Московский государственный университет имени М.В. Ломоносова

Физический факультет

Кафедра общей физики и волновых процессов

Отчет по практическому заданию №1

Основы математического моделирования

Выполнил студент 325 группы

Делекторский Никита Юрьевич

Москва

2023

1. Постановка задачи (задача № 10)

Используя схему бегущего счета и итерационные методы, решить задачу:

Сделаем замену переменной: . Тогда и . Перепишем поставленную задачу:

1. Характеристики уравнения

Исследуем существование однозначного решения в данной области определения, т.е. определим, претерпевает ли решение разрыв в области , где – любое. Для начала заметим, что из начального и граничного условия следует, что если , то . Составим уравнение характеристик и узнаем, будут ли характеристики пересекаться в этой области . Если точки пересечения есть, то это означает, что в этих точках существует столько решений, сколько характеристик в них пересекаются. Характеристика на которой :

Проинтегрируем уравнение характеристик:

Учитывая начальные и граничные условия, получаем два семейства характеристик:

1. При и различных :
2. При и различных :

Следует отметить, что при характеристики обоих семейств совпадают и имеют вид . График характеристик представлены на рисунке 1. Из него видно, что на полуинтервале характеристики не пересекаются при любых . Следовательно, в каждой точке области , где – любое, решение у функции существует, и оно единственно.

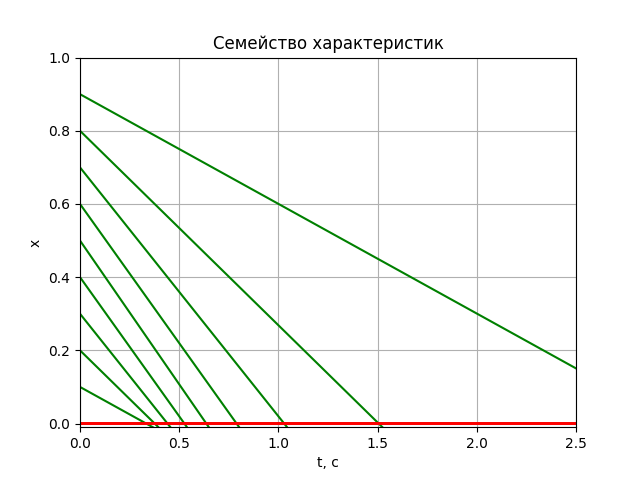


Рис. 1. Семейство характеристик для различных и . Зелёным цветом обозначены прямые при , красным – при

1. Метод решения и разностная аппроксимация

Введём разностную сетку:

где – шаг по времени, – шаг по , – число узлов вдоль оси .

Для начала приведём исходное уравнение к дивергентному виду, выделим производную сложной функции:

Введём функцию . Будем использовать четырехточечный шаблон, поскольку при использовании такого шаблона разностная схема безусловно устойчива, и этот шаблон имеет порядок аппроксимации , что будет показано далее.

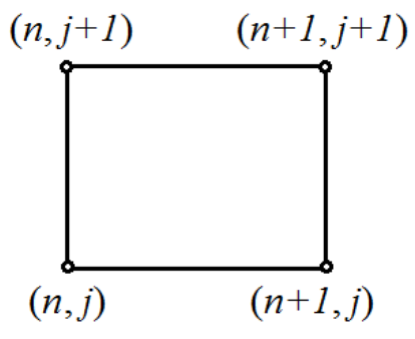
**

Рис. 2. Прямоугольный шаблон схемы бегущего счёта

В таком случае разностная аппроксимация в точке будет выглядеть следующим образом:

где .

Перепишем уравнение в следующем виде:

Будем решать это уравнение итерационным методом Ньютона. Предположим, что известно начальное приближение к корню уравнения.

Отсюда находим:

Получаем итерационную последовательность:

Суть метода Ньютона заключается в итерационной последовательности:

которая продолжается до тех пор, пока не будет достигнута необходимая точность :

1. Численное решение

Ниже представлен код программы реализующий численное решение поставленной задачи. Программа написана на языке Python. Количество шагов по оси координат задано (шаг по оси времени – (шаг ).

