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

Tiểu luận giữa kì môn đại số tuyến tính

Người hướng dẫn: **NGUYỄN VĂN KHOA**
Người thực hiện: **TẠ HOÀI SỸ NGUYÊN – 522H0074**
TRẦN HỮU ĐẠT – 522H0081
Lớp **: 22H50201**
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

Tiểu luận giữa kì môn đại số tuyến tính

Người hướng dẫn: ThS **NGUYỄN VĂN KHOA**
Người thực hiện: **TẠ HOÀI SỸ NGUYÊN**
TRẦN HỮU ĐẠT
Lớp **: 22H50201**
Khoá **: 16**

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2023

LỜI CẢM ƠN

Tôi xin gửi lời cảm ơn sâu sắc đến Thầy Nguyễn Văn Khoa đã hỗ trợ và giúp đỡ tôi trong suốt quá trình nghiên cứu và hoàn thành bài tiểu luận này. Thầy đã tận tình hướng dẫn và giúp đỡ tôi trong quá trình thực hiện. Những lời khuyên và nhận xét của thầy đã giúp tôi hiểu rõ hơn về những khó khăn hạn chế của bài tiểu luận này. Cuối cùng, Tôi cũng xin cảm ơn tất cả các thầy cô trong khoa Công Nghệ Thông Tin của Trường Đại học Tôn Đức Thắng, đã cung cấp cho tôi những điều kiện tốt nhất để thực hiện bài tiểu luận.

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)

MỤC LỤC

CHƯƠNG 1 : Phương pháp giải bài tập

<u>Câu 1:</u> Tạo ma trận A cấp 10x10 với các số ngẫu nhiên $\in [1, 100]$, ma trận B cấp 2x10 Thực hiện các yêu cầu sau:

Cách tạo ma trận: dùng hàm np.random.randint() có trong thư viện numpy để tạo một ma trận 10x10, với các phần tử trong ma trận được lấy ngẫu nhiên từ 1 đến 100 và gán nó vào biến A.
Các tham số trong hàm np.random.randint() lần lượt là:

- 0: giới hạn dưới của các phần tử số nguyên ngẫu nhiên được tạo ra.
- 101: giới hạn trên (ngoại trừ 101) của các phần tử số nguyên ngẫu nhiên được tạo ra.
- (10, 10): kích thước của ma trận 10x10 được tạo ra.

Cách tạo ma trận B và C cũng giống như ma trận A ở phía trên.

1a. Tính $A + A^T + CB + B^T C^T$ và in kết quả ra màn hình.

- **Câu a:** dùng để tính toán các phép toán cộng và nhân ma trận. Cụ thể, ma trận A cộng với ma trận chuyển vị của A, được biểu thị bởi A.T (.T là một phương thức để biểu thị ma trận chuyển vị trong thư viện Numpy). Sau đó cộng tổng của tích ma trận C và ma trận B, được biểu thị bởi C.dot(B) (numpy.dot() là phương thức được dùng để nhân hai ma trận với nhau trong thư viện numpy), với tích của ma trận chuyển vị của B với ma trận chuyển vị của C, được biểu thị bởi (B.T).dot(C.T). Sau cùng, kết quả sẽ được lưu vào biến a.Đoạn code thứ hai dùng để in ra màn hình giá trị của biến a.

1b. Tính $\frac{A}{10} + \left(\frac{A}{11}\right)^2 + \left(\frac{A}{12}\right)^3 + \cdots + \left(\frac{A}{17}\right)^8 + \left(\frac{A}{18}\right)^9 + \left(\frac{A}{19}\right)^{10}$ và in kết quả ra màn hình.

- Đầu tiên khai báo một list rỗng b
- Sau đó sử dụng 2 vòng lặp để tính toán biểu thức mà đề bài cho theo công thức: $\left(\frac{A}{i}\right)^j$. Vòng lặp đầu tiên lặp qua các giá trị từ 10 đến 19, và vòng lặp thứ hai lặp qua các giá trị từ 1 đến 10.
- Sau mỗi lần lặp, chương trình sẽ tính toán giá trị của biểu thức (A/i)**j (với I và j là các biến lặp). Sau đó, kết quả của biểu thức sẽ được cộng vào list b.

