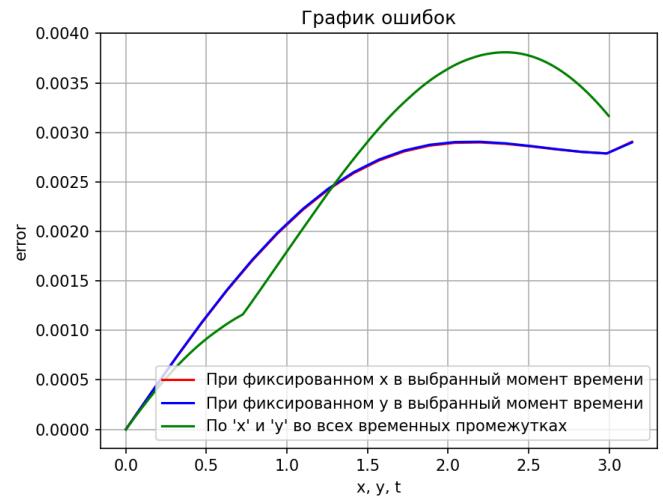
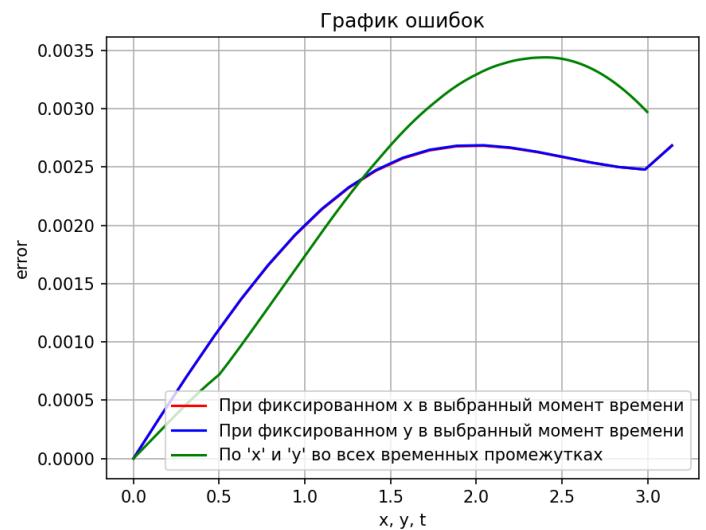
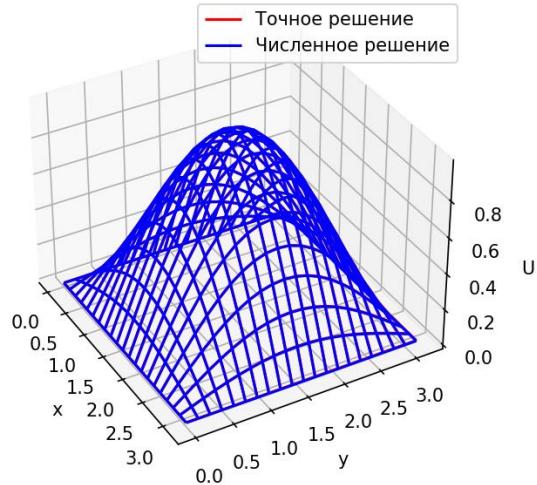
1) $a = 1, b = 1, \mu = 1$

