|  |  |
| --- | --- |
|  | |
| 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 №6**

Используя явную схему крест и неявную конечно-разностные схему решить начально-краевую задачу для дифференциального уравнения гиперболического типа. 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**

**Явная конечно-разностная схема «Крест»**

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:

Действуем способом, аналогичным тому, что применялся в предыдущей лабораторной работе: задаем пространственно-временную сетку и аппроксимируем производные в уравнении. Получаем явную конечно-разностную схему:

Граничные условия аппроксимируем с первым порядком точности:

В результате переход на новый временной слой представляется следующим алгоритмом:

где из граничных условий, .

В начальный момент времени значения определяются точно:

Воспользуемся аппроксимацией первого порядка по времени:

Явная схема условно устойчива с условием .

**Неявная конечно-разностная схема**

Для неявной схемы имеем СЛАУ, которая опять же решается прогонкой, так как полученная матрица является трёхдиагональной. Для этой СЛАУ:

В начальный момент времени значения определяются точно:

Воспользуемся аппроксимацией первого порядка по времени:

Граничные условия:

**Код программы**

В качестве параметров в коде были взяты числа N = 50, K = 10000,

**import** numpy **as** np, matplotlib.pyplot **as** plt  
  
  
**def** analyt\_func(x, t):  
 **return** np.sin(x - 2 \* t)  
  
  
**def** func\_border1(t):  
 **return** np.cos(2\*t)  
  
  
**def** func\_border2(t):  
 **return** np.sin(2\*t)+np.cos(2\*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  
 sig = t\*t\*4/(h\*h)  
 **for** j **in** range(N):  
 U[0, j] = np.sin(x[j])  
 **if** approx\_st == 1:  
 U[1][j] = np.sin(x[j]) - 2\*np.cos(x[j]) \* tau  
 **if** approx\_st == 2:  
 U[1][j] = np.sin(x[j]) - 2\*np.cos(x[j]) \* tau + 2\*np.sin(x[j]) \* tau \*\* 2 / 2  
  
 **for** k **in** range(1, K - 1):  
 t += tau  
 **for** j **in** range(1, N - 1):  
 U[k + 1, j] = sig\*U[k, j + 1] +2\*(1-sig)\*U[k, j]+sig\*U[k, j - 1]-U[k - 1, j]  
  
 **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])/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+2\*h)  
  
 **elif** approx\_bo == 3:  
 U[k + 1, 0] = (func\_border1(t) - h/tau/2\*U[k,0]-U[k+1,1]\*2\*4/h/2)/(-2\*4/h/2-h/tau/2)  
 U[k + 1, N - 1] = (func\_border2(t) + h\*U[k,N-1]/2/tau + 2\*4/h/2\*U[k+1,N-1])/(2\*4/h/2+h/2/tau)  
 U[k + 1, 0] = np.sin(x[0]-2\*t)  
 U[k + 1, N - 1] = np.sin(x[N-1]-2\*t)  
 **return** U  
  
  
**def** implicit(K, t, tau, h, x, approx\_st, approx\_bo):  
 N = len(x)  
 sig = 4 \* tau \*\* 2 / h \*\* 2  
 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]) - 2\*np.cos(x[j]) \* tau  
 **if** approx\_st == 2:  
 U[1][j] = np.sin(x[j]) - 2\*np.cos(x[j]) \* tau + 2\*np.sin(x[j]) \* 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] = sig  
 b[j] = -(1 + 2 \* sig)  
 c[j] = sig  
 d[j] = U[k-1,j]-2\*U[k,j]  
  
 **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] = -2 \* 4 / h / 2 - h / tau / 2  
 c[0] = 1 \* 2 \* 4 / h / 2  
 d[0] = func\_border1(t) - 1 \* h / tau / 2 \* U[k, 0]  
  
 a[N - 1] = -1 \* 2 \* 4 / h / 2  
 b[N - 1] = 2 \* 4 / h / 2 + h / tau / 2  
  
 d[N - 1] = func\_border2(t) + h/tau/2\*U[k,N-1]  
  
 b[0] = 1  
 c[0] = 0  
 d[0] = np.sin(x[0]-2\*t)  
 a[N - 1] = 0  
 b[N - 1] = 1  
 d[N - 1] = np.sin(x[N-1]-2\*t)  
  
