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

CHAPTER 1. METHODOLOGY OF SOLVING TASKS

1.1 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.

1.2 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.

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.

- 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.

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.

- 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).

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.

- 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).

CHAPTER 2. SOURCE CODES AND OUTPUTS

2.1 Import Python libraries

2.1.1 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:

```
#Ngo Chi Thuan - 523H0102

#Import lib
import numpy as np
```

Figure 2.1.1.1: Import numpy library

After that, starting initialize matrices:

```
A = np.random.randint(1, (100+1), (10, 10))
B = np.random.randint(1, (20+1), (2, 10))
C = np.random.randint(1, (20+1), (10, 2))
print("Matrix A =\n", A,"\n")
print("Matrix B =\n", B,"\n")
print("Matrix C =\n", C,"\n")
```

Figure 2.1.1.2: Initialize matrices required and print to screen

Where:

- A is a 10×10 matrix with random integers $\in [1, 100]$
- B is a 2×10 matrix with random integers $\in [1, 20]$
- C is a 10×2 matrix with random integers $\in [1, 20]$.

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: $low = 1, high = 100,\ size = (10, 10)$
- Matrix B: $low = 1, high = 20,\ size = (2, 10)$
- Matrix C: $low = 1, high = 20,\ size = (10, 2)$.

2.1.2 Output

After intialize matrices successfully, we can see the result:

```
Matrix A =
 [[ 50  73   3  37  11  68  25  87  91  74]
 [ 47  11  44  45  24  75  11  47  90  93]
 [ 89  46  97  43  65  85  10  73  89   6]
 [ 56  56  65  31  33  51  57  13  55  23]
 [ 46  28  40  50  20  91   4   2  29   3]
 [100  60  55  20  66  15  29  16  38  39]
 [ 66  76  95  69  76  27  68  71  66  18]
 [ 95  92  87  76  78  96  29  97  79  28]
 [ 40  46  28  30  80  34  51  97  58  62]
 [ 79  92   1  75  33  19  44  43  31  68]]

Matrix B =
 [[14  9  6  7  4 11  2 15  4 17]
 [ 7 10  7  7  7 12 12 15  8  5]]

Matrix C =
 [[ 5  4]
 [14  1]
 [16  4]
 [16  9]
 [19  6]
 [19  5]
 [14 20]
 [11 14]
 [ 8 16]
 [13  5]]
```

Figure 2.1.2: Output after import Python libraries and intialize matrices

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

2.2.1 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.

```
#a. Calculate and print result
Sol1a = np.add(np.add(np.add(A, A.T), np.matmul(C, B)), np.matmul(B.T, C.T))
print("Task 1a:\nA + A^T + CB + (B^T)(C^T) =\n", Sol1a)
```

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))”**.

2.2.2 Output

Result after calculated:

```
Task 1a:
A + A^T + CB + (B^T)(C^T) =
 [[296 408 402 443 413 572 485 569 407 475]
 [408 294 365 440 346 522 453 603 432 595]
 [402 365 442 407 353 513 409 624 373 412]
 [443 440 407 412 385 523 504 639 389 541]
 [413 346 353 385 276 549 386 597 377 476]
 [572 522 513 523 549 568 548 761 468 609]
 [485 453 409 504 386 548 672 800 541 486]
 [569 603 624 639 597 761 800 944 692 598]
 [407 432 373 389 377 468 541 692 436 401]
 [475 595 412 541 476 609 486 598 401 628]]
```

Figure 2.2.2: Output of task 1a

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

2.3.1 Source code

We can see that the above equation has a fomula:

$$\sum_{i=1}^{10}\left(\frac{A}{10+(i-1)}\right)^{i}$$

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 10×10 zero matrix named **"Sol1b"** and create a temporary named **"temp1b"** to store the value of denominator. Because, the denominator changes if **"i"** changes.

```
#b. Calculate and print result
Sol1b = np.zeros((A.shape[0], A.shape[1]))

for i in range (1, (10+1)):
  temp1b = 1/(10+(i-1))
  Sol1b = np.add(Sol1b, (np.linalg.matrix_power((temp1b*A), i)))
print("Task 1b:\n(A/10) + (A/11)^2 + (A/12)^3 + ... + (A/17)^8 + (A/18)^9 + (A/19)^10 = \n", Sol1b)
```

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, $\boldsymbol{i = 1}$ and before calculate $\boldsymbol{Sol1b = 0}$, let $\boldsymbol{temp1b = \frac{1}{(10)}}$ and $\boldsymbol{Sol1b = \left(0 + \frac{A}{10}\right)^{1} = \frac{A}{10}}$, 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.

