Отчет по лабораторной работе №5
по курсу «Численные методы»

Студент группы 8О-306: Киреев А. К.
Работа выполнена: 13.10.2022
Преподаватель: Пивоваров Д.Е.
Отчет сдан:
Итоговая оценка:
Подпись преподавателя:

1. Тема работы

ЧИСЛЕННОЕ РЕШЕНИЕ УРАВНЕНИЙ ПАРАБОЛИЧЕСКОГО ТИПА. ПОНЯТИЕ О МЕТОДЕ КОНЕЧНЫХ РАЗНОСТЕЙ. ОСНОВНЫЕ ОПРЕДЕЛЕНИЯ И КОНЕЧНО-РАЗНОСТНЫЕ СХЕМЫ

2. Цель работы

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 tau, h.

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

u(0,t) = 0,

u(1,t) = 1,

u(x,0) = x + sin(pi * x),

Аналитическое решение:
U(x, t) = x + exp(-pi^2 * a * t)* sin(pi * x)

3. Ход выполнения работы

В лабораторной реализована явная, неявная схема и схема Кранка-Николсона. Также реализована функция подсчета погрешности и функция отрисовки полученного решения. 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 implicit_method, explicit_method, crank_nicolson_method

sys.path.append(".")


def analytical_solution(a: float, x: float, t: float) -> float:
   assert(a > 0.0)
   return x + np.exp(-np.pi**2 * a * t) * np.sin(np.pi * x)


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 x + np.sin(np.pi * x)


def u_left_border():
   return 0.0


def u_right_border():
   return 1.0


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):
   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('numerical')
   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('analytic')
   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, 1.0 + h/2.0, step=h)
   t: np.ndarray = np.arange(0, t_bound + tau/2.0, step=tau)

   kwargs = {
       "u_initial": u_initial,
       "u_left_border": u_left_border,
```

```
        "u_right_border": u_right_border,
        "a": a,
        "h": h,
        "tau": tau,
        "l": 0.0,
        "r": 1.0,
        "t_bound": t_bound
    }

    analytical = analytical_grid(a, x, t)

    print("---------------- EXPLICIT ----------------")
    sol = explicit_method(**kwargs)
    print(np.round(sol, 3))
    print("\nError: ", error(sol[-1], analytical[-1]))
    print("-----------------------------------------\n")
    print("---------------- IMPLICIT ----------------")
    sol = implicit_method(**kwargs)
    print(np.round(sol, 3))
    print("\nError: ", error(sol[-1], analytical[-1]))
    print("-----------------------------------------\n")
    print("------------- CRANK-NICOLSON -------------")
    sol = crank_nicolson_method(**kwargs)
    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)
```

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


def explicit_method(u_initial: Callable, u_left_border: Callable, u_right_border: Callable,
                    a: float, h: float, tau: float,
                    l: float, r: float, t_bound: float) -> np.ndarray:
    if a * tau / h**2 > 0.5:
        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[0, 0] = u_left_border()
    u[0, -1] = u_right_border()

    for k in range(len(t) - 1):
        u[k+1] = u[k] + tau * a * second_derivative(u[k], step=h)

    return u


def hybrid_method(u_initial: Callable, u_left_border: Callable, u_right_border: Callable,
                  a: float, h: float, tau: float,
                  l: float, r: float, t_bound: float, theta: float) -> np.ndarray:
    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[0, 0] = u_left_border()
    u[0, -1] = u_right_border()

    for k in range(len(t) - 1):
        matrix: np.ndarray = np.zeros(shape=(len(x) - 2, len(x) - 2))
        matrix[0] += np.array(
            [
```

```python
                -(1.0 + (2.0 * theta * a * tau) / h**2),
                (theta * a * tau) / h**2
            ]
            + [0.0] * (len(matrix) - 2)
        )
        target: List[float] = [(theta - 1.0) * a * tau * u[k][0] / h**2 +
                               (2.0 * (1.0 - theta) * a * tau / h**2 - 1.0) * u[k][1] +
                               (theta - 1.0) * a * tau * u[k][2] / h**2 -
                               theta * a * tau * u_left_border() / h**2]

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

        matrix[-1] += np.array(
            [0.0] * (len(matrix) - 2)
            + [
                theta * a * tau / h ** 2,
                -(1.0 + (2.0 * theta * a * tau) / h ** 2)
            ]
        )
        target += [(theta - 1.0) * a * tau * u[k][-3] / h**2 +
                   (2.0 * (1.0 - theta) * a * tau / h**2 - 1.0) * u[k][-2] +
                   (theta - 1.0) * a * tau * u[k][-1] / h**2 -
                   theta * a * tau * u_right_border() / h**2]

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

    return u


def implicit_method(**kwargs) -> np.ndarray:
    return hybrid_method(**kwargs, theta=1.0)


def crank_nicolson_method(**kwargs) -> np.ndarray:
    return hybrid_method(**kwargs, theta=0.5)
```

