

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



TIỂU LUẬN CUỐI KÌ MÔN ĐẠI SỐ TUYẾN TÍNH ỨNG
DỤNG CHO CÔNG NGHỆ THÔNG TIN

Người hướng dẫn: **PHẠM QUỐC DUY**

Người thực hiện: **LƯU HỮU TRÍ**

Lớp : **22050301**

Khoá : **26**

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2023

LỜI CẢM ƠN

Lời đầu tiên, em xin trân trọng cảm ơn quý thầy cô Khoa Công nghệ Thông tin trường Đại học Tôn Đức Thắng đã tạo điều kiện thuận lợi về cơ sở vật chất, cũng như hỗ trợ về tin thần trong việc hoàn thành tiểu luận.

Đặc biệt, em xin chân thành cảm ơn thầy Phạm Quốc Duy đã trực tiếp giảng dạy tận tình, chi tiết để có đủ kiến thức trong quá trình học và giúp đỡ, hỗ trợ em khắc phục những khó khăn, thiếu sót trong suốt quá trình hoàn thành bài tiểu luận.

Em xin chân thành cảm ơn!

TIỂU LUẬN 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 Thầy Phạm Quốc Duy. 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.

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 29 tháng 4 năm 2023

Tác giả

(ký tên và ghi rõ họ tên)

Lưu Hữu Trí

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

Sử dụng các kiến thức đã học trong môn Đại số tuyến tính và kết hợp với sử dụng ngôn ngữ lập trình Python để giải quyết các bài toán. Thông qua đó cải thiện được kỹ năng, tư duy lập trình và củng cố các kiến thức lý thuyết từ môn học.

Tiểu luận giữa kì môn Đại số tuyến tính ứng dụng cho Công nghệ Thông tin, ngoài các phần: lời cảm ơn, lời cam kết, mục lục, tài liệu tham khảo... tiểu luận gồm 3 chương:

- Chương 1: Cơ sở lý thuyết
- Chương 2: Thuật toán giải quyết các yêu cầu
- Chương 3: Source code và kết quả

MỤC LỤC

LỜI CẢM ƠN	2
PHẦN XÁC NHẬN VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN	4
TÓM TẮT.....	5
MỤC LỤC	6
CHƯƠNG 1 – CƠ SỞ LÝ THUYẾT.....	7
1.1 Thư viện NumPy	7
1.2 Các hàm và phương thức built-in trong Python	8
CHƯƠNG 2 – THUẬT TOÁN GIẢI QUYẾT CÁC YÊU CẦU	8
2.1 Câu 1d	8
2.2 Câu 1e	9
2.3 Câu 1f	10
2.4 Câu 1g	10
2.5 Câu 1h	11
CHƯƠNG 3 – SOURCE CODE VÀ KẾT QUẢ	13
3.1 Khai báo thư viện và input.....	13
3.2 Câu 1d.....	14
3.3 Câu 1e.....	15
3.4 Câu 1f.....	15
3.5 Câu 1g.....	17
3.6 Câu 1h	17
TÀI LIỆU THAM KHẢO	19

CHƯƠNG 1 – CƠ SỞ LÝ THUYẾT

1.1 Thư viện NumPy

1.1.1 Giới thiệu chung

NumPy (Numeric Python) là một thư viện toán học mã nguồn mở được trang bị các hàm số đã được tối ưu để tăng hiệu quả làm việc với ma trận và mảng lớn với tốc độ xử lý nhanh hơn nhiều so với Python đơn thuần.

Tiền thân của NumPy là Numeric, ban đầu được phát triển bởi Jim Huguenin cùng với sự đóng góp của các nhà phát triển khác. Năm 2005, Travis Oliphant đã tạo ra NumPy bằng cách kết hợp các tính năng của Numarray vào Numeric.

