|  |  |
| --- | --- |
|  | |
| 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 17 | |
|  | |
|  | Выполнил: Черных С. Д. |
|  | 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 «Крест»**

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

В начальный момент времени значения определяются точно:

Воспользуемся аппроксимацией первого порядка по времени:

Граничные условия:

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 №6 (2).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. Исследовать зависимость погрешности от сеточных параметров.  import math import typing import matplotlib.pyplot as plt  $$u\_x(0, t) = - exp(-t) cos(2t) \\ u(\pi, t) = - exp(-t - \pi) cos(2t) \\ u(x, 0) = exp(-x) cos(x) \\ u\_t(x, 0) = -exp(-x) cos(x)$$  Аналитическое решение  $$ \alpha \frac{\partial u(0, t)}{\partial x} + \beta u(0, t) = \phi\_0 (t) \\ \\ \gamma \frac{\partial u(l, t)}{\partial x} + \delta u(l, t) = \phi\_l (t) $$ Входные условия left, right = 0, math.pi a = 1 b = 2 c = -3 d = 2 alpha = 1 betta = 0 gamma = 0 delta = 1  def phi\_0(t:float) -> float: #### diff  return -1 \* math.exp(-t) \* math.cos(2 \* t)   def phi\_l(t:float) -> float: #### not diff  return -1 \* math.exp(-t - math.pi) \* math.cos(2 \* t)   def psi\_1(x:float) -> float:  return math.exp(-x) \* math.cos(x)  def psi\_2(x:float) -> float:  return -1 \* math.exp(-x) \* math.cos(x)   def f(x:float, t:float) -> float:  return 0  def U(x:float, t:float) -> float:  return math.exp(-t - x) \* math.cos(x) \* math.cos(2 \* t)  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 функиции графики  def plot\_graphs(new\_X:list, new\_T:list, found\_U:list, U\_true:list, s:str='') -> None:  plt.plot(new\_X, U\_true[len(new\_T) // 4 ], label='Аналитическое решение' + ' t = '+ str(round(new\_T[len(new\_T) // 4 ], 4)))  plt.plot(new\_X, found\_U[len(new\_T) // 4 ], label=s + ' t = '+ str(round(new\_T[len(new\_T) // 4 ], 4)), linestyle='dashdot')  plt.plot(new\_X, U\_true[len(new\_T) // 2 ], label='Аналитическое решение' + ' t = '+ str(round(new\_T[len(new\_T) // 2 ], 4)))  plt.plot(new\_X, found\_U[len(new\_T) // 2 ], label=s + ' t = '+ str(round(new\_T[len(new\_T) // 2 ], 4)), linestyle='dashdot')  plt.plot(new\_X, U\_true[-1], label='Аналитическое решение' + ' t = '+ str(round(new\_T[-1], 4)))  plt.plot(new\_X, found\_U[-1],label=s + ' t = '+ str(round(new\_T[-1], 4)), linestyle='dashdot')  plt.legend()  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, d:float, alpha:float, betta:float, gamma:float, delta:float, find\_u:typing.Callable, t\_end:float=1, appoximation:int=1, appoximation\_start:float=1) -> (list, list): # H, error  H = [right/(n - 1) for n in N]  ERROR = []  for n in N:  XX, TT, UU = find\_u(left, right, a, b, c, d, alpha, betta, gamma, delta, n=n, t\_end=t\_end, appoximation=appoximation, appoximation\_start=appoximation\_start)  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 от h  def h\_error\_plot(H:list, ERROR:list) -> None:  plt.plot(H, ERROR)  plt.xlabel("h")  plt.ylabel("error")  plt.show()  рендж  def frange(start:float, stop:float, step:float) -> float:  while start < stop:  yield start  start += step  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)]  def get\_x(x\_0:float, x\_l:float, h:float) -> list:  return [i for i in