

GIỚI THIỆU VỀ ĐẠI SỐ TUYẾN TÍNH

Nguyễn Mạnh Hùng

AI Academy Vietnam

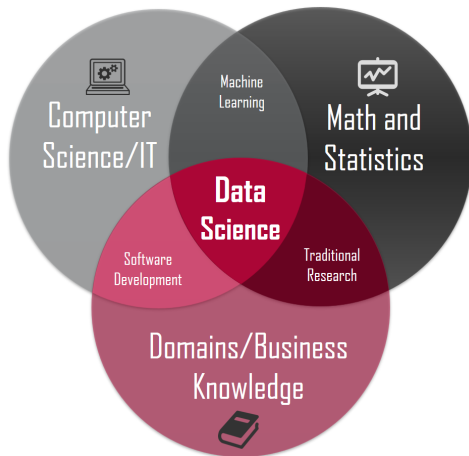
July, 2024

Nội dung

- 1 Giới thiệu
- 2 Biểu diễn dữ liệu bằng các khái niệm toán học
- 3 Thư viện NumPy
 - Giới thiệu về thư viện NumPy
 - Mảng ndarray
 - Các thao tác cơ bản với mảng ndarray
 - Đọc và ghi dữ liệu trên tệp

Khoa học dữ liệu (data science) là gì?

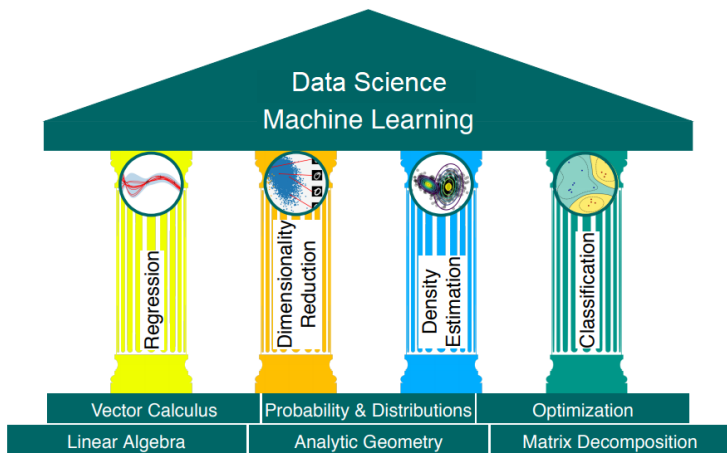
- Khoa học dữ liệu là một lĩnh vực liên ngành, là sự kết hợp của công nghệ thông tin, mô hình hóa và quản trị kinh doanh.
- Khoa học dữ liệu hướng đến việc sử dụng các phương pháp tiếp cận khoa học để trích xuất thông tin giá trị từ dữ liệu.



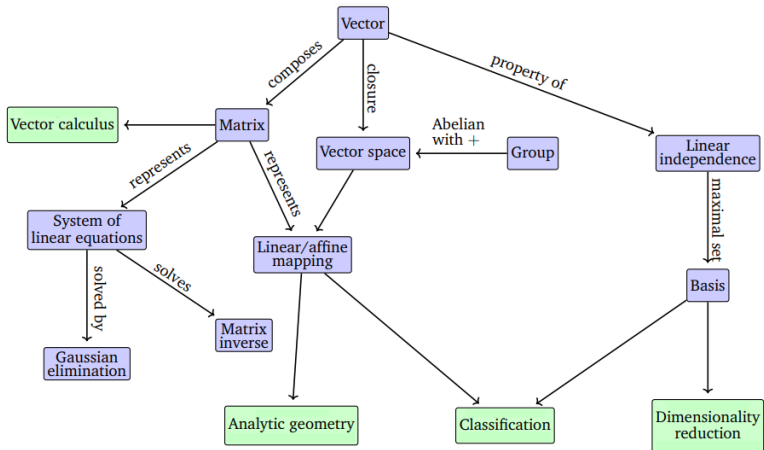
Học máy (machine learning)

- Học máy là việc nghiên cứu và thiết kế các thuật toán tự động trích xuất thông tin có giá trị từ dữ liệu.
- Thuật ngữ "tự động" có nghĩa là học máy quan tâm đến các phương pháp tổng quát có thể được áp dụng cho nhiều bộ dữ liệu trong quá trình giải quyết bài toán.
- Có ba khái niệm cốt lõi của học máy:
 - **dữ liệu** (data): là thành phần trọng tâm của học máy.
 - **mô hình** (model): được thiết kế để mô tả quá trình tạo ra dữ liệu, dựa trên dữ liệu được cung cấp. Một mô hình tốt có thể được sử dụng để dự báo những điều có thể xảy ra trong tương lai.
 - **học** (learning): sử dụng dữ liệu để xác định các tham số tốt nhất của mô hình.

