Лабораторная работа №7

Выполнила Прудникова А. А. М8О-408Б-20

Вариант 4

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. Исследовать зависимость погрешности от сеточных параметров.

Ввод [1]:
```
import numpy as np
import random
import matplotlib.pyplot as plt
import math
import sys
import ipywidgets as widgets
import warnings

from functools import reduce
from mpl_toolkits.mplot3d import Axes3D
from ipywidgets import interact
from IPython.display import display
```

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$$

$$\begin{cases} u_x(0,\ y) = \phi_0(y) = e^y \\ u_x(\pi,\ y) = \phi_1(y) = -e^y \\ u(x,\ 0) = \sin x \\ u(x, 1) = e \sin x \end{cases}$$

$$u(x, t) = e^y \sin x$$

Ввод [2]:
```
def psi_0(x):
    return math.sin(x)

def psi_1(x):
    return math.sin(x) * math.e

def phi_0(y):
    return math.exp(y)

def phi_1(y):
    return -math.exp(y)

def u(x, y):
    return math.exp(y)*math.sin(x)
```

Ввод [3]:

```
class Schema:
    def __init__(self, psi0 = psi_0, psi1 = psi_1, phi0 = phi_0, phi1 = phi_1
                 lx1 = math.pi, ly0 = 0, ly1 = 1, solver="zeidel", relax=0.1,
        self.psi1 = psi1
        self.psi0 = psi0
        self.phi0 = phi0
        self.phi1 = phi1
        self.lx0 = lx0
        self.ly0 = ly0
        self.lx1 = lx1
        self.ly1 = ly1
        self.eps = epsilon
        self.method = None
        if solver == "zeidel":
            self.method = self.zeidel_step
        elif solver == "simple":
            self.method = self.simple_eiler_step
        elif solver == "relaxation":
            self.method = lambda x, y, m: self.relaxation_step(x, y, m, relax
        else:
            raise ValueError("Wrong solver name")


    def zeidel_step(self, X, Y, M):
        return self.relaxation_step(X, Y, M, w=1)

    def relaxation_step(self, X, Y, M, w):
        norm = 0.0
        hx2 = self.hx * self.hx
        hy2 = self.hy * self.hy

        for i in range(1, self.Ny - 1):
            diff = w*((-2*self.hx*self.phi0(Y[i][0]) + 4*M[i][1] - M[i][2])/3
            M[i][0] += diff
            diff = abs(diff)
            norm = diff if diff > norm else norm
            for j in range(1, self.Nx - 1):
                diff = hy2*(M[i][j-1] + M[i][j+1])
                diff += hx2*(M[i-1][j] + M[i+1][j])
                diff /= 2*(hy2 + hx2)
                diff -= M[i][j]
                diff *= w
                M[i][j] += diff
                diff = abs(diff)
                norm = diff if diff > norm else norm
            diff = w*((2*self.hx*self.phi1(Y[i][-1]) + 4*M[i][-2] - M[i][-3])/
            M[i][-1] += diff
            diff = abs(diff)
            norm = diff if diff > norm else norm

        return norm

    def simple_eiler_step(self, X, Y, M):
        temp = [[0.0 for _ in range(self.Nx)] for _ in range(self.Ny)]
        norm = 0.0
        hx2 = self.hx * self.hx
```

```
        hy2 = self.hy * self.hy

        for i in range(1, self.Ny - 1):
            temp[i][0] = (-2*self.hx*self.phi0(Y[i][0]) + 4*M[i][1] - M[i][2]
            diff = abs(temp[i][0] - M[i][0])
            norm = diff if diff > norm else norm
            for j in range(1, self.Nx - 1):
                temp[i][j] = hy2*(M[i][j-1] + M[i][j+1])
                temp[i][j] += hx2*(M[i-1][j] + M[i+1][j])
                temp[i][j] /= 2*(hy2 + hx2)
                diff = abs(temp[i][j] - M[i][j])
                norm = diff if diff > norm else norm
            temp[i][-1] = (2*self.hx*self.phi1(Y[i][-1]) + 4*M[i][-2] - M[i][
            diff = abs(temp[i][0] - M[i][0])
            norm = diff if diff > norm else norm

        for i in range(1, self.Ny - 1):
            M[i] = temp[i]

        return norm


    def set_l0_l1(self, lx0, lx1, ly0, ly1):
        self.lx0 = lx0
        self.lx1 = lx1
        self.ly0 = ly0
        self.ly1 = ly1


    def _compute_h(self):
        self.hx = (self.lx1 - self.lx0) / (self.Nx - 1)
        self.hy = (self.ly1 - self.ly0) / (self.Ny - 1)


    @staticmethod
    def nparange(start, end, step = 1):
        now = start
        e = 0.00000000001
        while now - e <= end:
            yield now
            now += step

    def init_values(self, X, Y):
        ans = [[0 for _ in range(self.Nx)] for _ in range(self.Ny)]
        for j in range(self.Nx):
            coeff = (self.psi1(X[-1][j]) - self.psi0(X[0][j])) / (self.ly1 -
            addition = self.psi0(X[0][j])
            for i in range(self.Ny):
                ans[i][j] = coeff*(Y[i][j] - self.ly0) + addition
        return ans


    def __call__(self, Nx=10, Ny=10):
        self.Nx, self.Ny = Nx, Ny
        self._compute_h()

        x = list(self.nparange(self.lx0, self.lx1, self.hx))
```

```
        y = list(self.nparange(self.ly0, self.ly1, self.hy))
        X = [x for _ in range(self.Ny)]
        Y = [[y[i] for _ in x] for i in range(self.Ny)]
        ans = self.init_values(X, Y)

        self.itters = 0

        while(self.method(X, Y, ans) >= self.eps):
            self.itters += 1

        return X, Y, ans
```