1.1.2 Các hàm cần lưu ý của thư viện NumPy

1.1.2.1 `numpy.random.randint`

- Hàm này trả về một số nguyên ngẫu nhiên hoặc một ma trận các số nguyên có kích thước do người dùng truyền vào.
- Cú pháp:

`random.randint(<giới hạn trên>, <(số hàng, số cột)>)`

- Trong đó:
 - **Giới hạn trên:** là số lớn nhất mà hàm sẽ trả về
 - **Số hàng, số cột:** là số hàng và số cột của ma trận trả về

1.1.2.2 `numpy.flip`

- Hàm đảo ngược thứ tự của các phần tử trong mảng
- Cú pháp:

`numpy.flip(<mảng cần đảo ngược>)`

1.1.2.3 `numpy.array`

- Hàm `numpy.array` sẽ tạo ra một mảng hoặc các list có sẵn sang kiểu dữ liệu `ndarray`
- Cú pháp:
$$\text{numpy.array}(<\text{list ban đầu}>)$$

1.1.2.4 Phép nhân hai ma trận

- Để nhân hai ma trận, ta sử dụng toán tử “@”.
- Cú pháp:

$$<\text{ma trận}> @ <\text{ma trận}>$$

1.1.2.5 Ma trận chuyển vị

- Để chuyển vị ma trận trong NumPy, ta có thể dùng phương thức “.T”.
- Cú pháp:

$$<\text{ma trận}>.T$$

1.2 Các hàm và phương thức built-in trong Python

Ngoài các hàm được hỗ trợ bởi thư viện NumPy, trong chương trình này còn sử dụng các hàm được Python xây dựng sẵn (built-in) như: `print()`, cấu trúc rẽ nhánh `if...else`, vòng lặp `for`, phương thức `append()`, các toán tử toán học.

CHƯƠNG 2 – THUẬT TOÁN GIẢI QUYẾT CÁC YÊU CẦU

2.1 Câu 1d

2.1.1 Yêu cầu

Lưu các số nguyên lẻ ở ma trận A vào một vector mới và in vector đó ra màn hình.

2.1.2 Thuật toán

- **Bước 1:** Khởi tạo vector *odd_int* gồm các số lẻ trong ma trận A bằng cú pháp được hỗ trợ trong python: *odd_int = A[A % 2 != 0]*
- **Bước 2:** In biến *odd_int* ra màn hình.

2.2 Câu 1e

2.2.1 Yêu cầu

Lưu các số nguyên tố trong ma trận A vào một vector mới và in vector đó ra màn hình.

2.2.2 Thuật toán

- **Bước 1:** Tạo hàm kiểm tra số nguyên tố *isPrimeNumber(n)*
 - **Bước 1.1:** Truyền vào số nguyên *n*.
 - **Bước 1.2:** Vì số nguyên tố nhỏ nhất là 2 nên nếu *n* nhỏ hơn 2 thì trả về kết quả *False*.
 - **Bước 1.3:** Dùng vòng lặp *for* để duyệt các số từ 2 đến $n/2 + 1$, nếu *n* chia hết cho bất kì số nào trong khoảng 2 đến $n/2 + 1$ thì trả về kết quả *False*.
 - **Bước 1.4:** Nếu đi qua hết vòng lặp *for* nghĩa là *n* không chia hết cho bất kì số nào trong khoảng 2 đến $n/2 + 1$, ta sẽ trả về kết quả *True*.
- **Bước 2:** Tạo một list rỗng *prime_nums* dùng để chứa các số nguyên tố trong mảng A
- **Bước 3:** Dùng 2 vòng lặp *for* lồng nhau để truy cập đến từng phần tử trong ma trận A, dùng hàm kiểm tra số nguyên tố vừa tạo để kiểm tra phần tử đó có phải là số nguyên tố hay không. Nếu là số nguyên tố thì thêm vào list rỗng *prime_nums* bằng hàm *append()*.

- **Bước 4:** Dùng hàm `numpy.array()` trong thư viện NumPy để ép kiểu dữ liệu list của `prime_nums` thành vector và in vector `prime_nums` ra màn hình.