Cơ sở toán học cho Khoa học dữ liệu



Sơ đồ các khái niệm trong Đại số tuyến tính



Mô hình màu RGB

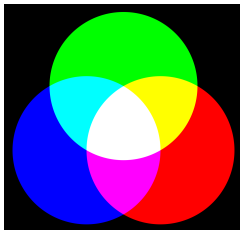
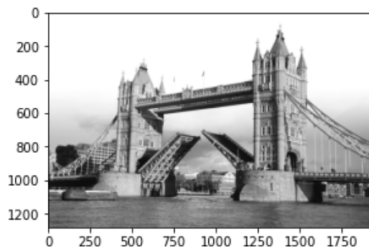


Figure 1: Phối trộn màu gốc (đỏ, xanh lục, xanh lam - RGB) để tạo ra các màu khác nhau

Color	HTML / CSS Name	Decimal Code (R,G,B)
	Black	(0,0,0)
	White	(255,255,255)
	Red	(255,0,0)
	Lime	(0,255,0)
	Blue	(0,0,255)
	Yellow	(255,255,0)
	Cyan / Aqua	(0,255,255)
	Magenta / Fuchsia	(255,0,255)
	Silver	(192,192,192)
	Gray	(128,128,128)
	Maroon	(128,0,0)
	Olive	(128,128,0)
	Green	(0,128,0)
	Purple	(128,0,128)
	Teal	(0,128,128)
	Navy	(0,0,128)

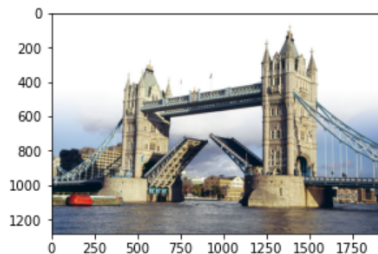
Biểu diễn ảnh xám bởi ma trận



```
[ [255.      255.      ... 255.      255.      ]
  [255.      255.      ... 255.      255.      ]
  [255.      255.      ... 255.      255.      ]
  ...
  [105.333336 108.333336 ... 85.333336 87.333336]
  [104.666664 107.333336 ... 84.333336 86.333336]
  [101.666664 103.666664 ... 81.333336 81.333336]]
```

Figure 2: Mỗi pixel được biểu diễn bởi một số nằm trong khoảng $[0, 255]$

Biểu diễn ảnh màu bởi một tensor bậc 3



```
[[[255 255 255] [255 255 255] ... [255 255 255] [255 255 255]]
 [[255 255 255] [255 255 255] ... [255 255 255] [255 255 255]]
 ...
 [[ 86  96 132] [ 87 100 135] ... [ 68  83 102] [ 70  85 104]]
 [[ 83  93 129] [ 85  95 131] ... [ 65  80  99] [ 65  80  99]]]
```

Figure 3: Mỗi pixel được biểu diễn một véc tơ (r, g, b)

Dữ liệu bảng

- Xét tập dữ liệu về nhân sự của một công ty cho trong bảng:

<i>Name</i>	<i>Gender</i>	<i>Degree</i>	<i>Postcode</i>	<i>Age</i>	<i>Annual salary</i>
<i>Aditya</i>	<i>M</i>	<i>MSc</i>	<i>W21BG</i>	<i>36</i>	<i>89563</i>
<i>Bob</i>	<i>M</i>	<i>PhD</i>	<i>EC1A1BA</i>	<i>47</i>	<i>123543</i>
<i>Chloé</i>	<i>F</i>	<i>BEcon</i>	<i>SW1A1BH</i>	<i>26</i>	<i>23989</i>
<i>Daisuke</i>	<i>M</i>	<i>BSc</i>	<i>SE207AT</i>	<i>68</i>	<i>138769</i>
<i>Elisabeth</i>	<i>F</i>	<i>MBA</i>	<i>SE10AA</i>	<i>33</i>	<i>113888</i>

- Mỗi một dòng của bảng dữ liệu biểu diễn một quan sát.
- Mỗi một cột biểu diễn một thuộc tính.
- Có thể bỏ qua thuộc tính 'Name' vì nó không mang lại nhiều thông tin khi giải quyết các bài toán.

