BÀI 3: MA TRẬN VÀ HỆ PHƯƠNG TRÌNH TUYẾN TÍNH

Mục tiêu:

- *Tính toán được ma trận khả nghịch, phương trình ma trận.*
- *Tìm hiểu một số hàm xử lý về ma trận của gói scipy.*

Nội dung chính:

1. Tóm tắt một số phép xử lý ma trận của NumPy và Scipy

Dưới đây là một số các hàm xử lý ma trận của NumPy:

1.1. Một số phép xử lý ma trận của NumPy và Scipy

- **Khởi tạo ma trận:**

Khai báo thêm thư viện scipylinalg và một số các vector khởi tạo:

>>> from **scipy** import **linalg** **# Tải gói linalg của scipy vào bộ nhớ (để sử dụng)**

>>> import **numpy** as np **# Tải gói numpy vào sử dụng với tên gọi là np**

[Ôn tập một số lệnh đã học]

>>> a = np.**array**([1, 2, 3])

>>> b = np.**array**([(1+9j, 2j, 3j), (4j, 5j, 6j)]) # số phức

>>> c = np.**array**([[(0.5, 1.5, 10), (3,2,1)] , [(6,5,4), (7,8,9)]]) # ma trận số thực

Các kiểu khai báo và khởi tạo ma trận:

>>> A = np.**matrix**(np.random.random((2,2)))

>>> B = np.**asmatrix**(b) **# Chuyển b thành dạng ma trận**

>>> C = np.**mat**(np.random.random((10,5)))

>>> D = np.**mat**([[4, 3], [2, 6]])

>>> F = np.**eye**(3, k=1) **# tạo ma trận đường chéo. 3 là ma trận 3x3, k=1 là đường chéo nằm phía trên đường chéo chính (k = 0).**

…….. ← sinh viên điền kết quả

Thử một số câu lệnh liên quan:

>>> F = np.**eye**(3, k=0)

…….. ← sinh viên điền kết quả

>>> F = np.**eye**(3, k=-1)

…….. ← sinh viên điền kết quả

- **Các phép xử lý đơn giản**

  Xem hạng ma trận:

  >>> **np.linalg.matrix_rank(C)**

  …….. ← sinh viên điền kết quả

  Tính ma trận nghịch đảo:

  >>> **A.I**

  …….. ← sinh viên điền kết quả

  >>> **linalg.inv(A)** **# đây là lệnh của gói scipy, nên có thể sử dụng scipy.linalg.inv(A)**

  …….. ← sinh viên điền kết quả

  Định thức ma trận:

  >>> **linalg.det(A)**

  …….. ← sinh viên điền kết quả

  Chuyển đổi ma trận:

  >>> **A.T** **#chuyển vị (dòng ←→ cột)**

  …….. ← sinh viên điền kết quả

  >>> **A.H** **# chuyển vị liên hợp**

  …….. ← sinh viên điền kết quả

- **Giải các loại phương trình tuyến tính:**

>>> **linalg.solve(A, b)**

…….. ← sinh viên điền kết quả

>>> **E = np.mat(a).T**

…….. ← sinh viên điền kết quả

>>> **linalg.lstsq(F, E)**

…….. ← sinh viên điền kết quả

- **Các hàm trên ma trận**

Giới thiệu một số hàm trên ma trận. Các bạn sinh viên thực tập trên ma trận A và D khai báo bên trên:

- Cộng ma trận:

>>> **np.add(A, D)**

…….. ← sinh viên điền kết quả

- Trừ ma trận:

>>> **np.subtract(A, D)**

…….. ← sinh viên điền kết quả

- Chia ma trận:

>>> **np.divide(A, D)**

…….. ← sinh viên điền kết quả

- Nhân ma trận:

>>> **A @ D** **# phiên bản Python 3.x**

…….. ← sinh viên điền kết quả

>>> **np.multiply(D, A)**

…….. ← sinh viên điền kết quả

>>> **np.dot(A, D)**

…….. ← sinh viên điền kết quả

>>> **np.vdot(A, D)**

…….. ← sinh viên điền kết quả

- Hàm lũy thừa và logarith ma trận:

>>> **linalg.expm(A)**

…….. ← sinh viên điền kết quả

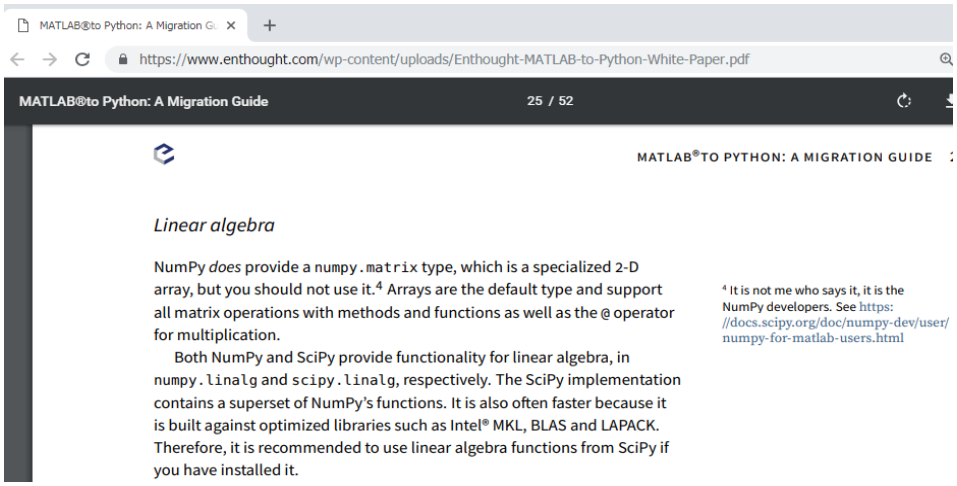
>>> **linalg.logm(A)**

…….. ← sinh viên điền kết quả

1.2. Thông tin khuyến cáo/lựa chọn sử dụng gói phần mềm

Gói Numpy có hỗ trợ kiểu dữ liệu numpy.matrix với đầy đủ các lệnh xử lý. Cụ thể là các tính năng xử lý đại số có trong gói numpy.linalg. Tuy vậy, những người phát triển ra hệ thống numpy khuyên người sử dụng hạn chế sử dụng gói numpy.linalg. Thay vào đó, người sử dụng được khuyên nên sử dụng gói **scipy.linalg** với những tính năng tương tự. Mặc dù Scipy được xây dựng trên nền tảng Numpy nhưng về tốc độ (đặc biệt khi xử lý dữ liệu lớn), gói Scipy tính toán nhanh hơn gói Numpy do được hỗ trợ các thư viện tối ưu như Intel MKL, BLAS và LAPACK.

Tham khảo tại: https://www.enthought.com/wp-content/uploads/Enthought-MATLAB-to-Python-White-Paper.pdf



2. Tích (nhân) ma trận với ma trận

Mục này ôn lại một số kiến thức lý thuyết và giới thiệu một số loại ma trận đặc biệt.

2.1. Nhân ma trận

Nhắc lại: (giả định chọn chỉ số bắt đầu từ 1) Tích hai ma trận A kích thước $m \times n$ với ma trận B kích thước $n \times p$ là ma trận C có kích thước $m \times p$ với mỗi phần tử của ma trận C là:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

Bài tập minh họa: Tìm siêu thị bán hàng rẻ

Ba ông Cảnh, Tùng, Hiếu dự định sẽ mua cuốn, bún, bánh ngọt và bánh mì với số lượng khác nhau. Được biết, 4 loại hàng hóa này được bày bán tại 2 siêu thị. Như vậy, mỗi người này nên

chọn mua ở cửa hàng nào để tổng tiền là ít nhất. Giả định, nhu cầu của mỗi người và giá mỗi sản phẩm được cho ở mỗi cửa hàng như dưới đây:

	Cuốn	Bún	Bánh ngọt	Bánh mì
Cảnh	6	5	3	1
Tùng	3	6	2	2
Hiếu	3	4	3	1

Giá tại mỗi siêu thị:

	S1	S2
Cuốn	1.50	1.00
Bún	2.00	2.50
Bánh ngọt	5.00	4.50
Bánh mì	16.00	17.00

Như vậy, nếu chọn siêu thị 1 và P2, người P1 sẽ phải trả số tiền là:

$$6x\ 1.50 + 5*\ 2.00 + 3*\ 5.00 + 1*\ 16.00 = 50 \text{ đồng}$$

Và:

$$6x\ 1.00 + 5*\ 2.50 + 3*\ 4.50 + 1*\ 17.00 = 49 \text{ đồng}$$

Như vậy, đối với ông Cảnh, giá ở siêu thị S2 rẻ hơn giá ở siêu thị S1. Với những người khác, chúng ta có thể thực hiện phép nhân ma trận:

$$Ma\ trận\ nhu\ cầu: P = \begin{pmatrix} 6 & 5 & 3 & 1 \\ 3 & 6 & 2 & 2 \\ 3 & 4 & 3 & 1 \end{pmatrix}; Ma\ trận\ giá: Q = \begin{pmatrix} 1.50 & 1.00 \\ 2.00 & 2.50 \\ 5.00 & 4.50 \\ 16.00 & 17.00 \end{pmatrix}$$

Và kết quả là:

$$P.Q = \begin{pmatrix} 50 & 49 \\ 58.50 & 61 \\ 43.50 & 43.50 \end{pmatrix}$$

Từ đó, chúng ta thấy ông Cảnh nên mua ở siêu thị S2, ông Tùng nên mua ở siêu thị S1 và ông Hiếu có thể mua ở 1 trong hai siêu thị vì giá bằng nhau.

2.2. Ma trận khả nghịch

Ma trận A $n \times n$ gọi là khả nghịch và ma trận khả nghịch của A là A^{-1} khi:

$$AA^{-1} = I_n \text{ hoặc } A^{-1}A = I_n \text{ với } I_n \text{ là ma trận đơn vị cấp } n$$

Các tính chất của ma trận nghịch đảo:

Tính chất của phép toán nghịch đảo ma trận:

$$(AB)^{-1} = B^{-1}A^{-1}$$

2.3. Ma trận hội tụ

Sinh viên hãy lập trình tính toán các ma trận lũy thừa bậc k của các ma trận sau:

- ***Ma trận phân kỳ:***

$$A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}^k ; B = \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix}^k ;$$

>>> import **numpy** as **np**

- Tính ma trận A:

```
>>> A = np.array([ [0,1], [1,0]])
>>> print(A)
```

…….. ← sinh viên điền kết quả

```
>>> temp = A.dot(A)     # tính toan lần thứ 1
>>> print(temp)
```

…….. ← sinh viên điền kết quả

```
>>> k= 6
>>> for i in range(k-1):
        temp = temp.dot(A)
        print (temp)
        print('---')
```

…….. ← sinh viên điền kết quả và nhận xét

……..
……..
……..
……..

- Tính ma trận B:

```
>>> B = np.array([ [0,-1], [-1,0]])
>>> print(B)
```

…….. ← sinh viên điền kết quả

```
>>> temp = B.dot(B)     # tính toan lần thứ 1
>>> print(temp)
```

…….. ← sinh viên điền kết quả

```
>>> k= 5
>>> for i in range(k-1):
        temp = temp.dot(B)
```

```
    print (temp)
    print('---')
```

…….. ← sinh viên điền kết quả và nhận xét

……..
……..
……..
……..

- ***Ma trận hội tụ - Convergent matrix:***

$$C = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{2} & 1 \\ 0 & 0 & \frac{1}{2} \end{pmatrix}$$

Hãy viết chương trình bằng Python để tính C^k và tính khi $k \to \infty$:

```
>>> C = np.array([ [1, 0, 0], [0, 0.5, 1], [0, 0, 0.5] ])
>>> print(C)
```

…….. ← sinh viên điền kết quả

```
>>> temp = C.dot(C)     # tính toan lần thứ 1
>>> print(temp)
```

…….. ← sinh viên điền kết quả

```
>>> k= 1000
>>> for i in range(k-1):
        temp = temp.dot(C)
>>> print(temp)
```

…….. ← sinh viên điền kết quả và nhận xét

……..
……..
……..
……..

Sau đó, thực hiện thêm 1 lần tích 1000 lần nữa:

```
>>> k= 1000
>>> for i in range(k-1):
        temp = temp.dot(C)
```

>>> print(temp)

…….. ← sinh viên điền kết quả và nhận xét

……..

……..

……..

……..