- Sau khi hoàn thành vòng lặp, list b sẽ chứa giá trị đã tính toán. Sau đó chương trình sử dụng phương thức np.array() trong thư viện để chuyển đổi list b thành mảng numpy
- Sau cùng là in kết quả của của b lên màn hình.

1c. Lưu các hàng lẻ của ma trận A thành một ma trận mới và in kết quả đó ra màn hình.

- Đoạn code "c = A[0:10:2, :]" dùng để lấy các hàng lẻ của ma trận A ra và gán vào ma trận c.
- Cú pháp lát cắt "0:10:2" có nghĩa là "bắt đầu từ vị trí thứ 0, kết thúc tại vị trí thứ 9 và có bước nhảy là 2 trong đoạn [0, 9], Ký hiệu " : " sau dấu phẩy có nghĩa là "bao gồm tất cả các cột".
- Sau cùng là in giá trị của c ra màn hình

1d. Lưu các số lẻ của ma trận A thành một ma trận mới và in kết quả đó ra màn hình.

- Lấy tất cả các phần tử trong mảng "A" có giá trị là số lẻ và lưu trữ chúng trong mảng "d". Sau đó chuyển mảng d thành vector bằng phương thức có trong thư viện numpy "np.array()". Cuối cùng là in vector d ra màn hình.

1e. Lưu các số nguyên tố của ma trận A thành một ma trận mới và in kết quả đó ra màn hình.

- Đầu tiên, định nghĩa một hàm kiểm tra số nguyên tố isPrime() có tham số là một số nguyên:
  - Số nguyên tố là số chỉ có hai ước là 1 và chính nó. Cho nên, đoạn code này hoạt động bằng cách:
  - Đầu tiên, so sánh số nguyên cần được kiểm tra với 1, nếu số đó nhỏ hơn số 1 thì ta trả về "False".
  - Sau đó, ta chạy vòng lặp for với biến i nằm trong khoảng từ 2 đến làm tròn chữ số nguyên của $\sqrt{n}$. Nếu như số cần kiểm tra (n) chia hết cho i, thì ta trả về "False".
  - Khi chạy xong vòng lặp, cuối cùng thì ta trả về "True".
- khai báo list rỗng e.
- Sau đó, Sử dụng vòng lặp để duyệt qua từng phần tử của ma trận A. Cụ thể, với mỗi giá trị của i từ 0 đến A.shape[0] (số dòng của ma trận), và với mỗi giá trị của j từ 0 đến A.shape[1] (số cột của ma trận), điều kiện

lọc "isPrime (A[i, j]) == True" kiểm tra xem giá trị phần tử đó có là số nguyên tố không. Nếu có, giá trị đó được thêm vào list e bằng phương thức "append()".

- Kết quả cuối cùng của đoạn code này là danh sách e chứa tất cả các giá trị là số nguyên tố trong ma trận A.
- Sau cùng, dùng phương thức "np.array()" để chuyển list e thành mảng numpy và in ra màn hình bằng câu lệnh "print()".

1f. Cho một ma trận D = CB, đảo ngược các phần tử trong các hàng lẻ của ma trận D và in kết quả ra màn hình.

- Để có được ma trận D, nhân hai ma trận C với B với nhau thông qua phương thức (C.dot(B) – đây là hàm nhân ma trận có trong thư viện numpy)
- Tiếp đến, khai báo một list rỗng "D_rev"
- Sau đó, dùng vòng lặp để duyệt qua từng hàng của D. Cụ thể, với mỗi giá trị của i từ 0 đến D.shape[0] (số dòng của ma trận). Nếu "i == 0" hoặc "i % 2 == 0" (với i là số chẵn) thì đảo ngược các phần tử của dòng đó bằng câu lệnh "np.flip()", sau đó dùng câu lệnh "append" để cho dòng đã được đảo ngược vào trong list rỗng "D_rev". Ngược lại, nếu i không phải là số chẵn thì ta dùng câu lệnh "append" để cho dòng không đảo ngược vào list rỗng "D_rev".
- Tiếp đến, sử dụng phương thức "np.array()" để biến list "D_rev" trở thành mảng numpy.
- Cuối cùng, in giá trị của mảng numpy "D_rev" ra màn hình.

