Санкт-Петербургский Национальный Исследовательский Университет Информационных Технологий, Механики и Оптики

Факультет инфокоммуникационных технологий

**Лабораторная работа №1**

Выполнил:

Бабаев Р.С.

Проверил

Мусаев А.А.

Санкт-Петербург,

2022

**Оглавление**

[**Введение** 3](#_Toc117118366)

[**1** **Решение задач** 4](#_Toc117118367)

[**1.1 Программа «FizzBuzz»** 4](#_Toc117118368)

[**1.2 Транспонирование матрицы** 4](#_Toc117118369)

[**1.3 Определение ранга матрицы** 5](#_Toc117118370)

[**1.4 Умножение матриц** 7](#_Toc117118371)

[**1.5 Пример работы программы №2** 7](#_Toc117118372)

[**1.6 Решение задачи с использованием библиотеки numpy** 9](#_Toc117118373)

[**1.7 Нахождение обратной матрицы 3х3** 9](#_Toc117118374)

[**1.8 Пример работы программы №3** 11](#_Toc117118375)

[**Заключение** 12](#_Toc117118376)

[**Список литературы** 13](#_Toc117118377)

# **Введение**

Данная работа представляет собой отчёт о выполненных заданиях:

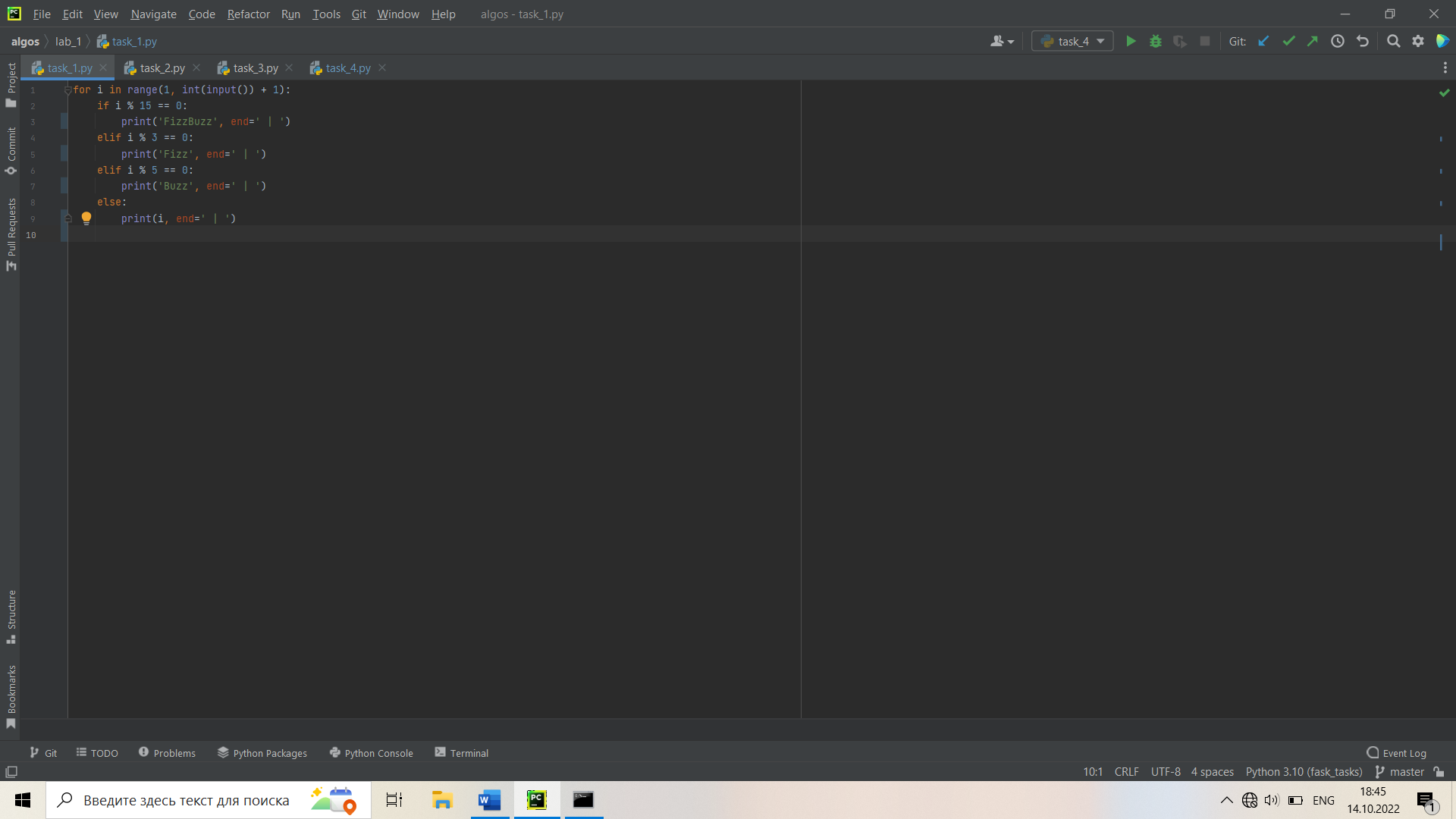
1. Написать программу «FizzBuzz»;
2. Написать программу для транспонирования, определения ранга матрицы и умножения матриц без использования готовых библиотек;
3. Выполнить второе задание с использованием библиотеки numpy;
4. Написать программу для возведения матрицы 3x3 в степень -1 также без и с numpy, а также сравнить время выполнения своей функции с аналогом из numpy;
5. Проанализировать достоинства и недостатки numpy

# **Решение задач**

## **1.1 Программа «FizzBuzz»**

На рисунке 1.1 представлено решение первой задачи. «На вход» подается число и с помощью цикла for и блока if-elif-else для проверки на кратность последовательно (от единицы до введенного номера) выводится соответствующее слово или число.

