Note) Slides and example codes are available:
https://github.com/mint-lab/prog_meets_math

Python Meets Mathematics: Linear Algebra

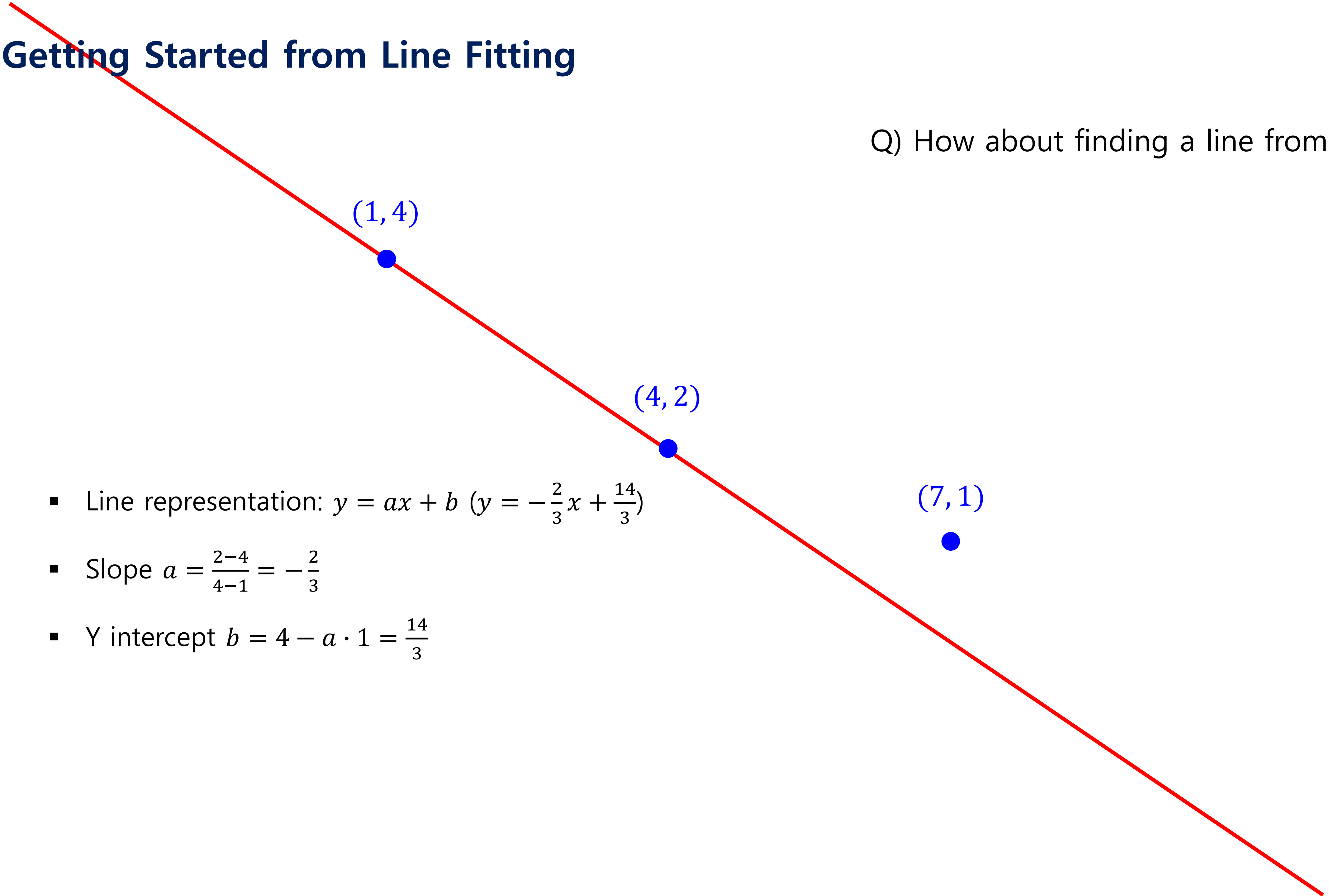
Sunglok Choi, Assistant Professor, Ph.D.
Computer Science and Engineering Department, SEOULTECH
sunglok@seoultech.ac.kr | https://mint-lab.github.io/