Chuyển về định dạng số

- Mặc dù trong nhiều bài toán, dữ liệu không nhất thiết có định dạng số (dữ liệu gene, văn bản và ảnh, biểu đồ xã hội,...), ta có thể chuyển dữ liệu về định dạng số.
- Chuyển định dạng số:

Male/Female → +1/ - 1
 Bachelor/Master/PhD → 1/2/3
 Postcode → (Latitude, Longitude)

<i>Gender</i>	<i>Degree</i>	<i>Latitude</i>	<i>Longitude</i>	<i>Age</i>	<i>Annual salary</i>
+1	2	51.5073	0.1290	36	89.563
+1	3	51.5074	0.1275	47	123.543
-1	1	51.5071	0.1278	26	23.989
+1	1	51.5075	0.1281	68	138.769
-1	2	51.5074	0.1278	33	113.888

Mô tả dữ liệu bằng véc tơ, ma trận

- Mỗi một quan sát cho ta một bộ số x_n trong không gian D -chiều gồm các số thực, đó là một véc tơ trong không gian \mathbb{R}^D :

$$x_2 = (1, 3, 51.5074, 0.1275, 47, 123.543)$$

- Tập dữ liệu là một mảng hai chiều gồm các số thực, hay một ma trận kích thước $N \times D$, với N là số quan sát được thực hiện:

$$X = \begin{pmatrix} 1 & 2 & 51.5073 & 0.1290 & 36 & 89.563 \\ 1 & 3 & 51.5074 & 0.1275 & 47 & 123.543 \\ -1 & 1 & 51.5071 & 0.1278 & 26 & 23.989 \\ 1 & 1 & 51.5075 & 0.1281 & 68 & 138.769 \\ -1 & 2 & 51.5074 & 0.1278 & 33 & 113.888 \end{pmatrix}$$

Xây dựng mô hình dự báo

- Xét bài toán dự báo thu nhập theo độ tuổi: Một người 60 tuổi sẽ có thu nhập bằng bao nhiêu?
- Kí hiệu y_n là thu nhập gắn với quan sát x_n . Tập dữ liệu được viết dưới dạng các cặp quan sát: $\{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$

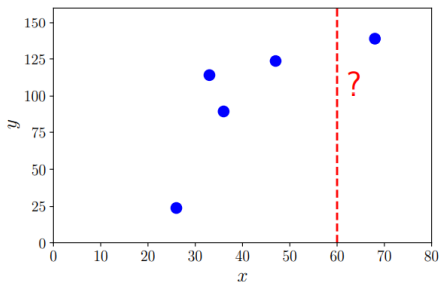
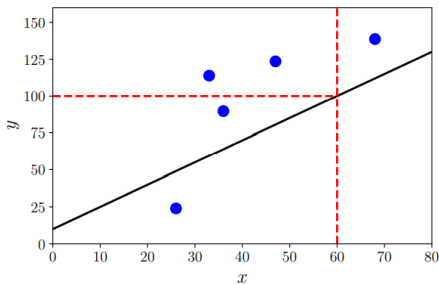


Figure 4: Dữ liệu (độ tuổi, thu nhập) cho bài toán hồi quy.

Xây dựng mô hình dự báo

- Để giải quyết bài toán, ta xây dựng một hàm dự báo, đầu vào là một quan sát (véc tơ các thuộc tính) và đầu ra là một số thực:

$$f : \mathbb{R}^D \rightarrow \mathbb{R}, \quad f(x) = \theta^T x \text{ (hồi quy tuyến tính)}$$



Tối ưu hóa hàm mất mát

- Việc xây dựng mô hình là đi tìm véc tơ tham số θ phù hợp nhất với tập dữ liệu.
- Để xác định mô hình phù hợp như thế nào với dữ liệu, ta định nghĩa hàm mất mát dạng ma trận:

$$\frac{1}{N} \|y - X\theta\|^2$$

- Véc tơ tham số θ được tìm sao cho hàm mất mát đạt cực tiểu.