 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 = (0.5\*np.pi - 0) / N  
 tau = time / K  
 x = np.arange(0, 0.5\*np.pi + h / 2 - 1e-4, h)  
 T = np.arange(0, time, tau)  
 t = 0  
  
 approx\_st = 1  
 approx\_bo = 2  
 dt = 100  
  
 print(**'Проверка:'**)  
 **if** (2 \* tau / h \*\* 2 <= 1):  
 print(2 \* tau / h \*\* 2, **' <= 1'**)  
 **else**:  
 print(**'2\*tau/h\*\*2 > 1'**)  
 U1 = implicit(K, t, tau, h, x, approx\_st, approx\_bo)  
 U2 = explicit(K, t, tau, h, x, approx\_st, approx\_bo)  
 U\_analytic = analyt\_func(x, T[dt])  
  
 error1 = abs(U\_analytic - U1[dt, :])  
 error2 = abs(U\_analytic - U2[dt, :])  
  
 plt.title(**"График точного и численного решения задачи"**)  
 plt.plot(x, U\_analytic, label=**"Точное решение"**, color=**"red"**)  
 plt.scatter(x, U2[dt, :], label=**"явная схема"**)  
 plt.scatter(x, U1[dt, :], label=**"неявная схема"**)  
 plt.xlabel(**"x"**)  
 plt.ylabel(**"y"**)  
 plt.grid()  
 plt.legend()  
 plt.show()  
  
 print(**"Максимальная ошибка явного метода: "**, max(error2))  
 print(**"Максимальная ошибка неявного метода: "**, max(error1))  
  
 H = np.zeros(3)  
 E1 = np.zeros(3)  
 E2 = np.zeros(3)  
 E3 = np.zeros(3)  
 n = 5  
 **for** i **in** range(2):  
 n = int(n \* 2)  
 h = np.pi / n  
 x = np.arange(0, np.pi, h)  
 H[i] = h  
 E1[i] = max(abs(analyt\_func(x, T[dt]) - explicit(K, t, tau, h, x, approx\_st, approx\_bo)[dt, :]))  
 E2[i] = max(abs(analyt\_func(x, T[dt]) - implicit(K, t, tau, h, x, approx\_st, approx\_bo)[dt, :]))  
  
 **from** scipy.interpolate **import** PchipInterpolator  
  
 Y1\_reverse = E1[::-1]  
 Y2\_reverse = E2[::-1]  
 X\_reverse = H[::-1]  
 x = np.arange(0, np.pi, np.pi / 40)  
 Y1\_reverse[0] =max(abs(analyt\_func(x, T[dt]) - explicit(K, t, tau, np.pi / 40, x, approx\_st, approx\_bo)[dt, :]))  
 Y2\_reverse[0] = max(abs(analyt\_func(x, T[dt]) - implicit(K, t, tau, np.pi / 40, x, approx\_st, approx\_bo)[dt, :]))  
  
 X\_reverse[0] = np.pi / 40  
 print(X\_reverse)  
  
 pchip\_reverse1 = PchipInterpolator(X\_reverse, Y1\_reverse)  
 pchip\_reverse2 = PchipInterpolator(X\_reverse, Y2\_reverse)  
  
 xnew\_reverse = np.linspace(min(X\_reverse), max(X\_reverse), 1000)  
 ynew\_reverse1 = pchip\_reverse1(xnew\_reverse)  
 ynew\_reverse2 = pchip\_reverse2(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.plot(xnew\_reverse, ynew\_reverse1, label=**''**, linewidth=2)  
 plt.plot(xnew\_reverse, ynew\_reverse2, label=**''**, linewidth=2)  
  
 plt.title(**'График погрешности'**)  
 plt.xlabel(**'h'**)  
 plt.ylabel(**'error'**)  
 plt.legend()  
 plt.grid(**True**)  
  
 plt.show()  
  
 **return** 0  
  
  
N = 50  
K = 7000  
time = 1  
main(N, K, time)

