VIETNAM GENERAL CONFEDERATION OF LABOR

**TON DUC THANG UNIVERSITY**

**FACULTY OF INFORMATION TECHNOLOGY**



**NGO CHI THUAN – 523H0102**

**MID-TERM ESSAY**

**APPLIED LINEAR ALGEBRA FOR INFORMATION TECHNOLOGY**

**SOFTWARE ENGINEERING**

**HO CHI MINH CITY, 2024**

VIETNAM GENERAL CONFEDERATION OF LABOR

**TON DUC THANG UNIVERSITY**

**FACULTY OF INFORMATION TECHNOLOGY**



**NGO CHI THUAN – 523H0102**

**MID-TERM ESSAY**

**APPLIED LINEAR ALGEBRA FOR INFORMATION TECHNOLOGY**

**SOFTWARE ENGINEERING**

Advised by

**MS. Nguyen Van Khoa**

**HO CHI MINH CITY, 2024**

**ACKNOWLEDGEMENT**

I would like to express my sincere gratitude to MS. Nguyen Van Khoa, my instructor and mentor, for his valuable guidance and support throughout my mid-term essay. He has been very patient in providing me with constructive feedback and suggestions to show me new way to solve a problem. He has also encouraged me to explore new technologies, show me a lot different way to solve a problem and techniques to enhance my essay. I have learned a lot from his expertise. I am honored and privileged to have him as my teacher and supervisor.

*Ho Chi Minh city, 26th March 2024.*

*Author*

*(Signature and full name)*

***Thuan***

Ngo Chi Thuan

**DECLARATION OF AUTHORSHIP**

We hereby declare that this is our own project and is guided by MS. Nguyen Van Khoa; The content research and results contained herein are central and have not been published in any form before. The data in the tables for analysis, comments and evaluation are collected by the main author from different sources, which are clearly stated in the reference section.

In addition, the project also uses some comments, assessments as well as data of other authors, other organizations with citations and annotated sources.

**If something wrong happens, we’ll take full responsibility for the content of my project.** Ton Duc Thang University is not related to the infringing rights, the copyrights that We give during the implementation process (if any).

*Ho Chi Minh city, 26th March 2024.*

*Author*

*(Signature and full name)*

***Thuan***

Ngo Chi Thuan

**MIDTERM ESSAY**

**APPLIED LINEAR ALGEBRA FOR INFORMATION TECHNOLOGY**

**ABSTRACT**

This essay focus on solving eigth questions in task 1 of mid-term essay Applied Linear Algebra for Infomation Technology (a, b, c, d, e, f, g and h) about matrix operations and provide detailed solutions to those questions. From there, understand more about how to handle matrices in Python.

**TABLE OF CONTENT**

[LIST OF FIGURES vii](#_Toc163205464)

[CHAPTER 1. METHODOLOGY OF SOLVING TASKS 1](#_Toc163205465)

[1.1 Task 1d: Save odd integer numbers in the matrix A into a new vector, and print the resultant vector to the screen. 1](#_Toc163205466)

[1.2 Task 1e: Save prime numbers in the matrix A into a new vector, and print the resultant vector to the screen. 1](#_Toc163205467)

[1.3 Task 1f: Given a matrix *D = CB*, reverse elements in the odd rows of the matrix D, and print the resultant matrix to the screen. 2](#_Toc163205468)

[1.4 Task 1g: Regarding the matrix A, find the rows which have maximum count of prime numbers, and print the rows to the screen. 2](#_Toc163205469)

[1.5 Task 1h: Regarding the matrix A, find the rows which have the longest contiguous odd numbers sequence, and print the rows to the screen. 2](#_Toc163205470)

[CHAPTER 2. SOURCE CODES AND OUTPUTS 3](#_Toc163205471)

[2.1 Import Python libraries 3](#_Toc163205472)

[2.1.1 Source code 3](#_Toc163205473)

[2.1.2 Output 4](#_Toc163205474)

[2.2 Task 1a: Calculate A + A^T + CB + (B^T)( C^T), and print the results 4](#_Toc163205475)

[2.2.1 Source code 4](#_Toc163205476)

[2.2.2 Output 5](#_Toc163205477)