График точного и численного решения задачи



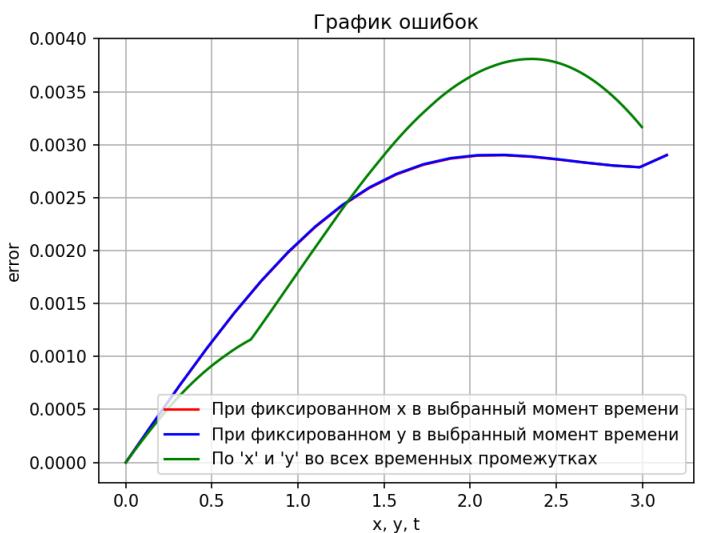
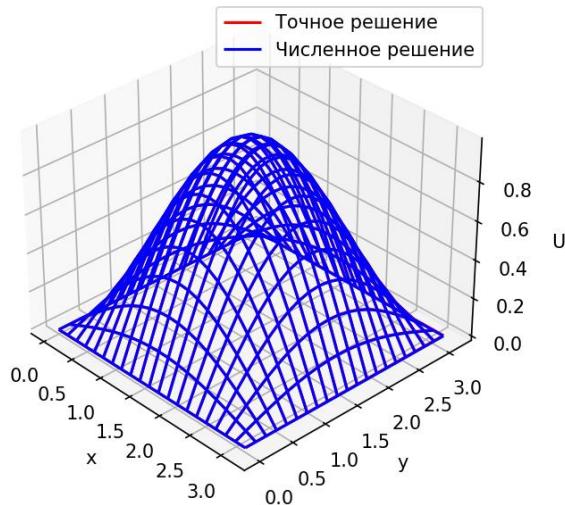
2) $a = 2, b = 1, \mu = 1$

График точного и численного решения задачи



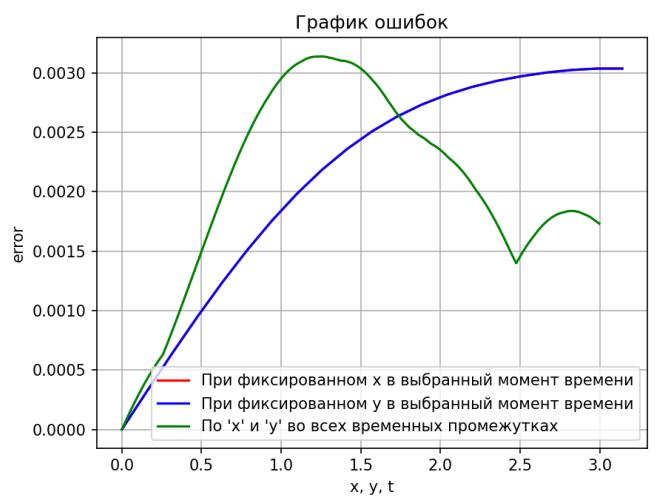
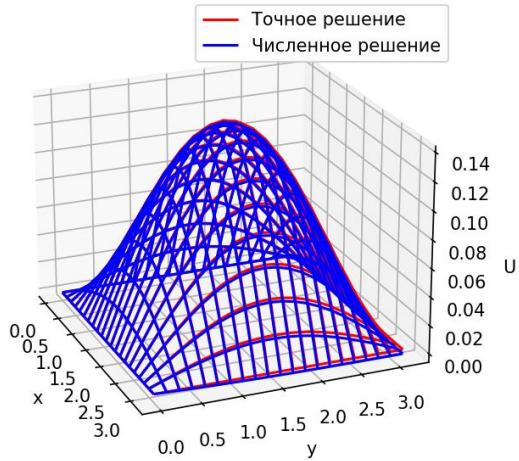
$$3) \quad a = 1, b = 2, \mu = 1$$

