TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG
KHOA CÔNG NGHỆ THÔNG TIN

BÀI TIỂU LUẬN GIỮA KÌ MÔN ĐẠI SỐ TUYẾN TÍNH CHO CÔNG NGHỆ THÔNG TIN

MIDTERM ESSAY: Applied Linear Algebra for IT

CODE: 501032

Người hướng dẫn: **ThS. HUỲNH THỊ THU THỦY**

Người thực hiện: **NGUYỄN ĐÌNH VIỆT HOÀNG – 522H0120**

Lớp **: 22H50302**

Khoá **: 26**

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2023

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG
KHOA CÔNG NGHỆ THÔNG TIN

BÀI TIỂU LUẬN GIỮA KÌ MÔN ĐẠI SỐ TUYẾN TÍNH CHO CÔNG NGHỆ THÔNG TIN

MIDTERM ESSAY: Applied Linear Algebra for IT

CODE: 501032

Người hướng dẫn: **ThS. HUỲNH THỊ THU THỦY**

Người thực hiện: **NGUYỄN ĐÌNH VIỆT HOÀNG – 522H0120**

Lớp **: 22H50302**

Khoá **: 26**

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2023

LỜI CẢM ƠN

Em xin phép cảm ơn cô và các bạn trong lớp đã giúp đỡ, hướng dẫn em trong suốt quá trình thực hiện các bài tập rèn luyện được cô giao trên lớp và về nhà. Em cũng xin gửi ngàn lời cảm ơn sâu sắc đến trường vì đã cung cấp một không gian học tập một cách tốt nhất để tụi em có thể học tập một cách thuận tiện nhất. Em xin cảm ơn!

TIỂU LUẬN ĐƯỢC HOÀN THÀNH TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG

Tôi xin cam đoan đây là sản phẩm tiểu luận của riêng tôi và được sự hướng dẫn của ThS. Huỳnh Thị Thu Thủy. Các nội dung nghiên cứu, kết quả trong tiểu luận này là trung thực và chưa công bố dưới bất kỳ hình thức nào trước đây.

Nếu phát hiện có bất kỳ sự gian lận nào tôi xin hoàn toàn chịu trách nhiệm về nội dung tiểu luận của mình. Trường đại học Tôn Đức Thắng không liên quan đến những vi phạm tác quyền, bản quyền do tôi gây ra trong quá trình thực hiện (nếu có).

TP. Hồ Chí Minh, ngày 27 tháng 04 năm 2023

Tác giả

(ký tên và ghi rõ họ tên)

Hoàng

Nguyễn Đình Việt Hoàng

PHẦN XÁC NHẬN VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN

Phần xác nhận của GV hướng dẫn

Tp. Hồ Chí Minh, ngày tháng năm
(kí và ghi họ tên)

Phần đánh giá của GV chấm bài

Tp. Hồ Chí Minh, ngày tháng năm
(kí và ghi họ tên)

