Some Vietnamese seminar lectures on Linear algebra (with Python codes) for Student aiming to Data Science.

Book · June 2019

1 author:

Canh Khuu Minh
Ho Chi Minh City University of Technology (HCMUT)

THỰC HÀNH ĐẠI SỐ TUYẾN TÍNH

TÀI LIỆU PHỤC VỤ SINH VIÊN NGÀNH KHOA HỌC DỮ LIỆU

Nhóm biên soạn: TS. Hoàng Lê Minh – Khưu Minh Cảnh – Huỳnh Thái Học

TP.HCM - Năm 2019

MỤC LỤC

BÀI 1: GIỚI THIỆU PHẦN MỀM PYTHON VÀ NUMPY

Mục tiêu:

- *Nắm vững được Python để viết các đoạn lệnh.*
- *Nâng cao kỹ năng lập trình với tinh thần/cách nghĩ một cách đại số và vẻ đẹp của ngôn ngữ.*
- *Sử dụng được gói Numpy, SciPy để thực hiện các tác vụ đơn giản xử lý đại số.*

Nội dung chính:

1. Môi trường và một số lưu ý về lập trình Python với đại số tuyến tính

Với nền tảng kiến thức cơ bản về lập trình cơ bản Python, sinh viên cần thực hành thêm các lệnh trong ngôn ngữ Python dưới đây để có những kỹ năng xử lý cho các bài toán đại số:

1.1. Gói phần mềm Python tích hợp

Trong nội dung thực hành này, gói phần mềm Anaconda sẽ được sử dụng để minh họa về các thao tác tính toán về đại số tuyến tính với các câu lệnh Python. Anaconda là tập hợp các gói thư viện thực thi trên nền tảng Python.

Sinh viên có thể tự cài đặt gói Anaconda vào máy tính cá nhân để nghiên cứu, học tập và thực hành cũng như làm bài tập được giao từ trang web: https://www.anaconda.com/distribution phiên bản 3.x (1/2019 là 3.7).

Lưu ý 1: Khi cài đặt, nếu trong máy tính đã có các hệ thống sử dụng Python thì các biến môi trường không cần phải thay đổi để không ảnh hưởng đến các chương trình khác. Vị trí cài đặt có thể chọn là: C:\Anaconda hoặc D:\Anaconda để việc sử dụng dễ dàng.

Lưu ý 2: Hiện tại, ngoài Anaconda, nhiều gói phần mềm Python tương tự. Để cài đặt và sử dụng, yêu cầu cần đọc rõ các gói hỗ trợ, đặc biệt là các thư viện: numpy, scipy, networkx, sympy. Do đó, với hệ sinh thái phong phú của Python, sinh viên có thể sử dụng thư viện khác trong nghiên cứu. Tuy nhiên, trong lớp học, giảng viên chỉ giới thiệu các thư viện nền tảng và thống nhất sử dụng.

1.2. Môi trường thử nghiệm và lập trình

Mặc dù có nhiều môi trường phát triển phần mềm, trong khuôn khổ thực hành này, sinh viên được khuyến nghị sử dụng IDE có tên là IDLE nằm trong gói phần mềm Anaconda. Tập tin thực thi của IDLE là idle.exe có thể tìm thấy trong thư mục Anaconda sau khi cài đặt.

IDLE cung cấp một cửa sổ dòng lệnh đơn giản nhưng hiệu quả và dễ dàng thực thi câu lệnh.

Sau khi khởi động idle, ứng dụng sẽ có tên là **Python 3.x.y Shell** (ví dụ: 3.6.3) với dấu nhắc >>> được thể hiện trên màn hình để thực hiện các câu lệnh.

1.3. Các phép xử lý với danh sách

- Lưu ý về các phép toán xử lý trên list

Danh sách (**list**) là đối tượng thường được sử dụng trong Python. Như tên gọi, mục tiêu chính của cấu trúc dữ liệu danh sách chính là lưu trữ những đối tượng. Do đó, các phép tương tác trên cấu trúc dữ liệu **list** hướng đến xử lý về lưu trữ dữ liệu hơn là tính toán. Cụ thể là: phép toán + hai danh sách mang ý nghĩa ghép nối 2 danh sách. Ví dụ:

>>> danhsach1 = [1. , 3.]

>>> danhsach2 = [5. , 7.]

>>> danhsach = danhsach1 + danhsach2

.... ← Sinh viên điền kết quả.

>>> danhsach_gapdoi = 2 * danhsach

.... ← Sinh viên điền kết quả.

>>> danhsach * 2

.... ← Sinh viên điền kết quả.

>>> danhsach / 2

.... ← Sinh viên điền kết quả.

- Ghép các danh sách bằng lệnh zip:

Giả sử có 1 bạn học 4 môn với thứ tự các môn và điểm số như 3 danh sách bên dưới, chúng ta cần ghép cặp từng môn học

>>> mon_hoc = ["ToanCC", "DSTT", "ToanRR", "LaptrinhCB"]

>>> thu_tu = [2, 3, 4, 1]

>>> diem_so = [10, 9, 8, 7]

>>> anh_xa = **zip**(thu_tu, mon_hoc, diem_so)

>>> anh_xa

.... ← Sinh viên điền kết quả.

>>> tap_hop = set(anh_xa)   # ←**chuyển thành dạng tập hợp**

>>> tap_hop

.... ← Sinh viên điền kết quả.

Sau đó, từ dữ liệu được ghép cặp bằng zip, chúng ta có thể phân rã bằng phương pháp sau:

>>> lay_TT, lay_monhoc , lay_diem = zip(*anh_xa)

>>> lay_monhoc

.... ← Sinh viên điền kết quả.

- Xây dựng danh sách: Bên cạnh những cách khai báo danh sách thông thường như: khai báo sẵn hoặc khai báo list rỗng rồi bổ sung phần tử bằng lệnh append, cơ chế sinh tập hợp list trong Python còn có thể thực hiện bằng các cách sau:

```
>>> import itertools
>>> tap_sinh = list(itertools.chain(range(4), range(5,10), range(15,20) ))
>>> tap_sinh
```

.... ← Sinh viên điền kết quả.

Ngoài ra, với phép toán zip bên trên, chúng ta có thể tạo ra các tập là tập sinh từ 2 hoặc nhiều tập.

Ví dụ: Cần tạo 1 bộ 3 thành phần gồm: danh sách từ 0 đến 3; danh sách từ 7 đến 11 và ngược lại một danh sách từ 10. Lệnh Python như sau:

>>> **list**(**zip**(range(4), range(7, 12), *reversed*(range(11))))

.... ← Sinh viên điền kết quả.

1.4. Lệnh thực thi một tập tin python

Trong môi trường dòng lệnh Python, một tập tin Python (*.py) được thực thi thông qua hàm **execfile**. Sinh viên thực hiện minh họa dưới đây:

Bước 1: Tạo một tập tin thucthi1.py và đặt tại một thư mục (trong hình minh họa là đặt tại 'd:/')

Python 2.7.8: thucthi1.py - D:/thucthi1.py

```
mylist = [100, 200, 300]
a,b,c = mylist
print a, b, c
```

Bước 2: Thực thi tập tin trên. Sinh viên thử nghiệm các lệnh sau và ghi kết quả đạt được:

>>> a = b = c = 0

>>> mylist = []

>>> **execfile**('d:/thucthi1.py') # hoặc câu lệnh sau >>> **execfile**('d:\\thucthi1.py')

.... ← Sinh viên điền kết quả.

Ngoài ra, khi đã có một tập tin Python (.py) cùng thư mục, chúng ta có thể dễ dàng truy xuất đến hàm (def) của nó thông qua lệnh **from <tên tập tin> import <tên hàm xử lý>**. Ví dụ sau:

- Giả sử bổ sung thêm vào đường dẫn D:\ tập tin có tên là trieugoi.py (*lưu ý: đường dẫn 'D:\' là đường dẫn minh họa, tùy thuộc vào máy tính sinh viên đang thực tập để chọn đường dẫn thích hợp*).

Python 2.7.8: trieugoi.py - D:/trieugoi.py

```
def tao_danhsach():
    danhsach = [100, 200, 300]
    return danhsach
```

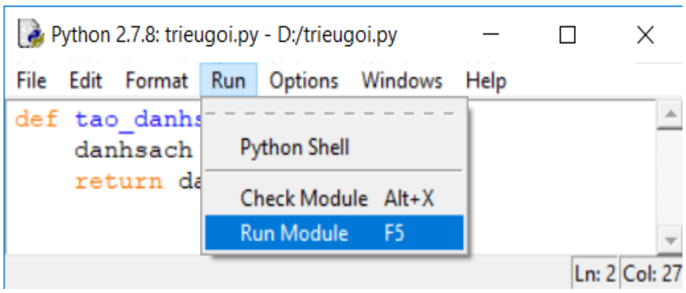
- Điều chỉnh tập tin thucthi1.py như sau:

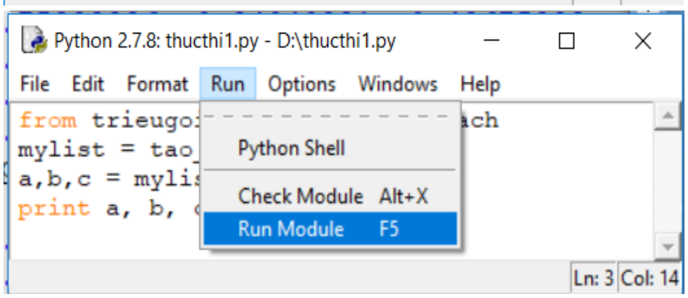
Python 2.7.8: thucthi1.py - D:\thucthi1.py

```
from trieugoi import tao_danhsach
mylist = tao_danhsach()
a,b,c = mylist
print a, b, c
```

- Sau đó thử nghiệm là đoạn script trên:
  >>> a = b = c = 0
  >>> mylist = []
  >>> execfile('d:/thucthi1.py')

Lưu ý: Trong một số IDE (như IDLE được tích hợp với gói Anaconda), để "biên dịch" vào môi trường chúng ta phải thực hiện việc Run Module để "tải" thư viện vào IDE (trước khi sử dụng) hoặc thực thi với IDE tập tin thucthi1.py:





1.5. Khối lệnh bắt đầu với with

Từ khóa with được sử dụng để bắt đầu một khối đoạn lệnh với ý nghĩa là định nghĩa ngữ cảnh thực thi xác định.

Ví dụ: Để xây dựng một hàm đọc số dòng của một tập tin, chúng ta có thể viết như sau:

```
>>> def dem_dong(ten_taptin):
        with open(ten_taptin,'r') as taptin:
            return len(taptin.readlines())
```

Về ý nghĩa, từ khóa with mở ra một khối lệnh xử lý với đối tượng được trả về sau một lệnh. Trong ví dụ trên, đó là đối tượng *taptin* dạng tập tin (file) được tạo thành từ lệnh **open** và xử lý trong khối lệnh đó. Với phong cách viết code như nêu một mệnh đề toán học và bên dưới là các triển khai cụ thể, cho thấy sự sáng sủa trong phong cách viết code của Python để người sử dụng dễ dàng định hướng khi xây dựng chương trình và trong kiểm lỗi.

Lưu ý: Từ phiên bản 2.6 trở đi, lệnh **with** được sử dụng trực tiếp. Trước đó, trong phiên bản 2.5, để sử dụng lệnh with, chúng ta phải khai báo thư viện: **from __future__ import with_statement**.

2. Các gói thư viện cơ bản tích hợp trong Python hỗ trợ toán học

Dưới đây là các gói thư viện cơ bản của Python sinh viên cần được trang bị để hỗ trợ tính toán đại số:

- **Gói math**: Python hỗ trợ tính toán các phép toán số học cơ bản với thư viện math. Để sử dụng thư viện, chúng ta đơn giản import vào như sau: >> **import math** và sử dụng.

