Министерство образования Республики Беларусь

Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Факультет компьютерных систем и сетей

Кафедра информатики

Лабораторная работа №15

«Метод сеток решения задачи Дирихле для уравнения Пуассона»

Выполнил:

студент гр. 953506

Кондрашов И.Д.

Проверил:

Анисимов В.Я.

Минск 2021

# Цель работы

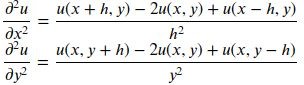
* изучить метод разностных аппроксимаций для уравнения Пуассона;
* составить алгоритмы решения задачи Дирихле для уравнения Пуассона методом сеток применимыми для организации вычислений на ПЭВМ;
* составить программы решения задачи Дирихле для уравнения Пуассона по разработанным алгоритмам;
* выполнить тестовые примеры и проверить правильность работы программ;
* получить численное решение заданной задачи Дирихле для уравнения Пуассона.

## Задание 1.

Требуется следующую промоделировать следующий процесс: пластина прямоугольной формы с вырезом на одной из сторон жестко закреплена по краям и равномерно нагружена по площади, прогиб пластины определяется из уравнения Пуассона. При этом рассчитывается прогиб как функция 𝑊(𝑥,𝑦), по данным из варианта: 𝐴, 𝐵 – размеры пластины; ℎ − ее толщина; 𝑅 – радиус выреза; 𝑃 – нагрузка; Е − модуль упругости; ν – коэффициент Пуассона, граничное условие 𝑊 = 0. Дифференциальное уравнение Пуассона имеет вид:

где ***D*** *= Eh3/(12(1 - v2))*- изгибная жесткость, 𝐸 – модуль упругости, ℎ − толщина пластины, ν – коэффициент Пуассона.

Для решения данного дифференциального уравнения аппроксимируем вторые производные как:



Тогда, для каждого внутреннего узла составим разностную схему вида:

Упростим выражение:



Table

Description automatically generated

Для остальных (граничных и внешних точек) значение 𝑊𝑖,𝑗 = 0. Данную разностную схему можно решать как систему линейных уравнений. При этом решение разностной схемы необходимо представить в виде вектора.

Для более эффективной с точки зрения скорости вычисления и памяти реализации алгоритма необходимо использовать разреженные матрицы (так как на одной строке матрицы ненулевыми будут только 5 значений) и решать такую систему можно при помощи встроенных эффективных алгоритмов для разреженных матриц. В данном случае используется метод сопряженных градиентов.

Text

Description automatically generated

Text

Description automatically generated

Визуализация решения

Вычислим решение разностной схемы с шагом ℎ = 4.

A screenshot of a computer

Description automatically generated with low confidence

Chart, surface chart

Description automatically generated

Вычислим решение с шагом h = 8.

A screenshot of a computer

Description automatically generated with medium confidence

Chart, surface chart

Description automatically generated

Вычислим решение с шагом равным h = 16.

A screenshot of a computer

Description automatically generated with medium confidence

Chart, surface chart

Description automatically generated

**Полный листинг кода**

import numpy as np  
import matplotlib.pyplot as plt  
import typing as tp  
import scipy.sparse.linalg as spa\_linalg  
  
from math import \*  
  
  
VARIANT\_NUMBER = 10  
width = 200  
height = 140  
radius = 40  
thickness = 2  
P = 70 \* 10\*\*9  
E = 60  
v = 0.28  
function = lambda x, y: P / (E \* thickness\*\*3 / (12 \* (1 - v\*\*2)))  
  
  
def solve(width: float, height: float, radius: float,  
 function: tp.Callable[[float, float], float], step: float) -> np.array:  
 cols\_amount = int(ceil(width / step)) + 1  
 rows\_amount = int(ceil(height / step)) + 1  
  
 total\_equations\_amount = cols\_amount \* rows\_amount  
 A = np.zeros((total\_equations\_amount, total\_equations\_amount))  
  
 b = np.zeros(total\_equations\_amount)  
  
 def get\_mapped\_index(i: int, j: int) -> int:  
 return cols\_amount \* i + j  
  
 def is\_bound\_point(i: int, j: int) -> bool:  
 x, y = i \* step, j \* step  
 if np.isclose(x, 0) or np.isclose(y, 0):  
 return True  
 elif x >= height or y >= width:  
 return True  
 elif x \*\* 2 + (y - width / 2) \*\* 2 <= radius \*\* 2:  
 return True  
 else:  
 return False  
  
 for i in range(rows\_amount):  
 for j in range(cols\_amount):  
 if not is\_bound\_point(i, j):  
 A[get\_mapped\_index(i, j), get\_mapped\_index(i, j - 1)] = 1  
 A[get\_mapped\_index(i, j), get\_mapped\_index(i, j + 1)] = 1  
 A[get\_mapped\_index(i, j), get\_mapped\_index(i - 1, j)] = 1  
 A[get\_mapped\_index(i, j), get\_mapped\_index(i + 1, j)] = 1  
 A[get\_mapped\_index(i, j), get\_mapped\_index(i, j)] = -4  
  
 b[get\_mapped\_index(i, j)] = function(i \* step, j \* step) \* step \*\* 2  
 else:  
 A[get\_mapped\_index(i, j), get\_mapped\_index(i, j)] = 1  
  
 solution = spa\_linalg.cg(A, b)[0]  
 return solution.reshape((rows\_amount, cols\_amount))  
  
  
def plot\_solution(solution: np.array, width: float, height: float, step: float):  
 y = np.arange(0, height + step, step)  
 x = np.arange(0, width + step, step)  
 X, Y = np.meshgrid(x, y)  
  
 fig = plt.figure(figsize=(10, 10))  
 ax = plt.axes(projection='3d')  
 ax.plot\_surface(X, Y, solution, cmap='rainbow')  
 plt.show()  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
  
 solution = solve(width, height, radius, function, 4)  
 plot\_solution(solution, width, height, 4)  
  
 solution = solve(width, height, radius, function, 8)  
 plot\_solution(solution, width, height, 8)  
  
 solution = solve(width, height, radius, function, 16)  
 plot\_solution(solution, width, height, 16)

## Вывод.

В ходе лабораторной работе я разработал функцию для решения задачи Дирихле для уравнения Лапласа и проверил его на примере реального физического процесса, также я получил визуальное подтверждение данной гипотезы в виде графиков.