График точного и численного решения задачи



$$4) \quad a = 1, b = 1, \mu = 2$$

График точного и численного решения задачи



2. Метод дробных шагов

1) $a = 1, b = 1, \mu = 1$

График точного и численного решения задачи

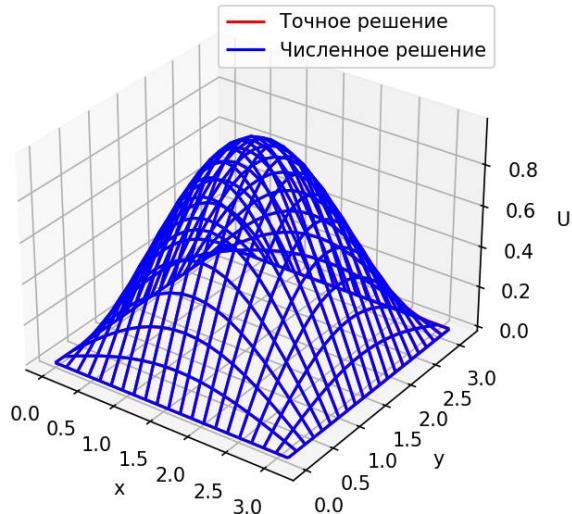
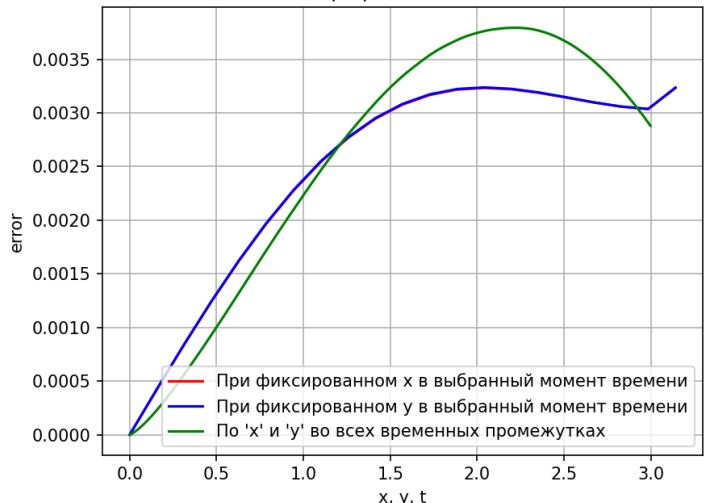