Пример работы программы представлен на рисунке 1.2.



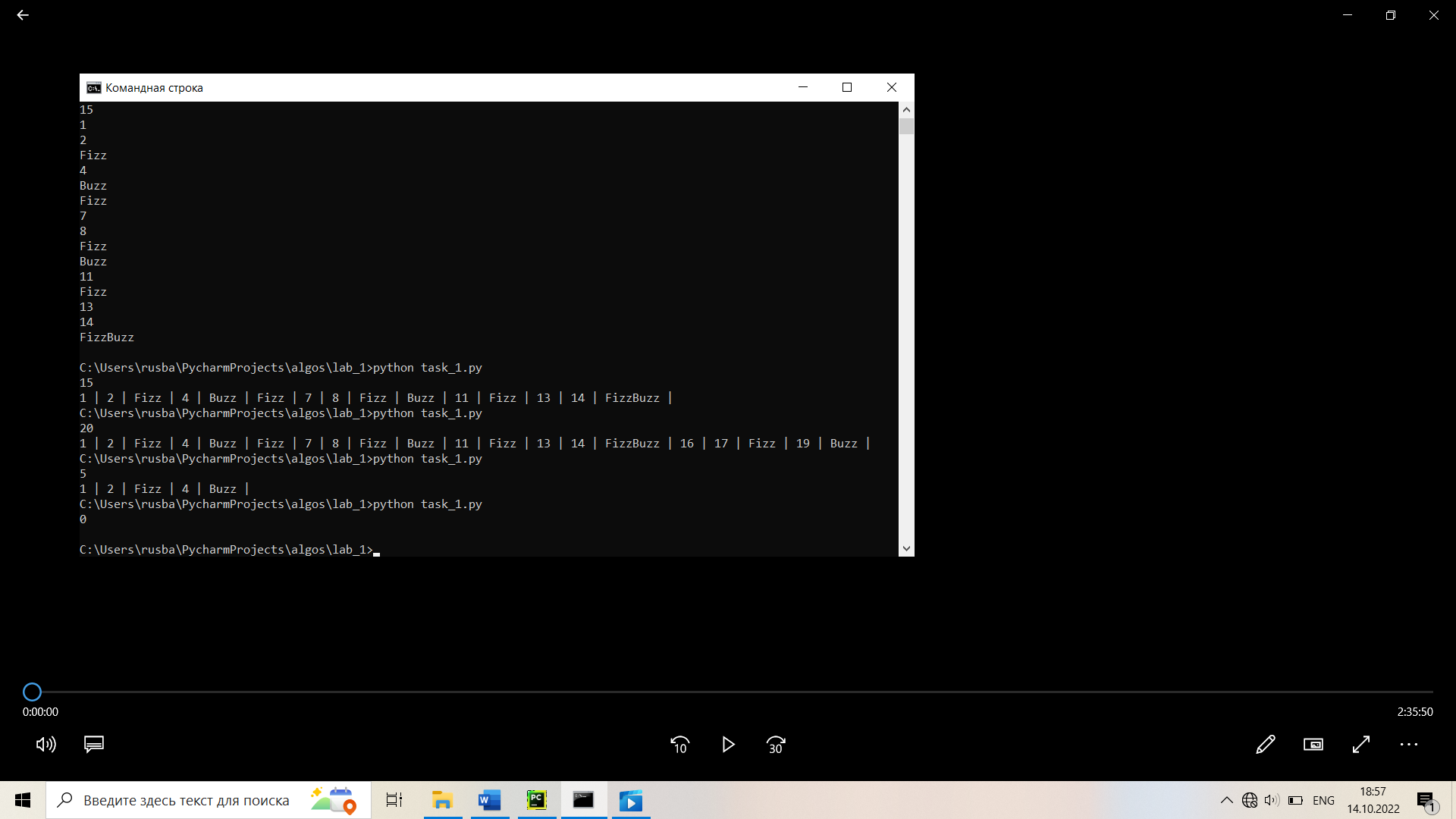
Рисунок 1.1 – Решение задачи «FizzBuzz»

Рисунок 1.2 – Пример работы программы №1.

## **1.2 Транспонирование матрицы**

При транспонировании исходной матрицы создается новая, в которой строки равны столбцам исходной матрицы, а столбцы – строкам. Алгоритм транспонирования реализован в функции transpose(A), где A – исходная матрица (см. рисунок 1.3): создаётся пустой list t\_A, который с помощью цикла for последовательно заполняется строками из i-тых элементов всех строк исходной матрицы, где i – номер столбца. Данная функция возвращает заполненный список t\_A.

