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

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

Xử Lí Tính Toán Các Ma Trận Và Vector Bằng Ngôn Ngữ Python

Người hướng dẫn: **TS Nguyễn Thị Diễm Hằng**

Người thực hiện: **Võ Kim Long - 52200226**

Lớp **: 22050301**

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 MÔN ĐẠI SỐ TUYẾN TÍNH CHO CÔNG NGHỆ THÔNG TIN

Xử Lí Tính Toán Các Ma Trận Và Vector Bằng Ngôn Ngữ Python

Người hướng dẫn: **TS Nguyễn Thị Diễm Hằng**
Người thực hiện: **Võ Kim Long**
Lớp **: 22050301**
Khoá **: 26**

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2023

LỜI CẢM ƠN

Xuyên suốt quá trình thực hiện bài tiểu luận môn Đại số tuyến tính cho Công nghệ thông tin, em xin gửi lời cảm ơn chân thành đến cô Nguyễn Thị Diễm Hằng đã tận tình hướng dẫn và trao dồi cho em những kiến thức thật bổ ích. Em rất vui vì có thể đồng hành cùng cô trong học kì vừa qua, sự nhiệt huyết trong giảng dạy và vui vẻ của cô là động lực để em cố gắng hoàn thành bài tiểu luận này một cách tốt nhất. Chân thành cảm ơn cô.

ĐỒ ÁN ĐƯỢC HOÀN THÀNH
TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG

Tôi xin cam đoan đây là sản phẩm đồ án của riêng tôi và được sự hướng dẫn của TS Nguyễn Thị Diễm Hằng;. Các nội dung nghiên cứu, kết quả trong đề tài này là trung thực và chưa công bố dưới bất kỳ hình thức nào trước đây. Những số liệu trong các bảng biểu phục vụ cho việc phân tích, nhận xét, đánh giá được chính tác giả thu thập từ các nguồn khác nhau có ghi rõ trong phần tài liệu tham khảo.

Ngoài ra, trong đồ án còn sử dụng một số nhận xét, đánh giá cũng như số liệu của các tác giả khác, cơ quan tổ chức khác đều có trích dẫn và chú thích nguồn gốc.

Nếu phát hiện có bất kỳ sự gian lận nào tôi xin hoàn toàn chịu trách nhiệm về nội dung đồ án 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 13 tháng 4 năm 2023

Tác giả

(ký tên và ghi rõ họ tên)

Võ Kim Long

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

Bài tiểu luận môn Đại số tuyến tính cho Công nghệ thông tin nghiên cứu về việc thao tác với các ma trận và vector bằng phép toán và các phép biến đổi, chọn lọc ra các phần tử mong muốn. Đây là một lĩnh vực rất phổ biến trong xử lí và phân tích dữ liệu của Công nghệ thông tin.

Các hướng giải quyết vấn đề trên gồm thực hiện bằng ngôn ngữ Python và sử dụng các hàm có sẵn của thư viện math, numpy, sympy, matplotlib và xây dựng một số hàm tự định nghĩa như hàm kiểm tra số nguyên tố, hàm tìm số lớn nhất của mảng,.. để thực phép toán cộng trừ nhân chia với ma trận, tìm ma trận chuyển vị, lũy thừa ma trận, chọn ra phần tử thuộc ma trận là số nguyên tố hay số lẻ và lưu vào vector mới theo yêu cầu bài toán.

Kết quả đạt được là có thể làm việc với ma trận và vector một cách nhanh chóng, dễ dàng, thành thạo hơn trong việc sử dụng các hàm có sẵn, từ đó tăng đáng kể hiệu suất xử lí bài toán và có cơ hội tiếp cận với các thuật toán phức tạp.

Cuối cùng, một số phát hiện cơ bản khi hoàn thành bài tiểu luận như ma trận và vector là định nghĩa quan trọng của Đại số tuyến tính, đặc biệt có mối quan hệ mật thiết với mảng một chiều, mảng hai chiều dùng để lưu trữ dữ liệu và tính toán, phân tích trong việc lập trình. Đây là những kiến thức được áp dụng rất nhiều khi học tập và làm việc, đặc biệt là chuyên ngành khoa học dữ liệu.

