Московский авиационный институт

(Национальный исследовательский университет)

Факультет прикладной математики и физики

Кафедра вычислительной математики и программирования

**Лабораторная работа №5**

«ЧИСЛЕННЫЕ МЕТОДЫ РЕШЕНИЯ ОБЫКНОВЕННЫХ ДИФФЕРЕНЦИАЛЬНЫХ УРАВНЕНИЙ»

Вариант 4

Выполнил: Железнов Д.Е.

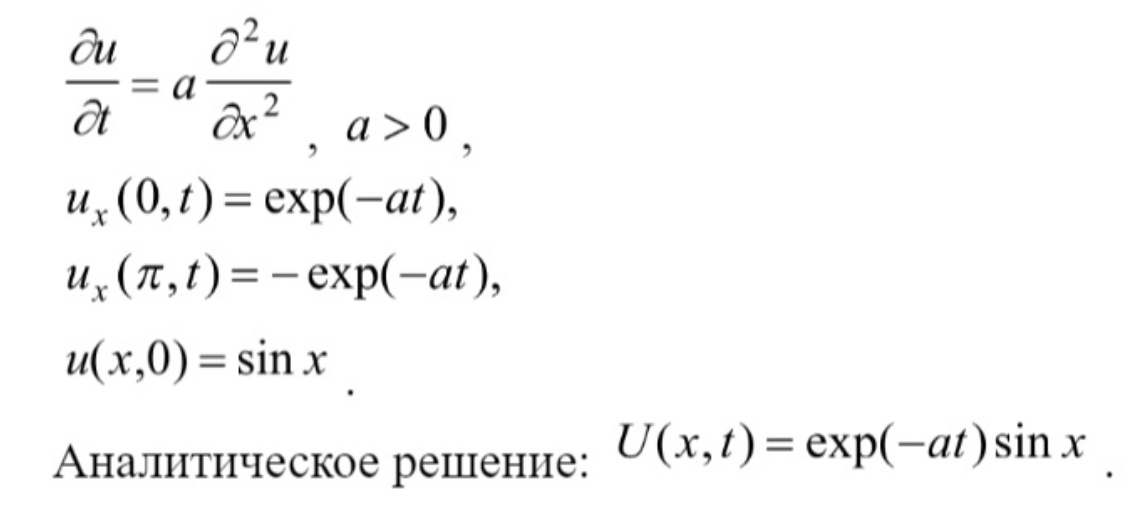
Группа: М8О-409Б-20

Проверил: Пивоваров Д.Е.

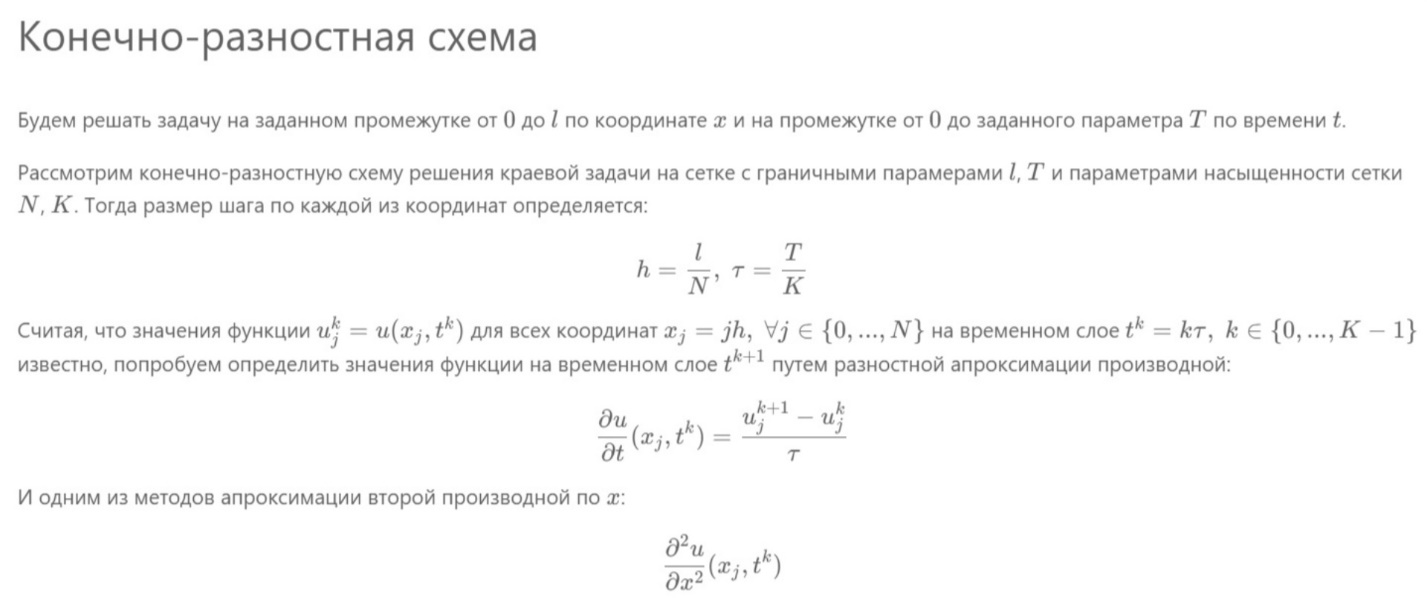
Дата:

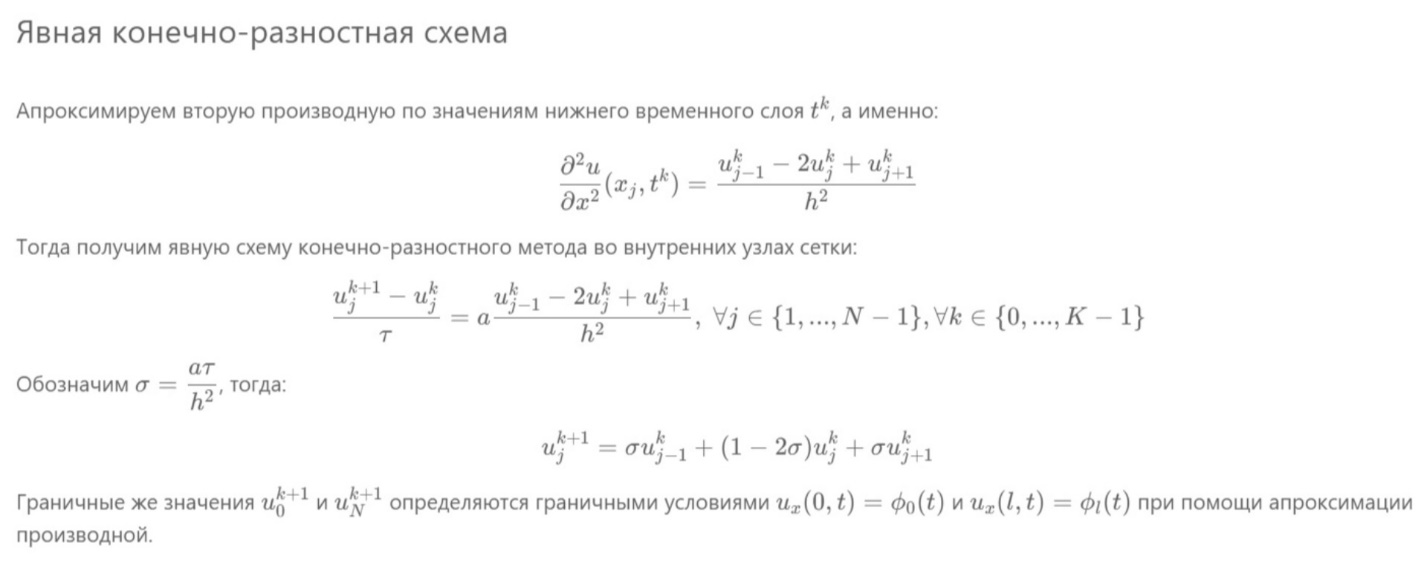
Оценка:

