**Министерство науки и высшего образования Российской Федерации**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ**

**ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ «мэи»**

**Институт информационных и вычислительных технологий**

**Кафедра математического и компьютерного моделирования**

КУРСОВАЯ РАБОТА

**по дисциплине «Численные методы»**

Тема: Начально-краевая задача для двумерного уравнения колебаний

|  |  |  |
| --- | --- | --- |
| Студенты  гр. А-18-21 |  | Липкин Г.М. |
|  | (подпись) | Перьков А.М. |
| Руководитель | (подпись) | Амосова О.А. |
|  | (подпись) |  |

Москва, 2023

**ЗАДАНИЕ**

Первая краевая задача для уравнения с постоянными коэффициентами. Схема с весами. Моделирование нестационарных процессов в зависимости от правой части уравнения.

*δ²u / δt² = Δu + f(x, y, t), 0 < x < a, 0 < y < b, t > 0*

*u(x, y, 0) = φ(x, y), δu/δt = ψ(x, y), 0 < x < a, 0 < y < b*

*u|Г = 0, t ≥ 0*

**содержание**

|  |  |  |
| --- | --- | --- |
|  | Введение ………………………………………………………….. | 4 |
| 1. | Дискретизация задачи |  |
| 1.1. | Составление разностной схемы …………………………………………….. | 5 |
| 1.2. | Подготовка тестовых примеров …………………………………………….. | 6 |
| 2. | Решение тестового примера 1 |  |
| 2.1. | Результат решения ………………………………………………………….. | 8 |
| 2.2. | Расчёт погрешностей ……………………………………………………….. | 10 |
| 3. | Решение тестового примера 2 |  |
| 3.1. | Результат решения ………………………………………………………….. | 11 |
| 3.2. | Расчёт погрешностей ……………………………………………………….. | 13 |
| 4. | Заключение ……………………………………………………….. | 14 |
| 5. | Приложение ………………………………………………………. | 15 |

**введение**

Задача заключается в численном решении уравнения и создания 3д отображения получившейся функции в зависимости от времени. Также необходимо оценить абсолютную погрешность и погрешность по Рунге полученного решения.

Так как аналитическое решение задачи выходит за рамки курса, задача вычислялась только численно на тестовых примерах.

Для получения решения составим разностную схему первого порядка по методу разностных производных.

**1. ДИСКРЕТЕЗАЦИЯ ЗАДАЧИ**

**1.1. Составление разностной схемы**

Пусть [0, a], [0, b], [0, T), где i = 0,1,…,n, j = 0,1,…,n, k = 0,1,…,m.

*Начальное условие:*

*Разностная схема:*

**1.2. Подготовка тестовых примера**

***Тестовый пример 1:***

Найдём значение правой части

Подставим граничные условия, чтобы найти 2 первых слоя

Полученный тестовый пример:

***Тестовый пример 2:***

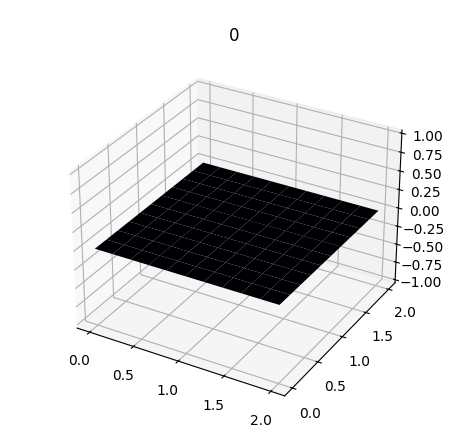
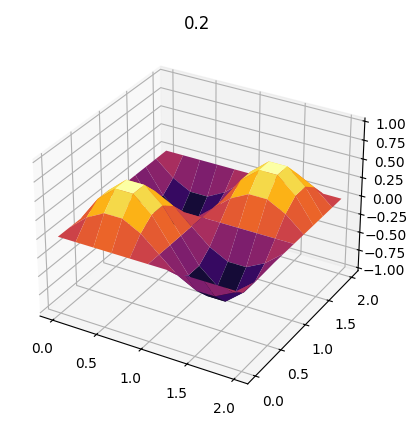
Найдём значение правой части