График ошибок



2) $a = 2, b = 1, \mu = 1$

График точного и численного решения задачи

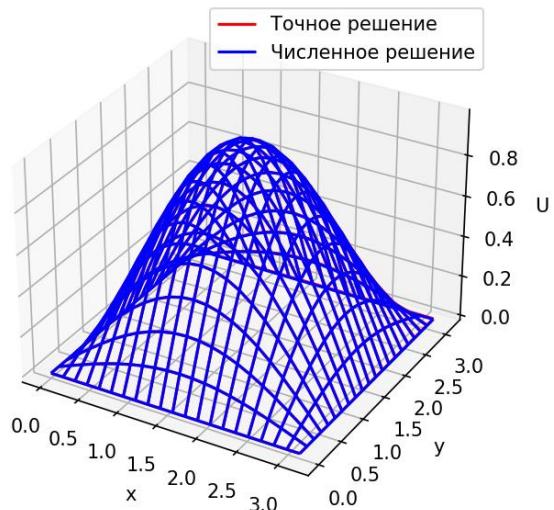
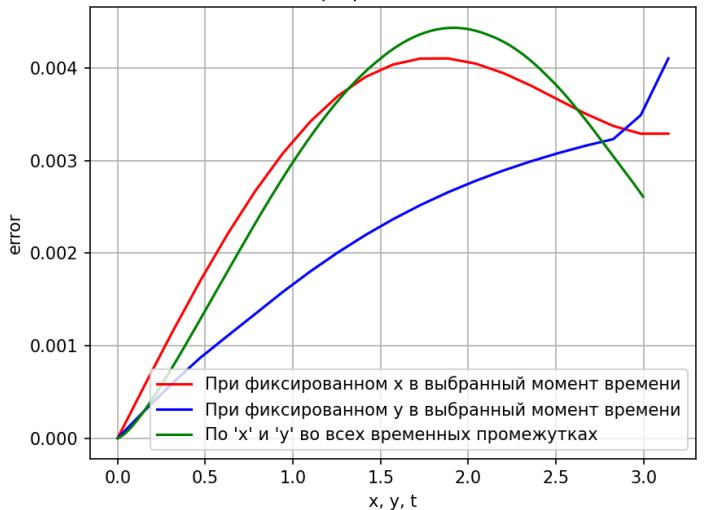
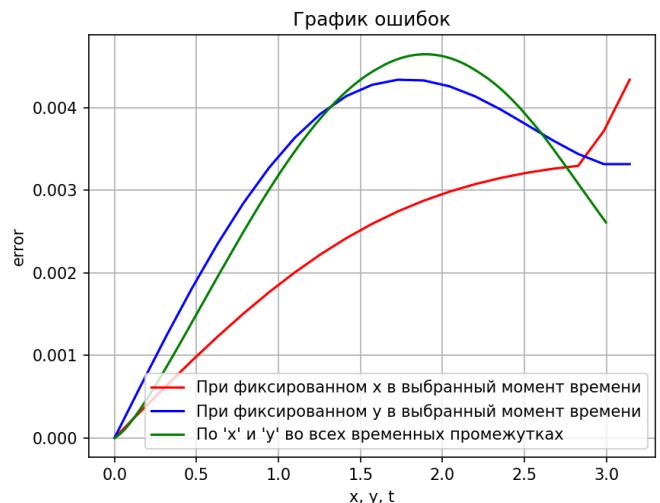
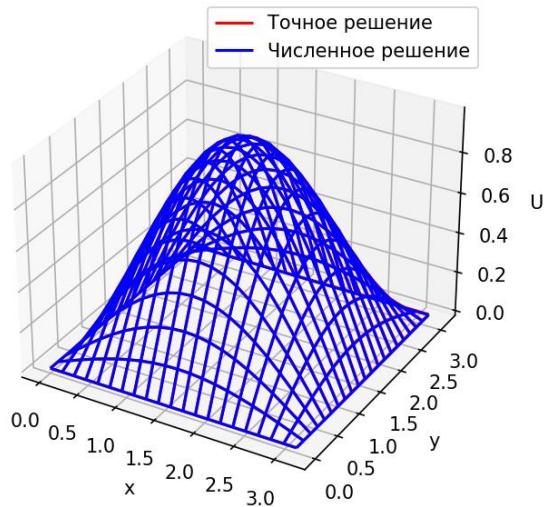


График ошибок



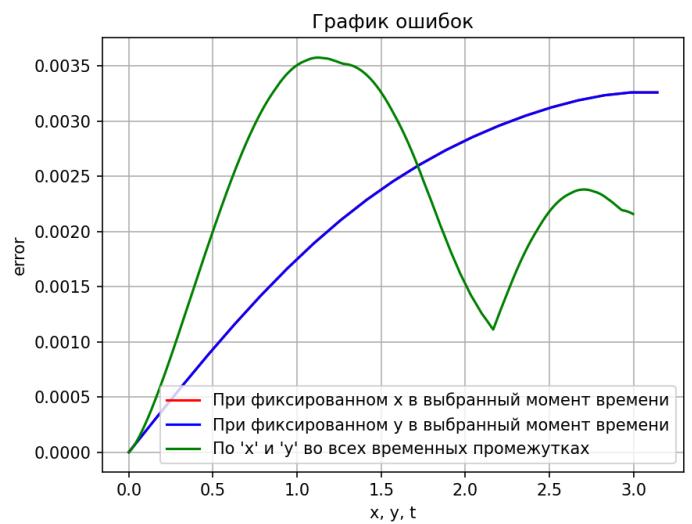
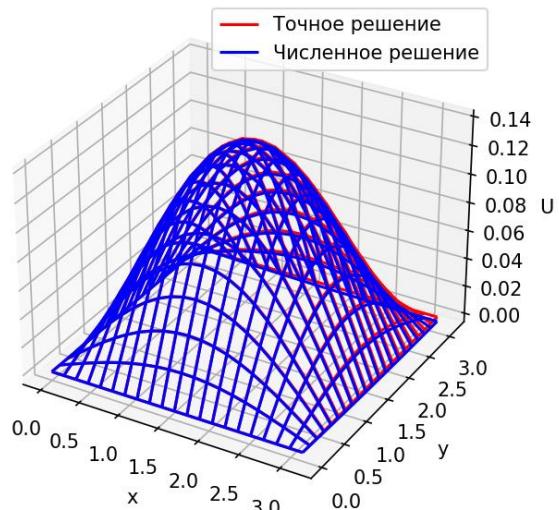
3) $a = 1, b = 2, \mu = 1$

