Московский авиационный институт

(Национальный исследовательский университет)

Институт №8 «Компьютерные науки и прикладная математика»

Кафедра вычислительной математики и программирования

Лабораторные работы

по курсу «Численные методы»

Вариант 5

Выполнил: Черных С. Д.

Группа: М8О-405Б-20

Проверил: доц. Иванов И. Э.

Дата:

Оценка:

Москва, 2023

Лабораторная работа №5

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 сеточного параметра h.

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \sin(\pi x)$$

$$u(0,t) = 0$$
$$u(1,t) = 0$$
$$u(x,0) = 0$$

Аналитическое решение: $U(x,t) = \frac{1}{\pi^2}(1 - \exp(-\pi^2 t))\sin(\pi x)$

Теоретические сведения

Конечно-разностная схема

Будем решать задачу на заданном промежутке от 0 до l по координате x и на промежутке от 0 до заданного параметра T по времени t.

Рассмотрим конечно-разностную схему решения краевой задачи на сетке с граничными параметрами l, T и параметрами насыщенности сетки N, K. Тогда размер шага по каждой из координат определяется:

$$h = \frac{l}{N}, \tau = \frac{T}{K}$$

Однако, учитывая условие устойчивости $\sigma = \frac{\alpha^2 \tau}{h^2} \leq \frac{1}{2}$ для явной конечно-разностной схемы, $\tau \leq \frac{h^2}{2\alpha^2}$, в данной лабораторной работе был взят $\tau = \frac{h^2}{2\alpha^2}$

Считая, что значения функции $u_j^k = u(x_j, t^k)$ для всех координат $x_j = jh, \forall j \in \{0, .., N\}$ на временном слое $t^k = \mathrm{k}\tau, \mathrm{k} \in \{0, .., \mathrm{K} - 1\}$ известны, попробуем определить значения функции на временном слое t^{k+1} путем разностной аппроксимации производной:

$$\frac{\partial u}{\partial t}\left(x_j, t^k\right) = \frac{u_j^{k+1} - u_j^k}{\tau}$$

И одним из методов аппроксимации второй производной по x:

$$\frac{\partial^2 u}{\partial x^2}\left(x_j, t^k\right)$$

Явная конечно-разностная схема

Аппроксимируем вторую производную по значениям нижнего временного слоя t^k:

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

Тогда получим явную схему конечно-разностного метода во внутренних узлах сетки:

$$\frac{u_j^{k+1} - u_j^k}{\tau} = a\frac{u_{j-1}^k - 2u_j^k + u_{j+1}^k}{h^2} + b\frac{u_{i+1}^k + u_{i-1}^k}{2h} + c\,u^k + f_j^k$$

$$\forall \mathrm{k} \in \{0, .., \mathrm{K}-1\}$$

$$\forall \mathrm{j} \in \{1, .., \mathrm{N}-1\}$$

Обозначим $\sigma = \frac{a^2\tau}{h^2}$, тогда:

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

Граничные значения u_0^{k+1} и u_N^{k+1}определяются граничными условиями

$$u_0^{k+1} = -\frac{\alpha/h}{\beta - \frac{\alpha}{h}}u_1^{k+1} + \frac{\varphi_0(t^{k+1})}{\beta - \frac{\alpha}{h}}, \qquad u_N^{k+1} = \frac{\gamma/h}{\delta + \frac{\gamma}{h}}u_{N-1}^{k+1} + \frac{\varphi_l(t^{k+1})}{\delta + \frac{\gamma}{h}}$$

где $\alpha = 0, \beta = 1, \gamma = 0,\ \delta = 1$.

Явная схема является условно устойчивой, с условием $\sigma \le \frac{1}{2}$.

Неявная конечно-разностная схема

Аппроксимируем вторую производную по значениям верхнего временного слоя t^{k+1}:

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

Тогда получим явную схему конечно-разностного метода во внутренних узлах сетки:

$$\frac{u_j^{k+1} - u_j^k}{\tau} = a\,\frac{u_{j-1}^{k+1} - 2u_j^{k+1} + u_{j+1}^{k+1}}{h^2} + b\,\frac{u_{j+1}^{k+1} + u_{j-1}^{k+1}}{2h} + c\,u_j^k + f_j^{k+1}$$

$\forall \mathrm{j} \in \{1,..,\mathrm{N}-1\}, \forall \mathrm{k} \in \{0,..,\mathrm{K}-1\}$.

Обозначим $\sigma = \frac{a^2\tau}{h^2}$,. Тогда значения функции на слое можно найти эффективным образом с помощью методом прогонки, где СЛАУ, кроме крайних двух уравнений, определяется коэффициентами, в данном варианте,

$a_j = \sigma, b_j = -(1+2\sigma), c_j = \sigma, d_j = -u_j^k - \tau f_j^k$ уравнений:

$$a_j u_{j-1}^{k+1} + b_j u_j^{k+1} + c_j u_{j+1}^{k+1} = d_j, \forall \mathrm{j} \in \{1,\dots,\mathrm{N}-1\}$$

Первое и последнее уравнение системы, содержащие u_0^{k+1} и u_N^{k+1}, определяются граничными условиями

$$u_0^{k+1} = -\frac{\alpha/h}{\beta - \frac{\alpha}{h}} u_1^{k+1} + \frac{\varphi_0(t^{k+1})}{\beta - \frac{\alpha}{h}}, \qquad u_N^{k+1} = \frac{\gamma/h}{\delta + \frac{\gamma}{h}} u_{N-1}^{k+1} + \frac{\varphi_l(t^{k+1})}{\delta + \frac{\gamma}{h}}$$

где, в данном варианте, $\alpha = 0, \beta = 1, \gamma = 0,\ \delta = 1$.

