|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **Министерство науки и высшего образования Российской Федерации**  Калужский филиал  федерального государственного бюджетного  образовательного учреждения высшего образования  ***«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»***  ***(КФ МГТУ им. Н.Э. Баумана)*** |

**ФАКУЛЬТЕТ** ***ИУК «Информатика и управление»***

**КАФЕДРА** \_\_***ИУК4 «Программное обеспечение ЭВМ, информационные технологии»***

**ЛАБОРАТОРНАЯ РАБОТА №5**

**«Применение базовых методов решения ДУЧП2 эллиптического типа»**

**ДИСЦИПЛИНА: «Моделирование»**

|  |  |  |
| --- | --- | --- |
| Выполнил: студент гр. ИУК4-62Б | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ( Карельский М.К. )  (Подпись) |
| Проверил: | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ( Никитенко У.В. )  (Подпись) |
| Дата сдачи (защиты):  Результаты сдачи (защиты): | | |
|  | - Балльная оценка:  - Оценка: | |

Калуга, 2023

**Цель:** сформировать практические навыки анализа возможностей построения и выделения наиболее важных свойств объектов моделей для моделирования и использования специализированных программных пакетов и библиотек для стандартных вычислений и визуализации результатов численного или приближенно-аналитического решения ДУЧП2 эллиптического типа на основе сравнения результатов.

**Задачи:** решить уравнение, указанное в варианте численными методами и оценить точность аппроксимации. Оценить устойчивость и сходимость. Выбрать среду для проведения расчетов и вычислительного эксперимента. Написать программу, реализующую решение разностной задачи. Оценить результаты расчетов. Визуализировать результаты, сравнить результаты, выдвинуть и обосновать гипотезу целесообразности использования того или иного метода в зависимости от предложенной задачи и ее вариаций, точности результата, трудоемкости, сложности алгоритма, сложности обоснования применимости метода, вычислительной эффективности алгоритма.

**Задание:**

Рассматривается задача Дирихле для эллиптического уравнения

Пусть – прямоугольник, а

Здесь – достаточное гладкие функции такие, что , где – постоянные

**Вариант 12**

Найти решение задачи

Итерационным методом с чебышевским набором параметров.

В качестве критерия конца вычислений использовать условие:

Отладить решение задачи на функции

**Решение:**

Исходя из условия имеем:

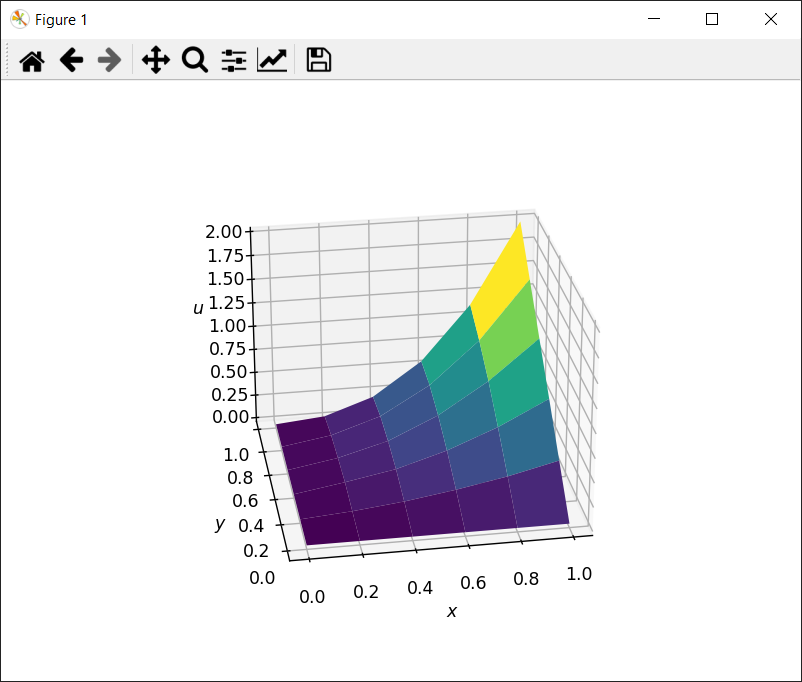
Согласно методу с чебышевским набором параметров:

В формуле присутствуют следующие элементы:

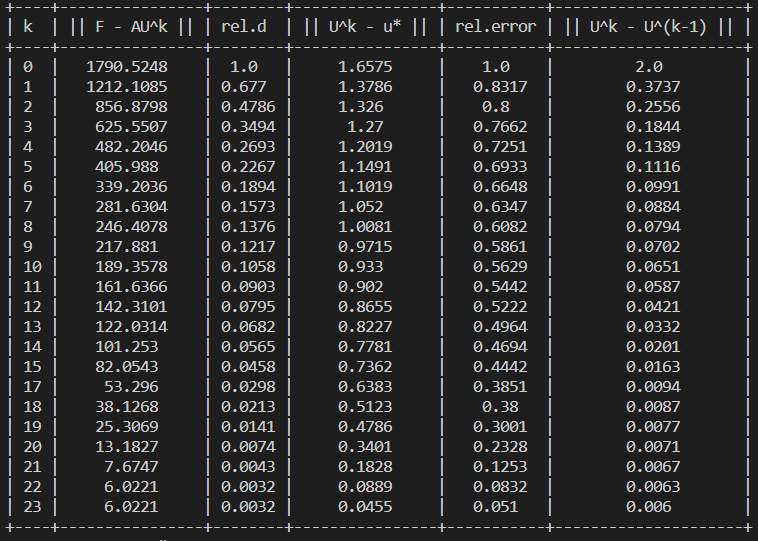
Граничные условия:

Выбор точности при :

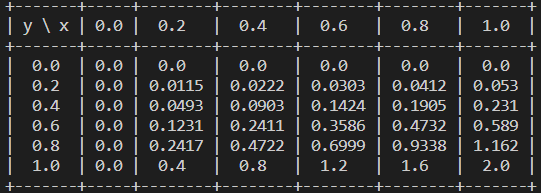
Полученное решение:



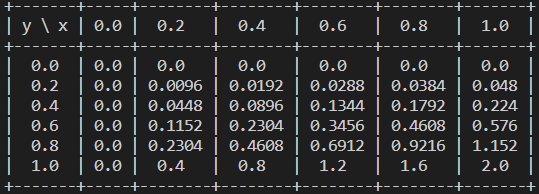
**Рис. 1.** График функции



**Рис. 2.** Характеристики



**Рис. 3.** Решение на крупной сетке



**Рис. 4.** Точное решение на крупной сетке

**Вывод:** в ходе выполнения лабораторной работы были получены практические навыки анализа возможностей построения и выделения наиболее важных свойств объектов моделей для моделирования и использования специализированных программных пакетов и библиотек для стандартных вычислений и визуализации результатов численного или приближенно-аналитического решения ДУЧП2 эллиптического типа на основе сравнения результатов.

**ПРИЛОЖЕНИЯ**

**Листинг:**

import numpy as np

import matplotlib.pyplot as plt

from prettytable import PrettyTable

f\_p = lambda x = None, y = None: 1 + x/2

f\_q = lambda x = None, y = None: 1

f\_f = lambda x, y: -6\*x\*y - 2\*x - y\*\*2\*(y + 1)/2

f\_mu = lambda x, y: x\*y\*\*2\*(1 + y)

l\_x = 1

l\_y = 1

c\_1 = 1

c\_2 = 1.5

d\_1 = 1

d\_2 = 1

f\_u\_exact = lambda x, y: x\*y\*\*2\*(1 + y)

l\_u = lambda x, y, u, i, j, h\_x, h\_y: \

    f\_p(x + h\_x/2, y) \* (u[i + 1, j] - u[i, j]) / pow(h\_x, 2) \

    - f\_p(x - h\_x/2, y) \* (u[i, j] - u[i - 1, j]) / pow(h\_x, 2) \

    + f\_q(x, y + h\_y/2) \* (u[i, j + 1] - u[i, j]) / pow(h\_y, 2) \

    - f\_q(x, y - h\_y/2) \* (u[i, j] - u[i, j - 1]) / pow(h\_y, 2)

