VIETNAM GENERAL CONFEDERATION OF LABOR
TON DUC THANG UNIVERSITY
FACULTY OF INFORMATION TECHNOLOGY

TRUONG PHUC NGUYEN – 520H0392

MITERM ESSAY

APPLIED LINEAR ALGEBRA FOR IT

HO CHI MINH CITY, 2024

VIETNAM GENERAL CONFEDERATION OF LABOR
TON DUC THANG UNIVERSITY
FACULTY OF INFORMATION TECHNOLOGY

TRUONG PHUC NGUYEN – 520H0392

MIDTERM ESSAY

APPLIED LINEAR ALGEBRA FOR IT

Advised by
Dr. Huynh Thi Thu Thuy

HO CHI MINH CITY, 2024

ACKNOWLEDGEMENT

Dear Dr. Huynh Thi Thu Thuy

I wanted to take a moment to express my heartfelt gratitude for the incredible lecture you gave on "Applied Linear Algebra for IT". Your expertise, passion, and commitment to teaching were evident throughout the lecture, and I learned so much from you in a short period of time.

Once again, thank you for your exceptional lecture and for sharing your knowledge and expertise with us. Your contribution to my education and future success is deeply appreciated.

Thank you sincerely for everything.

Ho Chi Minh city, 9th April 2024

Author

(Signature and full name)

Truong Phuc Nguyen

DECLARATION OF AUTHORSHIP

We hereby declare that this is our own project and is guided by Dr. Huynh Thi Thu Thuy; 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, 9th April 2024

Author

(Signature and full name)

Truong Phuc Nguyen

TABLE OF CONTENT

CHAPTER 1. METHODOLOGY FOR SOLVING TASKS

1.1 Task 1d

Using python operators → **A[A%2==1]** will return odd values in matrix A.

1.2 Task 1e

- Define an **isPrime()** function that will check if a number is prime.
- Create an empty vector called **task1e** to save the resultant vector.
- Conduct a for loop through matrix A, check each value by **isPrime()** then append to **task1e** if true.

1.3 Task 1f

- Calculate matrix D by using **numpy.dot(C,B)**
- Create a **task1f** matrix that copy matrix D
- Select odd rows: **task1f[::2]**
- Then reverse row values by **task1f[::2,::-1]**

1.4 Task 1g

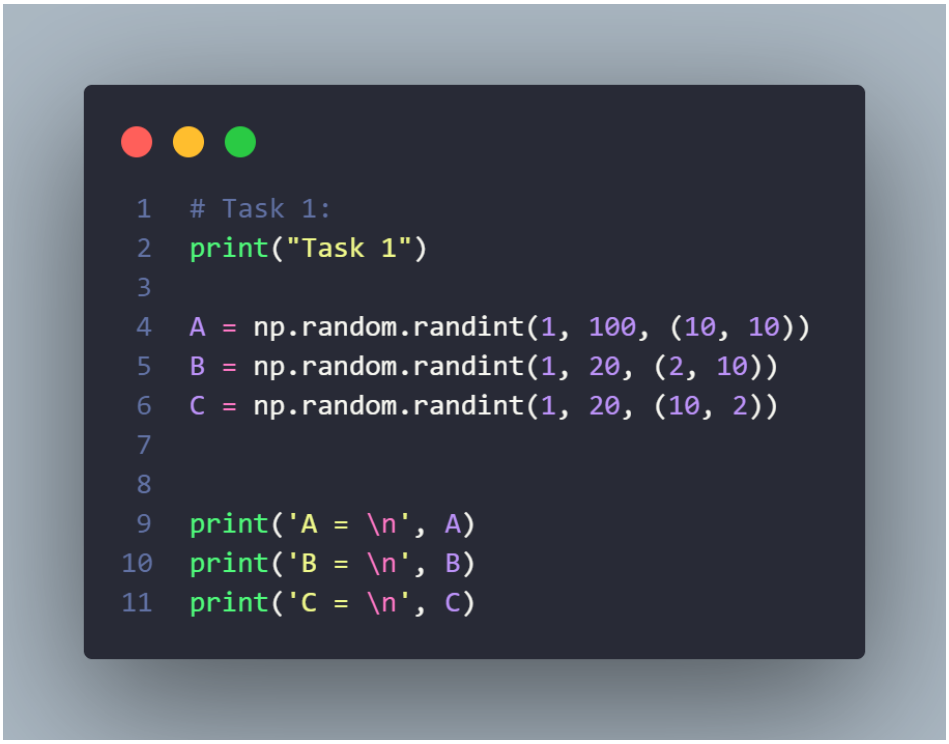
- Already have **isPrime()** function in task 1e
- Define a **maxPrimeRow()** to do the requirement:
    - Create an empty **temp** vector to save the resultant
    - Iterate each row to count the number of prime number
      → get the **maxCount** of prime number in matrix A
- When iterate over the rows, if find a row have the **maxCount** > current **maxCount**
  → Update **maxCount** value and store the maxCount row to **temp**

