Отчет по лабораторной работе №6
по курсу «Численные методы»

Студент группы 8О-406: Киреев А. К.
Работа выполнена: 6.11.2022
Преподаватель: Пивоваров Д.Е.
Отчет сдан:
Итоговая оценка:
Подпись преподавателя:

1. Тема работы

МЕТОД КОНЕЧНЫХ РАЗНОСТЕЙ ДЛЯ РЕШЕНИЯ УРАВНЕНИЙ ГИПЕРБОЛИЧЕСКОГО ТИПА

2. Цель работы

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 U(x, t). Исследовать зависимость погрешности от сеточных параметров tau, h.

(d^2u)/(dt^2) = a * (d^2u)/(dx^2), a > 0,

u_x(0,t) - u(0,t) = 0,

u_x(pi,t) - u(pi, t) = 0,

u(x,0) = sin(x) + cos(x),

u_t(x,0) = -a(sin(x) + cos(x))

Аналитическое решение:
U(x, t) = sin(x - at) + cos(x + at)

3. Ход выполнения работы

Реализована явная и неявная конечно-разностная схемы , реализована двухточечная аппроксимация с первым порядком, трехточечная аппроксимация с вторым порядком, двухточечная аппроксимация с вторым порядком. Также из прошлого семестра были взяты функции для вычисления производных, метод прогонки и логгер. Графики выводятся при помощи библиотеки matplotlib.

Код на Python:

main.py:

```
import numpy as np
import sys
import matplotlib.pyplot as plt
from matplotlib import cm

from methods import explicit_method, implicit_method
from approximator import Approx3p2a, Approx2p2a, Approx2p1a

sys.path.append(".")


def analytical_solution(a: float, x: float, t: float) -> float:
   return np.sin(x - a * t) + np.cos(x + a * t)


def analytical_grid(a: float, x: np.ndarray, t: np.ndarray) -> np.ndarray:
   grid: np.ndarray = np.zeros(shape=(len(t), len(x)))
   for i in range(len(t)):
       for j in range(len(x)):
           grid[i, j] = analytical_solution(a, x[j], t[i])
   return grid


def u_initial(x: np.ndarray) -> np.ndarray:
   return np.sin(x) + np.cos(x)


def u_t_initial(a: float, x: np.ndarray) -> np.ndarray:
   return -a * (np.sin(x) + np.cos(x))


def u_x2_initial(x: np.ndarray) -> np.ndarray:
   return -u_initial(x)


def error(numeric: np.ndarray, analytical: np.ndarray) -> np.ndarray:
   return np.abs(numeric - analytical)


def draw(numerical: np.ndarray, analytical: np.ndarray,
        x: np.ndarray, t: np.ndarray,
        title_lhs: str, title_rhs: str):
   fig = plt.figure(figsize=plt.figaspect(0.7))
   xx, tt = np.meshgrid(x, t)

   ax = fig.add_subplot(1, 2, 1, projection='3d')
   plt.title(title_lhs)
   ax.set_xlabel('x', fontsize=20)
   ax.set_ylabel('t', fontsize=20)
   ax.set_zlabel('u', fontsize=20)
   ax.plot_surface(xx, tt, numerical, cmap=cm.coolwarm, linewidth=0, antialiased=True)

   ax = fig.add_subplot(1, 2, 2, projection='3d')
   ax.set_xlabel('x', fontsize=20)
   ax.set_ylabel('t', fontsize=20)
   ax.set_zlabel('u', fontsize=20)
   plt.title(title_rhs)
   ax.plot_surface(xx, tt, analytical, cmap=cm.coolwarm, linewidth=0, antialiased=True)

   plt.show()


if __name__ == "__main__":
   a = float(input("Enter parameter 'a': "))
   h = float(input("Enter step 'h': "))
   tau = float(input("Enter step 'tau': "))
   t_bound = float(input("Enter time border: "))
   x: np.ndarray = np.arange(0, 3.14 + h/2.0, step=h)
   t: np.ndarray = np.arange(0, t_bound + tau/2.0, step=tau)
```

```
    kwargs = {
        "u_initial": u_initial,
        "u_t_initial": u_t_initial,
        "u_x2_initial": u_x2_initial,
        "a": a,
        "h": h,
        "tau": tau,
        "l": 0.0,
        "r": 3.14,
        "t_bound": t_bound
    }

    analytical = analytical_grid(a, x, t)

    print("---------------- EXPLICIT (2p2a) ----------------")
    approx = Approx2p2a()
    sol = explicit_method(**kwargs, approx=approx)
    print(np.round(sol, 3))
    print("\nError: ", error(sol[-1], analytical[-1]))
    print("-------------------------------------------------\n")
    print("--------------- ANALYTICAL ---------------")
    print(np.round(analytical, 3))

    draw(sol, analytical, x, t, 'explicit', 'analytic')

    print("\n\n---------------- IMPLICIT (2p2a) ----------------")
    approx = Approx2p2a()
    sol = implicit_method(**kwargs, approx=approx)
    print(np.round(sol, 3))
    print("\nError: ", error(sol[-1], analytical[-1]))
    print("-------------------------------------------------\n")
    print("--------------- ANALYTICAL ---------------")
    print(np.round(analytical, 3))

    draw(sol, analytical, x, t, 'implicit', 'analytic')
```

methods.py:

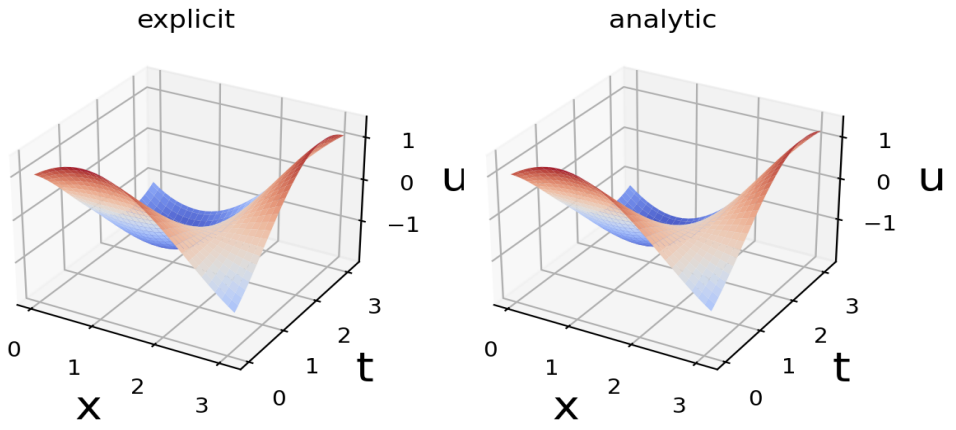
```
import numpy as np
from typing import List, Callable

from logger import base_logger
from sweep import sweep_solve
from derivatives import second_derivative
from approximator import Approx


def explicit_method(u_initial: Callable, u_t_initial: Callable, u_x2_initial: Callable,
                    a: float, h: float, tau: float,
                    l: float, r: float, t_bound: float,
                    approx: Approx) -> np.ndarray:
    sigma: float = a * tau**2 / h**2
    if sigma > 1.0:
        base_logger.warning("WARNING : explicit method is not stable")

    x: np.ndarray = np.arange(l, r + h / 2.0, step=h)
    t: np.ndarray = np.arange(0, t_bound + tau / 2.0, step=tau)
    u: np.ndarray = np.zeros(shape=(len(t), len(x)))

    u[0] = u_initial(x)
    u[1] = u[0] + tau * u_t_initial(a, x) + tau**2 * u_x2_initial(x) / 2.0

    for k in range(1, len(t) - 1):
        u[k+1] = 2.0 * u[k] - u[k-1] + tau**2 * a * second_derivative(u[k], step=h)
        u[k+1, 0] = approx.explicit_0(h, sigma, u, k, tau)
        u[k+1, -1] = approx.explicit_l(h, sigma, u, k, tau)

    return u


def implicit_method(u_initial: Callable, u_t_initial: Callable, u_x2_initial: Callable,
                    a: float, h: float, tau: float,
                    l: float, r: float, t_bound: float,
                    approx: Approx) -> np.ndarray:
    sigma: float = a * tau ** 2 / h ** 2
    x: np.ndarray = np.arange(l, r + h/2.0, step=h)
```

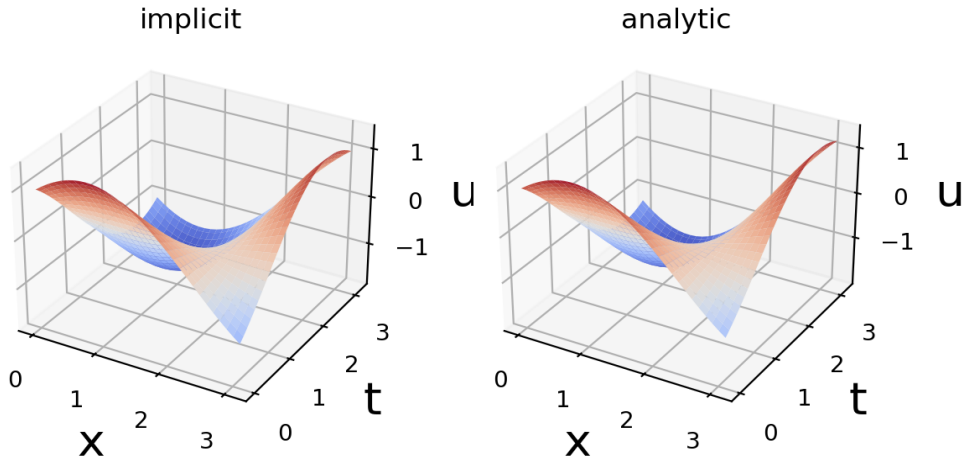
```
    t: np.ndarray = np.arange(0, t_bound + tau/2.0, step=tau)
    u: np.ndarray = np.zeros(shape=(len(t), len(x)))

    u[0] = u_initial(x)
    u[1] = u[0] + tau * u_t_initial(a, x) + tau**2 * u_x2_initial(x) / 2.0

    for k in range(1, len(t) - 1):
        start = approx.implicit_0(h, sigma, u, k)
        end = approx.implicit_1(h, sigma, u, k)
        matrix: np.ndarray = np.zeros(shape=(len(x), len(x)))
        matrix[0] += np.array(
            [start[0], start[1]] + [0.0] * (len(matrix) - 2)
        )
        target: List[float] = [start[2]]

        for i in range(1, len(matrix) - 1):
            matrix[i] += np.array(
                [0.0] * (i - 1)
                + [
                    1.0,
                    -(2.0 + 1.0 / sigma),
                    1.0
                ]
                + [0.0] * (len(matrix) - i - 2)
            )
            target += [(-2.0 * u[k][i] + u[k-1][i]) / sigma]

        matrix[-1] += np.array(
            [0.0] * (len(matrix) - 2) + [end[0], end[1]]
        )
        target += [end[2]]

        u[k+1] = sweep_solve(matrix, np.array(target)).tolist()

    return u
```