Gợi ý đáp án:

$$A^k = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{2^k} & \frac{k}{2^{k-1}} \\ 0 & 0 & \frac{1}{2^k} \end{pmatrix} \Longrightarrow A^\infty := \begin{pmatrix} \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

2.4. Ma trận Markov

Ma trận Markov được định nghĩa là ma trận vuông gồm các phần tử dương và **tổng các phần tử của theo cột sẽ bằng 1**. Trong thống kê, ý nghĩa chính của ma trận là sự chuyển hóa trạng thái, dự báo tương lai.

Ví dụ: Giả sử chúng ta chỉ có hai hãng giày A và B. Khách hàng của giày A lần sau mua lại sản phẩm của A là 80% và 20% khách hàng mua cho giày B. Ngược lại, khách hàng của hãng giày B 70% lần sau mua lại giày B và 30% mua giày A. Khi đó, chúng ta có ma trận gọi là Markov như sau:

$$M = \begin{pmatrix} 0.8 & 0.3 \\ 0.2 & 0.7 \end{pmatrix}$$

Theo lý thuyết, đặc tính của ma trận Markov M là dự báo khả năng cho các lần sau. Ví dụ: M^2 là khả năng mua ở lần thứ 2; M^3 là khả năng mua ở lần thứ 3;… Chúng ta có thể tính toán:

$$M^2 = \begin{pmatrix} 0.7 & 0.45 \\ 0.3 & 0.55 \end{pmatrix}, M^3 = \begin{pmatrix} 0.65 & 0.525 \\ 0.35 & 0.475 \end{pmatrix} \ldots, \boldsymbol{M^{100}} \approx \begin{pmatrix} \mathbf{0.6} & \mathbf{0.6} \\ \mathbf{0.4} & \mathbf{0.4} \end{pmatrix} = \boldsymbol{M^\infty}$$

Đến M^{100}, chúng ta bắt đầu thấy được sự hội tụ của các sản phẩm và thị phần khách hàng. Xét về mặt tính toán, việc tính tích các ma trận sẽ là công việc rất lớn, đặc biệt với ma trận Markov lớn thể hiện nhiều, lên đến hàng trăm sản phẩm (giả định như vậy). Do đó, về mặt công cụ toán học, chúng ta không tính tích ma trận mà chúng ta có thể sử dụng phương pháp giá trị riêng/vector riêng (các chương sau sẽ đề cập).

Sinh viên thực tập các lệnh sau:

>>> M = np.array([[0.8, 0.3], [0.2, 0.7]])

>>> MM = M.dot(M) **# tính M2**

>>> print (MM)

……

……

```
>>> MM = M.dot(M)   # tính M3
>>> print (MM)
……
……
>>> for i in range(100):   # tính các M^ khác.
        MM = MM.dot(M)
>>> print (MM)   # kết luận xu hướng của M^∞
……
……
```

Sinh viên thử lại các lệnh trên với ma trận M như sau:

$$M = \begin{pmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{pmatrix}$$

3. Phương trình ma trận

3.1. Khái niệm về tách ma trận

Một ứng dụng cơ bản khác của phép nhân ma trận là nén dữ liệu hoặc mã hóa dữ liệu. Ý tưởng cơ bản là bảng dữ liệu A sẽ được "tách" thành hai bảng dữ liệu B và C nhỏ hơn thỏa điều kiện nhân ma trận A = B x C và được truyền đồng thời trên các phương tiện khác nhau. Khi nhận đủ dữ liệu B và C, hệ thống sẽ khôi phục dữ liệu A bằng cách nhân hai ma trân B và C với nhau.

Ví dụ: Thay vì gửi khối dữ liệu ma trận (như ma trận ảnh hoặc ma trận về tỉ giá ngoại tệ tại các địa phương; giá các sản phẩm dầu khí của tập đoàn,… tại các địa phương/đại lý/chi nhánh):

$$\begin{pmatrix} a & b & c & d & e & f \\ a & b & c & d & e & f \\ a & b & c & d & e & f \\ a & b & c & d & e & f \end{pmatrix}$$

Thì chúng ta có thể gửi 2 vector. Từ 2 vector, ma trận sẽ được hình thành bằng phép nhân:

$$v_1 = \begin{pmatrix} a \\ b \\ c \\ d \\ e \\ f \end{pmatrix}; v_2 = (1 \quad 1 \quad 1 \quad 1)$$