[2.3 Task 1b: Calculate A/10 + (A/11)^2 + … + (A/19)^10, and print the results to the screen 5](#_Toc163205478)

[2.3.1 Source code 5](#_Toc163205479)

[2.3.2 Output 6](#_Toc163205480)

[2.4 Task 1c: Save odd rows of the matrix A into a new matrix, and print the resultant matrix to the screen 8](#_Toc163205481)

[2.4.1 Source code 8](#_Toc163205482)

[2.4.2 Output 8](#_Toc163205483)

[2.5 Task 1d: Save odd integer numbers in the matrix A into a new vector, and print the resultant vector to the screen 9](#_Toc163205484)

[2.5.1 Source code 9](#_Toc163205485)

[2.5.2 Output 10](#_Toc163205486)

[2.6 Task 1e: Save prime numbers in the matrix A into a new vector, and print the resultant vector to the screen 10](#_Toc163205487)

[2.6.1 Source code 10](#_Toc163205488)

[2.6.2 Output 11](#_Toc163205489)

[2.7 Task 1f: Given a matrix *D = CB*, reverse elements in the odd rows of the matrix D, and print the resultant matrix to the screen 12](#_Toc163205490)

[2.7.1 Source code 12](#_Toc163205491)

[2.7.2 Output 13](#_Toc163205492)

[2.8 Task 1g: Regarding the matrix A, find the rows which have maximum count of prime numbers, and print the rows to the screen 14](#_Toc163205493)

[2.8.1 Source code 14](#_Toc163205494)

[2.8.2 Output 15](#_Toc163205495)

[2.9 Task 1h: Regarding the matrix A, find the rows which have the longest contiguous odd numbers sequence, and print the rows to the screen 15](#_Toc163205496)

[2.9.1 Source code 15](#_Toc163205497)

[2.9.2 Output 17](#_Toc163205498)

[CHAPTER 3. CONCLUSION 18](#_Toc163205499)

[REFERENCES 19](#_Toc163205500)

**LIST OF FIGURES**

[Figure 2.1.1.1: Import numpy library 3](#_Toc162811355)

[Figure 2.1.1.2: Initialize matrices required and print to screen 3](#_Toc162811356)

[Figure 2.1.2: Output after import Python libraries and intialize matrices 4](#_Toc162811357)

[Figure 2.2.1: Source code of task 1a 5](#_Toc162811358)

[Figure 2.2.2: Output of task 1a 5](#_Toc162811359)

[Figure 2.3.1: Source code of task 1b 6](#_Toc162811360)

[Figure 2.3.2: Output of task 1b 7](#_Toc162811361)

[Figure 2.4.1: Source code of task 1c 8](#_Toc162811362)

[Figure 2.4.2: Output of task 1c 8](#_Toc162811363)

[Figure 2.5.1: Source code of task 1d 9](#_Toc162811364)

[Figure 2.5.2: Output of task 1d 10](#_Toc162811365)

[Figure 2.6.1: Source code of task 1e 11](#_Toc162811366)

[Figure 2.6.2: Output of task 1e 12](#_Toc162811367)

[Figure 2.7.1: Source code of task 1f 12](#_Toc162811368)

[Figure 2.7.2: Output of task 1f 13](#_Toc162811369)

[Figure 2.8.1: Source code of task 1g 14](#_Toc162811370)

[Figure 2.8.2: Output of task 1g 15](#_Toc162811371)

[Figure 2.9.1: Source code 1h 16](#_Toc162811372)

[Figure 2.9.2: Output of task 1h 17](#_Toc162811373)

# METHODOLOGY OF SOLVING TASKS

## Task 1d: Save odd integer numbers in the matrix A into a new vector, and print the resultant vector to the screen.

* Initialize a variable to count the number of odd integer numbers in matrix A
* Using two loops to count the quantity of odd numbers in matrix A
* Initialize an empty vector to store all odd numbers which have been counted
* Using two loops to add all the odd numbers into the empty vector
* Print the result.