Bảng minh họa một số hàm, hằng và các xử lý toán học cơ bản trong thư viện **math** của python:

TT	Nhóm	Tên hàm	Ý nghĩa	Lệnh minh họa
1	Hàm lấy giá trị	floor	Giá trị nguyên lớn nhất không vượt số thực (số sàn)	>>> x = 5.4 >>> math.floor(x)
2		ceil	Giá trị nguyên nhỏ nhất lớn hơn số thực (số trần)	>>> x = 5.4 >>> math.ceil(x)
3		fabs	Lấy giá trị tuyệt đối của	>>> math.fabs(-x)
4	Mũ và lũy thừa	exp	Hàm lấy mũ e^x	>>> x = 1 >>> math.exp(x)
5		expm1	Hàm lấy mũ $e^x - 1$	>>> x = 1e-4 >>> math.expm1(x)
6		pow(x,y)	Tính giá trị e^x	>>> math.exp(2,3)
7		log	Hàm lấy logarith theo cơ số, mặc định là cơ số e.	>>> math.log(3) >>> math.log(8, 2)
8		log2	Lấy log cơ số 2 (từ phiên bản 3.5)	>>> math.log2(1024)
9		log10	Lấy log cơ số 10	>>> math.log10(1000)
10	Lượng giác	pi (viết thường)	Lấy giá trị số Pi.	>>> radius = 5 >>> S = math.pi *(radius ** 2)
11		radians(x)	Chuyển độ sang giá trị radian	>>> goc = 60 >>> radi = math.radians(goc)
12		degrees(x)	Chuyển giá trị radian sang độ	>>> math.degrees(1.0) >>> math.degrees(math.pi)
13		sin(rad)	Tính sin của góc radian	>>> math.sin(radi)
14		cos(rad)	Tính cos của góc radian	>>> math.cos(radi)
15		tan(rad)	Tính tan của góc radian	>>> math.tan(radi)
16		asin(x)	Lấy giá trị arcsin(x)	>>> math.degrees(math.asin(0.5))
17		acos(x)	Lấy giá trị arccos(x)	>>> math.degrees(math.acos(0.5))
18		atan(x)	Lấy giá trị arctan(x)	>>> math.atan(x)
19	Tính toán số học	sqrt(x)	Hàm lấy căn số thực	>>> math.sqrt(16) >>> math.sqrt(-9)
20		factorial(k)	Tính giai thừa của một số nguyên. Lỗi phát sinh khi nhập số thực	>>> math.factorial(5) # = 120
21		gcd(a, b)	[Phiên bản 3.5 trở lên] Ước số chung lớn nhất của hai số nguyên	>>> math.gcd(18, 27) # = 9
22		fsum(dãy)	Tính tổng của một list/…	>>> a = [1,2, 3, 4,5] >>> math.fsum(a)

23		hypot(x,y)	Tính $\sqrt{x^2+y^2}$ với x,y là các số thực.	>>> math.hypot(3,4)
24	Hằng số	inf	Số vô cùng	>>> math.inf
25		nan	Không phải giá trị số. Để kiểm một biến có phải là số thực hay không	>>> math.nan

Sinh viên thực hành các lệnh:

>>> x = 1

>>> math.expm1(x) == math.exp(x) - 1

.... ← Sinh viên điền kết quả.

>>> x = 1e-5 # lưu ý: viết chữ e dính liền dấu – và số 1.

>>> math.expm1(x) == math.exp(x)- 1

.... ← Sinh viên điền kết quả (True hay False?).

>>> math.**exp(x) – 1**

.... ← Sinh viên điền kết quả.

>>> math.**expm1(x)**

.... ← Sinh viên điền kết quả.

Như vậy, với các số có giá trị nhỏ, nếu sử dụng phương pháp tính **exp(x)-1** sẽ gây sai số!

Sinh viên có thể tham khảo và tiếp cận nhiều hàm khác tại: https://docs.python.org/3/library/math.html. Ngoài ra, một số gói thư viện sẽ được giới thiệu và thực hành kết hợp trong các bài tiếp theo như:

- **Gói random**: liên quan đến các vấn đề ngẫu nhiên.
- **Gói itertools**: liên quan đến các bài toán tập hợp như tổ hợp/chỉnh hợp.
- **Gói datetime**: liên quan đến xử lý thời gian, đặc biệt với dữ liệu có ngày giờ.
- **Gói numbers**: hỗ trợ xử lý các dạng số chung.
- **Gói cmath**: hỗ trợ tính toán số phức.
- **Gói decimal**: hỗ trợ tính toán và xử lý số thực.
- **Gói statistics**: hỗ trợ các công cụ tính các chỉ số thống kê cơ bản như: trung bình, mode, trung phương (median)… của một tập số.

3. Làm quen với thư viện numpy

numpy là một nền tảng tính toán của Python. Tài liệu của numpy có thể tìm thấy dễ dàng trên mạng, như: https://docs.scipy.org/doc/numpy/numpy-ref-1.16.1.pdf (khoảng 1372 trang tính đến 01/2019). Chúng ta chỉ quan tâm đến một số lệnh của **numpy** liên quan đến các bài toán đại số tuyến tính. Các lệnh còn lại sinh viên có thể chủ động tự tìm hiểu và nghiên cứu để sử dụng.

Để sử dụng numpy, trước tiên, chúng ta phải khai báo thư viện:

>>> **import numpy as np**

Giải pháp khai báo như thế sẽ được sử dụng trong các bài thực hành bên dưới. Khi đó, để sử dụng thư viện numpy, chúng ta phải sử dụng tiền tố **np.**

Lưu ý: Cách khác hơn (không giới thiệu trong bài thực hành này), chúng ta có thể khai báo như sau:

>>> **from** numpy **import** *

Với cách khai báo này, chúng ta có thể sử dụng trực tiếp các hàm của numpy mà không cần phải bổ sung tên thư viện numpy (**np.**) ở phía trước các câu lệnh. Tuy nhiên, nhược điểm của phương pháp là khó theo dõi các hàm thuộc numpy hoặc của thư viện khác với người mới/đang tiếp cận lập trình numpy.

3.1. Một số lệnh cơ bản numpy xử lý vector

Khác với kiểu danh sách ở Python chuẩn, numpy hỗ trợ việc định nghĩa một vector theo nghĩa đại số:

>>> vec1 = np.**array**([1., 3., 5.])

>>> vec1 * 2

.... ← Sinh viên điền kết quả.

>>> vec1 * vec1

.... ← Sinh viên điền kết quả và cho biết đó phép nhân gì?

....

>>> vec1 /2

.... ← Sinh viên điền kết quả.

>>> vec1 + vec1

.... ← Sinh viên điền kết quả.

>>> vec2 = array([2., 4.])

>>> vec1 + vec2

.... ← Sinh viên điền kết quả và lí do.

....

>>> vec3 = array([2., 4., 6.])

>>> vec1 + vec3

.... ← Sinh viên điền kết quả.

>>> vec1 / vec3

.... ← Sinh viên điền kết quả.

>>> vec1 * vec3

.... ← Sinh viên điền kết quả.

>>> 2* vec1 + 5* vec3

.... ← Sinh viên điền kết quả.

Lưu ý:

Trong numpy, kiểu dữ liệu vector có tên là **array** là kiểu dữ liệu có kích thước không thay đổi và chỉ 1 kiểu dữ liệu cho toàn bộ vector. Tuy vậy, một số nét tương đồng giữa kiểu dữ liệu list trong python và vector (array) trong Python là:

- Truy xuất đến phần tử của vector:
  >>> vec3[2]

  .... ← Sinh viên điền kết quả.

- Tạo vector thông qua câu lệnh linspace(phần tử đầu, phần tử cuối, số lượng phần tử):
  >>> vec4 = np.**linspace**(0, 20, 5)
  >>> vec4

  .... ← Sinh viên điền kết quả.

Ngoài ra, cách thức để tạo nhanh các vector như sau:

- Tạo các vector toàn 0:
  >>> vec5 = np.**zeros**(5)

.... ← Sinh viên điền kết quả.

- Tạo các vector toàn 1:
  >>> vec6 = np.**ones**(5)

  .... ← Sinh viên điền kết quả.

- Tạo vector ảo (giá trị rỗng):
  >>> vec7 = np.**empty**(5)

  .... ← Sinh viên điền kết quả.

- Tạo vector các giá trị ngẫu nhiên từ 0 đến 1:
  >>> np.**rand**(5)   # hoặc **np.random.random(5)** nếu 1 trong 2 lệnh bị lỗi

  .... ← Sinh viên điền kết quả.

Các lệnh xử lý:

Các lệnh dưới đây sử dụng lệnh khai báo vector v như sau:

>>> v = np.array([1., 3., 5.])

Sinh viên thực tập các xử lý trên đối tượng vector của numpy:

- Lệnh lấy tổng các thành phần của vector:
  >>> np.**sum**(v)

  .... ← Sinh viên điền kết quả.

- Lệnh xem số chiều của một vector:
  >>> v.**shape**

  .... ← Sinh viên điền kết quả.

- Thử nghiệm "chuyển vị" vector:
  >>> v.**transpose**()

  .... ← Sinh viên điền kết quả.

- Lệnh lấy một phần của vector:
  >>> v1 = v[:2]
  >>> v1

  .... ← Sinh viên điền kết quả.

  >>> v[0] = 5

>>> v

.... ← Sinh viên điền kết quả.

>>> v1

.... ← Sinh viên điền kết quả.

>>> v1[:2] = [1., 2., 3.]

.... ← Sinh viên điền kết quả.

....

>>> v1[:2] = [1., 2]
>>> v

.... ← Sinh viên điền kết quả.

Tuy nhiên, lưu ý với các phép gán có sử dụng các toán tử:
>>> v3 = 2 * v[:2] + v1 * 3
>>> v3

.... ← Sinh viên điền kết quả.

>>> v1 = [4, 6]
>>> v3

.... ← Sinh viên điền kết quả.

>>> v

.... ← Sinh viên điền kết quả.

- Các phép toán trên vector: trên vector, chúng ta có thể thực hiện phép toán như: lấy sin/cos: *Lưu ý rằng, các toán tử lấy căn bậc 2 (sqrt), sin, cos cũng phải là các toán tử của numpy*
  >>> v + 10.0
  ……………………………………………………… ← Sinh viên điền kết quả.
  >>> np.**sqrt**(v)
  ……………………………………………………… ← Sinh viên điền kết quả.
  >>> np.**cos**(v)
  ……………………………………………………… ← Sinh viên điền kết quả.
  >>> np.**sin**(v)
  ……………………………………………………… ← Sinh viên điền kết quả.

- Tích vô hướng của hai vector (kết quả là một số):

$$\vec{v}_1 = (a, b), \vec{v}_2 = (c, d): \vec{v}_1\vec{v}_2 = ac + bd$$

>>> np.dot(v1, v3)

………………………………………………………… ← Sinh viên điền kết quả.

>>> v1.dot(v3)

………………………………………………………… ← Sinh viên điền kết quả.

>>> v3.dot(v1)

………………………………………………………… ← Sinh viên điền kết quả.

- Độ dài (chuẩn 2) và chuẩn 1 của vector (độ dài theo Euclide):

$$\vec{v}_1 = (a, b): |\vec{v}_1| = \sqrt{a^2 + b^2};\ \|\vec{v}_1\| = |a| + |b|$$

>>> np.**linalg**.norm(v)

………………………………………………………… ← Sinh viên điền kết quả.

Lưu ý: gói thư viện **numpy.linalg** là gói thư viện chuyên xử lý các bài toán của đại số tuyến tính. Gói thư viện này chúng ta sẽ làm quen trong các chương tiếp theo.

- Độ dài (kích thước) của vector:

>>> len(v)

………………………………………………………… ← Sinh viên điền kết quả.

- Kết nối các vector

*Lưu ý: Vector trong numpy không có khái niệm ghép nối (append) như list của python. Tuy nhiên, 2 lệnh thay thế là **hstack** và **vstack** sẽ kết nối vector theo chiều dọc và chiều ngang tương ứng. Sinh viên thực hành các lệnh dưới đây:*

>>> np.**hstack**([v, v3])

………………………………………………………… ← Sinh viên điền kết quả.

