

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ     
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего профессионального образования

**ДАЛЬНЕВОСТОЧНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ**

**Институт математики и компьютерных технологий**

**Департамент математического и компьютерного моделирования**

**ОТЧЕТ**

**по лабораторной работе** **№ 3**

«Метод квадратного корня»

Выполнил: студент гр. Б9120-02.03.01сцт

Пограничный К. О.

Проверил: преподаватель

Кузнецов К. С.

Владивосток 2023

**Постановка задания**

Дана система линейных алгебраических уравнений в виде Ax=b.

Решить следующие две системы с помощью метода квадратного корня:

Изображение выглядит как Шрифт, текст, рукописный текст, белый

Автоматически созданное описание

Изображение выглядит как текст, Шрифт, белый, типография

Автоматически созданное описание

**Метод решения**

Задача была выполнена на языке программирования Python с использованием средств библиотеки NumPy.

Матрицы a\_1 и a\_2 были созданы из начальных данных, а именно из коэффициентов, стоящих при (x1, …, x3) для первой матрицы и при (x1, …, x4) для второй матрицы соответственно. Векторы b\_1 и b\_2, по аналогии, были созданы из свободных членов соответствующих СЛАУ.

Был реализован алгоритм, решающий данные СЛАУ методом квадратного корня.

**Алгоритм решения**

Он заключается в представлении матрицы A в виде .

1. В первую очередь необходимо создать матрицу U и найти каждый из ее элементов, опираясь на следующие формулы:
2. Далее нужно записать транспонированную копию найденной матрицы:
3. После вычисления матриц U и необходимо решить СЛАУ в два этапа:

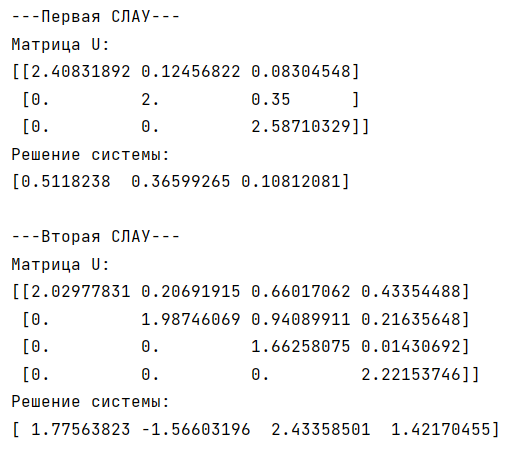
**Использованные функции**

1. numpy.array() – встроенная функция библиотеки NumPy для создания списков(массивов).
2. numpy.zeros() – встроенная функция библиотеки NumPy, которая создает матрицу из нулей указанной формы и типа данных.
3. numpy.sqrt() – встроенная функция библиотеки NumPy, которая находит квадратный корень аргумента.
4. numpy.linalg.tensorsolve() - используется для разрешения тензорного уравнения Ax=b относительно x.

**Подробный анализ кода**

1. Создание матриц a\_1 и a\_2, а также двух векторов b\_1 и b\_2, производилось в главной функции при помощи метода numpy.array().
2. С помощью функции “\_\_init\_\_()” класса “Solution” находим верхнюю треугольную матрицу U, опираясь на уравнения, описанные в постановке задачи.
3. Функция “display\_u\_matrix()” класса “Solution” выводит матрицу U в терминале.
4. Функция “get\_solution()” класса “Solution” разрешает уравнения:
5. Функция “display\_solution()” класса “Solution” соответственно выводит результат решения СЛАУ в терминале.

**Результаты тестов**



**Вывод**

Был реализован метод квадратного корня, позволяющий представить квадратную матрицу в виде произведения , которое в свою очередь позволяет решить СЛАУ, где число неизвестных переменных равно количеству уравнений в системе, имеющую вектор свободных коэффициентов.

**Приложение**

Листинг программы:

import numpy as np  
  
  
class Solution:  
 def \_\_init\_\_(self, a: np.array, b: np.array):  
 self.dim = a.shape[0]  
 self.b = b  
  
 self.u = np.zeros(a.shape, dtype=float)  
  
 self.u[0][0] = np.sqrt(a[0][0])  
  
 for i in range(0, self.dim):  
 for j in range(i):  
 if j > 0:  
 self.u[0][j] = a[0][j] / self.u[0][0]  
 sum = 0  
 for k in range(i - 1):  
 sum += pow(self.u[k][i], 2)  
  
 self.u[i][i] = np.sqrt(a[i][i] - sum)  
 for j in range(i + 1, self.dim):  
 if i < j:  
 sum = 0  
 for k in range(i - 1):  
 sum += self.u[k][i] \* self.u[k][j]  
  
 self.u[i][j] = ((a[i][j] - sum) / self.u[i][i])  
  
 def get\_solution(self):  
 self.y = np.linalg.tensorsolve(self.u.T, self.b)  
 return np.linalg.tensorsolve(self.u, self.y)  
  
 def display\_u\_matrix(self):  
 print(f"Матрица U:\n{self.u}")  
  
 def display\_solution(self):  
 print(f"Решение системы:\n{self.get\_solution()}\n")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 a\_1 = np.array([  
 [5.8, 0.3, 0.2],  
 [0.3, 4, 0.7],  
 [0.2, 0.7, 6.7]  
 ])  
 b\_1 = np.array([3.1, 1.7, 1.1])  
  
 print("---Первая СЛАУ---")  
 sol\_1 = Solution(a\_1, b\_1)  
 sol\_1.display\_u\_matrix()  
 sol\_1.display\_solution()  
  
 a\_2 = np.array([  
 [4.12, 0.42, 1.34, 0.88],  
 [0.42, 3.95, 1.87, 0.43],  
 [1.34, 0.87, 3.2, 0.31],  
 [0.88, 0.43, 0.31, 5.17]  
 ])  
 b\_2 = np.array([11.17, 0.115, 9.909, 9.349])  
  
 print("---Вторая СЛАУ---")  
 sol\_2 = Solution(a\_2, b\_2)  
 sol\_2.display\_u\_matrix()  
 sol\_2.display\_solution()