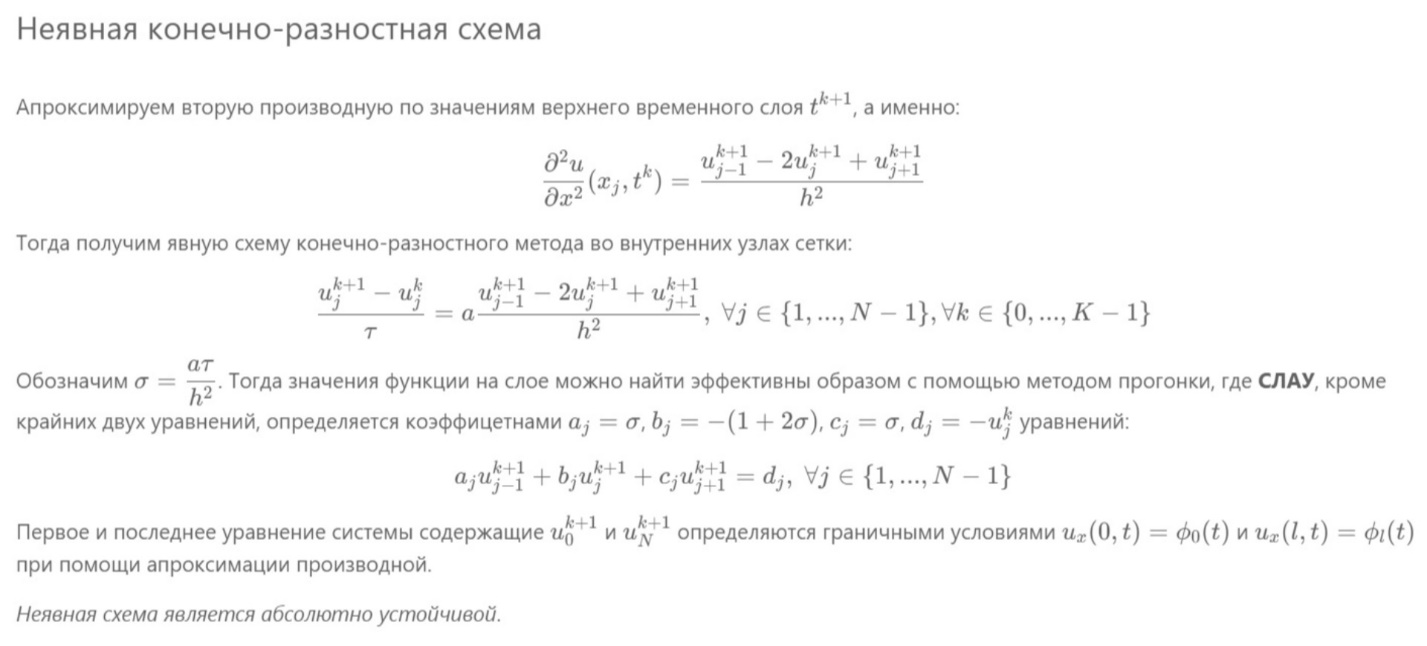
Задание: Используя *явную и неявную конечно-разностные схемы*, а также *схему Кранка - Николсона*, решить начально-краевую задачу для дифференциального уравнения параболического типа. Осуществить реализацию трех вариантов аппроксимации граничных условий, содержащих производные: *двухточечная аппроксимация 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* и *h*.