Bài tập thực hành:

Minh họa về xử lý vector như sau: Giả định có vector có chiều dài $2n$, n là số tự nhiên dương. Chúng ta muốn chuyển đoạn từ $n + 1$ đến $2n$ về đầu vector đồng thời chuyển dấu các giá trị đó, nghĩa là:

$$(x_0, x_1, .., x_{n-1}, \boldsymbol{x_n}, \boldsymbol{x_{n+1}}, \dots, \boldsymbol{x_{2n-1}}) \rightarrow (-\boldsymbol{x_n}, \boldsymbol{x_{n+1}}, \dots, -\boldsymbol{x_{2n-1}}, x_0, x_1, \dots, x_{n-1})$$

Đoạn mã xử lý:

```
>>> def symp(v):
        n = len(v) // 2
        return np.hstack([-v[-n:], v[:n]])

>>> v6 = np.array([1., 3., 5., 7., 9., 11.])
>>> v7 = symp(v6)
>>> v7
```

………………………………………………………… ← Sinh viên điền kết quả.

3.2. Thể hiện ma trận bằng numpy

Ma trận được thể hiện trong numpy là nhiều dãy [] trong kiểu khai báo **array**.

Ví dụ: ma trận $M_1 = \begin{pmatrix} 9 & 12 \\ 23 & 30 \end{pmatrix}$ được khai báo như sau:

>>> **M1 = np.array([[9, 12], [23,30]])**

………………………………………………………… ← Sinh viên điền kết quả thể hiện trên màn hình

…………………………………………….....................

Khi đó, giả sử chúng ta có một vector $u = (2,1)$ thì *tích* giữa ma trận và vector M_1u sẽ là:

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

>>> **tichM1u = M1.dot(u)**

>>> print(tichM1u)

………………………………………………………… ← Sinh viên điền kết quả và giải thích.

Lưu ý: tích này sẽ khác kết quả với tích dưới đây:

Tích ma trận:

>>> tichuM1 = u.**dot**(M1)

>>> print(tichuM1)

………………………………………………………… ← Sinh viên điền kết quả và giải thích.

Sinh viên thực hành và hãy cho biết kêt quả hai phép toán dưới đây:

>>> **np**.**dot**(M1, u)

………………………………………………………… ← Sinh viên điền kết quả và giải thích.

>>> **np**.**dot**(u, M1)

………………………………………………………… ← Sinh viên điền kết quả và giải thích.

Bài tập trên lớp: Sinh viên thực hành và cho biết kết quả các lệnh sau:

Lệnh: >>> **mat1 = np.zeros([5,5])**

Câu hỏi: Cho biết mat1**?**

Lệnh: >>> **mat2 = np.ones([5,5])**

Câu hỏi: Cho biết mat2**?**

Lệnh: >>> **mat3 = mat1 + 2* mat2**

Câu hỏi: Cho biết mat3**?**

Lệnh: >>> **mat4 = mat3**

Lệnh: >>> **mat3[3][2] = 10**

Câu hỏi: Cho biết mat3 và mat4? (mat3 thay đổi thì mat4 có thay đổi theo không?)

Lệnh: >>> **mat5 = np.copy(mat3)**

Lệnh: >>> **mat3[3][2] = 10**

Câu hỏi: Cho biết mat3, mat4 và mat5? (mat3 thay đổi thì mat4 và mat5 có thay đổi theo không?)

Lệnh: >>> **mat6 = np.empty([4, 5])**

Câu hỏi: hãy cho biết mat6?

Lệnh: >>> **mat7 = np.identity(4)**

Câu hỏi: hãy cho biết mat7?

Lệnh: >>> **mat8 = np.rand([4,5]) # hoặc lệnh: mat8 = np.random.random([4,5]) nếu một trong hai lệnh báo lỗi!**

Câu hỏi: hãy cho biết mat8?

4. Giới thiệu thư viện SciPy

Scipy là thư viện xử lý trên nền tảng của numpy. Scipy có nhiều gói xử lý khác nhau để phân tích dữ liệu, như: tích phân, tối ưu, nội suy, biến đổi Fourier, xử lý tín hiệu, xuất nhập tập tin, thống kê, xử lý ảnh nhiều chiều,... Trong đó, nhóm xử lý đại số (như gói **scipy.linalg** và gói **scipy.sparse**) hỗ trợ người sử dụng xử lý tính toán về đại số. Cụ thể:

- Gói **scipy.linalg**: xử lý các thuật toán đại số.
- Gói **scipy.sparse**: xử lý các bài toán về giá trị riêng thưa (sparse eigenvalue).

Để sử dụng SciPy, chúng ta phải import thư viện vào:

>>> **import scipy as sp**

Sau đó, những hàm thuộc thư viện sẽ được triệu gọi bằng cách: **sp.**

Trong các bài thực hành tiếp theo chúng ta sẽ làm quen nhiều lệnh hơn với thư viện scipy để xử lý về ma trận…

5. Tài nguyên Python trên mạng về tính toán đại số

Sympy là một trong những gói phần mềm toán học (hệ CAS – Computer Algebra System) miễn phí trên mạng. Sympy được tích hợp sẵn các gói thư viện thực thi như math, numpy,… để người sử dụng có thể tiếp cận. Địa chỉ của Sympy tại: www.sympy.org và địa chỉ sử dụng sympy trực tuyến là https://live.sympy.org.

Sympy hỗ trợ các thư viện và hàm để để giải quyết các vấn đề về: tính toán tổ hợp, giải phương trình, giải tích, toán rời rạc, vẽ đồ thị, ma trận, hình học,… với nền tảng là gói mpmath. Cụ thể hơn, các tính năng của sympy sinh viên có thể tìm hiểu thêm tại: https://www.sympy.org/en/features.html.

Tuy nhiên, tính đến 1/2019, nhược điểm của sympy là hệ thống đang sử dụng là Python 2.7 (chưa phải là 3.x) và một số thư viện như scipy, networkx.

Trong các bài thực hành tiếp theo, một số vấn đề sẽ được thực hiện trên môi trường Sympy. Tuy vậy, đặc điểm lưu ý chính là Sympy có những đối tượng riêng của SymPy khi tương tác. Ví dụ sau có thể thực hiện trên môi trường IDE idle của Anaconda: Sử dụng Sympy để tính toán định thức ma trận.

Sinh viên thực hành:

>>> import sympy as sp

>>> M = sp.Matrix([[1, 3], [2, 4]])

>>> M

>>> M.det()

……………………………………………………………………..

>>> **v1 = sp.Matrix([5,7])**

……………………………………………………………………..

>>> **M.dot(v1)**

………………………………………………………………………..

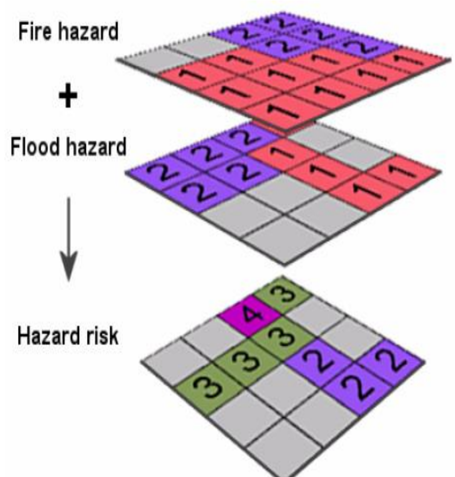
Tài liệu tham khảo:

1. http://ctr.maths.lu.se/na/courses/NUMA01/course_media/material/unit06_u7JvFS1.pdf
2. http://www-star.st-and.ac.uk/~pw31/CompAstro/IntroToPython.pdf
3. http://codingthematrix.com/slides/The_Matrix.pdf
4. Sách Pro Python của Marty Alchin, NXB: Apress, 2010.
5. https://meshlogic.github.io/posts/jupyter/linear-algebra/linear-algebra-numpy-2/

BÀI TẬP CHƯƠNG 1

[Yêu cầu sinh viên nộp bài]

Tìm chỗ đóng quân: Ông **Tùng** là một chỉ huy quân sự cao cấp. Ông **Tùng** nhận được bản đồ của một khu vực sườn núi có cây bao phủ được phân thành lưới 4x5 với mỗi ô lưới có kích thước làm 100 mét cho một chiến dịch hành quân. Bản đồ về các nguy cơ được xây dựng tại khu vực đó với các nhóm thông tin được cung cấp như sau:



TT	Nhóm nguy cơ	Cấp độ của nguy cơ				
		Không có	*Thấp*	*Trung bình*	*Cao*	*Rất cao*
1	Cháy rừng	0	1	2	3	5
2	Lũ quét	0	1	2	4	8
3	Sạt lở núi	0	1	3	5	9
4	Bệnh dịch	0	1	3	5	7
5	Lộ bí mật	0	5	10	15	20

Được biết: những nơi an toàn là những nơi có điểm tổng mức nguy cơ không vượt ngưỡng 5 điểm (≤ 5). Cho các lưới mô tả các bản đồ nguy cơ cụ thể: lưới A: bản đồ nguy cơ cháy rừng; lưới B: bản đồ nguy cơ lũ quét; lưới C: bản đồ nguy cơ sạt lở núi; lưới D: bản đồ nguy cơ dịch bệnh; lưới E: bản đồ nguy cơ lộ bí mật. Các lưới được thể hiện dưới dạng ma trận như sau:

$$A = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 \\ 3 & 1 & 0 & 1 & 1 \\ 5 & 2 & 0 & 1 & 2 \\ 2 & 0 & 1 & 2 & 3 \end{bmatrix}; B = \begin{bmatrix} 1 & 1 & 2 & 2 & 1 \\ 2 & 2 & 2 & 0 & 2 \\ 0 & 1 & 2 & 4 & 2 \\ 1 & 4 & 1 & 2 & 2 \end{bmatrix}; C = \begin{bmatrix} 0 & 5 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 3 \\ 1 & 3 & 1 & 3 & 1 \\ 0 & 1 & 3 & 3 & 0 \end{bmatrix}$$

$$D = \begin{bmatrix} 3 & 1 & 1 & 0 & 1 \\ 5 & 0 & 0 & 3 & 7 \\ 7 & 0 & 0 & 3 & 5 \\ 5 & 0 & 3 & 5 & 3 \end{bmatrix}; E = \begin{bmatrix} 0 & 0 & 0 & 10 & 0 \\ 0 & 0 & 15 & 0 & 0 \\ 0 & 5 & 15 & 5 & 0 \\ 0 & 20 & 5 & 0 & 0 \end{bmatrix}$$

Sinh viên hãy sử dụng ngôn ngữ Python và thư viện Numpy để giúp ông **Tùng** tính toán các kịch bản sau:

A. ***Chọn các vị trí đóng quân:***

a. An toàn trong chiến dịch ngắn hạn 1-2 ngày (có yếu tố tránh lộ bí mật; không quan tâm đến các yếu tố khác);

b. An toàn trong tập luyện thời bình (không cần xét yếu tố tránh lộ bí mật).

c. An toàn theo mùa khô (không có lũ, không sạt núi nhưng có cháy rừng và bệnh dịch).

d. An toàn trong mùa mưa (có lũ, có lỡ núi, có bệnh dịch mà không có cháy rừng).

e. An toàn trong 8 tháng (có đủ các mùa và có yếu tố tránh lộ bí mật)

B. ***Diện tích và số lượng quân:*** Tính diện tích các vị trí trên và tính toán số lượng quân có thể đóng tại khu vực đó theo các kịch bản trên, biết rằng trung bình mỗi chiến sỹ cần $10\mathrm{m}^2$ diện tích sinh hoạt an toàn./

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. Dẫn nhập – 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 >>> import scipy as linalg, sparse	Lệnh để nạp thư viện numpy vào bộ nhớ để xử lý

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)
	>>> print(M_upper)
		Ma trận trên từ đường chéo:
		……………………………………………………
		……………………………………………………
		……………………………………………………

10	>>> 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:

$$y-1=6 \Rightarrow y=\cdots$$

$$y-z=8\ \Rightarrow z=y-8=\cdots$$

$$x+z=1 \Rightarrow x=1-z=\cdots$$

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

$$x^2+2=6: Đúng\ hoặc\ sai\ với\ x=..$$

$$2x^2=8: Đúng\ hoặc\ sai\ với\ x=..$$

$$z^2+4=5: Đúng\ hoặc\ sai\ với\ z=..$$

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**(B)

>>> 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 như sau:

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

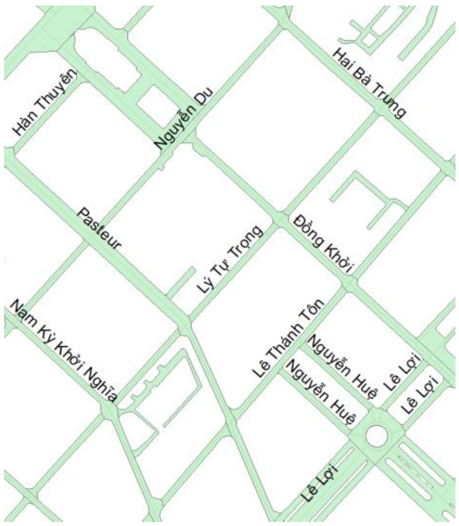
Trong đó, $x_1, x_2, \dots, x_n$ là các biến thực cần tìm và a_{ij}, b_j là các hằng số thực.

