Московский авиационный институт

(Национальный исследовательский университет)

Факультет прикладной математики и физики

Кафедра вычислительной математики и программирования

**Лабораторная работа №6**

«ЧИСЛЕННЫЕ МЕТОДЫ РЕШЕНИЯ ОБЫКНОВЕННЫХ ДИФФЕРЕНЦИАЛЬНЫХ УРАВНЕНИЙ»

Вариант 4

Выполнил: Эсмедляев Е.Р.

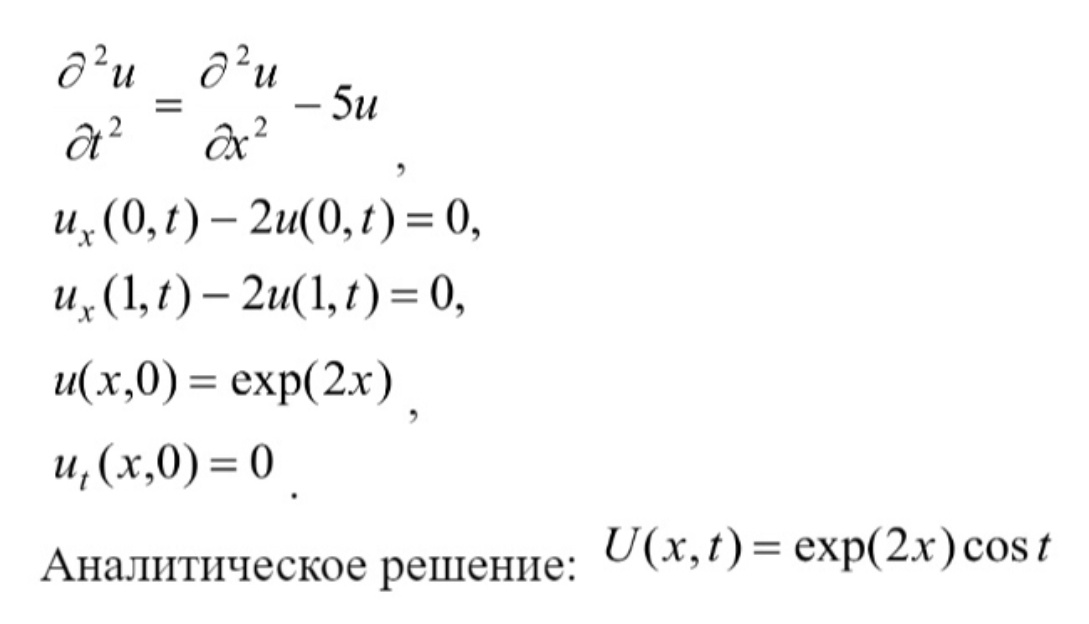
Группа: М8О-409Б-20

Проверил: Пивоваров Д.Е.