TÓM TẮT

Trước khi giải quyết vấn đề bài toán, tôi đã đọc tập tin trên 10 lần nhưng không hiểu một số từ vựng tiếng Anh chuyên ngành nên tôi đã dùng google translate để dịch từ ngữ đó. Và sau khi tôi đã dịch xong đề bài tôi đã hiểu được yêu cầu của nó. Đầu tiên, tôi phát hiện được là bài toán này cần tạo ma trận nhiều chiều A, B, C để giải quyết các câu a, b, c, d, e, f, g, h. Tiếp theo là giải quyết câu a bằng cách sử dụng ma trận chuyển vị và sử dụng phương thức "dot" của thư viện numpy để nhân các ma trận lại với nhau và in ra màn hình kết quả có được. Với câu b thì ta sử dụng vòng lặp for để tính lũy thừa của từng phân số bằng phương thức "power" của thư viện numpy và in kết quả ra màn hình. Đối với câu c thì ta sẽ tạo một mảng "mask" Boolean để xét đúng sai của nó, đối với True thì là hàng lẻ và đối với False thì là hàng chẵn; bây giờ tôi sẽ tạo một biến "A_odd" để lưu trữ các hàng lẻ của ma trận A đã được khởi tạo trước đó và sử dụng "mask" để trích xuất các hàng lẻ của ma trận A bằng cách sử dụng phép toán truy cập mảng trong NumPy. Kết quả cuối cùng là ma trận "A_odd" chứa các hàng lẻ của ma trận A và được in ra màn hình. Còn đối với câu d thì ta sẽ tạo một danh sách rỗng "odd_numbers" để lưu trữ các số lẻ trong ma trận A. Đoạn code sử dụng vòng lặp "for" để lặp từng phần tử trong ma trận A. Với mỗi phần tử trong ma trận A, nếu giá trị của nó là số lẻ, thì nó được thêm vào danh sách "odd_numbers". Và sau khi lặp xong toàn bộ ma trận A, danh sách "odd_numbers" sẽ chứa tất cả các số lẻ có trong ma trận. Danh sách "odd_numbers" được chuyển đổi thành một mảng NumPy bằng cách sử dụng hàm "numpy.array()". Cuối cùng mảng Numpy "odd_numbers" chứa tất cả các số lẻ trong ma trận A và được in ra màn hình. Tiếp tục với câu e thì ta sẽ đi định nghĩa một hàm tên là "is_prime()" để kiểm tra xem một số "num" có phải là số nguyên tố hay không và một vecto chứa các số nguyên tố trong ma trận A, cụ thể là nếu "num" nhỏ hơn hoặc bằng 1, hàm sẽ trả về "False" vì các số nhỏ hơn 2 không phải là số nguyên tố. Sau đó, vòng lặp "for" duyệt qua tất cả các số từ 2 đến căn bậc hai của "num" (chuyển đổi thành kiểu số nguyên bằng hàm "int(np.sqrt(num)) + 1"), nếu "num" chia hết cho một số trong đoạn này, nghĩa là num không phải là số nguyên tố và hàm sẽ trả về False. Sau khi có hàm "is_prime(num)" để kiểm tra số nguyên tố, đoạn code tiếp theo tạo một vector

“prime_vector” rỗng và duyệt qua tất cả các phần tử trong ma trận A với vòng lặp “for” lồng nhau. Với mỗi phần tử “A[k, l]” trong ma trận A, nếu phần tử này là số nguyên tố (kiểm tra bằng hàm “is_prime(A[k, l])”), nó sẽ được thêm vào vector “prime_vector” bằng phương thức “append()”. Cuối cùng, đoạn code in ra màn hình vector prime_vector chứa các số nguyên tố trong ma trận A. Với câu f thì sử dụng thư viện NumPy của Python để thực hiện các phép toán trên ma trận. Cụ thể, đầu tiên, ma trận tích của hai ma trận C và B được tính bằng hàm “np.dot()” và gán vào biến F. Sau đó, vòng lặp for được sử dụng để duyệt qua các hàng của ma trận F, bắt đầu từ hàng thứ 1 và với bước nhảy bằng 2 (vì chỉ xử lý các hàng lẻ). Trong mỗi vòng lặp, các phần tử của hàng lẻ được đảo ngược thứ tự bằng cách sử dụng chỉ số dòng F[y,:] và slicing với bước -1, tức là lấy các phần tử theo thứ tự ngược lại. Cuối cùng, kết quả ma trận F sau khi hoàn thành việc đảo ngược được in ra màn hình bằng lệnh “print()”. Với câu g thì ta sử dụng hàm “is_prime()” được định nghĩa để kiểm tra xem một số có phải là số nguyên tố hay không. Hàm này nhận đầu vào là một số nguyên “number” và trả về True nếu “number” là số nguyên tố và False nếu không phải. Để kiểm tra xem một số có phải là số nguyên tố hay không, hàm này sử dụng vòng lặp “for” để duyệt qua các số từ 2 đến căn bậc hai của số “number” và kiểm tra xem số “number” có chia hết cho các số này hay không. Nếu có, số “number” không phải là số nguyên tố và hàm trả về False. Tiếp theo, một danh sách “prime_counts” được tạo ra bằng cách duyệt qua từng hàng của ma trận A. Đối với mỗi hàng, hàm “is_prime()” được áp dụng cho từng số trong hàng đó để tính toán số lượng số nguyên tố trong hàng. Số lượng các số nguyên tố được tính toán bằng cách sử dụng hàm “sum()” để đếm số lượng giá trị True trong danh sách Boolean được trả về bởi hàm “is_prime()”. Tiếp theo, biến “max_count” được tạo ra để lưu trữ số lượng số nguyên tố lớn nhất trong tất cả các hàng của ma trận A. Biến này được tìm bằng cách sử dụng hàm “max()” trên danh sách “prime_counts”. Sau đó, một danh sách “max_rows” được tạo ra bằng cách duyệt qua từng hàng của ma trận A và xác định các hàng có số lượng số nguyên tố bằng với “max_count”. Danh sách “max_rows” chứa các chỉ số của các hàng này. Cuối cùng, các hàng trong ma trận A có chỉ số được lưu trữ trong danh sách “max_rows” được in ra màn hình bằng vòng lặp for. Với câu h ta sẽ khởi tạo biến “max_len” và