График точного и численного решения задачи



4) $a = 1, b = 1, \mu = 2$

График точного и численного решения задачи



Листинг программы:

```
import numpy as np, matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d

def analyt_func(mu, x, y, t):
    return np.sin(x) * np.sin(y) * np.sin(mu * t)

def func(a, b, mu, x, y, t):
    return np.sin(x) * np.sin(y) * (mu * np.cos(mu * t) + (a + b) * np.sin(mu * t))

def func_border1(mu, y, t):
    return 0

def func_border2(mu, y, t):
    return -np.sin(y) * np.sin(mu * t)

def func_border3(mu, x, t):
    return 0

def func_border4(mu, x, t):
    return -np.sin(x) * np.sin(mu * t)

def norm(U1, U2):
    max = 0
    for i in range(U1.shape[0]):
        for j in range(U1.shape[1]):
            if abs(U1[i, j] - U2[i, j]) > max:
                max = abs(U1[i, j] - U2[i, j])

    return max

def run_through(a, b, c, d, s):
    P = np.zeros(s + 1)
    Q = np.zeros(s + 1)

    P[0] = -c[0] / b[0]
    Q[0] = d[0] / b[0]

    k = s - 1
    for i in range(1, s):
        P[i] = -c[i] / (b[i] + a[i] * P[i - 1])
```

```

        Q[i] = (d[i] - a[i] * Q[i - 1]) / (b[i] + a[i] * P[i - 1])
        P[k] = 0
        Q[k] = (d[k] - a[k] * Q[k - 1]) / (b[k] + a[k] * P[k - 1])

    x = np.zeros(s)
    x[k] = Q[k]

    for i in range(s - 2, -1, -1):
        x[i] = P[i] * x[i + 1] + Q[i]

    return x

def variable_directions(a1, b1, mu, x, y, hx, hy, K, tau, t):
    U = np.zeros((K, len(x), len(y)))
    sigma_a = (a1 * tau) / (2 * hx**2)
    sigma_b = (b1 * tau) / (2 * hy**2)

    for k in range(K):
        for i in range(len(x)):
            U[k, i, 0] = func_border3(mu, x[i], t)
            U[k, i, -1] = func_border4(mu, x[i], t) * hy + U[k, i, -2]
    for k in range(K):
        for j in range(len(y)):
            U[k, 0, j] = func_border1(mu, y[j], t)
            U[k, -1, j] = func_border2(mu, y[j], t) * hx + U[k, -2, j]

    for k in range(1, K):
        U_temp = np.zeros((len(x), len(y)))
        a = np.zeros(len(y))
        b = np.zeros(len(y))
        c = np.zeros(len(y))
        d = np.zeros(len(y))
        t += tau / 2
        for i in range(1, len(x) - 1):
            for j in range(1, len(y) - 1):
                a[j] = -sigma_a
                b[j] = 1 + 2 * sigma_a
                c[j] = -sigma_a
                d[j] = (func(a1, b1, mu, x[i], y[j], t) * tau) / 2 + sigma_b * (U[k - 1, i, j + 1] - 2
* U[k - 1, i, j] + U[k - 1, i, j - 1]) \
                    + U[k - 1, i, j]
                b[0] = 1
                c[0] = 0
                d[0] = func_border3(mu, x[i], t)
                a[-1] = -1
                b[-1] = 1
                d[-1] = func_border4(mu, x[i], t) * hy
                u_new = run_through(a, b, c, d, len(d))