### **Вывод программы**

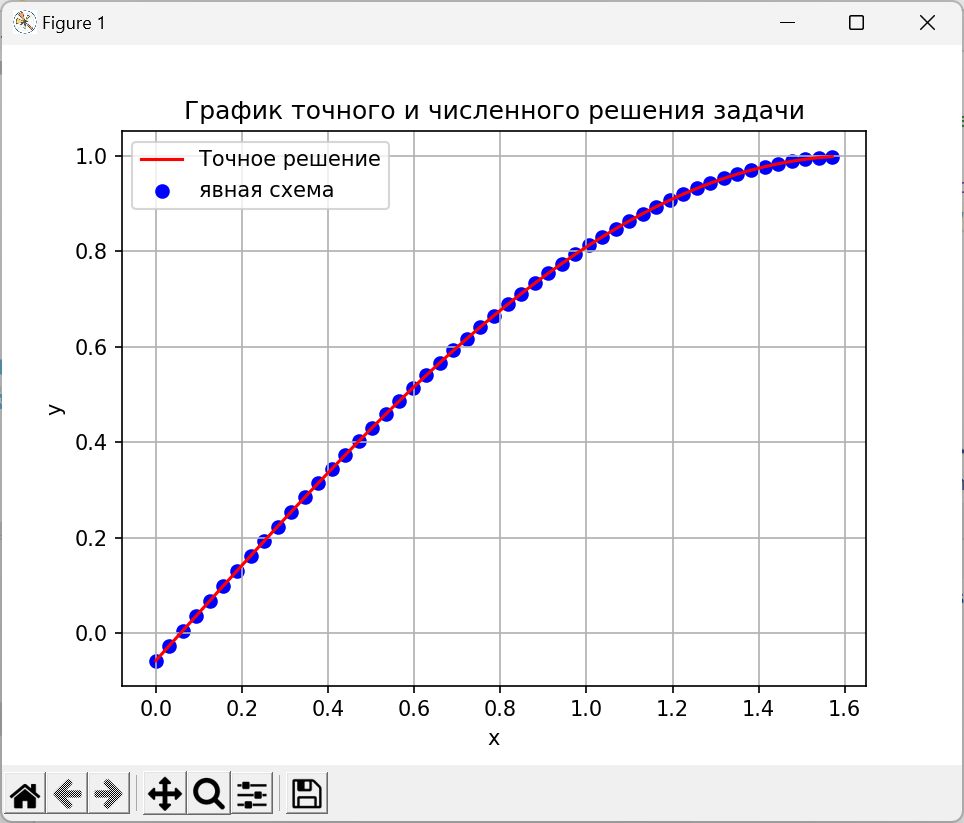
Проверка:

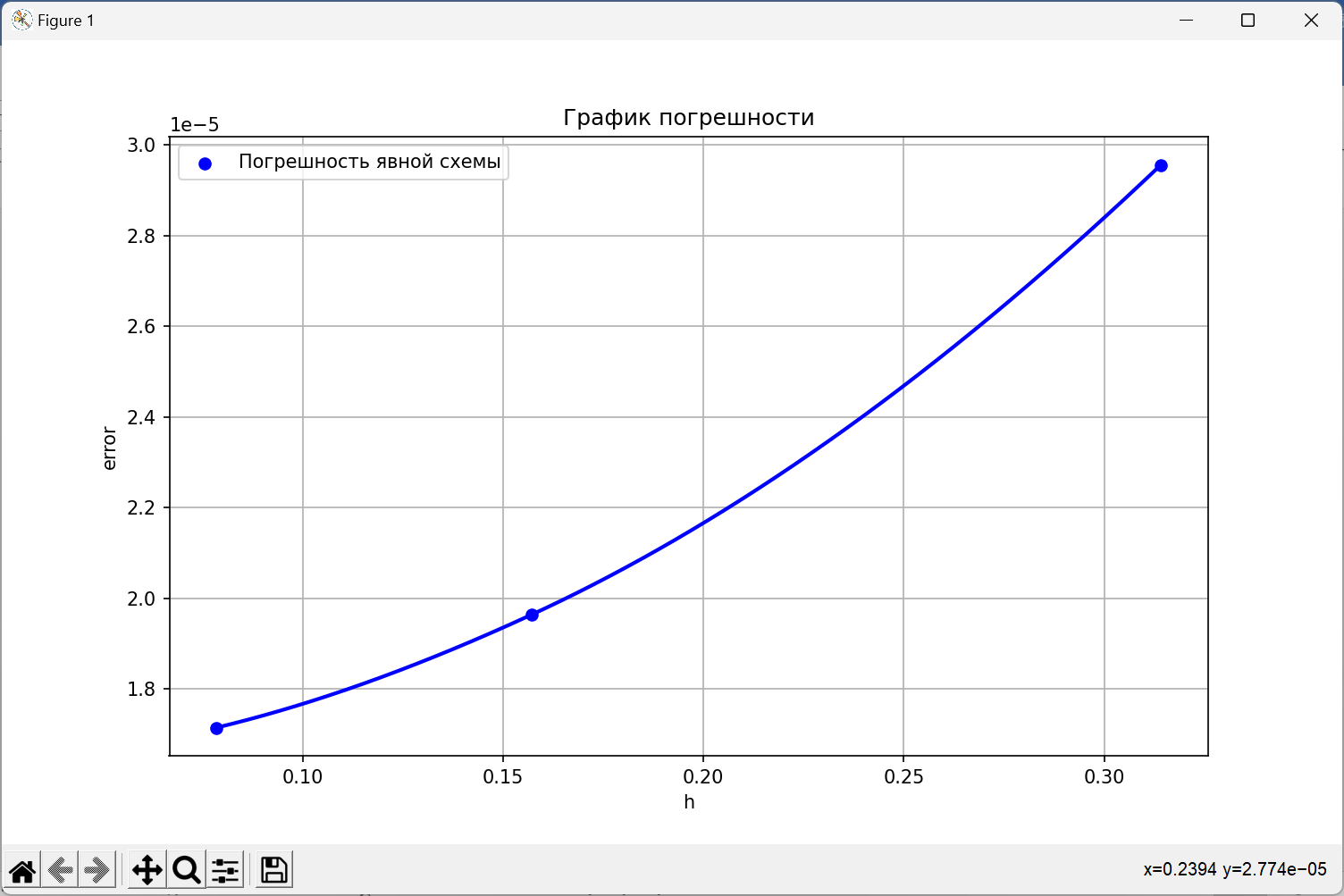
0.2894890961209651 <= 1

Максимальная ошибка явного метода: 4.095550484417565e-06

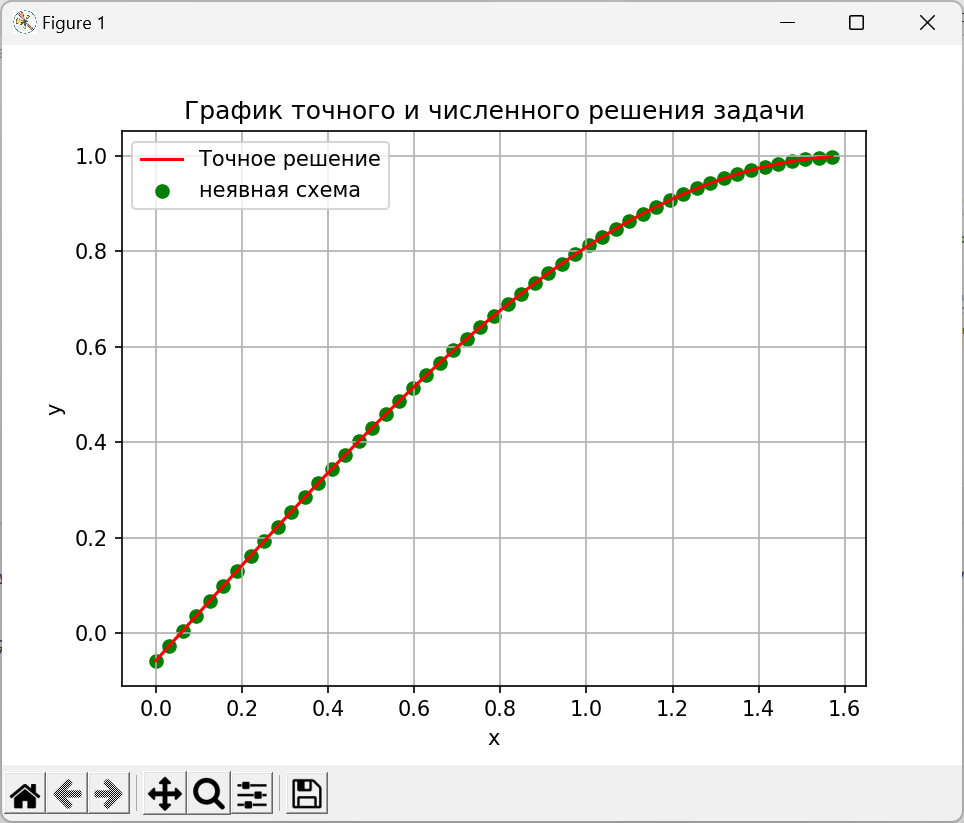
Максимальная ошибка неявного метода: 4.107426191612973e-06