2.3 Câu 1f

2.3.1 Yêu cầu

Cho ma trận $D = CB$, hãy đảo ngược các phần tử trong các dòng lẻ của ma trận D và in kết quả ra màn hình.

2.3.2 Thuật toán:

- **Bước 1:** Tính ma trận $D = C * B$.
- **Bước 2:** Dùng vòng lặp `for`, với i chạy từ 0 đến $len(D)$, kiểm tra nếu i là số chẵn thì đảo ngược các dòng đó bằng hàm `numpy.flip()` trong thư viện NumPy.
- **Bước 3:** In ma trận D đã được đảo ngược các hàng lẻ ra màn hình.

2.4 Câu 1g

2.4.1 Yêu cầu

Trong ma trận A , tìm các hàng có nhiều số nguyên tố nhất và in các hàng đó ra màn hình.

2.4.2 Thuật toán:

- **Bước 1:** Tạo list `count_prime` rỗng để lưu số lượng số nguyên tố của từng hàng và biến `max_count` để lưu giá trị lớn nhất trong `count_prime`.
- **Bước 2:** Dùng 2 vòng lặp `for` lồng nhau để đếm số lượng số nguyên tố của từng hàng trong ma trận A .
 - Bước 2.1: Ở vòng lặp cha, truy cập đến các hàng trong A , khởi tạo biến `count = 0`.

- Bước 2.2: Ở vòng lặp con, truy cập đến phần tử trong hàng ở vòng lặp cha. Sử dụng lại hàm kiểm tra số nguyên tố ở câu 1e để kiểm tra, nếu là số nguyên tố thì biến *count* tăng lên 1 đơn vị.
 - Bước 2.3: Kết thúc vòng lặp con, gán giá trị của *count* cho *max_count* nếu *count* lớn hơn *max_count* và thêm giá trị của *count* vào *prime_num* bằng hàm *append()*.
 - Bước 2.4: Bước 2.1 sẽ được lặp lại cho đến khi hết vòng lặp cha.
- **Bước 3:** Nếu ma trận A có số nguyên tố (biến *max_count* lớn hơn 0):
- Bước 3.1: Sử dụng vòng lặp *for*, biến *i* chạy từ 0 đến *len(A)* (với *len(A)* là số lượng hàng có trong ma trận A)
 - Bước 3.2: So sánh giá trị biến *max_count* với *prime_nums[i]* nếu bằng nhau (tức là hàng thứ *i* của A có nhiều số nguyên tố nhất) thì in hàng tương ứng ra màn hình.
- **Bước 4:** Nếu ma trận A không có số nguyên tố (biến *max_count* bằng 0) thì in ra màn hình ma trận A không có số nguyên tố rồi kết thúc.

2.5 Câu 1h

2.5.1 Yêu cầu

Trong ma trận A, tìm các hàng có chuỗi số lẻ liên tiếp dài nhất và in các hàng đó ra màn hình.

2.5.2 Thuật toán

- **Bước 1:** Tạo hàm *odd_sequence_len(A)*, truyền vào một vector và trả về kết quả là độ dài của chuỗi số lẻ liên tiếp dài nhất.