MỤC LỤC

LỜI CẢM ƠN i
ĐỒ ÁN ĐƯỢC HOÀN THÀNH ii
TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG ii
PHẦN XÁC NHẬN VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN iii
TÓM TẮT iv
MỤC LỤC 1
CHƯƠNG 1 – PHƯƠNG PHÁP GIẢI QUYẾT BÀI TOÁN 3
1.1 Cách thước sử dụng để xử lí các câu 1d, 1e, 1f, 1g và 1h 3
1.1.1 Phương pháp giải quyết câu 1d 3
1.1.2 Phương pháp giải quyết câu 1e 4
1.1.3 Phương pháp giải quyết câu 1f 5
1.1.4 Phương pháp giải quyết câu 1g 6
1.1.5 Phương pháp giải quyết câu 1h 7
1.2 Nội dung của chương này 9
CHƯƠNG 2 – MÃ NGUỒN VÀ KẾT QUẢ CHƯƠNG TRÌNH 10
2.1 Hình ảnh mã nguồn khởi tạo ma trận A, B và C theo yêu cầu bài toán 10
2.2 Hình ảnh mã nguồn và kết quả của câu 1a 10
2.3 Hình ảnh mã nguồn và kết quả của câu 1b 11
2.4 Hình ảnh mã nguồn và kết quả của câu 1c 13
2.5 Hình ảnh mã nguồn và kết quả của câu 1d 13
2.6 Hình ảnh mã nguồn và kết quả của câu 1e 14
2.7 Hình ảnh mã nguồn và kết quả của câu 1f 15
2.8 Hình ảnh mã nguồn và kết quả của câu 1g 16
2.9 Hình ảnh mã nguồn và kết quả của câu 1h 18
TÀI LIỆU THAM KHẢO 20

DANH MỤC CÁC HÌNH ẢNH

DANH MỤC HÌNH

Hình 2.1: Khởi tạo ma trận A, B và C. 10
Hình 2.2.1: Hình ảnh mã nguồn câu 1a.....10
Hình 2.2.2: Hình ảnh kết quả câu 1a.....10
Hình 2.3.1: Hàm calSequence để tính tổng cho câu 1b. 11
Hình 2.3.2: Hình ảnh mã nguồn câu 1b. 11
Hình 2.3.3: Hình ảnh kết quả câu 1b.....12
Hình 2.4.1: Hình ảnh mã nguồn câu 1c.....13
Hình 2.4.2: Hình ảnh kết quả câu 1c. 13
Hình 2.5.1: Hình ảnh mã nguồn câu 1d. 13
Hình 2.5.2: Hình ảnh vector kết quả câu 1d.....14
Hình 2.6.1: Hàm isPrime để kiểm tra số n nhận vào có phải là số nguyên tố. 14
Hình 2.6.2: Hình ảnh mã nguồn câu 1e.....14
Hình 2.6.3: Hình ảnh vector kết quả câu 1e.....15
Hình 2.7.1: Hình ảnh mã nguồn câu 1f.....15
Hình 2.7.2: Hình ảnh kết quả câu 1f. 15
Hình 2.8.1: Hàm isPrime để kiểm tra số n nhận vào có phải là số nguyên tố. 16
Hình 2.8.2: Hàm findMaxArr để trả về số lượng nhất trong mảng. 16
Hình 2.8.3: Hình ảnh mã nguồn câu 1g. 17
Hình 2.8.4: Hình ảnh kết quả câu 1g.....17
Hình 2.9.1: Hàm findMaxArr để trả về số lượng nhất trong mảng. 18
Hình 2.9.2: Hình ảnh mã nguồn câu 1h. 18
Hình 2.9.3: Hình ảnh kết quả câu 1h.....19

CHƯƠNG 1 – PHƯƠNG PHÁP GIẢI QUYẾT BÀI TOÁN

1.1 Cách thước sử dụng để xử lí các câu 1d, 1e, 1f, 1g và 1h

- Trước tiên, dùng lệnh **import numpy as np** để sử dụng thư viện numpy có sẵn của ngôn ngữ Python.

- Dùng hàm **np.random.randint** để khởi tạo các ma trận A, B, và C theo yêu cầu bài toán.

1.1.1 Phương pháp giải quyết câu 1d

Yêu cầu 1d: Hãy lưu những phần tử là số lẻ của ma trận A vào một vector mới, in vector kết quả đó ra màn hình.

