Московский авиационный институт

(Национальный исследовательский университет)

Институт №8 «Информационные технологии и прикладная математика»

Кафедра вычислительной математики и программирования

```
Лабораторная работа № 7
по курсу «Численные методы».

Тема: «МЕТОД КОНЕЧНЫХ РАЗНОСТЕЙ ДЛЯ РЕШЕНИЯ
        УРАВНЕНИЙ ЭЛЛИПТИЧЕСКОГО ТИПА».
```

```
Студент: Кондратьев Е.А.
Группа: 80-406Б
Преподаватель: Ревизников Д.Л.
Преподаватель: Пивоваров Д.Е.
```

```
Москва, 2022
```

Лабораторная работа №7

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*. 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 τ и h.

Вариант №4

```
import ipywidgets as widgets
from ipywidgets import interact
from IPython.display import display
import random
import·matplotlib.pyplot·as·plt
import·math
import sys
import warnings
import numpy as np
from functools import reduce
from mpl_toolkits.mplot3d import Axes3D
import plotly.offline as offline
from plotly.graph_objs import *
```

Уравнение:

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

Граничные условия:

$$\begin{cases} u(0,\, y) = \phi_0(y) = \cos y \\ u(\frac{\pi}{2},\, y) = \phi_1(y) = 0 \\ u(x,\, 0) = \psi_0(x) = \cos x \\ u(x,\, \frac{\pi}{2}) = \psi_1(x) = 0 \end{cases}$$

Аналитическое решение:

$$u(x, t) = \cos x \cos y$$

```
def psi_0(x):
    return math.cos(x)

def psi_1(x):
    return 0.0

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

def phi_1(y):
    return 0.0

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

Конечно-разностная схема

Будем решать задачу на заданном промежутке от 0 до l_x по координате x и на промежутке от 0 до l_y по координате y.

Рассмотрим конечно-разностную схему решения краевой задачи на сетке с граничными параметрами l_x, l_y и параметрами насыщенности сетки N_x, N_y. Тогда размер шага по каждой из координат будет:

$$h_x = \frac{l_x}{N_x - 1},\; h_y = \frac{N_y}{N_y - 1}$$

Теперь давайте определим связь между дискретными значениями функции с помощью разностной апроксимации производной:

$$\frac{\partial^2 u}{\partial x^2}(x_j, y_i) + \frac{\partial^2 u}{\partial y^2}(x_j, y_i) + 2u(x_j, y_i) = \frac{u_{j-1,i} - 2u_{j,i} + u_{j+1,i}}{h_x^2} + \frac{u_{j,i-1} - 2u_{j,i} + u_{j,i+1}}{h_y^2} + 2u_{j,i}$$

Теперь выразим $u_{i,j} = \frac{h_y^2(u_{j-1,i} + u_{j+1,i}) + h_x^2(u_{j,i-1} + u_{j,i+1})}{2(h_x^2 + h_y^2 - h_y^2 h_x^2)}$, это будет основой для применения итерационных методов решения СЛАУ.

Для расчета $u_{j,0}$ и $u_{0,i}$ используем граничные условия.

Начальная инициализация

Поскольку в нашем варианте известны граничные значения $u(x, l_{y0})$ и $u(x, l_{y1})$, то для начальной инициаизации значений в сетке можно использовать линейную интерполяцию при фиксированном $x = x_j$ для улучшения сходимости:

$$u_{j,i} = \frac{u(x_j, l_{y1}) - u(x_j, l_{y0})}{l_{y1} - l_{y0}} \cdot (y_i - l_{y0}) + u(x_j, l_{y0})$$

Методы решения СЛАУ

Для решения СЛАУ можно воспользоваться итерационными методами, такими как *метод простых иттераций*, *метод Зейделя* и *метод верхних релаксаций*. Первые два метода были изучены нами ранее, когда как последний является небольшой модификацией *метода Зейделя* с добавлением параметра *w*, который позволяет регулировать скорость сходимости метода. *w* = 1 соответствует методу Зейделя. Итерационный метод при *w* > 1 - *метод верхней релаксации*.

Реализация

```
class Schema:
    def __init__(self, psi0 = psi_0, psi1 = psi_1, phi0 = phi_0, phi1 = phi_1,
                 lx0 = 0, lx1 = math.pi/2, ly0 = 0, ly1 = math.pi/2,
                 solver="zeidel", relax=0.1, epsilon = 0.01):
        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.ZeidelStep
        elif solver == "simple":
            self.method = self.SimpleEulerStep
        elif solver == "relaxation":
            self.method = lambda x, y, m: self.RelaxationStep(x, y, m, relax)
        else:
            raise ValueError("Wrong solver name")
```