## Task 1e: Save prime numbers in the matrix A into a new vector, and print the resultant vector to the screen.

* Define a function to check a number is prime or not
  + If that number less or equal one, that’s not a prime number
  + If there exists a value between two and one-half of that number that can be divided by that number, that’s not a prime number. If it doesn’t exist a number can be divided by that number, that’s a prime number.
* Initialize a variable to count the number of prime numbers in the matrix
* Using two loops and the function which created to check prime number to count prime numbers in matrix
* Initialize an empty vector to store all prime numbers
* Using two loops and the last function to add prime numbers into the new empty vector
* Print the result.

## Task 1f: Given a matrix *D = CB*, reverse elements in the odd rows of the matrix D, and print the resultant matrix to the screen.

* Create matrix D by mutiply B and C matrices
* Create a copy matrix of matrix D
* Define a function to reverse the input row
* Using a loop to reverse odd rows
* Print the result.

## Task 1g: Regarding the matrix A, find the rows which have maximum count of prime numbers, and print the rows to the screen.

* Create a variable to count the number of prime numbers
* Initialize an empty array has the same size with the number of rows of matrix A
* Define a function to count the quality of prime numbers in each rows
* Using a loop to add the number of prime numbers in each rows which has been counted before to the empty array
* Find which row has the highest counted value
* Print the result (the row which has the most prime numbers).

## Task 1h: Regarding the matrix A, find the rows which have the longest contiguous odd numbers sequence, and print the rows to the screen.

* Initialize an empty array to store data
* Define a function to count the longest contiguous odd  
  numbers sequence of input row
  + Using two variables to store the number of contiguous odd numbers and the highest number of contiguous odd numbers counted in each row
* Use a loop to add data with the longest contiguous odd number into the empty array
* Find the the longest contiguous odd numbers sequence from the data of each row
* Print the result (the row has the the longest contiguous odd  
  numbers sequence).

# SOURCE CODES AND OUTPUTS

## Import Python libraries

### Source code

In this essay, we need to work a lot with matrices and array so we need to import a Python library such as numpy:

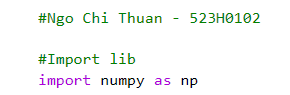


Figure 2.1.1.1: Import numpy library

After that, starting initialize matrices:

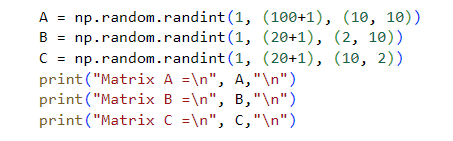


Figure 2.1.1.2: Initialize matrices required and print to screen

Where:

* A is a matrix with random integers
* B is a matrix with random integers
* C is a matrix with random integers .

These matrice has been initialized by using this syntax:

**“np.random.randint(low, high + 1, size)”**

So, we have parameters of these matrices:

* Matrix A:
* Matrix B:
* Matrix C: .

### Output

After intialize matrices successfully, we can see the result:

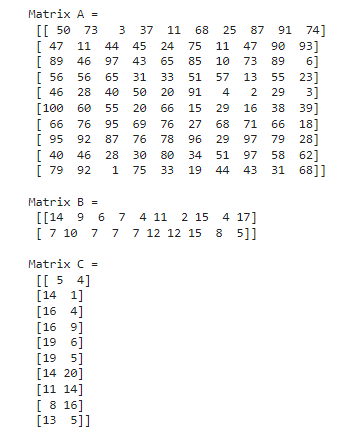


Figure 2.1.2: Output after import Python libraries and intialize matrices

## Task 1a: Calculate A + A^T + CB + (B^T)( C^T), and print the results

### Source code

To calculate these complicate task, we need to use some functions such as **“np.add()”** to add two matrices together, **“np.matmul()”** to multiply two matrice and **“np.transpose()”** or **“.T”** to transpose a matrix.

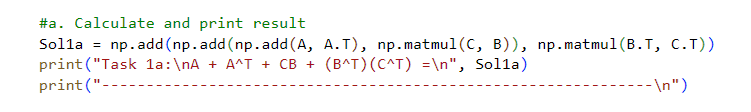


Figure 2.2.1: Source code of task 1a

First, we need to add matrix A with the transpostion matrix of A by using **“np.add(A, A.T)”**, then we add the result in the previous equation with the result from multiply matrices C and B by using **“np.matmul(C, B)”** so we have **“np.add(np.add(A, A.T), matmul(C, B))”**. Finally, Add it to the result of the multiplication of the transpostion matrix of matrix B and the transpostion matrix of matrix C we have: **“np.add(np.add(np.add(A, A.T), np.matmul(C, B)), np.matmul(B.T, C.T))”**.

