Московский авиационный институт

(Национальный исследовательский университет)

Факультет прикладной математики и физики

Кафедра вычислительной математики и программирования

Отчет по лабораторным работам

по курсу «Численные методы»

на тему

«Численные методы решения ДУЧП параболического типа»

Выполнил: Изосимов Н.А.

Группа: М8О-409Б-19

Проверил: Пивоваров Д.Е.

Дата:

Оценка:

Москва, 2022

ЛАБОРАТОРНАЯ РАБОТА №3. РЕШЕНИЕ ДИФФЕРЕНЦИАЛЬНЫХ УРАВНЕНИЙ ЭЛЛИЧЕСКОГО ТИПА

Задание

Решить краевую задачу для дифференциального уравнения эллиптического типа. Аппроксимацию уравнения произвести с использованием центрально-разностной схемы. Для решения дискретного аналога применить следующие методы: метод простых итераций (метод Либмана), метод Зейделя, метод простых итераций с верхней релаксацией. Вычислить погрешность численного решения путем сравнения результатов с приведенным в задании аналитическим решением $U(x, y)$. Исследовать зависимость погрешности от сеточных параметров h_x, h_y.

Вариант 5

Уравнение:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = -u,$$

Граничные условия:

$$u_x(0, y) = cos(y),$$
$$u_x(1, y) - u(1, y) = 0,$$
$$u(x, 0) = x,$$
$$u(x, \frac{\pi}{2}) = 0$$

Аналитическое решение:

$$U(x, t) = xcos(y)$$

Код программы

```
import numpy as np
import scipy.interpolate
from copy import deepcopy
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = [30, 10]
# границы
lx = 1
ly = np.pi / 2
# точность итерационного процесса
eps = 1e-05
# разбиение
```

```
Nx = 15
Ny = 15
# параметр релаксации
teta = 1.5
# шаги
hx = lx/Nx
hy = ly/Ny

def phi1(y):
    return np.cos(y)
def phi2():
    return 0
def phi3(x):
    return x
def phi4(x):
    return 0
def analytic_solution(x, y):
    return x * np.cos(y)

def norm(v1, v2):
    return np.amax(np.abs(v1 - v2))

def find_error(grid):
    u_correct  =  np.array([[analytic_solution(i*hx,j  *hy)  for  j  in
range(Ny+1)] for i in range(Nx+1)])
    return norm(u_correct, grid)

def view_mode(grid):

    y_array = np.array([i*hy for i in range(Ny+1)])

    fig, ax = plt.subplots()

    x = [int(Nx * 0.05), int(Nx * 0.1), int(Nx * 0.25)]
    colors = ['green', 'blue', 'red']

    for i in range(len(x)):
        u_correct = analytic_solution(x[i]*hx, y_array)
        u_calculated = grid[x[i]]
        ax.plot(y_array, u_correct, color='black')
        ax.plot(y_array, u_calculated, color=colors[i], linestyle = '--
', label='x=%s'%round(x[i]*hx, 2))

    ax.set_xlabel('y')
    ax.set_ylabel('U(x, y)')
    ax.grid()
```