Giới thiệu về thư viện NumPy

- NumPy (viết tắt của "Numerical Python") là package cơ bản để tính toán trong Python, đặc biệt đối với phân tích dữ liệu.
- NumPy cung cấp giao diện hiệu quả để lưu trữ và hoạt động trên các bộ đệm dữ liệu dày đặc.
- Tương tự như kiểu dữ liệu danh sách (list) tích hợp sẵn trong Python, mảng NumPy cho phép lưu trữ và xử lý dữ liệu hiệu quả hơn đối với các mảng có kích thước lớn.
- NumPy đã được cài đặt sẵn cùng với Anaconda. Để sử dụng thư viện NumPy, ta chỉ việc khai báo:

```
In : import numpy as np  
      np.__version__
```

```
Out : '1.18.5'
```


Ndarray

- Thư viện NumPy được xây dựng dựa trên một đối tượng chính: **ndarray** (N-dimensional array).
- Để khởi tạo ndarray, ta sử dụng hàm **array**(<danh sách>).
- Các thuộc tính của mảng:
 - dtype**: kiểu dữ liệu của phần tử trong mảng.
 - ndim**: số chiều của mảng.
 - size**: số phần tử trong mảng.
 - shape**: kích thước của mảng.

```
In [2]: import numpy as np
a=np.array([[1,2,3],[4,5,6]])
print(a)
print("Kiểu dữ liệu:",a.dtype)
print("Số chiều:",a.ndim)
print("Số phần tử:",a.size)
print("Kích thước:",a.shape)
```

```
[[1 2 3]
 [4 5 6]]
Kiểu dữ liệu: int32
Số chiều: 2
Số phần tử: 6
Kích thước: (2, 3)
```

Một số cách khởi tạo mảng ndarray

```
In [3]: np.zeros(5,dtype=int)
```

```
Out[3]: array([0, 0, 0, 0, 0])
```

```
In [4]: np.ones((2,3),dtype=float)
```

```
Out[4]: array([[1., 1., 1.],  
               [1., 1., 1.]])
```

```
In [5]: np.full((3,3),1.21)
```

```
Out[5]: array([[1.21, 1.21, 1.21],  
               [1.21, 1.21, 1.21],  
               [1.21, 1.21, 1.21]])
```

```
In [6]: np.arange(0,10,2)
```

```
Out[6]: array([0, 2, 4, 6, 8])
```

```
In [7]: np.linspace(0,1,5)
```

```
Out[7]: array([0. , 0.25, 0.5 , 0.75, 1.  ])
```

```
In [8]: np.random.random((2,2))
```

```
Out[8]: array([[0.50657786, 0.37013013],  
               [0.53458741, 0.76772808]])
```

```
In [9]: np.random.normal(0,1,(2,2))
```

```
Out[9]: array([[ 0.36293793,  0.55272843],  
               [-0.70357584,  0.34184057]])
```

```
In [10]: np.random.randint(0,45,(3,3))
```

```
Out[10]: array([[30, 10, 20],  
                [ 2, 30, 41],  
                [30, 32, 34]])
```

```
In [11]: np.eye(3)
```

```
Out[11]: array([[1., 0., 0.],  
                [0., 1., 0.],  
                [0., 0., 1.]])
```

Các kiểu dữ liệu của NumPy

Data type	Description
bool_	Boolean (True or False) stored as a byte
int_	Default integer type (same as C long; normally either int64 or int32)
intc	Identical to C int (normally int32 or int64)
intp	Integer used for indexing (same as C ssize_t; normally either int32 or int64)
int8	Byte (−128 to 127)
int16	Integer (−32768 to 32767)
int32	Integer (−2147483648 to 2147483647)
int64	Integer (−9223372036854775808 to 9223372036854775807)
uint8	Unsigned integer (0 to 255)
uint16	Unsigned integer (0 to 65535)
uint32	Unsigned integer (0 to 4294967295)
uint64	Unsigned integer (0 to 18446744073709551615)
float_	Shorthand for float64
float16	Half-precision float: sign bit, 5 bits exponent, 10 bits mantissa
float32	Single-precision float: sign bit, 8 bits exponent, 23 bits mantissa
float64	Double-precision float: sign bit, 11 bits exponent, 52 bits mantissa
complex_	Shorthand for complex128
complex64	Complex number, represented by two 32-bit floats
complex128	Complex number, represented by two 64-bit floats