Rõ ràng một ma trận có $m \times n$ phần tử khi phân rã thành $m + n$ phần tử thì việc chuyển trên mạng sẽ nhanh và chính xác, đồng thời, việc tính toán cũng nhanh chóng hơn khi bên nhận có những đặc điểm tính toán như: CPU/GPU và chung nền tảng phần mềm… Suy ra, với m và n lớn thì việc phân tách ma trận có $m \times n$ phần tử thành hai ma trận có $m \times k, k < n$ và $k \times n, k < m$ phần tử thì số phần tử luân chuyển sẽ giảm và đồng thời tăng tính bảo mật được dữ liệu.

3.2. Phân rã ma trận LU

LU Decomposition (phân rã LU) là tách ma trận A thành tích 2 ma trận L và U với:

- L là ma trận tam giác dưới, nghĩa là các phần tử bên trên đường chéo chính đều bằng 0.
- U là ma trận tam giác trên, nghĩa là các phần tử nằm dưới đường chéo chính đều bằng 0.

Minh họa với ma trận 4x4 như sau:

$$\begin{pmatrix} \alpha_{11} & 0 & 0 & 0 \\ \alpha_{21} & \alpha_{22} & 0 & 0 \\ \alpha_{31} & \alpha_{32} & \alpha_{33} & 0 \\ \alpha_{41} & \alpha_{42} & \alpha_{43} & \alpha_{44} \end{pmatrix} \cdot \begin{pmatrix} \beta_{11} & \beta_{12} & \beta_{13} & \beta_{14} \\ 0 & \beta_{22} & \beta_{23} & \beta_{24} \\ 0 & 0 & \beta_{33} & \beta_{24} \\ 0 & 0 & 0 & \beta_{44} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$$

Sinh viên thực hành lệnh trên gói linalg của scipy:

>>> M = np.array([[0.8, 0.3], [0.2, 0.7]])

>>> P, L, U = linalg.lu(M) **# lệnh lina.lu sẽ tách ma trận M thành 3 ma trận P, L và U**

>>> print (P)

……

>>> print (L)

……

>>> print (U)

……

Thử lại với phép nhân LU có bằng ma trận M ban đầu hay không:

>>> L.**dot**(U)

……

4. Bài toán ứng dụng 1 – Căn bản về mật mã học, mã hóa thông tin, mật khẩu

Mật mã được chuyển thể từ tiếng gốc là ***krytop*** nguồn gốc tiếng Hy Lạp, nghĩa là che dấu (hidden). Mật mã là một thông điệp được viết theo một mã bí mật. Ở giai đoạn sơ khai của mật mã, người ta mã hóa bằng cách "dịch chuyển" và "xoay vòng". Ví du: với bảng mã quy định từng con số dưới đây, chúng ta sẽ thực hiện việc mã hóa bằng cách dịch chuyển. Theo đó, giả định mã được bắt đầu từ số 05 là khoảng trắng như bảng bên dưới:

05 = ' '	11 = D	17 = I	23 = Ơ	29 = T	00 = (DẤU SẮC) '
06 = A	12 = Đ	18 = K	24 = Ô	30 = U	01 = (DẤU HUYỀN) `
07 = Ă	13 = E	19 = L	25 = P	31 = Ư	02 = (DẤU HỎI) ?
08 = Â	14 = Ê	20 = M	26 = Q	32 = V	03 = (DẤU NGÃ) ~
09 = B	15 = G	21 = N	27 = R	33 = X	04 = (DẤU NẶNG) .
10 = C	16 = H	22 = O	28 = S	34 = Y	35 = không có mã

Khi vượt quá giá trị 34 sẽ được "xoay vòng" về giá trị 0. Ví dụ: nếu chuỗi được dịch một đoạn là 5 số thì các mã thay đổi như sau: 'A' → 11 (=6+5) nghĩa là 'A' → 'D'; tương tự: 'B' = 14 (=9+5) nghĩa là 'B' biến thành 'Ê'... Ví dụ:

Với chuỗi cần mã hóa là: '**K H O A H O . C D Ư ~ L I Ê . U**' thì:

Mã của chuỗi là: 18 16 22 06 05 16 22 04 10 05 11 31 03 05 19 17 14 04 30

Và sau khi "dịch" mã, chuỗi mã hóa là: Ơ N R D C N R B C H ` Â C Ô O L B '

Trong trường hợp này, mã hóa đơn giản và khả năng tìm ra sẽ tăng lên nếu có nhiều văn bản vì sự "dịch chuyển" tuân theo quy luật cố định nên các từ thay thế sẽ tuân theo quy luật.