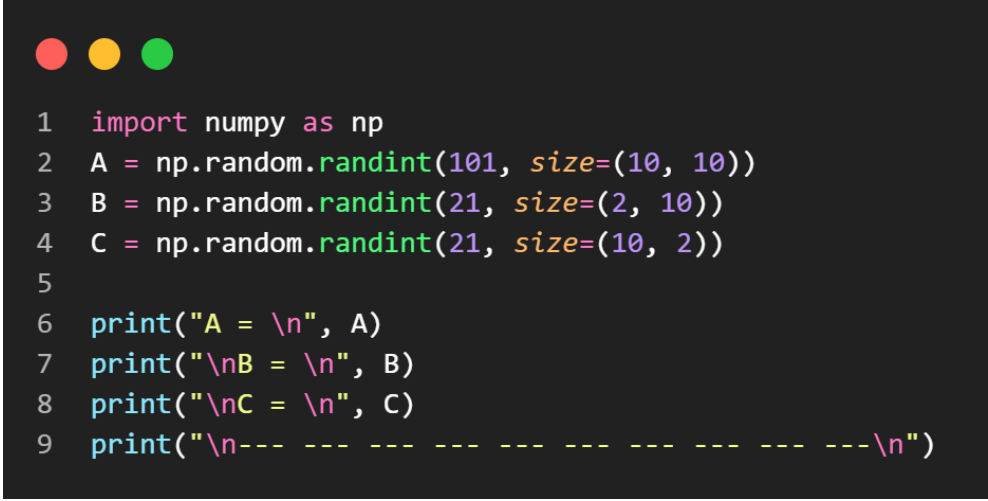
- Bước 1.1: Khởi tạo biến $length = 0$, dùng để đếm các phần tử lẻ liên tiếp nhau trong vector A truyền vào và biến $max_length = 0$, dùng để lưu giá trị lớn nhất của $length$ (hay độ dài lớn nhất của chuỗi số nguyên lẻ liên tiếp).
 - Bước 1.2: Sử dụng vòng lặp *for*, cho biến i là các giá trị trong vector A truyền vào.
 - Bước 1.3: Nếu i là số lẻ thì biến $length$ tăng lên một đơn vị. sau đó so sánh biến $length$ và max_length , nếu biến $max_length < length$ thì gán giá trị của $length$ cho max_length . Nếu i là số chẵn thì gán $length = 0$.
 - Bước 1.4: Hàm trả về giá trị của biến max_length .
- **Bước 2:** Khai báo biến $max_odd_len = 0$, dùng để lưu giá trị của hàng có độ dài của chuỗi số nguyên lẻ lớn nhất.
 - **Bước 3:** Dùng vòng lặp *for* để truyền từng hàng của ma trận A vào hàm $odd_sequence_len$ đã tạo ở bước 1 và gán giá trị trả về cho biến max_odd_len nếu giá trị trả về này lớn hơn giá trị của biến max_odd_len .
 - **Bước 4:** In các hàng thỏa điều kiện ra màn hình.
 - Bước 4.1: Nếu ma trận A không có số lẻ (biến max_odd_len có giá trị 0) thì in ra màn hình ma trận A không có số lẻ rồi kết thúc
 - Bước 4.2: Nếu ma trận A có số lẻ (biến max_odd_len có giá trị khác không), dùng vòng lặp *for* để kiểm tra độ dài của từng hàng trong ma trận A, in ra các hàng có độ dài bằng với giá trị của max_odd_len .

CHƯƠNG 3 – SOURCE CODE VÀ KẾT QUẢ

3.1 Khai báo thư viện và input

3.1.1 Source code

Input gồm 3 ma trận $A_{10 \times 10}$, $B_{2 \times 10}$ và $C_{10 \times 2}$ được tạo ngẫu nhiên từ hàm `random.randint()` trong thư viện NumPy.



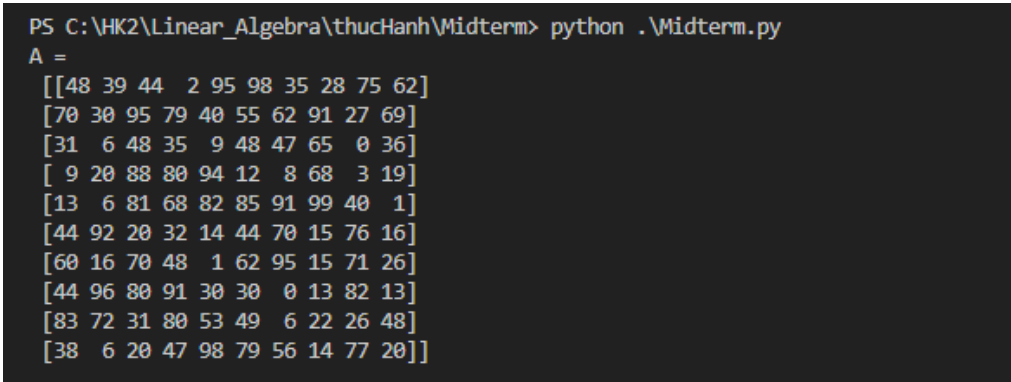
```

1  import numpy as np
2  A = np.random.randint(101, size=(10, 10))
3  B = np.random.randint(21, size=(2, 10))
4  C = np.random.randint(21, size=(10, 2))
5
6  print("A = \n", A)
7  print("\nB = \n", B)
8  print("\nC = \n", C)
9  print("\n--- --- --- --- ---\n")