```
def RelaxationStep(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):
        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 - hx2 * hy2)
            diff -= M[i][j]
            diff *= w
            M[i][j] += diff
            diff = abs(diff)
            norm = diff if diff > norm else norm
    return norm


def SimpleEulerStep(self, X, Y, M):
    tmp = [[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):
        tmp[i][0] = M[i][0]
        for j in range(1, self.Nx - 1):
            tmp[i][j] = hy2 * (M[i][j - 1] + M[i][j + 1])
            tmp[i][j] += hx2 * (M[i - 1][j] + M[i + 1][j])
            tmp[i][j] /= 2 * (hy2 + hx2 - hx2 * hy2)
            diff = abs(tmp[i][j] - M[i][j])
            norm = diff if diff > norm else norm
        tmp[i][-1] = M[i][-1]
    for i in range(1, self.Ny - 1):
        M[i] = tmp[i]
    return norm


def ZeidelStep(self, X, Y, M):
    return self.RelaxationStep(X, Y, M, w=1)


def Set_l0_l1(self, lx0, lx1, ly0, ly1):
    self.lx0 = lx0
    self.lx1 = lx1
    self.ly0 = ly0
    self.ly1 = ly1


def CalculateH(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 InitializeValues(self, X, Y):
        ans = [[0.0 for _ in range(self.Nx)] for _ in range(self.Ny)]
        for j in range(1, self.Nx - 1):
            coeff = (self.psi1(X[-1][j]) - self.psi0(X[0][j])) / (self.ly1 - self.ly0)
            addition = self.psi0(X[0][j])
            for i in range(self.Ny):
                ans[i][j] = coeff*(Y[i][j] - self.ly0) + addition
        for i in range(self.Ny):
            ans[i][0] = self.phi0(Y[i][0])
            ans[i][-1] = self.phi1(Y[i][-1])
        return ans


    def __call__(self, Nx=10, Ny=10):
        self.Nx, self.Ny = Nx, Ny
        self.CalculateH()
        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.InitializeValues(X, Y)
        self.itters = 0
        while(self.method(X, Y, ans) >= self.eps):
            self.itters += 1
        return X, Y, ans
```

Результаты

Вычисление погрешностей

Вычисление погрешности приближенного решения по формуле:

$$MSE = \frac{\sum\limits_{i=0}^{Ny}\sum\limits_{j=0}^{Nx}(y_{ij} - \hat{y_{ij}})^2}{N_x \cdot N_y}$$

```
def Error(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
```

Построение зависимости погрешности от шага h.

```
def GetStepHandError(solver, real_f):
    h , e = [], []
    for N in range(5, 50, 4):
```

```
        x, y, z = solver(N, N)
        h.append(solver.hx)
        e.append(Error(x, y, z, real_f))
    return h, e
```

Сходимость методов

```
schema = Schema(epsilon=0.001, solver="simple")
schema(10, 10)
print("Количество итераций метода простых итераций = {}".format(schema.itters))
```

```
Количество итераций метода простых итераций = 45
```

```
schema = Schema(epsilon=0.001)
schema(10, 10)
print("Количество итераций метода Зейделя = {}".format(schema.itters))
```

```
Количество итераций метода Зейделя = 29
```

```
schema = Schema(epsilon=0.001, solver="relaxation", relax=1.5)
schema(10, 10)
print("Количество итераций метода релаксаций = {}".format(schema.itters))
```