Mô tả chi tiết cách giải:

- Khởi tạo một danh sách rỗng để lưu những phần tử là số lẻ của ma trận A.

- **Khởi tạo vòng lặp for** để duyệt qua các dòng của ma trận A:

+ Vòng lặp có biến i chạy từ 0 đến len(A): tức i chạy từ index = 0 đến index = số dòng của ma trận A trừ 1.

- **Trong vòng lặp đó, khởi tạo thêm vòng lặp for** để duyệt qua từng cột của mỗi dòng:

+ Vòng lặp này có biến j chạy từ 0 đến len(A[i]): tức j chạy từ index = 0 đến index = số cột của từng dòng trừ 1.

- Như vậy, vòng lặp sẽ duyệt qua từng phần tử của mỗi dòng.

- Khi duyệt qua từng phần tử, xem xét điều kiện **if** nếu phần tử đó chia lấy dư cho 2 có bằng 1 bằng toán tử **%** (vì cần tìm số lẻ) :

+ Nếu đúng: dùng lệnh **append** để thêm phần tử đó vào danh sách đã khởi tạo trước đó, sau đó duyệt qua phần tử tiếp theo.

+ Nếu sai: duyệt qua phần tử tiếp theo.

- Khi các số lẻ của ma trận A đã được thêm vào danh sách, dùng lệnh **np.array** để biến danh sách thành vector kết quả.

- Cuối cùng, dùng lệnh **print** để in vector kết quả ra màn hình.

1.1.2 Phương pháp giải quyết câu 1e

Yêu cầu 1e: Hãy lưu những phần tử là số nguyên tố của ma trận A vào một vector mới, in vector kết quả đó ra màn hình.

Mô tả chi tiết cách giải:

- Viết hàm bằng từ khóa **def** để **kiểm tra có phải số nguyên tố** với tham số là n, nếu n < 1 hàm trả về False, nếu n = 2 hàm trả về True, nếu n > 2 chạy vòng lặp **for** và xét điều kiện nếu n chia hết bằng toán tử **%,** nếu chia hết hàm trả về False, ngược lại nếu chạy hết vòng lặp mà không chia hết, hàm sẽ trả về True.

- **Khởi tạo một danh sách rỗng** lưu những số nguyên tố của ma trận A.

- **Khởi tạo vòng lặp for** để duyệt qua các dòng của ma trận A:

+ Vòng lặp có biến i chạy từ 0 đến len(A): tức i chạy từ index = 0 đến index = số dòng của ma trận A trừ 1.

- **Trong vòng lặp đó, khởi tạo thêm vòng lặp for** để duyệt qua từng cột của mỗi dòng:

+ Vòng lặp này có biến j chạy từ 0 đến len(A[i]): tức j chạy từ index = 0 đến index = số cột của dòng trừ 1.

- Như vậy, vòng lặp sẽ duyệt qua từng phần tử của mỗi dòng.

- Khi duyệt qua từng phần tử, **truyền phần tử vào hàm kiểm tra số nguyên tố**:

+ Nếu hàm trả về True: dùng lệnh **append** thêm phần tử đó vào danh sách đã khởi tạo trước đó và duyệt qua phần tử tiếp theo.

+ Nếu hàm trả về False: duyệt qua phần tử tiếp theo.

- Khi các số nguyên tố của ma trận A đã được thêm vào danh sách, dùng lệnh **np.array** để biến danh sách thành vector kết quả.

- Cuối cùng, dùng lệnh **print** để in vector kết quả ra màn hình.

1.1.3 Phương pháp giải quyết câu 1f

Yêu cầu 1f: Cho ma trận D = CB, hãy đảo ngược các phần tử ở dòng lẻ ma trận D, in ma trận kết quả ra màn hình **(Chú thích: Dòng lẻ là dòng bắt đầu từ index = 0).**

Mô tả chi tiết cách giải:

- Dùng lệnh **np.dot(C, B)** để tìm ma trận D.

- **Khởi tạo vòng lặp for** duyệt qua từng dòng của ma trận D, **bước nhảy bằng 2** (vì cần dòng lẻ):

  + Tức chỉ duyệt qua các dòng có index 0, 2, 4, 6, 8 vì dòng 1 tương ứng với index 0, dòng 3 tương ứng với index 2, dòng 5 tương ứng với index 4, dòng 7 tương ứng với index 6 và dòng 9 tương ứng với index 8.

