|  |  |
| --- | --- |
|  | |
| 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 10 | |
|  | |
|  | Выполнил: Черных С. Д. |
|  | Группа: М8О-405Б-20 |
|  | Проверил: доц. Иванов И. Э. |
|  | Дата: |
|  | Оценка: |
|  | |
| Москва, 2023 | |

# **Лабораторная работа №8**

Используя схемы переменных направлений и дробных шагов, решить двумерную начально-краевую задачу для дифференциального уравнения параболического типа. В различные моменты времени вычислить погрешность численного решения путем сравнения результатов с приведенным в задании аналитическим решением U(x,y,t). Исследовать зависимость от сеточных параметров.

в)

Аналитическое решение:

### **Теоретические сведения**

**Метод переменных направлений**

Разобьём шаг по времени на два шага. На первом дробном временном слое неявно аппроксимируем дифференциальный оператор , а на следующем слое аппроксимируем уже явно дифференциальный оператор . Получаем следующую схему:

На первом дробном шаге с помощью *J − 1* скалярных прогонок в направлении *x* получаем распределение сеточной функции на первом временном полуслое. На втором шаге с помощью *I−1* скалярных прогонок в направлении y получаем распределение сеточной функции на втором временном полуслое.

На двух границах области заданы условия первого рода, на них сеточная функция определяется однозначно для любого момента времени. На двух “правых” границах заданы граничные условия второго рода, аппроксимируем дифференциальные операторы следующим образом:

Сеточная функция на нулевом слое определяется начальным условием.

Схема имеет второй порядок точности по времени и является абсолютно устойчивой (двумерный случай).

**Метод дробных шагов**

Этот метод использует только неявные конечно разностные-операторы. Разобьём шаг по времени на два шага . На первом дробном шаге проводится аппроксимация только одного из пространственных дифференциальных операторов, например . Тогда на следующем дробном шаге аппроксимируется . Получаем следующую схему:

Такая схема имеет первый порядок точности по времени и является абсолютно устойчивой.

