THỰC HÀNH ĐẠI SỐ TUYẾN TÍNH

TÀI LIỆU PHỤC VỤ SINH VIÊN NGÀNH KHOA HỌC DỮ LIỆU

Nhóm Giảng viên biên soạn: TS. Hoàng Lê Minh – Khưu Minh Cảnh – Hoàng Thị Kiều Anh – Lê Công Hiếu – Trần Ngọc Việt – Nguyễn Công Nhựt – Đỗ Đình Thủ - Nguyễn Thị Thanh Bình – Trần Kim Mỹ Vân – Phạm Trọng Nghĩa – Nguyễn Hữu Trí Nhật – Lê Thị Ngọc Huyên – Nguyễn Thị Tuyết Mai – Nguyễn Viết Cường – Huỳnh Thái Học và các Giảng viên khác.

TP.HCM - Năm 2020

MỤC LỤC

CHƯƠNG 6: CÁC KHÁI NIỆM: KHÔNG GIAN VECTOR, TÍCH TRONG VÀ TRỊ RIÊNG, VECTOR RIÊNG

Mục tiêu:

- *Khái niệm về không gian véc tơ, tổ hợp tuyến tính.*
- *Tích trong (inner product).*
- *Trị riêng, vector riêng của ma trận và ứng dụng.*

Nội dung chính:

1. Khái niệm về không gian vector và tích trong

1.1.Tóm tắt lý thuyết

Một không gian vector bao gồm:

- Một tập hợp $\mathcal{V}$.
- Một phép "cộng" + trên $\mathcal{V}$: $\mathcal{V} \times \mathcal{V} \to \mathcal{V}$
- Một phép nhân số: $\mathbb{R} \times \mathcal{V} \to \mathcal{V}$
- Một phần tử trung hòa (distinguised element) đối với phép "cộng": $0 \in \mathcal{V}$

Và thỏa các tính chất sau:

- Tính giao hoán (commutative): $x + y = y + x, \forall x, y \in \mathcal{V}$
- Tính kết hợp (associative): $(x + y) + z = x + (y + z), \forall x, y, z \in \mathcal{V}$
- Phần tử trung hòa: $0 + x = x, \forall x \in \mathcal{V}$
- Phần tử nghịch đảo: $\forall x \in \mathcal{V}, \exists(-x) \in \mathcal{V}: x + (-x) = 0$
- $(\alpha\beta)x = \alpha(\beta x), \forall \alpha, \beta \in \mathbb{R}, x \in \mathcal{V}$
- Phân phối với số thực: $\alpha(x + y) = \alpha x + \alpha y, \forall \alpha \in \mathbb{R}, x, y \in \mathcal{V}$
- Phân phối với vector: $(\alpha + \beta)x = \alpha x + \beta x, \forall \alpha, \beta \in \mathbb{R}, x \in \mathcal{V}$
- $1x = x, x \in \mathcal{V}$

Các ví dụ về các không gian vector:

- $\mathcal{V}_1 = \mathbb{R}^n$ với 2 phép toán cộng (vector) và nhân (số thực).
- $\mathcal{V}_2 = \{0\}, với\ 0 \in \mathbb{R}^n$.
- $\mathcal{V}_3 = bộ\ (v_1, v_2, \dots, v_k) = \{\alpha_1 v_1 + \alpha_2 v_2 + \cdots + \alpha_k v_k | \alpha_i \in \mathbb{R}\}\ và\ \ v_1, v_2, \dots, v_k \in \mathbb{R}^n$

Không gian vector con (subspaces):

Là tập con của không gian vector. Ví dụ các không gian $\mathcal{V}_1$, $\mathcal{V}_2$ và $\mathcal{V}_3$ bên trên là các không gian con của không gian $\mathbb{R}^n$.