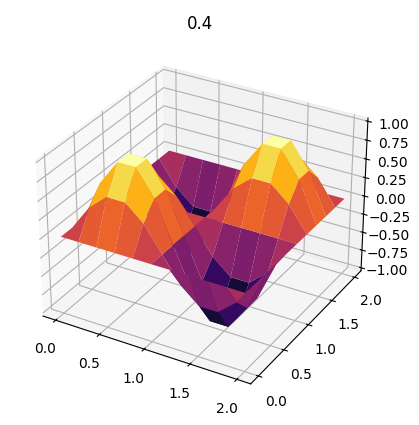
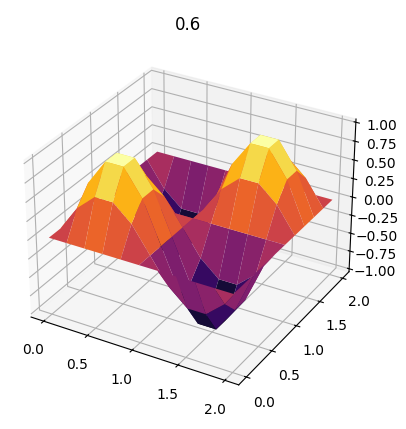
Подставим граничные условия, чтобы найти 2 первых слоя

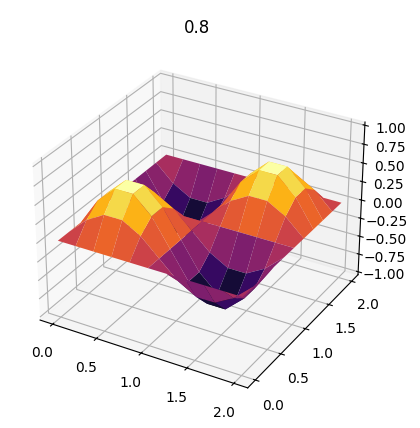
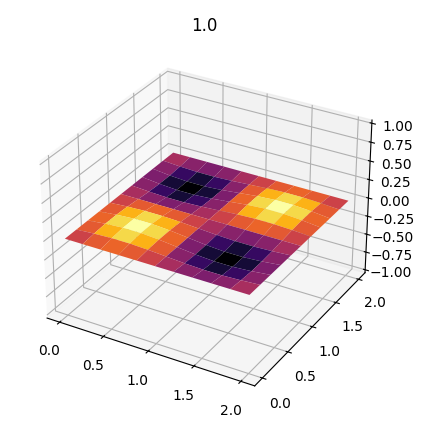
Полученный тестовый пример:

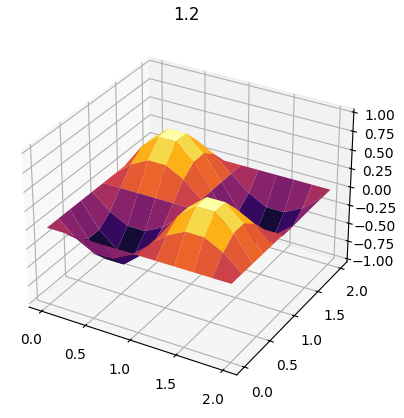
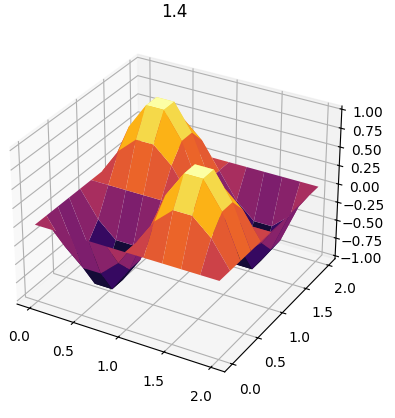
**2. РЕШЕНИЕ ТЕСТОВОГО ПРИМЕРА 1**

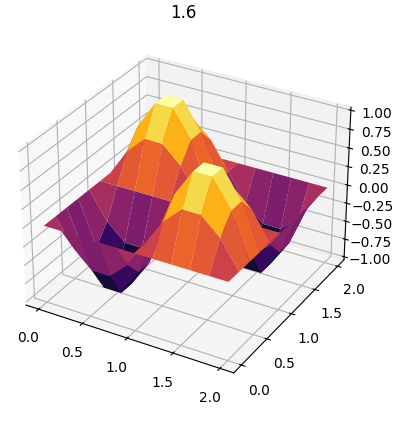
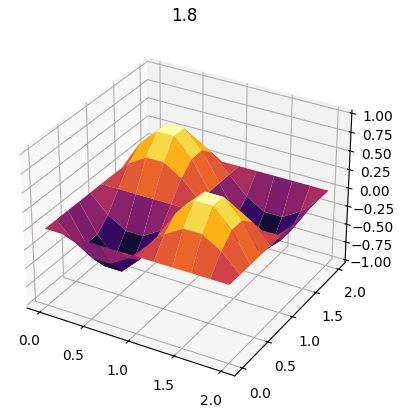
**2.1. Результат решения**