### Output

Result after calculated:

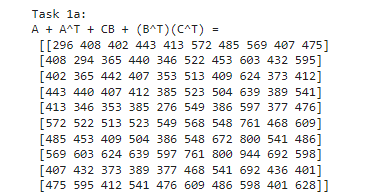


Figure 2.2.2: Output of task 1a

## Task 1b: Calculate A/10 + (A/11)^2 + … + (A/19)^10, and print the results to the screen

### Source code

We can see that the above equation has a fomula:

So, we need to create a loop to run from 1 to 10 to add each equation together:

First of all, we need to intialize a zero matrix with the same size with matrix A. Because, if we want to add two matrices, both of them must be the same size. In this case, our zero matrix named **“Sol1b”** and create a temporary named **“temp1b”** to store the value of denominator. Because, the denominator changes if **“i”** changes.

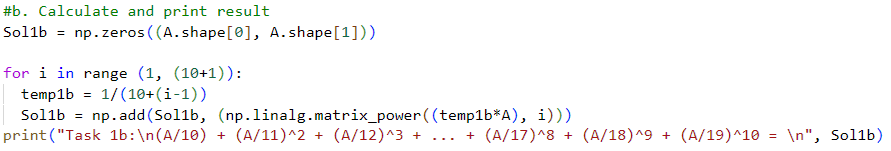


Figure 2.3.1: Source code of task 1b

Second, create a loop with the corresponding number of running in this case is from 1 to 10 using **“for i in range(1, (10+1))”** and using two function: **“np.add()”** to calculate addition of two matrices and **“np.linalg.matrix\_power()”** to Calculate a matrix to the power of a number. In the first run, and before calculate , let and , and it’ll keep addition of new matrix calculated with new **i** each loop with **Sol1b** after loop we have the result of that problem.

### Output

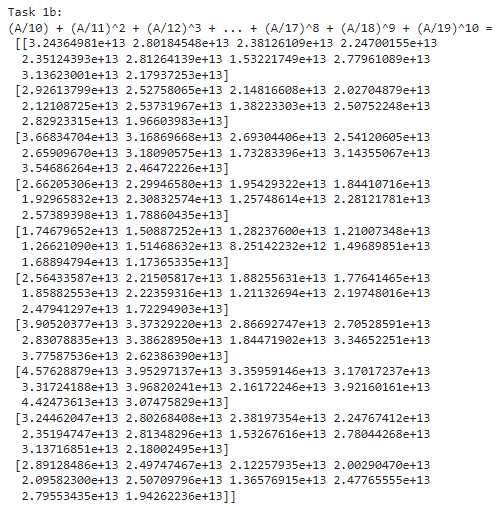
Output recived after calculate problem in task 1b:

Figure 2.3.2: Output of task 1b

## Task 1c: Save odd rows of the matrix A into a new matrix, and print the resultant matrix to the screen

### Source code

We using the syntax **<matrix>[start:end:step,:]** to get the specific rows but still keep columns of the matrix. In this case we need odd rows and the first row is odd row and the first index of row start with **0** so , we want it get till the end of matrix so we will leave **end** parameter. The step of every odd number to get to the next closest odd number is **2** so .

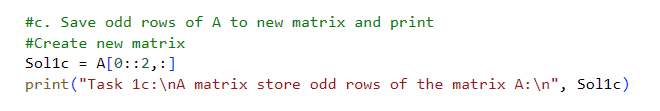


Figure 2.4.1: Source code of task 1c

In this case, we’ll need to initialize a matrix named **“Sol1c” to store all odd rows**. We have:

Finally, print the final matrix which store odd rows of matrix A.