Зависимость погрешности от параметра h

Вычисление погрешности приближенного решения:

$$RMSE = \sqrt{\frac{\sum_{i=0}^{Ny} \sum_{j=0}^{Nx} (y_i - \hat{y}_i)^2}{N_x \cdot N_y}}$$

Ввод [4]:

```
def epsilon(x, y, z, f):
    ans = 0.0
    for i in range(len(z)):
        for j in range(len(z[i])):
            ans += (z[i][j] - f(x[i][j], y[i][j]))**2
    return (ans / (len(z) * len(z[0])))**0.5
```

Ввод [8]:

```
def get_graphic_h(solver, real_f):
    h = []
    e = []
    for N in range(4, 80, 3):
        x, y, z = solver(N, N)
        h.append(solver.hx)
        e.append(epsilon(x, y, z, real_f))
    return h, e
```

Ввод [9]:

```
explict = Schema(epsilon=0.00001)
```

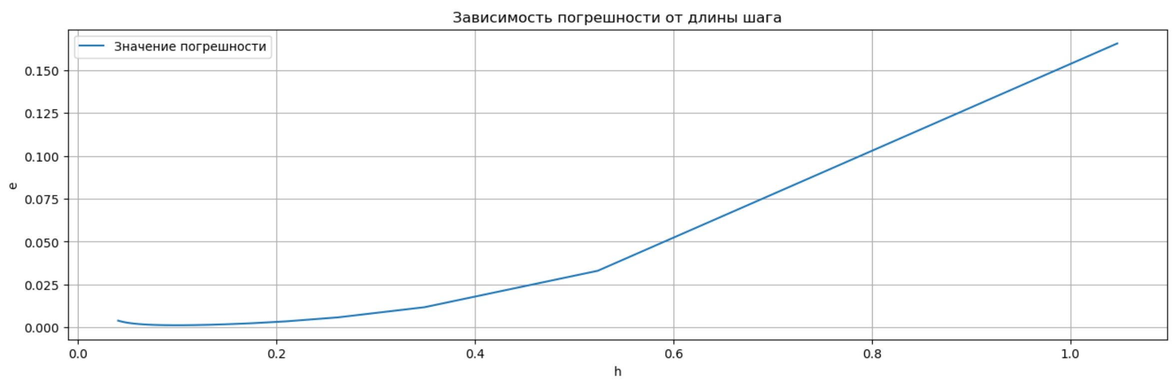
Ввод [7]:

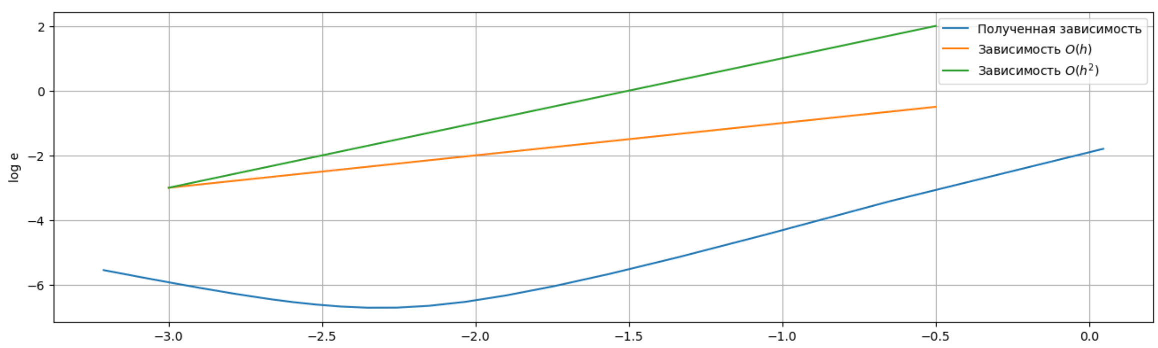
```
plt.figure(figsize = (16, 10))

plt.subplot(2, 1, 1)
plt.title("Зависимость погрешности от длины шага")
h, e = get_graphic_h(explict, u)

plt.plot(h, e, label="Значение погрешности")
plt.xlabel("h")
plt.ylabel("e")
plt.legend()
plt.grid()

plt.subplot(2, 1, 2)
plt.plot(list(map(math.log, h)), list(map(math.log, e)), label="Полученная за
plt.plot([-3, -0.5], [-3, -0.5], label="Зависимость $O(h)$")
plt.plot([-3, -0.5], [-3, 2], label="Зависимость $O(h^2)$")
plt.xlabel("log h")
plt.ylabel("log e")
plt.legend()
plt.grid()
```





Ввод [10]:
```
def real_z(lx0, lx1, ly0, ly1, f):
    x = np.arange(lx0, lx1 + 0.005, 0.005)
    y = np.arange(ly0, ly1 + 0.005, 0.005)
    X = np.ones((y.shape[0], x.shape[0]))
    Y = np.ones((x.shape[0], y.shape[0]))
    Z = np.ones((y.shape[0], x.shape[0]))
    for i in range(Y.shape[0]):
        Y[i] = y
    Y = Y.T
    for i in range(X.shape[0]):
        X[i] = x
    for i in range(Z.shape[0]):
        for j in range(Z.shape[1]):
            Z[i, j] = f(X[i, j], Y[i, j])
    return X, Y, Z
```

Сходимость методов

Ввод [11]:
```
schema = Schema(epsilon=0.001, solver="simple")
schema(10, 10)
print("Всего иттераций метода простых иттераций:", schema.itters)

schema = Schema(epsilon=0.001)
schema(10, 10)
print("Всего иттераций метода Зейделя:", schema.itters)

schema = Schema(epsilon=0.001, solver="relaxation", relax=1.5)
schema(10, 10)
print("Всего иттераций метода релаксаций:", schema.itters)
```

```
Всего иттераций метода простых иттераций: 41
Всего иттераций метода Зейделя: 27
Всего иттераций метода релаксаций: 13
```

Ввод [ ]:
```

```