Độc lập tuyến tính:

Tập các vector $\{v_1, v_2, \dots, v_k\}$ gọi là độc lập tuyến tính nếu:

$$\alpha_1 v_1 + \alpha_2 v_2 + \cdots + \alpha_k v_k = 0 \Longrightarrow \alpha_1 = \alpha_2 = \cdots = \alpha_k = 0$$

Lưu ý các điều kiện tương đương:

- Các hệ số α_i là duy nhất, nghĩa là:

  $$\alpha_1 v_1 + \alpha_2 v_2 + \cdots + \alpha_k v_k = \beta_1 v_1 + \beta_2 v_2 + \cdots + \beta_k v_k$$

  khi và chỉ khi

  $$\alpha_1 = \beta_1, \alpha_2 = \beta_2, \dots, \alpha_k = \beta_k$$

- Không vector v_i được biểu diễn như một tổ hợp tuyến tính của các vector $v_1, \dots, v_{i-1}, v_{i+1}, \dots, v_k$

Cơ sở và số chiều:

Tập các vector $\{v_1, v_2, \dots, v_k\}$ là cơ sở của không gian $\mathcal{V}$ nếu:

- $\{v_1, v_2, \dots, v_k\}$ độc lập tuyến tính hoặc mọi $v \in \mathcal{V}$ có thể được biểu diễn duy nhất:

  $$v = \alpha_1 v_1 + \alpha_2 v_2 + \cdots + \alpha_k v_k$$

Cho một không gian vector $\mathcal{V}$, số lượng các vector trong cơ sở bằng với số lượng vector trong bất kì cơ sở nào của nó và được gọi là **số chiều** của $\mathcal{V}$, kí hiệu là $dim\mathcal{V}$

Kí hiệu: $\dim\{0\} = 0$ và $dim\mathcal{V} = \infty$ nếu $\mathcal{V}$ không có cơ sở.

Hạng của ma trận:

$$rank(A) = \dim\mathcal{R}\,(A)$$

Các tính chất của hạng ma trận:

- $rank(A) = rank(A^T)$
- $rank(A)$ là số cột (hoặc dòng) độc lập lớn nhất của ma trận A.
- Từ đó, ta có mệnh đề: $rank(A) + dim = n$

1.2.Tích vô hướng, trực giao và ứng dụng

Một không gian tích trong (**inner product space**) V là một không gian vector được trang bị một phép toán $<,>$ tính giá trị cho mọi cặp vector $u, v \in V$ và $w \in V$ với các tính chất như sau:

- $< u, u > \geq 0$, biểu thức sẽ bằng 0 nếu $u = 0$
- $< u, v > = < v, u >$: tính giao hoán
- $< \alpha u + v, w > = \alpha < u, w > + < v, w >$

Xét tích trong theo định nghĩa sau với $x, y \in V$

$$< x, y > := x^T y = x_1 y_1 + x_2 y_2 + \cdots + x_n y_n$$

Dễ thấy hàm $f(y) = < x, y >$ là một hàm tuyến tính từ $R^n \to R$. Ví dụ: trong $\mathbb{R}^3$, xét hai vector $a = (1,2,3)$ và $b = (4,5,6)$, khi đó tích vô hướng của hai vector là: 1*4+2*5+3*6 = 32.

Thể hiện ở Python là:

```
>>> import numpy as np

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

>>> b = np.array([4,5,6])

>>> tich = np.inner(a,b)  # tích vô hướng

>>> print (tich)
```

…….

*Lưu ý thêm: nếu vector ở dạng 1 dòng, cả hai hàm **numpy.inner**(a,b) và **numpy.dot**(a,b) đều có kết quả như nhau. Xét a và b như trên, ta có:*

```
>>> np.dot(a,b)                # kết quả có giống như np.inner(a,b)?
```

……

Tuy nhiên, khi biến vector ở dạng tập hợp các vector (như ma trận 2 chiều gồm nhiều dòng), cơ chế tính toán của numpy.dot và numpy.inner sẽ khác nhau. Cụ thể là:

- *Hàm **numpy.dot*** tính tích hai ma trận. Do đó, yêu cầu của 2 vector nếu không phải vector 1 chiều thì phải đảm bảo cho việc nhân ma trận (cụ thể là: $(m, n) \times (n, p) = (m, p)$).
- *Hàm **numpy.inner*** tính toán tích vô hướng của các vector; tuy nhiên, nếu biến ở dạng tập các vector, thì các vector của tập đó cần đảm bảo cùng số lượng phần tử (nghĩa là một mà trận). Ví dụ: chúng ta không thể .inner 2 vector này: $p = array([[1,2,3], [\mathbf{1}, \mathbf{1}, \mathbf{1}, \mathbf{1}]])$ và $q = array([[4,5,6], [\mathbf{2}, \mathbf{2}, \mathbf{2}, \mathbf{2}]])$. Nếu tính inner của 2 vector 2x2 sẽ ra ma trận 2x2.