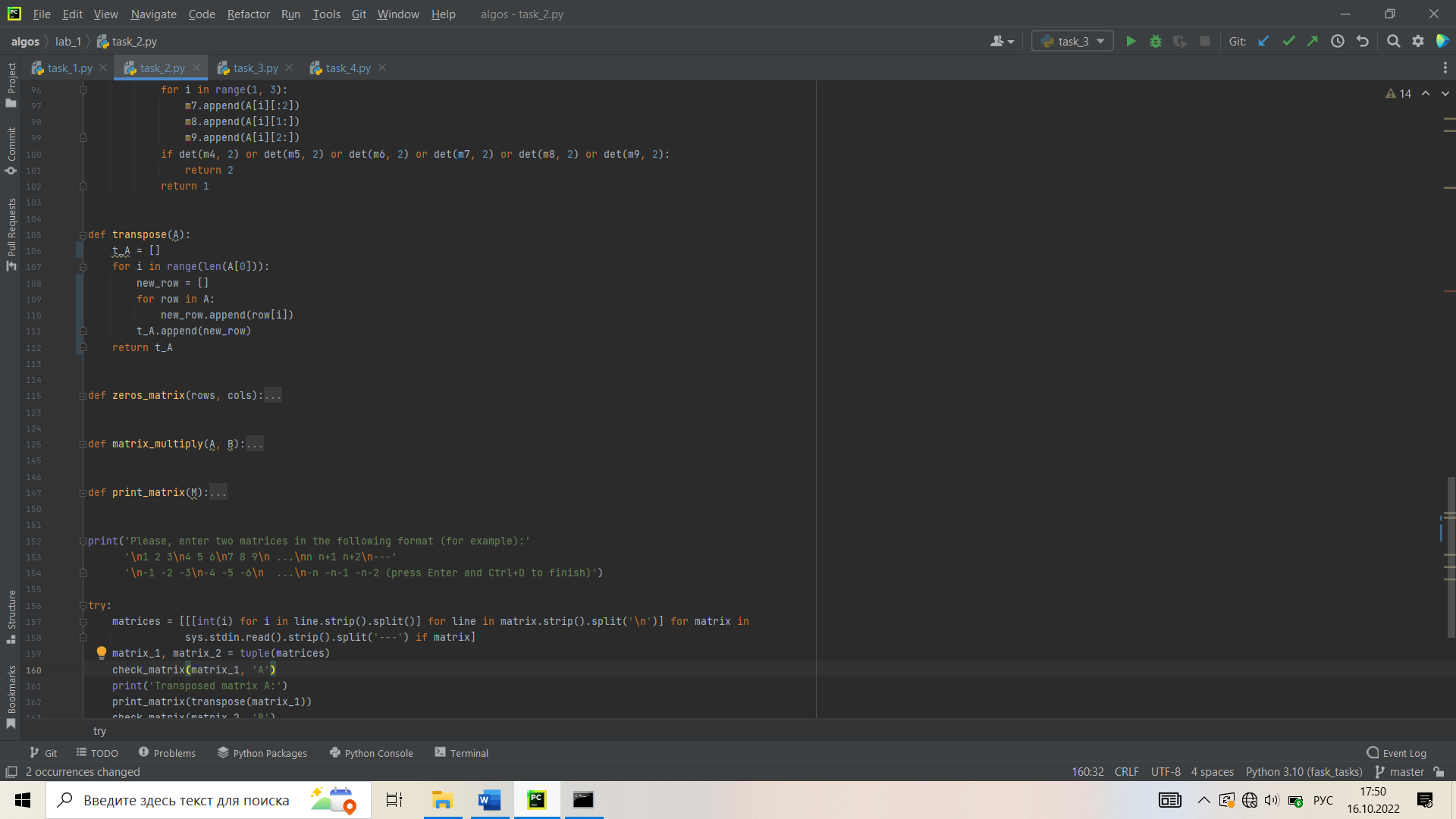


Рисунок 1.3 – Функция transpose(A)

## **1.3 Определение ранга матрицы**

Рангом исходной матрицы называется число, равное максимальному порядку минора данной матрицы с ненулевым определителем (если используется метод окаймляющих миноров). Это также значит, что ранг не может быть больше порядка (). В решении представлена возможность вычислить ранг матрицы с порядком не больше 3. Для этого реализованы функция rank(A, max\_rank), где A – исходная матрица и max\_rank – максимально допустимое значение ранга, а также функция det(A, size), которая возвращает определитель данной матрицы А с указанным порядком size (см. рисунок 1.4). Таким образом, в функции rank(A, max\_rank) последовательно проверяются все миноры и вычисляются их определители: сначала проверяется, не нулевая ли это матрица, затем вычисляется её определитель и только потом проверяются все остальные миноры (см. рисунок 1.5).