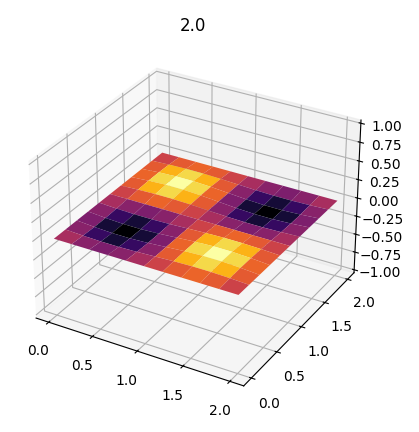
**1.** **2.** ****

**3.** **4.** ****

**5.** **6.** ****

**7.** **8.** ****

**9.** **10.** ****

**11.** ****

На изображениях 1-11 показан результат одного полного колебания, полученного в результате работы функции на временном промежутке [0, 2] секунд вычисляемый с шагом 0.001 с сеткой 10 на 10.

**2.2. Расчёт погрешностей**

Абсолютная погрешность решения: 0.10787115412417425

Погрешность по Рунге: 0.43224571446686944

Данные результаты погрешности получены при разбиении пластины на 100 частей (10 на 10) с шагом по времени 0.001.

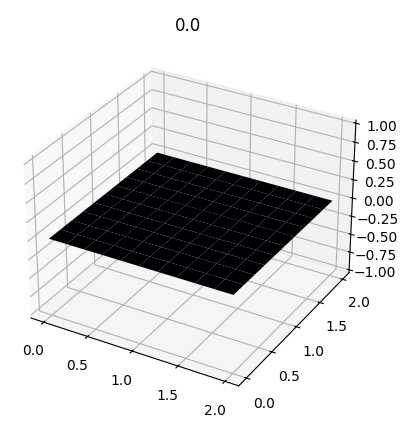
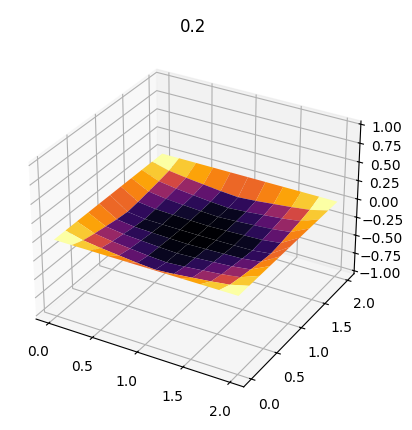
При увеличении числа разбиений до 400 (20 на 20) погрешность значительно уменьшается

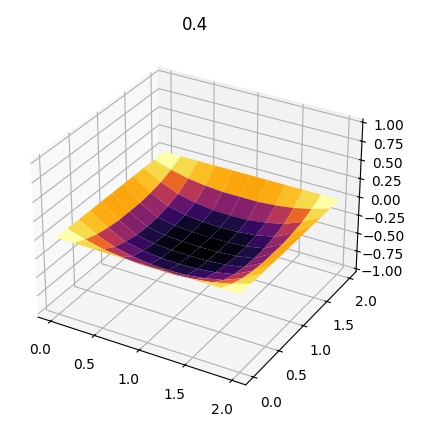
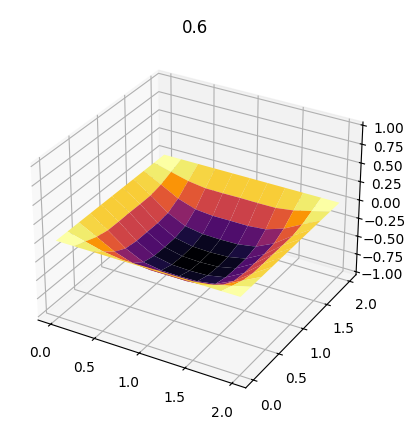
Абсолютная погрешность: 0.028703794186701437