[Đọc thêm] Với tích vô hướng, ta có một số định nghĩa và tính chất như sau:

- **Độ dài/chuẩn của vector:**
  Chuẩn 2: $\|u\| = \sqrt{< u, u >}$ (thường được sử dụng, nếu không đề cập thêm thì kí hiệu $\|u\|$ là chuẩn 2)
  Ví dụ: Sinh viên thực hiện các lệnh sau:
  >>> import numpy as np
  >>> a = np.array([1,2,-3])
  >>> **np.linalg.norm(a)** # hoặc >>> np.linalg.norm(a, 2)
  …… ← sinh viên điền kết quả

  Chuẩn 1: $\|u\|_1 = \sum|u_i|$
  Ví dụ:
  >>> a = np.array([1,2,-3])
  >>> np.linalg.norm(a,1)
  …… ← sinh viên điền kết quả

  Về bản chất toán học, "chuẩn" bậc k của một vector được định nghĩa là:

$$\|v\|_k = \sqrt[k]{\sum_{i=1}^{n}|v_i|^n}$$

  Do đó, chúng ta có thể kiểm tra hàm do numpy cung cấp và hàm viết dưới đây như sau:

```
>>> def chuan(v,k):
        tong = 0
        for i in range(len(v)):
                tong = tong + abs(v[i]**k)
        ketqua = math.pow(tong, 1.0/k)  # <- lay can bac k cua tong
        return ketqua
```

  Sử dụng:
  >>> import math
  >>> import numpy as np
  >>> a = np.array([1,2,-3])
  >>> chuan(a, 1)
  …… ← sinh viên ghi kết quả
  >>> np.linalg.norm(a,1)
  …… ← sinh viên ghi kết quả
  >>> chuan(a, 2)
  …… ← sinh viên ghi kết quả
  >>> np.linalg.norm(a,2)

……. ← sinh viên ghi kết quả
>>> chuan(a, 3)
……. ← sinh viên ghi kết quả
>>> np.linalg.norm(a,3)
……. ← sinh viên ghi kết quả
>>> chuan(a, 1000)
……. ← sinh viên ghi kết quả
>>> np.linalg.norm(a,**1000**)
……. ← sinh viên ghi kết quả
……. ← **lỗi gì nếu lỗi xảy ra?**
……. ←

Lưu ý rằng: gói scipy.linalg cũng có hàm norm và sử dụng như numpy.linalg.norm:
>>> from scipy import linalg
>>> linalg.norm(a)
……. ← sinh viên ghi kết quả
>>> linalg.norm(a, 100)
……. ← sinh viên ghi kết quả
>>> linalg.norm(a, 1000)
……. ← sinh viên ghi kết quả

- Khoảng cách (distance) giữa hai vector u và v: $\|u - v\|$
- Góc giữa hai vector:

$$\theta = \arccos\left(\frac{< u, v >}{\|u\|.\|v\|}\right)$$

- Hai vector gọi là vuông góc với nhau khi: $< u, v > = 0$
- Phép chiếu trực giao (orthogonal projection) của u trên **không gian phát sinh bởi v** (space spanned):

$$p = \left(\frac{\langle u, v\rangle}{\langle v, v\rangle}\right) v$$

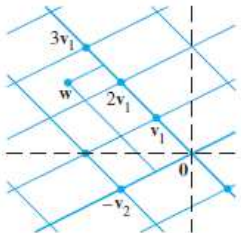
- Hai vector trực giao khi:

$$\|u \pm v\|^2 = \|u\|^2 + \|v\|^2$$

Lưu ý: trường hợp thông thường là chuẩn bậc 2 $\|u - v\|^2$, với định lý Pythagorean, chúng ta có thể thêm dấu +, nghĩa là $\|u \pm v\|^2$.