### Output

Result:

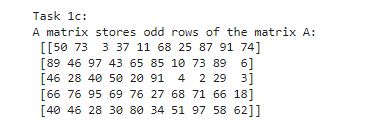


Figure 2.4.2: Output of task 1c

## Task 1d: Save odd integer numbers in the matrix A into a new vector, and print the resultant vector to the screen

### Source code

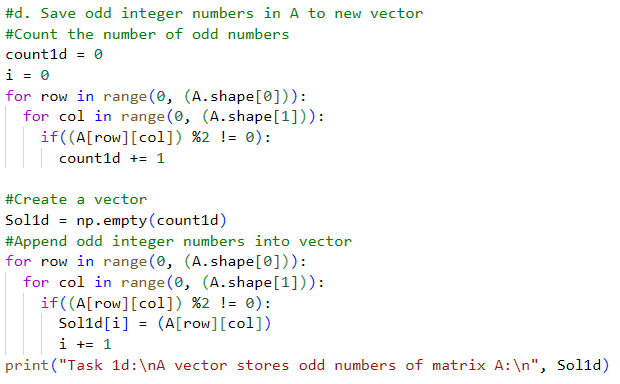


Figure 2.5.1: Source code of task 1d

First of all, we need to count the number of odd numbers in matrix A to identify the size of vector which store odd integer numbers of matrix A, to do that we need to initialize a variable to count and has the value equal 0. In this case, I create a variable named **“count1d”**. After that using two loop to count the numbers of odd integer numbers:

**“for row in range(0 , (A.shape[0])):**

**for col in range(0, (A.shape[1])):**

**if((A[row][col] %2 != 0)):**

**count1d +=1”**

After have the size by counting, we initialize an empty vector with size equals the number of odd integer numbers in matrix A named **“Sol1d”**:

**” Sol1d =np.empty(count1d)”**

Then, using two loops again to put all odd integer numbers in matrix A to the new empty vector:

**“for row in range(0 , (A.shape[0])):**

**for col in range(0, (A.shape[1])):**

**if((A[row][col] %2 != 0)):**

**Sol1d[i] = A[row][col]**

**i +=1”**

Finally, print the new vector to screen.

### Output

Result:

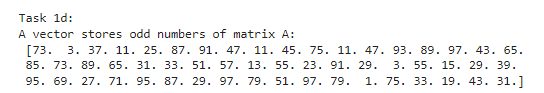


Figure 2.5.2: Output of task 1d

## Task 1e: Save prime numbers in the matrix A into a new vector, and print the resultant vector to the screen

### Source code

We need to define a function to check if a number is a prime number first to make problem become easier to solve:

Define a function named **“Prime”** return **“False”** if the number isn’t a prime number and return **“True”** if it is a prime number:

* If the number, return **“False”** (number is not a prime number)
* If from 2 to the half of that number, there is one number divisible by that number, return **“False”**. Else return **“True”** (number is a prime number).

Initialize a variable to count the number of prime numbers in A

Using two loops and **“Prime”** function to check every element in matrix A and the count variable will increase 1 if a number is prime.

Then, initialize an empty vector with size equals the number of prime **(count prime variable)** and using two loops to add prime numbers into new vector.

Finally, print the new vector on screen which stores prime numbers of matrix A.

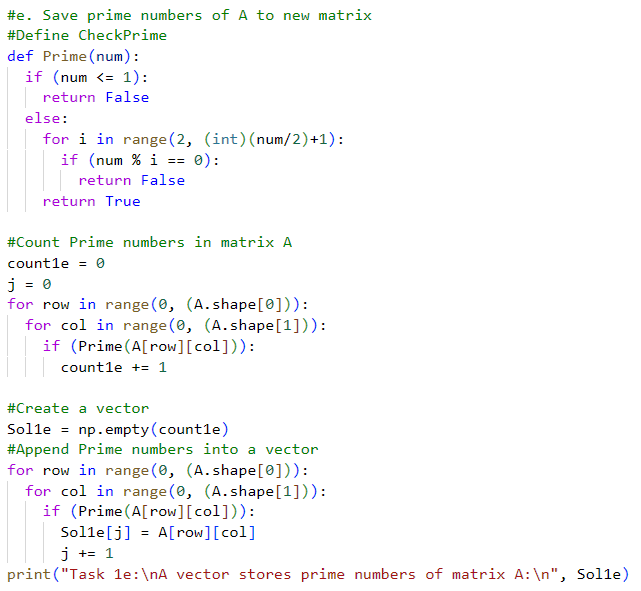


Figure 2.6.1: Source code of task 1e

### Output

Result:

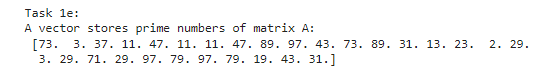


Figure 2.6.2: Output of task 1e

## Task 1f: Given a matrix *D = CB*, reverse elements in the odd rows of the matrix D, and print the resultant matrix to the screen

### Source code

So matrix D is created by multiply two matrices C and B, using function **“D = np.matmul(C, B)”** to multiply matrix C with matrix B and store that result to D.