Khi khởi tạo một mảng, có thể xác định kiểu dữ liệu như ví dụ sau:

```
np.zeros(10, dtype='int16')
```

hoặc

```
np.zeros(10, dtype=np.int16)
```

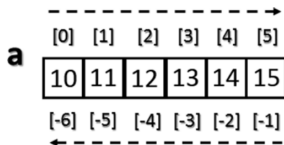
Các thao tác cơ bản với mảng ndarray

Bao gồm các nhóm thao tác sau đây:

1. Lấy thuộc tính của mảng
(*dtype*, *ndim*, *size*, *shape*)
2. Truy cập phần tử của mảng
3. Trích xuất mảng con
4. Thay đổi kích thước của mảng
5. Nối và tách mảng
6. Tính toán với mảng
7. Sắp xếp các phần tử trong mảng

Các thao tác cơ bản với mảng ndarray

2. Truy cập phần tử của mảng



A

	[,0]	[,1]	[,2]
[0,]	10	11	12
[1,]	13	14	15
[2,]	16	17	18

```
In [12]: a=np.array((2,13,5,-24,7))
a[3]
```

Out[12]: -24

```
In [13]: A=np.random.randint(-2,4,(3,3))
print(A)
A[1,1]
```

```
[[-1  1 -2]
 [ 1  2  2]
 [ 3  1  0]]
```

Out[13]: 2

Các thao tác cơ bản với mảng ndarray

3. Trích xuất mảng con

Cú pháp tương tự như đối với kiểu danh sách trong Python:

$x[\text{start} : \text{stop} : \text{step}]$

Các giá trị mặc định $\text{start} = 0$, $\text{stop} = \text{số phần tử}$, $\text{step} = 1$.

● Mảng 1 chiều:

```
In [14]: a=np.array([0,1,2,3,4,5,6,7,8])  
a[1:7:2]
```

```
Out[14]: array([1, 3, 5])
```

```
# Khi step nhận giá trị âm,  
# start và stop hoán đổi cho nhau  
In [15]: a[5:2:-1]
```

```
Out[15]: array([5, 4, 3])
```

Các thao tác cơ bản với mảng ndarray

- Mảng 2 chiều:

```
In [16]: A=np.array([[1,2,-3],[2,0,4],[-3,1,2]])
          A[:2,:2] # 2 hàng, 2 cột đầu tiên
```

```
Out[16]: array([[1, 2],
                [2, 0]])
```

```
In [17]: A=np.array([[1,2,-3],[2,0,4],[-3,1,2]])
          print(A[0,:],A[:,1]) # hàng 1, cột 2
```

```
[ 1  2 -3] [2 0 1]
```

- Thay đổi trên mảng con cũng làm mảng ban đầu thay đổi theo:

```
In [18]: A=np.array([[1,2,-3],[2,0,4],[-3,1,2]])
          A_sub=A[:2,:2]
          A_sub[0,0]=99
          A
```

```
Out[18]: array([[99,  2, -3],
                [ 2,  0,  4],
                [-3,  1,  2]])
```

Các thao tác cơ bản với mảng ndarray

- Nếu muốn tạo ra một mảng con và thay đổi trên mảng con không ảnh hưởng gì đến mảng ban đầu, ta sử dụng phương thức **copy()**:

```
In [19]: A=np.array([[1,2,-3],[2,0,4],[-3,1,2]])
A_sub_copy=A[:2,:2].copy()
A_sub_copy[0,0]=99
print(A_sub_copy)
A
```

```
[[99  2]
 [ 2  0]]
```

```
Out[19]: array([[ 1,  2, -3],
                [ 2,  0,  4],
                [-3,  1,  2]])
```


Các thao tác cơ bản với mảng ndarray

4. Thay đổi kích thước của mảng

- Tạo ra một mảng mới bằng phương thức **reshape()**:

```
In [20]: x=np.arange(1,7)
          print(x)
          x.reshape((2,3))
```

```
[1 2 3 4 5 6]
```

```
Out[20]: array([[1, 2, 3],
                [4, 5, 6]])
```

- Thay đổi mảng bằng cách thay đổi thuộc tính **shape**:

```
In [21]: x=np.arange(1,7)
          x.shape=(2,3)
          x
```

```
Out[21]: array([[1, 2, 3],
                [4, 5, 6]])
```