2. Đọc thêm: Tổ hợp tuyến tính

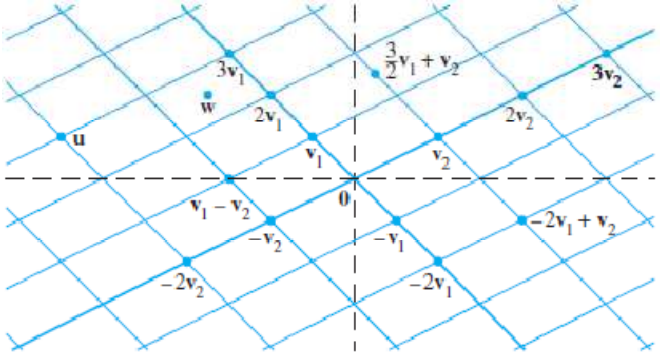
Tổ hợp tuyến tính (linear combination)



Cho p vector $v_1, v_2, \dots, v_p$ trong $\mathbb{R}^n$ và p số thực $c_1, c_2, \dots, c_p$, vector y là tổ hợp tuyến tính của tập các vector v_i khi y được định bởi:

$$y = c_1v_1 + c_2v_2 \dots + c_pv_p$$

Ví dụ: Xét vector $v_1 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$ và $v_2 = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$, tổ hợp tuyến tính của v_1 và $v_2$hình thành nên các vector khác như: $u = 3v_1 - 2v_2$, $w = \frac{5}{2}v_1 - \frac{1}{2}v_2$ như sau:



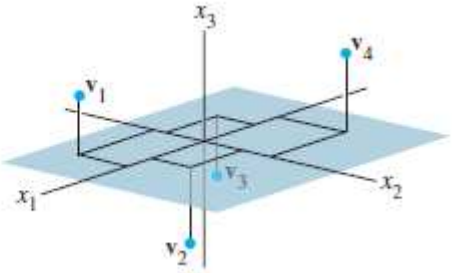
Bài toán Thiết kế ô tô hoặc chất hàng vào container/tàu thủy

Điểm trọng tâm của ô tô sẽ được sử dụng khi thiết kế các hệ thống cân bằng điện tử (giảm thiểu tình huống xe bị lật khi phanh ở tốc độ cao). Cho k vị trí $v_1, v_2, \dots, v_p$ trong không gian $\mathbb{R}^3$ tương ứng mỗi vị trí là một phụ tùng của chiếc xe ô tô đang được nghiên cứu sản xuất. Tại mỗi vị trí $v_j, j = 1..k$ các phụ tùng có trọng lượng là m_j kg. Tổng trọng lượng của xe là $m = m_1 + \cdots + m_k$. Trọng tâm của hệ được tính theo phương trình dưới đây:

$$\bar{v} = \frac{1}{m}[m_1v_1 + \cdots + m_kv_k]$$

Hãy tìm điểm trọng tâm của hệ giả định như sau:

Point	Mass
$\mathbf{v}_1 = (5, -4, 3)$	2 g
$\mathbf{v}_2 = (4, 3, -2)$	5 g
$\mathbf{v}_3 = (-4, -3, -1)$	2 g
$\mathbf{v}_4 = (-9, 8, 6)$	1 g



Giải:

```
>>> import numpy as np
>>> m = 10
>>> v1 = np.array([5,-4,3])
>>> v2 = np.array([4,3,-2])
>>> v3 = np.array([-4,-3,-1])
>>> v4 = np.array([-9,8,6])
>>> mi = np.array([2,5,2,1])
>>> M = np.array([v1,v2,v3,v4])
>>> MT = M.transpose()
>>> MT
…… ← sinh viên điền kết quả
……
……
>>> v = (1.0/m)*MT.dot(mi)
…… ← sinh viên điền kết quả
>>> print (v)
…… ← sinh viên điền kết quả
```

3. Tri riêng, vector riêng của ma trận và ứng dụng

3.1.Bài toán dẫn nhập

Mô tả bài toán:

[Mô hình Markov] Tỉ lệ thất nghiệp ở một thành phố thường thay đổi theo thời gian. Xét mô hình đơn giản sau, được gọi là mô hình Markov. Mô hình mô tả xác suất của sự chuyển dịch từ có việc làm sang thất nghiệp. Chúng ta có các giả định với thời gian được xét theo đơn vị là tuần:

- Với 1 người thất nghiệp ở tuần nào đó, người đó sẽ có xác suất p có việc làm ở tuần tiếp theo. Và như vậy, nghĩa là người đó sẽ có xác suất $1-p$ vẫn thất nghiệp ở tuần tiếp theo.
- Tương tự, với một người đang có việc làm, gọi q là xác suất người đó vẫn còn đi làm. Điều này có nghĩa là xác suất người đó thất nghiệp là $1-q$.

