CHƯƠNG 2: MA TRẬN VÀ HỆ PHƯƠNG TRÌNH TUYẾN TÍNH

Mục tiêu:

- *Nắm vững được ma trận, các phép biến đổi sơ cấp, hệ phương trình tuyến tính.*
- *Minh họa sử dụng hệ phương trình tuyến tính để giải quyết một số bài toán.*

Nội dung chính:

1. Một số hàm về xử lý vector với Python

Trong bài trước, chúng ta thấy rằng chỉ đơn thuần Python có thể hỗ trợ cài đặt các phép xử lý cơ bản cho vector. Dưới đây, chúng ta tổng kết việc xử lý vector thông qua 4 hàm cơ bản:

- Hàm "scale" tỉ số vector bằng một giá trị thực: $\vec{v} = (a, b) \Longrightarrow k\vec{v} = (ka, kb)$

  >>> def ***scale***(a, v):

  return [a*vi for vi in v]

  >>> v = [3,5,7]

  >>> scale(10, v)

  …….. ← sinh viên điền kết quả

- Hàm lấy tổng hai vector: $\vec{v} = (a, b), \vec{w} = (c, d) \Longrightarrow \vec{v} + \vec{w} = (a + c, b + d)$

  >>> def ***sumvector***(v, w):

  return [vi+wi for (vi, wi) in zip(v, w)]

  >>> v = [3,5,7]

  >>> w = [2,4,6]

  >>> sumvector(v, w)

  …….. ← sinh viên điền kết quả

- Hàm nhân 2 vector vô hướng: $\vec{v} = (a, b), \vec{w} = (c, d) \Longrightarrow \vec{v} * \vec{w} = a \times c + b \times d$

  >>> def ***dotvector***(v, w):

  return sum([vi*wi for (vi, wi) in zip(v, w)])

  >>> dotvector(v, w)

  …….. ← sinh viên điền kết quả

- Hàm tính chiều dài một vector: $\vec{v} = (a, b) \Longrightarrow \vec{v} * \vec{v} = a \times a + b \times b$

  >>> def ***lenvector***(v):

  return dotvector(v,v)

  >>> lenvector(w)

  …….. ← sinh viên điền kết quả

Tuy vậy, nhược điểm xử lý thuần Python là hỗ trợ các vector một chiều. Tính toán các vector nhiều chiều với những hàm tự viết sẽ không hiệu quả bằng việc sử dụng các thư viện được xây dựng, như tốc độ tính toán trong các gói Numpy và Scipy đã được tối ưu đồng thời các phép toán được cài đặt về cơ bản theo toán học là đầy đủ.

2. Bài toán ứng dụng 1 – Phân loại tuyến tính

Phân loại tuyến tính (linear classifiers) là một khái niệm trong trí tuệ nhân tạo, cụ thể hơn là trong máy học. Ý nghĩa của nó là việc tính toán ra điểm số (score) của một vector x đưa vào hệ thống. Cụ thể, đó là hệ:

$$Score = w.x$$

Trong đó,

- x là vector các đặc trưng mà chúng ta thu thập được và mong muốn phân loại;
- w là vector thể hiện sự quan trọng của các đặc trưng mà bộ phân loại;
- Phép nhân là tích giữa hai vector trên.

Thông thường, nếu phân thành hai loại thì chúng ta gọi đó là phân loại nhị phân (binary classification). Khi đó, nếu điểm (score) vượt ngưỡng nào đó (thường là 0) thì sẽ thuộc nhóm 1 và ngược lại là nhóm 2.

Lưu ý: Trong một số trường hợp, hệ phân loại tuyến tính sẽ thêm một vector gọi là intercept b:

$$Score = w.x + b$$

Giả định bỏ qua giá trị b, chỉ xét phương trình bên trên.