Неявная схема является абсолютно устойчивой.

Схема Кранка-Николсона

Поскольку, как правило, решение в зависимости от времени лежит между значениями явной и неявной схемы, имеет смысл получить смешанную аппроксимацию пространственных производных.

Явно-неявная схема для $\forall j \in \{1, .., N-1\}, \forall k \in \{0, .., K-1\}$ будет выглядеть следующим образом:

$$\frac{u_j^{k+1} - u_j^k}{\tau} = \theta\left(a\frac{u_{j-1}^k - 2u_j^k + u_{j+1}^k}{h^2} + b\frac{u_{i+1}^k + u_{i-1}^k}{2h} + c\,u^k + f_j^k\right) + (1-\theta)\left(a\,\frac{u_{j-1}^{k+1} - 2u_j^{k+1} + u_{j+1}^{k+1}}{h^2} + b\frac{u_{j+1}^{k+1} + u_{j-1}^{k+1}}{2h} + c\,u_j^k + f_j^{k+1}\right)$$

При значении параметра $\theta = \frac{1}{2}$ схема являет собой схему Кранка-Николсона.

Первое и последнее уравнение системы, содержащие u_0^{k+1} и u_N^{k+1}, определяются граничными условиями

$$u_0^{k+1} = -\frac{\alpha/h}{\beta - \frac{\alpha}{h}}u_1^{k+1} + \frac{\varphi_0(t^{k+1})}{\beta - \frac{\alpha}{h}}, \qquad u_N^{k+1} = \frac{\gamma/h}{\delta + \frac{\gamma}{h}}u_{N-1}^{k+1} + \frac{\varphi_l(t^{k+1})}{\delta + \frac{\gamma}{h}}$$

где $\alpha = 0, \beta = 1, \gamma = 0,\ \delta = 1.$

Схема Кранка-Николсона является абсолютно устойчивой.

Программа выполнена на языке python, в среде разработки Jupyter Notebook. Код программы будет в виде незапущенного кода в Jupyter Notebook, вывод программы, сокращён до графиков, построенных в среде Jupyter Notebook.

Код программы

Лабораторная работа №5 (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 U(x, t). Исследовать зависимость погрешности от сеточных параметров τ, h.

вариант 5:

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \sin\pi x$$

$ u(0, t)=0 $ $ u(1, t)=0 $ $ u(x, 0)=0 $

Аналитическое решение:

$$ U(x, t)= \frac{1}{\pi ^ 2}(1 - \exp{(-\pi ^ 2 t)}) \sin(\pi x)

Граничные условия первого рода при $x = 0, x = l$, в нашем случае $l = 1$

```
import matplotlib.pyplot as plt
import math
import typing
```

Существую граничные условия вида:

$$ \alpha \frac{\partial u(0, t)}{\partial x} + \beta u(0, t) = \phi_0 (t) \\ \\   \gamma \frac{\partial u(l, t)}{\partial x} + \delta u(l, t) = \phi_l (t) $$

Входные условия

```
left, right = 0, 1
alpha = 0
betta = 1
gamma = 0
delta = 1
a = 1 # u_xx
b = 0 # u_x
c = 0 #u

def f(x, t):
    return math.sin(math.pi * x)

def phi_0(t):
    return 0
def phi_l(t):
    return 0

def psi(x):
    return 0
```

Аналитическое решение

```
def U(x, t):
    return 1/(math.pi ** 2) * (1 - math.exp((-math.pi ** 2) * t)) * math.sin(math.p
i * x)

def real_U(X:list, T:list) -> list:
    n = len(X)
    m = len(T)
    U_true = [[0] * n for _ in range(m)]
    for k in range(m):
        for j in range(n):
            U_true[k][j] = U(X[j], T[k])
    return U_true
```

Вспомогательные функции

Три графика

```
def plot_graphs(new_X:list, new_T:list, found_U:list, U_true:list, s:str='') -> Non
e:
    plt.plot(new_X, U_true[len(new_T) // 4 ], color='blue', label='Аналитическое ре
шение')
    plt.plot(new_X, found_U[len(new_T) // 4 ], color='red', label=s, linestyle='das
hdot')
```

```
    plt.legend()
    plt.text(0.05, 0.1, s='t = ' + str(new_T[len(new_T) // 4]))
    plt.show()

    plt.plot(new_X, U_true[len(new_T) // 2 ], color='blue', label='Аналитическое решение')
    plt.plot(new_X, found_U[len(new_T) // 2 ], color='red', label=s, linestyle='dashdot')
    plt.legend()
    plt.text(0.05, 0.1, s='t = ' + str(new_T[len(new_T) // 2]))
    plt.show()

    plt.plot(new_X, U_true[len(new_T) - 1], color='blue', label='Аналитическое решение')
    plt.plot(new_X, found_U[len(new_T) - 1], color='red', label=s, linestyle='dashdot')
    plt.legend()
    plt.text(0.05, 0.1, s='t = ' + str(new_T[len(new_T) - 1]))
    plt.show()
```

рендж

```
def frange(start:float, stop:float, step:float) -> float:
    while start < stop:
        yield start
        start += step
```