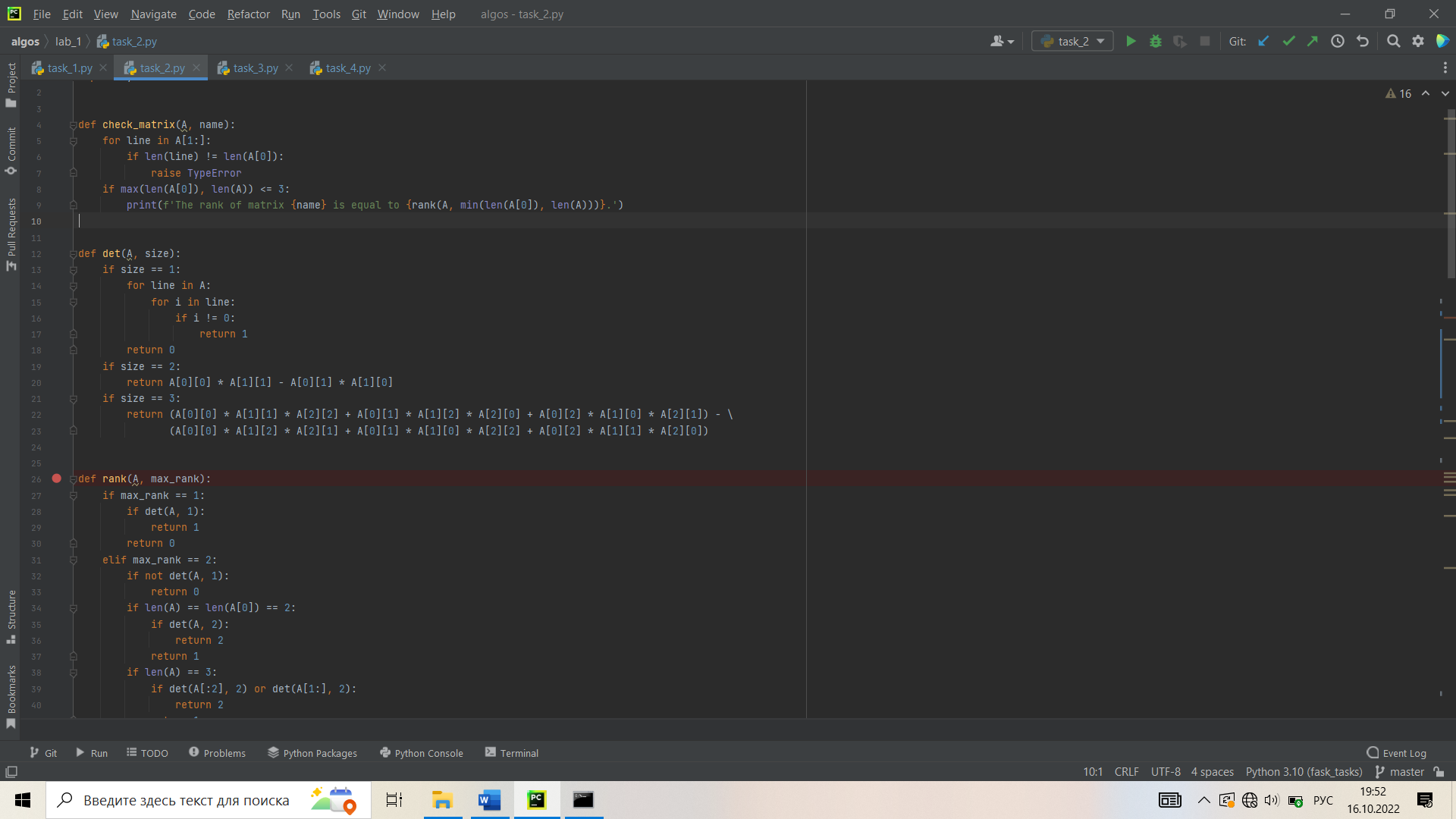


Рисунок 1.4 – Функция det(A, size)