M = 20

N = M

u\_0 = np.zeros((N, M))

u\_0\_diff = 0

u = np.zeros((N, M))

xs = np.linspace(0, l\_x, N)

ys = np.linspace(0, l\_y, M)

print(ys[-1])

print(xs[-1])

h\_x = l\_x/N

h\_y = l\_y/M

delta = 4/pow(h\_x, 2) \* pow(np.sin(np.pi\*h\_x/2/np.pi), 2) \

    +  8/pow(h\_y, 2) \* pow(np.sin(np.pi\*h\_y/2/np.pi), 2)

Delta = 4/pow(h\_x, 2) \* pow(np.cos(np.pi\*h\_x/2/np.pi), 2) \

    +  8/pow(h\_y, 2) \* pow(np.cos(np.pi\*h\_y/2/np.pi), 2)

n = 23

def FU(x, y, h\_x, h\_y, last\_u, i, j, k):

    tau\_k = 2/(Delta + delta + (Delta - delta)\*np.cos((2\*k-1)\*np.pi/(2\*n)))

    p\_plus = f\_p(x + h\_x/2, y)

    p\_minus = f\_p(x - h\_x/2, y)

    q\_plus = f\_q(x, y + h\_y/2)

    q\_minus = f\_q(x, y - h\_y/2)

    return last\_u[i, j] + tau\_k\*(p\_plus\*(last\_u[i + 1, j] - last\_u[i, j])/h\_x\*\*2 - \

                                 p\_minus\*(last\_u[i, j] - last\_u[i - 1, j])/h\_x\*\*2 + \

                                 q\_plus\*(last\_u[i, j + 1] - last\_u[i, j])/h\_y\*\*2 - \

                                 q\_minus\*(last\_u[i, j] - last\_u[i, j - 1])/h\_y\*\*2 + \

                                    f\_f(x, y))

lambda\_max = 1 - pow(h\_x, 2) / 4 \* delta

lambda\_min = 1 - pow(h\_x, 2) / 4 \* Delta

k\_list = []

exact\_diff = []

last\_diff = []

rel\_d = []

rel\_error = []

discrepancies = []

u\_exact = np.array([[f\_u\_exact(x, y) for x in xs] for y in ys])

k = 0

eps = 5e-2

k\_max = np.log(1/eps) / 4 / eps

LU = np.zeros((N, M))

F = np.zeros((N, M))

for i in range(1, N-1):

    for j in range(1, M-1):

        LU[i, j] = l\_u(xs[i], ys[j], u\_exact, i, j, h\_x, h\_y)

        F[i, j] = f\_f(xs[i], ys[j])

print(f'|| F-Au^\* || = {np.amax(np.abs(LU + F))}')

u = np.zeros((N, M))

LU = np.zeros((N, M))

F = np.zeros((N, M))

last\_u = np.copy(u)

last\_last\_u = np.copy(u)

u[:, 0] = f\_mu(xs, 0)

u[:, -1] = f\_mu(xs, l\_x)

u[0, :] = f\_mu(0, ys)

u[-1, :] = f\_mu(l\_y, ys)

for i in range(1, N-1):

    for j in range(1, M-1):

        u[i, j] = FU(xs[i], ys[j], h\_x, h\_y, last\_u, i, j, 0)

for i in range(1, N-1):

    for j in range(1, M-1):

        LU[i, j] = l\_u(xs[i], ys[j], u, i, j, h\_x, h\_y)

