|  |  |
| --- | --- |
|  | |
| 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 5 | |
|  | |
|  | Выполнил: Черных С. Д. |
|  | 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**

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:

Однако, учитывая условие устойчивости для явной конечно-разностной схемы, , в данной лабораторной работе был взят

Считая, что значения функции для всех координат

на временном слое известны,

попробуем определить значения функции на временном слое путем

разностной аппроксимации производной:

И одним из методов аппроксимации второй производной по 𝑥:

**Явная конечно-разностная схема**

Аппроксимируем вторую производную по значениям нижнего временного слоя

Тогда получим явную схему конечно-разностного метода во внутренних узлах сетки:

Обозначим , тогда:

Граничные значения и определяются граничными условиями

где

Явная схема является условно устойчивой, с условием .

**Неявная конечно-разностная схема**

Аппроксимируем вторую производную по значениям верхнего временного слоя

Тогда получим явную схему конечно-разностного метода во внутренних узлах сетки:

.

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

уравнений:

Первое и последнее уравнение системы, содержащие и определяются граничными условиями

где, в данном варианте,

Неявная схема является абсолютно устойчивой.

**Схема Кранка-Николсона**

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

значениями явной и неявной схемы, имеет смысл получить смешанную

аппроксимацию пространственных производных.

Явно-неявная схема для будет выглядеть

следующим образом:

При значении параметра схема являет собой схему Кранка-Николсона.

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

где

Схема Кранка-Николсона является абсолютно устойчивой.