“max_rows”. Biến “max_len” lưu trữ độ dài của chuỗi số lẻ liên tiếp dài nhất đã tìm được, và biến “max_rows” lưu trữ các chỉ số hàng của các hàng có chuỗi số lẻ liên tiếp dài nhất. Sau đó, vòng lặp đầu tiên được sử dụng để duyệt qua từng hàng của ma trận A. Đối với mỗi hàng, vòng lặp thứ hai được sử dụng để duyệt qua từng phần tử của hàng đó. Nếu phần tử hiện tại là một số lẻ, biến “curr_len” được tăng lên 1 và nếu giá trị của “curr_len” lớn hơn giá trị của “max_curr_len”, “max_curr_len” được cập nhật bằng giá trị của “curr_len”. Nếu phần tử hiện tại là một số chẵn, “curr_len” được đặt lại thành 0. Trong cả hai trường hợp, vòng lặp thứ hai tiếp tục duyệt qua các phần tử của hàng đó. Sau khi vòng lặp thứ hai kết thúc, nếu “max_curr_len” lớn hơn “max_len”, biến “max_len” được cập nhật thành “max_curr_len” và “max_rows” được cập nhật với một danh sách chỉ chứa chỉ số hàng hiện tại. Nếu “max_curr_len” bằng “max_len”, chỉ số hàng hiện tại được thêm vào danh sách “max_rows”. Cuối cùng, các hàng trong ma trận A có chỉ số được lưu trữ trong danh sách “max_rows” được in ra màn hình bằng vòng lặp “for”. Mỗi hàng in ra màn hình chứa chuỗi số lẻ liên tiếp dài nhất.

MỤC LỤC

CHƯƠNG 1 – PHƯƠNG PHÁP GIẢI QUYẾT NHIỆM VỤ

- Đoạn mã bắt đầu bằng việc nhập thư viện NumPy với tên định danh là np.
- Sau đó, nó tạo ra ba ma trận A, B và C bằng cách sử dụng hàm np.random.randint().
- Ma trận A có kích thước 10x10 và chứa các số nguyên ngẫu nhiên trong khoảng từ 1 đến 100.
- Ma trận B có kích thước 2x10 và chứa các số nguyên ngẫu nhiên trong khoảng từ 1 đến 20.
- Ma trận C có kích thước 10x2 và chứa các số nguyên ngẫu nhiên trong khoảng từ 1 đến 20.