Ошибка в зависимотсти от h

```
def error (U_my:list, U_true:list) -> float:
    return sum([abs(a - b) for a, b in zip(U_my, U_true)])
def get_error_array_with_h(N:list, left:float, right:float, a:float, b:float, c:float, alpha:float, betta:float, gamma:float, delta:float, find_u:typing.Callable, t_end:float=1, appoximation:int=1, tetta:float=-10) -> (list, list): # H, error
    H = [right/n for n in N]
    ERROR = []
    for n in N:
        if tetta == -10:
            XX, TT, UU = find_u(left, right, a, b, c, alpha, betta, gamma, delta, n=n, t_end=t_end, appoximation=appoximation)
        else:
            XX, TT, UU = find_u(left, right, a, b, c, alpha, betta, gamma, delta, n=n, t_end=t_end, appoximation=appoximation, tetta=tetta)
        U_true = real_U(XX, TT)
        t = len(TT) // 2
        ERROR.append(error(UU[t], U_true[t]))

    return H, ERROR
```

График ошибки в зависимости о h

```
def h_error_plot(H:list, ERROR:list) -> None:
    plt.plot(H, ERROR)
    plt.xlabel("h")
    plt.ylabel("error")
    plt.show()
```

получение T и X

```
def get_t(t0:float, t_end:float, tau:float) -> list:
    return [i for i in frange(t0, t_end + tau, tau)]
```

```python
def get_x(x_0:float, x_l:float, h:float) -> list:
    return [i for i in frange(x_0, x_l + h, h)]
```

Явная конечно разностная схема

Имеется уравнение вида

$$\frac{\partial u}{\partial t} = \alpha\frac{\partial^2 u}{\partial x^2} + b\frac{\partial u}{\partial x} + cu + f(x,t)$$

решения для уравнения имеет вид

$$\frac{u_j^{k+1} - u_j^k}{\tau} = \frac{u_{j+1}^k - 2u_j^k + u_{j-1}^k}{h^2} + f_j^k$$

приведём его к будее удобному виду для итерации

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

Для начальных условий

$$u_j^1 = \frac{\tau}{h^2}\left(u_{j+1}^0 - 2u_j^0 + u_{j-1}^0\right) + \tau f_j^0 + u_j^0$$

$ u^k_0 = \phi_0(t^k) $, $u^k_n = \phi_l(t^k) $, $ u^0_j = \Psi{(x_j)}$

Условие устойчивости $\sigma = \frac{\alpha^2\tau}{h^2} <= \frac{1}{2}$

так как в моём варианте $\alpha = 1$ можно упростить условие до $\frac{ \tau}{h^2} < \frac{1}{2} \implies \tau <= \frac{h^2}{2} $

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

```python
def find_U_explicit(left:float, right:float, a:float, b:float, c:float, alpha:float
, betta:float, gamma:float, delta:float, n:int=10, t_end:float=1, appoximation:int=
1) -> (list, list, list):  # добавить апроксимацию
    def approx(appoximation:int, k:int):
        if appoximation == 1:
            U_values[k][0] = (h * phi_0(T[k]) - alpha * U_values[k][1])/(h * betta
- alpha)
            U_values[k][-1] = (h * phi_l(T[k]) + gamma * U_values[k][-2] )/(h * del
ta + gamma)
        elif appoximation == 2:
            U_values[k][0] = (h * (2 * a - b * h) * phi_0(T[k]) - 2 * alpha * a * U
_values[k][1] - alpha * (h**2) * U_values[k-1][0] / tau) / (alpha * c * (h**2) + be
tta * h * (2 * a - b * h) - 2 * alpha * a - alpha * (h**2) / tau)
            U_values[k][-1] = (h * (2 * a + b * h) * phi_l(T[k]) + 2 * gamma * a *
U_values[k][-2] + gamma * (h**2) * U_values[k-1][-1] / tau) / (2 * gamma * a + gamm
a * (h**2) / tau - c * gamma * (h**2) + delta * h * (2 * a + b * h))
        elif appoximation == 3:
            U_values[k][0] = (2 * h * phi_0(T[k]) - 4 * alpha * U_values[k][1] + al
pha * U_values[k][2]) / (2 * h * betta - 3*alpha)
            U_values[k][-1] = (2 * h * phi_l(T[k]) + 4 * gamma * U_values[k][-2] -
gamma * U_values[k][-3]) / (2 * h * delta + 3 * gamma)

    n = n + 1 # n + начальные условия
    h = right / (n - 1)
    tau = (h**2) / 2 / a ** 2  # для того, чтобы выполнялось условие устойчивости
    sigma = tau  / h**2
```

```
    X = get_x(left, right, h)
    T =  get_t(0, t_end, tau)


    m = len(T)
    U_values = [[0] * n for _ in range(m)]  # сеточные значения


    for j in range(n):
        U_values[0][j] = psi(X[j])

    for j in range(m):
        U_values[j][0] = phi_0(T[j])
        U_values[j][n-1] = phi_l(T[j])

    for k in range(1, m):
        for j in range(1, n-1):
            U_values[k][j] = sigma * U_values[k-1][j + 1] + (1 - 2 * sigma) * U_val
ues[k-1][j] + sigma * U_values[k-1][j-1] + tau * f(X[j], T[k-1])

    approx(appoximation, len(T) - 1)

    return X, T, U_values
```