Chúng ta thử nghiệm các đoạn lệnh dưới đây để thấy khả năng xử lý của Python với một phân loại tuyến tính:

>>> import numpy as np

>>> scores = np.**array**([-1, 1, 2, -3, 5, -4])

…….. ← sinh viên điền kết quả

>>> scores >= 0

…….. ← sinh viên điền kết quả

>>> scores < 0

…….. ← sinh viên điền kết quả

>>> **np.select**([scores >=0, scores < 0],[‘so duong’, ‘so am’])

…….. ← sinh viên điền kết quả

……

Kết quả câu lệnh bên trên là phương cách phân loại dữ liệu, theo đó, lệnh select phân thành 2 loại. Ví dụ dưới đây cho ta thấy lệnh np.select có thể phân loại thành 3 loại:

>>> scores = np.array([-1, 1, 2, 0, -3, 5, 0, -4])

>>> **np.select**([scores >0, scores ==0, scores < 0],['so duong', 'so 0', 'so am'])

…….. ← sinh viên điền kết quả

……

3. Thực hành xử lý ma trận

3.1. Cơ bản về xử lý ma trận

Một ma trận là dãy các số theo hai chiều, còn gọi là danh sách của danh sách các số/phần tử. Khi đó, các ma trận chỉ có 1 dòng hoặc chỉ có 1 cột là các ma trận đặc biệt. Ngoài ra, một dạng ma trận đặc biệt khác là ma trận thưa. Ma trận thưa là ma trận có nhiều phần tử mang giá trị 0 và ít phần tử mang giá trị khác 0. Xử lý các ma trận thưa kích thước lớn là một vấn đề lớn trong khoa học khi cần tăng tốc và giảm bộ nhớ lưu trữ.

Theo đó, trong **numpy**, để khai báo ma trận, chúng ta có các lệnh cơ bản như sau:

- ***np.mat***: để khai báo một ma trận
- ***np.asmatrix***: để chuyển đổi một vector thành một ma trận.
- ***np.random.random((m,n))***: tạo ra một bảng số ngẫu nhiên có m dòng và n cột.

Thực hành: Sinh viên thực hiện ôn luyện các lệnh sau:

STT	Lệnh thực hành	Diễn giải/Kết quả
1	>>> import numpy as np >>> from scipy import linalg, sparse	Lệnh để nạp thư viện numpy vào bộ nhớ để xử lý. Tải 2 thư viện linalg và sparse của scipy vào bộ nhớ để sử dụng.

2	>>> D = np.**mat**([[3,4], [5,6]])
	>>> print D
	Lưu ý: với phiên bản 3.x, lệnh là >>> **print (D)**
		……
		……

3	>>> C = np.**mat**(np.random.random((5,7))) >>> print C	…… …… …… …… ……
4	>>> A = np.**mat**(np.random.random((2,2))) >>> print A	…… ……

5	>>> b = np.array([(1+5j, 2j, 3j),(4, 5, 6)])
	>>> B = np.**asmatrix**(b)
	>>> print b
	>>> print B
		……
		……
		……
		……

6	>>> A.**T**	Chuyển vị ma trận (đảo cột thành dòng và ngược lại): …… ……

7	>>> A.**I**
	Hoặc sử dụng lệnh của thư viện scipy:
	>>> **linalg.inv**(A)
		Ma trận nghịch đảo:
		……
		……

8	>>> M = np.array([[-1,3,2],[0,-2,1],[1,5,-2]])
	>>> M_lower = np.**tril**(M)
	>>> print(M_lower)
		Ma trận dưới từ đường chéo:
		……
		……
		……

9	>>> M = np.array([[-1,3,2],[0,-2,1],[1,5,-2]]) >>> M_upper = np.**triu**(M)	Ma trận trên từ đường chéo: …… ……

```
>>> print(M_upper)
```
……

```
>>> M = np.array([[-1,3,2],[0,-2,1],[1,5,-2]])
>>> v_diag = np.diag(M) #vector đường chéo
>>> print (v_diag)
>>> M_diag = np.diag(v_diag)
>>> print (M_diag)
```
Vector và ma trận đường chéo:
……
……
……

Ma trận đơn vị I_n là ma trận có đường chéo là 1, các phần tử khác là 0.

Ví dụ: $I_2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ bằng lệnh: **I2 = np.identiy(2)**.

Thực hiện tính toán đơn giản trên ma trận

Bài toán xác định hai ma trận bằng nhau:

Ta có: hai vector/ma trận bằng nhau khi các phần tử của chúng bằng nhau.