frange(x\_0, x\_l+h, h)]  Прогонка  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]))   res = [Q[-1]]   for i in range(len(B) - 2, -1, -1):  res.append(P[i] \* res[-1] + Q[i])   return res[::-1] Явная схема крест def explicit\_cross(left:float, right:float, a:float, b:float, c:float, d:float, alpha:float, betta:float, gamma:float, delta:float, n:int=10, t\_end:float=1, appoximation:int=1, appoximation\_start:int=1) -> (list, list, list):  def approx(appoximation:int, k:float) -> None:  if appoximation == 1:  U\_values[k][0] = (h \* phi\_0(T[k]) - alpha \* U\_values[k][1]) / (betta \* h - alpha)  U\_values[k][-1] = (h \* phi\_l(T[k]) + gamma \* U\_values[k][-2]) / (gamma + delta \* h)  if appoximation == 2:  U\_values[k][0] = (2 \* h \* phi\_0(T[k]) - 4 \* alpha \* U\_values[k][1] + alpha \* U\_values[k][2]) / (2 \* betta \* h - 3 \* alpha)  U\_values[k][-1] = (2 \* h \* phi\_l(T[k]) + 4 \* gamma \* U\_values[k][-2] - gamma \* U\_values[k][-3]) / (2 \* delta \* h + 3 \* gamma)  if appoximation == 3:  U\_values[k][0] = (phi\_0(T[k]) \* h \* tau \* (2 \* a \*\* 2 - b \* h) - 2 \* alpha \* a \*\* 2 \* U\_values[k][1] \* tau - alpha \* h \*\* 2 \* U\_values[k - 1][0]) / (  -2 \* alpha \* a \*\* 2 \* tau - alpha \* h \*\* 2 + c \* h \*\* 2 \* tau + betta \* h \* tau \* (2 \* a \*\* 2 - b \* h))  U\_values[k][-1] = (phi\_l(T[k]) \* h \* tau \* (2 \* a \*\* 2 + b \* h) + 2 \* gamma \* a \*\* 2 \* U\_values[k][-2] \* tau + gamma \* h \*\* 2 \* U\_values[k - 1][ - 1]) / (  2 \* gamma \* a \*\* 2 \* tau + gamma \* h \*\* 2 - c \* h \*\* 2 \* tau + delta \* h \* tau \* (2 \* a \*\* 2 + b \* h))    n = n   h = right / (n - 1)  tau = (h\*\*2) / 2 / a\*\*2 # для того, чтобы выполнялось условие устойчивости  X = get\_x(left, right, h)  T = get\_t(0, t\_end, tau)  mu = b \* tau\*\*2 / 2 / h  sigma = alpha\*\*2 \* tau \*\* 2 / h\*\*2   U\_values = [[0] \* n for \_ in range(len(T))]   for j in range(n):  U\_values[0][j] = psi\_1(X[j])  if appoximation\_start == 1:  U\_values[1][j] = U\_values[0][j] + tau \* psi\_2(X[j])  if appoximation\_start == 2:  U\_values[1][j] = psi\_1(X[j]) + tau \* psi\_2(X[j]) + (tau\*\*2) \* (-d \* psi\_2(X[j])   + a \* (psi\_1(X[j]) - psi\_1(X[j] - h)) / h + b \* (psi\_2(X[j]) - psi\_2(X[j] - h)) / h + c \* psi\_1(X[j]) + f(X[j],0)) / 2   for k in range(2, len(T)):  for j in range(1, len(X) - 1):  U\_values[k][j] = ((sigma + mu) \* U\_values[k-1][j + 1] + (-2 \* sigma + 2 + c \* tau\*\*2) \* U\_values[k-1][j] + (sigma - mu) \* U\_values[k-1][j - 1] + (-1 + d\*tau/2) \* U\_values[k - 2][j] + tau\*\*2 \* f(X[j], T[k-1]))/(1 + d \* tau / 2)  approx(appoximation, k=k)    return X, T, U\_values  Демонстация работы  XX, TT, UU = explicit\_cross(left, right, a, b, c, d, alpha, betta, gamma, delta, n=20, t\_end=3, appoximation=2, appoximation\_start=2) U\_true = real\_U(XX, TT)  plot\_graphs(XX, TT, UU, U\_true, 'Явная схема крест')  N = [5, 10, 15, 20, 30, 40, 50] H, ERROR = get\_error\_array\_with\_h(N, left, right, a, b, c, d, alpha, betta, gamma, delta, explicit\_cross, t\_end=3, appoximation=1, appoximation\_start=1) h\_error\_plot(H, ERROR) Неявная схема крест def implicit\_cross(left:float, right:float, a:float, b:float, c:float, d:float, alpha:float, betta:float, gamma:float, delta:float, n:int=10, t\_end:float=1, appoximation:int=1, appoximation\_start:int=1) -> (list, list, list):  