демонтрация работы алгоритма

```
XX, TT, UU = find_U_explicit(left, right, a, b, c, alpha, betta, gamma, delta, n=25
, t_end=15, appoximation=1)
U_true = real_U(XX, TT)

plot_graphs(XX, TT, UU, U_true, 'Явная конечно разностная схема')
```

Ошибка и её значение от h

```
N = [5, 10, 15, 20, 25, 30, 35, 40]
H, ERROR = get_error_array_with_h(N, left, right, a, b, c, alpha, betta, gamma, del
ta, find_U_explicit, t_end=1, appoximation=1)
h_error_plot(H, ERROR)
```

Неявная конечно разностная схема

решения для уравнения имеет вид

$$\frac{u_j^{k+1} - u_j^k}{\tau} = \frac{u_{j+1}^{k+1} - 2u_j^{k+1} + u_{j-1}^{k+1}}{h^2} + f_j^{k+1}$$

приведём его к будее удобному виду для итерации

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

Для начальных условий

$$u_j^1 = \frac{\tau}{h^2}\left(u_{j+1}^1 - 2u_j^1 + u_{j-1}^1\right) + \tau f_j^1 + u_j^0$$

\$ u^{k+1}_0 = \phi_0(t^{k+1}) \$, \$u^{k+1}_n = \phi_l(t^{k+1}) \$, \$ u^0_j = \Psi{(x_j)}\$

```
def solve_PQ(A0:list, A1:list, A2:list, B:list) -> list:
    P = [-A2[0] / A1[0]]
    Q = [B[0] / A1[0]]
    for i in range(1, len(B)):
        P.append(-A2[i] / (A1[i] + A0[i] * P[i - 1]))
        Q.append((B[i] - A0[i] * Q[i - 1]) / (A1[i] + A0[i] * P[i - 1]))
```

```python
    res = [Q[-1]]

    for i in range(len(B) - 2, -1, -1):
        res.append(P[i] * res[-1] + Q[i])

    return res[::-1]

def find_U_implicit(left:float, right:float, a:float, b:float, c:float, alpha:float
, betta:float, gamma:float, delta:float, n:int=10, t_end:float=1, appoximation:int=
1) -> (list, list, list):
    def approx(appoximation:int, k:int):
        if appoximation == 1:
            A1[0] = (betta - alpha) / h
            A2[0] = alpha / h
            B[0] = phi_0(T[k+1]) / (betta - alpha / h)

            A0[-1] = -gamma / h
            A1[-1] = delta + gamma / h
            B[-1] = phi_l(T[k+1]) / (delta + gamma / h)
        if appoximation == 2:

            A1[0] = 2 * a / h + h / tau - c * h - betta * (2 * a - b * h) / alpha
            A2[0] = - 2 * a / h
            B[0] = h * U[k-1][0] / tau -  phi_0(T[k]) * (2 * a - b * h) / alpha

            A0[-1] = - 2 * a / h
            A1[-1] = 2 * a / h + h / tau - c * h + delta * (2 * a + b * h) / gamma
            B[-1] = h * U[k-1][-1] / tau + phi_l(T[k]) * (2 * a + b * h) / gamma
        if appoximation == 3:
            A1[0] = 2 * betta * h - 3 * alpha
            A2[0] = 4 * alpha
            B[0] = 2 * h * phi_0(T[k+1])

            A0[-1] = -4 * gamma
            A1[-1] = 2 * h * delta + 3 * gamma
            B[-1] = 2 * h * phi_l(T[k+1])

    if alpha == 0 and appoximation == 2:
        appoximation = 3
    n = n + 1 # n + начальные условия
    h = right / (n - 1)
    tau = (h**2) / 2 / a**2  # для того, чтобы выполнялось условие устойчивости
    sigma = tau  / h**2
    X = get_x(left, right, h)
    T =  get_t(0, t_end, tau)


    m = len(T)
    U_values = [[0] * n for _ in range(m)]  # сеточные значения

    for k in range(m-1):
        A0 = [0 for _ in range(n)]
        A1 = [0 for _ in range(n)]
        A2 = [0 for _ in range(n)]
        B = [0 for _ in range(n)]

        for j in range(1, n-1):
            A0[j] = sigma
            A1[j] = -(1 + 2*sigma)
            A2[j] = sigma
            B[j] = -U_values[k][j] - f(X[j], T[k]) * tau
        approx(appoximation, k)
```

```
        U_values[k+1] = solve_PQ(A0, A1, A2, B)
    return X, T, U_values
```

Демонстрация работы

```
XX2, TT2, UU2 = find_U_implicit(left, right, a, b, c, alpha, betta, gamma, delta, n
=25, t_end=15, appoximation=1)
U_true2 = real_U(XX2, TT2)

plot_graphs(XX2, TT2, UU2, U_true2, 'Неявная конечно разностная схема')

N = [5, 10, 15, 20, 25, 30, 35, 40]
H, ERROR = get_error_array_with_h(N, left, right, a, b, c, alpha, betta, gamma, del
ta, find_U_implicit, t_end=1, appoximation=1)
h_error_plot(H, ERROR)
```