4. Результаты

```
(venv) ak@MacBook-Air-AK lab6 % python3 main.py
Enter parameter 'a': 0.9
Enter step 'h': 0.1
Enter step 'tau': 0.05
Enter time border: 3.14
---------------- EXPLICIT (2p2a) ----------------
[[ 1.     1.095  1.179 ... -0.732 -0.849 -0.958]
 [ 0.954  1.044  1.124 ... -0.698 -0.81  -0.913]
 [ 0.905  0.991  1.067 ... -0.662 -0.769 -0.863]
 ...
 [-1.202 -1.316 -1.415 ...  0.918  1.051  1.175]
 [-1.168 -1.278 -1.375 ...  0.89   1.019  1.139]
 [-1.131 -1.238 -1.331 ...  0.86   0.985  1.101]]

Error:  [0.12373938 0.13600182 0.14879763 0.16278707 0.17817959 0.19357474
 0.2062784  0.21451027 0.21992882 0.2250992  0.22903799 0.22877681
 0.22573445 0.22246927 0.21627013 0.20656223 0.1966329  0.18418988
 0.16883331 0.15364007 0.13552085 0.11631166 0.09686257 0.07490099
 0.053871   0.03088242 0.00862691 0.01410611 0.03671422 0.05862885
 0.08056527 0.10106264]
-------------------------------------------------

--------------- ANALYTICAL ---------------
[[ 1.     1.095  1.179 ... -0.732 -0.849 -0.958]
 [ 0.954  1.044  1.125 ... -0.698 -0.81  -0.914]
 [ 0.906  0.992  1.068 ... -0.663 -0.769 -0.868]
 ...
 [-1.309 -1.433 -1.543 ...  0.958  1.111  1.253]
 [-1.283 -1.405 -1.513 ...  0.939  1.089  1.229]
 [-1.255 -1.374 -1.48  ...  0.918  1.065  1.202]]


---------------- IMPLICIT (2p2a) ----------------
[[ 1.     1.095  1.179 ... -0.732 -0.849 -0.958]
 [ 0.954  1.044  1.124 ... -0.698 -0.81  -0.913]
 [ 0.905  0.991  1.067 ... -0.662 -0.768 -0.865]
 ...
 [-1.112 -1.217 -1.309 ...  0.841  0.964  1.077]
```

```
[-1.079 -1.181 -1.271 ...  0.814  0.933  1.043]
[-1.044 -1.143 -1.229 ...  0.785  0.9    1.006]]

Error:  [0.21133465 0.23170265 0.25082405 0.26851214 0.28452804 0.29858884
0.31038542 0.31960936 0.32598366 0.32928953 0.32938189 0.32619089
0.31971257 0.30999517 0.29712741 0.28123141 0.26245928 0.24099129
0.21703406 0.19081833 0.16259648 0.13263977 0.10123522 0.0686822
0.03528897 0.00136917 0.03276144 0.06678795 0.10039921 0.13329056
0.16516676 0.19574482]
--------------------------------------------------

--------------- ANALYTICAL ---------------
[[ 1.     1.095  1.179 ... -0.732 -0.849 -0.958]
[ 0.954  1.044  1.125 ... -0.698 -0.81  -0.914]
[ 0.906  0.992  1.068 ... -0.663 -0.769 -0.868]
...
[-1.309 -1.433 -1.543 ...  0.958  1.111  1.253]
[-1.283 -1.405 -1.513 ...  0.939  1.089  1.229]
[-1.255 -1.374 -1.48  ...  0.918  1.065  1.202]]
```





5. Выводы

В данной лабораторной работе я научился находить численное решение уравнений гиперболического типа при помощи метода конечных разностей, а также аппроксимировать граничные значения разными способами.