Từ đó, nếu gọi:

- x_t là người đang có việc làm ở tuần t.
- y_t là người đang thất nghiệp ở tuần t.

Khi đó, chúng ta có hệ phương trình:

$$\begin{cases} x_{t+1} = qx_t + py_t \\ y_{t+1} = (1-q)x_t + (1-p)y_t \end{cases}$$

Thể hiện ở dạng ma trận: $v_{t+1} = Av_t$, với:

$$A = \begin{pmatrix} q & p \\ 1-q & 1-p \end{pmatrix}, v_t = \begin{pmatrix} x_t \\ y_t \end{pmatrix}$$

Gọi A là ma trận chuyển trạng thái và v_t là vector trạng tháu của hệ thống.

Bài toán đặt ra là: Trong khoảng thời gian dài hệ thống sẽ ở trạng thái nào? Trạng thái cân bằng là gì (là tỉ lệ dự kiến về thất nghiệp của thành phố)?

Phân tích

Trạng thái sau thứ t tuần được mô tả lần lượt như sau:

- Tuần 1: $v_1 = Av_0$
- Tuần 2: $v_2 = Av_1 = A(Av_0) = A^2v_0$
- Tuần 3: $v_3 = Av_2 = A(A^2v_0) = A^3v_0$
- → Tuần thứ k: $v_t = A^tv_0$

Giả định chúng ta có số liệu sau: $p = 0.136$; $q = 0.998$. Tại thời điểm t_0, nghĩa là $t = 0$, giả định chúng ta có: $x_0 = 0.95$ và $y_0 = 0.05$. Trạng thái sau 100 tuần về tình trạng thất nghiệp/có việc làm của thành phố sẽ là:

$$\begin{pmatrix} x_{100} \\ y_{100} \end{pmatrix} = \begin{pmatrix} 0.998 & 0.136 \\ 0.002 & 0.864 \end{pmatrix}^{100} \cdot \begin{pmatrix} 0.95 \\ 0.05 \end{pmatrix} = (\dots ? \dots)$$

Sinh viên sử dụng numpy để tính toán ra giá trị trên:

>>> …….. # khai báo ma trận A

……..

>>> …….. # khai báo vector v0

……..

>>> …….. # tính vector A^100 (gợi ý: lệnh lặp)

(hoặc sử dụng lệnh matrix_power của gói linalg của numpy)

>>> …….. #

……..

>>> …….. # tính vector x100 và y100

……..

Rõ ràng việc tính toán lũy thừa ma trận là vấn đề phức tạp và cần thời gian để tính. Trong các hệ thống, nếu tính toán giá trị lũy thừa lớn hơn cho các ma trận lớn hơn thì thời gian tính toán cần nhiều. Ví dụ: tính toán lũy thừa cho ma trận về "xu hướng xếp hạng" của các trang web,… Từ đó, chúng ta có những khái niệm xử lý để giảm thiểu tính toán như sau:

Trạng thái cân bằng (Markov)

Trạng thái cân bằng là một vector trạng thái $v = \begin{pmatrix} x \\ y \end{pmatrix}, x, y \geq 0, x + y = 1$ và thỏa $Av = v$.