We need to initialize a new matrix equals to D. In this case it named **“D\_rev”** using D\_rev = **“np.copy(D)”** to do it.

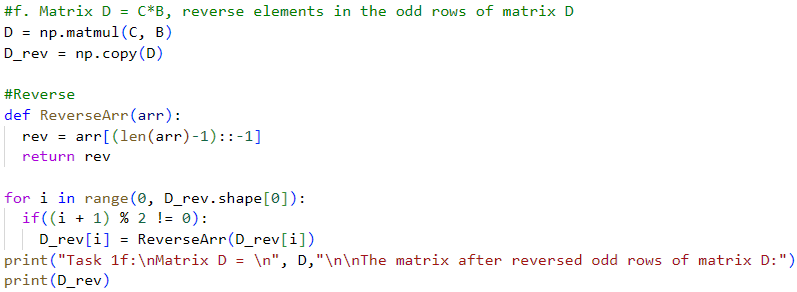


Figure 2.7.1: Source code of task 1f

Define a function named ReverseArr to reverse an input row **“arr”**:

* Initialize an array named **“rev”** to store the reversed input row
* Using function **“rev = arr[(len(arr)-1)::-1]”** to reverse an array it will place the value in last index of **“arr”** to the begin index of rev array (begin index is zero) and keeping replace until value at final index of **“rev”** array equals value at first index of **“arr”** array.

Using a loop run from until (“i” present for row’s index of matrix **“D\_rev”**):

* Only reverse a row if that is an odd row so we will using **“if((i + 1) % 2 != 0)”** and reverse that row:

**“for i in range (0, D\_rev.shape[0]):**

**if ((i + 1) % 2 != 0):**

**D\_rev[i] = ReverseArr(D\_rev[i])”**

After reversed all odd row of matrix D, using **“print(D\_rev)”** to print reversed matrix out.

### Output

Result:

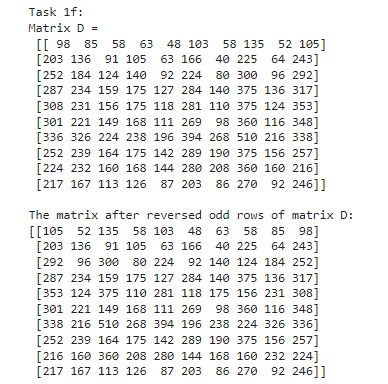


Figure 2.7.2: Output of task 1f

## Task 1g: Regarding the matrix A, find the rows which have maximum count of prime numbers, and print the rows to the screen

### Source code

Initialize an empty array named **“DataPrime”** to store the maximum count of prime numbers of each row

Define new function named **“CountPrime”** to count prime of each row:

* Starting with a count variable named **“countp”** and this value is 0
* Using a loop and **“Prime”** function which used in **“Task 1e”** to count the number of prime numbers in input row **(“arr”)**.

Create a loop to add the number of prime numbers have been counted in each row to **“DataPrime”** array

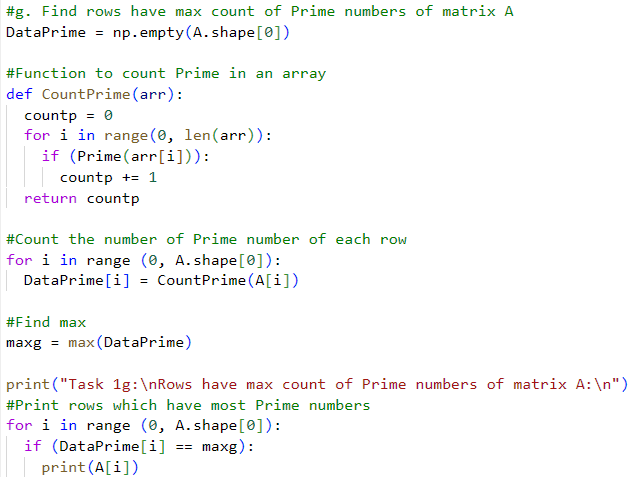


Figure 2.8.1: Source code of task 1g

Using **“max()”** function to find the maximum value in **“DataPrime”** (find the maximum number of prime numbers in all row of matrix A), stored in **“maxg”**

Create a loop to compare and print row if that row has the number of prime numbers equal to **“maxg”**.