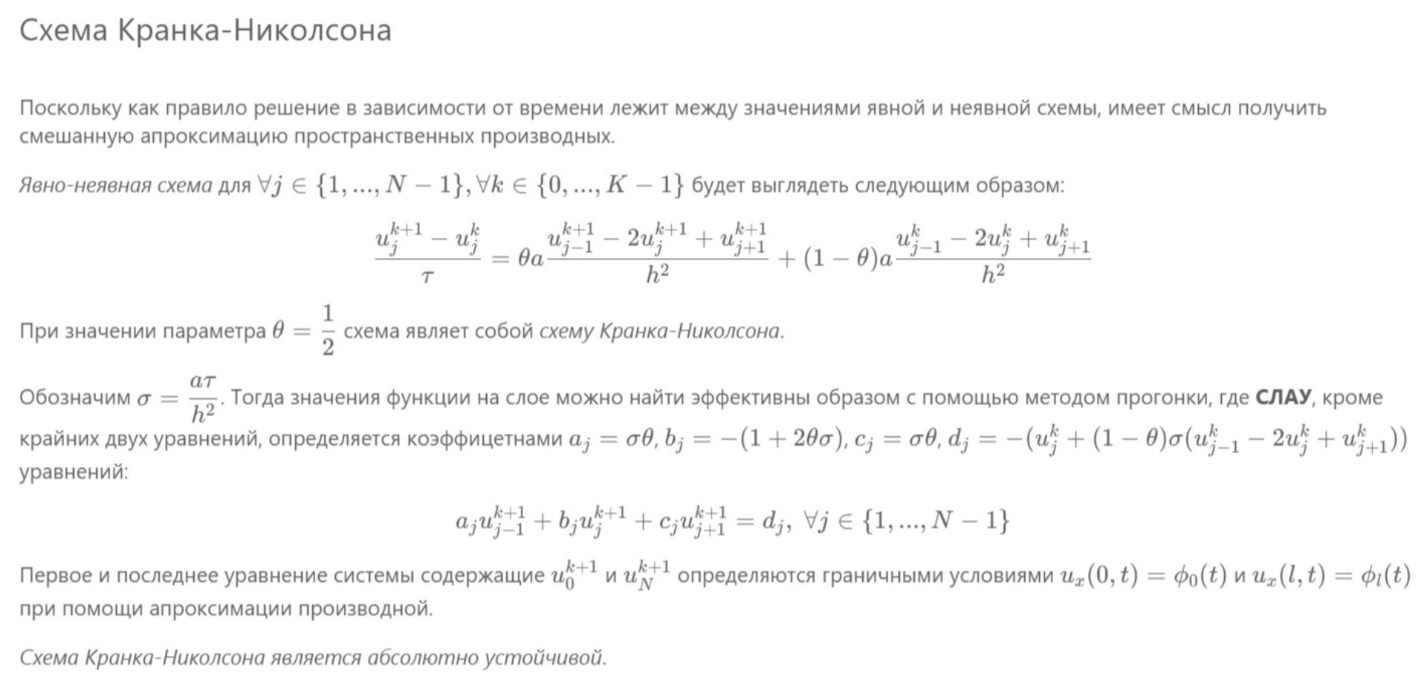


Теоретическая часть:





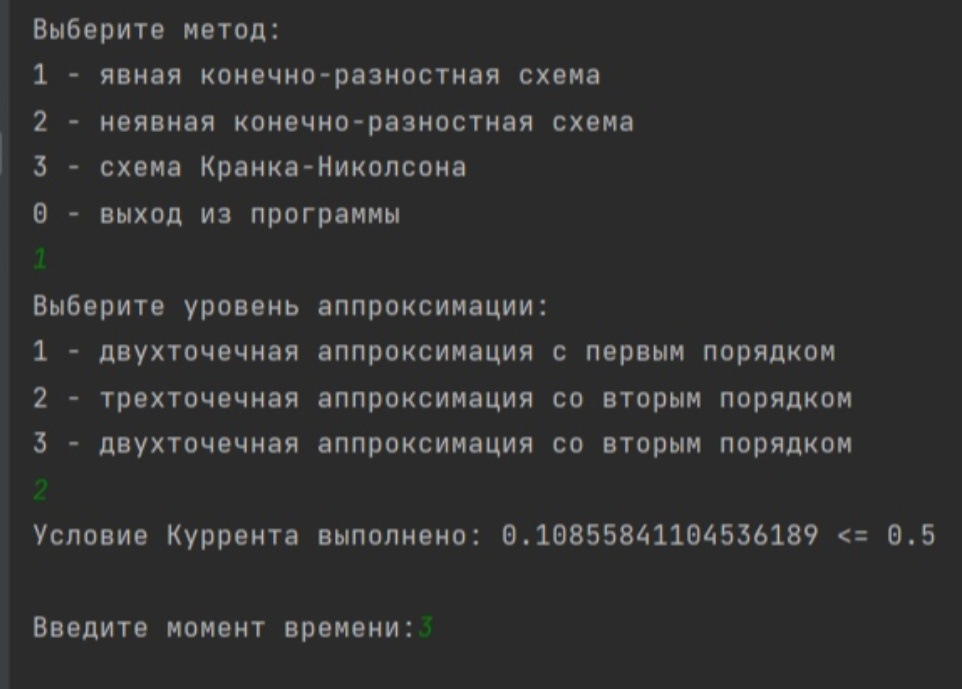


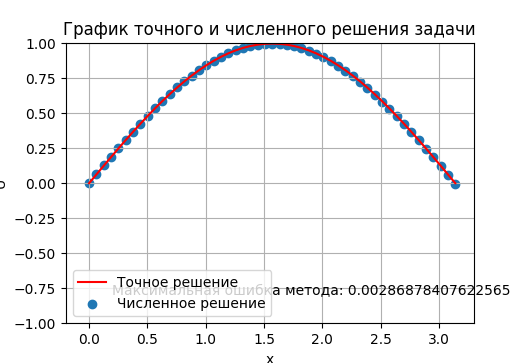


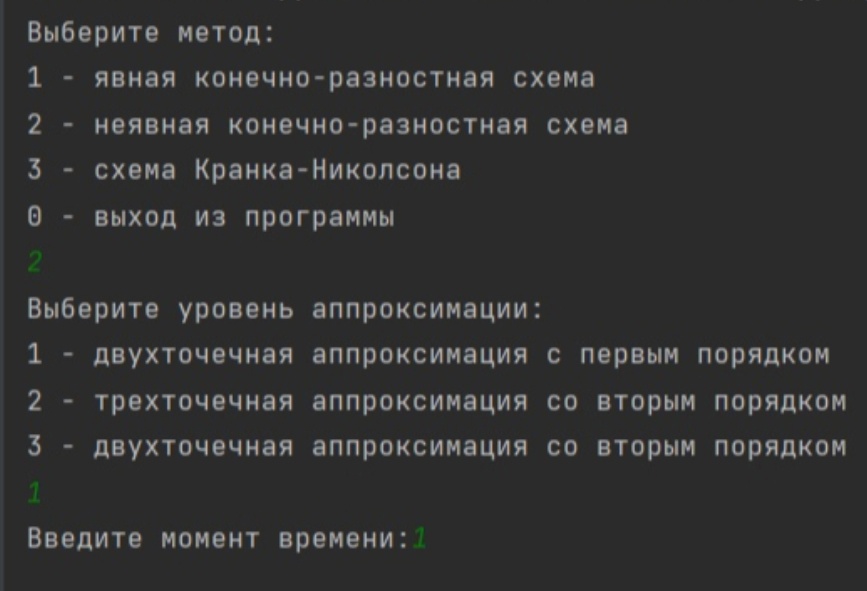
**Код программы:**

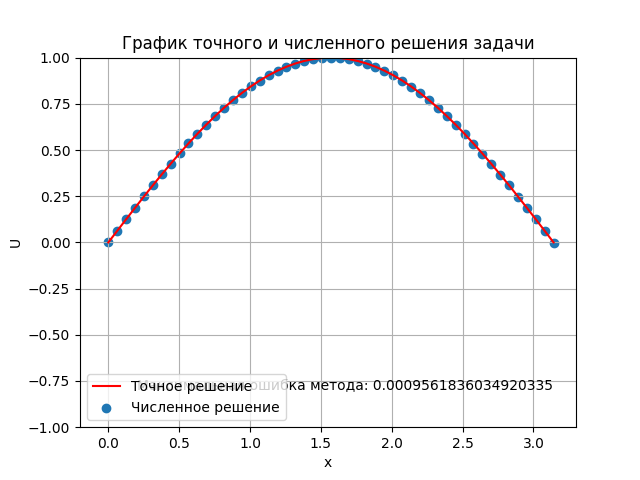
import numpy as np  
import matplotlib.pyplot as plt  
  
def analyt\_func(x, a, t):  
 return np.exp(-a \* t) \* np.sin(x)  
  
def func\_border1(a, t):  
 return np.exp(-a \* t)  
  
def func\_border2(a, t):  
 return -np.exp(-a \* t)  
  
def run\_through(a, b, c, d, s):  
 P = np.zeros(s + 1)  
 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, t) - U[k + 1, 1]) / (h - 1)  
 U[k + 1, N - 1] = (h \* func\_border2(a, t) + U[k + 1, N - 2]) / (h + 1)  
 elif approx == 2:  
 U[k + 1, 0] = (2 \* h \* func\_border1(a, t) - 4 \* U[k + 1, 1] + U[k + 1, 2]) / (2 \* h - 3)  
 U[k + 1, N - 1] = (2 \* h \* func\_border2(a, 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, 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, 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, t)  
  
 a[N - 1] = -1 / h  
 b[N - 1] = 1 + 1 / h  
 d[N - 1] = func\_border2(a1, 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, 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, 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, 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, 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 / N  
 tau = time / K  
 x = np.arange(0, np.pi + h / 2, h)  
 T = np.arange(0, time + tau / 2, tau)  
 a = 1  
 b = 2  
 c = -1  
 t = 0  
  
 while True:  
 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, 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, T[i]) - U[i, :]))  
  