Bên cạnh đó, nhân ma trận là kỹ thuật đơn giản khác được sử dụng để mã hóa và giải mã thông điệp với mức độ "đoán" khó hơn. Cụ thể, cũng với giả định bảng mã trên và chuỗi cần mã hóa như cũ:

Chuỗi cần mã hóa: K H O A H O . C D Ư ~ L I Ê . U

Chuỗi số tương ứng: 13 11 17 01 00 11 17 **34** 05 00 06 26 **33** 00 14 12 09 **34** 25

Khi đó, người ta sẽ sử dụng một ma trận nxn bí mật để tính toán giá trị mã mới. Theo đó, chuỗi sẽ được cắt thành nhiều đoạn, mỗi đoạn có độ dài là k để thực hiện nhân theo dòng với ma trận khả nghịch

Sinh viên giải thích vì sao ma trận phải khả nghịch:

……

Ví dụ: kết quả khi chọn độ dài mỗi chuỗi là 3 là:

[13 11 17] [01 00 11] [17 34 05] [00 06 26] [33 00 14] [12 09 34] [25 35 35]

[K H O] [A _ H] [O . C] [_ D Ư] [~ _ L] [I Ê .] [U]

và giả định ma trận là:

$$E = \begin{pmatrix} 1 & -2 & 2 \\ -1 & 1 & 3 \\ 1 & -1 & -4 \end{pmatrix}$$

Và chúng ta thực hiện việc tính toán tích ma trận. Cụ thể, thành phần đầu tiên sẽ được mã hóa:

$$CHỮ'KHO': [13 \quad 11 \quad 17] \times \begin{pmatrix} 1 & -2 & 2 \\ -1 & 1 & 3 \\ 1 & -1 & -4 \end{pmatrix} = [19 \quad \mathbf{-32} \quad -9] \quad \rightarrow \textit{kết quả } 1$$

$$CHỮ'A\_H': [1 \quad 0 \quad 11] \times \begin{pmatrix} 1 & -2 & 2 \\ -1 & 1 & 3 \\ 1 & -1 & -4 \end{pmatrix} = [12 \quad -13 \quad \mathbf{-42}] \quad \rightarrow \textit{kết quả } 2$$

Rõ ràng so với mã hóa ở phương pháp bên trên, chữ H (giá trị 11) sẽ được mã hóa thành 2 giá trị hoàn toàn khác nhau (-32 và -42) nên việc (đoán) sẽ khó khăn.

Tương tự: sinh viên hãy sử dụng phần mềm để tính toán các giá trị tiếp theo:

$$CHỮ'O.C': [17 \quad 34 \quad 5] \times \begin{pmatrix} 1 & -2 & 2 \\ -1 & 1 & 3 \\ 1 & -1 & -4 \end{pmatrix} = [? \quad ? \quad ?] \quad \rightarrow \textit{kết quả } 3 \ldots\ldots\ldots\ldots\ldots$$

$$CHỮ'\_DƯ': [0 \quad 6 \quad 26] \times \begin{pmatrix} 1 & -2 & 2 \\ -1 & 1 & 3 \\ 1 & -1 & -4 \end{pmatrix} = [? \quad ? \quad ?] \quad \rightarrow \textit{kết quả } 4 \ldots\ldots\ldots\ldots\ldots$$

$$CHỮ'\sim L': [33 \quad 0 \quad 14] \times \begin{pmatrix} 1 & -2 & 2 \\ -1 & 1 & 3 \\ 1 & -1 & -4 \end{pmatrix} = [? \quad ? \quad ?] \quad \rightarrow \textit{kết quả } 5 \ldots\ldots\ldots\ldots\ldots$$

$$CHỮ\ 'IÊ.': [12 \quad 9 \quad 34] \times \begin{pmatrix} 1 & -2 & 2 \\ -1 & 1 & 3 \\ 1 & -1 & -4 \end{pmatrix} = [? \quad ? \quad ?] \quad \rightarrow kết\ quả\ 6 \dots\dots\dots\dots\dots$$

$$CHỮ\ 'U'\ và\ hai\ kí\ tự\ rỗng: [25 \quad 35 \quad 35] \times \begin{pmatrix} 1 & -2 & 2 \\ -1 & 1 & 3 \\ 1 & -1 & -4 \end{pmatrix} = [? \quad ? \quad ?] \quad \rightarrow kết\ quả\ 7$$