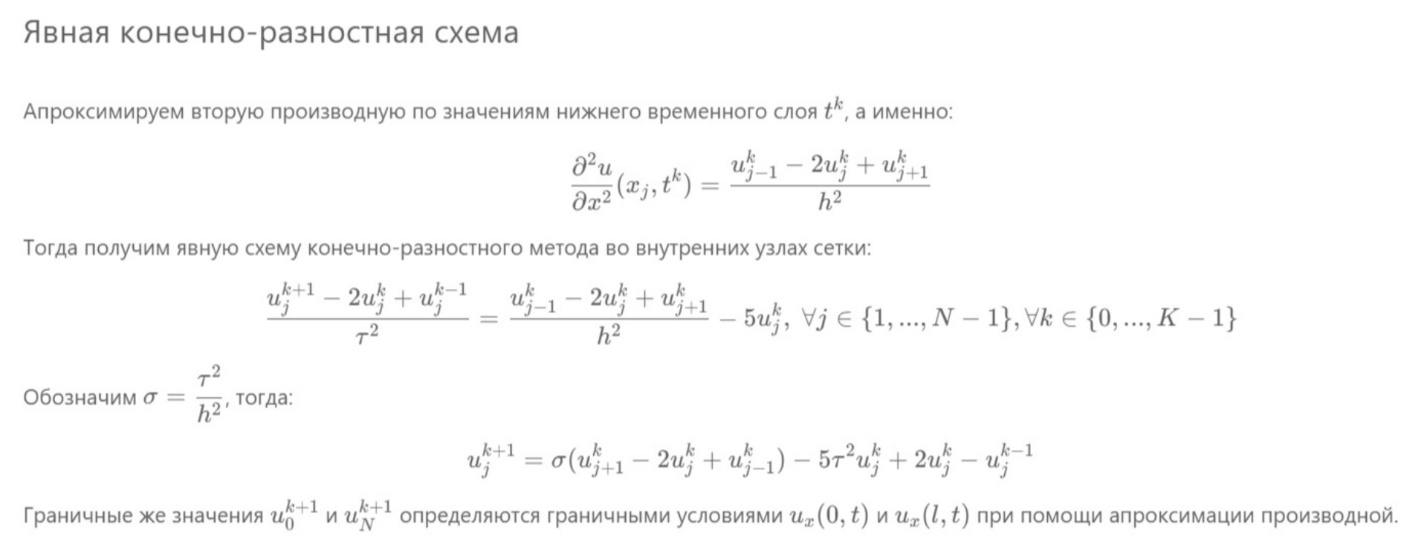
Дата:

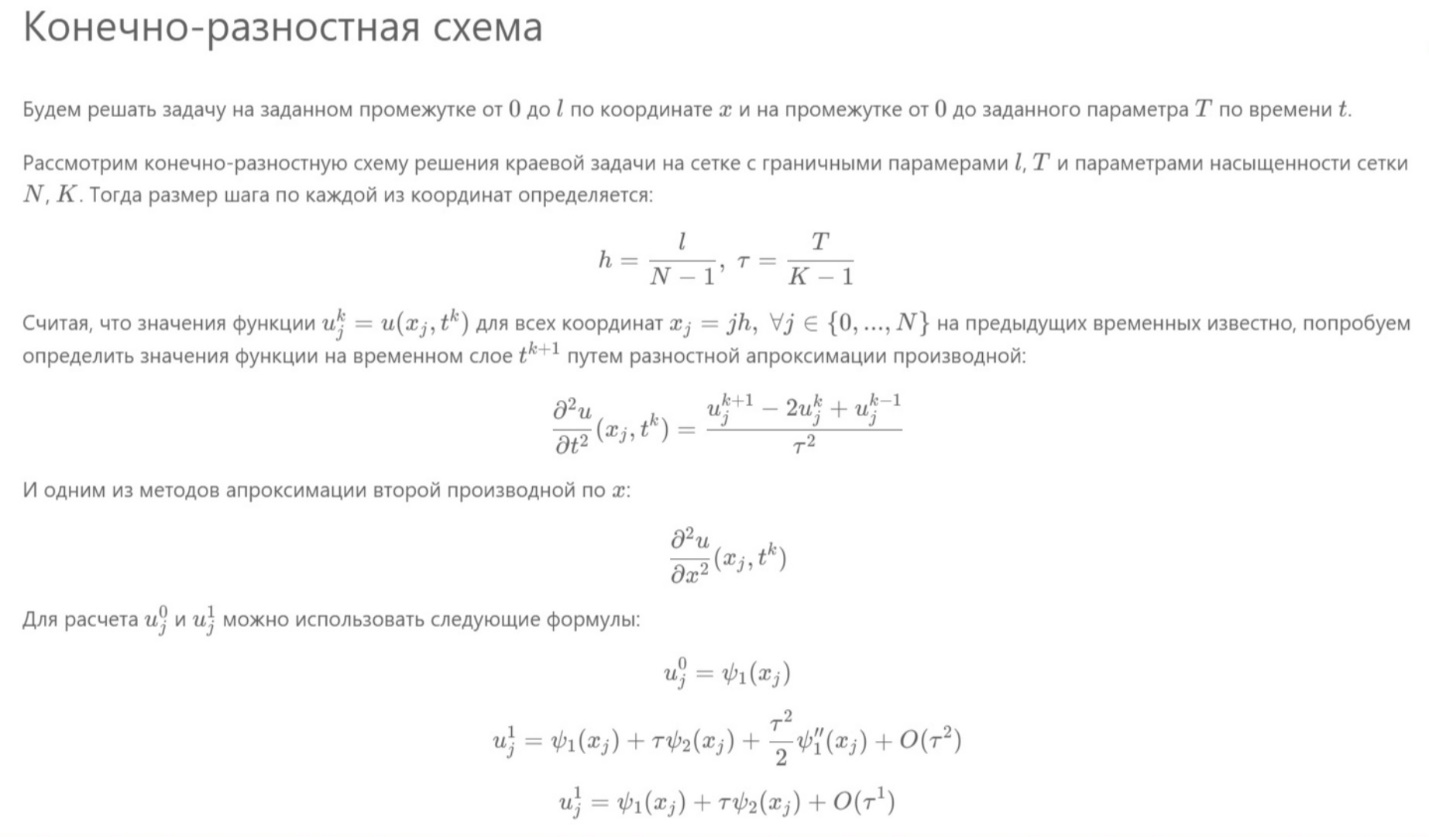
Оценка:

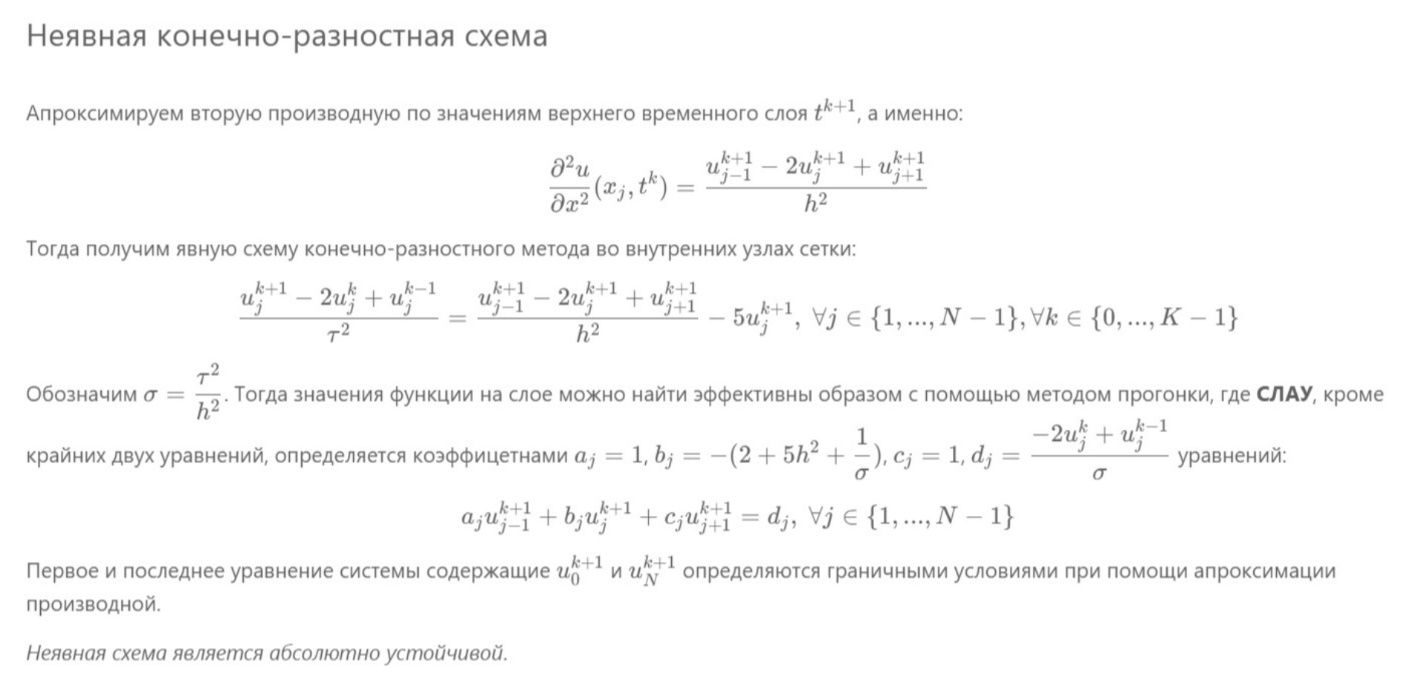
Задание: Используя *явную схему крест и неявную схему*, решить начально-краевую задачу для дифференциального уравнения гиперболического типа. Аппроксимацию второго начального условия произвести с первым и со вторым порядком. Осуществить реализацию трех вариантов аппроксимации граничных условий, содержащих 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 *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 Ux0(t):  
 return 0  
  
