|  |  |
| --- | --- |
|  | |
| 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 11 | |
|  | |
|  | Выполнила: Попова Н. С. |
|  | 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.

|  |
| --- |
| Аналитическое решение: |

### **Теоретические сведения**

**Конечно-разностная схема**

На границах х = 0 и х = даны значения искомой функции u(x,t) в виде

,

т.е. заданы граничные условия первого рода, и, кроме того, заданы начальные условия u(x,0) = ψ (x), 0 ≤ x ≤ , t= 0.

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:

Считая, что значения функции для всех координат

на временном слое известны,

попробуем определить значения функции на временном слое через граничные условия.

**Явная конечно-разностная схема**

Явная схема конечно-разностного метода во внутренних узлах сетки:

Граничные значения и определяются граничными условиями.

*.*

Явная схема является условно устойчивой, с условием .

**Неявная конечно-разностная схема**

Неявная схема конечно-разностного метода во внутренних узлах сетки:

.

Граничные значения и определяются граничными условиями.

*.*

Тогда значения функции на верхнем временном слое можно найти из решения СЛАУ с трехдиагональной матрицей. Эта СЛАУ в форме, пригодной для использования метода прогонки, имеет вид:

Первое и последнее уравнение системы, содержащие и определяются граничными условиями

Неявная схема является абсолютно устойчивой.

**Схема Кранка-Николсона**

Поскольку, как правило, решение в зависимости от времени лежит между

значениями явной и неявной схемы.

Явно-неявная схема для будет выглядеть

следующим образом:

При значении параметра схема являет собой схему Кранка-Николсона.

Обозначим . Тогда значения функции на слое можно найти эффективным образом с помощью методом прогонки, где СЛАУ, кроме крайних двух уравнений, определяется коэффициентами

уравнений:

Первое и последнее уравнение системы, содержащие и , определяются граничными условиями

*.*

Схема Кранка-Николсона является абсолютно устойчивой.

**Код программы**

В качестве параметров в коде были взяты числа N = 50, K = 10000, момент времени = 3000 (

**import** numpy **as** np  
**import** matplotlib.pyplot **as** plt  
  
**def** f(x, t):  
 **return** np.exp(-2\*t)\*np.sin(x)  
  
**def** f\_border\_0\_t(t):  
 **return** 0  
  
**def** f\_border\_pi\_t(t):  
 **return** np.exp(-2\*t)  
  
**def** f\_border\_x\_0(x):  
 **return** np.sin(x)  
  
  
**def** progonka(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):  
 N = len(x)  
 t1 = 0  
  
 U = np.zeros((K, N))  
 sig = 2 \* tau / h \*\* 2  
 **for** j **in** range(N):  
 U[0, j] = f\_border\_x\_0(x[j])  
  
 **for** k **in** range(K - 1):  
 t1 =+ tau  
 **for** j **in** range(1, N - 1):  
 U[k + 1, j] = sig \* (U[k, j + 1] - 2 \* U[k, j] + U[k, j - 1]) + U[k, j]  
  
  
 U[k + 1, 0] = np.exp(-2\*t[k+1])\*np.sin(x[0])  
 U[k + 1, N - 1] = np.exp(-2\*t[k+1])\*np.sin(x[N-1])  
  
  
 **return** U  
  
**def** implicit(K, t, tau, h, x):  
  
 N = len (x)  
  
 U = np.zeros((K, N))  
 sig = 2 \* tau / h \*\* 2  
  
 **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)  
 a[0] = 0  
 b[0] = 1  
 c[0] = 0  
 d[0] = 0  
 a[N - 1] = 0  
 b[N - 1] = 1  
 c[N - 1] = 0  
 d[N - 1] = np.exp(-2\*t[k+1])\*np.sin(x[N-1])  
 **for** j **in** range(1, N - 1):  
 a[j] = sig  
 b[j] = -(1 + 2 \* sig)  
 c[j] = sig  
 d[j] = -U[k, j]  
 U[k+1] = progonka(a, b, c, d, N)  
  
  
  
  
 **return** U  
  
  
**def** Krank\_Nikolson(K, t, tau, h, x, theta):  
 N = len(x)  
 U\_ex = explicit(K, t, tau, h, x)  
 U\_im = implicit(K, t, tau, h, x)  
  