- **Trong vòng lặp đó, khởi tạo thêm vòng lặp for** chỉ duyệt đến cột giữa của mỗi dòng:

  + Vòng lặp có j chạy từ index = 0 đến index bằng số cột của mỗi dòng chia 2 trừ 1.

- Như vậy, vòng lặp sẽ duyệt qua từng phần tử của những dòng lẻ.

- Bắt đầu đảo ngược các phần tử:

  + **Khởi tạo biến tạm temp.**

  + Gán biến tạm temp bằng phần tử D[i][j], cho phần tử D[i][j] bằng phần tử D[i][len(D[i] – 1 – j), và gán lại phần tử D[i][len(D[i]) – 1 – j) bằng biến temp.

  + Ví dụ: với i = 0 (tức dòng index = 0):

    j = 0: Phần tử 0 đổi phần tử 9

    j = 1: Phần tử 1 đổi phần tử 8

    j = 2: Phần tử 2 đổi phần tử 7

    j = 3: Phần tử 3 đổi phần tử 6

    j = 4: Phần tử 4 đổi phần tử 5 (**Đảo ngược xong, vì vậy vòng lặp chỉ cần duyệt đến cột giữa của dòng**).

+ Tương tự với các dòng còn lại.

- Cuối cùng, dùng lệnh **print** để in ma trận kết quả.

1.1.4 Phương pháp giải quyết câu 1g

Yêu cầu 1g: Dựa vào ma trận A, hãy tìm những dòng nào có số lượng tối đa số nguyên tố, in những dòng đó ra màn hình.

Mô tả chi tiết cách giải:

- **Viết hàm** bằng từ khóa **def** để **kiểm tra có phải số nguyên tố** với tham số là n, nếu n < 1 hàm trả về False, nếu n = 2 hàm trả về True, nếu n > 2 chạy vòng lặp **for** và xét điều kiện nếu n chia hết bằng toán tử **%,** nếu chia hết hàm trả về False, ngược lại nếu chạy hết vòng lặp mà không chia hết, hàm sẽ trả về True.

- **Viết thêm hàm** bằng từ khóa **def** để **trả về số lớn nhất của mảng** với tham số nhận vào là một mảng, chạy vòng lặp **for** để so sánh từng cặp phần tử liền kề của mảng xem số nào lớn hơn, và gán vào biến max, cuối cùng sẽ trả về biến max.

- **Khởi tạo một danh sách rỗng** để lưu số lượng số nguyên tố của từng dòng.

- **Khởi tạo vòng lặp for** để duyệt qua các dòng của ma trận A:

+ Vòng lặp có biến i chạy từ 0 đến len(A): tức i chạy từ index = 0 đến index = số dòng của ma trận A trừ 1.

- **Trong vòng lặp đó, khởi tạo biến đếm tên count** và gán giá trị bằng 0 để đếm số nguyên tố cũng như **khởi tạo thêm vòng lặp for** để duyệt qua từng cột của mỗi dòng:

- Như vậy, vòng lặp sẽ duyệt qua từng phần tử của mỗi dòng.

- Khi duyệt qua từng phần tử, **truyền phần tử đó vào hàm kiểm tra số nguyên tố**:

+ Bắt đầu từ dòng đầu tiên, truyền lần lượt phần tử vào hàm kiểm tra số nguyên tố trước đó.

+ Nếu hàm trả về True: biến đếm sẽ cộng thêm 1.

+ Nếu hàm trả về False: duyệt qua phần tử tiếp theo.

+ Khi đếm xong một dòng, dùng lệnh **append lưu giá trị của biến đếm đó vào danh sách đã khởi tạo trước đó**, và biến đếm quay lại giá trị bằng 0.

+ Tiếp tục tương tự với các dòng khác.

- **Khởi tạo biến tên maxPrime** để lưu số lượng số nguyên tố nhiều nhất.

- **Truyền danh sách chứa số lượng số nguyên tố của từng dòng vào hàm tìm số lớn nhất** để tìm **bao nhiêu số nguyên tố là lớn nhất** và gán cho biến maxPrime.