1.5 Task 1h

- Define a **longestOddSeq()** to do the requirement
  - Create an empty vector **temp** to save the resultant.
  - Initialize **countOdd** value to save the value of maximum contiguous odd number.
  - Conduct a for loop to check if an element is odd number
    → **countOdd** +1
  - If an element is even number
    → reset **countOdd**
  - If the current count value > **countOdd**
    → Store the current row to **temp**.

CHAPTER 2. SOURCE CODES AND OUTPUT

2.1 Create 3 matrices



```
# Task 1:
print("Task 1")

A = np.random.randint(1, 100, (10, 10))
B = np.random.randint(1, 20, (2, 10))
C = np.random.randint(1, 20, (10, 2))


print('A = \n', A)
print('B = \n', B)
print('C = \n', C)
```

Figure 1 Create matrices A, B, C

```
Task 1
A =
 [[82 42 32 47 52 68 84 84 43 68]
 [74 77 34 22  1 63 25 99 34 16]
 [18 13 43 84 16 41 88 57 91 73]
 [99 91  3 18 84 55 32 64 96 58]
 [49 81 71 34 93 48 54 19 57 50]
 [96 59 13 87 34 35 63 81 75  2]
 [96 95 96  9 26  5 52 82 16 33]
 [52 60 72 63 45 32 41 84 37 34]
 [56 17 49 67 66 26 54 19 52 70]
 [40 21 21 24 86 81 54 80 73 24]]
B =
 [[ 5  5 10  1 14  8  3 15  7 11]
 [ 2  8 16  8 16 13 17 10  7 11]]
```

```
C =
 [[ 8  9]
 [14 12]
 [ 6  3]
 [ 3 11]
 [13  8]
 [ 7 13]
 [10 18]
 [ 1 18]
 [16 15]
 [11 13]]
```

Ouput 3 matrices

2.2 Coding and Output

2.2.1 Task 1a

```
# Task 1a: Calculate A + A^T + C*B + (B^T)*(C^T)
print('Task 1a\n',  A + A.T + np.dot(C, B) + np.dot(B.T, C.T))
```

Output:

```
Task 1a
 [[280 322 310 263 438 406 443 387 328 376]
 [322 486 433 326 599 529 560 638 433 482]
 [310 433 302 323 477 419 641 547 603 511]
 [263 326 323 218 413 420 391 427 397 351]
 [438 599 477 413 806 596 683 641 734 729]
 [406 529 419 420 596 520 624 590 564 560]
 [443 560 641 391 683 624 776 762 569 649]
 [387 638 547 427 641 590 762 558 579 618]
 [328 433 603 397 734 564 569 579 538 652]
 [376 482 511 351 729 560 649 618 652 576]]
```

2.2.2 Task 1b

```python
# Task 1b: calculate (A/10)^1 + (A/11)^2 + (A/12)^3 + ... + (A/19)^10
print('\nTask 1b\n', np.sum([(np.linalg.matrix_power(A/(i+10), i+1)) for i in range(10)], axis=0))
```