Các thao tác cơ bản với mảng ndarray

5. Nối và tách mảng

- Nối mảng bằng hàm **concatenate()**, **vstack()**, và **hstack()**

```
In [22]: x=np.array([1,1,1])
         y=np.array([2,2,2])
         np.concatenate([x,y])
```

```
Out[22]: array([1, 1, 1, 2, 2, 2])
```

```
In [23]: a=np.zeros((2,3),dtype=int)
         b=np.ones((2,3),dtype=int)
         np.concatenate([a,b])
```

```
Out[23]: array([[0, 0, 0],
               [0, 0, 0],
               [1, 1, 1],
               [1, 1, 1]])
```

```
In [24]: np.concatenate([a,b],axis=1)
```

```
Out[24]: array([[0, 0, 0, 1, 1, 1],
               [0, 0, 0, 1, 1, 1]])
```

Nối mảng theo chiều
ngang và chiều dọc

```
In [25]: np.vstack([a,b])
```

```
Out[25]: array([[0, 0, 0],
               [0, 0, 0],
               [1, 1, 1],
               [1, 1, 1]])
```

```
In [26]: np.hstack([a,b])
```

```
Out[26]: array([[0, 0, 0, 1, 1, 1],
               [0, 0, 0, 1, 1, 1]])
```

Các thao tác cơ bản với mảng ndarray

- Tách mảng bằng hàm **split()**, **vsplit()**, và **hsplit()**

In [27]: `x=np.array([1,1,1,2,2,3,3,3])`
`x1,x2,x3=np.split(x,[3,5])`
`print(x1,x2,x3,sep=" ")`

[3,5] là chỉ số của phần tử mà mảng tách tại đó

[1 1 1] [2 2] [3 3 3]

In [28]: `x=np.arange(9).reshape((3,3))`
`x`

Out[28]: `array([[0, 1, 2],`
`[3, 4, 5],`
`[6, 7, 8]])`

In [29]: `upper,lower=np.vsplit(x,[1])`
`print(upper,lower,sep="\n")`

[[0 1 2]]
 [[3 4 5]
 [6 7 8]]

Tách mảng theo hàng/cột tại chỉ số tương ứng

In [30]: `left,right=np.hsplit(x,[2])`
`print(left,right,sep="\n")`

[[0 1]
 [3 4]
 [6 7]]
 [[2]
 [5]
 [8]]

Các thao tác cơ bản với mảng ndarray

6. Tính toán với mảng

- NumPy cung cấp các phương thức đơn giản và linh hoạt để tính toán một cách tối ưu với các mảng dữ liệu lớn.
- Tính toán trên mảng NumPy có thể rất nhanh nhờ sử dụng các phép toán được **véc tơ hóa**, thực hiện thông qua các hàm phổ quát của Python (ufuncs - universal functions):

$$\begin{array}{r} \mathbf{a} \quad \begin{array}{|c|c|c|c|} \hline 0 & 1 & 2 & 3 \\ \hline \end{array} \\ \quad \quad \quad \begin{array}{cccc} + & + & + & + \end{array} \\ \mathbf{b} \quad \begin{array}{|c|c|c|c|} \hline 4 & 5 & 6 & 7 \\ \hline \end{array} \\ \quad \quad \quad \begin{array}{cccc} \downarrow & \downarrow & \downarrow & \downarrow \end{array} \\ \mathbf{a + b} \quad \begin{array}{|c|c|c|c|} \hline 4 & 6 & 8 & 10 \\ \hline \end{array} \end{array}$$

Các thao tác cơ bản với mảng ndarray

• Tính toán số học với mảng:

```
In [31]: x=np.arange(4)
print("x =", x)
print("x + 5 =", x + 5)
print("x - 5 =", x - 5)
print("x * 2 =", x * 2)
print("x / 2 =", x / 2)
print("x // 2 =", x // 2)
print("-x =", -x)
print("x ** 2 =", x ** 2)
print("x % 2 =", x % 2)
```

```
x = [0 1 2 3]
x + 5 = [5 6 7 8]
x - 5 = [-5 -4 -3 -2]
x * 2 = [0 2 4 6]
x / 2 = [0.  0.5 1.  1.5]
x // 2 = [0 0 1 1]
-x = [ 0 -1 -2 -3]
x ** 2 = [0 1 4 9]
x % 2 = [0 1 0 1]
```

Các phép toán số học này có thể thực hiện bởi các hàm tích hợp sẵn trong NumPy:

Operator	Equivalent ufunc
+	np.add
-	np.subtract
-	np.negative
*	np.multiply
/	np.divide
//	np.floor_divide
**	np.power
%	np.mod