def Uxl(t):  
 return 0  
  
def U(x):  
 return np.exp(2 \* x)  
  
def Analitic(x, t):  
 return np.exp(2 \* x) \* np.cos(t)  
  
def progonka(a, b, c, d, s):  
 P = np.zeros(s)  
 Q = np.zeros(s)  
  
 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  
  
x0 = 0  
xl = 1  
*# t = 2*param\_a = 1  
param\_c = -5  
  
  
def autofill(x0, space\_step, m, n, param\_a, time\_step, aprox\_f=1):  
 Uarray = np.zeros([n, m])  
  
 tmp\_x = x0  
 for j in range(m):  
 Uarray[0][j] = U(tmp\_x)  
 if aprox\_f == 1:  
 Uarray[1][j] = U(tmp\_x)  
 if aprox\_f == 2:  
 Uarray[1][j] = U(tmp\_x) + \  
 (param\_a\*\*2 \* 4 \* U(tmp\_x) + param\_c \* U(tmp\_x))\  
 \* time\_step \*\* 2 / 2  
 tmp\_x += space\_step  
 return Uarray  
  
  
def explicit(t, m, n, aprox, aprox\_f, ans\_time, method\_name, aprox\_name, aprox\_f\_name):  
 x0 = 0  
 xl = 1  
  
 space\_step = (xl - x0) / (m - 1)  
 time\_step = t / (n - 1)  
  
 X = np.arange(x0, xl + space\_step, space\_step)  
  
 Uarray = autofill(x0, space\_step, m, n, param\_a, time\_step, aprox\_f)  
  
 sigma = param\_a\*\*2 \* time\_step\*\*2 / space\_step\*\*2  
  
 alpha = 1.  
 betta = -2.  
 gamma = 1.  
 delta = -2.  
  