**Код программы**

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**.**

|  |
| --- |
| Используя схемы переменных направлений и дробных шагов, решить двумерную начально-краевую задачу для дифференциального уравнения параболического типа. В различные моменты времени вычислить погрешность численного решения путем сравнения результатов с приведенным в задании аналитическим решением .  Исследовать зависимость погрешности от сеточных параметров , , .  $$ u(0, y, t) = 0 \ u\_x(\pi, y, t) = -\sin{y} \sin{\mu t} \ u(x, 0, t) = 0 \ u\_y(x, \pi, t) = -\sin{x} \sin{\mu t} \ u(x, y, 0) = 0  В)  аналитическое решение  import math import typing from typing import List import matplotlib.pyplot as plt import copy import plotly.graph\_objects as go from functools import lru\_cache Входные условия x\_left = 0 x\_right = math.pi y\_left = 0 y\_right = math.pi  a = 1 b = 2 mu = 1  def phi\_1(y:float, t:float, mu:float=1) -> float:  return 0  def phi\_2(y:float, t:float, mu:float=1) -> float:  return -math.sin(y) \* math.sin(mu \* t)   def phi\_3(x:float, t:float, mu:float=1) -> float:  return 0  def phi\_4(x:float, t:float, mu:float=1) -> float:  return -math.sin(x) \* math.sin(mu \* t)  def psi(x:float, y:float, mu:float=1) -> float:  return 0  def f(x:float, y:float, t:float, mu:float=1, a:float=1, b:float=2) -> float:  return math.sin(x) \* math.sin(y) \* (mu \* math.cos(mu \* t) + (a + b) \* math.sin(mu \* t))   def U(x:float, y:float, t:float, mu:float=1) -> float:  return math.sin(x) \* math.sin(y) \* math.sin(mu \* t)  def real\_U(X:List[float], Y:List[float], T:List[float]) -> List[List[List[float]]]:  U\_true = [[[0] \* len(X) for \_ in range(len(Y))] for \_\_ in range(len(T))]  for i in range(len(T)):  for k in range(len(Y)):  for j in range(len(X)):  U\_true[i][k][j] = U(X[j], Y[k], T[i])  return U\_true Вспомогательные функиции def local\_error (U\_my:list, U\_true:list) -> float:  return sum([abs(a - b) for a, b in zip(U\_my, U\_true)]) / len(U\_my)  def get\_error\_array\_with\_h(N:list, x\_left:float, x\_right:float, y\_left:float, y\_right:float, a:float, b:float,   mu:float, find\_u:typing.Callable,t\_end:float=1) -> (list, list): # H, error     H\_x = [x\_right/(n - 1) for n in N]  H\_y = [y\_right/(n - 1) for n in N]  ERROR\_X = []  ERROR\_Y = []  for n in N:  XX, YY, TT, UU = find\_u(x\_left, x\_right, y\_left, y\_right, a, b, mu, n\_x=n, n\_y=n, t\_end=t\_end)  U\_true = real\_U(XX, YY, TT)  t = len(TT) // 2  y = len(YY) // 2  x = len(XX) // 2  ERROR\_X.append(local\_error(UU[t][y], U\_true[t][y]))  ERROR\_Y.append(local\_error(UU[t][:][x], U\_true[t][:][x]))    return H\_x, H\_y, ERROR\_X, ERROR\_Y  def h\_error\_plot(H:list, ERROR:list, s:str=' x') -> None:  plt.plot(H, ERROR)  plt.xlabel("h" + s)  plt.ylabel("error")  plt.show()  def frange(start:float, stop:float, step:float) -> float:  while start < stop:  yield start  start += step  def get\_y(y0:float, y\_end:float, h:float) -> List[float]:  return [i for i in frange(y0, y\_end+h, h)]  def get\_x(x\_0:float, x\_l:float, h:float) -> List[float]:  return [i for i in frange(x\_0, x\_l+h, h)]  def get\_t(t\_0:float, t\_end:float, h:float) -> List[float]:  return [i for i in frange(t\_0, t\_end+h, h)]  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 border\_conds(U\_values:List[List[float]], X:List[float], Y:List[float], T:List[float], mu:float, hx:float, hy:float) -> None:   for i in range(len(Y)):  for i in range(len(X)):  U\_values[0][i][j] = psi(X[j], Y[i], mu=mu)  for k in range(len(T)):  for i in range(len(X)):  U\_values[k][i][0] = phi\_3(X[i], T[k], mu=mu)  U\_values[k][i][-1] = phi\_4(X[i], T[k], mu=mu) \* 2 \* hy + U\_values[k][i][-2]  for j in