Từ các kết quả tìm được bên trên, hãy tìm cách phục hồi bằng cách lấy dãy mã hóa nhân với nghịch đảo ma trận. Sinh viên thực hiện:

5. Bài toán ứng dụng 2 – Bài toán loan tin

Bài toán truyền tin: Giả định ta có một nhóm n người $P_1, \dots, P_n$. Gọi ma trận A có $a_{ij} = 1$ nếu người i **có** thể gửi thông tin đến người j, và $a_{ij} = 0$ nếu người người i **không** thể gửi thông tin đến người j. Dễ thấy ma trận A là ma trận vuông.

Giả định, quy định các giá trị $a_{ii} = 0$ với mọi $i = 1, \dots, n$ và kí hiệu $P_i \rightarrow P_j$ nghĩa là P_i có thể gửi thông tin đến P_j. Ví dụ: cho ma trận sau:

$$A = \begin{pmatrix} 0 & 1 & 0 & \mathbf{1} \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ \mathbf{1} & 1 & 0 & 0 \end{pmatrix}$$

Nghĩa là trong ma trận A vừa cho (giả định chỉ số bắt đầu từ 1), ta có: $P_1 \rightarrow P_2$, $P_1 \rightarrow P_4$, $P_2 \rightarrow P_3$, $P_3 \rightarrow P_1$, $P_3 \rightarrow P_4 \dots$

Nhìn vào đây, chúng ta thấy $P_1 \rightarrow P_4$ và $P_4 \rightarrow P_1$. Do đó, 2 vị trí 1 và 4 luôn trao đổi thông tin với nhau (vì luôn có sự trao đổi hai chiều).

Từ đó, chúng ta thực hiện việc nhân ma trận A với chính nó, ta được A^2 như sau:

$$A^2 = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 2 & 0 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{1} \\ 1 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & \mathbf{1} & 0 & 1 \\ 0 & \mathbf{0} & 1 & 0 \\ 1 & \mathbf{0} & 0 & 1 \\ 1 & \mathbf{1} & 0 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & \mathbf{2} & 0 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix} = A^2$$

Giá trị a_{32}^2 được tính: $a_{32}^2 = \boldsymbol{a_{31}a_{12}} + a_{32}a_{22} + a_{33}a_{32} + \boldsymbol{a_{34}a_{42}} = \mathbf{1} + 0 + 0 + \mathbf{1} = 2 > 0$

Từ đây, chúng ta có thể thấy, kết quả thể hiện việc "truyền tin". Cụ thể hơn, có hai "cách"/"con đường" để P_3 có thể liên hệ với P_2 với **2** bước đi:

- Phải thông qua người 1: 3 → 4 → 2, nghĩa là: $P_3 \longrightarrow P_1 \wedge P_1 \longrightarrow P_2$
- Phải thông qua người 4: 3 → 4 → 2, nghĩa là: $P_3 \longrightarrow P_4 \wedge P_4 \longrightarrow P_2$

Lưu ý: $a_{23}^2 = 0$ *nghĩa là không có cách đi 2 bước từ* P_2 *sang* P_3.

Để dễ thấy hơn, chúng ta có thể tính toán ma trận bậc 3. Sinh viên thực hiện các câu lệnh sau:

```
>>> A = np.array([ [0,1,0,1],[0,0,1,0],[1,0,0,1],[1,1,0,0]])
```

>>> temp = A.dot(A) **# lúc này temp = A^2**

>>> print(temp)

…….. ← sinh viên điền kết quả và nhận xét

>>> temp = temp.dot(A) **# lúc này temp = A^3**

>>> print(temp)

…….. ← sinh viên điền kết quả và nhận xét

……..