 for k in range(1, n - 1):  
 for j in range(1, m - 1):  
 Uarray[k + 1][j] = \  
 Uarray[k][j + 1] \* sigma +\  
 Uarray[k][j] \* (-2 \* sigma + 2 + param\_c \* (time\_step\*\*2)) + \  
 Uarray[k][j - 1] \* sigma - \  
 Uarray[k - 1][j]  
 if aprox == 1:  
 Uarray[k + 1][0] = alpha \* Uarray[k][1] / \  
 (alpha - space\_step \* betta)  
 Uarray[k + 1][m - 1] = gamma \* Uarray[k][m - 2] / \  
 (gamma + space\_step \* delta)  
 *# Uarray[k + 1][0] = ((-alpha / space\_step) /  
 # (betta - alpha / space\_step))\  
 # \* Uarray[k + 1][1]\  
 # + Ux0((k + 1) \* time\_step) / (betta - alpha / space\_step)  
 # Uarray[k + 1][m - 1] = ((gamma / space\_step) /  
 # (delta + gamma / space\_step))\  
 # \* Uarray[k + 1][m - 2]\  
 # + Uxl((k + 1) \* time\_step) / (delta + gamma / space\_step)* if aprox == 2:  
 Uarray[k + 1][0] = \  
 (Ux0((k + 1) \* time\_step) +  
 alpha / 2 / space\_step \* Uarray[k + 1][2] -  
 2 \* alpha / space\_step \* Uarray[k + 1][1]) /\  
 (-3 \* alpha / 2 / space\_step + betta)  
 Uarray[k + 1][m - 1] = \  
 (Uxl((k + 1) \* time\_step) -  
 alpha / 2 / space\_step \* Uarray[k + 1][m - 3] +  
 2 \* alpha / space\_step \* Uarray[k + 1][m - 2]) /\  
 (3 \* alpha / 2 / space\_step + betta)  
 if aprox == 3:  
 Uarray[k + 1][0] = \  
 (Ux0((k + 1) \* time\_step) -  
 alpha \* space\_step / time\_step / 2 \* Uarray[k][0] -  
 Uarray[k + 1][1] \* alpha \* 2 \* param\_a / space\_step / 2) /\  
 (alpha \* (-2 \* param\_a / space\_step / 2 -  
 space\_step / time\_step / 2 +  
 param\_c \* space\_step / 2) + betta)  
  
 Uarray[k + 1][m - 1] = \  
 (Uxl((k + 1) \* time\_step) +  
 alpha \* (space\_step \* Uarray[k][m - 1] / 2 / time\_step +  
 2 \* param\_a / space\_step / 2 \*  
 Uarray[k + 1][m - 2])) /\  
 (alpha \* (2 \* param\_a / space\_step / 2 +  
 space\_step / 2 / time\_step -  
 param\_c \* space\_step / 2) + betta)  
 in\_array = int(ans\_time / time\_step)  
 ans\_t = in\_array \* time\_step  
  
 plt.figure(figsize=(12, 4))  
  
 plt.subplot(121)  
 plt.plot(X, Analitic(X, ans\_t), color='red', label='Analytical')  
 plt.plot(X, Uarray[in\_array], label='Explicit')  
 plt.title(f"Solution using {method\_name} Method\nApproximation: {aprox\_name}, Initial Cond.: {aprox\_f\_name}")  
 plt.legend(loc='lower left')  
 plt.grid()  
  
 plt.subplot(122)  
 T = np.arange(0, 1 + time\_step, time\_step) *# Ограничение по времени от 0 до 1* max\_analitic\_in\_it\_time = []  
 for k in T:  
 in\_it\_time = Analitic(X, k)  
 in\_arr = int(k / time\_step)  
 max\_analitic\_in\_it\_time.append(max(abs(in\_it\_time - Uarray[in\_arr])))  
 plt.plot(T, max\_analitic\_in\_it\_time, color='red', label='Erroнr')  
 plt.title("Error over Time") *# Уточнение, что ошибка рассчитана по времени* plt.xlabel('Time') *# Подпись оси X* plt.xlim(0, 1) *# Устанавливаем пределы для оси X от 0 до 1* plt.legend(loc='upper left')  
 plt.grid()  
 plt.show()  
  
  
def implicit(t, m, n, aprox, aprox\_f, ans\_time,method\_name, aprox\_name, aprox\_f\_name ):  
 x0 = 0  
 xl = 1  
  
 space\_step = (xl - x0) / (m - 1)  
 time\_step = t / (n - 1)  
  
 X = np.arange(x0, xl + space\_step, space\_step)  
  