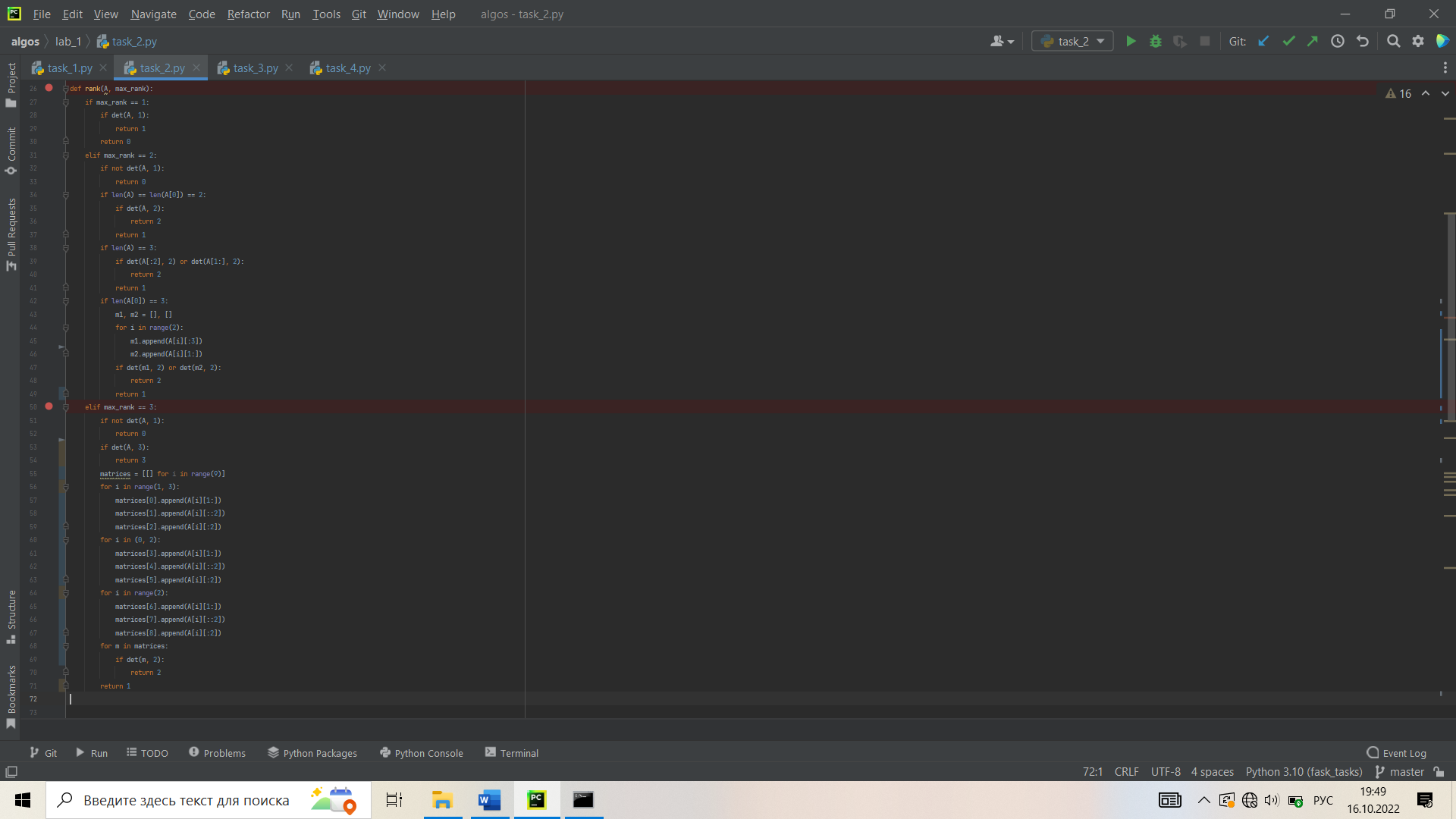


Рисунок 1.5 – Функция rank(A, max\_rank)

## **1.4 Умножение матриц**

При умножении двух матриц всегда получается квадратная, при этом матрицы – некоммутативны, а это значит, что порядок полученной матрицы равен количеству столбцов в матрице слева (если мы умножаем слева направо). Каждый элемент i-строки j-столбца полученной матрицы равен сумме произведений элементов i-строки k-столбца первой матрицы и k-строки j-столбца второй. Из этого также следует, что матрицы можно перемножить только и только тогда, когда количество столбцов первой матрицы равно количеству строк второй матрицы. В решении для умножения матриц реализована функция matrix\_multiply(A, B), где A и B – первая и вторая матрица соответственно (см. рисунок 1.6). Так, сначала создаётся нулевая матрица нужного порядка, которая затем заполняется соответствующими суммами.