**import** numpy **as** np  
**import** matplotlib.pyplot **as** plt  
**from** numpy **import** arctan, sin, pi, log, sqrt  
**from** numba **import** njit  
**from** tqdm **import** tqdm  
**import** time  
  
*# Задаем начальные условия*L = 1 *# Длина сетки*T = 2.5 *# Интервал времени*Nx = 800  
Nt = 4000  
h = L/Nx *# Шаг по пространству*tau = T/Nt *# Шаг по времени*eps = 1e-8  
  
  
**def** draw\_characteristics():  
 **for** x\_0 **in** np.arange(0, 1.1, 0.1):  
 x\_1 = x\_0 + arctan(-sin(pi \* x\_0)) \* T  
 plt.plot(T, x\_1, color=**'green'**, label=**'$t\_0 = 0$'**)  
  
 **for** t\_0 **in** range(5):  
 x\_2 = (T - t\_0) \* arctan(0)  
 plt.plot(T, x\_2, color=**'red'**, label=**'$x\_0 = 0$'**)  
  
 plt.ylim(-0.01, 1)  
 plt.xlim(0, max(T))  
 plt.grid()  
 plt.title(**'Семейство характеристик'**)  
 plt.show()  
  
@njit  
**def** f(x):  
 **return** arctan(x) \* x - log(sqrt(1+x\*x))  
  
  
X = np.linspace(0, 1, Nx) *# Разбивка рассматриваемой области по координате x*T = np.linspace(0, T, Nt) *# Разбивка интервала времени по времени*u\_1 = np.zeros((Nx, Nt)) *# Пустой массив для хранения значений решения*u\_1[:, 0] = sin(pi\*X)  
  
u\_2 = np.zeros((Nx, Nt)) *# Пустой массив для проверки*u\_2[:, 0] = sin(pi\*X)  
  
  
@njit *# Библиотека для ускорения вычислений*  
**def** four\_point(y): *# Реализация четырёхточечной схемы*  
 **for** n **in** range(Nx-1):  
 **for** j **in** range(Nt-1):  
 un1 = y[n + 1][j]  
 un2 = y[n + 1][j] + 2\*eps  
 **while** abs(un1 - un2) > eps:  
 F = (y[n][j + 1] - y[n][j] + un2 - y[n + 1][j]) /2 / tau + (f(un2) - f(y[n][j + 1]) + f(y[n + 1][j]) - f(y[n][j])) / 2 / h  
 dF = 1 / 2 / tau + np.arctan(un2) / 2 / h  
 un1 = un2  
 un2 = un2 - F / dFy[n + 1][j + 1] = un2  
 **return** y  
  
@njit *# Библиотека для ускорения вычислений*  
**def** three\_point(y): *# Реализация трёхточечной схемы*  
 **for** n **in** range(Nx-1):  
 **for** j **in** range(Nt-1):  
 un1 = y[n][j]  
 un2 = y[n][j] + 2 \* eps  
 **while** abs(un1 - un2) > eps:  
 F = (un2 - y[n + 1][j]) / tau + (f(y[n + 1][j]) - f(y[n][j])) / h  
 dF = 1 / 2 / tau + np.arctan(un2) / 2 / h  
 un1 = un2  
 un2 = un2 - F / dF  
 *# print('check', n, j, un2)* y[n + 1][j + 1] = un2  
 **return** y

timer = time.time()  
u\_1 = four\_point(u\_1)  
print(**f'Четырёхточечная схема посчтина за {**time.time()-timer**} секунд'**)  
  
timer = time.time()  
u\_2 = three\_point(u\_2)  
print(**f'Трёхточечная схема посчтина за {**time.time()-timer**} секунд'**)  
  
draw\_characteristics()  
  