Task 1d

- Tác vụ tiếp theo là tìm tất cả các số lẻ trong ma trận A và lưu chúng vào danh sách được gọi là odd_numbers.
- Để làm được điều này, mã khởi tạo một danh sách rỗng gọi là odd_numbers.
- Sau đó, nó lặp qua từng phần tử trong ma trận A bằng hai vòng lặp for lồng nhau.
- Nếu một phần tử trong ma trận A là số lẻ, nó được thêm vào danh sách odd_numbers.
- Cuối cùng, danh sách odd_numbers được chuyển đổi thành một mảng NumPy và được in ra màn hình.

Task 1e

- Hàm “is_prime(num)” được định nghĩa để kiểm tra xem một số có phải là số nguyên tố hay không. Nó sử dụng phương pháp kiểm tra xem số đó có chia hết cho một số từ 2 đến căn bậc hai của số đó hay không để xác định xem số đó có phải là số nguyên tố hay không.

- Tiếp theo, vòng lặp được sử dụng để tìm tất cả các số nguyên tố trong ma trận A và lưu chúng vào một vector mới gọi là prime_vector.
- Vòng lặp đầu tiên duyệt qua từng hàng của ma trận A, vòng lặp thứ hai duyệt qua từng cột của ma trận A và sử dụng hàm is_prime(num) để kiểm tra xem giá trị hiện tại có phải là số nguyên tố hay không. Nếu giá trị hiện tại là số nguyên tố, nó được thêm vào vector prime_vector.
- Kết quả của vector prime_vector được in ra màn hình.

Task 1f

- Tính tích của hai ma trận B và C bằng hàm np.dot() trong numpy.
- Tiến hành lặp qua các hàng của ma trận kết quả F và đảo ngược thứ tự các phần tử trong hàng đó nếu hàng đó có chỉ số là số lẻ bằng cách sử dụng phép cắt [::-1] trong vòng lặp for.
- In ma trận kết quả F ra màn hình.

Task 1g

- Hàm is_prime được định nghĩa để kiểm tra một số nguyên có phải là số nguyên tố hay không. Hàm này sử dụng một vòng lặp for để duyệt qua tất cả các số từ 2 đến căn bậc hai của số đó. Nếu số đó chia hết cho một trong các số này, hàm sẽ trả về False, ngược lại trả về True.
- Sau đó, mã tiếp tục với việc tính số lượng số nguyên tố trong mỗi hàng của ma trận A. Việc này được thực hiện bằng cách sử dụng một danh sách đếm (prime_counts) được khởi tạo bằng cách lặp qua mỗi hàng của ma trận A và sử dụng hàm is_prime để đếm số lượng số nguyên tố trong hàng đó.
- Sau đó, chúng ta tìm các hàng có số lượng số nguyên tố lớn nhất bằng cách tìm giá trị lớn nhất trong danh sách đếm và lưu trữ các chỉ số của các hàng có giá trị đó. Cuối cùng, mã in các hàng tương ứng với các chỉ số đó.

Task 1h

- Chương trình tiếp tục với việc định nghĩa một hàm is_prime để kiểm tra xem một số có phải là số nguyên tố hay không.

- Để tìm các hàng trong ma trận A có dãy số lẻ liên tiếp dài nhất, chương trình sử dụng vòng lặp for để lặp qua các hàng của ma trận A. Trong mỗi vòng lặp, chương trình sử dụng một biến curr_len để đếm độ dài của dãy số lẻ liên tiếp hiện tại.
- Nếu số hiện tại là số lẻ, curr_len được tăng lên một đơn vị và kiểm tra xem giá trị này có lớn hơn giá trị của biến max_curr_len hay không. Nếu có, max_curr_len được cập nhật bằng giá trị của curr_len. Nếu số hiện tại là số chẵn, curr_len được đặt lại bằng 0.
- Nếu max_curr_len lớn hơn giá trị hiện tại của max_len, max_len và max_rows được cập nhật để lưu trữ các giá trị tương ứng. Nếu max_curr_len bằng giá trị hiện tại của max_len, chỉ có giá trị của max_rows được cập nhật.
- Cuối cùng, chương trình in ra màn hình các hàng trong ma trận A có dãy số lẻ liên tiếp dài nhất bằng cách lặp qua các giá trị của max_rows và in các hàng tương ứng của ma trận A.