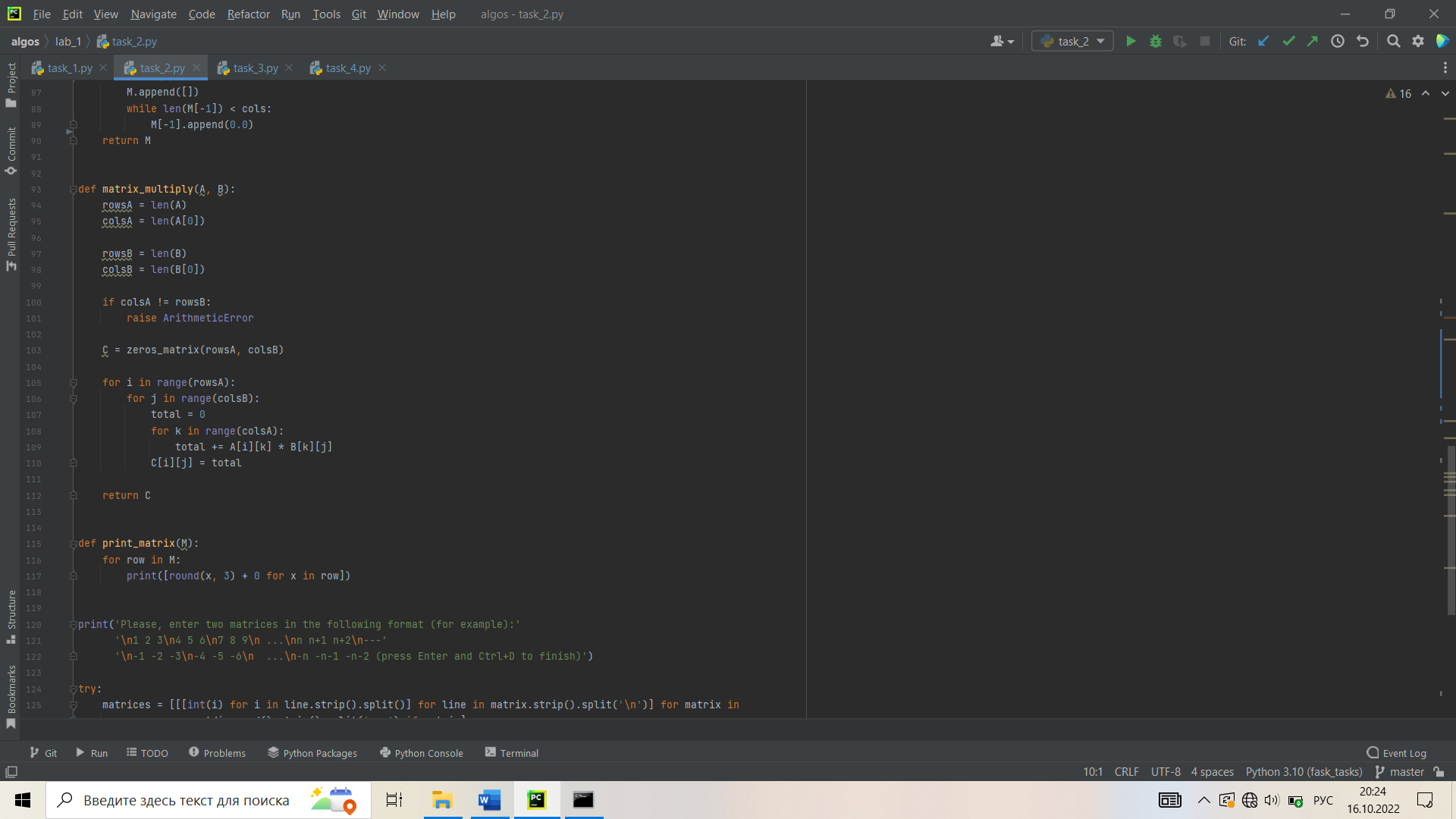


Рисунок 1.6 – Функция matrix\_multiply(A, B)

## **1.5 Пример работы программы №2**

Таким образом, в программе были реализованы все необходимые функции: транспонирование, определение ранга матрицы и умножение матриц. Сама же программа взаимодействует с пользователем следующим образом:

1. Приветствует его и объясняет на примере, как нужно подавать «на вход» матрицы. Ввод матрицы реализован с помощью потокового ввода и библиотеки sys: сначала пользователь построчно через пробел вводит каждый элемент первой матрицы, затем на следующей строки печатает три тире «---» и тем же самым образом вводит элементы второй матрицы. Для того чтобы завершить ввод, необходимо перейти на новую строку и нажать клавиши Ctrl+D. Если матрица введена некорректно или данная матрица не существует, то программа выдаст ошибку.
2. После того как программа приняла матрицы, она печатает их ранг (если в матрице не больше трёх строк и столбцов), транспонирует каждую и выводит в транспонированном виде, а также перемножает с обеих сторон и также выдает результат.

Пример работы данной программы представлен на рисунке 1.7.