 **if** theta == 0:  
 **return** U\_ex  
 **elif** theta == 1:  
 **return** U\_im  
 **else**:  
 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  
  
  
  
time = 1  
N = 50  
K = 10000  
h = (0.5 \* np.pi - 0) / N  
tau = time / K  
x = np.arange(0, 0.5 \* np.pi + h / 2 - 1e-4, h)  
x [N] = np.pi/2  
t = np.arange(0, time, tau)  
  
dt = 3000  
print(**'Проверка:'**)  
**if** (2\*tau/h\*\*2 <= 1/2):  
 print(2\*tau/h\*\*2,**' <= 1/2'**)  
**else**:  
 print(**'2\*tau/h\*\*2 > 1/2'**)  
U1 = explicit(K, t, tau, h, x)  
U2 = implicit(K, t, tau, h, x)  
U3 = Krank\_Nikolson(K, t, tau, h, x, 0.5)  
U\_analytic = f(x, t[dt])  
  
  
error1 = abs(U\_analytic - U1[dt, :])  
error2 = abs(U\_analytic - U2[dt, :])  
error3 = abs(U\_analytic - U3[dt, :])  
  
  
plt.title(**"График точного и численного решения задачи"**)  
plt.plot(x, U\_analytic, label=**"Точное решение"**, color=**"red"**)  
plt.scatter(x, U1[dt, :], label=**"Явный метод"**)  
plt.scatter(x, U2[dt, :], label=**"Неявный метод"**)  
plt.scatter(x, U3[dt, :], label=**"Кранк-Николсон"**)  
plt.xlabel(**"x"**)  
plt.ylabel(**"U"**)  
  
plt.legend()  
plt.grid()  
plt.show()  
  
print(**"Максимальная ошибка явного метода: "**, max(error1))  
print(**"Максимальная ошибка неявного метода: "**, max(error2))  
print(**"Максимальная ошибка метода Кранка - Николсона: "**, max(error3))  
  
  
  
H = np.zeros(3)  
E1 = np.zeros(3)  
E2 = np.zeros(3)  
E3 = np.zeros(3)  
n = 5  
**for** i **in** range (3):  
 n = int(n\*2)  
 h = np.pi / n  
 x = np.arange(0, np.pi, h)  
 H[i] = h  
 E1[i] = max(abs(f(x, t[dt]) - explicit(K, t, tau, h, x)[dt, :]))  
 E2[i] = max(abs(f(x, t[dt]) - implicit(K, t, tau, h, x)[dt, :]))  
 E3[i] = max(abs(f(x, t[dt]) - Krank\_Nikolson(K, t, tau, h, x,0.5)[dt, :]))  
  
**from** scipy.interpolate **import** PchipInterpolator  
  
Y1\_reverse = E1[::-1]  
Y2\_reverse = E2[::-1]  
Y3\_reverse = E3[::-1]  
X\_reverse = H[::-1]  
  
pchip\_reverse1 = PchipInterpolator(X\_reverse, Y1\_reverse)  
pchip\_reverse2 = PchipInterpolator(X\_reverse, Y2\_reverse)  
pchip\_reverse3 = PchipInterpolator(X\_reverse, Y3\_reverse)  
  
xnew\_reverse = np.linspace(min(X\_reverse), max(X\_reverse), 1000)  
ynew\_reverse1 = pchip\_reverse1(xnew\_reverse)  
ynew\_reverse2 = pchip\_reverse2(xnew\_reverse)  
ynew\_reverse3 = pchip\_reverse3(xnew\_reverse)  
  
plt.figure(figsize=(10, 6))  
  
plt.scatter(X\_reverse, Y1\_reverse, label=**'Погрешность явной схемы'**, zorder=6)  
plt.scatter(X\_reverse, Y2\_reverse, label=**'Погрешность неявной схемы'**, zorder=6)  
plt.scatter(X\_reverse, Y3\_reverse, label=**'Погрешность метод Кранка - Николсона'**, zorder=6)  
  
plt.plot(xnew\_reverse, ynew\_reverse1, label=**''**, linewidth=2)  
plt.plot(xnew\_reverse, ynew\_reverse2, label=**''**, linewidth=2)  
plt.plot(xnew\_reverse, ynew\_reverse3, label=**''**, linewidth=2)  
  
plt.title(**'График погрешности'**)  
plt.xlabel(**'h'**)  
plt.ylabel(**'error'**)  
plt.legend()  
plt.grid(**True**)  
  
plt.show()