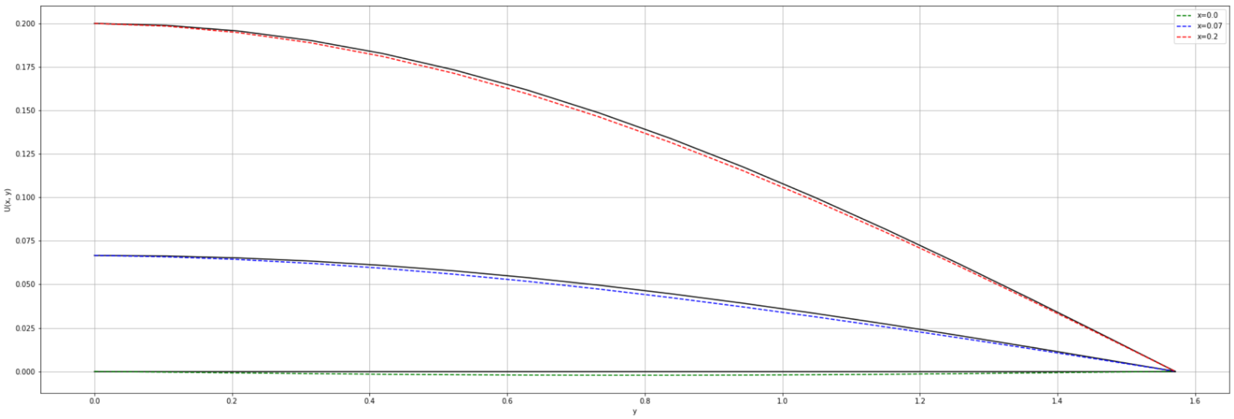
```
    ax.legend()

def Liebman_method():
    grid = np.zeros((Nx+1, Ny+1))

    # верхний и нижний слой
    for i in range(Nx + 1):
        grid[i][0] = phi3(i * hx)
        grid[i][Ny] = phi4(i * hx)

    # Итерационный процесс
    k = 0
    while True:
        grid_next = deepcopy(grid)
        for i in range(1, Nx):
            for j in range(1, Ny):
                grid_next[i][j]    =    1/(2/hx**2+2/hy**2-1)*((grid[i-
1][j]+grid[i+1][j])/hx**2+(grid[i][j-1]+grid[i][j+1])/hy**2)

        # Аппроксимация граничных условий 2-го и 3-го рода
        for j in range(Ny):
            grid_next[0][j] = grid_next[1][j]-hx*phi1(j*hy)
            grid_next[Nx][j] = (grid_next[Nx-1][j]+hx*phi2())/(1-hx)

        # Условие прекращения цикла
        if norm(grid, grid_next) < eps:
            break

        grid = deepcopy(grid_next)
        k += 1

    return grid, k

def Seidel_method():
    grid = np.zeros((Nx+1, Ny+1))

    # Заполняем граничные условия 1-го рода
    for i in range(Nx + 1):
        grid[i][0] = phi3(i * hx)
        grid[i][Ny] = phi4(i * hx)

    # Итерационный процесс
    k = 0
    while True:
        grid_prev = deepcopy(grid)
        for i in range(1, Nx):
```

```
            for j in range(1, Ny):
                grid[i][j]   =   1/(2/hx**2+2/hy**2   -   1)*((grid[i-
1][j]+grid[i+1][j])/hx**2+(grid[i][j-1]+grid[i][j+1])/hy**2)

        # Аппроксимация граничных условий 2-го и 3-го рода
        for j in range(Ny):
            grid[0][j] = grid[1][j]-hx*phi1(j*hy)
            grid[Nx][j] = (grid[Nx-1][j]+hx*phi2())/(1-hx)

        # Условие прекращения цикла
        if norm(grid, grid_prev) < eps:
            break

        k += 1

    return grid, k
def Relax_method():
    grid = np.zeros((Nx+1, Ny+1))

    # Заполняем граничные условия 1-го рода
    for i in range(Nx + 1):
        grid[i][0] = phi3(i * hx)
        grid[i][Ny] = phi4(i * hx)

    # Итерационный процесс Зейделя
    k = 0
    while True:
        grid_prev = deepcopy(grid)
        for i in range(1, Nx):
            for j in range(1, Ny):
                grid[i][j]   =   1/(2/hx**2+2/hy**2   -   1)*((grid[i-
1][j]+grid[i+1][j])/hx**2+(grid[i][j-1]+grid[i][j+1])/hy**2)

        # Аппроксимация граничных условий 2-го и 3-го рода
        for j in range(Ny):
            grid[0][j] = grid[1][j]-hx*phi1(j*hy)
            grid[Nx][j] = (grid[Nx-1][j]+hx*phi2())/(1-hx)

        # Релаксация
        grid = teta*grid+(1-teta)*grid_prev

        # Условие прекращения цикла
        if norm(grid, grid_prev) < eps:
            break

        k += 1
```

```
    return grid, k
```