CHƯƠNG 2 – MÃ NGUỒN VÀ KẾT QUẢ ĐẦU RA

522H0120.py A

522H0120.py > ...

```python
import numpy as np
# Task 1
# Tao ma tran A, B va C với cac so nguyen ngau nhien
A = np.random.randint(1, 101, size=(10, 10))
print("Matrix A:\n", A)

B = np.random.randint(1, 21, size=(2, 10))
print("Matrix B:\n", B)

C = np.random.randint(1, 21, size=(10, 2))
print("Matrix C:\n", C)


# Cau a
# Tinh toan theo yeu cau de bai
result1 = A + A.T + np.dot(C, B) + np.dot(B.T, C.T)
# In kết quả ra màn hình
print("Result of a:\n", result1)
```

Source code ma trận A, B, C và câu a.

```
Matrix A:
 [[ 66  13  45  17  51  69  35 100  18  40]
 [ 27   7  88  50  30   7  81  22  47  66]
 [ 39  68  12  72  41  83  22  47  11  97]
 [ 25   1  46  10 100  81  41  40  18  12]
 [ 82  98  11  93  13  38  92   3   7  67]
 [ 51  36  78  37  16  48  92  35  35  81]
 [ 35  50  40  12  24  39  44   5  55  75]
 [ 64  32  36  75  38  71  20  81  11  34]
 [ 81   3  42  24  20  88  25   3  63  67]
 [ 35   2  94  45  87  29   7  47  28  79]]
```

Tạo ma trận A.

```
Matrix B:
 [[18 20  2 10  8  1  6 12  9  2]
 [16  2  6 16  6  8 15 18 14  5]]
Matrix C:
 [[19 15]
 [ 7  5]
 [ 9  6]
 [ 8 20]
 [ 3 20]
 [10  8]
 [20 12]
 [19  1]
 [12  5]
 [20 14]]
```

Tạo ma trận B và C.

```
Result of a:
 [[1296  656  470  936  749  567  961 1020  776  772]
 [ 656  314  392  401  314  306  672  610  433  535]
 [ 470  392  132  440  286  286  318  343  272  363]
 [ 936  401  440  820  727  514  793  777  594  597]
 [ 749  314  286  727  314  345  666  595  460  504]
 [ 567  306  286  514  345  244  427  397  377  302]
 [ 961  672  318  793  666  427  688  610  575  512]
 [1020  610  343  777  595  397  610  654  433  616]
 [ 776  433  272  594  460  377  575  433  482  520]
 [ 772  535  363  597  504  302  512  616  520  378]]
```

In kết quả câu a ra màn hình.

```python
# Cau b
# Tinh toan theo yeu cau de bai
result2 = 0
for i in range(11, 18):
    result2 += np.power(A/i, i-9)
result2 += np.power(A/18, 9)
result2 += np.power(A/19, 10)
result2 += np.power(A/10, 1)
# In ket qua ra man hinh
print("Result of b:\n", result2)
```