- **Khởi tạo vòng lặp for** để duyệt qua từng phần tử của danh sách, **tìm xem phần tử tại index thứ mấy của danh sách bằng với giá trị biến maxPrime.**

- Khi đã có index, đó cũng là số thứ tự của dòng có số lượng số nguyên tố nhiều nhất (**vì vị trí phần tử tương ứng với vị trí của dòng do chúng được lưu lần lượt**).

- Dùng lệnh **print trong vòng lặp** để lần lượt in ra các dòng có số lượng số nguyên tố nhiều nhất.

1.1.5 Phương pháp giải quyết câu 1h

Yêu cầu 1h: Dựa vào ma trận A, hãy tìm những dòng nào có số số lượng số lẻ liên tục nhiều nhất, in những dòng đó ra màn hình.

Mô tả chi tiết cách giải:

- **Viết hàm** bằng từ khóa **def** để **trả về số lớn nhất của mảng** với tham số nhận vào là một mảng, chạy vòng lặp **for** để so sánh từng cặp phần tử liền kề của mảng xem số nào lớn hơn, và gán vào biến max, cuối cùng sẽ trả về biến max.

- **Khởi tạo danh sách rỗng tên là lst4** lưu số lượng số lẻ liên tục của từng dòng.

- **Khởi tạo vòng lặp for** để duyệt qua các dòng của ma trận A:

+ Vòng lặp có biến i chạy từ 0 đến len(A): tức i chạy từ index = 0 đến index = số dòng của ma trận A trừ 1.

- **Trong vòng lặp đó, khởi tạo biến đếm** bằng 0, **khởi tạo danh sách rỗng khác tên là temp** và **khởi tạo thêm vòng lặp for** để duyệt qua từng cột của mỗi dòng:

- Như vậy, vòng lặp sẽ duyệt qua từng phần tử của mỗi dòng:

- Khi duyệt qua từng phần tử của dòng, sẽ xem xét các điều kiện bằng lệnh **if, else**:

+ (Lệnh **if**) Nếu phần tử đó là số lẻ: biến đếm cộng thêm 1, và nếu duyệt đến phần tử cuối cùng của dòng thì biến đếm được thêm luôn vào danh sách temp. **Điều này để tránh trường hợp nếu các phần tử liền kề đang lẻ liên tục và đến phần tử cuối cùng vẫn là số lẻ, biến đếm sẽ không được thêm vào danh sách (do không nhảy qua lệnh else).**

+ (Lệnh **else**) Nếu không phải số lẻ: nếu biến đếm lớn hơn 0 thì thêm biến đếm vào danh sách temp (chứng tỏ có xuất hiện số lẻ thì mới được thêm vào danh sách), sau đó cho biến đếm quay lại bằng 0.

+ Khi đã đếm xong một dòng, **truyền danh sách temp vào hàm tìm số lớn nhất của mảng** để xem dòng đó có số lượng số lẻ liên tiếp bao nhiêu là lớn nhất, và truyền giá trị đó vào danh sách rỗng lst4 ban đầu bằng hàm **append**.

+ Biến đếm quay lại bằng 0, danh sách temp quay lại rỗng

+ Tương tự các dòng còn lại.

- **Khởi tạo biến maxContOdd** để lưu số lượng số lẻ liên tiếp nhiều nhất.

- Sau khi đã đến từng dòng xong, **truyền danh sách ban đầu lst4** chứa số lượng số lẻ liên tiếp nhiều nhất của từng dòng **vào hàm tìm số lớn nhất và lưu giá trị trả về vào biến tên maxContOdd.**

- **Khởi tạo vòng lặp for** để duyệt qua từng phần tử của danh sách lst4, tìm xem **phần tử tại index thứ mấy của danh sách bằng với giá trị biến maxContOdd.**

- Khi đã có index, đó cũng là số thứ tự của dòng có số lượng số lẻ liên tiếp nhiều nhất (**vì vị trí biến đếm tương ứng với vị trí của dòng do chúng được lưu lần lượt**).

- Dùng lệnh **print** trong vòng lặp để lần lượt in ra các cột có số lượng số nguyên tố nhiều nhất.

1.2 Nội dung của chương này

Chương này trình bày chi tiết các bước giải bài toán của các câu 1d, 1e, 1f, 1g và 1h từ việc khởi tạo danh sách, vòng lặp để duyệt qua từng phần tử, viết hàm, thêm phần tử vào danh sách,..