        F[i, j] = f\_f(xs[i], ys[j])

discrepancy\_0 = np.amax(np.abs(LU + F))

print(f'|| F-AU^0 || = {discrepancy\_0}')

u = np.zeros((N, M))

while len(exact\_diff) == 0 or exact\_diff[-1] > eps:

    last\_last\_u = np.copy(last\_u)

    last\_u = np.copy(u)

    u[:, 0] = f\_mu(xs, 0)

    u[:, -1] = f\_mu(xs, l\_x)

    u[0, :] = f\_mu(0, ys)

    u[-1, :] = f\_mu(l\_y, ys)

    for i in range(1, N-1):

        for j in range(1, M-1):

            u[i, j] = FU(xs[i], ys[j], h\_x, h\_y, last\_u, i, j, k)

    LU = np.zeros((N, M))

    F = np.zeros((N, M))

    for i in range(1, N-1):

        for j in range(1, M-1):

            LU[i, j] = l\_u(xs[i], ys[j], u, i, j, h\_x, h\_y)

            F[i, j] = f\_f(xs[i], ys[j])

    if k == 0:

        u\_0 = np.copy(u)

        u\_0\_diff = np.amax(np.abs(u\_0 - u\_exact))

    k\_list.append(k)

    exact\_diff.append(np.amax(np.abs(u - u\_exact)))

    last\_diff.append(np.amax(np.abs(u - last\_u)))

    rel\_error.append(np.amax(np.abs(u - u\_exact))/u\_0\_diff)

    discrepancies.append(np.amax(np.abs(LU + F)))

    rel\_d.append(discrepancies[-1] / discrepancy\_0)

    k += 1

    if k >= n: break

table = PrettyTable()

table.add\_column("k", np.array(k\_list).round(4))

table.add\_column("|| F - AU^k ||", np.array(discrepancies).round(4))

table.add\_column("rel.d", np.array(rel\_d).round(4))

table.add\_column("|| U^k - u\* ||", np.array(exact\_diff).round(4))

table.add\_column("rel.error", np.array(rel\_error).round(4))

table.add\_column("|| U^k - U^(k-1) ||", np.array(last\_diff).round(4))

print(table)

xs = np.linspace(0, l\_x, 6)

ys = np.linspace(0, l\_y, 6)

h\_x = (xs[1] - xs[0]) / 2

h\_y = (ys[1] - ys[0]) / 2

U = np.zeros((6, 6))

for k in range(130):

    last\_u = np.copy(U)

    U[:, 0] = f\_mu(xs, 0)

    U[:, -1] = f\_mu(xs, l\_x)

    U[0, :] = f\_mu(0, ys)

    U[-1, :] = f\_mu(l\_y, ys)

    for i in range(1, 5):

        for j in range(1, 5):

            U[i, j] = FU(xs[i], ys[j], h\_x, h\_y, last\_u, i, j, k)

fig = plt.figure()

ax = plt.axes(projection='3d')

X, Y = np.meshgrid(xs, ys)

ax.plot\_surface(X, Y, U, rstride=1, cstride=1,

    cmap='viridis', edgecolor='none')

ax.set\_xlabel('$x$')

ax.set\_ylabel('$y$')

ax.set\_zlabel('$u$')

plt.show()

table = PrettyTable()

xs = xs.round(5)

ys = ys.round(5)

U = U.round(5)

table.add\_column("y \ x", ys)

for k in range(len(xs)):

    table.add\_column(f"{xs[k]}", U[:, k])

print(table)

u\_exact = np.array([[f\_u\_exact(x, y) for x in xs] for y in ys])

table = PrettyTable()

xs = xs.round(5)

ys = ys.round(5)

u\_exact = u\_exact.round(5)

table.add\_column("y \ x", ys)

for k in range(len(xs)):

    table.add\_column(f"{xs[k]}", u\_exact[:, k])

print(table)