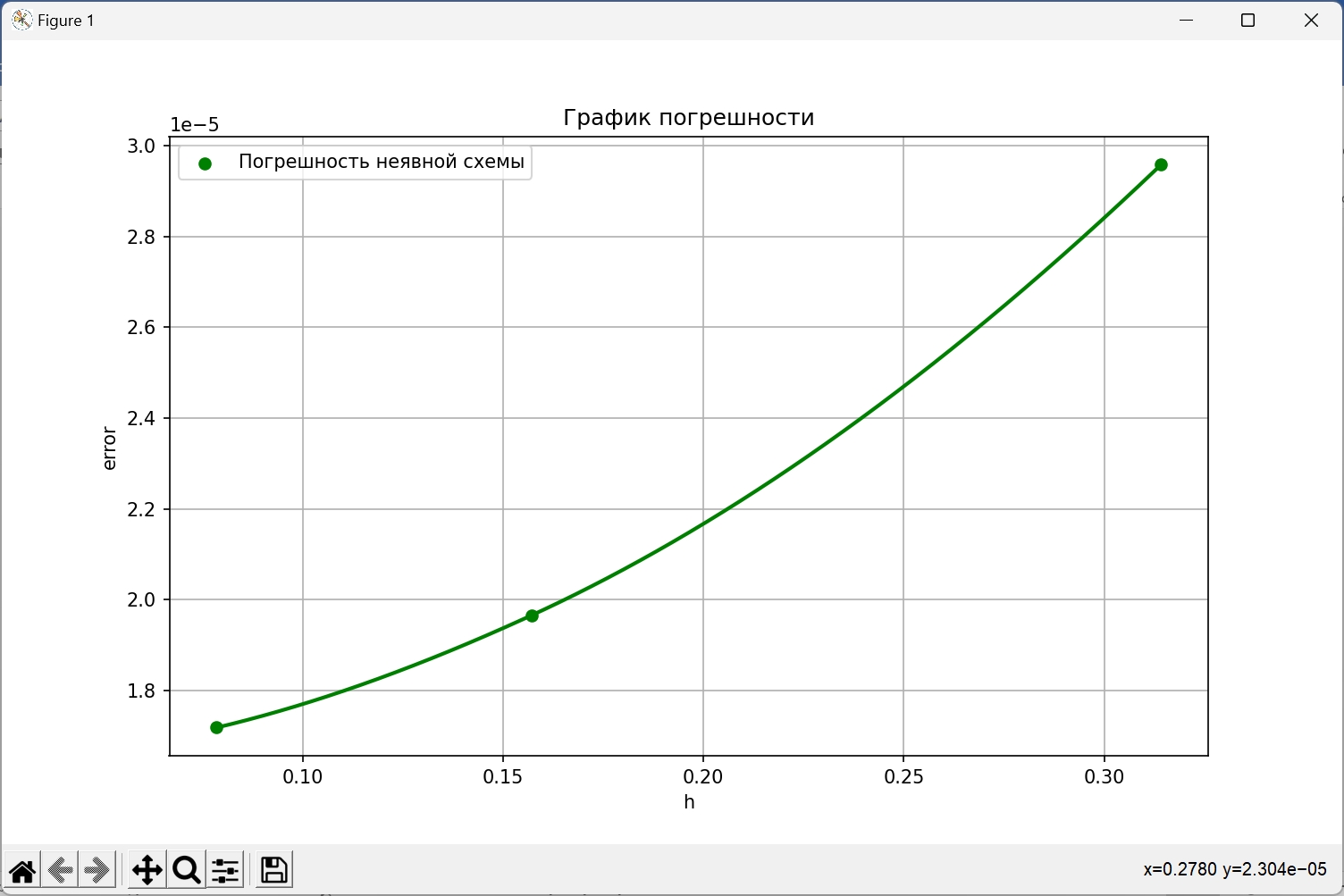
**Явная схема**





**Неявная схема**





**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. При этом не стоит забывать, что явный метод применим только при выполнении критерия устойчивости, что накладывает значительные ограничения на построение численного решения.