Đây là những cách làm quan trọng và cơ bản để người mới bắt đầu học lập trình có thể làm quen với việc xử lí ma trận và vector cũng như nâng cao tư duy và khả năng nhạy bén, từ đó áp dụng vào những bài toán thực tế nâng cao hơn, đặc biệt là ở mảng xử lí và phân tích dữ liệu.

CHƯƠNG 2 – MÃ NGUỒN VÀ KẾT QUẢ CHƯƠNG TRÌNH

Tổng quan: Chương này sẽ chứa các hình ảnh mã nguồn của từng bài toán và kết quả đầu ra tương ứng ở "Phần lập trình".

2.1 Hình ảnh mã nguồn khởi tạo ma trận A, B và C theo yêu cầu bài toán

```
# Initialize matrix A, B and C regarding to essay's requirement
A = np.random.randint(1, 101, size = (10, 10))
B = np.random.randint(1, 21, size = (2, 10))
C = np.random.randint(1, 21, size = (10, 2))
```

Hình 2.1: Khởi tạo ma trận A, B và C với các phần tử số nguyên và kích thước ngẫu nhiên bằng hàm np.random.randint .

2.2 Hình ảnh mã nguồn và kết quả của câu 1a

```
# Question a:
a =  A + A.T + np.dot(C,B) + np.dot(B.T, C.T)
print("a. ", a)

print()
```

Hình 2.2.1: Hình ảnh mã nguồn câu 1a. Dùng lệnh ".T" để tìm ma trận chuyển vị và np.dot để nhân hai ma trận.

```
a.  [[604 299 269 539 603 208 599 563 571 552]
 [299 196 370 239 503 251 348 405 316 530]
 [269 370 378 541 550 234 486 674 360 520]
 [539 239 541 440 649 437 481 659 535 665]
 [603 503 550 649 854 390 737 808 668 950]
 [208 251 234 437 390 144 314 362 135 400]
 [599 348 486 481 737 314 532 726 466 812]
 [563 405 674 659 808 362 726 752 558 998]
 [571 316 360 535 668 135 466 558 606 673]
 [552 530 520 665 950 400 812 998 673 886]]
```

Hình 2.2.2: Hình ảnh kết quả câu 1a.

2.3 Hình ảnh mã nguồn và kết quả của câu 1b

```
def calSequence (value, n): # Function to return sequence value of question b
    res = value
    for i in range (1, n):
      res = np.dot(res, value)
    return res
```

Hình 2.3.1: Hàm calSequence để tính tổng cho câu 1b.

```
# Question b:
b = 0
for i in range (1, 11):
  temp = A / (i + 9)
  b += calSequence(temp, i)
print("b. ", b)

print()
```

Hình 2.3.2: Hình ảnh mã nguồn câu 1b. Khởi tạo vòng lặp và truyền biến temp vào hàm calSequence (hình 2.3.1) rồi tính tổng.

```
b.  [[3.22312636e+13 2.54062430e+13 2.85298088e+13 2.74856576e+13
  1.97614354e+13 2.55415426e+13 2.43383571e+13 3.19319063e+13
  3.31143604e+13 2.24068351e+13]
 [2.43296193e+13 1.91777869e+13 2.15356025e+13 2.07474041e+13
  1.49168502e+13 1.92799286e+13 1.83716919e+13 2.41036442e+13
  2.49962133e+13 1.69137216e+13]
 [2.35206331e+13 1.85401037e+13 2.08195138e+13 2.00575438e+13
  1.44208301e+13 1.86388384e+13 1.77608173e+13 2.33021793e+13
  2.41650686e+13 1.63513011e+13]
 [2.55541916e+13 2.01430533e+13 2.26195383e+13 2.17916876e+13
  1.56676348e+13 2.02503253e+13 1.92963898e+13 2.53168493e+13
  2.62543413e+13 1.77650103e+13]
 [3.05453281e+13 2.40773109e+13 2.70374982e+13 2.60479311e+13
  1.87277924e+13 2.42055464e+13 2.30652746e+13 3.02616221e+13
  3.13822232e+13 2.12348239e+13]
 [2.74373087e+13 2.16274199e+13 2.42864051e+13 2.33975272e+13
  1.68222198e+13 2.17426064e+13 2.07183600e+13 2.71824712e+13
  2.81890490e+13 1.90741581e+13]
 [2.15744874e+13 1.70060545e+13 1.90968585e+13 1.83979489e+13
  1.32276103e+13 1.70966159e+13 1.62912508e+13 2.13741098e+13
  2.21656045e+13 1.49983478e+13]
 [2.03756878e+13 1.60611047e+13 1.80357336e+13 1.73756504e+13
  1.24926167e+13 1.61466360e+13 1.53860175e+13 2.01864438e+13
  2.09339570e+13 1.41649640e+13]
 [2.27627558e+13 1.79427065e+13 2.01486713e+13 1.94112506e+13
  1.39561693e+13 1.80382644e+13 1.71885285e+13 2.25513361e+13
  2.33864244e+13 1.58244365e+13]
 [3.01023187e+13 2.37281108e+13 2.66453686e+13 2.56701414e+13
  1.84561887e+13 2.38544920e+13 2.27307508e+13 2.98227251e+13
  3.09270751e+13 2.09268627e+13]]
```