2.3.2 Output

Output recived after calculate problem in task 1b:

```
Task 1b:
(A/10) + (A/11)^2 + (A/12)^3 + ... + (A/17)^8 + (A/18)^9 + (A/19)^10 =
 [[3.24364981e+13 2.80184548e+13 2.38126109e+13 2.24700155e+13
  2.35124393e+13 2.81264139e+13 1.53221749e+13 2.77961089e+13
  3.13623001e+13 2.17937253e+13]
 [2.92613799e+13 2.52758065e+13 2.14816608e+13 2.02704879e+13
  2.12108725e+13 2.53731967e+13 1.38223303e+13 2.50752248e+13
  2.82923315e+13 1.96603983e+13]
 [3.66834704e+13 3.16869668e+13 2.69304406e+13 2.54120605e+13
  2.65909670e+13 3.18090575e+13 1.73283396e+13 3.14355067e+13
  3.54686264e+13 2.46472226e+13]
 [2.66205306e+13 2.29946580e+13 1.95429322e+13 1.84410716e+13
  1.92965832e+13 2.30832574e+13 1.25748614e+13 2.28121781e+13
  2.57389398e+13 1.78860435e+13]
 [1.74679652e+13 1.50887252e+13 1.28237600e+13 1.21007348e+13
  1.26621090e+13 1.51468632e+13 8.25142232e+12 1.49689851e+13
  1.68894794e+13 1.17365335e+13]
 [2.56433587e+13 2.21505817e+13 1.88255631e+13 1.77641465e+13
  1.85882553e+13 2.22359316e+13 1.21132694e+13 2.19748016e+13
  2.47941297e+13 1.72294903e+13]
 [3.90520377e+13 3.37329220e+13 2.86692747e+13 2.70528591e+13
  2.83078835e+13 3.38628950e+13 1.84471902e+13 3.34652251e+13
  3.77587536e+13 2.62386390e+13]
 [4.57628879e+13 3.95297137e+13 3.35959146e+13 3.17017237e+13
  3.31724188e+13 3.96820241e+13 2.16172246e+13 3.92160161e+13
  4.42473613e+13 3.07475829e+13]
 [3.24462047e+13 2.80268408e+13 2.38197354e+13 2.24767412e+13
  2.35194747e+13 2.81348296e+13 1.53267616e+13 2.78044268e+13
  3.13716851e+13 2.18002495e+13]
 [2.89128486e+13 2.49747467e+13 2.12257935e+13 2.00290470e+13
  2.09582300e+13 2.50709796e+13 1.36576915e+13 2.47765555e+13
  2.79553435e+13 1.94262236e+13]]
```

Figure 2.3.2: Output of task 1b

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

2.4.1 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 $\mathbf{start = 0}$, 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 $\boldsymbol{step = 2}$.

```
#c. Save odd rows of A to new matrix and print
#Create new matrix
Sol1c = A[0::2,:]
print("Task 1c:\nA matrix store odd rows of the matrix A:\n", Sol1c)
```

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:

$$"\boldsymbol{Sol1c = A[0 :: 2, :]}"$$

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

2.4.2 Output

Result:

```
Task 1c:
A matrix stores odd rows of the matrix A:
 [[50 73  3 37 11 68 25 87 91 74]
 [89 46 97 43 65 85 10 73 89  6]
 [46 28 40 50 20 91  4  2 29  3]
 [66 76 95 69 76 27 68 71 66 18]
 [40 46 28 30 80 34 51 97 58 62]]
```

Figure 2.4.2: Output of task 1c

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

2.5.1 Source code

```
#d. Save odd integer numbers in A to new vector
#Count the number of odd numbers
count1d = 0
i = 0
for row in range(0, (A.shape[0])):
  for col in range(0, (A.shape[1])):
    if((A[row][col]) %2 != 0):
      count1d += 1

#Create a vector
Sol1d = np.empty(count1d)
#Append odd integer numbers into 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
print("Task 1d:\nA vector stores odd numbers of matrix A:\n", Sol1d)
```

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.

2.5.2 Output

Result:

```
Task 1d:
A vector stores odd numbers of matrix A:
 [73.  3. 37. 11. 25. 87. 91. 47. 11. 45. 75. 11. 47. 93. 89. 97. 43. 65.
 85. 73. 89. 65. 31. 33. 51. 57. 13. 55. 23. 91. 29.  3. 55. 15. 29. 39.
 95. 69. 27. 71. 95. 87. 29. 97. 79. 51. 97. 79.  1. 75. 33. 19. 43. 31.]
```

Figure 2.5.2: Output of task 1d

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

2.6.1 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 $\leq$ **1**, 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.

```
#e. Save prime numbers of A to new matrix
#Define CheckPrime
def Prime(num):
  if (num <= 1):
    return False
  else:
    for i in range(2, (int)(num/2)+1):
      if (num % i == 0):
        return False
    return True

#Count Prime numbers in matrix A
count1e = 0
j = 0
for row in range(0, (A.shape[0])):
  for col in range(0, (A.shape[1])):
    if (Prime(A[row][col])):
      count1e += 1

#Create a vector
Sol1e = np.empty(count1e)
#Append Prime numbers into a vector
for row in range(0, (A.shape[0])):
  for col in range(0, (A.shape[1])):
    if (Prime(A[row][col])):
      Sol1e[j] = A[row][col]
      j += 1
print("Task 1e:\nA vector stores prime numbers of matrix A:\n", Sol1e)
```