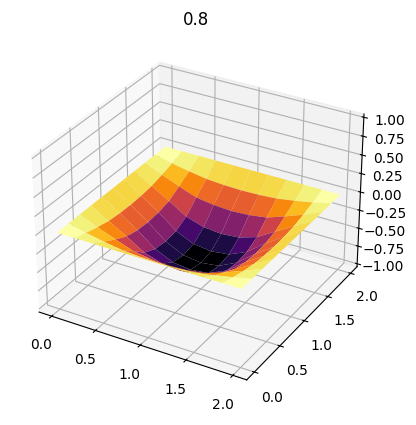
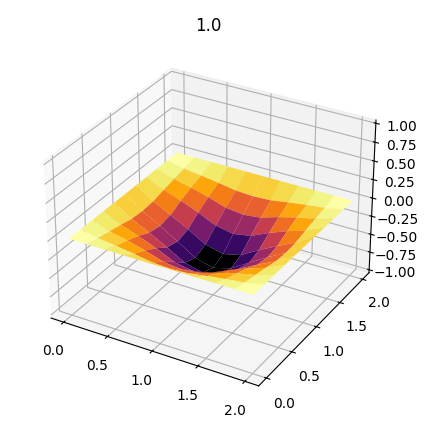
Погрешность по Рунге: 0.08222248858192371

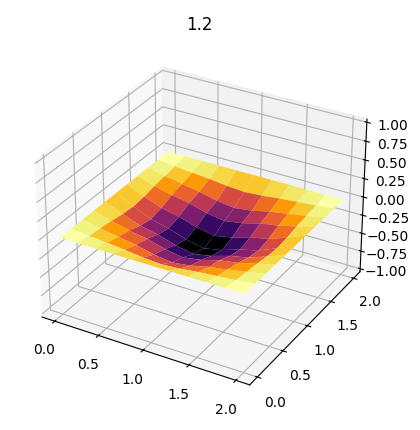
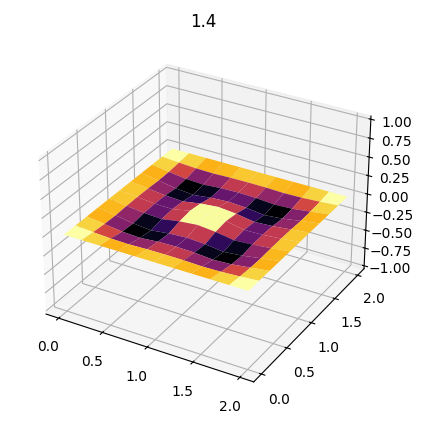
**3. РЕШЕНИЕ ТЕСТОВОГО ПРИМЕРА 2**

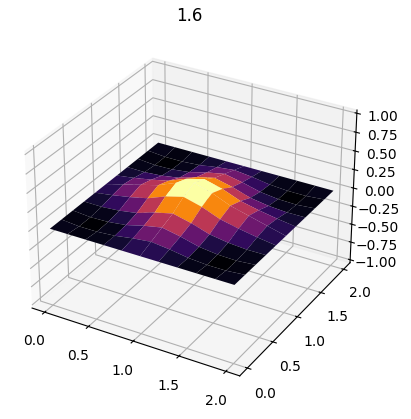
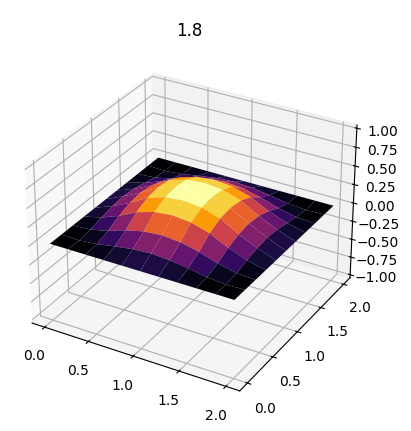
**3.1. Результат решения**