```
Количество итераций метода релаксаций = 12
```

График погрешностей

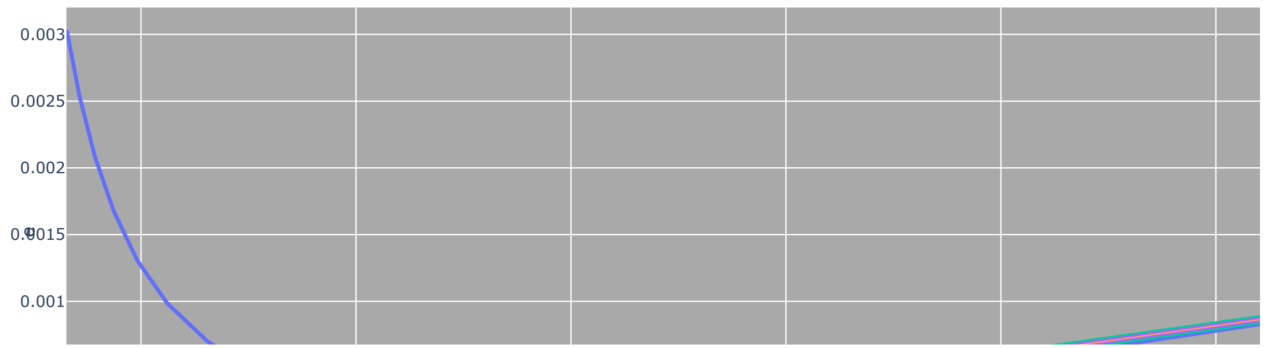
Построим график зависимости погрешности ϵ от размера шага h_x по координате x, но уменьшать размер шага будем пропорционально уменьшению шага h_y по координате y.

```
explict1 = Schema(epsilon=0.00001, solver="simple", relax=1.55)
h1, e1 = GetStepHandError(explict1, u)
explict2 = Schema(epsilon=0.000001, solver="simple", relax=1.55)
h2, e2 = GetStepHandError(explict2, u)
explict3 = Schema(epsilon=0.0000001, solver="simple", relax=1.55)
h3, e3 = GetStepHandError(explict3, u)
explict4 = Schema(epsilon=0.00000001, solver="simple", relax=1.55)
h4, e4 = GetStepHandError(explict4, u)
explict5 = Schema(epsilon=0.000000001, solver="simple", relax=1.55)
h5, e5 = GetStepHandError(explict5, u)
```

```
trace1 = Scatter(
    x = h1,
    y = e1,
    line= dict(width=3),
    name = 'Значение погрешности при точности метода epsilon=0.00001',
    mode = 'lines',
    text = ('(x, y)'),
    showlegend = True
)

trace2 = Scatter(
    x = h2,
```

```
    y = e2,
    line= dict(width=3),
    name = 'Значение погрешности при точности метода epsilon=0.000001',
    mode = 'lines',
    text = ('(x, y)'),
    showlegend = True
)

trace3 = Scatter(
    x = h3,
    y = e3,
    line= dict(width=7),
    name = 'Значение погрешности при точности метода epsilon=0.0000001',
    mode = 'lines',
    text = ('(x, y)'),
    showlegend = True
)

trace4 = Scatter(
    x = h4,
    y = e4,
    line= dict(width=4),
    name = 'Значение погрешности при точности метода epsilon=0.00000001',
    mode = 'lines',
    text = ('(x, y)'),
    showlegend = True
)

trace5 = Scatter(
    x = h5,
    y = e5,
    line= dict(width=1),
    name = 'Значение погрешности при точности метода epsilon=0.000000001',
    mode = 'lines',
    text = ('(x, y)'),
    showlegend = True
)

data = [trace1, trace2, trace3, trace4, trace5]

layout = Layout(
    title = 'Зависимость погрешности от длины шага',
    xaxis = dict(title = 'h'),
    yaxis = dict(title = 'e'),
    plot_bgcolor = 'darkgray',
    paper_bgcolor = 'gray'
)

fig = Figure(data = data, layout = layout)
offline.iplot(fig)
```

Зависимость погрешности от длины шага



Сетка для реальной функции

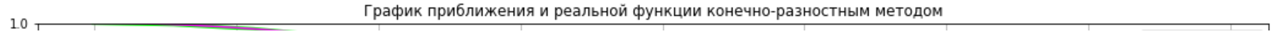
```
def realZ(lx0, lx1, ly0, ly1, f):
    x = np.arange(lx0, lx1 + 0.005, 0.005)
    y = np.arange(ly0, ly1 + 0.005, 0.005)
    X = np.ones((y.shape[0], x.shape[0]))
    Y = np.ones((x.shape[0], y.shape[0]))
    Z = np.ones((y.shape[0], x.shape[0]))
    for i in range(Y.shape[0]):
        Y[i] = y
    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


def plotDependence(Nx = 5, Ny=5, eps=1):
    schema = Schema(epsilon=eps, solver="simple", relax=1.4)
    x, y, z = schema(Nx, Ny)
    plt.figure(figsize=(18, 20))
    xr, yr, zr=realZ(0, math.pi/2, 0, math.pi / 2, u)
    for i in range(1, 6):