Quy trình giải hệ tuyến tính với numpy là:

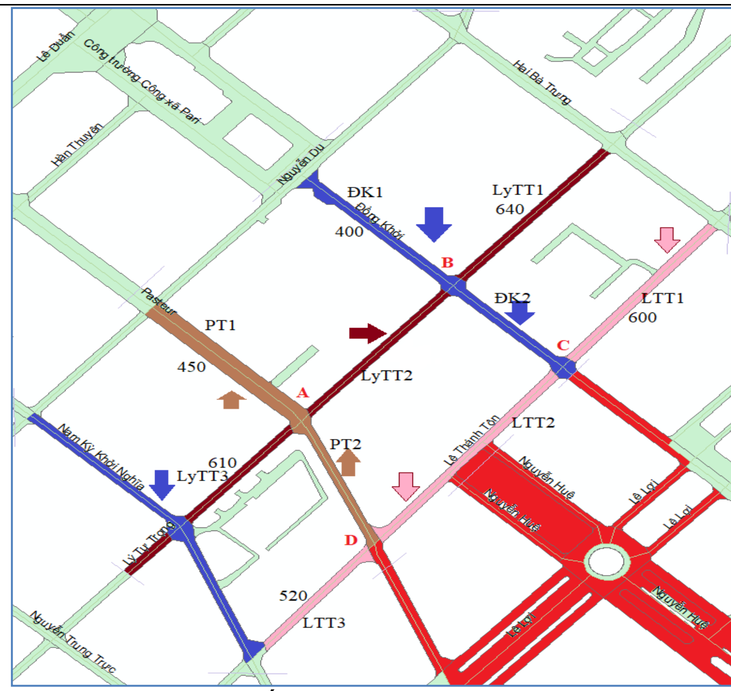
- **Bước 1: Thành lập ma trận A và vector với các hệ số a_{ij}, b_j.**
- **Bước 2: Tính toán ma trận nghịch đảo A^{-1} của A.**
- **Bước 3: Tìm vector nghiệm x bằng việc nhân ma trận nghịch đảo với vector.**

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

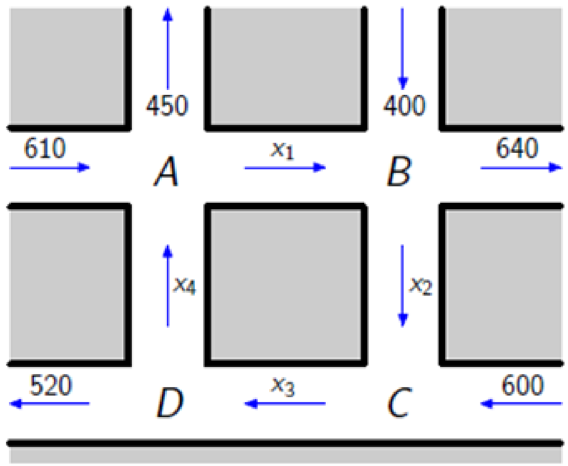


Hình thông tin số phương tiện tại các đoạn đường

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:

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

Hoặc hệ tương đương:

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

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)**

Hoặc lệnh: >>> **A_nghichdao = np.invert(A)**

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

>>> X = A_nghichdao * b

>>> X

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

....

....

....

....

....

BÀI TẬP BÀI 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 $\displaystyle\int_0^1 \frac{x(x-3)}{(x-1)^2(x+2)}\,dx$.

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**.

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 đường chéo.**

  .... ← 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

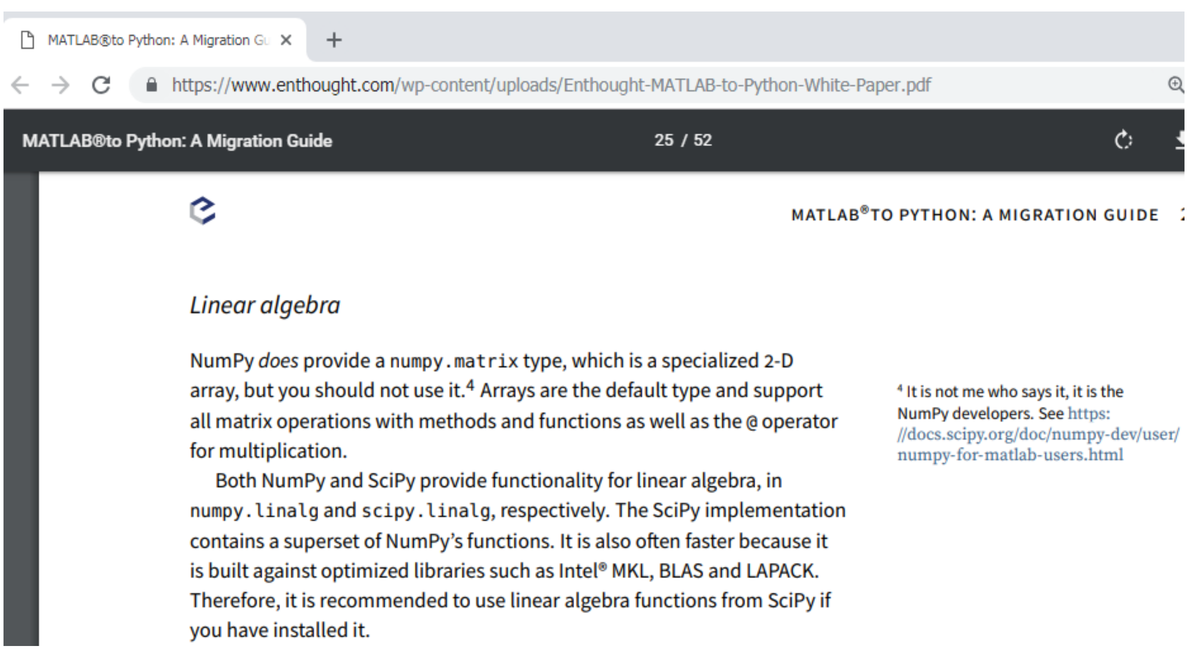
>>> **linalg.logm(A)**

.... ← 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



Linear algebra

NumPy *does* provide a `numpy.matrix` type, which is a specialized 2-D array, but you should not use it.[4] Arrays are the default type and support all matrix operations with methods and functions as well as the @ operator for multiplication.

Both NumPy and SciPy provide functionality for linear algebra, in `numpy.linalg` and `scipy.linalg`, respectively. The SciPy implementation contains a superset of NumPy's functions. It is also often faster because it is built against optimized libraries such as Intel® MKL, BLAS and LAPACK. Therefore, it is recommended to use linear algebra functions from SciPy if you have installed it.

[4] It is not me who says it, it is the NumPy developers. See https://docs.scipy.org/doc/numpy-dev/user/numpy-for-matlab-users.html

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:

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

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 viết/mô tả kết quả câu lệnh

....

....

....

....

....

- 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 viết/mô tả kết quả câu lệnh

....

....

....

....

....

- ***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 viết/mô tả kết quả câu lệnh

....

....

....

....

....

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 viết/mô tả kết quả câu lệnh

....

....

....

....

....

Gợi ý đáp án:

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

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:

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

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:

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

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:

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

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

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:

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

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

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

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

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

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 viết/mô tả kết quả câu lệnh

....

....

....

....

....

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

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

....

....

....

....

....

Gợi ý kết quả tính toán ma trận bậc 3:

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

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

>>> print (temp)

>>> print (sumA)
```

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

....

....

....

....

....

(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:

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

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.

BÀI 4: ĐỊNH THỨC

Mục tiêu:

- *Khái niệm và các tính chất, tính toán, quy tắc Cramer.*
- *Sử dụng sympy để tính toán hình thức các công thức đại số.*
- *Các ứng dụng của định thức.*

Nội dung chính:

1. Định thức và các tính chất

- **Phép thế của một tập hợp hữu hạn:**

Cho tập $X = \{x_1, x_2, \dots, x_n\}$ hoặc không mất tính tổng quát và gọn hơn, với tập $X = \{1,2, \dots, n\}$.

Khi đó, một **song ánh** $\sigma: X \to X$ gọi là một phép thế của X. Kí hiệu S_n là tập hợp các phép thế của X. Người ta thường biểu diễn phép thế như sau:

$$\sigma = \begin{pmatrix} 1 & 2 & \dots & n \\ \sigma(1) & \sigma(2) & \dots & \sigma(n) \end{pmatrix}$$

Vì σ là một song ánh nên $\sigma(i) \neq \sigma(j), i \neq j$. Do đó, $\sigma(1), \sigma(2), \dots, \sigma(n)$ chính là một hoán vị của tập $\{1,2, \dots, n\}$. Như vậy, dễ dàng thấy rằng số lượng phần tử trong tập S_n bằng $n!$. Dấu của phép thế σ được kí hiệu là $sgn(\sigma)$ là tích:

$$\prod_{\{i,j\}} \frac{i-j}{\sigma(i) - \sigma(j)}$$

Ví dụ cụ thể với tập $\{1,2,3,4\}$ và xét một phép thế sau:

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

Phép thế σ khi đó có dấu là: $sgn(\sigma) = \frac{1-2}{4-3} . \frac{1-3}{4-2} . \frac{1-4}{4-1} . \frac{2-3}{3-2} . \frac{2-4}{3-1} . \frac{3-4}{2-1} = 1$

Tính chất quan trọng cần lưu ý: $sgn(\sigma\tau) = sgn(\sigma). sgn(\tau)$.

Sinh viên thực hành: Xây dựng hàm tính toán dấu của một phép thế bằng Python:

Đầu vào:

- Tập X: cho n phần tử, tập $X = \{1,2, \dots, n\}$.
- Phép thế: 1 hoán vị của X.

Đầu ra: dấu của phép thế.

Thực hiện phương pháp tính cơ bản nhất như sau: Lần lượt tính giá trị bên trên tử số và giá trị ở mẫu số và chia tử số cho mẫu số và lặp đến hết các cặp số.

Sinh viên thực hành mã chương trình chi tiết dưới đây:

>>> **import numpy as np**

>>> n = 4

>>> X = np.array(range(1,n+1))

>>> sigma = np.array([4,3,2,1])

```
>>> def sgn_by_def(sigma):
        ket_qua = 1.0
        for i in range(len(X)-1):
                for j in range(i+1,len(X)):
                        ket_qua = ket_qua * ((X[i]-X[j])/(sigma[i]-sigma[j]))
        return int(ket_qua)
```

>>> sigma = np.array([2,1,3,4])

>>> sgn_by_def(sigma)

.... ← Sinh viên điền kết quả

Và thử nghiệm tính toán thêm một số phép thế khác (kiểm chứng bằng tính toán trên giấy):

>>> sigma = np.array([1,2,3,4])

>>> sgn_by_def(sigma)

.... ← Sinh viên điền kết quả

>>> sigma = np.array([4,3,2,1])

>>> sgn_by_def(sigma)

.... ← Sinh viên điền kết quả

- **Định thức:**

Định thức là giá trị liên quan đến một ma trận vuông. Định thức chứa nhiều thông tin của một ma trận như mối quan hệ giữa các vector tạo nên ma trận đó. Ví dụ: với ma trận khả nghịch (có ma trận nghịch đảo) thì định thức sẽ khác 0.

Với ma trận 2x2, định thức được tính theo công thức:

$$\begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc$$

Một cách tổng quát, theo định nghĩa, với ma trận A vuông $n \times n$ phần tử, kí hiệu của định thức $\det(A)$ hoặc $|A|$ được tính toán theo công thức:

$$\det(A) = |A| = \sum_{\sigma \in S_n} sgn(\sigma) a_{1\sigma(1)} a_{2\sigma(2)} \dots a_{n\sigma(n)}$$

Theo đó, để tính định thức một ma trận bất kỳ, chúng ta phải thực hiện các bước sau:

- Bước 1: Phát sinh tập $\mathrm{S_n} = \{\sigma_1, \sigma_2, \dots, \sigma_n\}$. Thật chất như trên đã phân tích, đây là phát sinh tập hoán vị của tập $\mathrm{X} = \{1, 2, \dots, \mathrm{n}\}$.
- Bước 2: Ban đầu đặt giá trị định thức bằng 0.
- Bước 3: *Lặp:* Với mỗi $\sigma_i \in \mathrm{S_n}$, tính toán dấu $\mathrm{sgn}(\sigma_i)$. Sau đó tính toán giá trị định thức cộng dồn các giá trị: $sgn(\sigma_i).\, a_{1\sigma_i(1)} a_{2\sigma_i(2)} \dots a_{n\sigma_i(n)}$

Sinh viên thực hành đoạn mã minh họa tính toán định thức của một ma trận theo định nghĩa.

- *Bước 1:*