range(len(Y)):  U\_values[k][0][j] = phi\_1(Y[j], T[k], mu=mu)  U\_values[k][-1][j] = phi\_2(Y[j], T[k], mu=mu) \* 2 \* hx + U\_values[k][-2][j]  def initial\_T(U\_values:List[List[float]], X:List[float], Y:List[float], mu:float) -> None:  for j in range(len(Y)):  for i in range(len(X)):  U\_values[0][j][j] = psi(X[i], Y[j], mu=mu) Метод переменных направлений идём по j = по x неявно, по y явно  идём по i = по x явно, по y неявно  def changing\_directions(x\_left:float, x\_right:float, y\_left:float, y\_right:float, a:float, b:float, mu:float,   n\_x:float=10, n\_y:float=10, t\_end:float=1) -> (List[float], List[float], List[List[float]], List[List[float]]):   t\_right = t\_end  hx = x\_right / (n\_x - 1)  hy = y\_right / (n\_y - 1)  tau = (hx \*\* 2) / 2 / a\*\*2     X = get\_x(x\_left, x\_right, hx)  Y = get\_y(y\_left, y\_right, hy)  T = get\_t(0, t\_right, tau)    U\_values = [[[0] \* len(X) for \_ in range(len(Y))] for \_\_ in range(len(T))]  sigma\_x = (a \* tau) / (hx \*\* 2)  sigma\_y = (b \* tau) / (hy \*\* 2)    initial\_T(U\_values, X, Y, mu)  A0 = [0 for \_ in range(len(X))]  A1 = [0 for \_ in range(len(X))]  A2 = [0 for \_ in range(len(X))]  B = [0 for \_ in range(len(X))]  for k in range(1, len(T)):  U\_part = [[0 for \_ in range(len(Y))] for \_ in range(len(X))]   for i in range(1, len(X) - 1):  A1[0] = -1  A2[0] = 0  B[0] = phi\_1(Y[0], T[k-1] + tau/2, mu=mu)   for j in range(1, len(Y) - 1):  A0[j] = - sigma\_x / 2  A1[j] = 1 + sigma\_x  A2[j] = - sigma\_x /2  B[j] = f(X[i], Y[j], T[k-1] + tau/2, mu=mu) \* tau / 2 + sigma\_y \* (U\_values[k-1][j+1][i] - 2 \* U\_values[k-1][j][i] + U\_values[k-1][j-1][i]) / 2 + U\_values[k-1][j][i]  A0[-1] = 0  A1[-1] = -1  A2[-1] = 0  B[-1] = phi\_2(Y[-1], T[k-1] + tau/2, mu=mu) \* 2 \* hx + U\_values[k-1][j][-2]   tmp\_res = solve\_PQ(A0, A1, A2, B)   U\_part[i] = tmp\_res   for j in range(1, len(Y) - 1):  A1[0] = -1  A2[0] = 0  B[0] = phi\_3(X[0], T[k], mu=mu)  for i in range(1, len(X) - 1):  A0[i] = - sigma\_y / 2  A1[i] = 1 + sigma\_y  A2[i] = - sigma\_y / 2  B[i] = f(X[i], Y[j], T[k-1] + tau/2, mu=mu) \* tau / 2 + sigma\_x \* (U\_part[i][j+1] - 2 \* U\_part[i][j] + U\_part[i][j-1]) / 2 + U\_part[i][j]  A0[-1] = 0  A1[-1] = -1   B[-1] = phi\_4(X[-1], T[k], mu=mu) \* 2 \* hy + U\_values[k-1][-2][i]    tmp\_res = solve\_PQ(A0, A1, A2, B)  U\_values[k][j] = tmp\_res   return X, Y, T, U\_values  Демонстрация работы  X, Y, T, UU = changing\_directions(x\_left, x\_right, y\_left, y\_right, a, b, mu, n\_x=20, n\_y=20, t\_end=1) U\_true = real\_U(X, Y, T)  fig = go.Figure(data=[go.Surface(z=UU[len(T) // 2], x=X, y=Y)]) fig.show()  fig = go.Figure(data=[go.Surface(z=U\_true[len(T) // 2], x=X, y=Y)]) fig.show()  fig = go.Figure(data=[go.Surface(z=UU[len(T) // 2], x=X, y=Y, colorscale='Reds', name='Метод переменных направлений'), go.Surface(z=U\_true[len(T) // 2], x=X, y=Y, colorscale='Blues', name='Аналитическое решение')]) fig.update\_layout(title='Метод переменных направлений - красный, Аналитическое решение - синий', showlegend=True) fig.show()  Графики ошибок  N = [10, 15, 20, 30, 40] H\_X, H\_Y, ERROR\_X, ERROR\_Y = get\_error\_array\_with\_h(N, x\_left, x\_right, y\_left, y\_right, a, b, mu, find\_u=changing\_directions, t\_end=1)  h\_error\_plot(H\_X, ERROR\_X) h\_error\_plot(H\_Y, ERROR\_Y, s=' y')  T[1] - T[0] Метод дробных шагов Здесь слой становится "полностью вирутальныйм"  идём по j =  идём по i = по x явно, по y неявно  def partial\_steps(x\_left:float, x\_right:float, y\_left:float, y\_right:float, a:float, b:float, mu:float,   n\_x:float=10, n\_y:float=10, t\_end:float=1) -> (List[float], List[float], List[List[float]], List[List[float]]):  t\_right = t\_end  hx = x\_right / (n\_x - 1)  hy = y\_right / (n\_y - 1)  tau = (hx\*\*2) / 2 / a\*\*2     X = get\_x(x\_left, x\_right, hx)  Y = get\_y(y\_left, y\_right, hy)  T = get\_t(0, t\_right, tau)    U\_values = [[[0] \* len(X) for \_ in range(len(Y))] for \_\_ in range(len(T))]  sigma\_x = (a \* tau) / (hx \*\* 2)  sigma\_y = (b \* tau) / (hy \*\* 2)    initial\_T(U\_values, X, Y, mu)  A0 = [0 for \_ in range(len(X))]  A1 = [0 for \_ in range(len(X))]  A2 = [0 for \_ in range(len(X))]  B = [0 for \_ in range(len(X))]  for k in range(1, len(T)):  U\_part = [[0 for \_ in range(len(Y))] for \_ in range(len(X))]  t\_part = T[k-1] + tau/2   for i in range(1, len(X) - 1):  A1[0] = -1  A2[0] = 0  B[0] = phi\_1(Y[0], t\_part, mu=mu)   for j in range(1, len(Y) - 1):  A0[j] = - sigma\_x  A1[j] = 1 + 2 \* sigma\_x  A2[j] = - sigma\_x  B[j] = f(X[i], Y[j], T[k-1], mu=mu) \* tau / 2 + U\_values[k-1][j][i]  A0[-1] = 0  A1[-1] = -1  A2[-1] = 0  B[-1] = phi\_2(Y[-1], t\_part, mu=mu) \* 2 \* hx + U\_values[k][j][-2] ### тут надо подумать что брать U\_values   tmp\_res = solve\_PQ(A0, A1, A2, B)   U\_part[i] = tmp\_res   t\_part += tau/2  for j in range(1, len(Y) - 1):  A1[0] = -1  A2[0] = 0  B[0] = phi\_3(X[0], t\_part, mu=mu)  for i in range(1, len(X) - 1):  A0[i] = - sigma\_y  A1[i] = 1 + 2 \* sigma\_y  A2[i] = - sigma\_y  B[i] = f(X[i], Y[j], T[k], mu=mu) \* tau / 2 + U\_part[i][j]  A0[-1] = 0  A1[-1] = -1   B[-1] = phi\_4(X[-1], t\_part, mu=mu) \* 2 \* hy + U\_values[k][-2][i]# аналогично предидущему варианту с шраниццами  tmp\_res = solve\_PQ(A0, A1, A2, B) # езультат, заносим в U\_values    U\_values[k][j] = tmp\_res   return X, Y, T, U\_values  Демонстрация работы  XX, YY, TT, UUU = partial\_steps(x\_left, x\_right, y\_left, y\_right, a, b, mu, n\_x=20, n\_y=20, t\_end=1) U\_true2 = real\_U(XX, YY, TT)  fig = go.Figure(data=[go.Surface(z=UUU[len(T) // 2], x=XX, y=YY)]) fig.show()  fig = go.Figure(data=[go.Surface(z=U\_true2[len(T) // 2], x=XX, y=YY)]) fig.show()  fig = go.Figure(data=[go.Surface(z=UUU[len(TT) // 2], x=XX, y=YY, colorscale='Reds', name='Метод дробных шагов'), go.Surface(z=U\_true2[len(TT) // 2], x=XX, y=YY, colorscale='Blues', name='Аналитическое решение')]) fig.update\_layout(title='Метод дробных шагов - красный, Аналитическое решение - синий', showlegend=True) fig.show()  Гафик ошибок  N = [10, 15, 20, 30, 40] H\_X, H\_Y, ERROR\_X, ERROR\_Y = get\_error\_array\_with\_h(N, x\_left, x\_right, y\_left, y\_right, a, b, mu, find\_u=partial\_steps)  h\_error\_plot(H\_X, ERROR\_X) h\_error\_plot(H\_Y, ERROR\_Y, s=' y') |

### **Вывод программы**

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

|  |
| --- |
| Метод переменных направлений, график численного решения и график аналитического решения      Две поверхности на одном графике        График ошибки в зависимости от шага(метод переменных направлений)    Метод дробных шагов, график численного решения и график аналитического решения      Две поверхности на одном графике      График ошибки в зависимости от шага(метод дробных шагов) |

**Заключение:**

В данной лабораторной работе построены конечно-разностные схемы (метод переменных направлений и дробных шагов) для дифференциальных уравнений параболического типа с двумя пространственными и одной временной переменными.

Проведена аппроксимация краевых условий второго рода с первым порядком точности**.**

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

Исходя из графиков зависимости ошибки от длины шага можно сделать вывод, что для поставленной задачи метод переменных направлений оказался точнее метода дробных шагов. Также было показано, что с увеличением шага дробления практически линейно растет погрешность каждого метода.