### **Вывод программы**

Проверка:

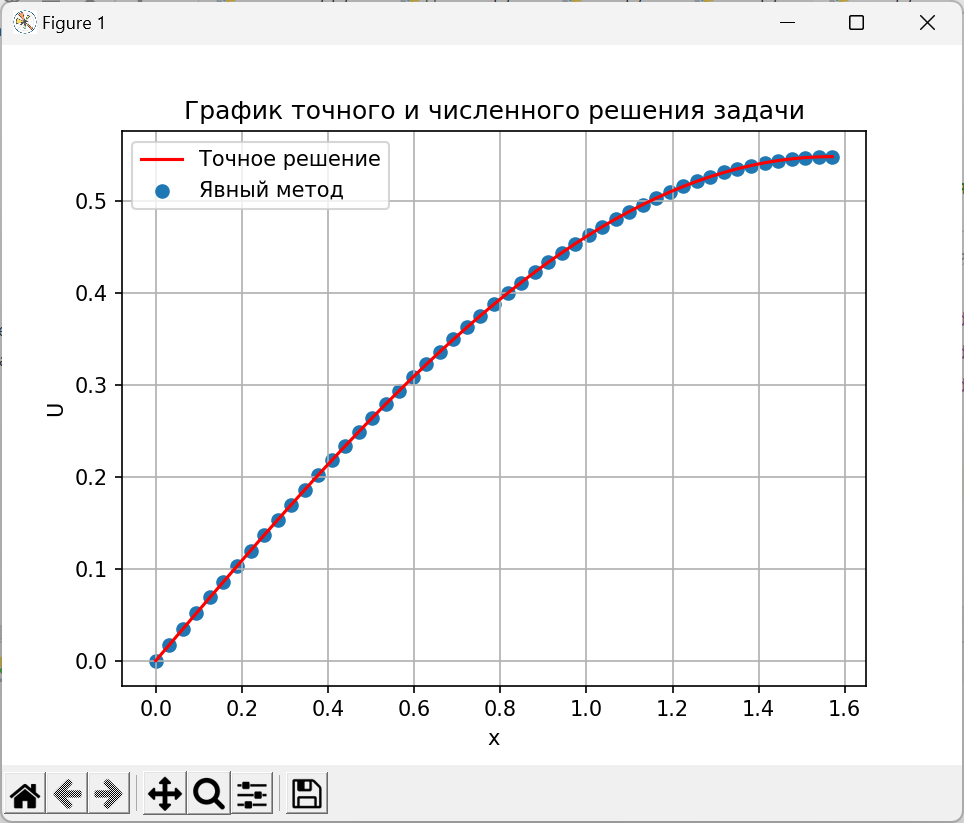
0.20264236728467555 <= 1/2

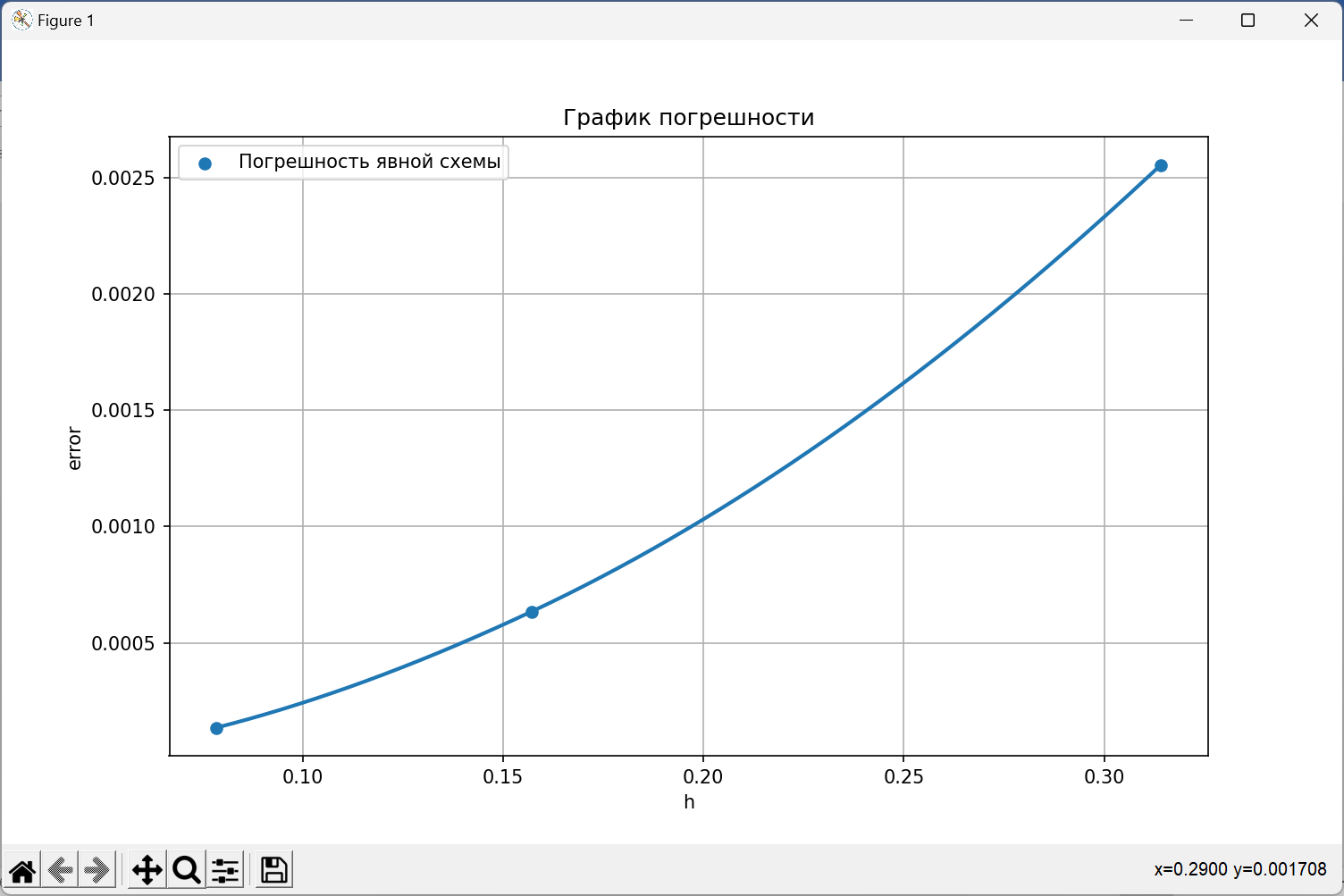
Максимальная ошибка явного метода: 2.283846856354632e-06