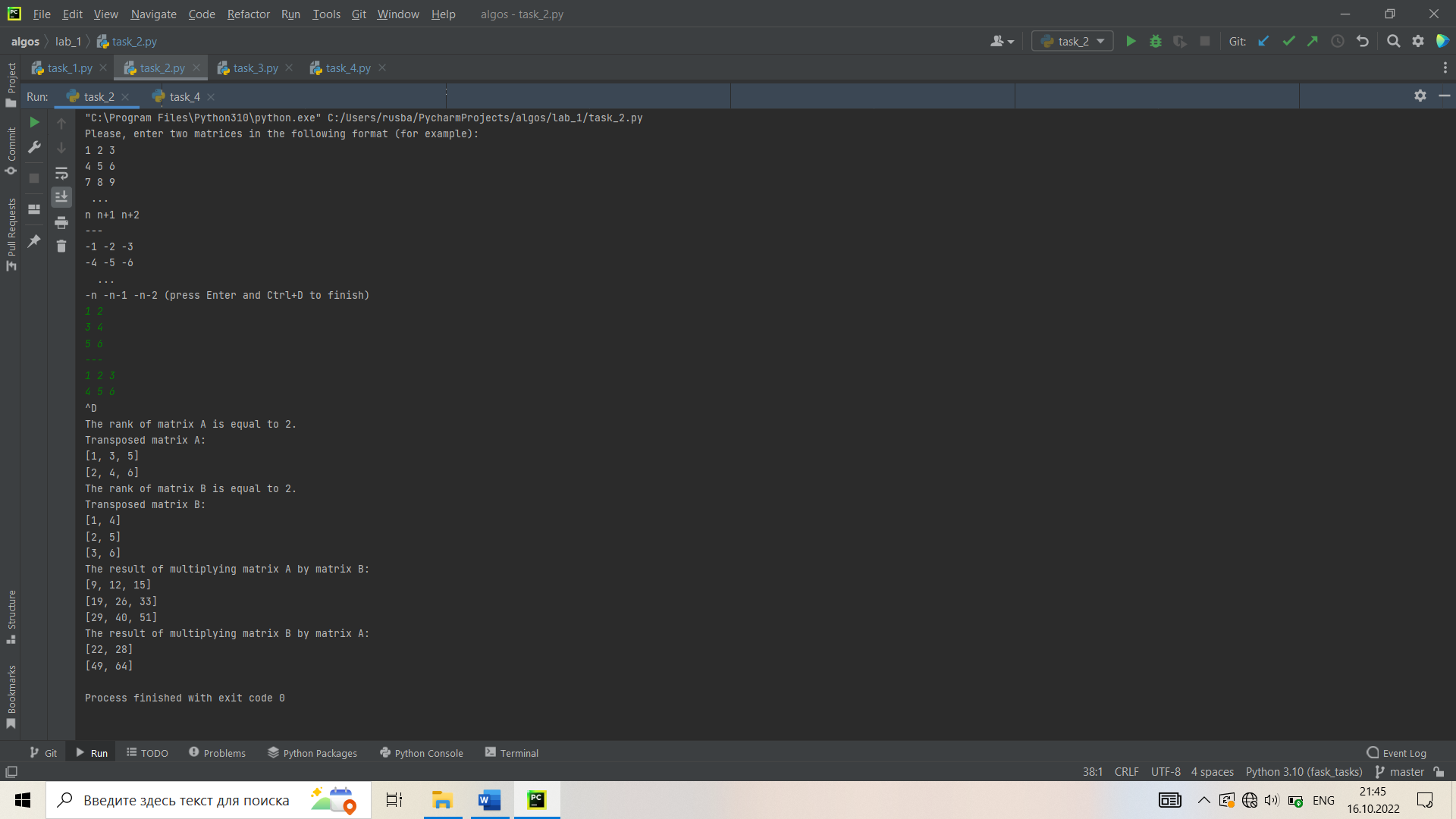


Рисунок 1.7 – Пример работы программы №2

## **1.6 Решение задачи с использованием библиотеки numpy**

Для решения задачи с использованием библиотеки numpy была написанf ровно такая же программа с одним исключением: вместо функций, описанных выше, использовались готовые функции из библиотеки numpy:

* numpy.transpose(A) – для транспонирования матрицы;
* numpy.dot(A, B) – для умножения матриц;
* numpy.linalg.matrix\_rank(A) – для определения ранга матрицы.

Использование данных функций значительно упрощает и ускоряет процесс разработки, при этом возможностей для работы с матрицами в библиотеке значительно больше. Более того, в numpy есть специальный тип данных matrix. Так как numpy использует библиотеки, написанные на C++ и хорошо оптимизирован, он крайне эффективен (по объему занимаемой памяти) и быстр. Однако ввиду различия типов данных с «обычным» Python перевод сущностей numpy в сущности Python и наоборот требует больших затрат.

## **1.7 Нахождение обратной матрицы 3х3**

Обратной матрицей называют такую матрицу, которая при умножении на исходную дает единичную. Обратная матрица существует в том случае, если исходная является квадратной и невырожденной (), и притом только одна. В приведенном решении обратная матрица находится с помощью метода элементарных преобразований: он заключается в том, что если преобразовать исходную матрицу в единичную, а затем повторить все те же действия над единичной матрицей, то в результате из единичной можно получить матрицу, обратную исходной.

Итак, для начала необходимо убедиться, что данная матрица является квадратной и невырожденной. Для этого были реализованы функции check\_squareness(A) (см. рисунок 1.8) и check\_non\_singular(A) (см. рисунок 1.9) соответственно.

Изображение выглядит как текст, снимок экрана, монитор, электроника

Автоматически созданное описание

Рисунок 1.8 – Функция check\_squareness(A)