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 №5 (1). 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, h.  вариант 5:  $ u(0, t)=0 $ $ u(1, t)=0 $ $ u(x, 0)=0 $  Аналитическое решение:  $$ U(x, t)= \frac{1}{\pi ^ 2}(1 - \exp{(-\pi ^ 2 t)}) \sin(\pi x)  Граничные условия первого рода при , в нашем случае  import matplotlib.pyplot as plt import math import typing  Существую граничные условия вида:  $$ \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, 1 alpha = 0 betta = 1 gamma = 0 delta = 1 a = 1 # u\_xx b = 0 # u\_x c = 0 #u  def f(x, t):  return math.sin(math.pi \* x)  def phi\_0(t):  return 0 def phi\_l(t):  return 0  def psi(x):  return 0  Аналитическое решение  def U(x, t):  return 1/(math.pi \*\* 2) \* (1 - math.exp((-math.pi \*\* 2) \* t)) \* math.sin(math.pi \* x)  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 функции Три графика  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 ], color='blue', label='Аналитическое решение')  plt.plot(new\_X, found\_U[len(new\_T) // 4 ], color='red', label=s, linestyle='dashdot')  plt.legend()  plt.text(0.05, 0.1, s='t = ' + str(new\_T[len(new\_T) // 4]))  plt.show()   plt.plot(new\_X, U\_true[len(new\_T) // 2 ], color='blue', label='Аналитическое решение')  plt.plot(new\_X, found\_U[len(new\_T) // 2 ], color='red', label=s, linestyle='dashdot')  plt.legend()  plt.text(0.05, 0.1, s='t = ' + str(new\_T[len(new\_T) // 2]))  plt.show()   plt.plot(new\_X, U\_true[len(new\_T) - 1], color='blue', label='Аналитическое решение')  plt.plot(new\_X, found\_U[len(new\_T) - 1], color='red', label=s, linestyle='dashdot')  plt.legend()  plt.text(0.05, 0.1, s='t = ' + str(new\_T[len(new\_T) - 1]))  plt.show()  рендж  def frange(start:float, stop:float, step:float) -> float:  while start < stop:  yield start  start += step  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, alpha:float, betta:float, gamma:float, delta:float, find\_u:typing.Callable, t\_end:float=1, appoximation:int=1, tetta:float=-10) -> (list, list): # H, error  H = [right/n for n in N]  ERROR = []  for n in N:  if tetta == -10:  XX, TT, UU = find\_u(left, right, a, b, c, alpha, betta, gamma, delta, n=n, t\_end=t\_end, appoximation=appoximation)  else:  XX, TT, UU = find\_u(left, right, a, b, c, alpha, betta, gamma, delta, n=n, t\_end=t\_end, appoximation=appoximation, tetta=tetta)  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 о h  def h\_error\_plot(H:list, ERROR:list) -> None:  plt.plot(H, ERROR)  plt.xlabel("h")  plt.ylabel("error")  plt.show()  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)] Явная конечно разностная схема Имеется уравнение вида  решения для уравнения имеет вид  приведём его к будее удобному виду для итерации  Для начальных условий  $ u^k\_0 = \phi\_0(t^k) $, $u^k\_n = \phi\_l(t^k) $, $ u^0\_j = \Psi{(x\_j)}$  Условие устойчивости  так как в моём варианте можно упростить условие до $\frac{ \tau}{h^2} < \frac{1}{2} \implies \tau <= \frac{h^2}{2} $  def find\_U\_explicit(left:float, right:float, a:float, b:float, c:float, alpha:float, betta:float, gamma:float, delta:float, n:int=10, t\_end:float=1, appoximation:int=1) -> (list, list, list): # добавить апроксимацию  def approx(appoximation:int, k:int):  if appoximation == 1:  U\_values[k][0] = (h \* phi\_0(T[k]) - alpha \* U\_values[k][1])/(h \* betta - alpha)  U\_values[k][-1] = (h \* phi\_l(T[k]) + gamma \* U\_values[k][-2] )/(h \* delta + gamma)  elif appoximation == 2:  U\_values[k][0] = (h \* (2 \* a - b \* h) \* phi\_0(T[k]) - 2 \* alpha \* a \* U\_values[k][1] - alpha \* (h\*\*2) \* U\_values[k-1][0] / tau) / (alpha \* c \* (h\*\*2) + betta \* h \* (2 \* a - b \* h) - 2 \* alpha \* a - alpha \* (h\*\*2) / tau)  U\_values[k][-1] = (h \* (2 \* a + b \* h) \* phi\_l(T[k]) + 2 \* gamma \* a \* U\_values[k][-2] + gamma \* (h\*\*2) \* U\_values[k-1][-1] / tau) / (2 \* gamma \* a + gamma \* (h\*\*2) / tau - c \* gamma \* (h\*\*2) + delta \* h \* (2 \* a + b \* h))  elif appoximation == 3:  U\_values[k][0] = (2 \* h \* phi\_0(T[k]) - 4 \* alpha \* U\_values[k][1] + alpha \* U\_values[k][2]) / (2 \* h \* betta - 3\*alpha)  U\_values[k][-1] = (2 \* h \* phi\_l(T[k]) + 4 \* gamma \* U\_values[k][-2] - gamma \* U\_values[k][-3]) / (2 \* h \* delta + 3 \* gamma)    n = n + 1 # n + начальные условия  h = right / (n - 1)  tau = (h\*\*2) / 2 / a \*\* 2 # для того, чтобы выполнялось условие устойчивости  sigma = tau / h\*\*2  X = get\_x(left, right, h)  T = get\_t(0, t\_end, tau)    m = len(T)  U\_values = [[0] \* n for \_ in range(m)] # cеточные значения     for j in range(n):  U\_values[0][j] = psi(X[j])   for j in range(m):  U\_values[j][0] = phi\_0(T[j])  U\_values[j][n-1] = phi\_l(T[j])    for k in range(1, m):  for j in range(1, n-1):  U\_values[k][j] = sigma \* U\_values[k-1][j + 1] + (1 - 2 \* sigma) \* U\_values[k-1][j] + sigma \* U\_values[k-1][j-1] + tau \* f(X[j], T[k-1])    approx(appoximation, len(T) - 1)    return X, T, U\_values  демонтрация работы алгоритма  XX, TT, UU = find\_U\_explicit(left, right, a, b, c, alpha, betta, gamma, delta, n=25, t\_end=15, appoximation=1) U\_true = real\_U(XX, TT)  plot\_graphs(XX, TT, UU, U\_true, 'Явная конечно разностная схема')  Ошибка и её значение от h  N = [5, 10, 15, 20, 25, 30, 35, 40] H, ERROR = get\_error\_array\_with\_h(N, left, right, a, b, c, alpha, betta, gamma, delta, find\_U\_explicit, t\_end=1, appoximation=1) h\_error\_plot(H, ERROR) Неявная конечно разностная схема решения для уравнения имеет вид  приведём его к будее удобному виду для итерации  Для начальных условий  $ u^{k+1}\_0 = \phi\_0(t^{k+1}) $, $u^{k+1}\_n = \phi\_l(t^{k+1}) $, $ u^0\_j = \Psi{(x\_j)}$  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]  def find\_U\_implicit(left:float, right:float, a:float, b:float, c:float, alpha:float, betta:float, gamma:float, delta:float, n:int=10, t\_end:float=1, appoximation:int=1) -> (list, list, list):  def approx(appoximation:int, k:int):  if appoximation == 1:  A1[0] = (betta - alpha) / h  A2[0] = alpha / h  B[0] = phi\_0(T[k+1]) / (betta - alpha / h)   A0[-1] = -gamma / h  A1[-1] = delta + gamma / h  B[-1] = phi\_l(T[k+1]) / (delta + gamma / h)  if appoximation == 2:    A1[0] = 2 \* a / h + h / tau - c \* h - betta \* (2 \* a - b \* h) / alpha  A2[0] = - 2 \* a / h  B[0] = h \* U[k-1][0] / tau - phi\_0(T[k]) \* (2 \* a - b \* h) / alpha   A0[-1] = - 2 \* a / h  A1[-1] = 2 \* a / h + h / tau - c \* h + delta \* (2 \* a + b \* h) / gamma  B[-1] = h \* U[k-1][-1] / tau + phi\_l(T[k]) \* (2 \* a + b \* h) / gamma  if appoximation == 3:  A1[0] = 2 \* betta \* h - 3 \* alpha  A2[0] = 4 \* alpha  B[0] = 2 \* h \* phi\_0(T[k+1])   A0[-1] = -4 \* gamma  A1[-1] = 2 \* h \* delta + 3 \* gamma  B[-1] = 2 \* h \* phi\_l(T[k+1])   if alpha == 0 and appoximation == 2:  appoximation = 3  n = n + 1 # n + начальные условия  h = right / (n - 1)  tau = (h\*\*2) / 2 / a\*\*2 # для того, чтобы выполнялось условие устойчивости  sigma = tau / h\*\*2  X = get\_x(left, right, h)  T = get\_t(0, t\_end, tau)     m = len(T)  U\_values = [[0] \* n for \_ in range(m)] # cеточные значения    for k in range(m-1):  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, n-1):  A0[j] = sigma  A1[j] = -(1 + 2\*sigma)  A2[j] = sigma  B[j] = -U\_values[k][j] - f(X[j], T[k]) \* tau  approx(appoximation, k)  U\_values[k+1] = solve\_PQ(A0, A1, A2, B)  return X, T, U\_values  Демонстрация работы  XX2, TT2, UU2 = find\_U\_implicit(left, right, a, b, c, alpha, betta, gamma, delta, n=25, t\_end=15, appoximation=1) U\_true2 = real\_U(XX2, TT2)  plot\_graphs(XX2, TT2, UU2, U\_true2, 'Неявная конечно разностная схема')  N = [5, 10, 15, 20, 25, 30, 35, 40] H, ERROR = get\_error\_array\_with\_h(N, left, right, a, b, c, alpha, betta, gamma, delta, find\_U\_implicit, t\_end=1, appoximation=1) h\_error\_plot(H, ERROR) Cхема Кранка - Николсона def Krank\_Nikolson(left:float, right:float, a:float, b:float, c:float, alpha:float, betta:float, gamma:float, delta:float, n:int=10, t\_end:float=1, appoximation:int=1, tetta:float=0.3) -> list:  if tetta < 0 or tetta > 1:  print('Тетта лежит на отрезке [0, 1]')  return  if alpha == 0 and appoximation == 2:  appoximation = 3  X, T, U\_explicit = find\_U\_explicit(left, right, a, b, c, alpha, betta, gamma, delta, n, t\_end, appoximation=appoximation)  X2, T2, U\_implicit = find\_U\_implicit(left, right, a, b, c, alpha, betta, gamma, delta, n, t\_end, appoximation=appoximation)  m = len(U\_implicit)  n = len(U\_implicit[0])  U\_values = [[0] \* n for \_ in range(m)]   for k in range(m):  for j in range(n):  U\_values[k][j] = tetta \* U\_implicit[k][j] + (1 - tetta) \* U\_explicit[k][j]    return X2, T2, U\_values  Демонстрация с разными параметрами  tetta = 0.5  XX3, TT3, UU3 = Krank\_Nikolson(left, right, a, b, c, alpha, betta, gamma, delta, n=25, t\_end=20, appoximation=1, tetta=tetta) U\_true2 = real\_U(XX3, TT3)  plot\_graphs(XX2, TT2, UU2, U\_true2, 'Схема Кранка-Николсона')  N = [5, 10, 15, 20, 25, 30, 35, 40] H, ERROR = get\_error\_array\_with\_h(N, left, right, a, b, c, alpha, betta, gamma, delta, Krank\_Nikolson, t\_end=1, appoximation=1, tetta=tetta) h\_error\_plot(H, ERROR) |

**Вывод программы**

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

|  |
| --- |
| 'Явная конечно-разностная схема'        Ошибка и её значение от h      'Неявная конечно-разностная схема'        Ошибка и её значение от h    'Схема Кранка-Николсона'        Ошибка и её значение от h |

## **Заключение**

В данной лабораторной работе построены конечно-разностные схемы (явная, неявная, Кранка Николсона) для дифференциальных уравнений параболического типа с одной пространственной и одной временной переменной с граничными условиями первого рода.

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

Во всех трёх методах зависимость ошибки от длины шага линейная, и чем больше шаг, тем больше ошибка. Используя явный метод, стоит учитывать, что он применим только при выполнении критерия устойчивости.

Схема Кранка-Николсона работает точнее, чем неявная схема, а неявная точнее, чем явная. Зависимость погрешности схем от пространственного шага напоминает квадратичную, что подтверждает аппроксимационные свойства этих схем .