 Uarray = autofill(x0, space\_step, m, n, param\_a, time\_step, aprox\_f)  
  
 sigma = param\_a\*\*2 \* time\_step\*\*2 / space\_step\*\*2  
  
 alpha = 1  
 betta = -2  
 gamma = 1  
 delta = -2  
  
 for k in range(1, n - 1):  
 a = np.zeros(m)  
 b = np.zeros(m)  
 c = np.zeros(m)  
 d = np.zeros(m)  
  
 for j in range(1, m - 1):  
 a[j] = sigma  
 b[j] = -(1 + 2 \* sigma)  
 c[j] = sigma  
 d[j] = Uarray[k - 1][j] - \  
 (param\_c \* time\_step\*\*2 + 2) \* Uarray[k][j]  
 if aprox == 1:  
 b[0] = betta - alpha / space\_step  
 c[0] = alpha / space\_step  
 d[0] = Ux0((k + 1) \* time\_step)  
  
 a[m - 1] = - gamma / space\_step  
 b[m - 1] = delta + gamma / space\_step  
 d[m - 1] = Uxl((k + 1) \* time\_step)  
 elif aprox == 2:  
 k0 = alpha / 2 / space\_step / c[1]  
 c[0] = 2 \* alpha / space\_step + b[1] \* k0  
 b[0] = (-3 \* alpha / 2 / space\_step + betta) + a[1] \* k0  
 d[0] = Ux0((k + 1) \* time\_step) + d[1] \* k0  
  
 k1 = -(alpha / (space\_step \* 2)) / a[m - 2]  
 a[m - 1] = (-2 \* alpha / space\_step) + b[m - 2] \* k1  
 b[m - 1] = (3 \* alpha / 2 / space\_step + betta) + c[m - 2] \* k1  
 d[m - 1] = Uxl((k + 1) \* time\_step) + d[m - 2] \* k1  
 elif aprox == 3:  
 b[0] = (alpha \* (-2 \* param\_a / space\_step / 2 -  
 space\_step / time\_step / 2 +  
 param\_c \* space\_step / 2) + betta)  
 c[0] = alpha \* 2 \* param\_a / space\_step / 2  
 d[0] = \  
 (Ux0((k + 1) \* time\_step) -  
 alpha \* space\_step / time\_step / 2 \* Uarray[k][0])  
  
 a[m - 1] = -alpha \* 2 \* param\_a / space\_step / 2  
 b[m - 1] = alpha \* (2 \* param\_a / space\_step / 2 +  
 space\_step / time\_step / 2 -  
 param\_c \* space\_step / 2) + betta  
 d[m - 1] = \  
 (Uxl((k + 1) \* time\_step) +  
 alpha \* space\_step / time\_step / 2 \* Uarray[k][m - 1])  
  
 Y = progonka(a, b, c, d, m)  
 Uarray[k + 1] = Y  
  
 in\_array = int(ans\_time / time\_step)  
 ans\_t = in\_array \* time\_step  
  
 plt.figure(figsize=(12, 4))  
  
 plt.subplot(121)  
 plt.plot(X, Analitic(X, ans\_t), color='red', label='Analytical')  
 plt.plot(X, Uarray[in\_array], label='Implicit')  
 plt.title(f"Solution using {method\_name} Method\nApproximation: {aprox\_name}, Initial Cond.: {aprox\_f\_name}")  
 plt.legend(loc='lower left')  
 plt.grid()  
  
 plt.subplot(122)  
 T = np.arange(0, 1 + time\_step, time\_step) *# Ограничение по времени от 0 до 1* max\_analitic\_in\_it\_time = []  
 for k in T:  
 in\_it\_time = Analitic(X, k)  
 in\_arr = int(k / time\_step)  
 max\_analitic\_in\_it\_time.append(max(abs(in\_it\_time - Uarray[in\_arr])))  
 plt.plot(T, max\_analitic\_in\_it\_time, color='red', label='Error')  
 plt.title("Error over Time") *# Уточнение, что ошибка рассчитана по времени* plt.xlabel('Time') *# Подпись оси X* plt.xlim(0, 1) *# Устанавливаем пределы для оси X от 0 до 1* plt.legend(loc='upper left')  
 plt.grid()  
 plt.show()  
  
  
def modified\_main(t, m, n, ans\_time):  
 *# Parameters* t = 2  
 m = 100  
 n = 200  
 ans\_time = 1  
  
 space\_step = 1.0 / (m - 1)  
 time\_step = t / (n - 1)  
  