```

Hình 3.1.1. Khai báo thư viện và input

3.1.2 Kết quả



```

PS C:\HK2\Linear_Algebra\thucHanh\Midterm> python .\Midterm.py
A =
[[48 39 44  2 95 98 35 28 75 62]
 [70 30 95 79 40 55 62 91 27 69]
 [31  6 48 35  9 48 47 65  0 36]
 [ 9 20 88 80 94 12  8 68  3 19]
 [13  6 81 68 82 85 91 99 40  1]
 [44 92 20 32 14 44 70 15 76 16]
 [60 16 70 48  1 62 95 15 71 26]
 [44 96 80 91 30 30  0 13 82 13]
 [83 72 31 80 53 49  6 22 26 48]
 [38  6 20 47 98 79 56 14 77 20]]

```

Hình 3.1.2a. Kết quả của input

```

B =
[[ 9 15  4  6  3 10 13  6  5  6]
 [18  5  7 20  3  1  4 17 13  2]]

C =
[[ 2  4]
 [11 11]
 [10 12]
 [ 6  8]
 [16  7]
 [15 18]
 [14 20]
 [18 20]
 [20 12]
 [ 3 20]]

```

Hình 3.1.2b. Kết quả của input

3.2 Câu 1d

3.2.1 Source code

```

1  # d.
2  odd_int = A[A % 2 != 0]
3  print("\nd). The odd integer numbers vector: \n", odd_int)

```

Hình 3.2.1. Câu 1d

3.2.2 Kết quả

```

d). The odd integer numbers vector:
[39 95 35 75 95 79 55 91 27 69 31 35  9 47 65  9  3 19 13 81 85 91 99  1
 15  1 95 15 71 91 13 13 83 31 53 49 47 79 77]

```

Hình 3.2.2. Kết quả câu 1d

3.3 Câu 1e

3.3.1 Source code

```

1  # e.
2  # Ham kiem tra mot so nguyen co phai so nguyen to hay khong
3  def isPrimeNumber(n):
4      if n < 2:
5          return False
6
7      for i in range(2, int(n/2) + 1):
8          if (n % i == 0):
9              return False
10     return True
11
12
13     prime_nums = []
14     for K in A:
15         for i in K:
16             if isPrimeNumber(i):
17                 prime_nums.append(i)
18     prime_nums = np.array(prime_nums)
19     print("\ne). The prime numbers vector: \n", prime_nums)

```

Hình 3.3.1. Câu 1e

3.3.2 Kết quả

```

e). The prime numbers vector:
[ 2 79 31 47  3 19 13 71 13 13 83 31 53 47 79]

```

Hình 3.3.2. Kết quả câu 1e

3.4 Câu 1f

3.4.1 Source code

```

1  # f.
2  D = C@B
3  print("\nf). Matrix D = C * B: \n", D)
4
5  for i in range(0, len(D)):
6      if (i % 2 == 0):
7          D[i] = np.flip(D[i])
8  print("\nReverse element in the odd rows of the matrix D = C@B: \n", D)