Output:

```
Task 1b
 [[4.24574650e+13 3.56578620e+13 2.88307973e+13 2.89784018e+13
  3.13174560e+13 2.86088121e+13 3.47672855e+13 4.32800917e+13
  3.47459325e+13 2.76011778e+13]
 [3.09877545e+13 2.60250364e+13 2.10422750e+13 2.11500050e+13
  2.28571729e+13 2.08802585e+13 2.53750452e+13 3.15881523e+13
  2.53594605e+13 2.01448323e+13]
 [3.65007292e+13 3.06551030e+13 2.47858685e+13 2.49127639e+13
  2.69236516e+13 2.45950271e+13 2.98894739e+13 3.72079420e+13
  2.98711165e+13 2.37287630e+13]
 [4.16584171e+13 3.49867824e+13 2.82882026e+13 2.84330293e+13
  3.07280623e+13 2.80703951e+13 3.41129663e+13 4.24655622e+13
  3.40920151e+13 2.70817246e+13]
 [3.84868905e+13 3.23231788e+13 2.61345736e+13 2.62683742e+13
  2.83886827e+13 2.59333480e+13 3.15158881e+13 3.92325860e+13
  3.14965319e+13 2.50199468e+13]
 [3.87089299e+13 3.25096583e+13 2.62853495e+13 2.64199224e+13
  2.85524629e+13 2.60829630e+13 3.16977098e+13 3.94589277e+13
  3.16782420e+13 2.51642918e+13]
 [3.52583268e+13 2.96116726e+13 2.39422131e+13 2.40647893e+13
  2.60072311e+13 2.37578678e+13 2.88721033e+13 3.59414674e+13
  2.88543709e+13 2.29210895e+13]
 [3.64469535e+13 3.06099396e+13 2.47493517e+13 2.48760604e+13
  2.68839851e+13 2.45587917e+13 2.98454381e+13 3.71531247e+13
  2.98271078e+13 2.36938038e+13]
 [3.38181270e+13 2.84021221e+13 2.29642438e+13 2.30818131e+13
  2.49449116e+13 2.27874282e+13 2.76927623e+13 3.44733636e+13
  2.76757542e+13 2.19848299e+13]
 [3.53991112e+13 2.97299101e+13 2.40378130e+13 2.41608789e+13
  2.61110763e+13 2.38527317e+13 2.89873881e+13 3.60849802e+13
  2.89695848e+13 2.30126119e+13]]
```

2.2.3 Task 1c

```
# Task 1c: Save odd rows of the matrix A into a new matrix, and print the resultant matrix to the screen.
task1c = A[::2]
print('\nTask 1c = \n', task1c)
```

Output:

```
Task 1c =
 [[82 42 32 47 52 68 84 84 43 68]
 [18 13 43 84 16 41 88 57 91 73]
 [49 81 71 34 93 48 54 19 57 50]
 [96 95 96  9 26  5 52 82 16 33]
 [56 17 49 67 66 26 54 19 52 70]]
```

2.2.4 Task 1d

```
# Task 1d: Save odd integer numbers in the matrix A into a new vector, and print the resultant vector to the screen.
task1d = A[A % 2 == 1]
print('\nTask 1d = \n', task1d)
```

Output:

```
Task 1d =
 [47 43 77  1 63 25 99 13 43 41 57 91 73 99 91  3 55 49 81 71 93 19 57 59
 13 87 35 63 81 75 95  9  5 33 63 45 41 37 17 49 67 19 21 21 81 73]
```

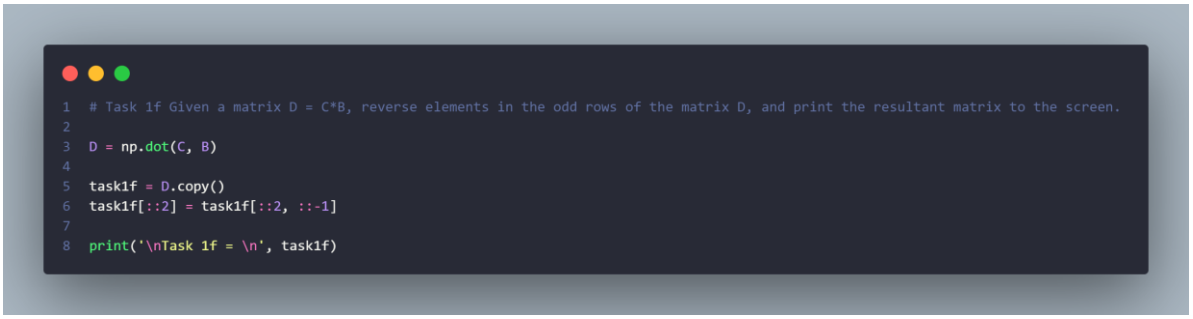
2.2.5 Task 1e

```python
# Task 1e: Save prime numbers in the matrix A into a new vector, and print the resultant vector to the screen.
def isPrime(n):
    if n < 2:
        return False
    for i in range(2, int(np.sqrt(n))+1):
        if n % i == 0:
            return False
    return True


task1e = []
for i in range(10):
    for j in range(10):
        if isPrime(A[i][j]):
            task1e.append(A[i][j])
print('\nTask 1e = \n', np.array(task1e))
```

Output:

```
Task 1e =
 [47 43 13 43 41 73  3 71 19 59 13  2  5 41 37 17 67 19 73]
```

