**ОТЧЁТ**

**ПО ЛАБОРАТОРНОЙ РАБОТЕ №10**

**Уравнения эллиптического типа**

**Вариант 9**

Выполнил студент 3 курса МОиАИС

Ходосевич Данила

***Цель работы***:

усвоить методы решения ***линейного дифференциального уравнения 2-го порядка эллиптического типа***.

Численное решение дифференциального уравнения в частных производных предполагает получение двумерной числовой таблицы приближенных значений *Uij* искомой функции *U*(*x,y)* с заданной точностью для некоторых значений аргументов

*xi Î* [*a*, *b*], *yj Î* [*c*, *d*]

***Задание.***

Решить эллиптическое уравнение

∂2U∂x2+∂2U∂y2=f(x,y)𝜕2U𝜕x2+𝜕2U𝜕y2=f(x,y)

методами 2-го порядка точности.

Сетки по x и по y взять равномерные.

Шаблон для разностной схемы:



Для решения разностных уравнений применить:

А) метод простой итерации

Б) метод Зейделя

Оценивать погрешность решения с помощью сравнения двух последовательных итераций.

Взять сетки размерами 5´5 ячеек и 10´10 ячеек и сравнить полученные решения.

Для всех вариантов [*a*, *b*] = [0; 10], [*c*, *d*] = [0; 10]. Погрешность решения 0,01.

Для всех вариантов граничные условия

U(x,c)=x+c, U(x,d)=x+d,U(x,c)=x+c, U(x,d)=x+d,

U(a,y)=a+y, U(b,y)=b+yU(a,y)=a+y, U(b,y)=b+y

|  |  |
| --- | --- |
| № вар. | Правая часть |

|  |  |
| --- | --- |
| 9 | f(x,y)=y(10−x)f(x,y)=y(10−x) |

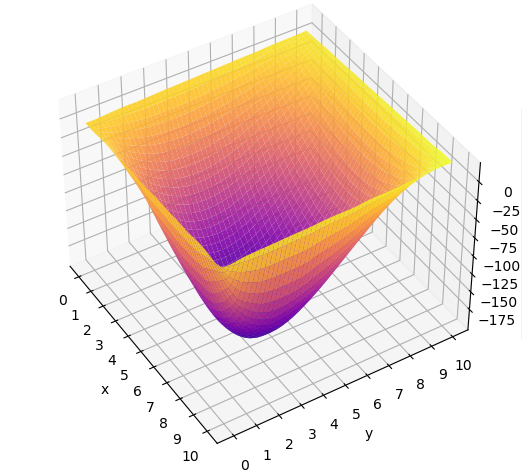
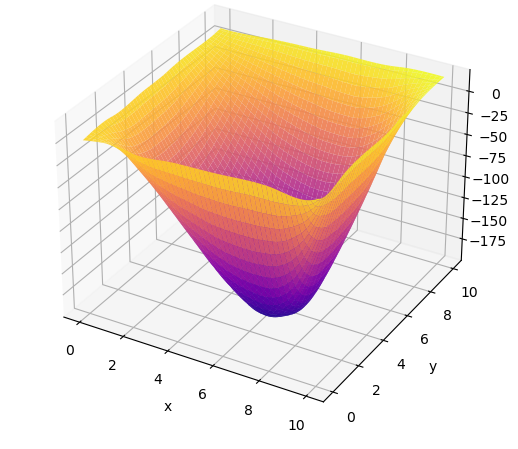
Метод простых итераций:

Для вычисления значений узлов на k+1 итерации, используем формулу:

.