4. Результаты

```
(venv) ak@MacBook-Air-AK lab5 % python3 main.py
Enter parameter 'a': 1.0
Enter step 'h': 0.2
Enter step 'tau': 0.01
Enter time border: 0.1
---------------- EXPLICIT ----------------
[[0.    0.788 1.351 1.551 1.388 1.   ]
 [0.    0.732 1.26  1.46  1.332 1.   ]
 [0.    0.681 1.178 1.378 1.281 1.   ]
 [0.    0.635 1.104 1.304 1.235 1.   ]
 [0.    0.593 1.037 1.237 1.193 1.   ]
 [0.    0.556 0.976 1.176 1.156 1.   ]
 [0.    0.522 0.921 1.121 1.122 1.   ]
 [0.    0.491 0.871 1.071 1.091 1.   ]
 [0.    0.463 0.826 1.026 1.063 1.   ]
 [0.    0.438 0.785 0.985 1.038 1.   ]
 [0.    0.415 0.749 0.949 1.015 1.   ]]

Error:  [0.         0.00362282 0.00586184 0.00586184 0.00362282 0.        ]
------------------------------------------

---------------- IMPLICIT ----------------
```

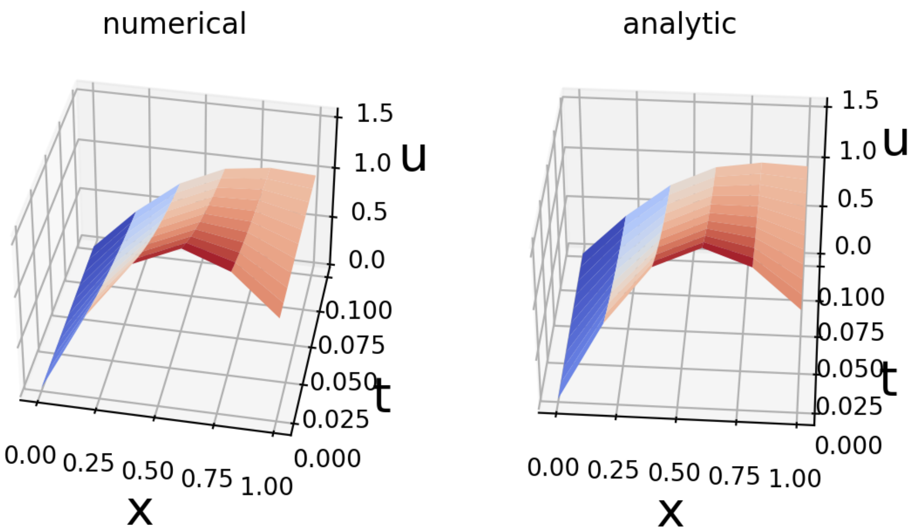
```
[[0.    0.788 1.351 1.551 1.388 1.   ]
 [0.    0.737 1.268 1.468 1.337 1.   ]
 [0.    0.69  1.192 1.392 1.29  1.   ]
 [0.    0.647 1.123 1.323 1.247 1.   ]
 [0.    0.608 1.06  1.26  1.208 1.   ]
 [0.    0.573 1.003 1.203 1.173 1.   ]
 [0.    0.54  0.95  1.15  1.14  1.   ]
 [0.    0.51  0.902 1.102 1.11  1.   ]
 [0.    0.483 0.858 1.058 1.083 1.   ]
 [0.    0.459 0.819 1.019 1.059 1.   ]
 [0.    0.436 0.782 0.982 1.036 1.   ]]

Error:  [0.         0.01704553 0.02758025 0.02758025 0.01704553 0.        ]
------------------------------------------

------------- CRANK-NICOLSON -------------
[[0.    0.788 1.351 1.551 1.388 1.   ]
 [0.    0.734 1.264 1.464 1.334 1.   ]
 [0.    0.686 1.186 1.386 1.286 1.   ]
 [0.    0.641 1.114 1.314 1.241 1.   ]
 [0.    0.601 1.049 1.249 1.201 1.   ]
 [0.    0.565 0.99  1.19  1.165 1.   ]
 [0.    0.531 0.936 1.136 1.131 1.   ]
 [0.    0.501 0.887 1.087 1.101 1.   ]
 [0.    0.474 0.843 1.043 1.074 1.   ]
 [0.    0.449 0.802 1.002 1.049 1.   ]
 [0.    0.426 0.766 0.966 1.026 1.   ]]

Error:  [0.         0.00696965 0.01127712 0.01127712 0.00696965 0.        ]
------------------------------------------

--------------- ANALYTICAL ---------------
[[0.    0.788 1.351 1.551 1.388 1.   ]
 [0.    0.733 1.262 1.462 1.333 1.   ]
 [0.    0.682 1.181 1.381 1.282 1.   ]
 [0.    0.637 1.107 1.307 1.237 1.   ]
 [0.    0.596 1.041 1.241 1.196 1.   ]
 [0.    0.559 0.981 1.181 1.159 1.   ]
 [0.    0.525 0.926 1.126 1.125 1.   ]
 [0.    0.495 0.877 1.077 1.095 1.   ]
 [0.    0.467 0.832 1.032 1.067 1.   ]
 [0.    0.442 0.791 0.991 1.042 1.   ]
 [0.    0.419 0.754 0.954 1.019 1.   ]]
```



5. Выводы

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