Bài tập dẫn nhập: Hãy xác định x, y và z để 2 ma trận bằng nhau:

$$A = \begin{bmatrix} 2 & x^2+2 & 3 \\ 6 & 0 & 1 \\ 8 & z^2+4 & y-z \end{bmatrix}, B = \begin{bmatrix} 2 & 6 & 3 \\ y-1 & 0 & x+z \\ 2x^2 & 5 & 8 \end{bmatrix}$$

Giải: Nhìn vào 2 ma trận trên, để hai ma trận trên bằng nhau thì mọi phần tử của chúng phải bằng nhau. Nghĩa là hệ phương trình cần giải là:

$$A_{12} = B_{12}, A_{21} = B_{21}, A_{23} = B_{23}, A_{31} = B_{31}, A_{32} = B_{32}, A_{33} = B_{33}$$
$$x^2 + 2 = 6;\ y - 1 = 6; x + z = 1; 2x^2 = 8; z^2 + 4 = 5; y - z = 8$$

Giải hệ trên, các nghiệm chúng ta tìm được là:

…….. ← Sinh viên tự giải x, y, z

Hướng dẫn giải:

$$y - 1 = 6 \Longrightarrow y = \cdots$$
$$y - z = 8 \Longrightarrow z = y - 8 = \cdots$$
$$x + z = 1 \Longrightarrow x = 1 - z = \cdots$$

Thử lại giá trị x, y, z vào các phương trình khác để kiểm sự hợp lý:

$$x^2 + 2 = 6: \text{Đúng hoặc sai với } x = ..$$
$$2x^2 = 8: \text{Đúng hoặc sai với } x = ..$$
$$z^2 + 4 = 5: \text{Đúng hoặc sai với } z = ..$$

Lưu ý: Bên cạnh đó, để giải được hệ trên, chúng ta có thể sử dụng thư viện Sympy. Với Sympy, các biến được khai báo là những đối tượng Symbol và các phương trình được lập thành danh sách như sau:

```
>>> import sympy as sp
>>> x = sp.Symbol('x')
>>> y = sp.Symbol('y')
>>> z = sp.Symbol('z')
>>> sp.solve([x*x+2-6, y-1-6,x+z-1, 2*x*x-8, z*z+4-5,y-z-8], [x, y, z])
```

…….. ← Sinh viên điền kết quả vào

Trong Python, để kiểm tra các phần tử của 2 danh sách có giá trị bằng nhau là sử dụng từ khóa **all**:

```
>>> x = [1,2,3]
>>> y = [1,2,3]
>>> print all([x[i]==y[i] for i in range(len(x))])
True
>>> y = [1,2,4]
>>> print all([x[i]==y[i] for i in range(len(x))])
False
>>>
```

Với numpy, lệnh so sánh hai ma trận là: ***numpy*.array_equal(a1, a2)** với a1, a2 là 2 array/dãy số.

```
>>> import numpy as np
>>> np.array_equal([1, 2], [1, 2])
True
>>> np.array_equal(np.array([1,2]), np.array([1,2]))
True
>>>
```

Sinh viên tự tìm hiểu thêm (nghiên cứu) sự khác biệt giữa hai lệnh sau:

- numpy.**array_equiv**(A, B)
- numpy.**allclose**(A, B,…)

3.2. Các phép biến đổi sơ cấp trên ma trận

Một số lệnh thực hành về các phép tính toán, biến đổi trên ma trận

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

Xây dựng ma trận 6x6 A với các phần tử và ma trận đơn vị 6x6 và ma trận I_6:

$$A = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 & 11 \\ 12 & 13 & 14 & 15 & 16 & 17 \\ 18 & 19 & 20 & 21 & 22 & 23 \\ 24 & 25 & 26 & 27 & 28 & 29 \\ 30 & 31 & 32 & 33 & 34 & 35 \end{bmatrix}; I = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

>>> A = np.reshape(np.arange(36.0), (6,6))

>>> print A **# hoặc print(A) ở phiên bản 3.x**

…….. ← sinh viên viết/mô tả kết quả câu lệnh

>>> I6 = np.identity(6)

>>> print I6

…….. ← sinh viên viết/mô tả kết quả câu lệnh

Sau đó, xem lại kích thước của ma trận (số lượng phần tử) và in đường chéo của ma trận A:

>>> A.size

…….. ← sinh viên viết/mô tả kết quả câu lệnh

>>> np.matrix.diagonal(A)