Максимальная ошибка неявного метода: 2.3443328241556927e-05

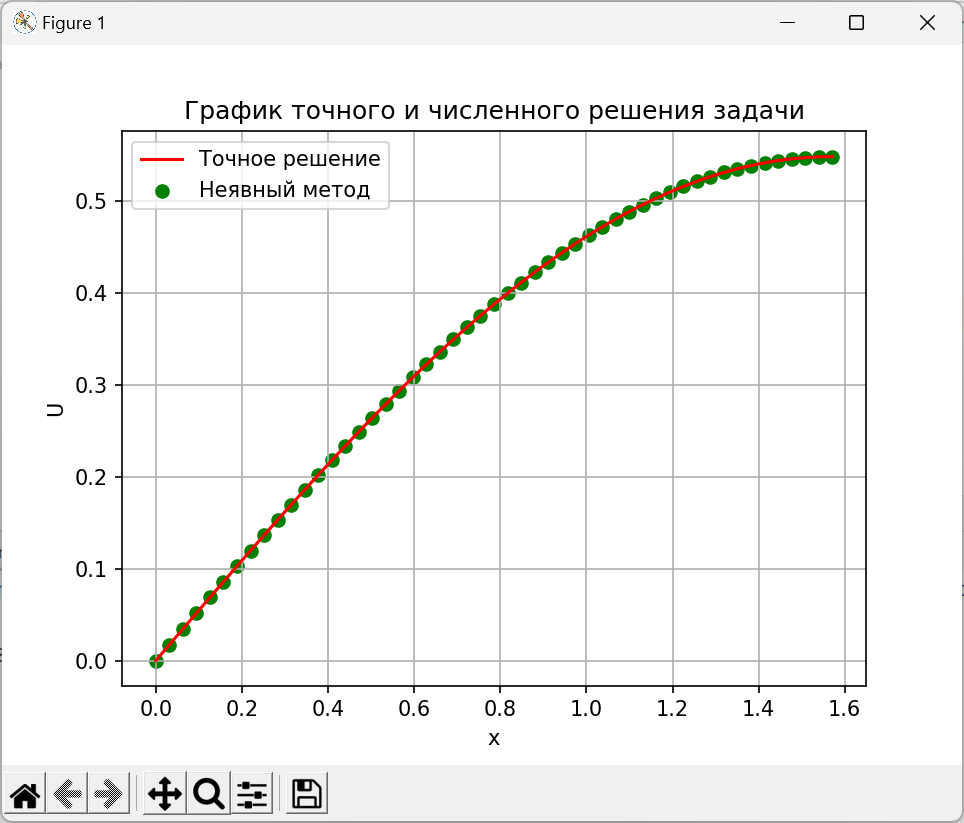
Максимальная ошибка метода Кранка - Николсона: 1.0579740692628903e-05