```

```

        U_temp[i] = u_new
        for j in range(len(y)):
            U_temp[0, j] = func_border1(mu, y[j], t)
            U_temp[-1, j] = func_border2(mu, y[j], t) * hx + U_temp[-2, j]

        a = np.zeros(len(x))
        b = np.zeros(len(x))
        c = np.zeros(len(x))
        d = np.zeros(len(x))
        t += tau / 2
        for j in range(1, len(y) - 1):
            for i in range(1, len(x) - 1):
                a[i] = -sigma_b
                b[i] = 1 + 2 * sigma_b
                c[i] = -sigma_b
                d[i] = (func(a1, b1, mu, x[i], y[j], t) * tau) / 2 + sigma_a * (U_temp[i + 1, j] - 2 *
U_temp[i, j] + U_temp[i - 1, j]) \
                    + U_temp[i, j]
            b[0] = 1
            c[0] = 0
            d[0] = func_border1(mu, y[j], t)
            a[-1] = -1
            b[-1] = 1
            d[-1] = func_border2(mu, y[j], t) * hx
            u_new = run_through(a, b, c, d, len(d))
            for i in range(len(u_new)):
                U[k, i, j] = u_new[i]
            for i in range(len(x)):
                U[k, i, 0] = func_border3(mu, x[i], t)
                U[k, i, -1] = func_border4(mu, x[i], t) * hy + U[k, i, -2]

        return U.transpose()

def fractional_step(a1, b1, mu, x, y, hx, hy, K, tau, t):
    U = np.zeros((K, len(x), len(y)))
    sigma_a = (a1 * tau) / hx**2
    sigma_b = (b1 * tau) / hy**2

    for k in range(K):
        for i in range(len(x)):
            U[k, i, 0] = func_border3(mu, x[i], t)
            U[k, i, -1] = func_border4(mu, x[i], t) * hy + U[k, i, -2]
    for k in range(K):
        for j in range(len(y)):
            U[k, 0, j] = func_border1(mu, y[j], t)
            U[k, -1, j] = func_border2(mu, y[j], t) * hx + U[k, -2, j]

    for k in range(1, K):

```

```

U_temp = np.zeros((len(x), len(y)))
a = np.zeros(len(y))
b = np.zeros(len(y))
c = np.zeros(len(y))
d = np.zeros(len(y))
t += tau / 2
for i in range(1, len(x) - 1):
    for j in range(1, len(y) - 1):
        a[j] = -sigma_a
        b[j] = 1 + 2 * sigma_a
        c[j] = -sigma_a
        d[j] = (func(a1, b1, mu, x[i], y[j], t) * tau) / 2 + U[k - 1, i, j]
    b[0] = 1
    c[0] = 0
    d[0] = func_border3(mu, x[i], t)
    a[-1] = -1
    b[-1] = 1
    d[-1] = func_border4(mu, x[i], t) * hy
    u_new = run_through(a, b, c, d, len(d))
    U_temp[i] = u_new
    for j in range(len(y)):
        U_temp[0, j] = func_border1(mu, y[j], t)
        U_temp[-1, j] = func_border2(mu, y[j], t) * hx + U_temp[-2, j]

a = np.zeros(len(x))
b = np.zeros(len(x))
c = np.zeros(len(x))
d = np.zeros(len(x))
t += tau / 2
for j in range(1, len(y) - 1):
    for i in range(1, len(x) - 1):
        a[i] = -sigma_b
        b[i] = 1 + 2 * sigma_b
        c[i] = -sigma_b
        d[i] = (func(a1, b1, mu, x[i], y[j], t) * tau) / 2 + U_temp[i, j]
    b[0] = 1
    c[0] = 0
    d[0] = func_border1(mu, y[j], t)
    a[-1] = -1
    b[-1] = 1
    d[-1] = func_border2(mu, y[j], t) * hx
    u_new = run_through(a, b, c, d, len(d))
    for i in range(len(u_new)):
        U[k, i, j] = u_new[i]
    for i in range(len(x)):
        U[k, i, 0] = func_border3(mu, x[i], t)
        U[k, i, -1] = func_border4(mu, x[i], t) * hy + U[k, i, -2]