Programming meets Mathematics

- ~~Calculus~~ **Differentiation**
  - Visualization using Matplotlib
  - Differentiation
    - Meaning: The **slope** of a tangent line (~ linear approximation) of a function
    - Finding a **derivative** using rules, SymPy, and numerical differentiation
- **Linear Algebra**
- **Optimization**
- **Probability**
- **Information Theory**

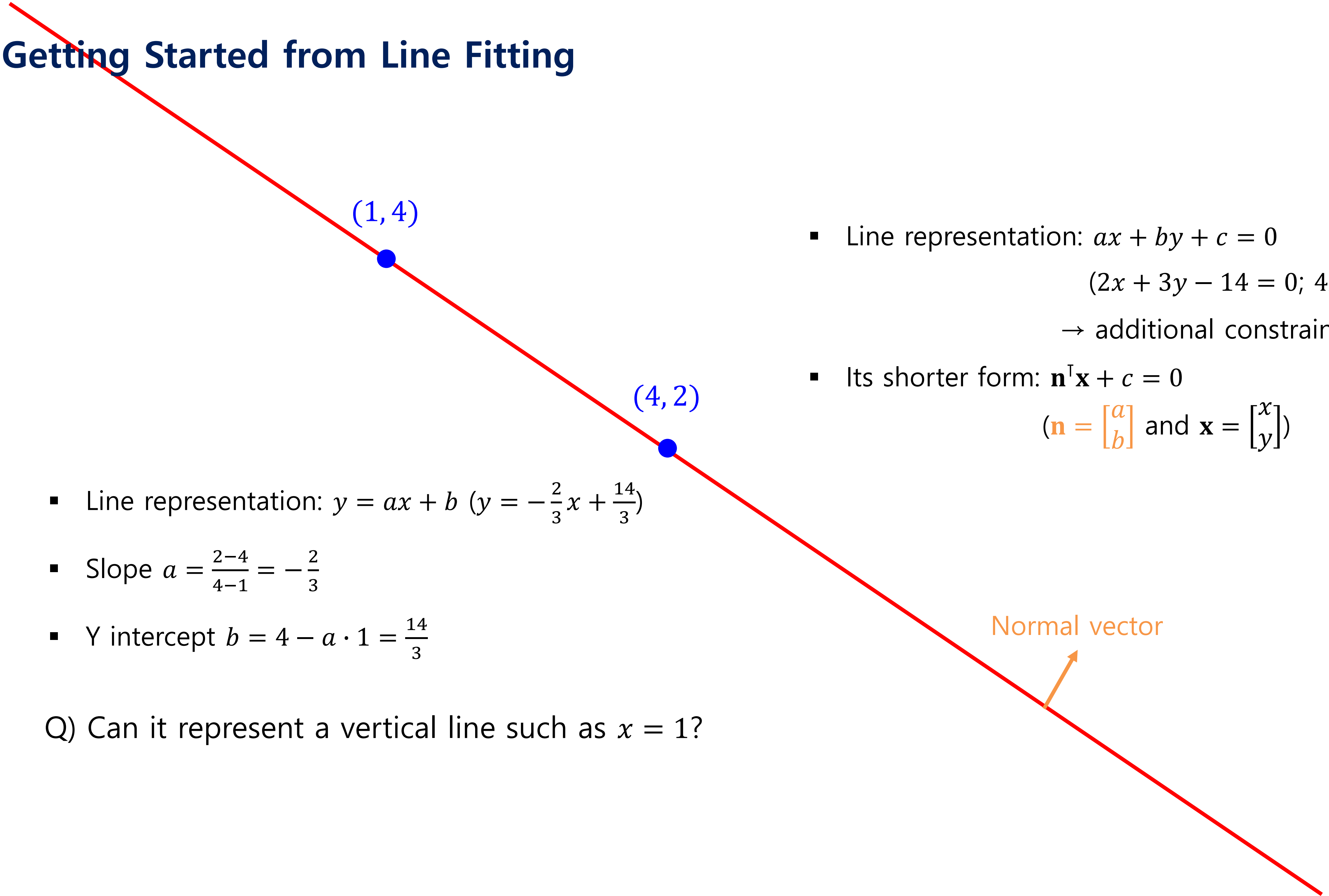
Getting Started from Line Fitting

Q) How about finding a line from three points?