Source code câu b.

```
Result of b:
 [[1.44606647e+04 1.17292656e+06 4.26693226e+03 1.91714266e+07
  4.26693226e+03 8.36947167e+00 1.71225209e+06 3.97543528e+05
  3.93520561e+06 2.34703341e+01]
 [6.15224538e+06 9.76040682e+03 2.76078712e+03 5.18930312e+01
  1.93823709e+02 4.36142981e+04 2.21239280e+05 5.51196186e+06
  3.05495065e+04 6.83497590e+02]
 [2.32683564e+07 2.50894552e+02 5.18930299e+04 2.54563046e+04
  2.97779025e+05 1.42364795e+07 2.38313319e-01 1.91714266e+07
  1.03072196e+06 1.39400192e+05]
 [3.93520561e+06 1.33262411e+06 5.27554593e+03 1.33262411e+06
  1.90090548e+05 6.92517140e+05 9.04280625e+05 2.11305399e+07
  9.04280625e+05 1.01282873e+05]
```

```
[2.56940287e+05 3.05412197e+01 1.42364795e+07 3.12365659e+06
 1.08779577e+00 5.18930299e+04 1.57365074e+07 3.12365659e+06
 9.04280625e+05 4.26693226e+03]
[4.16813571e+02 3.12365659e+06 1.91714266e+07 5.51196186e+06
 7.63791899e+06 2.54563046e+04 6.04449717e+05 3.93520561e+06
 1.93823709e+02 2.34703341e+01]
[9.43987454e+06 6.04449717e+05 6.15757328e+04 1.75154217e+04
 1.90090548e+05 1.17292656e+06 1.17292656e+06 4.16813571e+02
 2.56940287e+05 2.77748331e+06]
[1.57365074e+07 6.92517140e+05 1.39705727e+03 3.93366218e-01
 2.21239280e+05 7.63791899e+06 1.93651777e+06 2.77748331e+06
 1.39705727e+03 8.08373189e-01]
[3.23874665e+02 9.04280625e+05 5.07215057e+00 1.43324217e+00
 6.76773187e+01 2.56940287e+05 3.05495065e+04 2.34703341e+01
```

```
[3.23874665e+02 9.04280625e+05 5.07215057e+00 1.43324217e+00
 6.76773187e+01 2.56940287e+05 3.05495065e+04 2.34703341e+01
 3.05412197e+01 4.57991148e+05]
[8.82218690e+01 5.27554593e+03 9.43987454e+06 2.18697607e+06
 3.05412197e+01 5.18930312e+01 1.93651777e+06 1.73775929e+07
 7.28720782e+04 3.08726437e+00]]
```

In kết quả câu b ra màn hình.

```python
# Cau c
# Tao bien mat na de chon cac hang le cua A
mask = np.array([True, False] * 5)
# Chon cac hang le cua A de gan vao bien A_odd
A_odd = A[mask]
# In ket qua ra man hinh
print("The resultant matrix of c:\n", A_odd)
```

Source code câu c.

```
The resultant matrix of c:
 [[ 45  73  39  98  39  14  76  65  83  18]
 [100  27  52  48  63  95   2  98  72  58]
 [ 62  19  95  81   6  52  96  81  71  39]
 [ 91  68  53  46  60  73  73  29  62  80]
 [ 28  71  12   7  22  62  49  18  19  66]]
```

In kết quả câu c ra màn hình.

```
# Cau d
# Khoi tao danh sach trong de luu cac so nguyen le trong A
odd_numbers = []
# Lap qua tung phan tu trong A
for q in range(10):
    for j in range(10):
        if A[q, j] % 2 == 1:
            odd_numbers.append(A[q, j])
# Chuyen doi danh sach thanh mang numpy
odd_numbers = np.array(odd_numbers)
# In ket qua ra man hinh
print("The resultant vector of d:\n", odd_numbers)
```

Source code câu d.

```
The resultant vector of d:
 [45 73 39 39 65 83 87 43 37 21 51 61 49 31 27 63 95 83 69 71 99 71 19 95
 81 81 71 39 29 81 89 83 91 53 73 73 29 69  3 61 89 77  5 71  7 49 19 23
 91 19 21 77 97]
```