Схема Кранка - Николсона

```
def Krank_Nikolson(left:float, right:float, a:float, b:float, c:float, alpha:float,
betta:float, gamma:float, delta:float, n:int=10, t_end:float=1, appoximation:int=1,
tetta:float=0.3) -> list:
    if tetta < 0 or tetta > 1:
        print('Тетта лежит на отрезке [0, 1]')
        return
    if alpha == 0 and appoximation == 2:
        appoximation = 3
    X, T, U_explicit = find_U_explicit(left, right, a, b, c, alpha, betta, gamma, d
elta, n, t_end, appoximation=appoximation)
    X2, T2, U_implicit = find_U_implicit(left, right, a, b, c, alpha, betta, gamma,
delta, n, t_end, appoximation=appoximation)
    m = len(U_implicit)
    n = len(U_implicit[0])
    U_values = [[0] * n for _ in range(m)]

    for k in range(m):
        for j in range(n):
            U_values[k][j] = tetta * U_implicit[k][j] + (1 - tetta) * U_explicit[k]
[j]

    return X2, T2, U_values
```

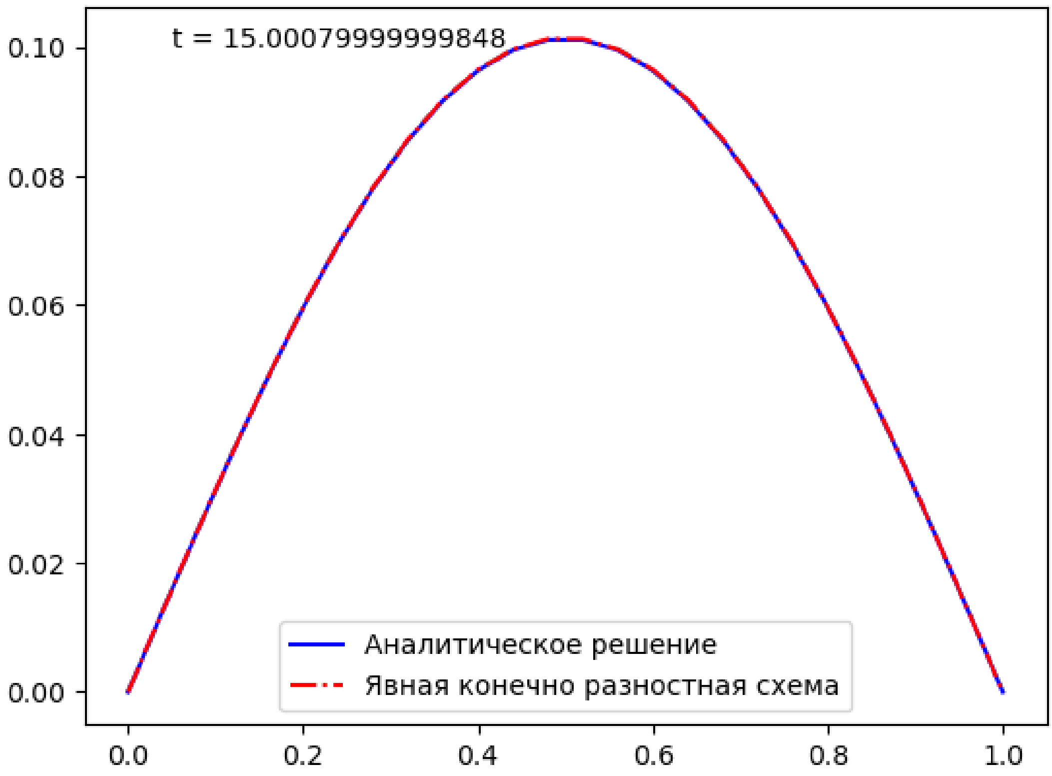
Демонстрация с разными параметрами

```
tetta = 0.5

XX3, TT3, UU3 = Krank_Nikolson(left, right, a, b, c, alpha, betta, gamma, delta, n=
25, t_end=20, appoximation=1, tetta=tetta)
U_true2 = real_U(XX3, TT3)

plot_graphs(XX2, TT2, UU2, U_true2, 'Схема Кранка-Николсона')

N = [5, 10, 15, 20, 25, 30, 35, 40]
H, ERROR = get_error_array_with_h(N, left, right, a, b, c, alpha, betta, gamma, del
ta, Krank_Nikolson, t_end=1, appoximation=1, tetta=tetta)
h_error_plot(H, ERROR)
```