Các thao tác cơ bản với mảng ndarray

- Giá trị tuyệt đối:

```
In [32]: x=np.array([1,-2,4,-5,1])
print(abs(x))
print(np.absolute(x))
print(np.abs(x))
```

```
[1 2 4 5 1]
[1 2 4 5 1]
[1 2 4 5 1]
```

- Hàm lượng giác

```
In [33]: phi=np.linspace(0,np.pi,3)
print("phi = ", phi)
print("sin(phi) = ", np.sin(phi))
print("cos(phi) = ", np.cos(phi))
print("tan(phi) = ", np.tan(phi))
```

```
phi = [0.          1.57079633 3.14159265]
sin(phi) = [0.0000000e+00 1.0000000e+00 1.2246468e-16]
cos(phi) = [ 1.0000000e+00 6.123234e-17 -1.0000000e+00]
tan(phi) = [ 0.0000000e+00 1.63312394e+16 -1.22464680e-16]
```

Các thao tác cơ bản với mảng ndarray

- Lũy thừa và logarit:

```
In [35]: x=np.array([1,2,3])
print("x =", x)
print("e^x =", np.exp(x))
print("2^x =", np.exp2(x))
print("3^x =", np.power(3, x))
print("ln(x) =", np.log(x))
print("log2(x) =", np.log2(x))
print("log10(x) =", np.log10(x))
```

```
x = [1 2 3]
e^x = [ 2.71828183  7.3890561 20.08553692]
2^x = [2.  4.  8.]
3^x = [ 3  9 27]
ln(x) = [0.          0.69314718 1.09861229]
log2(x) = [0.          1.          1.5849625]
log10(x) = [0.          0.30103  0.47712125]
```

Các thao tác cơ bản với mảng ndarray

- Các hàm tập hợp: là các hàm thực hiện tính toán trên toàn bộ mảng dữ liệu và trả về một giá trị.

Function Name	Description
<code>np.sum</code>	Compute sum of elements
<code>np.prod</code>	Compute product of elements
<code>np.mean</code>	Compute median of elements
<code>np.std</code>	Compute standard deviation
<code>np.var</code>	Compute variance
<code>np.min</code>	Find minimum value
<code>np.max</code>	Find maximum value
<code>np.argmin</code>	Find index of minimum value
<code>np.argmax</code>	Find index of maximum value
<code>np.median</code>	Compute median of elements
<code>np.percentile</code>	Compute rank-based statistics of elements
<code>np.any</code>	Evaluate whether any elements are true
<code>np.all</code>	Evaluate whether all elements are true

Các thao tác cơ bản với mảng ndarray

```
In [36]: data=np.random.randint(0,10,(3,4))
print("Dữ liệu: ",data,sep="\n")
print("Giá trị trung bình: ", data.mean())
print("Độ lệch tiêu chuẩn: ", data.std())
print("Giá trị nhỏ nhất: ", data.min())
print("Giá trị lớn nhất: ", data.max())
```

Dữ liệu:

```
[[6 9 5 1]
 [4 1 5 2]
 [7 1 3 8]]
```

Giá trị trung bình: 4.333333333333333

Độ lệch tiêu chuẩn: 2.6874192494328497

Giá trị nhỏ nhất: 1

Giá trị lớn nhất: 9

Các thao tác cơ bản với mảng ndarray

7. Sắp xếp các phần tử trong mảng

- Hàm **np.sort**: trả về một mảng với các phần tử được sắp xếp theo thứ tự tăng dần.
- Hàm **np.argsort**: trả về một mảng gồm các chỉ số của các phần tử được xếp theo thứ tự tăng dần.

```
In [37]: x=np.array([2,1,-4,3,5])
print("Mảng: ",x)
print("Sắp xếp tăng dần: ",np.sort(x))
print("Chỉ số các phần tử theo thứ tự:")
print(np.argsort(x))
```

```
Mảng: [ 2  1 -4  3  5]
Sắp xếp tăng dần: [-4  1  2  3  5]
Chỉ số các phần tử theo thứ tự:
[2 1 0 3 4]
```

Các thao tác cơ bản với mảng ndarray

- Sắp xếp trong hàng, cột của mảng nhiều chiều sử dụng tham số **axis**:

```
In [38]: X=np.random.randint(0,10,(3,3))
print(X)
print(np.sort(X,axis=0)," # Sắp xếp trong cột")
print(np.sort(X,axis=1)," # Sắp xếp trong hàng")
```

```
[[7 4 1]
 [3 6 0]
 [5 4 5]]
[[3 4 0]
 [5 4 1]
 [7 6 5]] # Sắp xếp trong cột
[[1 4 7]
 [0 3 6]
 [4 5 5]] # Sắp xếp trong hàng
```