 *# Построение графика ошибок* plt.figure()  
 plt.title("График ошибки по времени и пространству")  
 plt.plot(T, error\_time, label="По времени")  
 plt.plot(x, error, label="По пространству в выбранный момент времени")  
 plt.xlabel("Время и пространство")  
 plt.ylabel("Ошибка")  
 plt.legend()  
 plt.grid()  
 plt.show()  
  
 return 0  
  
N = 50  
K = 7000  
time = 3  
main(N, K, time)

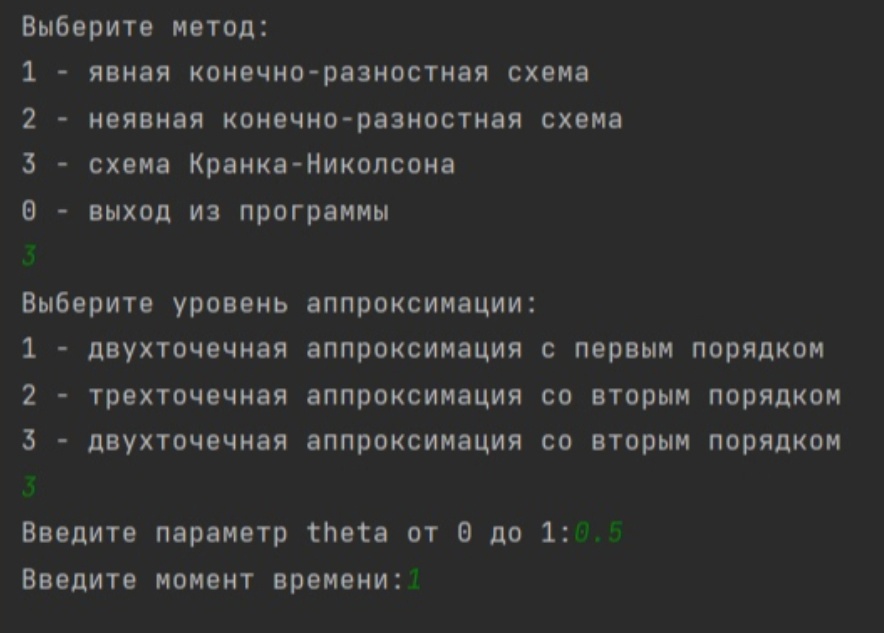
**Результат:**

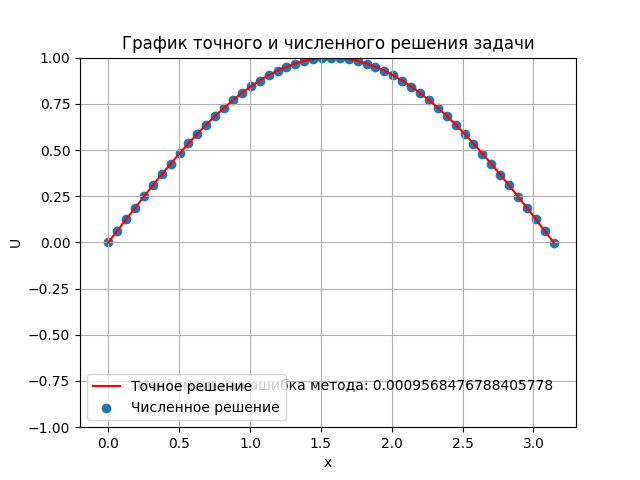
1) 



2) 



3) 



**Вывод:**

В ходе лабораторной работы были изучены *явная и неявная конечно-разностные схемы*, а также *схема Кранка - Николсона*, с их помощью решены начально-краевые задачи для дифференциального уравнения параболического типа. Осуществлена реализация трех вариантов аппроксимации граничных условий, содержащих производные: *двухточечная аппроксимация с первым порядком, трехточечная аппроксимация со вторым порядком, двухточечная аппроксимация со вторым порядком.* В различных моментах времени вычислена погрешность численного решения путем сравнения результатов с приведенным в задании аналитическим решением *u (x, t)*. Исследована зависимость погрешности от сеточных параметров *τ* и *h*.