Итерация продолжается до тех пор, пока выполняется условие

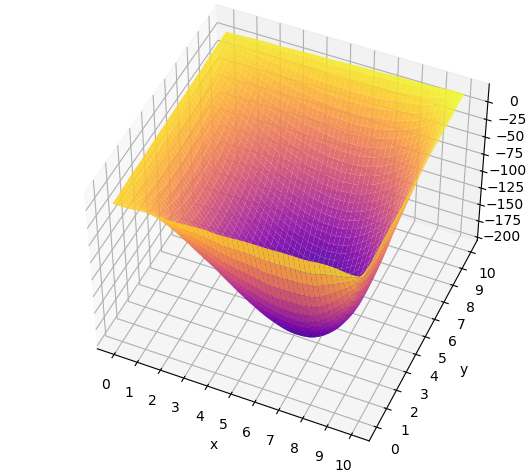
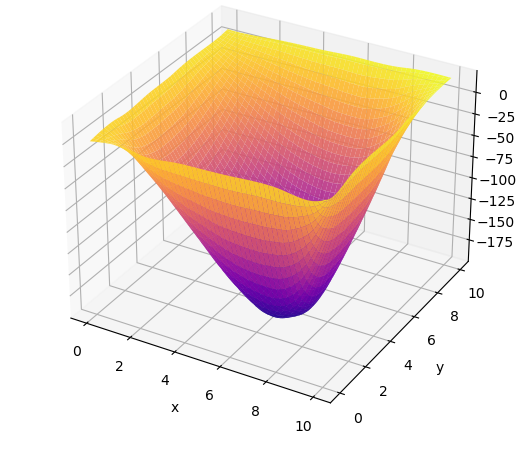
Сетка 5x5 ячеек Сетка 10x10 ячеек



Метод Зейделя

В методе Зейделя, вместо вычисления узлов на k+1 итерации, мы будем использовать текущую итерацию. В методе используем формулу:

Сетка 5х5 ячеек Сетка 10х10 ячеек



Приложения:

import numpy as np

import matplotlib.pyplot as plt

from scipy.interpolate import griddata

# Определение граничных условий

def boundary\_conditions():

return {

'Ux0': lambda x: x,

'Ux1': lambda x: x + 10,

'U0y': lambda y: y,

'U1y': lambda y: y + 10

}

# Правая часть уравнения

def equation\_rhs(x, y):

return y \* (10 - x)

# Функция для визуализации результата

def visualize\_solution(grid, step, method):

size = len(grid)

x\_vals = np.linspace(0, 10, size)

y\_vals = np.linspace(0, 10, size)

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

grid\_x, grid\_y = np.mgrid[0:10:100j, 0:10:100j]

grid\_z = griddata((x.flatten(), y.flatten()), np.array(grid).flatten(), (grid\_x, grid\_y), method='cubic')

fig = plt.figure(figsize=(8, 6))

ax = fig.add\_subplot(111, projection='3d')

ax.set\_title(f"Метод: {method}, Сетка: {size - 1}x{size - 1}")

ax.set\_xlabel("x")

ax.set\_ylabel("y")

ax.set\_zlabel("U")

surface = ax.plot\_surface(grid\_x, grid\_y, grid\_z, cmap="plasma", edgecolor='none', alpha=0.9)

ticks = np.linspace(0, 10, int(10 / step) + 1)

ax.set\_xticks(ticks)

ax.set\_yticks(ticks)

fig.colorbar(surface, ax=ax, shrink=0.5, aspect=10)

plt.show()

# Универсальная функция для решения методом итераций

def solve\_with\_iterations(step=0.1, epsilon=0.01, method='simple'):

size = int(10 / step) + 1

grid = np.zeros((size, size))

boundaries = boundary\_conditions()

# Установка граничных условий

for i in range(size):

x = step \* i

grid[i][0] = boundaries['Ux0'](x)

grid[i][-1] = boundaries['Ux1'](x)

for j in range(1, size):

y = step \* j

grid[0][j] = boundaries['U0y'](y)

grid[-1][j] = boundaries['U1y'](y)

# Итерационный процесс

while True:

max\_diff = 0

new\_grid = grid.copy() if method == 'simple' else grid

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

for j in range(1, size - 1):

x, y = step \* i, step \* j

updated\_value = (grid[i + 1][j] + grid[i - 1][j] + grid[i][j + 1] + grid[i][j - 1] - step\*\*2 \* equation\_rhs(x, y)) / 4

if method == 'simple':

new\_grid[i][j] = updated\_value

else:

max\_diff = max(max\_diff, abs(updated\_value - grid[i][j]))

grid[i][j] = updated\_value

max\_diff = max(max\_diff, abs(updated\_value - grid[i][j]))

grid = new\_grid if method == 'simple' else grid

if max\_diff < epsilon:

break

method\_name = "Простая итерация" if method == 'simple' else "Зейделя"

visualize\_solution(grid, step, method\_name)

# Вызов методов решения

solve\_with\_iterations(step=2.0, method='simple')

solve\_with\_iterations(step=1.0, method='simple')

solve\_with\_iterations(step=2.0, method='zeidel')

solve\_with\_iterations(step=1.0, method='zeidel')