```
>>> from itertools import permutations

>>> n = 3

>>> X = []

>>> for i in range (1, n+1):

        X.append(i)

>>> Sn = list(permutations(X))    # ← tạo hoán vị của tập X

>>> print (Sn)
```

.... ← Sinh viên điền kết quả

Lưu ý: khi đó, mỗi phần tử Sn(i) của tập Sn là một phép thế và được truy xuất có thứ tự từ 0 đến n!-1. Ví dụ: **Sn[3] =(2,3,1)**. Hơn nữa, khi đó, chúng ta có thể tìm vị trí của các phần tử như sau: >>> **Sn[3].index(2)** # sẽ là 0 vì 2 ở là vị trí đầu tiên của Sn[3].

- *Bước 2:*

```
>>> det = 0   # bước này có duy nhất 1 lệnh, có ý nghĩa khởi tạo giá trị ban đầu của định thức.
```

- Bước 3:

Xây dựng hàm tính dấu (sgn) cho tập Sn và giá trị ma trận

>>> **import numpy as np**

Dưới đây gồm 2 phần minh họa: Đoạn code thứ 1 minh họa tính toán ma trận hình thức. Nghĩa là liệt kê ra những phép toán cần tính toán theo công thức ma trận tổng quát vuông cấp n $A = (a_{ij})$. Và đoạn code thứ 2 là hiện thực để tính toán ma trận (ra cụ thể số):

Đoạn code 1:

```
>>> def phatsinh_dinhthuc(n):
        X = []
        for i in range(1, n+1):
                X.append(i)
        Sn = list(permutations(X))
        dinhthuc = ""
        for sn in Sn:
                sigma = np.array([1])
                sigma.resize([n])
                product = ""
                for i in range(1,n+1):
                        sigma[sn.index(i)] = i
                        product = product + "a"+str(i)+str(sn.index(i)+1)
                dau = sgn_by_def(sigma)
                if (dau != 1):
                        product = " - " + product
                else:
                        product = " + " + product
                dinhthuc = dinhthuc + product
        return dinhthuc
```

Lưu ý: sgn_by_def(sigma) là gọi hàm bên phía trên đã xây dựng.

Sau đó, chúng ta thử nghiệm các giá trị phát sinh của n=2,3:

```
>>> phatsinh_dinhthuc(2)
' + a11a22 - a12a21'
```

.... ← Sinh viên giải thích kết quả.

>>> phatsinh_dinhthuc(3)

.... ← Sinh viên điền kết quả và giải thích

....

Đoạn code 2:

```
>>> def tinhtoan_dinhthuc(A):
        X = []
        import math                 #  <-- bổ sung để sử dụng sqrt
        n = int(math.sqrt(A.size))  #  <-- với ma trận vuông, kích thuớc là căn số
        for i in range(1, n+1):
                X.append(i)
        Sn = list(permutations(X))
        dinhthuc = 0                 # ban đầu giá trị định thức là 0
        for sn in Sn:
                sigma = np.array([1])
                sigma.resize([n])
                product = 1           # đặt giá trị cho tích ban đầu là 1
                for i in range(1,n+1):
                        sigma[sn.index(i)] = i
                        product = product * A[i-1][sn.index(i)] # lưu ý chỉ số
                        #print (A[i-1][sn.index(i)])
                dau = sgn_by_def(sigma)
                if (dau != 1):                    # không xét trường hợp dấu =1
                        product = product * (-1)  #khi dấu  = -1 thì nhân với -1
                dinhthuc = dinhthuc + product
        return dinhthuc
```

Sử dụng đoạn code 2: Nhập một ma trận và tính giá trị định thức

>>> matran = **np.array**([[3,5,-8], [4, 12, -1], [2,5,3]])

>>> tinhtoan_dinhthuc(A)

.... ← Sinh viên điền kết quả kiểm tra

- **Các tính chất của ma trận:**

Dưới đây tóm tắt lại một số tính chất khác của định thức:

- Tính chất 1: định thức của ma trận đơn vị I bằng 1, nghĩa là:

$$det\ I = 1$$

- Tính chất 2: nếu thay đổi vị trí 2 dòng của ma trận thì dấu của định thức sẽ thay đổi.
- Tính chất 3a: Nếu nhân một dòng với giá trị thực t thì định thức của ma trận sẽ tăng lên t lần:

$$\begin{vmatrix} ta & tb \\ c & d \end{vmatrix} = t\begin{vmatrix} a & b \\ c & d \end{vmatrix}$$

- Tính chất 3b: Tính tuyến tính:

$$\begin{vmatrix} a+a' & b+b' \\ c & d \end{vmatrix} = \begin{vmatrix} a & b \\ c & d \end{vmatrix} + \begin{vmatrix} a' & b' \\ c & d \end{vmatrix}$$

- Tính chất 4: Nếu ma trận có 2 dòng giống nhau, như vậy định thức của ma trận đó bằng 0.
- Tính chất 5: Với 2 dòng khác nhau, lấy hiệu 2 dòng k lần thì giá trị định thức vẫn không đổi.

[Đọc thêm] Chứng minh:

$$\begin{vmatrix} a & b \\ c-ka & d-kb \end{vmatrix} = \begin{vmatrix} a & b \\ c & d \end{vmatrix} - \begin{vmatrix} a & b \\ ta & tb \end{vmatrix} = \begin{vmatrix} a & b \\ c & d \end{vmatrix} - t\begin{vmatrix} a & b \\ a & b \end{vmatrix} = \begin{vmatrix} a & b \\ c & d \end{vmatrix}$$

- Tính chất 6: Ma trận tồn tại 1 dòng 0 thì định thức sẽ bằng 0.
- Tính chất 7: Định thức của ma trận hình tam giác bằng tích của các số trên đường chéo.
- Tính chất 8: $\mathbf{det\,AB} = (\mathbf{det\,A})(\mathbf{det\,B})$. Suy ra:

$$det\ A^{-1} = \frac{1}{detA}$$

- Tính chất 9: định thức ma trận chuyển vị bằng ma trận gốc: $\det A^{T} = \det A$

[Đọc thêm] Chứng minh: Ma trận A được viết thành tích 2 ma trận L (ma trận tam giác dưới) và U (ma trận tam giác trên) thì:

$$A\ =\ L\ U\ \rightarrow |A| = |L||U|$$

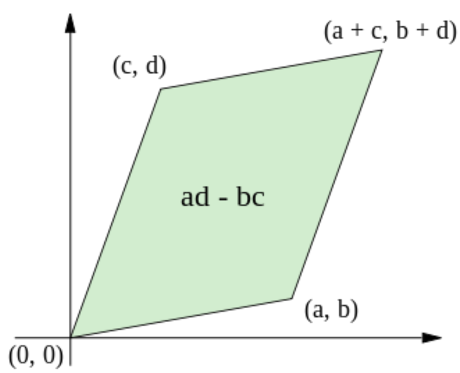
Do ma trận L và U là ma trận tam giác nên:

$$|L||U| = |U^T||L^T|$$

Với ma trận L có đường chéo bằng 1 nên định thức của L sẽ bằng 1. Do đó, còn lại ma trận nên ta dễ dàng có kết luận: $|A| = |A^T|$ (đpcm).

Biểu diễn hình học định thức

Giá trị định thức được hình dung như diện tích của hai vector (a,b) và (c,d) như hình sau:



Diễn giải: Biểu diễn định thức cùng với ma trận 2x2

$$M = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

Định thức (det) của ma trận M là:

$$det_M = ad - bc$$

Từ định nghĩa, dễ dàng thấy, giá trị định thức bằng 0 khi $ad = bc$.

Khi giá trị định thức âm, điều này có liên quan đến ý nghĩa về hướng của đối tượng hình học.

Với hình học, điều này sẽ tương ứng với diện tích bằng 0, nghĩa là: 2 đường thẳng ab và cd nằm trùng với nhau. Nói cách khác, vector (a, b) sẽ bằng một tỉ lệ với vector (c, d).

2. Định thức và ma trận khả nghịch

Mối liên hệ giữa ma trận khả nghịch A (bậc n, hay $n \times n$) và ma trận nghịch đảo, gọi là A^{-1} là tích $AA^{-1} = I_n$ I_n là ma trận đơn vị cấp n.

- **Ma trận hệ số kép (ma trận cofactor):**

Cho ma trận A (bậc n), gọi ma trận C (bậc n) là ma trận hệ số kép với các giá trị

$$c_{ij} = (-1)^{i+j}\det(A_{ij})$$

Với ma trận A_{ij} là ma trận $(n-1) \times (n-1)$ từ ma trận A bằng cách loại bỏ các phần tử trên dòng i và cột j.

- **Ma trận liên hợp adj (ma trận adjoint):**

Ma trận liên hợp adj là chuyển vị của ma trận hệ số kép.

- **Ma trận nghịch đảo A^{-1}:**

Định lý: Từ ma trận A khả nghịch (invertable matrix), chúng ta có thể tìm được ma trận A^{-1} bằng công thức tính toán thông qua định thức của A như sau:

$$A^{-1} = \frac{1}{\det(A)} adj(A)$$

Với ma trận 2x2, sinh viên cần nhớ các giá trị sau:

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \rightarrow adj(A) = \begin{bmatrix} d & -b \\ -c & a \end{bmatrix} \rightarrow A^{-1} = \frac{1}{|A|} adj(A) = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

3. Quy tắc Cramer

Quy tắc Cramer được đặt tên theo nhà toán học Gabriel Cramer (1704 – 1752), người sử dụng định thức để giản hệ tuyến tính n phương trình n có ẩn số. Quy tắc này ứng dụng với hệ có nghiệm duy nhất (nếu có nghiệm). Theo đó, ví dụ hệ Cramer với hệ 2 phương trình tuyến tính 2 ẩn như sau:

$a_{11}x_1 + a_{12}x_2 = b_1,$

$a_{21}x_1 + a_{22}x_2 = b_2.$

Chúng ta sẽ có nghiệm:

$$x_1 = \frac{b_1a_{22} - b_2a_{12}}{a_{11}a_{22} - a_{21}a_{12}} \text{ và } x_2 = \frac{b_2a_{11} - b_1a_{21}}{a_{11}a_{22} - a_{21}a_{12}} \text{ với điều kiện } a_{11}a_{22} - a_{21}a_{12} \neq 0.$$

Ở dạng ma trận, 2 nghiệm trên được biểu diễn như sau:

$$x_1 = \frac{\begin{vmatrix} b_1 & a_{12} \\ b_2 & a_{22} \end{vmatrix}}{\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix}}, x_2 = \frac{\begin{vmatrix} a_{11} & b_1 \\ a_{21} & b_2 \end{vmatrix}}{\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix}}, \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} \neq 0$$

Nếu đặt $|A_1| = \begin{vmatrix} b_1 & a_{12} \\ b_2 & a_{22} \end{vmatrix}$, $|A_2| = \begin{vmatrix} a_{11} & b_1 \\ a_{21} & b_2 \end{vmatrix}$ và $|A| = \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix}$

Khi đó, ta có: $x_1 = \frac{|A_1|}{|A|}$ và $x_2 = \frac{|A_2|}{|A|}$

Sinh viên thực hành giải hệ hai phương trình tuyến tính bằng Python:

- **Hệ hai phương trình tuyến tính bậc 1:**

$4x_1 - 2x_2 = 10,$

$3x_1 - 5x_2 = 11.$

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

>>> A = np.array([[4, -2],[3, -5]])   # khai báo ma trận A

>>> A1 = np.array([[10, -2],[11, -5]])

>>> A2 = np.array([[4, 10],[3, 11]])

>>> **from** scipy **import linalg** *# lưu ý: hàm tính định thức của ma trận là scipy.linalg.det(A)*

>>> detA = linalg.det(A)   # tính định thức cho ma trận A

>>> detA1 = linalg.det(A1)

>>> detA2 = linalg.det(A2)

>>> print (detA, detA1, detA2)

.... ← Sinh viên điền kết quả kiểm tra

>>> if (detA != 0):

x1 = detA1 / detA

x2 = detA2 / detA

print ("Hai nghiệm của phương trình là: ", x1, x2)

.... ← Sinh viên điền kết quả kiểm tra

Sinh viên sử dụng hàm **tinhtoan_dinhthuc()** tính toán các định thức A, A1, A2:

>>> .... ← Sinh viên viết các lệnh

>>> .... ← Sinh viên viết các lệnh

>>> .... ← Sinh viên viết các lệnh

- **Hệ ba phương trình tuyến tính bậc 1:**

Giải hệ sau bằng 2 phương pháp:

- Sử dụng hàm det() của lớp linalg của thư viện scipy.
- Sử dụng hàm **tinhtoan_dinhthuc()** viết bên trên

$$-x + 2y - 3z = 1$$

$$2x - 2y + z = 3$$

$$3x - 4y + 4z = 2$$

Thực hiện: Khai báo các ma trận (np.array(…)):

>>>.. ← Sinh viên viết các lệnh

>>> .... ← Sinh viên viết các lệnh

>>> .... ← Sinh viên viết các lệnh

- Tính toán giải theo lệnh **det**() của lớp **linalg** trong thư viện **scipy** và in nghiệm:

>>>.. ← Sinh viên viết các lệnh

>>> .... ← Sinh viên viết các lệnh

>>> .... ← Sinh viên viết các lệnh

>>> x, y, z = detX/det, detY/det, detZ/det ……………….. # tính toán x, y, z

>>> print (“Nghiệm của phương trình: ”, “x=”, x, “y=”, y, “z = “, z)

- Tính toán giải theo lệnh **tinhtoan_dinhthuc()** và in nghiệm:

>>>...← Sinh viên viết các lệnh

>>> ....← Sinh viên viết các lệnh

>>> .... ← Sinh viên viết các lệnh

>>> x, y, z = detX/det, detY/det, detZ/det ……………….. # tính toán x, y, z

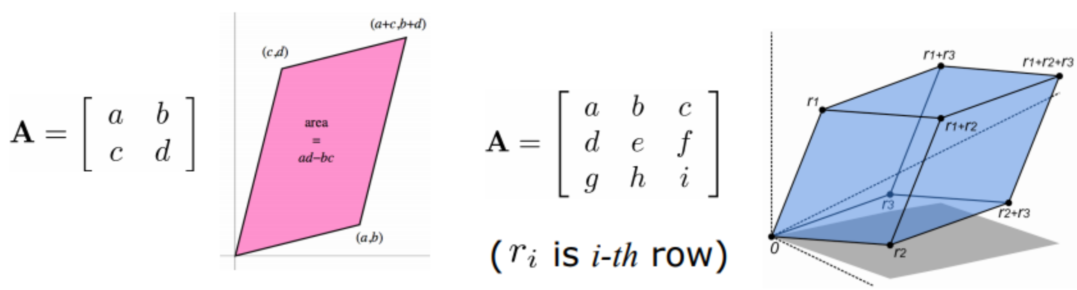
>>> print (“Nghiệm”, “x=”, x, “y=”, y, “z = “, z)

Với hệ n ẩn, n phương trình tuyến tính bậc 1 $Ax = b$, mỗi nghiệm sẽ là thương của 2 định thức: định thức của ma trận A_i và định thức của ma trận A với A_i là ma trận hình thành từ ma trận A thay thế cột thứ i bằng vector b^T.

$$x_1 = \frac{\det(A_1)}{\det(A)}, x_2 = \frac{\det(A_2)}{\det(A)}, \dots, x_n = \frac{\det(A_n)}{\det(A)}$$

Với $x = (x_1, x_2, \dots, x_n)$, $b = (b_1, b_2, \dots, b_n)$, điều kiện là: $det(A) \neq 0$.

4. Bài toán ứng dụng 1: Tính diện tích đa giác, thể tích và các phương trình đường, mặt.

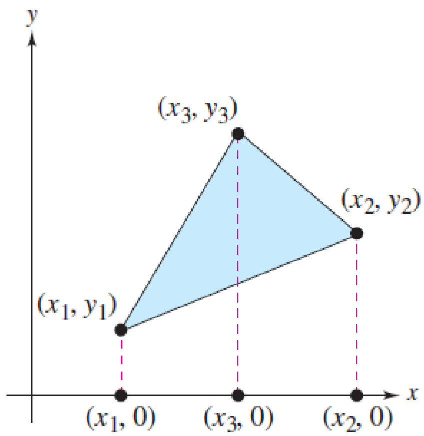


Tính toán định thức có thể áp dụng để tính diện tích của tam giác trong mặt phẳng. Giả định tam giác được cho bởi 3 điểm $A(x_1, y_1), B(x_2, y_2), C(x_3, y_3)$, khi đó, diện tích của tam giác ABC là:

$$S_{ABC} = \frac{1}{2} det \begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{bmatrix}$$

Thật vậy, không mất tính tổng quát, giả sử $y_i > 0$ và $x_1 \leq x_3 \leq x_2$.

Trường hợp (x_3, y_3) nằm phía trên đoạn nối giữa hai điểm (x_1, y_1) và (x_2, y_2) (như hình), thì chúng ta sẽ có 3 hình thang.



Hình thang 1: Giới hạn bởi các điểm: $(x_1, 0)$, (x_1, y_1), (x_3, y_3), $(x_3, 0)$.

Hình thang 2: Giới hạn bởi các điểm: $(x_3, 0)$, (x_3, y_3), (x_2, y_2), $(x_2, 0)$.

Hình thang 3: Giới hạn bởi các điểm: $(x_1, 0)$, (x_1, y_1), (x_2, y_2), $(x_2, 0)$.

Như vậy, dễ dàng thấy được diện tích của tam giác chính hiệu tổng hai diện tích hình thang đầu tiên với hình thang thứ 3.

Công thức thể hiện là:

$$S_{\Delta ABC} = \frac{1}{2}(y_1 + y_3)(x_3 - x_1) + \frac{1}{2}(y_1 + y_3)(x_3 - x_1) - \frac{1}{2}(y_1 + y_2)(x_2 - x_1)$$
$$= \frac{1}{2}(x_1y_2 + x_2y_3 + x_3y_1 - x_1y_3 - x_2y_1 - x_3y_2)$$

$$= \frac{1}{2}\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix}$$

Lưu ý 1: do diện tích là một đại lượng dương nên chúng ta có thể biện luận thêm một số giá trị của định thức:

- Định thức bằng 0: nghĩa là 3 điểm sẽ có sự phụ thuộc tuyến tính, nghĩa là chúng sẽ thẳng hàng.
- Định thức âm: do việc tính toán hiệu diện tích các hình thang nên giá trị này cần đổi dấu để được diện tích dương.

Lưu ý 2: có thể giải thích bằng cách tính ½ diện tích hình bình hành (bằng tích vector)

Ví dụ: Tính diện tích tam giác của 3 điểm sau A(1, 0), B(4,3), C(2,2)

- Tính diện tích tam giác:

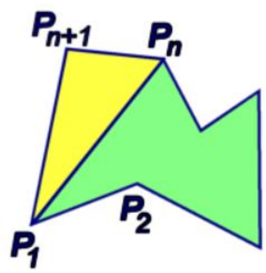
```
[Bằng gói Sympy]