…….. ← sinh viên viết/mô tả kết quả câu lệnh

Thành lập ma trận A mới bằng công thức $A = A + I$

>>> A = A + I6

>>> print A

…….. ← sinh viên viết/mô tả kết quả câu lệnh

Tính tích (nhân) ma trận với vector (là dạng ma trận đặc biệt):

$$vecB = \begin{bmatrix}1.\\2.\\3.\\4.\\5.\\6.\end{bmatrix}; C = A \times vecB = \begin{bmatrix}1 & 1 & 2 & 3 & 4 & 5\\6 & 8 & 8 & 9 & 10 & 11\\12 & 13 & 15 & 15 & 16 & 17\\18 & 19 & 20 & 22 & 22 & 23\\24 & 25 & 26 & 27 & 29 & 29\\30 & 31 & 32 & 33 & 34 & 36\end{bmatrix} \times \begin{bmatrix}1.\\2.\\3.\\4.\\5.\\6.\end{bmatrix} = \begin{bmatrix}a\\b\\c\\c\\d\\e\end{bmatrix}$$

$$với\ a = 1 \times 1 + 1 \times 2 + 2 \times 3 + 3 \times 4 + 4 \times 5 + 5 \times 6; \ldots$$

Sinh viên thực hiện các lệnh sau và ghi kết quả:

>>> vecB = np.array([1., 2., 3., 4., 5., 6.])

>>> C = A.**dot**(vecB)

>>> print C

…….. ← sinh viên viết/mô tả kết quả câu lệnh

Khai báo ma trận D 2x6

$$D = \begin{bmatrix}1 & 2 & 3 & 4 & 5 & 6\\1 & 0 & 1 & 0 & 1 & 0\end{bmatrix}$$

>>> D = np.array([[1., 2., 3., 4., 5., 6.], [1., 0., 1., 0., 1., 0.]])

>>> print D

…….. ← sinh viên viết/mô tả kết quả câu lệnh

Tính tích ma trận A và D: $A \times D$

>>> E = A.**dot**(D)

…….. ← sinh viên viết/mô tả kết quả câu lệnh

Xây dựng ma trận F kích thước 6x2:

$$F = \begin{bmatrix} 1 & 1 \\ 2 & 0 \\ 3 & 1 \\ 4 & 0 \\ 5 & 1 \\ 6 & 0 \end{bmatrix}$$

>>> F = np.array([[1., 1.], [2., 0.], [3., 1.], [4., 0], [5., 1], [6., 0.]])

>>> G = A.dot(F)

>>> print F

…….. ← sinh viên viết/mô tả kết quả câu lệnh

……..

……..

……..

……..

……..

>>> print G

…….. ← sinh viên viết/mô tả kết quả câu lệnh

……..

……..

……..

……..

……..

Tính ma trận nghịch đảo:

>>> np.linalg.inv(A)

…….. ← sinh viên viết/mô tả kết quả câu lệnh

……..

……..

……..

……..

……..

Câu hỏi: Lệnh sau sẽ trả về kết quả gì?

>>> np.linalg.inv(np.linalg.inv(A))

Trả lời: ……..

[Gợi ý: *sinh viên thử với các ví dụ thử nếu A khả nghịch và A không khả nghịch?*]

Lưu ý: Sinh viên có thể tìm hiểu thêm tại địa chỉ:

https://www.python-course.eu/matrix_arithmetic.php

4. Bài toán ứng dụng 2 – Tính toán dãy Fibonacci: Con đường tìm đến tỉ số vàng!

Dãy số Fibonacci là một khái niệm đẹp trong toán học với nhiều ứng dụng trong nghệ thuật, âm nhạc, thiết kế các mẫu, kiến trúc,... Dãy số Fibonacci bắt đầu từ 0, 1, 1, 2, 3, 5, 8,… với quy luật số sau bằng tổng hai số trước nó. Nghĩa là, gọi F là dãy Fibonacci, chúng ta có:

$$F_n = F_{n-1} + F_{n-2}$$

Để xây dựng một mô hình hệ thống tuyến tính, chúng ta cần bổ sung thêm một phương trình xác định F_{n-1}:

$$F_{n-1} = F_{n-1} + (0).F_{n-2}$$

Từ hai phương trình trên, chúng ta có thể xây dựng được hệ phương trình tuyến tính:

$$\begin{cases} F_n = F_{n-1} + F_{n-2} \\ F_{n-1} = F_{n-1} + (0).F_{n-2} \end{cases}$$

Thể hiện dưới dạng ma trận là:

$$\begin{bmatrix} F_n \\ F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} F_{n-1} \\ F_{n-2} \end{bmatrix}$$

Với giá trị vector ban đầu là: $[F_1 \quad F_0]^T = [1 \quad 0]^T$

Từ đó, chúng ta có thể tính toán các giá trị của dãy số Fibonacci bằng phép nhân ma trận với vector. Sinh viên thực hiện đoạn lệnh tính toán 11 số Fibonacci đầu tiên:

```
>>> import numpy as np
>>> A = np.array( [ [1,1], [1,0] ] )
>>> b = np.array([1, 0])
>>> n = 10
>>> for i in range(n):
        b = A.dot(b)
        print(b)
```

…… ← Sinh viên ghi nhận kết quả.