plt.pcolormesh(T, -X, u\_1, cmap=**'inferno'**)  
plt.colorbar()  
plt.title(**'Численное решение уравнения переноса\nЧетырёхточечная схема'**)  
plt.xlabel(**'t, с'**)  
plt.ylabel(**'x'**)  
plt.show()  
  
plt.pcolormesh(T, -X, u\_2, cmap=**'inferno'**)  
plt.colorbar()  
plt.title(**'Численное решение уравнения переноса\nТрёхточечная схема'**)  
plt.xlabel(**'t, с'**)  
plt.ylabel(**'x'**)  
plt.show()  
  
print(**f'\nМаксимальная ошибка составила {**np.max(abs(u\_1-u\_2))**}'**)  
print(**f'\nМинимальная ошибка составила {**np.min(abs(u\_1-u\_2))**}'**)  
  
plt.pcolormesh(T, X, u\_1-u\_2, cmap=**'inferno'**, vmin=-0.011, vmax=0.01)  
plt.colorbar()  
plt.title(**'Численное решение уравнения переноса\nРазность четырёхточечной и трёхточечной схем'**)  
plt.xlabel(**'t, с'**)  
plt.ylabel(**'x'**)  
plt.show()  
  
plt.pcolormesh(T, X, (u\_1-u\_2)/np.max(u\_1-u\_2), cmap=**'inferno'**)  
plt.colorbar()  
plt.title(**'Численное решение уравнения переноса\nНормированная разность четырёхточечной и трёхточечной схем'**)  
plt.xlabel(**'t, с'**)  
plt.ylabel(**'x'**)  
plt.show()  
  
x\_grid, t\_grid = np.meshgrid(-X, T)  
  
fig = plt.figure()  
ax\_3d = plt.subplot(projection=**'3d'**)  
ax\_3d.plot\_surface(x\_grid, t\_grid, (u\_1-u\_2).T, rstride=5, cstride=5, cmap=**'plasma'**)  
ax\_3d.set\_xlabel(**'x'**)  
ax\_3d.set\_ylabel(**'t'**)  
ax\_3d.set\_zlabel(**'ΔU'**)  
*# ax\_3d.set\_zlim(0.011, 0.01)*plt.title(**'Численное решение уравнения переноса\nРазность четырёхточечной и трёхточечной схем'**)  
plt.show()  
  
fig = plt.figure()  
ax\_3d = plt.subplot(projection=**'3d'**)  
ax\_3d.plot\_surface(x\_grid, t\_grid, (u\_1-u\_2).T/np.max(abs(u\_1-u\_2)), rstride=5, cstride=5, cmap=**'plasma'**)  
ax\_3d.set\_xlabel(**'x'**)  
ax\_3d.set\_ylabel(**'t'**)  
ax\_3d.set\_zlabel(**'ΔU'**)  
ax\_3d.set\_zlim(0, 1)  
plt.title(**'Численное решение уравнения переноса\nНормированная разность четырёхточечной и трёхточечной схем'**)  
plt.show()  
  
fig = plt.figure()  
ax\_3d = plt.subplot(projection=**'3d'**)  
ax\_3d.plot\_surface(x\_grid, t\_grid, u\_1.T, rstride=5, cstride=5, cmap=**'plasma'**)  
ax\_3d.set\_xlabel(**'x'**)  
ax\_3d.set\_ylabel(**'t'**)  
ax\_3d.set\_zlabel(**'U'**)  
*# ax\_3d.set\_zlim(0, 1)*plt.title(**'Численное решение уравнения переноса\nЧетырёхточечная схема'**)  
plt.show()