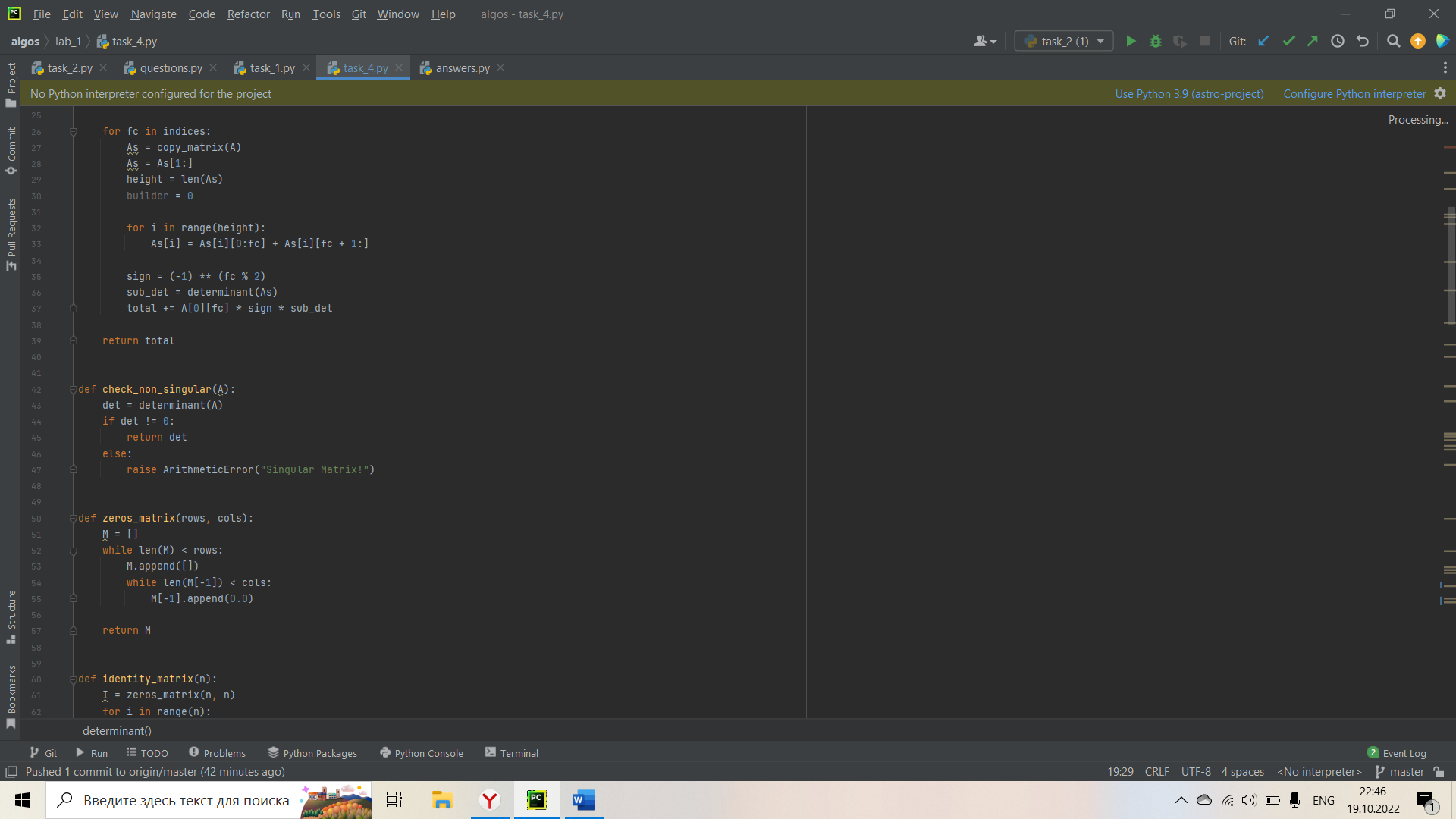


Рисунок 1.9 – Функция check\_non\_singular(A)

Теперь можно находить обратную матрицу. Алгоритм поиска описан в функции invert\_matrix(A, tol=None), где tol – количество знаков после запятой (см. рисунок 1.10). Для удобства непосредственно перед самими преобразованиями создаются копии для исходной и единичной матрицы, а также список индексов. Далее программа начинает проходить по исходной матрице и умножает первую строку на обратное числа, находящегося на главной диагонали (fdScaler). Затем она проходит по остальным строкам, вычитая из каждой первую, умноженную на первый элемент в соответствующей строке, и так далее. Таким образом, исходная матрица становится единичной, а единичная – обратной!

Также «для подстраховки» в конце с помощью функции check\_matrix\_equality(A, B, tol=None) осуществляется проверка, что произведение полученной матрицы на исходную действительно дает единичную (см. рисунок 1.11).

Изображение выглядит как текст, снимок экрана, монитор, электроника

Автоматически созданное описание

Рисунок 1.10 – Функция invert\_matrix(A, tol=None)

Изображение выглядит как текст, снимок экрана, монитор, электроника

Автоматически созданное описание

Рисунок 1.11 – Функция check\_matrix\_equality(A, B, tol=None)

В библиотеке numpy же для нахождения обратной матрицы есть функция numpy.linalg.inv(A).

## **1.8 Пример работы программы №3**

С точки зрения ввода матрицы, программа №3 реализована так же, как и программа №2, за тем исключением, что матрица обязательно должна иметь размер 3х3. Помимо непосредственно нахождения обратной матрицы, также было необходимо вычислить время выполнения собственной функции и ее аналога из библиотеки numpy – для этого был использован модуль timeit.

Примеры работы программы №3 представлены на рисунках 1.12 и 1.13.

Изображение выглядит как текст, снимок экрана, монитор, электроника

Автоматически созданное описание

Рисунок 1.12 – Пример работы программы №3 (1)

Изображение выглядит как текст, снимок экрана, монитор, электроника

Автоматически созданное описание

Рисунок 1.13 – Пример работы программы №3 (2)

Как можно заметить, версия из библиотеки numpy в среднем выполняет работу быстрее.

# **Заключение**

Поставленные задачи успешно выполнены. Написаны программы с использованием функций для транспонирования, умножения, нахождения ранга матриц и возведения их в -1 степень, реализованные вручную и взятые из библиотеки numpy, приведены примеры работы данных программ. Также были проанализированы преимущества и недостатки numpy.

Код каждой программы можно найти в Github [2].

# **Список литературы**

1. StackOverFlow [Электронный ресурс] – <https://stackoverflow.com/> (Дата обращения – 06.10.2022);
2. Github [Электронный ресурс] – https://github.com/HeraldOfWar/algos (Дата обращения – 06.10.2022).