Nhận xét: Mô hình trên là mô hình tuyến tính động (dynamic linear model) vì giá trị sau được tính từ giá trị trước. Do đó, chúng ta sẽ quay trở lại với các ứng dụng của dãy Fibonacci trong những chương sau.

5. Cơ bản về hệ phương trình tuyến tính và ứng dụng minh họa

5.1. Làm quen với giải hệ phương trình tuyến tính

Hệ phương trình tuyến tính là hệ các phương trình với các ẩn số đều bậc nhất, được định nghĩa sau:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\ \cdots\ \cdots\ \cdots\ \cdots\ \cdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n = b_m \end{cases}$$

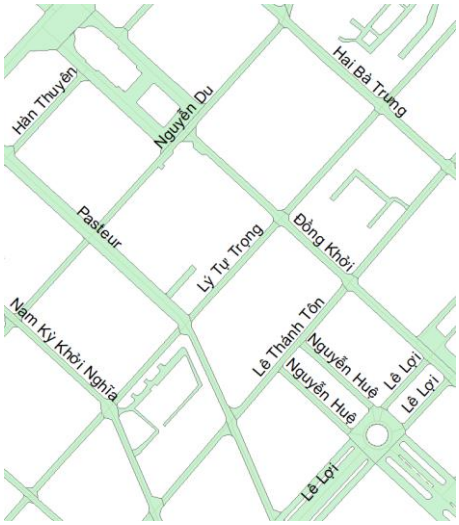
Trong đó, $x_1, x_2, \ldots, x_n$ là các biến thực cần tìm và a_{ij}, b_j là các hằng số thực.

Quy trình giải hệ tuyến tính với numpy là:

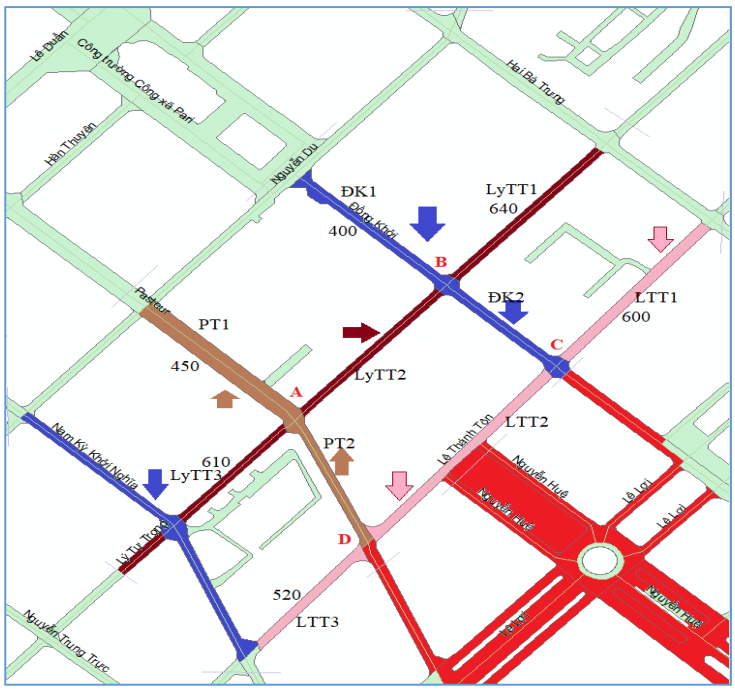
- **Bước 1: Thành lập ma trận A và vector với các hệ số a_{ij}, b_j.**
- **Bước 2: Tính toán ma trận nghịch đảo A^{-1} của A.**
- **Bước 3: Tìm vector nghiệm x bằng việc nhân ma trận nghịch đảo với vector.**