Với dữ liệu trên, nghĩa là $A = \begin{pmatrix} 0.998 & 0.136 \\ 0.002 & 0.864 \end{pmatrix}$, chúng ta sẽ đi tìm trạng thái cân bằng như sau:

$$\begin{pmatrix} 0.998 & 0.136 \\ 0.002 & 0.864 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} \Leftrightarrow \begin{pmatrix} 0.998 - 1 & 0.136 \\ 0.002 & 0.864 - 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

Giải ra, ta được:

$$-0.002x + 0.136y = 0 \Longrightarrow \begin{cases} x = 68y \\ y \text{ tùy ý} \end{cases}$$

Kết hợp với điều kiện $x, y \geq 0, x + y = 1$, chúng ta giải hệ:

$$\begin{cases} x - 68y = 0 \\ x + y = 1 \end{cases}$$

Sinh viên tự viết lệnh bằng với gói numpy hoặc scipy để giải hệ trên:

>>> import numpy as np # hoặc import scipy hoặc tùy sinh viên chọn

>>> …….. # thực hiện lệnh giải phương trình

……

Sinh viên thử nghiệm giải hệ trên bằng gói **sympy**:

```
>>> import sympy as sym
>>> x, y = sym.symbols('x y')
>>> xy = sym.Matrix([x,y])                    # khai báo vector: (x, y)^T
>>> A = sym.Matrix([[1, -68],[1,1]])
>>> v = sym.Matrix([0, 1])
>>> nghiem = sym.solve([A*xy-v])     # phương trình được chuyển về 1 phía
>>> print(sym.pretty(nghiem))
…… # ← sinh viên điền kết quả
```

Kết quả nghiệm là:

Gợi ý: $x = \frac{68}{69}$ *và* $y = \frac{?}{?}$

Như vậy cuối cùng, ta được tỉ lệ thất nghiệp sau 100 tuần sẽ là: $y = -- = \_\_\_\_ = \_\_\_\%$

Với hệ trên chúng ta có thể giải nhanh chóng bằng phương pháp trị riêng, vector riêng như trình bày dưới đây! [hạn chế việc tính mũ của ma trận, đặc biệt với các ma trận cực lớn]. Tuy nhiên, trước khi ứng dụng trị riêng, vector riêng, chúng ta xem những khái niệm sau:

Ma trận đường chéo là ma trận có các giá trị ngoài đường chéo bằng 0:

$$D = \begin{pmatrix} d_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & d_n \end{pmatrix}$$

Ma trận đường chéo có tính chất nếu mũ nó lên k lần thì chỉ có các giá trị ở đường chéo mũ lên:

$$D^k = \begin{pmatrix} d_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & d_n \end{pmatrix}^k = \begin{pmatrix} d_1^k & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & d_n^k \end{pmatrix}, k \in \mathbb{Z}$$

Với ma trận đường chéo trên, chúng ta dễ dàng tính toán lũy thừa. Tuy hầu hết các ma trận đều không phải ma trận đường chéo nhưng đôi khi các ma trận được chéo hóa, nghĩa là tồn tại ma trận đường chéo D và ma trận khả nghịch P để ma trận A được biểu diễn thành:

$$A = PDP^{-1}$$

3.2.Trị riêng và vector riêng của ma trận

Cho ma trận A có kích thước $n \times n$

Nếu một số $\lambda \in \mathbb{R}$ và một vector (có độ dài n) $x \neq 0$ thỏa: $Ax = \lambda x$ thì ta gọi λ là giá trị riêng (eigenvalue) của ma trận A, và x được gọi là vector riêng của A ứng với giá trị λ và $E_\lambda(A)$ là kí hiệu tập các vector riêng tương ứng với một giá trị riêng λ của ma trận A.

Lưu ý rằng: có thể có nhiều vector riêng tương ứng với một giá trị riêng.

Ví dụ:

Ma trận: $A = \begin{pmatrix} 1 & 6 \\ 5 & 2 \end{pmatrix}$ có vector riêng là $u = \begin{pmatrix} 6 \\ -5 \end{pmatrix}$ tương ứng trị riêng $\lambda = -4$

Sinh viên kiểm lại phát biểu trên:

>>> …… # khai báo A

……

>>> …… # khai báo u

……

>>> …… # Chứng minh Au = -4u

……

- Quy trình tính toán trị riêng của ma trận: Giải hệ $Ax = \lambda x$ hoặc hệ tương ứng

$$(A - \lambda I)x = 0$$

Nghĩa là giải hệ tính định thức:

$$det(A - \lambda I) = 0$$

Sinh viên thực hiện lệnh theo 2 cách: lệnh solve để giải phương trình và lệnh det (của sympy) để tính định thức.