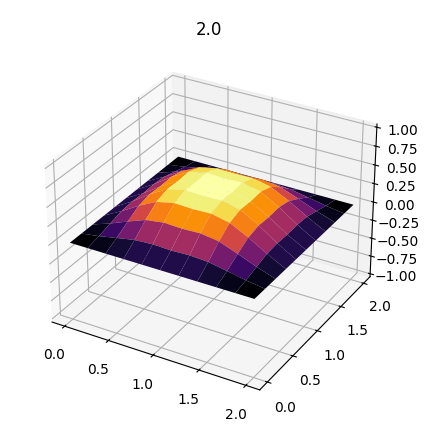
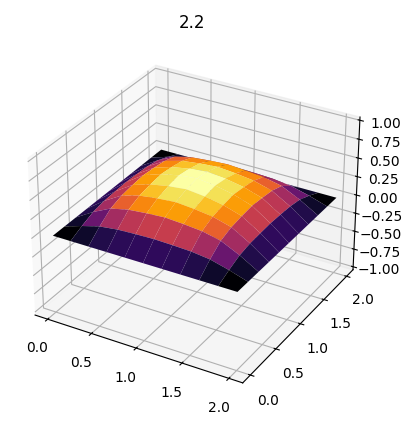
**1.** **2.** ****

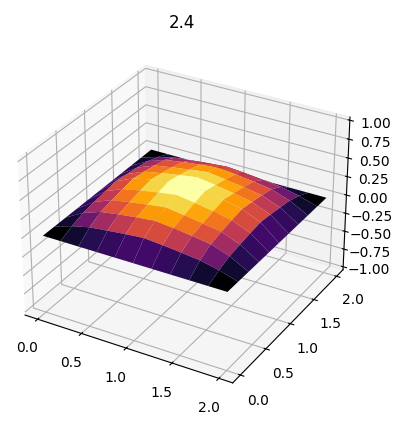
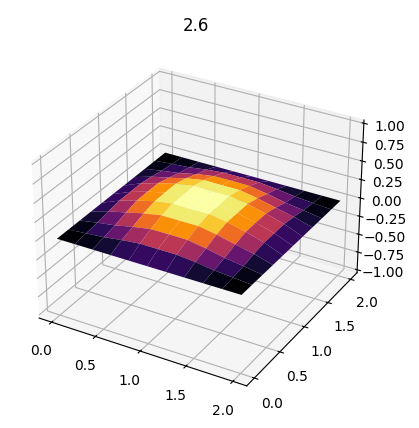
**3.** **4.** ****

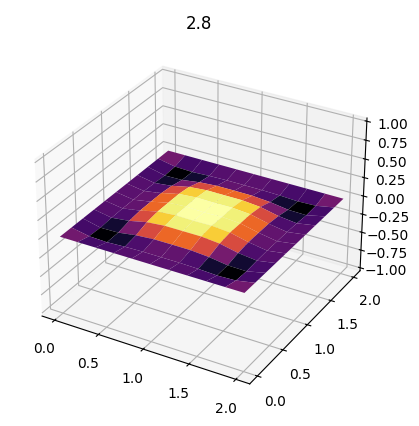
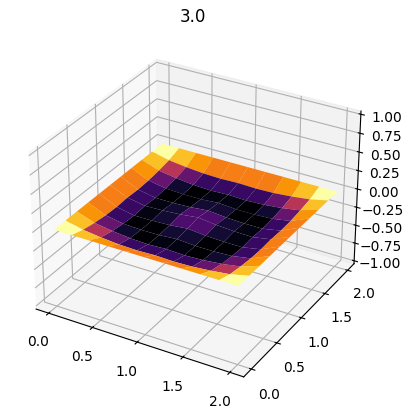
**5.** **6.** ****

**7.** **8.** ****

**9.** **10.** ****

**11.** **12.**

**13.** **14.** ****

**15.** **16.** ****

На изображениях 1-16 показан результат одного полного колебания, полученного в результате работы функции на временном промежутке [0, 3] секунд вычисляемый с шагом 0.001 с сеткой 10 на 10.

**3.2. Расчёт погрешностей**

Абсолютная погрешность решения: 0.7420149739404868

Погрешность по Рунге: 0.15128943133513562

Данные результаты погрешности получены при разбиении пластины на 100 частей (10 на 10) с шагом по времени 0.001.

При увеличении числа разбиений до 400 (20 на 20) погрешность увеличивается

Абсолютная погрешность: 0.8450817828317182

Погрешность по Рунге: 0.18106919441173686

**4. ЗАКЛЮЧЕНИЕ**

При решении задачи использовалась разностная схема, составленная с помощью разностных производных малого порядка, поэтому при численном решении наблюдается большая погрешность. Также из-за выбранного метода решения тестовый пример 2, содержащий модуль, был получен с большой погрешностью. Вычисление решения задачи трудоёмко по времени и по памяти , где n - количество разбиений прямоугольной области, m - количество разбиений по t.