```

```

return U.transpose()

def main(Nx, Ny, K, time):
    hx = (np.pi - 0) / Nx
    hy = (np.pi - 0) / Ny
    x = np.arange(0, np.pi + hx / 2 - 1e-4, hx)
    y = np.arange(0, np.pi + hy / 2 - 1e-4, hy)
    tau = time / K
    T = np.arange(0, time, tau)
    t = 0

    while (1):
        print("Выберите метод:\n"
              "1 - метод переменных направлений\n"
              "2 - метод дробных шагов\n"
              "0 - выход из программы")
        method = int(input())
        if method == 0:
            break
        print("Выберите параметры:\n"
              "1 - a = 1, b = 1, mu = 1\n"
              "2 - a = 2, b = 1, mu = 1\n"
              "3 - a = 2, b = 2, mu = 1\n"
              "4 - a = 1, b = 1, mu = 2")
        param = int(input())
        if param == 1:
            a, b, mu= 1, 1, 1
        elif param == 2:
            a, b, mu= 2, 1, 1
        elif param == 3:
            a, b, mu= 1, 2, 1
        elif param == 4:
            a, b, mu= 1, 1, 2
        if method == 1:
            U = variable_directions(a, b, mu, x, y, hx, hy, K, tau, t)
        if method == 2:
            U = fractional_step(a, b, mu, x, y, hx, hy, K, tau, t)

        dt = int(input("Введите момент времени:"))
        X, Y = np.meshgrid(x, y)
        U_analytic = analytic_func(mu, X, Y, T[dt])
        print(U[:, :, 0].shape)
        print(U_analytic.shape)
        error_x = []
        error_y = []
        error_t = []
        for i in range(len(x)):
            error_y.append(max(abs(U_analytic[:, i] - U[:, i, dt])))

```

```

for j in range(len(y)):
    error_x.append(max(abs(U_analytic[j, :] - U[j, :, dt])))
for k in range(K):
    error_t.append(norm(analyt_func(mu, X, Y, T[k]), U[:, :, k]))
plt.title("График ошибок")
plt.plot(x, error_y, label = "При фиксированном x в выбранный момент времени", color = "red")
plt.plot(y, error_x, label = "При фиксированном y в выбранный момент времени", color = "blue")
plt.plot(T, error_t, label = "По 'x' и 'y' во всех временных промежутках", color = "green")
plt.xlabel("x, y, t")
plt.ylabel("error")
plt.grid()
plt.legend()

fig = plt.figure()
ax = fig.add_subplot(projection='3d')
ax.set_title("График точного и численного решения задачи")
ax.plot_wireframe(X, Y, U_analytic, color = "red", label = "Точное решение")
ax.plot_wireframe(X, Y, U[:, :, dt], color = "blue", label = "Численное решение")
ax.set_xlabel("x")
ax.set_ylabel("y")
ax.set_zlabel("U")
ax.legend()
plt.show()

return 0

Nx = 20
Ny = 20
K = 1000
time = 3
main(Nx, Ny, K, time)

```