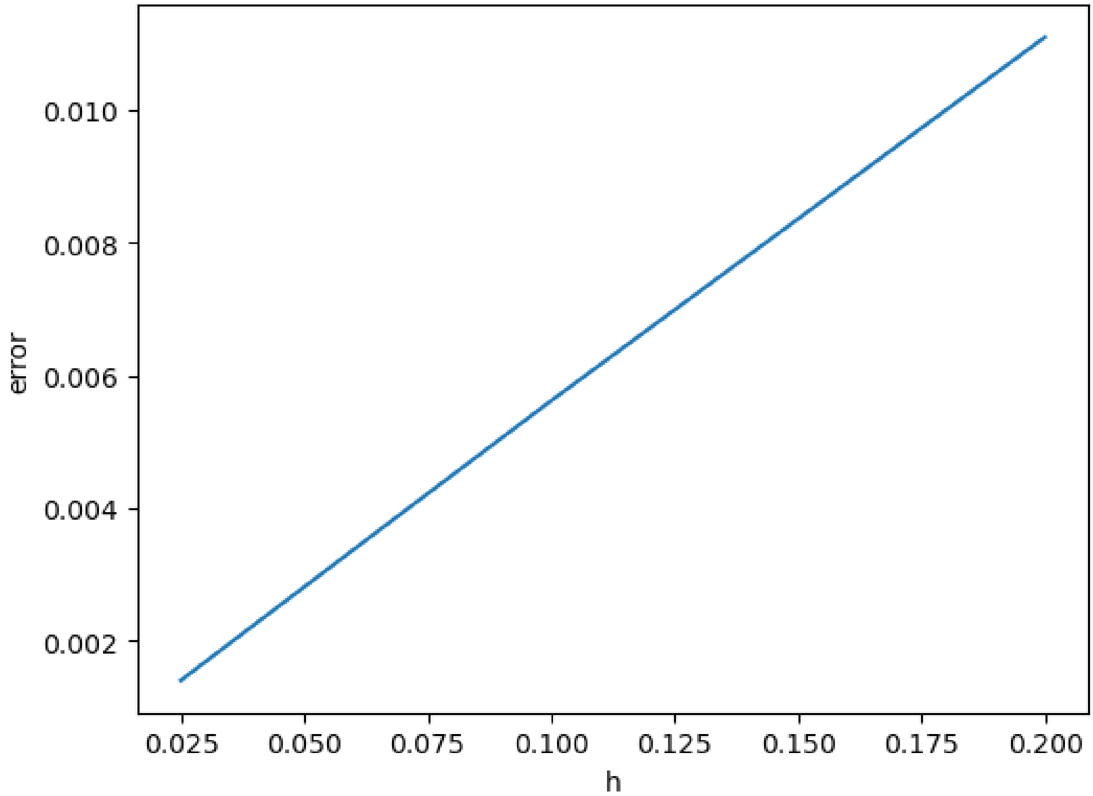
Вывод программы

Для вывода программы был взят вывод графиков, чтобы не дублировать код.

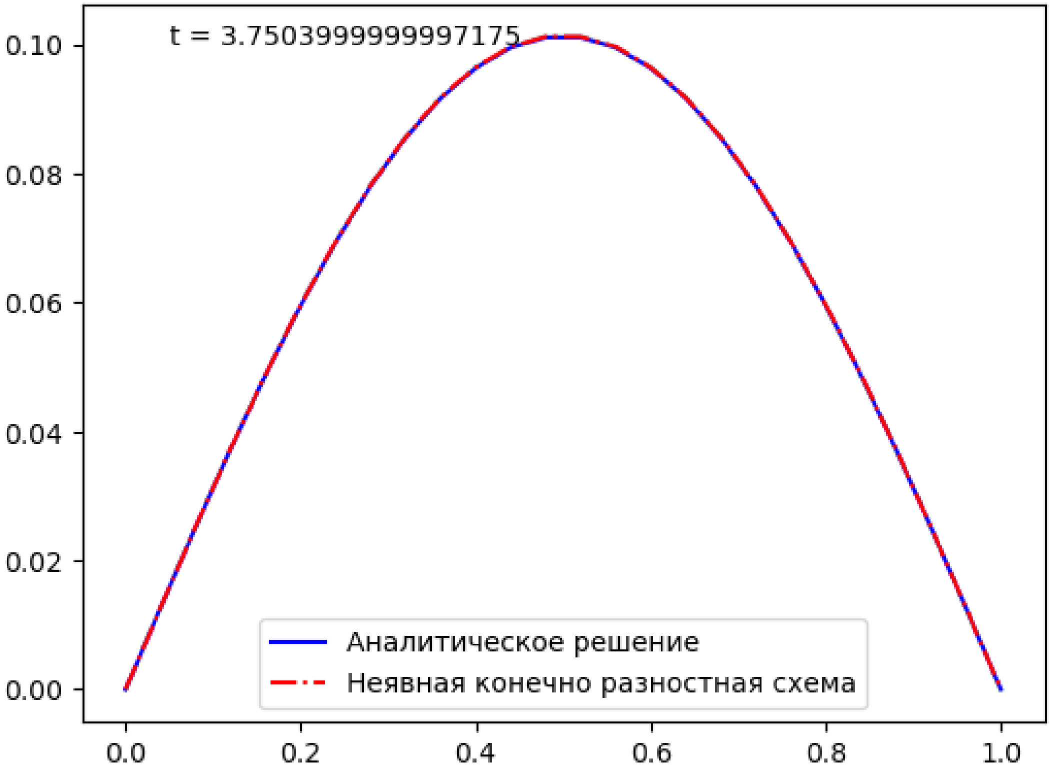
```
'Явная конечно-разностная схема'
```