**5. Приложение**

*test1.py*

from experiments import main

from numpy import pi, sin

import numpy as np

from glob import glob

l = 1.0

def real\_y(x, y, t):

return sin(x\*pi/l) \* sin(y\*pi/l) \* sin(t\*pi/l)

def f(x, y, t):

return pi\*pi/l/l \* sin(pi\*x/l) \* sin(pi\*y/l) \* sin(pi\*t/l)

def phi(x, y):

return 0

def psi(x, y):

return pi/l \* sin(pi\*x/l) \* sin(pi\*y/l)

if \_\_name\_\_ == "\_\_main\_\_":

ht = 0.001

n = 10

matrix\_update = False

prev\_update = False

prev\_filename = f"./test1\_diff\_with\_ht\_{ht \* 2}\_n\_{n / 2}.npy"

prev\_matrix\_filename = f"./test1\_diff\_with\_ht\_{ht\*2}\_n\_{n/2}\_matrix.npy"

filename = f"./test1\_diff\_with\_ht\_{ht}\_n\_{n}.npy"

matrix\_filename = f"./test1\_diff\_with\_ht\_{ht}\_n\_{n}\_matrix.npy"

file\_present = glob(filename) and glob(matrix\_filename)

prev\_present = glob(prev\_filename) and glob(prev\_matrix\_filename)

if matrix\_update or not file\_present:

matrix, matrix1 = main(

ht=ht, a=2, b=2, n=n, f=f, phi=phi, psi=psi, real\_y=real\_y

)

np.save(filename, matrix1)

np.save(matrix\_filename, matrix)

else:

matrix1 = np.load(filename)

matrix = np.load(matrix\_filename)

if prev\_update or not prev\_present:

prev\_matrix, prev\_matrix1 = main(

ht=ht \* 2,

a=2,

b=2,

n=int(n/2),

f=f,

phi=phi,

psi=psi,

real\_y=real\_y,

)

np.save(prev\_filename, prev\_matrix1)

np.save(prev\_matrix\_filename, prev\_matrix)

else:

prev\_matrix1 = np.load(prev\_filename)

prev\_matrix = np.load(prev\_matrix\_filename)

# print(matrix1)

print(f"Runge: {np.max(matrix[:-1:2, ::2, ::2] - prev\_matrix)}")

print(np.max(matrix1[:]))

*test2.py*

from experiments import main

from numpy import pi, sin

import numpy as np

from glob import glob

def phi(\_x: float, \_y: float):

return 0

def psi(x: float, y: float):

return abs(x - a / 2) \* abs(y - a / 2) - a / 2

def f(x: float, y: float, t: float):

return -abs(x - a / 2) \* abs(y - a / 2) \* sin(t)

def real\_y(x: float, y: float, t: float):

return abs(x - a / 2) \* abs(y - a / 2) \* sin(t)

if \_\_name\_\_ == "\_\_main\_\_":

ht = 0.001

a = 2

b = 2

n = 20

matrix\_update = False

prev\_update = False

prev\_filename = f"./test2\_diff\_with\_ht\_{ht \* 2}\_n\_{n / 2}.npy"

prev\_matrix\_filename = f"./test2\_diff\_with\_ht\_{ht\*2}\_n\_{n/2}\_matrix.npy"

filename = f"./test2\_diff\_with\_ht\_{ht}\_n\_{n}.npy"

matrix\_filename = f"./test2\_diff\_with\_ht\_{ht}\_n\_{n}\_matrix.npy"

file\_present = glob(filename) and glob(matrix\_filename)

prev\_present = glob(prev\_filename) and glob(prev\_matrix\_filename)

if matrix\_update or not file\_present:

matrix, matrix1 = main(

ht=ht, a=2, b=2, n=n, f=f, phi=phi, psi=psi, real\_y=real\_y

)

np.save(filename, matrix1)

np.save(matrix\_filename, matrix)

else:

matrix1 = np.load(filename)

matrix = np.load(matrix\_filename)

if prev\_update or not prev\_present:

prev\_matrix, prev\_matrix1 = main(

ht=ht \* 2,

a=2,

b=2,

n=int(n/2),

f=f,

phi=phi,

psi=psi,

real\_y=real\_y,

)

np.save(prev\_filename, prev\_matrix1)

np.save(prev\_matrix\_filename, prev\_matrix)

else:

prev\_matrix1 = np.load(prev\_filename)

prev\_matrix = np.load(prev\_matrix\_filename)