5.2. Bài toán ứng dụng 3 – Đếm số lượng xe vào khu vực trung tâm

Cho bản đồ khu vực trung tâm Thành phố Hồ Chí Minh (hình bên trái), gồm phố đi bộ Nguyễn Huệ và những con đường xung quanh như Đồng Khởi, từ Lý Tự Trọng, Pasteur từ Lê Thánh Tôn. Trong dịp Tết 2019 vừa qua, các tòa nhà cơ quan đều nghỉ lễ (không có người ra/vào cơ quan) và tại khu trung tâm phố đi bộ đã cấm các phương tiện giao thông ra vào (tô màu đỏ ở hình bên phải). Các camera được gắn tại các đoạn đường để đếm lưu lượng xe. Hệ thống camera ghi nhận được lưu lượng xe tại các đoạn đường như trong hình bên phải (và liệt kê trong bảng bên dưới).



Hình khu vực vị trí Trung tâm TP.HCM thu nhỏ



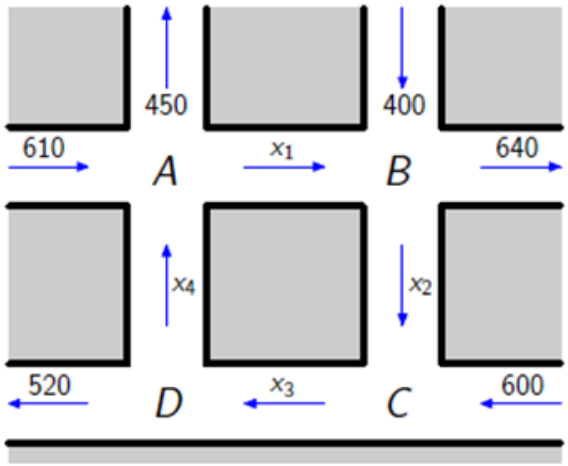
Hình thông tin số phương tiện tại các đoạn đường

Bảng lưu lượng xe tại các đoạn đường:

Mã đường	Đoạn đường	Chiều đi từ đoạn đường	Đến đoạn đường	Số lượng phương tiện

ĐK1	Đồng Khởi	Nguyễn Du	Lý Tự Trọng	400
ĐK2	Đồng Khởi	Lý Tự Trọng	Lê Thánh Tôn	Không có số liệu
LTT1	Lê Thánh Tôn	Hai Bà Trưng	Đồng Khởi	600
LTT2	Lê Thánh Tôn	Đồng Khởi	Pasteur	Không có số liệu
LTT3	Lê Thánh Tôn	Pasteur	Nam Kì Khởi Nghĩa	520
LyTT1	Lý Tự Trọng	Nam Kì Khởi Nghĩa	Pasteur	640
LyTT2	Lý Tự Trọng	Pasteur	Đồng Khởi	Không có số liệu
LyTT3	Lý Tự Trọng	Đồng Khởi	Hai Bà Trưng	610
PT1	Pasteur	Nguyễn Du	Lý Tự Trọng	450
PT2	Pasteur	Lý Tự Trọng	Lê Thánh Tôn	Không có số liệu

Hãy tính toán số lượng xe trong các đoạn đường hệ thống camera không có số liệu. Với tên gọi của các ngã tư lần lượt là A, B, C, D tương ứng như sau:



- A: Lý Tự Trọng – Pasteur.
- B: Lý Tự Trọng – Đồng Khởi.
- C: Lê Thánh Tôn – Đồng Khởi.
- D: Lê Thánh Tôn – Pasteur.

Sinh viên thực hành giải quyết vấn đề:

Số lượng xe giao thông tại các nút giao thông A, B, C và D lần lượt thỏa mãn các phương trình:

- Tại giao lộ A: PT2 + LyTT3 = PT1 + LyTT2 ↔ PT2 + 610 = 450 + LyTT2
- Tại giao lộ B: LyTT2 + ĐK1 = LyTT1 + ĐK2 ↔ LyTT2 + 400 = 640 + ĐK2
- Tại giao lộ C: LTT1 + ĐK2 = LTT2 ↔ 600 + DDK2 = LTT2
- Tại giao lộ D: LTT2 = PT2 + LTT3 ↔ LTT2 = PT2 + 520

Đặt $x_1 = LyTT3;\ x_2 = PT1; x_3 = ĐK2;\ x_4 = LTT2$ khi đó, ta có hệ phương trình:

$$\begin{cases} x_4 + 610 = 450 + x_1 \\ x_1 + 400 = x_2 + 640 \\ x_2 + 600 = x_3 \\ x_3 = x_4 + 520 \end{cases}$$