1g. Đối với ma trận A, hãy tìm các hàng có số nguyên tố nhiều nhất và in các hàng đó ra màn hình

- Khai báo biến "maxCount" để giữ giá trị số lượng số nguyên tố lớn nhất của mảng
- Khai báo một list rỗng "resultant" để lưu các dòng có nhiều số nguyên tố nhất
- Tiếp đến, duyệt qua từng hàng của ma trận A.
- Khởi tạo giá trị cho biến count bằng 0
- Duyệt qua từng phần tử trong hàng đó
- Kiểm tra xem phần tử đó có phải là số nguyên tố không bằng lời gọi hàm "isPrime(num)" đã được định nghĩa ở câu 1e, nếu phần tử đó là số nguyên tố thì tăng giá trị của "count" lên 1.
- Sau khi duyệt qua các phần tử của dòng đó, biến count sẽ lưu giữ giá trị số lượng số nguyên tố trong dòng đó.

- Nếu “count” lớn hơn giá trị của “maxCount”, thì gán giá trị của “maxCount” bằng “count” và gán giá trị của “resultant” bằng mảng chứa “row”.
- Nếu “count” bằng giá trị của “maxCount”, thì thêm “row” vào mảng “resultant” bằng phương thức “append”.
- Sau cùng, in giá trị của Resultant ra màn hình.
- Tiếp đến, sử dụng phương thức “np.array()” để biến list “resultant” trở thành mảng numpy
- Cuối cùng, sử dụng vòng lặp để in từng dòng của mảng numpy “resultant” ra màn hình

1h. Đối với ma trận A, hãy tìm các hàng có dãy số lẻ liền kề dài nhất và in các hàng đó ra màn hình.

- Khai báo biến “maxCount” để lưu giữ giá trị số lượng các số lẻ nguyên liên tiếp lớn nhất.
- Khai báo một list rỗng “results” để lưu giữ các dòng các số lẻ nguyên liên tiếp nhiều nhất.
- Tiếp đến, sử dụng vòng lặp duyệt qua từng hàng của ma trận A.
- Khởi tạo giá trị cho biến count bằng 0
- Tiếp đến, sử dụng vòng lặp duyệt qua từng cột của hàng hiện tại của ma trận A.
- Nếu duyệt đến phần tử cuối cùng của dòng thì bỏ qua
- Nếu hai phần tử liền kề đều là số lẻ thì tăng giá trị của “count” lên 1.
- Nếu “count” lớn hơn giá trị của “maxCount”, thì gán giá trị của “maxCount” bằng “count” và gán giá trị của “resultant” bằng mảng chứa “A[i]”.
- Nếu “count” bằng giá trị của “maxCount”, thì thêm “A[i]” vào mảng “results” bằng phương thức “append”.
- Sau cùng, in giá trị của Results ra màn hình.
- Tiếp đến, sử dụng phương thức “np.array()” để biến list “results” trở thành mảng numpy
- Cuối cùng, sử dụng vòng lặp để in từng dòng của mảng numpy “results” ra màn hình

CHƯƠNG 2 : Source code và output

1a.