2.2.6 Task 1f

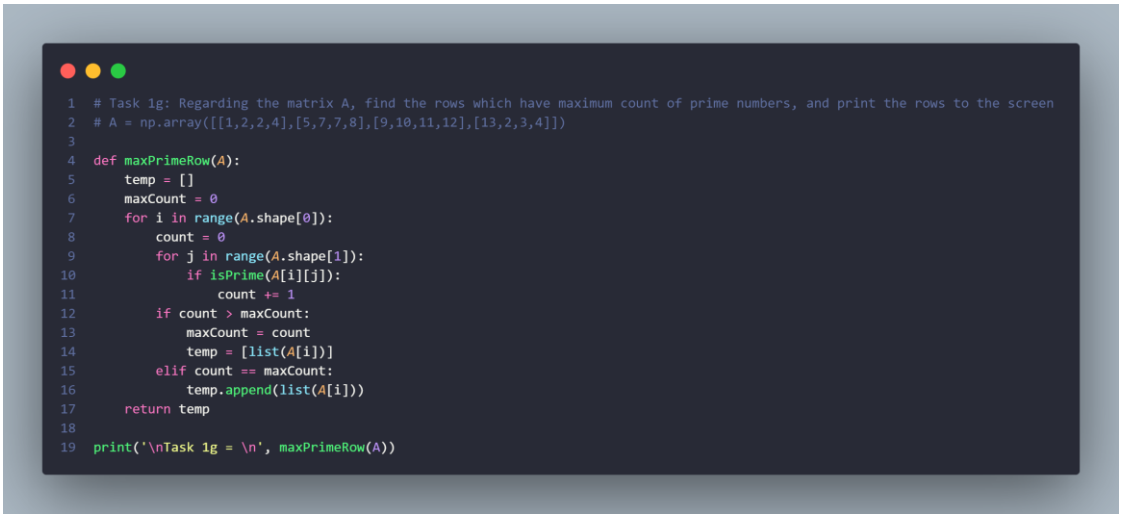


```python
# Task 1f Given a matrix D = C*B, reverse elements in the odd rows of the matrix D, and print the resultant matrix to the screen.

D = np.dot(C, B)

task1f = D.copy()
task1f[::2] = task1f[::2, ::-1]

print('\nTask 1f = \n', task1f)
```

Output:

```
Task 1f =
 [[187 119 210 177 181 256  80 224 112  58]
 [ 94 166 332 110 388 268 246 330 182 286]
 [ 99  63 120  69  87 132  30 108  54  36]
 [ 37 103 206  91 218 167 196 155  98 154]
 [231 147 275 175 208 310  77 258 129  81]
 [ 61 139 278 111 306 225 242 235 140 220]
 [308 196 330 336 314 428 154 388 194  86]
 [ 41 149 298 145 302 242 309 195 133 209]
 [341 217 390 303 323 464 136 400 200 110]
 [ 81 159 318 115 362 257 254 295 168 264]]
```

2.2.7 Task 1g



```python
# Task 1g: Regarding the matrix A, find the rows which have maximum count of prime numbers, and print the rows to the screen
# A = np.array([[1,2,2,4],[5,7,7,8],[9,10,11,12],[13,2,3,4]])

def maxPrimeRow(A):
    temp = []
    maxCount = 0
    for i in range(A.shape[0]):
        count = 0
        for j in range(A.shape[1]):
            if isPrime(A[i][j]):
                count += 1
        if count > maxCount:
            maxCount = count
            temp = [list(A[i])]
        elif count == maxCount:
            temp.append(list(A[i]))
    return temp

print('\nTask 1g = \n', maxPrimeRow(A))
```

Output:

```
Task 1g =
 [[18, 13, 43, 84, 16, 41, 88, 57, 91, 73]]
```

2.2.8 Task 1h

```python
# Task 1h: Regarding the matrix A, find the rows which have longest sequence of odd numbers, and print the rows to the screen
# A = np.array([[1,2,3,5,5,8],[5,7,7,8,9],[0,2,4,6,8],[8,2,9,3,7]])

def longestOddSeq(A):
    temp = []
    countOdd = 0
    for i in range(A.shape[0]):
        count = 0
        for j in range(A.shape[1]):
            if A[i][j] % 2 == 1:
                count += 1
            else:
                count = 0
            if count > countOdd:
                countOdd = count
                temp = [list(A[i])]
            elif count == countOdd:
                temp.append(list(A[i]))
    return temp

print('\nTask 1h = \n', longestOddSeq(A))
```

Output:

```
Task 1h =
 [[74, 77, 34, 22, 1, 63, 25, 99, 34, 16], [96, 59, 13, 87, 34, 35, 63, 81, 75, 2]]
```