Hoặc hệ tương đương:

$$\begin{cases} x_4 - x_1 = -160 \\ x_1 - x_2 = 240 \\ x_2 - x_3 = -600 \\ x_3 - x_4 = 520 \end{cases}$$

Trong gói numpy, hệ phương trình bên trên được giải như sau:

>>> import numpy as np

>>> A = np.matrix([[1,-1,0,0],[0,1,-1,0],[0,0,1,-1],[-1,0,0,1]])

>>> b = np.matrix([[-160],[240],[-600],[520]])

Lưu ý: trong một số phiên bản (3.x), lệnh dưới đây được chọn 1 trong 2:

Hoặc lệnh: **>>> A_nghichdao = np.linalg.inv(A) # báo ma trận singular!**

Hoặc lệnh: **>>> A_nghichdao = np.invert(A) # không giống ma trận nghịch đảo**

Sau đó, chúng ta có thể tìm giá trị X:

>>> X = A_nghichdao * b

>>> X

…….. ← sinh viên điền kết quả

……..

……..

……..

BÀI TẬP CHƯƠNG 2

Câu 1: Hãy sử dụng numpy để giải các phương trình sau:

- Vấn đề 1 (Problem 1): Tìm điểm giao giữa hai đường thẳng trong $\mathbb{R}^2$.
- Vấn đề 2 (Problem 2): Tìm giao điểm giữa ba mặt phẳng trong $\mathbb{R}^3$.
- Vấn đề 3 (Problem 3): Tìm các hệ số đa thức để đa thức thỏa các nghiệm.
- Vấn đề 4 (Problem 4)Tìm các hệ đa thức khi phân rã để tính tích phân.

Applications

Problem 1. Find the point of intersection of the lines $x - y = -2$ and $2x + 3y = 6$ in $\mathbb{R}^2$.

$$\begin{cases} x - y = -2 \\ 2x + 3y = 6 \end{cases}$$

Problem 2. Find the point of intersection of the planes $x - y = 2$, $2x - y - z = 3$, and $x + y + z = 6$ in $\mathbb{R}^3$.

$$\begin{cases} x - y = 2 \\ 2x - y - z = 3 \\ x + y + z = 6 \end{cases}$$

Method of undetermined coefficients often involves solving systems of linear equations.

Problem 3. Find a quadratic polynomial $p(x)$ such that $p(1) = 4$, $p(2) = 3$, and $p(3) = 4$.

Suppose that $p(x) = ax^2 + bx + c$. Then $p(1) = a + b + c$, $p(2) = 4a + 2b + c$, $p(3) = 9a + 3b + c$.

$$\begin{cases} a + b + c = 4 \\ 4a + 2b + c = 3 \\ 9a + 3b + c = 4 \end{cases}$$

Problem 4. Evaluate $\int_0^1 \frac{x(x-3)}{(x-1)^2(x+2)}\,dx$.

To evaluate the integral, we need to decompose the rational function $R(x) = \frac{x(x-3)}{(x-1)^2(x+2)}$ into the sum of simple fractions:

$$\begin{aligned} R(x) &= \frac{a}{x-1} + \frac{b}{(x-1)^2} + \frac{c}{x+2} \\ &= \frac{a(x-1)(x+2) + b(x+2) + c(x-1)^2}{(x-1)^2(x+2)} \\ &= \frac{(a+c)x^2 + (a+b-2c)x + (-2a+2b+c)}{(x-1)^2(x+2)}. \end{aligned}$$

$$\begin{cases} a + c = 1 \\ a + b - 2c = -3 \\ -2a + 2b + c = 0 \end{cases}$$

Câu 2: Hãy viết các câu lệnh của **sympy** để giải các phương trình ở **Câu 1**.

Câu 3: Tính lũy thừa bậc k của ma trận $\boldsymbol{F_k} = \begin{pmatrix} \mathbf{1} & \mathbf{1} \\ \mathbf{1} & \mathbf{0} \end{pmatrix}^{\boldsymbol{k}}$ và chứng minh ma trận $\boldsymbol{F_k}$ chứa 3 phần tử của $Fibo_{k+1}$, $Fibo_k$ và $Fibo_{k-1}$ của dãy Fibonacci (bắt đầu bằng 0 từ phần tử 0)