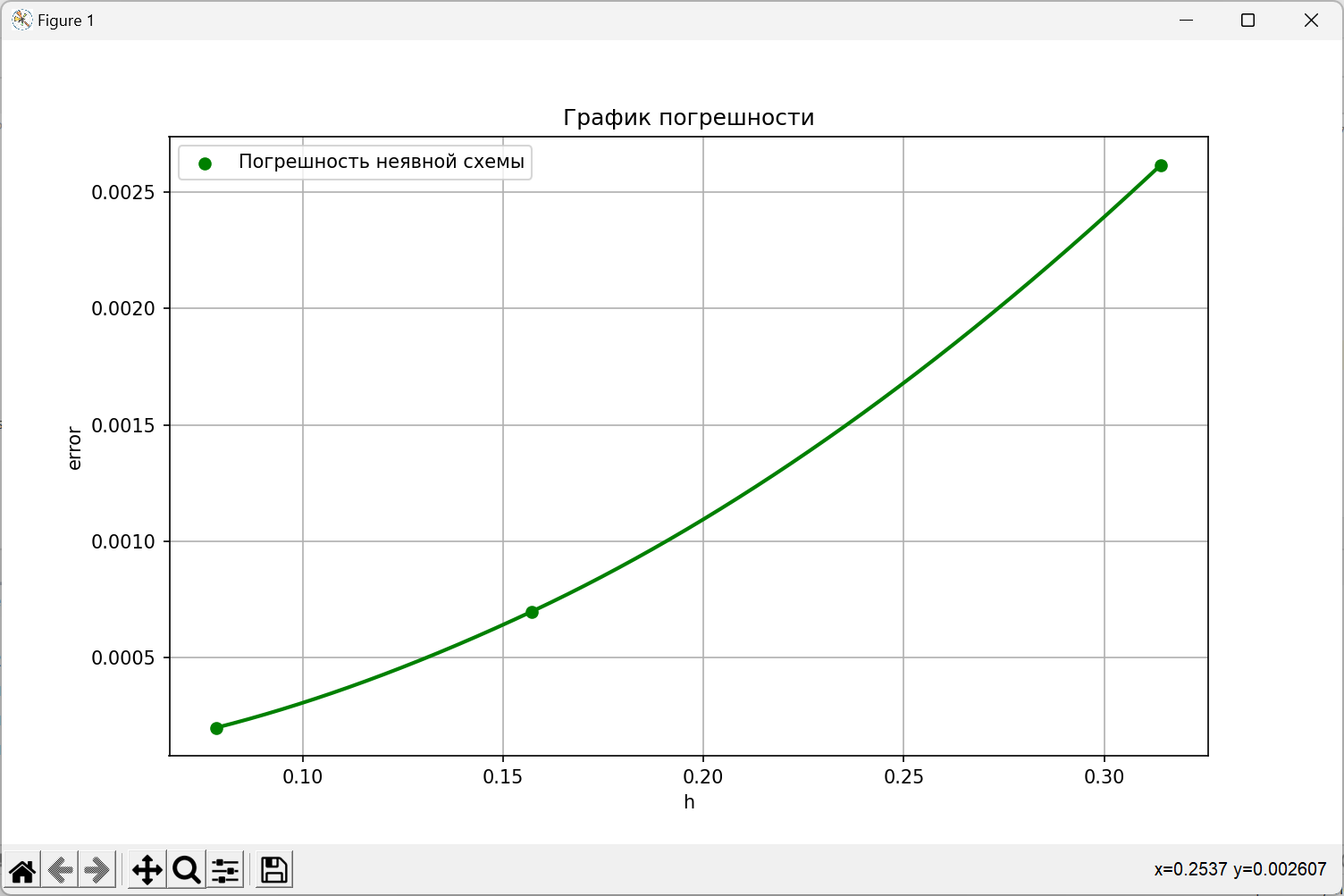
**Явная схема**



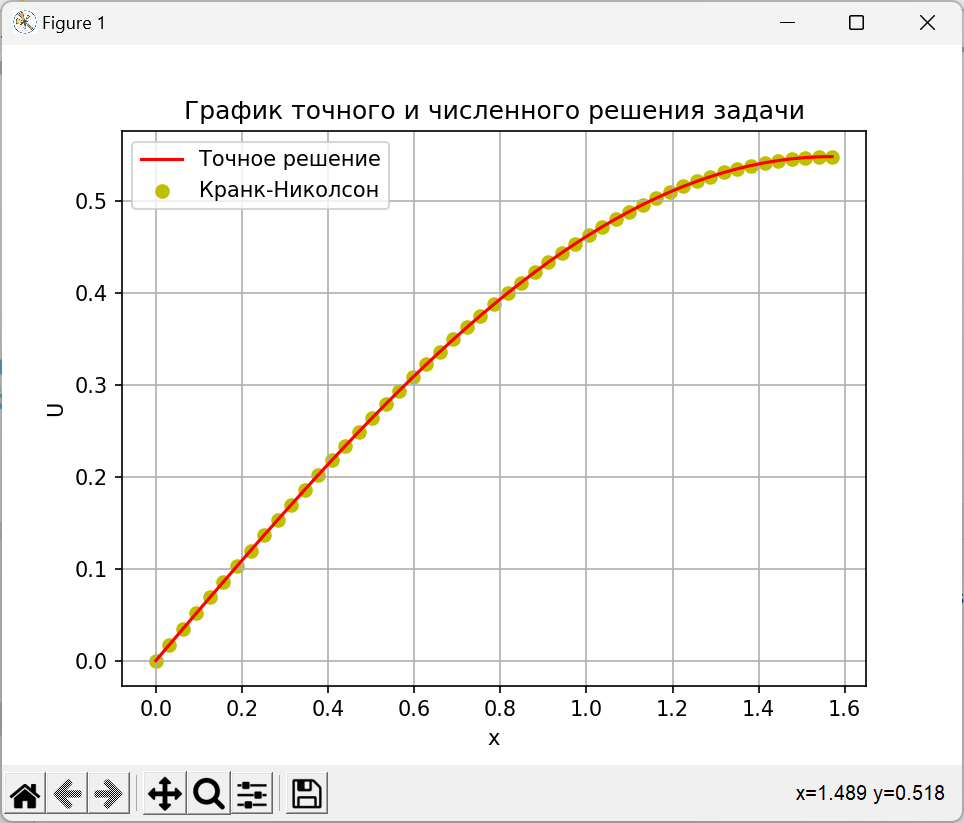


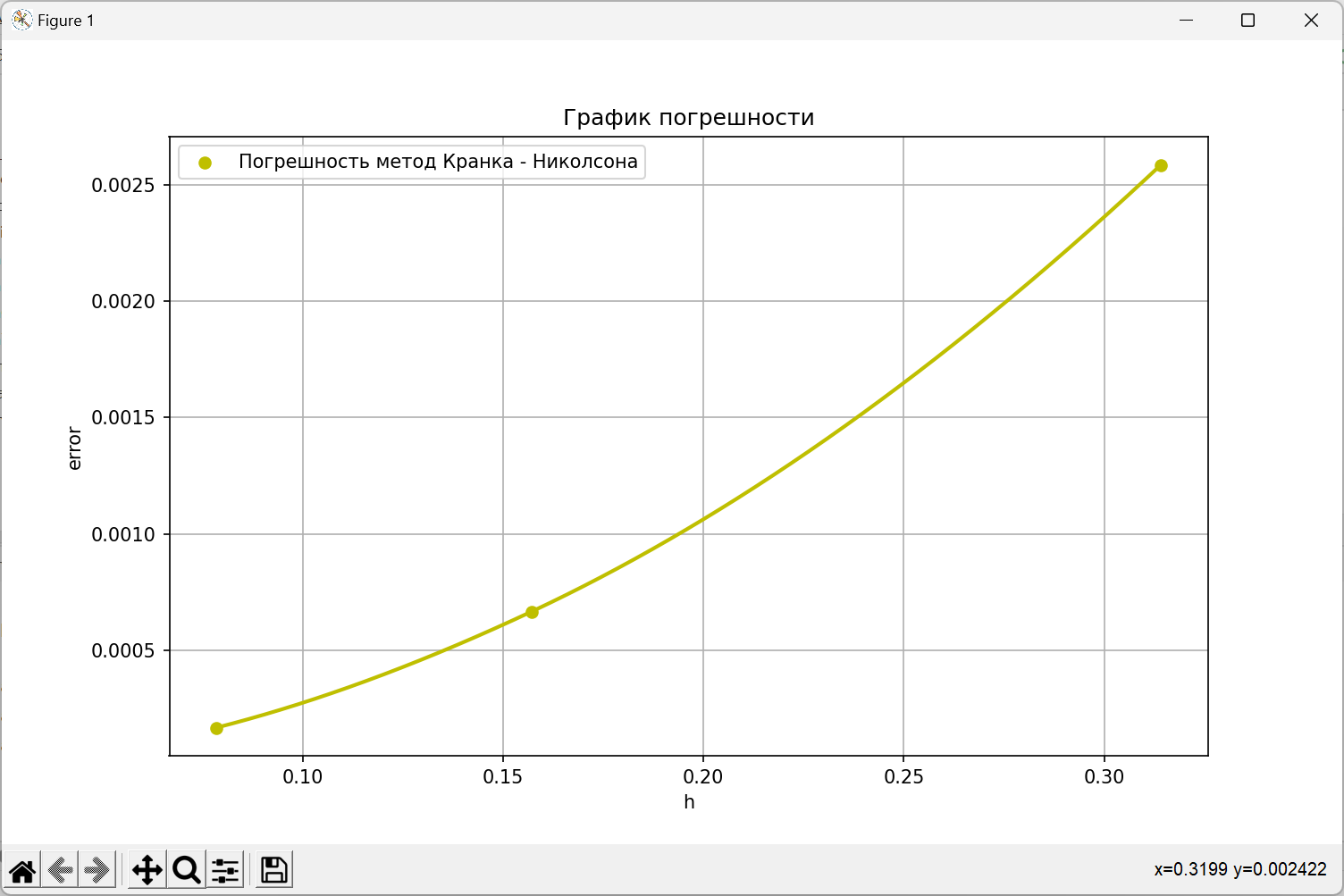
**Неявная схема**





**Схема Кранка-Николсона**





## 

## **Заключение:**

Явная схема оказалась эффективнее на мелких разбиениях времени, на

больших же ошибка чрезмерно возрастала в силу расходимости критерия

устойчивости.

Исходя из графика погрешности от длины шага можно сказать, что отклонение от искомого решения неявным методом в целом меньше, чем у явного и Кранка-Николсона. При этом не стоит забывать, что явный метод

применим только при выполнении критерия устойчивости, что накладывает

значительные ограничения на построение численного решения.