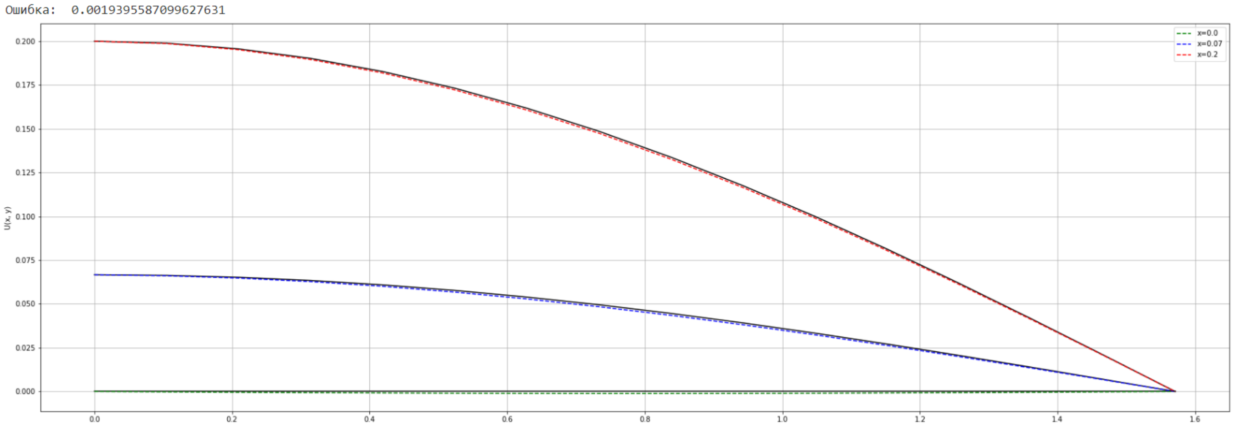
Результат работы программы

```
Кол-во итераций:  2231
Ошибка:  0.004042668824270512
```

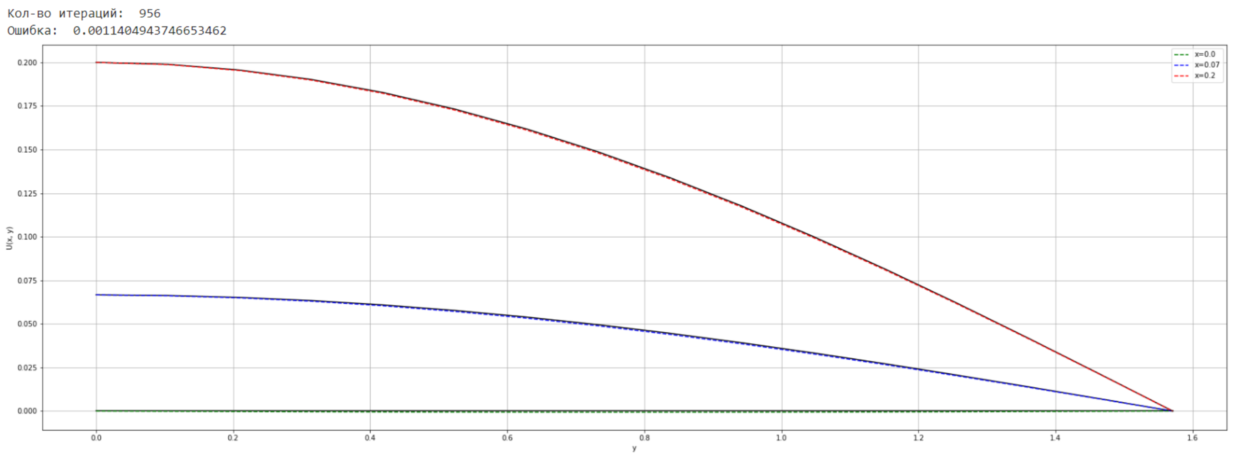


Метод Либмана

```
Кол-во итераций:  1338
Ошибка:  0.0019395587099627631
```



Метод Зейделя



Метод простых итераций с верхней релаксацией

Вывод

Из результатов выполненной лабораторной работы можно заключить, что все рассмотренные методы примерно в равной степени точны, однако количество итераций, за которое эта точность достигается, у каждого из них отличается. Самым быстрым оказался метод простых итераций с верхней релаксацией, за ним следует метод Зейделя, а самым медленным оказался метод простых итераций. Также из графика погрешностей можно заметить четкую тенденцию – при увеличении длины шага увеличивается и погрешность используемого метода.