1. Результаты

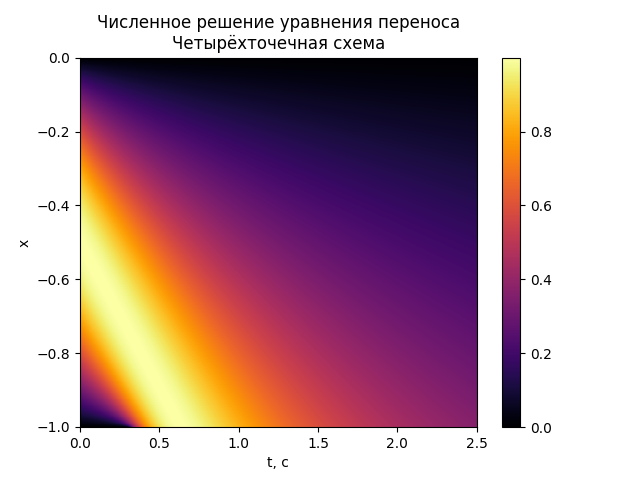


Рис. 3. Результаты численного решения в двумерном виде

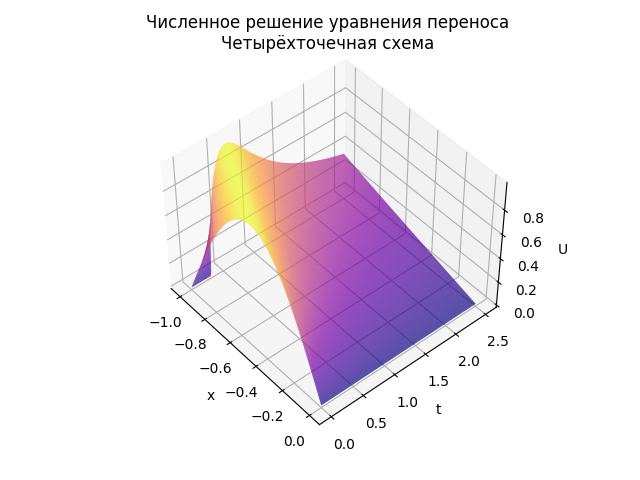


Рис. 4. Результаты численного решения в трёхмерном виде

1. Проверка результатов

Для проверки результатов достаточно убедиться, что другой шаблон даст такой же результат с маленьким отклонением на уровне погрешности. Используем трёхточечный шаблон, представленный на рисунке 5.

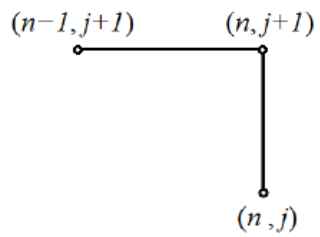


Рис. 5. Трёхточечный шаблон схемы бегущего счёта

Схема строится по аналогии с предыдущей. Сетка и все связанные с ней параметры должны быть такими же. На рисунке 6 представлены результаты численного моделирования

|  |  |
| --- | --- |
|  |  |

Рис. 6. Результаты численного моделирования для двух схем

Для большей наглядности разницы между результатами решений на рисунке 7 приведена их разность в *а)* двумерном и *б)* трёхмерном вариантах. Максимальная ошибка по модулю составила .

Что касается скорости расчёта, к обеим схемам была применена технология принудительной трансляции на язык Си для ускорения расчётов. Четырёхточечная схема была посчитана за 0.68 секунды, тогда как трёхточечная – за 8.21 секунды.

|  |  |
| --- | --- |
| *а)* | |
| *б)* |  |
|  |  |

1. Заключение

В настоящей работе было проведено численное решение начально-краевой задачи для квазилинейного уравнения переноса с использованием схемы бегущего счета и итерационных методов (постановку задачи см. в пункте 1).

Для поиска решения в области и была введена равномерная прямоугольная сетка с шагом по координате и с шагом по времени . Шаблон используемой схемы бегущего счета представлен на Рис. 2. Для решения трансцендентного уравнения относительно искомого значения сеточной функции на каждом шаге использовался итерационный метод Ньютона. Полное описание метода решения задачи представлено в пункте 3.

Программа была написана на языке Python без применения встроенных функций и библиотек, способных упростить решение задачи. Количество шагов по оси координат задано (шаг по оси времени – (шаг ). Полный код программы представлен в пункте 4.

По результатам сравнения решений двумя шаблонами схемам бегущего счёта в пункте 6 было установлено, что среднее отклонение по модулю равно 0.0004, поэтому решение можно считать верным.