>>> import sympy as sp
```

```
>>> TG = sp.Matrix([[1, 0, 1],[4, 3, 1], [2, 2, 1]])
>>> 1/2*TG.det()
```

Kết quả diện tích tam giác là: ....

- **Ứng dụng 1: Tính diện tích đa giác gồm n điểm (x_i, y_i):**

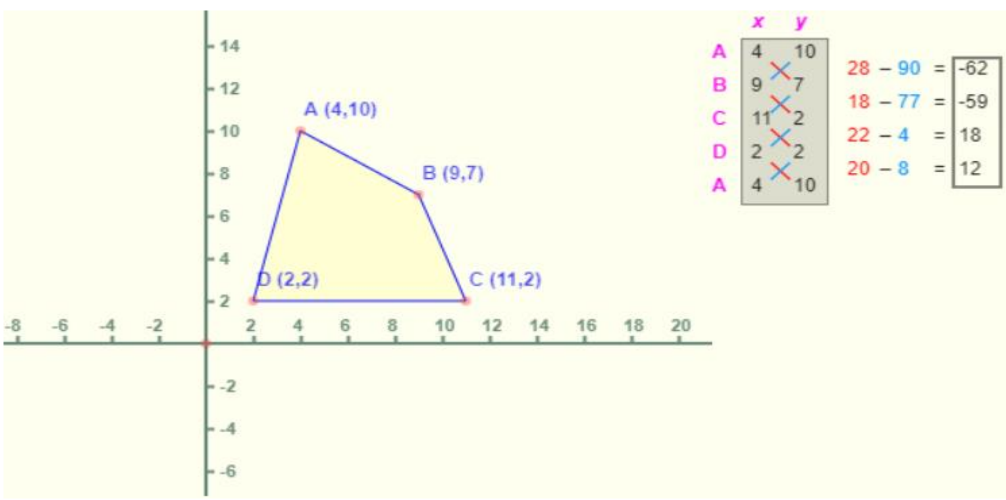
Bài toán: Cho một đa giác lồi (polygon) P có n đỉnh (xi, yi). Hãy:



Tính toán diện tích đa giác có *n* điểm. *Gợi ý: Diện tích đa giác bằng tổng các diện tích của các tam giác ghép thành nên đa giác đó.*

$$Diện\ tích_P = \frac{1}{2}\left(\begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \end{vmatrix} + \begin{vmatrix} x_2 & y_2 \\ x_3 & y_3 \end{vmatrix} .. + \begin{vmatrix} x_{n-1} & y_{n-1} \\ x_n & y_n \end{vmatrix} + \begin{vmatrix} \boldsymbol{x_n} & \boldsymbol{y_n} \\ \boldsymbol{x_1} & \boldsymbol{y_1} \end{vmatrix}\right)$$

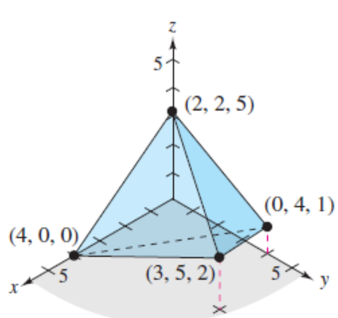
Minh họa:



Thực hành: Sinh viên hãy viết các câu lệnh Python để tính toán diện tích của n điểm:

- **Ứng dụng 2: Mở rộng bài toán tính diện tích thành thể tích hình tứ diện:**

Hãy tính thể tích của một tứ diện gồm bốn điểm (0, 4, 1), (4, 0, 0), (3, 5, 2) và (2, 2, 5).



Gợi ý: Thể tích là giá trị dương bằng $\frac{1}{6}$ giá trị định thức của ma trận bậc 4.

$$V = \mp\frac{1}{6}\begin{vmatrix} x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ x_3 & y_3 & z_3 & 1 \\ x_4 & y_4 & z_4 & 1 \end{vmatrix}$$

Giải:

```
>>> M = Matrix([[0, 4, 1, 1], [4, 0, 0, 1], [3, 5, 2, 1], [2, 2, 5, 1]])
```

```
>>> 1/6*M.det()
```

Giải bằng Sympy:

```
>>> M = Matrix([[0, 4, 1, 1], [4, 0, 0, 1], [3, 5, 2, 1], [2, 2, 5, 1]])
>>> M.det()
```

$$-72$$

```
>>> 1/6*M.det()
```

$$-12$$

```
>>>
```

Sinh viên kết luận: Thể tích của tứ diện là: ....

- **Ứng dụng 3: Kiểm 4 điểm nằm trên một mặt phẳng trong không gian ba chiều.**

Gợi ý: 4 điểm nằm trên mặt phẳng thì thể tích tứ diện bằng 0.

- **Ứng dụng 4: Phương trình mặt phẳng**

Hãy viết phương trình mặt phẳng đi qua 3 điểm (-1, 3, 2), (0, 1, 0), (-2, 0, 1).

Gợi ý: *Phương trình mặt phẳng đi qua 3 điểm là định thức:*

$$\begin{vmatrix} x & y & z & 1 \\ -1 & 3 & 2 & 1 \\ 0 & 1 & 0 & 1 \\ -2 & 0 & 1 & 1 \end{vmatrix}$$

Giải:

Lệnh thực thi:

Bước 1: Tham chiếu các thư viện và khai báo các biến

```
>>> from sympy import *  # lúc này ta sử dụng trực tiếp thư viện
```

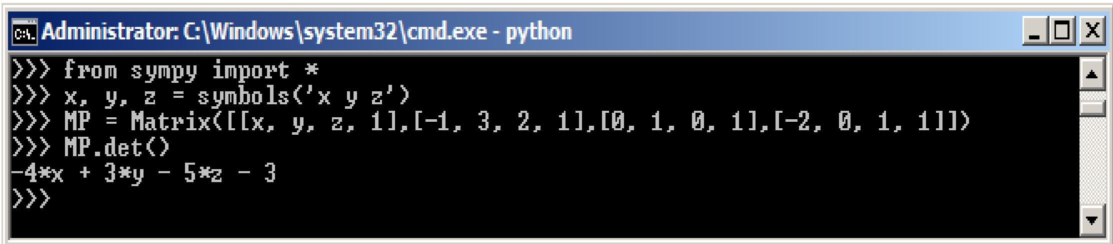
```
>>> x, y, z = symbols('x y z')
```

Bước 2: Khởi tạo ma trận:

```
>>> MP = Matrix([[x, y, z, 1],[-1, 3, 2, 1],[0, 1, 0, 1],[-2, 0, 1, 1]])
```

Bước 3: Thực thi

```
>>> MP.det()
```



Sinh viên hãy viết phương trình mặt phẳng:

....

5. Bài toán ứng dụng 3 – Tính quỹ đạo của hành tinh/vệ tinh

Đọc thêm: Ứng dụng đại số tuyến tính để tính quỹ đạo hành tinh xung quanh Mặt trời. Theo định lý Kepler thứ 1, các hành tinh xoay quanh hệ mặt trời theo quỹ đạo Ellipse. Gọi mặt trời (hoặc Trái đất) là 1 tâm của Ellipse (tâm thứ 2 là một tâm ảo), khi đó, quỹ đạo của hành tinh/vệ tinh được xác định thông khi biết 5 tọa độ của hành tinh/vệ tinh.

Công thức tổng quát đối với 1 ellipse là:

$$ax^2 + bxy + cy^2 + dx + ey + f = 0$$

Khi đó, 6 giá trị a, b, c, d, e, f được xác định thông qua định thức:

$$\begin{bmatrix} x^2 & xy & y^2 & x & y & 1 \\ x_1^2 & x_1y_1 & y_1^2 & x_1 & y_1 & 1 \\ x_2^2 & x_2y_2 & y_2^2 & x_2 & y_2 & 1 \\ x_3^2 & x_3y_3 & y_3^2 & x_3 & y_3 & 1 \\ x_4^2 & x_4y_4 & y_4^2 & x_4 & y_4 & 1 \\ x_5^2 & x_5y_5 & y_5^2 & x_5 & y_5 & 1 \end{bmatrix}$$

Hãy sử dụng gói Sympy để tính toán quỹ đạo các ellipse.

BÀI TẬP CHƯƠNG 4

Câu 1: Hãy viết chương trình để tính các ma trận:

a. Ma trận hệ số kép (cofactor matrix).
b. Ma trận liên hợp (adjoint matrix).

Từ một ma trận A nxn cho trước.

Câu 2: Hãy xây dựng ma trận tính quỹ đạo của một vật thể xoay theo quỹ đạo đường tròn.

Viết minh họa bằng một chương trình sử dụng gói Sympy (sinh viên tự viết) để tìm các phương trình từ các điểm biết được của đường tròn.

BÀI 5: ĐỊNH THỨC, MA TRẬN VÀ ỨNG DỤNG

Mục tiêu:

- *Ôn luyện về các lệnh xử lý ma trận và định thức trong gói numpy.*
- *Một số ứng dụng của định thức và ma trận.*
- *Giới thiệu về sai số, bình phương cực tiểu (giải hệ có số phương trình nhiều hơn ẩn)*

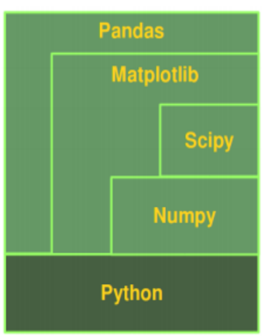
Nội dung chính:

1. Sử dụng array và matrix trong numpy

Dưới đây là tóm gọn một số lưu ý khi sử dụng kiểu ***matrix*** *và* ***array*** *trong gói* ***numpy****:*

a. Thông tin tóm gọn

- Numpy: là thư viện phần mềm hướng đến xây dựng kiểu dữ liệu, hàm xử lý/tính toán trên dữ liệu.
- Scipy: hướng đến các thuật toán toán học và các hàm thuận tiện trên nền tảng numpy.
- Sympy: hướng đến tính toán hình thức.



Một số lưu ý đặc biệt về dấu **[…]** và **(…)** khi sử dụng gói numpy:

Kí hiệu	Kí hiệu tương tự	Ý nghĩa
a(end)	a[-1]	Phần tử cuối của một vector
a(2,5)	a[1,4]	Phần tử của dòng 2, cột 5
a(2,:)	a[1] hoặc a[1,:]	Lấy toàn bộ dòng thứ 2
a(1:5,:)	a[0,5] hoặc a[:5] hoặc a[0:5,:]	Lấy 5 dòng đầu tiên của ma trận a
a(end-4:end.”)	a[-5:]	5 phần tử dòng cuối cùng

Các dạng sao chép mảng dữ liệu:

TT	Lệnh sao chép	Minh họa
1	Tạo 1 tham chiếu	B = A
2	Tạo một bản sao chép hoàn chỉnh	B = np.copy(A) B = A.copy()
3	Tạo một phiên bản copy từ dãy có sẵn	np.copyto(B,A) B[:] = a

Minh họa:

```
>>> import numpy as np
>>> D = np.array([[1,2],[3,4]])
>>> np.copyto(E, D)
```

....🡐 sinh viên ghi nhận kết quả: lỗi gì (nếu có)?

```
>>> E = np.array([[1,2],[3,5]])
>>> np.copyto(E, D)
```

.... 🡐 sinh viên ghi nhận kết quả

```
>>> print(E)
```

.... 🡐 sinh viên ghi nhận kết quả

b. Nên và không nên sử dụng kiểu dữ liệu numpy.matrix

Numpy hỗ trợ 2 kiểu dữ liệu để thể hiện ma trận, đó là numpy.array và numpy.matrix. Dưới đây là một số phân tích để hiểu hơn khi sử dụng kiểu dữ liệu numpy.matrix:

- *Nên sử dụng kiểu numpy.matrix:*
- Thuận tiện vì được hỗ trợ tất cả các hàm xử lý đại số tuyến tính.
- Hàm dựng uyển chuyển (tương thích với người sử dụng Matlab):
  Ví dụ:
  ```
  >>> import numpy as np
  >>> B = np.matrix("[1 2 3; 4 5 6]")
  ```
- Có đầy đủ các hàm **.H**, **.I** và **.A** hỗ trợ tính toán/xử lý nâng cao về đại số.
- Dữ liệu chỉ có ở dạng 2D. Tuy nhiên, mỗi phần tử có thể là một đối tượng.
  Ví dụ:
  ```
  >>> import numpy as np
  >>> x = np.array([['a','b'],['c','d']])
  ```

```
>>> y = 'x'
>>> C = np.matrix([[x,y],[1, 2]] )
>>> print (C)
```

.... # Sinh viên điền kết quả
....
....
....
....
....
....
....

- *Không nên/thể sử dụng kiểu numpy.matrix:*

- Kiểu matrix chỉ mô tả ma trận 2 chiều, không mô tả ma trận nhiều hơn 2 chiều.

```
>>> import numpy as np

>>> A = np.matrix([[[1,2],[3,4]],[[5,6],[7,8]]])   # ← lỗi. Sinh viên ghi nhận lỗi
```

....
....

```
>>> A = np.array([[[1,2],[3,4]],[[5,6],[7,8]]])    # sinh viên ghi nhận kết quả
>>> print(A)
```

....
....

Lưu ý:
- Hiện tại, nhiều gói xử lý khác có thể trả về đối tượng **array** thay vì trả về **matrix** nên khó khăn để tương thích trong cùng một hệ thống.
- Một số phép toán chồng nạp chưa được nhất quán.

c. Đối tượng matrix từ các hàm trong gói numpy.matlib

Để tương thích với người sử dụng Matlab, sau khi khai báo:

>>> from numpy import matlib

Chúng ta có thể sử dụng một số hàm sau để tạo đối tượng dạng matrix như sau:

empty Tạo ma trận rỗng	**zeros** Tạo ma trận toàn 0	**ones** Tạo ma trận toàn 1	**eye**
identity Tạo ma trận đơn vị	**repmat**	**Rand**	**Randn**

Sinh viên thực hành các lệnh sau:

>>> from numpy import matlib

>>> G = matlib.**identity**(5)   # cú pháp: matlib.identity(***n***)

>>> print (G)

.... # Sinh viên điền kết quả
....
....
....

>>> H = matlib.randn(3,2) # cú pháp: matlib.randn(****args***)

>>> print(H)

.... # Sinh viên điền kết quả
....
....
....

>>> K = matlib.zeros([4,4]) # ← cú pháp: matlib.zeros(***shape***)

>>> print(K)

.... # Sinh viên điền kết quả
....
....
....

2. Ứng dụng 2 –Liên phân số và số π

a. Liên phân số

Liên phân số, tiếng Anh gọi là Continued Fraction, là một dạng biểu diễn các số thực (hữu tỉ và vô tỉ) dưới dạng phân số nhiều tầng. Ví dụ đơn giản là:

$$\frac{9}{7} = 1 + \frac{1}{3 + \frac{1}{2}}$$

Từ giá trị trên, chúng ta có thể biểu diễn phân số theo cách "máy tính" có thể hiểu được là: $\frac{9}{7} = [1, 3, 2]$. Người ta chứng minh được rằng cách biểu diễn này sẽ duy nhất đối với mỗi số. Lưu ý, số $\frac{18}{14}$ cũng có biểu diễn tương tự số $\frac{9}{7}$. Giá trị phân số này là giá trị hữu hạn, ta gọi đó là liên phân số hữu hạn.

b. Liên phân số biểu diễn số π

Một ví dụ khác với số hữu tỉ: Theo lịch sử, trước khi có máy tính (thời "BC" – "Before Calculator" ☺), số π được xem là số mang giá trị $\frac{22}{7}$. Khi biểu diễn liên phân số giá trị $\frac{22}{7} = 3 + \frac{1}{7}$. Tuy vậy, khi toán học phát triển hơn (thời số thập phân, thời "AD" – viết tắt là "After Decimals" ☺ !) giá trị số $\pi = 3{,}14159265 \ldots$ và giá trị mới của π tương ứng với phân số $\frac{355}{113}$, hoặc ở dạng liên phân số, đó là:

$$\pi = 3 + \cfrac{1}{7 + \cfrac{1}{15 + \cfrac{1}{1 + \cfrac{1}{292 + \cdots}}}}$$

Từ đó, số π có thể biểu diễn thành tập các giá trị của liên phân số là: $\pi = [3, 7, 15, 1, 292, \ldots]$, cụ thể viết nhiều số hơn là: $\pi = [3, 7, 15, 1, 292, 1,1,1,2,1,3,1,14,2,1,1,2,2,2,2,1,84 \ldots]$. Từ biểu diễn trên, dễ dàng chúng ta có thể tìm thấy các số hữu tỉ gần với số π (tất nhiên sẽ có sai số), như: $\frac{3}{1}, \frac{22}{7}, \frac{333}{106}, \frac{355}{113}, \ldots$ Tổng quát hơn, công thức liên phân số là:

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

Với liên phân số, người ta chứng minh được rằng có một cách xác định p_n và q_n như sau:

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

Sinh viên thực hiện tính toán số π:

```
>>> import numpy as np
>>> c = [3, 7, 15, 1, 292, 1, 1, 1, 2, 1, 3, 1, 14, 2, 1, 1, 2, 2, 2, 2, 1, 84]
>>> M = np.mat([[1,2],[3,4]])  # khởi tạo ma trận 2x2
>>> for i in range(len(c)):
        ci = np.mat([[1,1],[1,0]])
        ci[0,0] = c[i]
```

```
        if (i==0):
            M = ci
        else:
            M = M.dot(ci)
>>> print(M)
>>> print (M[0,0]/M[1,0])  # in số Pi theo dãy số  biểu diễn liên phân số.
```

.... ← sinh viên viết giá trị số Pi tính được.

Ngày nay, kỹ thuật liên phân số được sử dụng lưu trữ cơ sở dữ liệu. Sinh viên sẽ được tiếp cận trong các chương trình tiếp theo.

3. Về phân tích hồi quy tuyến tính

[Mục này cung cấp thêm kiến thức cho sinh viên về dạng phương trình skiny]

a. Dẫn nhập: khái niệm về sai số

Bài toán minh họa: Khi lấy cây thước dài 200mm để đo chiều dài một cái bàn 800mm, nếu càng nhiều lượt đo thì số lượng "chiều dài" khác nhau sẽ tăng lên. Mặc dù chiều dài của cái bàn và độ chính xác cây thước được xem như là 1 con số không đổi. Điều đó được giải thích là do trong mỗi lượt đo đều có **những sai số (hiệu số giữa số đo và giá trị thực)** nhất định.

Lưu ý: các sai số gây ra ở đây là hoàn toàn mang tính chất ngẫu nhiên. Sinh viên cần phân biệt dữ liệu có "sai số" với dữ liệu bị "sai lầm". Khái niệm dữ liệu "sai lầm" trong đo đạc là khi:

- Sử dụng công cụ (như thước) bị sai (như cây thước bị cong, độ đo của cây thước không đúng).
- Người đo cố tình đọc sai giá trị đo.
- Thực hiện phương pháp đo sai, như: đo ẩu, ...

Như vậy, nếu gọi chiều dài của bàn là x thì chúng ta chỉ có 1 biến cần tìm. Trong khi đó, mỗi lượt đo được xem như một phương trình để tìm x. Do đó, chúng ta không thể xác định x một cách đơn giản bằng việc giải phương trình vì trên thực tế mỗi lượt đo có khả năng có khác biệt giá trị với nhau. Tuy vậy, chiều dài của bàn sẽ nằm ở một khoảng giá trị "hợp lý" mà không thể vượt ra ngoài được. Ví dụ: chiếc bàn dài khoảng 80cm thì nó sẽ giao động ở một giá trị gần với 80cm chứ không thể vượt lên 3 hay 5 hay 10 mét!

Từ đó, mở rộng ra, chúng ta sẽ có một dạng hệ phương trình cần xác định nghiệm mà trong đó điều quan trọng là chúng ta tìm được sự liên hệ phụ thuộc giữa các biến chứ không chỉ dừng lại ở một giá trị.

b. Phương pháp bình phương cực tiểu (mô hình tuyến tính)

Xét câu chuyện khởi nghiệp sau: Nhóm 03 bạn Lan – Linh – Loan trồng một loại cây giống chất lượng cao cung cấp xuất khẩu. Trong 5 lần đo chiều dài 1 cây mẫu, bạn Lan có số liệu đo như sau:

Ngày đo	Chiều cao (cm)
1	1.0
2	2.0
3	4.0
4	4.0
5	6.0

Vấn đề đặt ra là:

- Cây mẫu phát triển như thế nào trong 5 ngày qua (để so sánh với tiêu chuẩn và cây khác).
- Dự đoán chiều cao của cây giống trong ngày thứ 6.

Câu chuyện trên bắt đầu có tranh cãi khi:

- Bạn Linh cho rằng: cây đã phát triển theo phương trình $f(x) = 0.5 + x$.
- Bạn Loan cho rằng: cây đã phát triển theo phương trình $f(x) = 1.2x$.

Hỏi bạn nào đúng, bạn nào sai? Bạn hãy giúp 3 bạn trên tìm ra quy luật gần đúng nhất cho cây.

Với ví dụ trên, chúng ta sẽ nghiên cứu về phân tích hồi quy tuyến tính theo mô hình bình phương cực tiểu. Đây là một trong những phương pháp để tìm sự tương quan giữa một biến phụ thuộc Y với một hay nhiều biến độc lập X với mô hình sử dụng hàm tuyến tính (bậc 1). Các tham số của mô hình được ước lượng từ dữ liệu. Ở đó, người ta sẽ cố gắng tìm một "đường cong" gọi là "gần" nhất so với dữ liệu có được. "Đường cong" đó gọi là xấp xỉ cho dữ liệu và cũng để "dự báo" dữ liệu mới hoặc "lấp" những dữ liệu không tìm thấy. Và trong trường hợp này "đường cong" cần tìm chính là một đường thẳng (phương trình bậc 1) có giá trị tổng bình phương sai số nhỏ nhất tương ứng với dữ liệu đầu vào.

Xét qua sai số tổng bình phương của 2 bạn Linh và Loan trong bảng sau:

Mô hình của bạn Linh: $\boldsymbol{f(x) = 0.5 + x}$				Mô hình của bạn Loan: $\boldsymbol{f(x) = 1.2x}$			
Ngày $\boldsymbol{x_i}$	y_i	$f(x_i)$	$[y_i - f(x_i)]^2$	*Ngày* x_i	y_i	$f(x_i)$	$[y_i - f(x_i)]^2$
1	1	1.5	$(-0.5)^2$	1	1	1.2	$(-0.2)^2$
2	2	2.5	$(-0.5)^2$	2	2	2.4	$(-0.4)^2$
3	4	3.5	$(+0.5)^2$	3	4	3.6	$(+0.4)^2$
4	4	4.5	$(-0.5)^2$	4	4	4.8	$(-0.8)^2$
5	6	5.5	$(+0.5)^2$	5	6	6.0	$(0.0)^2$
		Tổng:	1.25			Tổng:	1.00

Nhìn vào bảng tính, chúng ta nhận thấy rằng bạn Loan có vẻ đúng hơn bạn Linh khi sai số nhỏ hơn (1.0 < 1.25). Tuy vậy, vấn đề đặt ra là sự tồn tại của một mô hình toán học cụ thể là mô hình tuyến tính (đường thẳng) đã được chứng minh mà có sai số nhỏ nhất hay không?

Dưới đây là **phương pháp hồi quy tuyến tính bằng bình phương cực tiểu**: Với tập gồm n bộ số $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ cho trước. Hãy tìm một phương trình bậc 1:

$$f(x) = a_0 + a_1 x$$

để giá trị tổng bình phương các sai số là cực tiểu:

$$\min[y_1 - f(x_1)]^2 + [y_2 - f(x_2)]^2 + \cdots + [y_n - f(x_n)]^2$$

Phương pháp giải:

Có thể xem các giá trị y_i là những giá trị đo được, còn các giá trị $f(x_i)$ là những giá trị

Chúng ta có các biến đổi như sau:

$$y_1 = f(x_1) + [y_1 - f(x_1)]$$

$$y_2 = f(x_2) + [y_2 - f(x_2)]$$

$$\dots$$

$$y_n = f(x_n) + [y_n - f(x_n)]$$

Đặt $e_i = [y_i - f(x_i)]$ và thay $f(x_i) = (a_0 + a_1 x_i)$, chúng ta sẽ có hệ sau:

$$y_1 = (a_0 + a_1 x_1) + e_1$$

$$y_2 = (a_0 + a_1 x_2) + e_2$$

$$\dots$$

$$y_n = (a_0 + a_1 x_n) + e_n$$

Đặt:

$$Y = \begin{bmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{bmatrix}, X = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \dots & \dots \\ 1 & x_n \end{bmatrix}, A = \begin{bmatrix} a_0 \\ a_1 \end{bmatrix}, E = \begin{bmatrix} e_1 \\ e_2 \\ \dots \\ e_n \end{bmatrix}$$

Khi đó, chúng ta có hệ sau:

$$Y = XA + E$$

Bằng phương pháp toán, người ta chứng minh được rằng, với hệ này, chúng ta sẽ được nghiệm

$$\boldsymbol{A} = (\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{X})^{-1}\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{Y}$$

và giá trị tổng bình phương sai số là: $\boldsymbol{E}^{\boldsymbol{T}}\boldsymbol{E}$.

Lưu ý: việc chứng minh 2 công thức trên sinh viên sẽ được học trong các môn học khác. Sinh viên chỉ cần hiểu và áp dụng công thức để giải.

Áp dụng giải: Từ đó, sinh viên thực hiện việc giải hệ trên:

$$X = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \\ 1 & 5 \end{bmatrix}, Y = \begin{bmatrix} 1 \\ 2 \\ 4 \\ 4 \\ 6 \end{bmatrix}$$

Ta có:

$$A = (X^T X)^{-1} X^T Y = \left(\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 & 5 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \\ 1 & 5 \end{bmatrix} \right)^{-1} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 & 5 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 4 \\ 4 \\ 6 \end{bmatrix} = \begin{bmatrix} -0.2 \\ 1.2 \end{bmatrix} = \begin{bmatrix} a_0 \\ a_1 \end{bmatrix}$$

Như vậy, phương trình đường thẳng gần đúng nhất sẽ là: $\boldsymbol{y} = -\mathbf{0.2} + \mathbf{1.2}\boldsymbol{x}$ và sai số là 0.8!

Sinh viên điền các lệnh Python tương ứng:

- Nhập X, Y và tính X^T

>>> import numpy as np

>>> X = np.array(....)

>>> Y = np.array(....)

>>> XT = ...

- Tính toán cơ bản các giá trị các giá trị $\mathrm{A1} = (\mathrm{X}^\mathrm{T}\mathrm{X})^{-1}$ và $\mathrm{A2} = \mathrm{X}^\mathrm{T}\mathrm{Y}$

>>> ....

>>>..

- Tính giá trị $\mathrm{A} = \mathrm{A1.A2}$ và sinh viên tự suy ra phương trình $f(x) = a_0 + a_1 x$ với $\mathrm{A} = \begin{bmatrix} \mathrm{a}_0 \\ \mathrm{a}_1 \end{bmatrix}$

>>> A = A1.dot(A2)

....

Suy ra phương trình và tính các f_i

Để từ đó tính ra sai số bằng tổng của các bình phương sai số $(f_i - y_i)$.

BÀI TẬP CHƯƠNG 5

Câu 1: Một người đi xạ trị bệnh ung thư bằng việc bơm thuốc vào người. Khi vừa xạ trị xong, lượng thuốc tồn động trong người ở mức 10 đơn vị. Sau đó 1 giờ sau lượng thuốc còn lại là 8; lần lượt 2, 3, 4 giờ sau lượng thuốc còn lại là 7, 5 và 2.

Hãy viết các lệnh bằng Python để tìm phương trình tuyến tính bằng phương pháp bình phương cực tiểu với tập các cặp số (x, y) để tìm công thức giảm lượng thuốc đối với bệnh nhân theo thời gian:

$$(x, y) = \{(0, 10), (1, 8), (2, 7), (3, 5), (4, 2)\}$$

[*Kết quả tham khảo để kiểm chứng: Phương trình tuyến tính là:* $y = 10.2 - 1.9x$]