Hình 2.3.3: Hình ảnh kết quả câu 1b.

2.4 Hình ảnh mã nguồn và kết quả của câu 1c

```
# Question c:
c = A[0:10:2] # Know that odd rows start with index = 0
print("c.", c)

print()
```

Hình 2.4.1: Hình ảnh mã nguồn câu 1c. Dùng hàm cắt bắt đầu bằng 0, dừng lại bằng 9 và bước nhảy bằng 2 **(Chú thích: dòng lẻ là dòng bắt đầu từ index = 0).**

```
c. [[ 73.  33.  66.  86.  28.  75.  82.  88.  91.  35.]
 [ 32.  36.  48.  91.   6.  64.  15.  98.  84.  17.]
 [ 79.  53.  97.  31.  40.  64.  44.   5.  85.  92.]
 [100.  70.  62.  31.   8.  35.  38.  84.   2.   3.]
 [ 23.  43.  65.  24.  18.   3.  61.  80.  95.  66.]]
```

Hình 2.4.2: Hình ảnh kết quả câu 1c.

2.5 Hình ảnh mã nguồn và kết quả của câu 1d

```
# Question d:
lst1 = []
for i in range (len(A)):
    for j in range (len(A[i])):
        if A[i][j] % 2 == 1:
            lst1.append(A[i][j])
d = np.array(lst1)
print("d. ", d)

print()
```

Hình 2.5.1: Hình ảnh mã nguồn câu 1d.

```
d.  [73 33 75 91 35 91 97 25 59 91 15 17  5 29 17 95 13 79 53 97 31  5 85 69
 39 65 55 63 31 35  3 21 71 77  7 51 13 23 43 65  3 61 95 71  3 91 39 99
 85]
```

Hình 2.5.2: Hình ảnh vector kết quả câu 1d.

2.6 Hình ảnh mã nguồn và kết quả của câu 1e

```
def isPrime (n): # Function to check whether the number is prime number
    if n == 1:
        return False
    elif n == 2:
        return True
    else:
        for i in range(2, n):
            if (n % i == 0):
                return False
        return True
```

Hình 2.6.1: Hàm isPrime để kiểm tra số n nhận vào có phải là số nguyên tố.

```
# Question e:
lst2 = []
for i in range(len(A)):
    for j in range(len(A[i])):
        if isPrime(A[i][j]):
            lst2.append(A[i][j])
e = np.array(lst2)
print("e. ", e)

print()
```

Hình 2.6.2: Hình ảnh mã nguồn câu 1e. Truyền từng phần tử vào hàm isPrime để kiểm tra số nguyên tố và thêm vào danh sách bằng lệnh append.

```
e.  [73  2 97 59 17  5 29 17 13 79 53 97 31  5 31  2  3 71  7 13 23 43  3 61
 71  3]
```

Hình 2.6.3: Hình ảnh vector kết quả câu 1e.