Source code

```
#a) calculate
print("-------------------- a ----------------------")
a = A + A.T + C.dot(B) + (B.T).dot(C.T)
print("a = \n", a)
```

Output

```
-------------------- a ----------------------
a =
 [[166 197 357 426 176 263 305 229 453 287]
 [197 170 424 325 397 408 268 229 291 484]
 [357 424 594 644 670 633 476 444 372 745]
 [426 325 644 696 716 709 393 363 506 686]
 [176 397 670 716 526 511 499 559 551 682]
 [263 408 633 709 511 556 556 477 559 712]
 [305 268 476 393 499 556 236 370 312 446]
 [229 229 444 363 559 477 370 260 228 416]
 [453 291 372 506 551 559 312 228 460 442]
 [287 484 745 686 682 712 446 416 442 716]]
```

1b.

Source code

```
#b) calculate
print("-------------------- b ---------------------")
b = [0]
print("A = \n", A)
for i in range(10, 20):
    for j in range(1, 11):
        b = b + (A/i)**j
np.array(b)
print("b = \n", b)
```

Output

```
-------------------- b ---------------------
A =
 [[18 55 35  2 83 82  6 16  4 24]
 [64 19 32  5 11 58  8 67 22 51]
 [59 10 33 19 33 10 27 12 81  5]
 [34 35 47 52 86 37 36  6 66  5]
 [52 91 56 38 98 93 72 35 55 76]
 [99 70 69 22 82 81 44 14 83 59]
 [53 60 12 86 71 98 56 23 60 60]
 [12 80 84 30 54 59 15 66 67 17]
 [18 45 32 59 31 40 57 48 85 21]
 [83 65 81 38 65 22 88 61 85 83]]
b =
[[1.54596183e+03 5.32578428e+07 6.75032695e+05 1.69339072e+00
  3.01351517e+09 2.67438591e+09 8.06729355e+00 5.91325873e+02
  4.13947507e+00 1.96262105e+04]
 [2.34337114e+08 2.44946981e+03 2.87751135e+05 5.83966282e+00
  5.12396531e+01 8.94464555e+07 1.55630885e+01 3.67157218e+08
  8.91319298e+03 2.55210691e+07]
 [1.05710262e+08 3.32362582e+01 3.85374472e+05 2.44946981e+03
  3.85374472e+05 3.32362582e+01 5.82420554e+04 8.15127540e+01
  2.37000893e+09 5.83966282e+00]
 [5.11988122e+05 6.75032695e+05 1.15419009e+07 3.08305917e+07
  4.27579259e+09 1.14859751e+06 8.83573717e+05 8.06729355e+00
  3.16819560e+08 5.83966282e+00]
 [3.08305917e+07 7.46505881e+09 6.34949376e+07 1.48335073e+06
  1.55140413e+10 9.25110149e+09 7.44155391e+08 6.75032695e+05
  5.32578428e+07 1.26606482e+09]
 [1.71503726e+10 5.64308980e+08 4.89980524e+08 8.91319298e+03
  2.67438591e+09 2.37000893e+09 6.09275178e+06 2.18208738e+02
  3.01351517e+09 1.05710262e+08]
 [3.71154563e+07 1.24590827e+08 8.15127540e+01 4.27579259e+09
  6.48640668e+08 1.55140413e+10 6.34949376e+07 1.33170038e+04
  1.24590827e+08 1.24590827e+08]
 [8.15127540e+01 2.09718099e+09 3.39088107e+09 1.56331540e+05
  4.45319440e+07 1.05710262e+08 3.60035675e+02 3.16819560e+08
  3.67157218e+08 9.62081461e+02]
 [1.54596183e+03 7.57286680e+06 2.87751135e+05 1.05710262e+08
  2.13045616e+05 2.42871770e+06 7.54717375e+07 1.41580552e+07
  3.81027001e+09 5.88122999e+03]
 [3.01351517e+09 2.72783364e+08 2.37000893e+09 1.48335073e+06
  2.72783364e+08 8.91319298e+03 5.36344460e+09 1.46456178e+08
  3.81027001e+09 3.01351517e+09]]
```

1c.