# print(matrix1)

print(f"Runge: {np.max(matrix[:-1:2, ::2, ::2] - prev\_matrix)}")

print(np.max(matrix1[:]))

*main.py*

import numpy as np

from numpy import pi

import matplotlib.pyplot as plt

from typing import Callable

def method\_step(\*, prev\_matrix, curr\_matrix, f, k, deltas\_matrix, hx, hy, ht, real\_y):

C1 = hx\*\*2 \* hy\*\*2

C2 = -(ht\*\*2) \* hy\*\*2

C3 = -(ht\*\*2) \* hx\*\*2

C4 = ht\*\*2 \* hx\*\*2 \* hy\*\*2

C0 = -2 \* (C1 + C2 + C3)

next\_matrix = np.zeros(prev\_matrix.shape)

for i in range(1, next\_matrix.shape[0] - 1):

for j in range(1, next\_matrix.shape[1] - 1):

next\_matrix[i][j] = (

-prev\_matrix[i][j]

- (

C0 \* curr\_matrix[i][j]

+ C2 \* (curr\_matrix[i + 1][j] + curr\_matrix[i - 1][j])

+ C3 \* (curr\_matrix[i][j + 1] + curr\_matrix[i][j - 1])

- C4 \* f(j \* hx, i \* hy, (k - 1) \* ht)

)

/ C1

)

deltas\_matrix[i][j] = abs(

next\_matrix[i][j] - real\_y(j \* hx, i \* hy, (k - 1) \* ht)

)

return next\_matrix

def show\_deltas(\*, deltas\_to\_show, a, b):

fig = plt.figure()

ax = fig.add\_subplot(projection="3d")

# ax.set\_zlim(-1, 1)

x, y = np.meshgrid(

np.linspace(0, a, deltas\_to\_show.shape[0], endpoint=True),

np.linspace(0, b, deltas\_to\_show.shape[1], endpoint=True),

)

ax.plot\_surface(x, y, deltas\_to\_show, cmap="inferno", rstride=1, cstride=1)

plt.show()

return 0

def show\_plot(\*, matrix\_to\_show, i, a, b, ht):

fig = plt.figure()

ax = fig.add\_subplot(projection="3d")

ax.set\_title(ht \* i)

ax.set\_zlim(-1, 1)

x, y = np.meshgrid(

np.linspace(0, a, matrix\_to\_show.shape[0], endpoint=True),

np.linspace(0, b, matrix\_to\_show.shape[1], endpoint=True),

)

ax.plot\_surface(x, y, matrix\_to\_show, cmap="inferno", rstride=1, cstride=1)

plt.show()

return 0

def main(

\*,

a: float,

b: float,

ht: float,

n: int,

real\_y: Callable[[float, float, float], float],

f: Callable[[float, float, float], float],

phi: Callable[[float, float], float],

psi: Callable[[float, float], float],

):

fps = int(1 / ht)

t\_steps = int(pi \* 2 \* fps)

x = np.linspace(0, a, n + 1, endpoint=True)

y = np.linspace(0, b, n + 1, endpoint=True)

x, y = np.meshgrid(x, y)

hx = a / n

hy = b / n

matrix = np.zeros((t\_steps, n + 1, n + 1))

deltas = np.zeros((t\_steps, n + 1, n + 1))

for i in range(1, n):

for j in range(1, n):

matrix[0][i][j] = phi(j \* hx, i \* hy)

matrix[1][i][j] = ht \* psi(j \* hx, i \* hy) + matrix[0][i][j]

show\_plot(matrix\_to\_show=matrix[0], ht=0, a=a, b=b, i=0)

show\_plot(matrix\_to\_show=matrix[1], ht=ht, a=a, b=b, i=1)

for k in range(2, t\_steps):

matrix[k] = method\_step(

prev\_matrix=matrix[k - 2],

curr\_matrix=matrix[k - 1],

f=f,

k=k,

deltas\_matrix=deltas[k],

hx=hx,

hy=hy,

ht=ht,

real\_y=real\_y,

)

show\_plot(matrix\_to\_show=matrix[k], i=k, ht=ht, a=a, b=b)

return matrix, deltas