Các thao tác cơ bản với mảng ndarray

- Sắp xếp từng phần: Nếu ta không cần sắp xếp toàn bộ mảng mà chỉ muốn tìm k —giá trị nhỏ nhất trong mảng, ta có thể dùng hàm **np.partition**.
- Cú pháp: **np.partition(<mảng>, <k>)** trả về một mảng có <k> phần tử nhỏ nhất nằm bên trái, phần còn lại nằm bên phải, theo thứ tự tùy ý.

```
In [39]: x=np.random.randint(0,10,(1,5))  
          print(x)  
          np.partition(x,3)
```

```
[[6 7 3 0 5]]
```

```
Out[39]: array([[0, 3, 5, 6, 7]])
```

3 số nhỏ nhất

Đọc và ghi dữ liệu mảng trên tệp (file)

1. Đọc và lưu file nhị phân (đuôi **.npy**)

- Hàm **np.save** sẽ lưu mảng dữ liệu ra file với phần mở rộng **<.npy>**.
- Hàm **np.load** sẽ lấy dữ liệu từ file (đuôi **.npy**) và gán cho một mảng.

```
In [40]: data=np.random.randint(0,10,(3,3))  
print(data)  
np.save('saved_data',data)
```

```
[[1 7 4]  
 [7 6 6]  
 [0 9 7]]
```

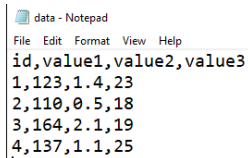
```
In [41]: loaded_data=np.load('saved_data.npy')  
loaded_data
```

```
Out[41]: array([[1, 7, 4],  
                [7, 6, 6],  
                [0, 9, 7]])
```

Đọc và ghi dữ liệu mảng trên tệp (file)

2. Đọc và lưu file text

- Hàm **np.savetxt** sẽ lưu mảng dữ liệu ra file text.
- Hàm **np.loadtxt** sẽ lấy dữ liệu từ file text và gán cho một mảng.



```
data - Notepad
File Edit Format View Help
id,value1,value2,value3
1,123,1.4,23
2,110,0.5,18
3,164,2.1,19
4,137,1.1,25
```

```
In [42]: data=np.loadtxt('data.txt', delimiter=',',skiprows=1)
data
```

```
Out[42]: array([[ 1. , 123. ,  1.4,  23. ],
 [ 2. , 110. ,  0.5,  18. ],
 [ 3. , 164. ,  2.1,  19. ],
 [ 4. , 137. ,  1.1,  25. ]])
```

Đọc và ghi dữ liệu mảng trên tệp (file)

- Hàm **np.genfromtxt()**: cho phép đọc dữ liệu dạng bảng từ file có định dạng text (.txt hoặc .csv).

```
In [43]: data = np.genfromtxt('data.txt', delimiter=',', names=True)  
data
```

```
Out[43]: array([(1., 123., 1.4, 23.), (2., 110., 0.5, 18.), (3., 16  
4., 2.1, 19.),  
              (4., 137., 1.1, 25.)],  
              dtype=[('id', '<f8'), ('value1', '<f8'), ('value2',  
'<f8'), ('value3', '<f8')])
```

- Khi đọc dữ liệu từ file, nếu có dữ liệu bị thiếu, hoặc không đúng định dạng số, hàm **genfromtxt** vẫn đọc được file và thay thế dữ liệu thiếu/sai bằng giá trị **nan**.

Tài liệu tham khảo

1. Charu C. Aggarwal; Linear Algebra and Optimization for Machine Learning, Springer, 2020.
2. Stephen Boyd, Lieven Vandenberghe; Introduction to Applied Linear Algebra: Vectors, Matrices, and Least Squares, Cambridge University Press, 2018.
3. David C. Lay, Steven R. Lay, Judi J. McDonald; Linear Algebra and Its Applications, Fifth edition, Pearson, 2016
4. Tom Lyche; Numerical Linear Algebra and Matrix Factorizations, Springer, 2020.
5. Gilbert Strang; Linear Algebra and Learning from Data, Wellesley- Cambridge Press, 2019.