def approx(appoximation:int, k:int) -> None:  if appoximation == 1:  A0[-1] = -gamma/h  A1[0] = betta - alpha/h  A1[-1] = delta + gamma/h  A2[0] = alpha / h  B[0] = phi\_0(T[k])  B[-1] = phi\_l(T[k])  if appoximation == 2:  coeff = a / (2 \* h) / A2[1]  coeff\_ = -gamma / (2 \* h) / A0[-2]  A0[-1] = -2 \* gamma / h + coeff\_ \* A1[-2]  A1[0] = -3 \* alpha / (2 \* h) + betta + A0[1] \* coeff  A1[-1] = 3 \* gamma / (2 \* h) + delta + A2[-2] \* coeff\_  A2[0] = 2 \* alpha / h + A1[1] \* coeff  B[0] = phi\_0(T[k]) + B[1] \* coeff  B[-1] = phi\_l(T[k]) + B[-2] \* coeff\_  if appoximation == 3:  A0[-1] = 0  A1[0] = 2 \* a \*\* 2 / h + h / tau - c \* h - betta / alpha \* (2 \* a \*\* 2 - b \* h)  A1[-1] = delta  A2[0] = -2 \* a \*\* 2 / h  B[0] = h / tau \* U\_values[k-1][0] - phi\_0(T[k]) \* (2 \* a \*\* 2 - b \* h) / alpha  B[-1] = phi\_l(T[k])    n = n   h = right / (n - 1)  tau = (h\*\*2) / 2 / a\*\*2 # для того, чтобы выполнялось условие устойчивости  X = get\_x(left, right, h)  T = get\_t(0, t\_end, tau)     U\_values = [[0] \* n for \_ in range(len(T))]   for j in range(n):  U\_values[0][j] = psi\_1(X[j])  if appoximation\_start == 1:  U\_values[1][j] = U\_values[0][j] + tau \* psi\_2(X[j])  if appoximation\_start == 2:  U\_values[1][j] = psi\_1(X[j]) + tau \* psi\_2(X[j]) + (tau\*\*2) \* (-d \* psi\_2(X[j]) + a \* (  psi\_1(X[j]) - psi\_1(X[j] - h)) / h + b \* (psi\_2(X[j]) - psi\_2(X[j] - h)) / h + c \* psi\_1(X[j]) + f(X[j],0)) / 2    for k in range(2, len(T)):  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, len(X) - 1):  A0[j] = 2 \* a - h \* b  A1[j] = 2 \* (h\*\*2) \* (-d / (2 \* tau) - 1 / (tau \*\* 2) + c) - 4 \* a  A2[j] = 2 \* a + h \* b  B[j] = -4 \* (h\*\*2) \* U\_values[k-1][j] / (tau\*\*2) + (2 \* (h\*\*2) / (tau\*\*2) - d \* (h\*\*2) / tau) \* U\_values[k-2][j] - 2 \* (h\*\*2) \* f(X[j],T[k])  approx(appoximation, k)  U\_values[k] = solve\_PQ(A0, A1, A2, B)  return X, T, U\_values  XX2, TT2, UU2 = implicit\_cross(left, right, a, b, c, d, alpha, betta, gamma, delta, n=15, t\_end=3, appoximation=1, appoximation\_start=2) U\_true2 = real\_U(XX2, TT2)  plot\_graphs(XX2, TT2, UU2, U\_true2, 'Неявная схема крест')  N = [5, 10, 15, 20, 30, 40, 50] H, ERROR = get\_error\_array\_with\_h(N, left, right, a, b, c, d, alpha, betta, gamma, delta, implicit\_cross, t\_end=3, appoximation=1, appoximation\_start=2) h\_error\_plot(H, ERROR) |

### **Вывод программы**

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

|  |
| --- |
| 'Явная схема крест'    Ошибка в зависимотсти от h    'Неявная схема крест'    Ошибка в зависимотсти от h |

**Заключение:**

В данной лабораторной работе построены конечно-разностные схемы (явная, неявная) для дифференциальных уравнений гиперболического типа с одной пространственной и одной временной переменной.

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

Исходя из графиков зависимости погрешности от длины шага можно сказать, что для меньших длин шага неявный метод имеет меньшее отклонение от аналитического решения и несколько меньшее значение ошибки, чем явный метод. Во обоих методах зависимость ошибки от длины шага линейная, и чем больше шаг, тем больше ошибка. Также стоит учитывать, что явный метод применим только при выполнении критерия устойчивости.