Gợi ý kết quả tính toán ma trận mũ 3:

$$A^3 = \begin{pmatrix} 1 & 1 & 1 & 2 \\ 1 & 2 & 0 & 0 \\ 1 & 2 & 2 & 1 \\ 2 & 1 & 1 & 1 \end{pmatrix} = A^2A = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ \mathbf{1} & \mathbf{2} & \mathbf{0} & \mathbf{1} \\ 0 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 0 & \mathbf{1} & 0 & 1 \\ 0 & \mathbf{0} & 1 & 0 \\ 1 & \mathbf{0} & 0 & 1 \\ 1 & \mathbf{1} & 0 & 0 \end{pmatrix}$$

Rõ ràng với $a_{32}^3 = \boldsymbol{a_{31}^2 a_{12}} + a_{32}^2 a_{22} + a_{33}^2 a_{32} + \boldsymbol{a_{34}^2 a_{42}} = \mathbf{1} + 0 + 0 + \mathbf{1} = 2$

Nghĩa là ở giai đoạn **3**, từ P_3 đến P_2 cũng có 2 cách gửi thông tin (có **3 cạnh**).

Tổng quát hơn, tổng số "cách" để P_i (i là dòng) gửi thông tin đến P_j (j là cột) trong giai đoạn thứ k có giá trị là $A + A^2 + A^3 + \cdots + A^k$.

Từ đó, chúng ta có:

$$A + A^2 + A^3 = \begin{pmatrix} 2 & 3 & 2 & 3 \\ 2 & 2 & 1 & 1 \\ 3 & 4 & 2 & 3 \\ 3 & 3 & 2 & 2 \end{pmatrix}$$

Lưu ý: Tổng trên có nghĩa là số con đường đi từ P_2 đến P_3 qua trung gian 0, 1, 2 nốt chỉ có 1. Trong trường hợp này, đó là con đường đi trực tiếp từ P_2 đến P_3.

Sinh viên thực hành tính tổng các $\sum A^i$:

```
>>> A = np.array([ [0,1,0,1],[0,0,1,0],[1,0,0,1],[1,1,0,0]])
>>> sumA = A
>>> temp = A.dot(A)
>>> k = 3
>>> sumA = sumA + temp
>>> for i in range(1, k-1):
        temp = temp.dot(A)
        sumA = sumA + temp
>>> temp, sumA
```

…….. ← sinh viên điền kết quả và nhận xét

(Gợi ý: có ___ cách để từ P_3 sang P_4 với 0, 1, 2 cạnh trung gian)

Sinh viên có thể tham khảo thêm tài liệu tại: http://math.mit.edu/~gs/linearalgebra/ila0601.pdf

BÀI TẬP CHƯƠNG 3

Câu 1: Viết bài thu hoạch (làm từng bước như bài toán ứng dụng 1) về nội dung:

Sinh viên hãy viết chương trình bằng Python hoặc sử dụng thư viện numpy/sympy để:

a. Tự chọn một ma trận khả nghịch 3x3. Sinh viên chứng minh ma trận đó khả nghịch (tồn tại ma trận nghịch đảo)
b. Nhập họ và tên hoặc mã số sinh viên (của sinh viên).
c. Mã hóa họ và tên hoặc mã số sinh viên (của sinh viên).
d. Thực hiện giải mã với ma trận được chọn.

Câu 2: Tính toán phân số từ liên phân số (continued fractions)

Với một phân số $\frac{p_n}{q_n}$ được biểu diễn dưới dạng liên phân số như sau:

$$\frac{p_n}{q_n} = c_0 + \frac{1}{c_1 + \frac{1}{c_2 + \ddots + \frac{1}{c_n}}}$$

Người ta chứng minh được rằng có một cách xác định p_n và q_n như sau:

$$\begin{pmatrix} c_0 & 1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} c_1 & 1 \\ 1 & 0 \end{pmatrix}\dots\begin{pmatrix} c_n & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} p_n & p_{n-1} \\ q_n & q_{n-1} \end{pmatrix}, n = 0,1,2,\dots$$

a. Xây dựng chương trình tính p_n và q_n khi có danh sách n giá trị $c_0, c_1, \dots, c_n$.
b. Hãy chứng minh điều trên.