        plt.subplot(5, 1, i)
        j = (Ny * (i - 1)) // 5
        jr = (len(xr) * (i - 1)) // 5
        plt.plot(x[j], z[j], label="y = " + str(y[j][0]), linewidth=3, color='#00FF00')
        plt.plot(xr[jr], zr[jr], label='real function', color='#FF00FF')
        plt.ylim(0, 1)
        plt.title('График приближения и реальной функции конечно-разностным методом')
        plt.grid()
        plt.legend()

listEpsil = [1.0 / (10.0**n) for n in range(6)]
interact(plotDependence, Nx=(15, 100, 2), Ny=(15, 100, 3), eps=listEpsil)
None
```

Nx 15

Ny 15

eps 1.0

График приближения и реальной функции конечно-разностным методом

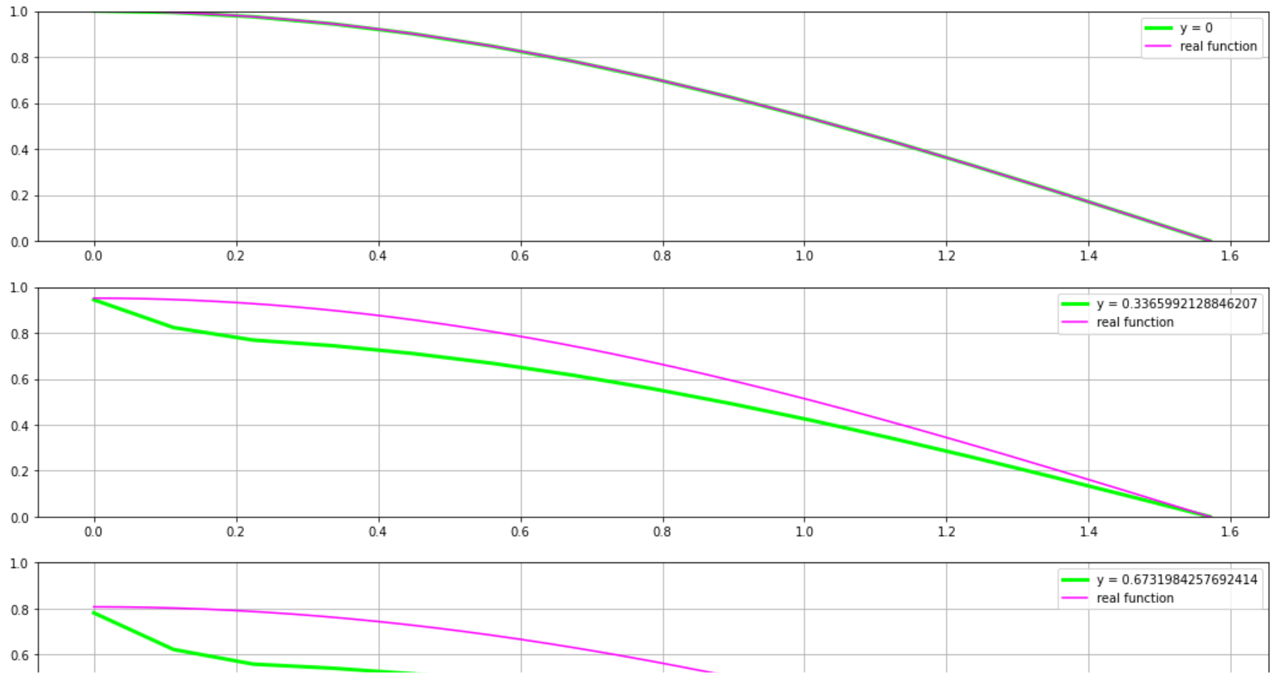



```
def plotDependence(Nx = 5, Ny=5, eps=1):
    schema = Schema(epsilon=eps, solver="simple", relax=1.4)
    x, y, z = schema(Nx, Ny)
    plt.figure(figsize=(18, 20))
    xr, yr, zr=realZ(0, math.pi/2, 0, math.pi / 2, u)
    for i in range(1, 6):

        plt.subplot(5, 1, i)
        j = (Ny * (i - 1)) // 5
        jr = (len(xr) * (i - 1)) // 5
        plt.plot(x[j], z[j], label="y = " + str(y[j][0]), linewidth=3, color='#00FF00')
        plt.plot(xr[jr], zr[jr], label='real function', color='#FF00FF')
        plt.ylim(0, 1)
        plt.grid()
        plt.legend()