```

Hình 3.4.1. Câu 1f

3.4.2 Kết quả

```

f). Matrix D = C * B:
[[ 90  50  36  92  18  24  42  80  62  20]
 [297 220 121 286  66 121 187 253 198  88]
 [306 210 124 300  66 112 178 264 206  84]
 [198 130  80 196  42  68 110 172 134  52]
 [270 275 113 236  69 167 236 215 171 110]
 [459 315 186 450  99 168 267 396 309 126]
 [486 310 196 484 102 160 262 424 330 124]
 [522 370 212 508 114 200 314 448 350 148]
 [396 360 164 360  96 212 308 324 256 144]
 [387 145 152 418  69  50 119 358 275  58]]

Reverse element in the odd rows of the matrix D = C@B:
[[ 20  62  80  42  24  18  92  36  50  90]
 [297 220 121 286  66 121 187 253 198  88]
 [ 84 206 264 178 112  66 300 124 210 306]
 [198 130  80 196  42  68 110 172 134  52]
 [110 171 215 236 167  69 236 113 275 270]
 [459 315 186 450  99 168 267 396 309 126]
 [124 330 424 262 160 102 484 196 310 486]
 [522 370 212 508 114 200 314 448 350 148]
 [144 256 324 308 212  96 360 164 360 396]
 [387 145 152 418  69  50 119 358 275  58]]

```

Hình 3.4.2. Kết quả câu 1f

3.5 Câu 1g

3.5.1 Source code

```

1  # g.
2  count_prime = [] # Lưu số lượng số nguyên tố của từng hàng
3  max_count = 0 # Lưu giá trị lớn nhất trong count_prime
4
5  for K in A:
6      count = 0 # Dùng để đếm các số nguyên tố trong hàng
7      for i in K:
8          if isPrimeNumber(i):
9              count += 1
10         if max_count < count:
11             max_count = count
12         count_prime.append(count)
13
14 # In các dòng có count số nguyên tố lớn nhất
15 if max_count > 0:
16     print("\ng). Rows have maximum count of prime numbers: ")
17     for i in range(0, len(A[0])):
18         if count_prime[i] == max_count:
19             print(A[i])
20 else:
21     print("\ng). The matrix A has no prime number")

```

Hình 3.5.2. Câu 1g

3.5.2 Kết quả

```

g). Rows have maximum count of prime numbers:
[83 72 31 80 53 49  6 22 26 48]

```

Hình 3.5.2. Kết quả câu 1g

3.6 Câu 1h

3.6.1 Source code

```

1  # h.
2  # Ham tra ve do dai cua chuoai le lien tuc trong tung hang
3  def odd_sequence_len(A):
4      length = 0
5      max_length = 0
6
7      for i in A:
8          if (i % 2 != 0):
9              length += 1
10             if max_length < length:
11                 max_length = length
12             else:
13                 length = 0
14
15         return max_length
16
17
18 max_odd_len = 0 # Luu do dai lon nhat cua chuoai le
19
20 for K in A:
21     if max_odd_len < odd_sequence_len(K):
22         max_odd_len = odd_sequence_len(K)
23
24 # In cac dong co chuoai le lien tuc dai nhat
25 if max_odd_len != 0:
26     print("\nh). Rows have longest contiguous odd numbers sequence: ")
27     for K in A:
28         if odd_sequence_len(K) == max_odd_len:
29             print(K)
30 else:
31     print("\nh). The matrix A has no odd numbers")

```

Hình 3.6.1. Câu 1h

3.6.2 Kết quả

```

h. Rows have longest contiguous odd numbers sequence:
[35 86 1 83 49 37 42 71 39 10]
[63 10 35 19 37 41 90 46 22 70]

```

Hình 3.6.2. Kết quả câu 1h

TÀI LIỆU THAM KHẢO

- [1] M. Khai, 17/12/2020. [Trực tuyến]. Available: <https://viblo.asia/p/tim-hieu-ve-thu-vien-numpy-trong-pythonphan-1-Do7542QXZM6>.
- [2] W3School, [Trực tuyến]. Available: <https://www.w3schools.com/python/>.
- [3] NumPy, [Trực tuyến]. Available: numpy.org.
- [4] Wikipedia, 15/6/2021. [Trực tuyến]. Available: <https://en.wikipedia.org/wiki/NumPy>.