Source code

```
print("-------------------- c ----------------------")

# Save odd rows of the A A into a new A, and print the resultant A to
# the screen

c = A[0:10:2, :]
print(A)
print(" c = \n",c)
```

Output

```
---------------------- c ----------------------
[[18 55 35  2 83 82  6 16  4 24]
 [64 19 32  5 11 58  8 67 22 51]
 [59 10 33 19 33 10 27 12 81  5]
 [34 35 47 52 86 37 36  6 66  5]
 [52 91 56 38 98 93 72 35 55 76]
 [99 70 69 22 82 81 44 14 83 59]
 [53 60 12 86 71 98 56 23 60 60]
 [12 80 84 30 54 59 15 66 67 17]
 [18 45 32 59 31 40 57 48 85 21]
 [83 65 81 38 65 22 88 61 85 83]]
 c =
 [[18 55 35  2 83 82  6 16  4 24]
 [59 10 33 19 33 10 27 12 81  5]
 [52 91 56 38 98 93 72 35 55 76]
 [53 60 12 86 71 98 56 23 60 60]
 [18 45 32 59 31 40 57 48 85 21]]
```

1d.

Source code

```
print("-------------------- d ---------------------")

d = A[A%2 != 0]
print(A)
np.array(d)

print("d = \n",d)
```

Output

```
-------------------- d ---------------------
[[18 55 35  2 83 82  6 16  4 24]
 [64 19 32  5 11 58  8 67 22 51]
 [59 10 33 19 33 10 27 12 81  5]
 [34 35 47 52 86 37 36  6 66  5]
 [52 91 56 38 98 93 72 35 55 76]
 [99 70 69 22 82 81 44 14 83 59]
 [53 60 12 86 71 98 56 23 60 60]
 [12 80 84 30 54 59 15 66 67 17]
 [18 45 32 59 31 40 57 48 85 21]
 [83 65 81 38 65 22 88 61 85 83]]
d =
 [55 35 83 19  5 11 67 51 59 33 19 33 27 81  5 35 47 37  5 91 93 35 55 99
 69 81 83 59 53 71 23 59 15 67 17 45 59 31 57 85 21 83 65 81 65 61 85 83]
```

1e.

Source code

```
print("-------------------- e ----------------------")
# Save prime numbers in the A A into a new vector, and print the resultant
# vector to the screen


def isPrime(n):
  if n <= 1:
    return False
  for i in range(2, int(n**0.5)+1):
    if n % i == 0:
      return False
  return True

e = []
for i in range(A.shape[0]):
    for j in range(A.shape[1]):
        if ( isPrime (A[i, j]) == True ):
            e.append(A[i, j])


print(A)
np.array(e)
print("e = \n", e)
```

Output

```
-------------------- e ----------------------
[[18 55 35  2 83 82  6 16  4 24]
 [64 19 32  5 11 58  8 67 22 51]
 [59 10 33 19 33 10 27 12 81  5]
 [34 35 47 52 86 37 36  6 66  5]
 [52 91 56 38 98 93 72 35 55 76]
 [99 70 69 22 82 81 44 14 83 59]
 [53 60 12 86 71 98 56 23 60 60]
 [12 80 84 30 54 59 15 66 67 17]
 [18 45 32 59 31 40 57 48 85 21]
 [83 65 81 38 65 22 88 61 85 83]]
e =
 [2, 83, 19, 5, 11, 67, 59, 19, 5, 47, 37, 5, 83, 59, 53, 71, 23, 59, 67, 17, 59, 31, 83, 61, 83]
```

1f.

Source code

```
print("-------------------- f ----------------------")
# Given a A , reverse elements in the odd rows of the A D, and
# print the resultant A to the screen.
#  A_rev.append(np.flip(A[i], axis = -1))

D = C.dot(B)
print(" D = \n ", D)

D_rev = []
for i in range(D.shape[0]):
        if ((i == 0) or (i % 2 == 0)):
            D_rev.append(np.flip(D[i], axis = -1))
        else:
            D_rev.append(D[i])

res = np.array(D_rev)
print(res)
```