 *# Iterate through the methods and approximations* for method\_choice in [1, 2]: *# 1 - Explicit, 2 - Implicit* for aprox in [1, 2, 3]: *# 1 - Two-point (first order), 2 - Three-point (second order), 3 - Two-point (second order)* for aprox\_f in [1, 2]: *# 1 - First order, 2 - Second order  
 # Check for stability in explicit method* if time\_step\*\*2 / space\_step\*\*2 >= 1 and method\_choice == 1:  
 print('Ошибка!\nПри таких параметрах Явный метод не устойчив!\nПожалуйста, измените параметры сетки.')  
 continue  
  
 method\_name = "Explicit" if method\_choice == 1 else "Implicit"  
 aprox\_name = f"Two-point (first order)" if aprox == 1 else \  
 f"Three-point (second order)" if aprox == 2 else \  
 f"Two-point (second order)"  
 aprox\_f\_name = "First order" if aprox\_f == 1 else "Second order"  
  
 if method\_choice == 1:  
 explicit(t, m, n, aprox, aprox\_f, ans\_time, method\_name, aprox\_name, aprox\_f\_name)  
 elif method\_choice == 2:  
 implicit(t, m, n, aprox, aprox\_f, ans\_time, method\_name, aprox\_name, aprox\_f\_name)  
  
*# Execute the modified main function*modified\_main(t=2, m=100, n=1000, ans\_time=1)

**Результат:**

Изображение выглядит как текст, линия, График, диаграмма

Автоматически созданное описание

Изображение выглядит как текст, линия, График, диаграмма

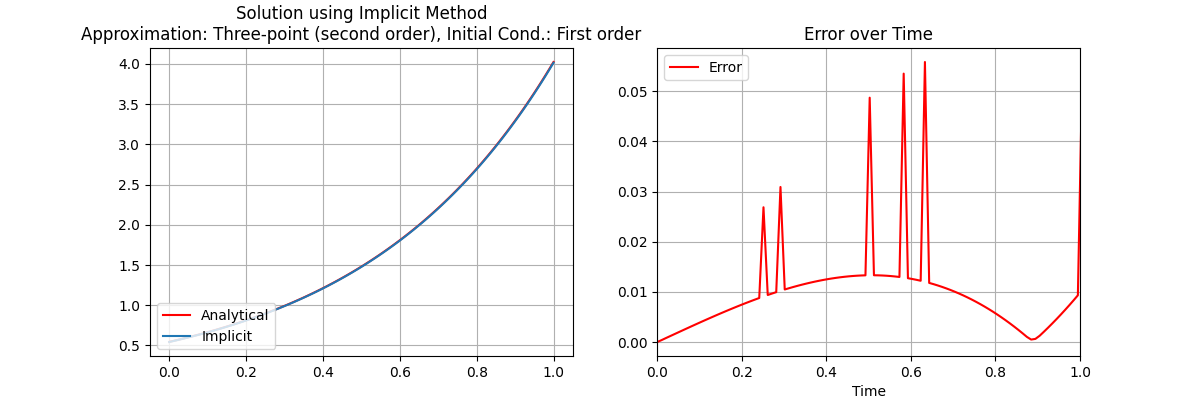
Автоматически созданное описание

Изображение выглядит как текст, линия, диаграмма, График

Автоматически созданное описание

Изображение выглядит как текст, линия, График, диаграмма

Автоматически созданное описание



Изображение выглядит как текст, линия, График, диаграмма

Автоматически созданное описание

Изображение выглядит как текст, линия, График, диаграмма

Автоматически созданное описание

**Вывод:**

В ходе лабораторной работы были изучены *явная схема крест и неявная схема*, решена с их помощью начально-краевую задача для дифференциального уравнения гиперболического типа. Проведена аппроксимация второго начального условия с первым и со вторым порядком. 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 различные моменты 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*.