In kết quả câu d ra màn hình.

```
# Cau e
# Xac dinh mot ham de kiem tra xem mot so co phai la so nguyen to hay khong
def is_prime(num):
    if num <= 1:
        return False
    for h in range(2, int(np.sqrt(num))+1):
        if num % h == 0:
            return False
    return True


# Tao mot vecto rong de luu tru cac so nguyen to trong ma tran A
prime_vector = []
# Lap qua tung phan tu trong ma tran A va kiem tra xem no co phai la so nguyen to hay khong
for k in range(10):
    for l in range(10):
        if is_prime(A[k, l]):
            prime_vector.append(A[k, l])
# In ket qua ra man hinh
print("The resultant vector of e:\n", prime_vector)
```

Source code câu e.

```
The resultant vector of e:
 [73, 83, 43, 37, 61, 31, 2, 83, 71, 71, 19, 71, 29, 89, 83, 53, 73, 73, 29, 3, 61, 89, 5, 71, 7, 19, 23, 19, 97]
```

In kết quả câu e ra màn hình.

```
# Cau f
# Tao ma tran F bang cach nhan B voi C su dung phuong thuc dot trong thu vien numpy
F = np.dot(C, B)
# Dao nguoc cac phan tu trong cac hang le cua F bang cach su dung phep cat [::-1]
for y in range(1, F.shape[0], 2):
    F[y, :] = F[y, ::-1]
# In ket qua ra man hinh
print("The resultant matrix of f:\n", F)
```

Source code câu f.

```
The resultant matrix of f:
 [[157 376 274 183 276 344 405  93 122 308]
 [290 112  89 375 306 270 181 272 352 155]
 [235 500 415 275 405 405 525 130 155 415]
 [189  71  59 240 187 183 124 187 228 106]
 [147 256 264 173 246 154 255  73  72 218]
 [325 139  93 450 427 255 162 239 404 140]
 [174 464 300 202 312 464 510 110 156 376]
 [125  43  41 150  99 135  94 143 148  80]
 [169 404 295 197 297 369 435 100 131 331]
 [295 111  92 375 293 285 193 291 356 165]]
```

In kết quả câu f ra màn hình.

```
# Cau g
# Xac dinh mot ham de kiem tra xem so do co phai la so nguyen to hay khong
def is_prime(number):
    if number < 2:
        return False
    for e in range(2, int(number ** 0.5) + 1):
        if number % e == 0:
            return False
    return True


# Dem so nguyen to trong moi hang cua A
prime_counts = [sum(is_prime(number) for number in row) for row in A]
# Tim cac hang co so luong so nguyen to toi da
max_count = max(prime_counts)
max_rows = [e for e, count in enumerate(prime_counts) if count == max_count]
# In ket qua ra man hinh
print("Rows with maximum count of prime numbers:")
for t in max_rows:
    print(A[t])
```

Source code câu g.

```
Rows with maximum count of prime numbers:
[87 43 37 21 26 51 61 86 49 31]
[91 68 53 46 60 73 73 29 62 80]
[96 69 34  3 61 89 77 80 34  5]
```

In kết quả câu g ra màn hình.

```
# Cau h
# Tim cac hang co day so le lien ke dai nhat
max_len = 0
max_rows = []

for z in range(A.shape[0]):
    row = A[z]
    curr_len = 0
    max_curr_len = 0
    for v in range(A.shape[1]):
        if row[v] % 2 == 1:
            curr_len += 1
            if curr_len > max_curr_len:
                max_curr_len = curr_len
        else:
            curr_len = 0
    if max_curr_len > max_len:
        max_len = max_curr_len
        max_rows = [z]
    elif max_curr_len == max_len:
        max_rows.append(z)

# In ket qua ra man hinh
print("Rows with longest contiguous odd numbers sequence:")
for row in max_rows:
    print(A[row])
```

Source code câu h.

```
Rows with longest contiguous odd numbers sequence:
[87 43 37 21 26 51 61 86 49 31]
[83 74 40 74 60 69 71 99 71 56]
[96 69 34  3 61 89 77 80 34  5]
[23 40 91 78 19 21 77 97 54 10]
```

In kết quả câu h ra màn hình.