listEpsil = [1.0 / (10.0**n) for n in range(6)]
interact(plotDependence, Nx=(15, 100, 2), Ny=(15, 100, 3), eps=listEpsil)
None
```

Nx 15

Ny 15

eps 1.0



Трёхмерное представление

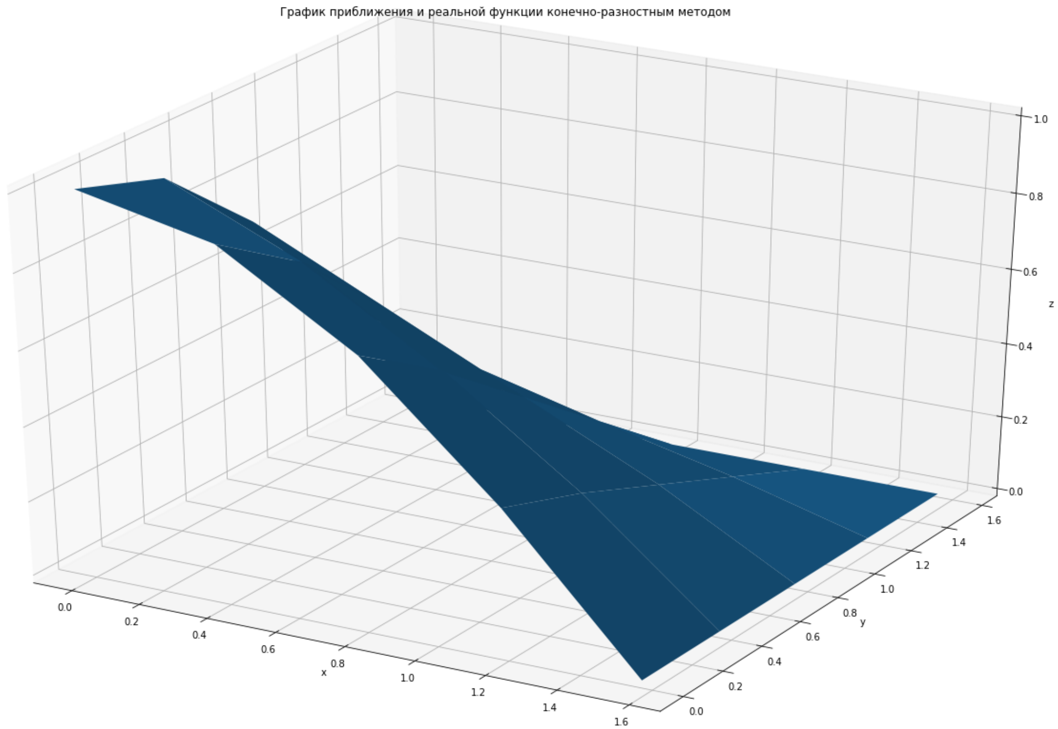
```
def plot_1(Nx = 5, Ny=5, eps=1, plot = False):
    schema = Schema(epsilon=eps, solver="simple", relax=1.4)
    x, y, z = schema(Nx, Ny)
    fig = plt.figure(num=1, figsize=(19, 12), clear=True)
    ax = fig.add_subplot(1, 1, 1, projection='3d')
    if plot:
        ax.plot_wireframe(*realZ(0, math.pi/2, 0, math.pi / 2, u), color="green")
    ax.plot_surface(np.array(x), np.array(y), np.array(z))
    ax.set(xlabel='x', ylabel='y', zlabel='z',
           title='График приближения и реальной функции конечно-разностным методом')
    fig.tight_layout()

listEpsil = [1.0 / (10.0**n) for n in range(6)]
interact(plot_1, Nx=(5, 100, 2), Ny=(5, 100, 3), eps=listEpsil,  plot_true = [False, True])
None
```

Nx 5

Ny 5

eps 1.0

plot



Вывод: для выполнения данной лабораторной работы нужно было решить краевую задачу для дифференциального уравнения эллиптического типа. Также нужно было аппроксимировать уравнения с использованием центрально-разностной схемы. Для решения дискретного аналога пришлось вспомнить метод простых итераций (метод Либмана), метод Зейделя и применить новый для меня метод простых итераций с верхней релаксацией. Вычислил погрешности численного решения путем сравнения с приведенным в задании аналитическим решением.

0 сек. выполнено в 17:05