2.7 Hình ảnh mã nguồn và kết quả của câu 1f

```
# Question f:
D = np.dot(C, B)
for i in range(0, len(D), 2): # Know that odd rows start with index = 0
    for j in range(0, int(len(D[i]) / 2)):
        temp = D[i][j]
        D[i][j] = D[i][len(D[i]) - j - 1]
        D[i][len(D[i]) - j - 1] = temp
print("f. ", D)

print()
```

Hình 2.7.1: Hình ảnh mã nguồn câu 1f. Nhân hai ma trận C và B bằng hàm np.dot để tìm ma trận D, sau đó khởi tạo vòng lặp duyệt qua ma trận D để đảo ngược phần tử ở các dòng lẻ **(Chú thích: dòng lẻ là dòng bắt đầu từ index = 0).**

```
f.  [[297 164  86  69  43 239  74  88  19 229]
 [156  66 246  78 276 192 180 222 186 324]
 [179 103 127 103 111 153  43 141  38  83]
 [287 107 402 136 477 309 295 364 322 563]
 [461 262 268 217 219 387 112 294  77 257]
 [ 21   6  23   9  31  17  17  21  21  37]
 [564 318 282 228 216 468 138 306  78 348]
 [368  98 378 154 528 276 280 346 358 632]
 [377 208 106  85  51 303  94 108  23 293]
 [166  76 282  86 306 222 206 254 206 358]]
```

Hình 2.7.2: Hình ảnh kết quả câu 1f sau khi đã đảo ngược các phần tử ở dòng lẻ.

2.8 Hình ảnh mã nguồn và kết quả của câu 1g

```
def isPrime (n): # Function to check whether the number is prime number
  if n == 1:
    return False
  elif n == 2:
    return True
  else:
    for i in range(2, n):
      if (n % i == 0):
        return False
    return True
```

Hình 2.8.1: Hàm isPrime để kiểm tra số n nhận vào có phải là số nguyên tố.

```
def findMaxArr(a): # Function to return max value in array
  if len(a) == 0:
    return 0
  max = a[0]
  for ele in a:
    if ele > max:
      max = ele
  return max
```

Hình 2.8.2: Hàm findMaxArr để trả về số lượng nhất trong mảng.

```python
# Question g:
lst3 = []
for i in range(0, len(A)):
    count = 0
    for j in range(0, len(A[i])):
        if isPrime(A[i][j]):
            count += 1
    lst3.append(count)
maxPrime = findMaxArr(lst3)
print("g. ", end = "")
for i in range (0, len(lst3)):
    if lst3[i] == maxPrime:
        print(A[i])

print()
```

Hình 2.8.3: Hình ảnh mã nguồn câu 1g. Kiểm tra từng phần tử bằng hàm isPrime và đếm chúng lưu vào mảng lst3, tìm giá trị lớn nhất của lst3 bằng hàm findMaxArr và in các dòng tương ứng.

```
g. [79 53 97 31 40 64 44  5 85 92]
```

Hình 2.8.4: Hình ảnh kết quả câu 1g.

2.9 Hình ảnh mã nguồn và kết quả của câu 1h

```
def findMaxArr(a): # Function to return max value in array
    if len(a) == 0:
        return 0
    max = a[0]
    for ele in a:
        if ele > max:
            max = ele
    return max
```

Hình 2.9.1: Hàm findMaxArr để trả về số lượng nhất trong mảng.

```
# Question h:
lst4 = []
for i in range (0, len(A)):
    count = 0
    temp = []
    for j in range(0, len(A[i])):
        if A[i][j] % 2 == 1:
            count += 1
            if j == len(A[i]) - 1:
                temp.append(count)
        else:
            if(count > 0):
                temp.append(count)
            count = 0
    lst4.append(findMaxArr(temp))
maxContOdd = findMaxArr(lst4)
print("h. ", end = "")
for i in range(0, len(lst4)):
    if lst4[i] == maxContOdd:
        print(A[i])
```

Hình 2.9.2: Hình ảnh mã nguồn câu 1h. Đếm số lượng số lẻ liên tiếp của từng dòng và lưu vào danh sách lst4, tìm giá trị lớn nhất của lst4 bằng hàm findMaxArr và in các dòng tương ứng.

```
h. [79 53 97 31 40 64 44  5 85 92]
[21 60 71 77  7 51 80 30 14 13]
```

Hình 2.9.3: Hình ảnh kết quả câu 1h.

TÀI LIỆU THAM KHẢO

1. https://numpy.org/doc/stable/reference/generated/numpy.ndarray.T.html
2. https://numpy.org/doc/stable/reference/random/generated/numpy.random.randint.html
3. https://www.geeksforgeeks.org/numpy-dot-python/
4. https://www.w3schools.com/python/ref_list_append.asp
5. https://datagy.io/numpy-zeros/