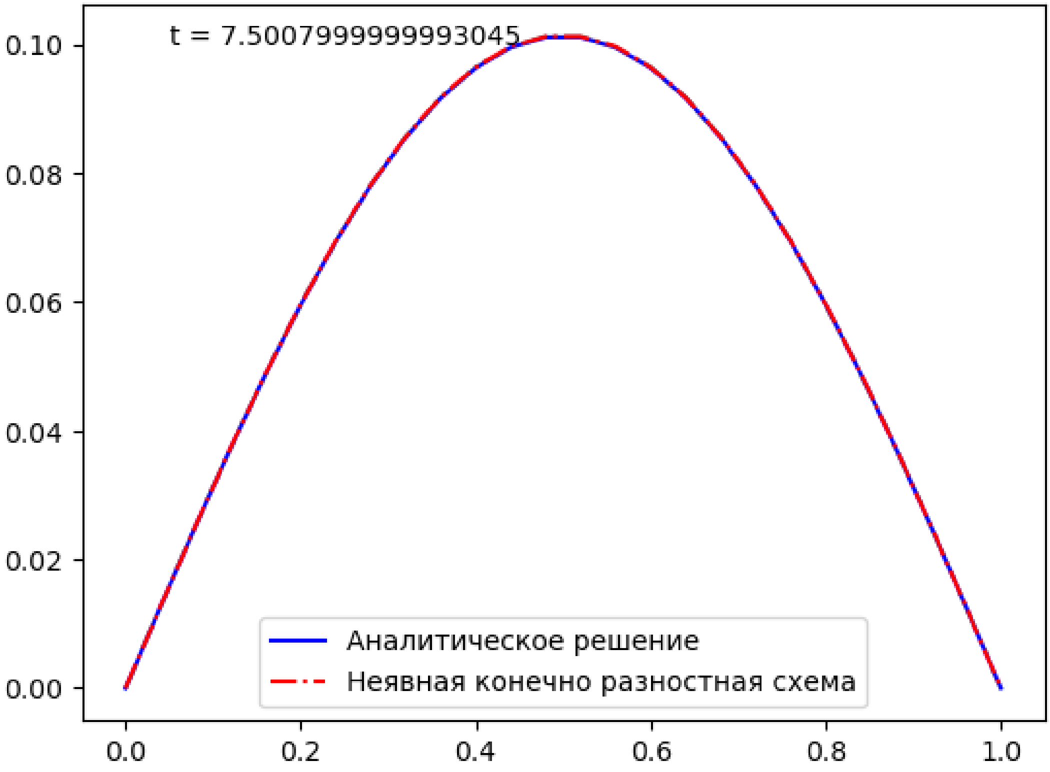


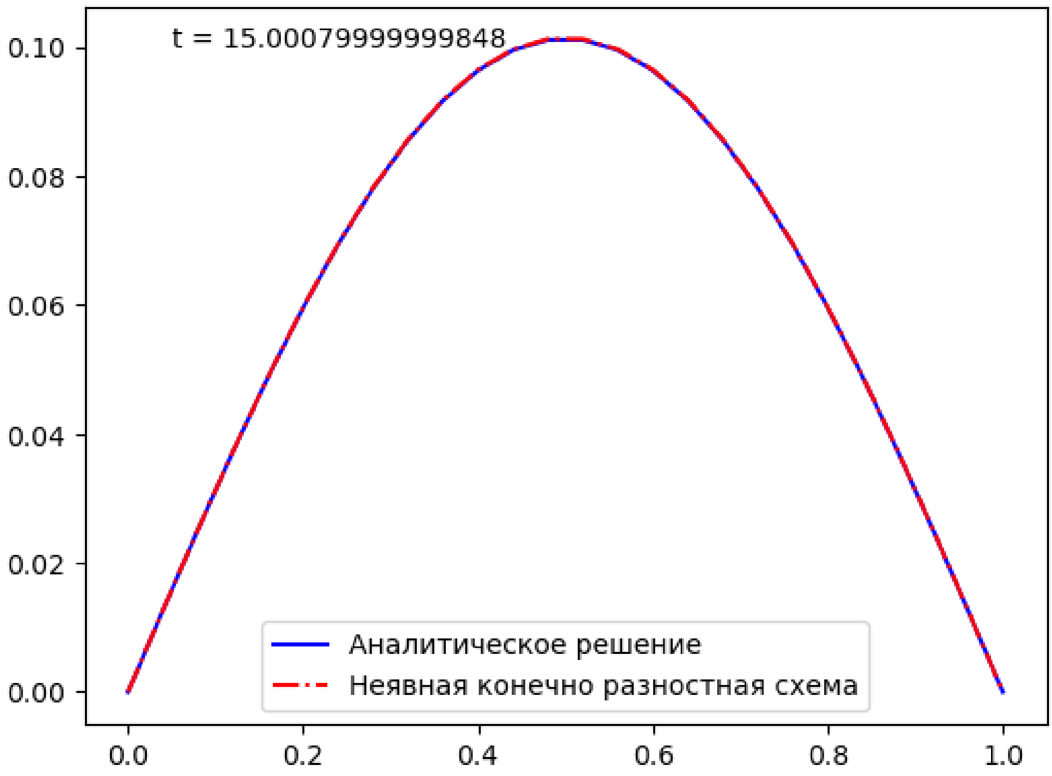
Ошибка и её значение от h



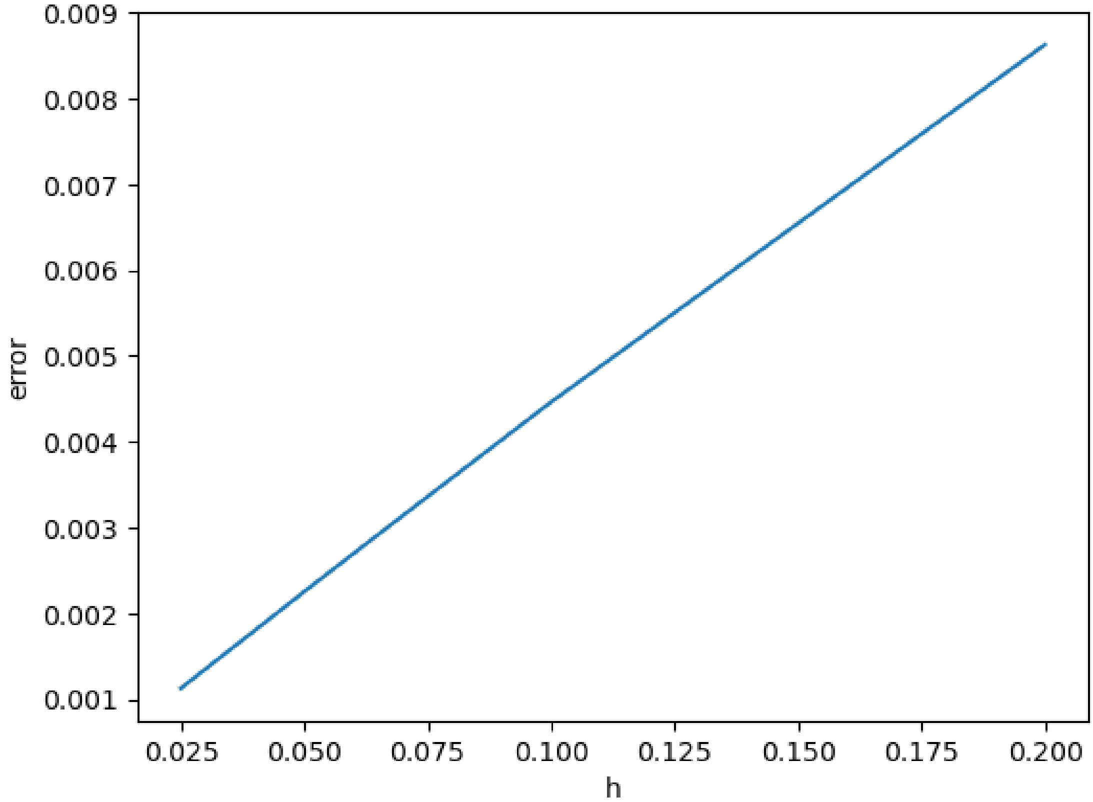
```
'Неявная конечно-разностная схема'
```





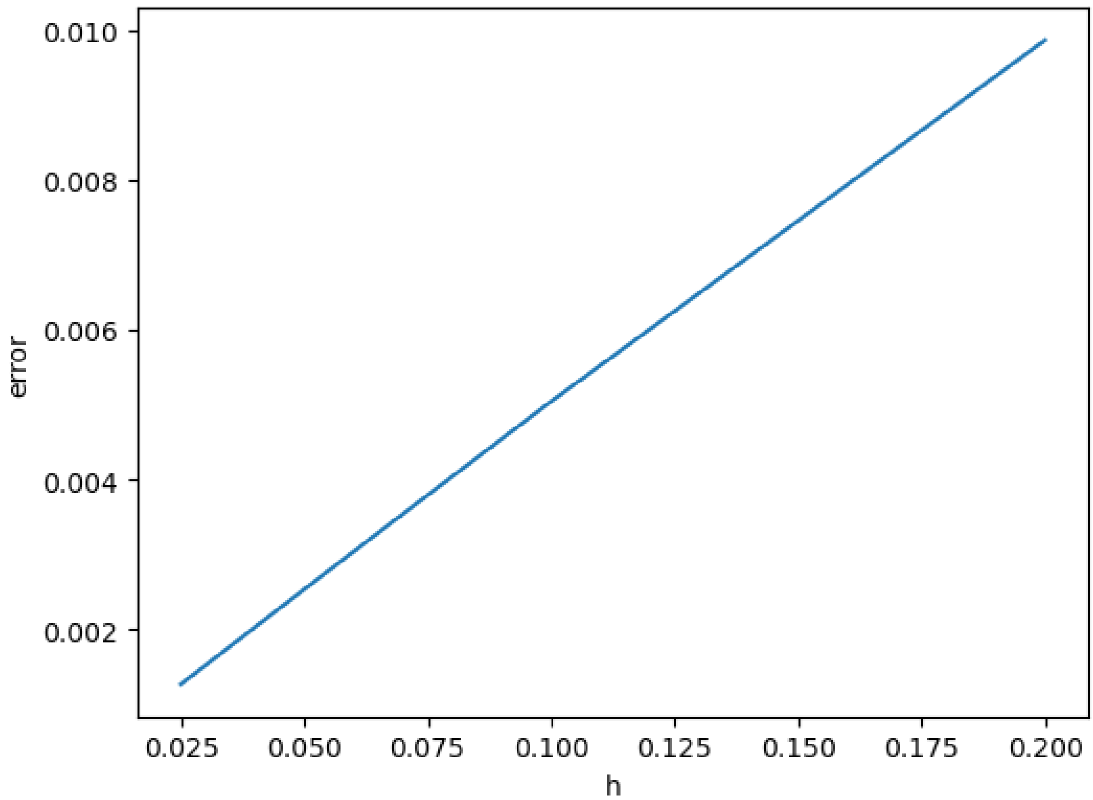


Ошибка и её значение от h



'Схема Кранка-Николсона'

Ошибка и её значение от h



Заключение

В данной лабораторной работе построены конечно-разностные схемы (явная, неявная, Кранка Николсона) для дифференциальных уравнений параболического типа с одной пространственной и одной временной переменной с граничными условиями первого рода.

Схемы были программно реализованы на языке программирования Python в среде разработки Jupyter Notebook.

Во всех трёх методах зависимость ошибки от длины шага линейная, и чем больше шаг, тем больше ошибка. Используя явный метод, стоит учитывать, что он применим только при выполнении критерия устойчивости.

Схема Кранка-Николсона работает точнее, чем неявная схема, а неявная точнее, чем явная. Зависимость погрешности схем от пространственного шага напоминает квадратичную, что подтверждает аппроксимационные свойства этих схем .