Ví dụ: Tính trị riêng bằng gói thư viện sympy cho ma trận $A = \begin{pmatrix} 2 & 3 \\ 3 & -6 \end{pmatrix}$

>>> import sympy

>>> x, y, Lambda = **sympy.symbols**('x y Lambda')

>>> I = **sympy**.eye(2)

```
>>> A = sympy.Matrix([[2,3],[3,-6]])
>>> phuongtrinh = sympy.Eq(sympy.det(Lambda*I-A), 0)
>>> nghiem = sympy.solve(phuongtrinh)
>>> print([sympy.N(phantu,4) for phantu in nghiem])
…… # ← sinh viên điền kết quả
>>> print (sympy.pretty(nghiem))
…… # ← sinh viên điền kết quả
```

- Quy trình tính toán vector riêng của ma trận:

Với mỗi giá trị của trị riêng, chúng ta tính toán tập các vector riêng bằng việc giải phương trình:

$$(A - \lambda I)x = 0$$

Ví dụ: với $\lambda = 3$ là một trị riêng vừa tìm được bên trên của ma trận $A = \begin{pmatrix} 2 & 3 \\ 3 & -6 \end{pmatrix}$, chúng ta tính toán được các vector riêng là phương trình $(A - 3I)x = 0$.

Sinh viên viết các lệnh để giải:

……

……

……

……

Gợi ý kết quả: với $\lambda = 3$, tập các vector riêng là: $E_3(A) = s\begin{pmatrix} 3 \\ 1 \end{pmatrix}, s \in \mathbb{R}$.

Lưu ý: khi giải $A - \lambda I$ một số trường hợp chúng ta biểu diễn kết quả vector riêng là một không gian vector được xây dựng từ tổ hợp tuyến tính của các vector độc lập tuyến tính.

Ví dụ:

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} -2x_2 + 4x_4 \\ tự\ do \\ -3x_4 \\ tự\ do \end{pmatrix} = \begin{pmatrix} -2s + 4t \\ s \\ -3t \\ t \end{pmatrix} = s\begin{pmatrix} -2 \\ 1 \\ 0 \\ 0 \end{pmatrix} + t\begin{pmatrix} 4 \\ 0 \\ -3 \\ 1 \end{pmatrix} = sv_1 + tv_2$$

3.3.Ứng dụng của trị riêng và vector riêng

Trị riêng và vector riêng được ứng dụng vào nhiều lĩnh vực tính toán ma trận và các đối tượng khác. Ví dụ: ứng dụng trong phương pháp kiểm tra nói thật/dối, AHP… Với ma trận, ứng dụng dễ thấy nhất là việc sử dụng trị riêng và vector riêng để ***tính toán lũy thừa (mũ) cho ma trận***.

Cụ thể, với ma trận chéo hóa được (diagonalizable), nghĩa là: ma trận A biểu diễn được ở dạng phép nhân của 3 ma trận P, P^{-1} và D như sau: $A = PDP^{-1}$. Nghĩa là, biểu thức trên sẽ tương ứng:

$$A = PDP^{-1} \Leftrightarrow D = P^{-1}AP \Leftrightarrow \boldsymbol{AP = PD}$$

Trong phương trình cuối cùng, chúng ta có thể chọn D gồm các giá trị riêng của A và cụ thể trên đường chéo) và P sẽ bao gồm các vector riêng của A và thể hiện bằng các cột.

Điều kiện chéo hóa **(để chứng minh điều bên trên)**

Cho ma trận $A(n \times n)$ có k trị riêng khác nhau: $\lambda_1, \lambda_2, \dots, \lambda_k$ và m_i là bậc tự do của các các hệ tuyến tính: $(A - \lambda_i I)x = 0, i = 1,2,..,k$

Định lý: Ma trận A chéo hóa được khi và chỉ khi $m_1 + m_2 + \cdots + m_k = n$. Trong trường hợp này, chúng ta có thể chọn ma trận đường chéo D và P như sau:

- Mỗi giá trị trên đường chéo D là giá trị riêng λ_i, với các giá trị λ_i được lặp lại m_i lần.
- P là ma trận gồm các vector riêng được dựng thành cột. Với mỗi giá trị riêng λ_i sẽ có m_i cột tương ứng.