### Output

Result:

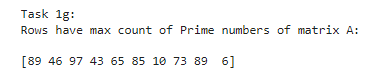


Figure 2.8.2: Output of task 1g

## Task 1h: Regarding the matrix A, find the rows which have the longest contiguous odd numbers sequence, and print the rows to the screen

### Source code

Initialize an empty array named **“DataOddSteak”** to store the maximum number of contiguous odd numbers sequence of each row

Define a function named **“OddSteak”** to count the maximum the maximum number of contiguous odd numbers sequence of each array

* Initialize two variables one for counting the curent odd steak which is named **“currentsteak”**, one for store maximum value number of contiguous odd numbers sequence and it’s named **“maxsteak”**
* Create a loop to count contiguous odd numbers:
  + If the the number is odd, **“currentsteak”** increase 1. If it not an odd number, if will set **“currentsteak”** = **0**, it means odd number steak has been broken.
  + Variable: **“currentsteak”** in each loop will compare to **“maxsteak”**, if two variables: **,** **“OddSteak”** function will **“maxsteak”** with value of **“currentsteak”**.

Initialize a loop to append **“currentsteak”** of each row to **“DataOddSteak”** array

Using **“max()”** function to find the maximum value in **“DataOddSteak”** (find the longest number of contiguous odd numbers sequence in all row of matrix A), stored in **“maxh”**

Create a loop to compare and print row if that row has the longest contiguous odd numbers sequence equal to **“maxh”**.

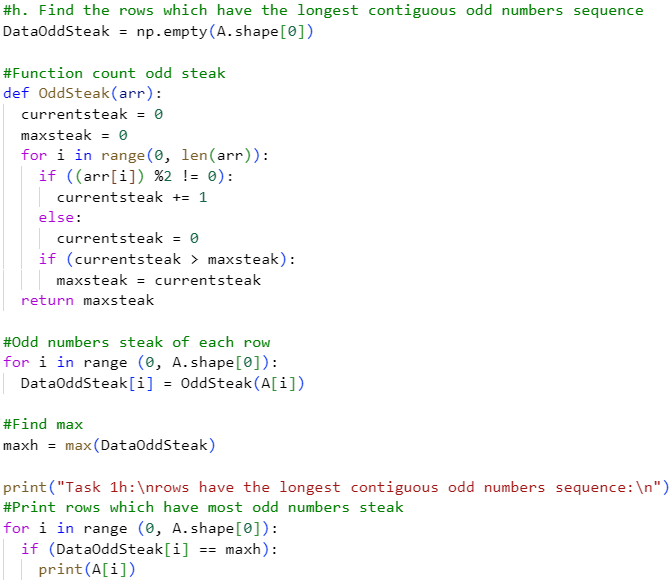


Figure 2.9.1: Source code 1h

### Output

Result:



Figure 2.9.2: Output of task 1h

# CONCLUSION

After solving and explain this mid-term essay, I knew about more information of matrix and vector. It really helps me improve my skill about problem solving.

# REFERENCES

AxelBoldt. (2001). *Prime number*. Retrieved from Wikipedia: <https://en.wikipedia.org/wiki/Prime_number>

Hugunin, J. (n.d.). *NumPy documentation*. Retrieved from NumPy: <https://numpy.org/doc/stable/>

Jain, S. (2008). *Python def Keyword*. Retrieved from GeeksforGeeks: <https://www.geeksforgeeks.org/python-def-keyword/>

Villiers, C. d. (2022, July 13). *NumPy's max() and maximum(): Find Extreme Values in Arrays*. Retrieved from RealPython: <https://realpython.com/numpy-max-maximum/>