Figure 2.6.1: Source code of task 1e

2.6.2 Output

Result:

```
Task 1e:
A vector stores prime numbers of matrix A:
 [73.  3. 37. 11. 47. 11. 11. 47. 89. 97. 43. 73. 89. 31. 13. 23.  2. 29.
  3. 29. 71. 29. 97. 79. 97. 79. 19. 43. 31.]
```

Figure 2.6.2: Output of task 1e

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

2.7.1 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.

```
#f. Matrix D = C*B, reverse elements in the odd rows of matrix D
D = np.matmul(C, B)
D_rev = np.copy(D)

#Reverse
def ReverseArr(arr):
  rev = arr[(len(arr)-1)::-1]
  return rev

for i in range(0, D_rev.shape[0]):
  if((i + 1) % 2 != 0):
    D_rev[i] = ReverseArr(D_rev[i])
print("Task 1f:\nMatrix D = \n", D,"\n\nThe matrix after reversed odd rows of matrix D:")
print(D_rev)
```

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 $\boldsymbol{i = 0}$ until $\boldsymbol{i = D\_rev.shape[0]}$ ("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.

2.7.2 Output

Result:

```
Task 1f:
Matrix D =
 [[ 98  85  58  63  48 103  58 135  52 105]
 [203 136  91 105  63 166  40 225  64 243]
 [252 184 124 140  92 224  80 300  96 292]
 [287 234 159 175 127 284 140 375 136 317]
 [308 231 156 175 118 281 110 375 124 353]
 [301 221 149 168 111 269  98 360 116 348]
 [336 326 224 238 196 394 268 510 216 338]
 [252 239 164 175 142 289 190 375 156 257]
 [224 232 160 168 144 280 208 360 160 216]
 [217 167 113 126  87 203  86 270  92 246]]

The matrix after reversed odd rows of matrix D:
[[105  52 135  58 103  48  63  58  85  98]
 [203 136  91 105  63 166  40 225  64 243]
 [292  96 300  80 224  92 140 124 184 252]
 [287 234 159 175 127 284 140 375 136 317]
 [353 124 375 110 281 118 175 156 231 308]
 [301 221 149 168 111 269  98 360 116 348]
 [338 216 510 268 394 196 238 224 326 336]
 [252 239 164 175 142 289 190 375 156 257]
 [216 160 360 208 280 144 168 160 232 224]
 [217 167 113 126  87 203  86 270  92 246]]
```

Figure 2.7.2: Output of task 1f

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

2.8.1 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

```
#g. Find rows have max count of Prime numbers of matrix A
DataPrime = np.empty(A.shape[0])

#Function to count Prime in an array
def CountPrime(arr):
  countp = 0
  for i in range(0, len(arr)):
    if (Prime(arr[i])):
      countp += 1
  return countp

#Count the number of Prime number of each row
for i in range (0, A.shape[0]):
  DataPrime[i] = CountPrime(A[i])

#Find max
maxg = max(DataPrime)

print("Task 1g:\nRows have max count of Prime numbers of matrix A:\n")
#Print rows which have most Prime numbers
for i in range (0, A.shape[0]):
  if (DataPrime[i] == maxg):
    print(A[i])
```

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”**.

2.8.2 Output

Result:

```
Task 1g:
Rows have max count of Prime numbers of matrix A:

[89 46 97 43 65 85 10 73 89  6]
```

Figure 2.8.2: Output of task 1g

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

2.9.1 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: “**currentsteak**” >

"**maxsteak**", **"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"**.

```
#h. Find the rows which have the longest contiguous odd numbers sequence
DataOddSteak = np.empty(A.shape[0])

#Function count odd steak
def OddSteak(arr):
  currentsteak = 0
  maxsteak = 0
  for i in range(0, len(arr)):
    if ((arr[i]) %2 != 0):
      currentsteak += 1
    else:
      currentsteak = 0
    if (currentsteak > maxsteak):
      maxsteak = currentsteak
  return maxsteak

#Odd numbers steak of each row
for i in range (0, A.shape[0]):
  DataOddSteak[i] = OddSteak(A[i])

#Find max
maxh = max(DataOddSteak)

print("Task 1h:\nrows have the longest contiguous odd numbers sequence:\n")
#Print rows which have most odd numbers steak
for i in range (0, A.shape[0]):
  if (DataOddSteak[i] == maxh):
    print(A[i])
```

Figure 2.9.1: Source code 1h

2.9.2 Output

Result:

```
Task 1h:
rows have the longest contiguous odd numbers sequence:
[56 56 65 31 33 51 57 13 55 23]
```

Figure 2.9.2: Output of task 1h

CHAPTER 3. 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/