Với cách chọn như vậy, (người ta) dễ dàng chứng minh được rằng điều kiện để P khả nghịch là khi và chỉ khi có n vector riêng độc lập tuyến tính.

Tóm lại, một ma trận chéo hóa được:

- Khi và chỉ khi có n vector riêng độc lập tuyến tính.
- Hoặc A có n trị riêng khác nhau từng đôi một.
- Hoặc A có dạng đối xứng (symetric)

Ví dụ:

Xét ma trận $A = \begin{pmatrix} 2 & 3 \\ 3 & -6 \end{pmatrix}$ như trên.Sau khi tính toán, ta thấy 2 giá trị riêng là $\lambda_1 = -7, \lambda_2 = 3$
[Nhắc lại, trị riêng là giải hệ phương trình $(A - \lambda I)x = 0$, lệnh trong scipy là: linalg.eigvals(A)].
Và mỗi λ_i đều là nghiệm đơn $m_i = 1$ nên $m_1 + m_2 = 1 + 1 = 2 = n$ (n là kích thước ma trận).

Chúng ta xây dựng ma trận chéo hóa như sau:

$$D = \begin{pmatrix} -7 & 0 \\ 0 & 3 \end{pmatrix} \text{ và } P \text{ sẽ có dạng như sau: } P = \begin{pmatrix} x(E_{-7}) & x(E_3) \\ y(E_{-7}) & y(E_3) \end{pmatrix}$$

Ta có: $E_{-7}: x = s\begin{pmatrix}-1/3\\1\end{pmatrix}; E_3 = s\begin{pmatrix}3\\1\end{pmatrix} \Longrightarrow P = \begin{pmatrix}-\frac{1}{3} & 3\\1 & 1\end{pmatrix}$ (dựng đứng vector riêng)

Sinh viên kiểm lại bằng lệnh Python và tính P^{-1}:

>>> import numpy as np

>>> A = np.array([[2,3],[3,-6]])

>>> D = np.array([[-7,0],[0,3]])

>>> P = np.array([[-1.0/3, 3],[1,1]])

>>> from numpy import linalg

>>> from numpy import linalg as LA

>>> P1 = LA.inv(P) # tính giá trị nghịch đảo của P

>>> print (P1)

…… # in ra P^(-1)

…… Sau đó thực hiện kiểm lại bằng 2 lệnh bên dưới:

>>> A.dot(P)

……

……

>>> P.dot(D)

……

……

Từ đó, chúng ta có thể tính toán giá trị của A^{1000} như sau:

$$A^{1000} = (PDP^{-1})^{1000} = (PDP^{-1})(PDP^{-1}) \dots (PDP^{-1}) = PD^{1000}P^{-1}$$

>>> P @ (D ** 1000) @ P1

……..

Lưu ý: thử tính các giá trị lũy thừa của ma trận đường chéo:

>>> print (D **2)

……..

……..

BÀI TẬP CHƯƠNG 6

Câu 1: [*Trắc nghiệm*] Trong Python 3, các câu lệnh sau đây, câu lệnh nào tính chuẩn 2 của vector a. Với a được xác định là:

```
>>> import numpy as np

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

a. `>>> mag = np.sqrt(a.dot(a))`
b. `>>> mag = np.sqrt(a @ a )`
c. `>>> mag = np.sqrt(np.inner(a,a))`
d. `>>> mag = lambda x : math.sqrt(sum(i** 2 for i in x))`
   `>>> mag(a)`

Câu 2: Sinh viên hãy viết các chương trình tính toán theo các công thức:

a. Tính toán khoảng cách giữa 2 vector u và v:

$$\|u - v\|$$

b. Tính góc θ của 2 vector u và v nhập vào:

$$\theta = \arccos\left(\frac{< u, v >}{\|u\|.\|v\|}\right)$$

c. Phép chiếu trực giao p với 2 vector u và v cho trước:

$$p = \left(\frac{\langle u, v\rangle}{\langle v, v\rangle}\right) v$$