Output

```
-------------------- f ----------------------
 D =
  [[  0  35  35  52  63  28  13  57  67  23]
 [  0  80 140  76 124 144  84 136 196  84]
 [  0 170 182 244 302 152  74 278 334 118]
 [  0 285 399 342 475 380 209 475 627 247]
 [  0 130 133 191 233 108  51 212 251  87]
 [  0 180 294 186 286 296 170 304 426 178]
 [  0  90  63 153 171  36   9 144 153  45]
 [  0 155 308 121 228 328 196 267 406 182]
 [  0 150 294 120 222 312 186 258 390 174]
 [  0 195 168 309 360 120  48 315 354 114]]
[[ 23  67  57  13  28  63  52  35  35   0]
 [  0  80 140  76 124 144  84 136 196  84]
 [118 334 278  74 152 302 244 182 170   0]
 [  0 285 399 342 475 380 209 475 627 247]
 [ 87 251 212  51 108 233 191 133 130   0]
 [  0 180 294 186 286 296 170 304 426 178]
 [ 45 153 144   9  36 171 153  63  90   0]
 [  0 155 308 121 228 328 196 267 406 182]
 [174 390 258 186 312 222 120 294 150   0]
 [  0 195 168 309 360 120  48 315 354 114]]
```

1g.

Source code

```
print("-------------------- g ---------------------")
#  Regarding the A A, find the rows which have maximum count of prime
# numbers, and print the rows to the screen.

maxCount = 0
resultant = []
for row in A:
    count = 0
    print("row: ", row)
    for num in row:
        if( isPrime(num) == True ):
            count += 1


    if (count > maxCount):
        maxCount = count
        resultant = [row]
    elif ( count == maxCount ):
        resultant.append(row)

print("Rows with maximum count of prime numbers:")
np.array(resultant)
for row in resultant:
    print(row)
```

Output

```
-------------------- g ---------------------
row:  [18 55 35  2 83 82  6 16  4 24]
row:  [64 19 32  5 11 58  8 67 22 51]
row:  [59 10 33 19 33 10 27 12 81  5]
row:  [34 35 47 52 86 37 36  6 66  5]
row:  [52 91 56 38 98 93 72 35 55 76]
row:  [99 70 69 22 82 81 44 14 83 59]
row:  [53 60 12 86 71 98 56 23 60 60]
row:  [12 80 84 30 54 59 15 66 67 17]
row:  [18 45 32 59 31 40 57 48 85 21]
row:  [83 65 81 38 65 22 88 61 85 83]
Rows with maximum count of prime numbers:
[64 19 32  5 11 58  8 67 22 51]
```

1h.

Source code

```
print("----------------------------- h -----------------------------")
#  Regarding the A A, find the rows which have the longest contiguous odd
# numbers sequence, and print the rows to the screen.



print("A \n", A)


maxCount = 0
results = []
for i in range (A.shape[0]):
    count = 0
    for j in range (A.shape[1] - 1):
        if (A[i, j + 1] is None):
            continue
        elif( ( A[i, j] % 2 != 0  ) and ( A[i, j + 1] % 2 != 0  ) ):
            count += 1

    if (count > maxCount):
        maxCount = count
        results = [A[i]]
    elif(count == maxCount):
        results.append(A[i])

np.array(results)
print("the rows which have the longest contiguous odd numbers sequence: ")
for row in results:
    print(row)
```

Output

```
----------------------------- h -----------------------------
A
 [[18 55 35  2 83 82  6 16  4 24]
 [64 19 32  5 11 58  8 67 22 51]
 [59 10 33 19 33 10 27 12 81  5]
 [34 35 47 52 86 37 36  6 66  5]
 [52 91 56 38 98 93 72 35 55 76]
 [99 70 69 22 82 81 44 14 83 59]
 [53 60 12 86 71 98 56 23 60 60]
 [12 80 84 30 54 59 15 66 67 17]
 [18 45 32 59 31 40 57 48 85 21]
 [83 65 81 38 65 22 88 61 85 83]]
the rows which have the longest contiguous odd numbers sequence:
[83 65 81 38 65 22 88 61 85 83]
```