- Line representation: $y = ax + b$ ($y = -\frac{2}{3}x + \frac{14}{3}$)
- Slope $a = \frac{2-4}{4-1} = -\frac{2}{3}$
- Y intercept $b = 4 - a \cdot 1 = \frac{14}{3}$

Getting Started from Line Fitting



- Line representation: $ax + by + c = 0$

  $(2x + 3y - 14 = 0;\ 4x + 6y - 28 = 0)$

  $\rightarrow$ additional constraint $a^2 + b^2 = 1$

- Its shorter form: $\mathbf{n}^\intercal \mathbf{x} + c = 0$

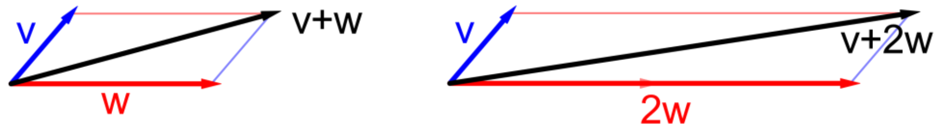
  ($\mathbf{n} = \begin{bmatrix} a \\ b \end{bmatrix}$ and $\mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix}$)

- Line representation: $y = ax + b$ ($y = -\frac{2}{3}x + \frac{14}{3}$)
- Slope $a = \frac{2-4}{4-1} = -\frac{2}{3}$
- Y intercept $b = 4 - a \cdot 1 = \frac{14}{3}$

Q) Can it represent a vertical line such as $x = 1$?

Linear Algebra

- **Algebra** is the study of mathematical objects and their manipulating rules (e.g. arithmetic operations).
- **Linear algebra** is the mathematical branch on linear equations represented in vector spaces.
  - **Linear equation**
    - e.g. $ax + b = 0$ (one-variable), $ax + by + c = 0$ (two-variable), …, $a_1x_1 + a_2x_2 + \cdots + a_nx_n + b = 0$ (n-variable)
    - Note) Nonlinear equations: $y = 0.1x^3 - 0.8x^2 - 1.5x + 5.4$, $y = \sin x$
  - **Vector space** (a.k.a. linear space): A set of vectors which can be 1) added together and 2) multiplied by scalars
    - Scalar: A single element
      - e.g. x (one-dimensional)



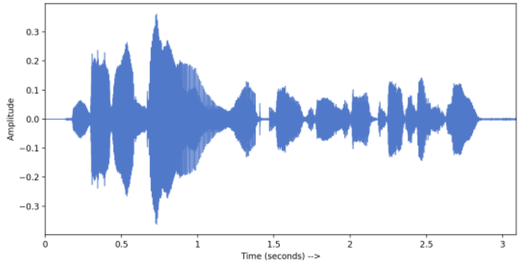
    - Vector: A finite ordered list of elements (a.k.a. tuple)
      - e.g. $[x, y]$ (two-dimensional row vector), …, $[x_1, x_2, \dots, x_n]^\intercal$ (n-dimensional column vector)
    - Matrix: A rectangular array (table) of elements arranged in rows and columns (a.k.a. tensor)
      - e.g. $m \times n$ matrix

$$\begin{array}{c c} & \begin{array}{cccc} 1 & 2 & \dots & n \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ \vdots \\ m \end{array} & \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ a_{31} & a_{32} & \dots & a_{3n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \end{array}$$

Image: Wikipedia (vector space, matrix)

Vector

- **Why a vector?**
  - To represent physical quantities
    - e.g. Speed (magnitude; 속력 in Korean) vs. velocity (magnitude+*direction*; 속도 in Korean)
  - To represent data and models
    - e.g. A point $[1, 4]$, a line $2x + 3y - 14 = 0 \rightarrow [2, 3, -14]$, and its normal vector $[2, 3]$ or $[0.554 \ldots, 8.832 \ldots]$
    - e.g. Sound data

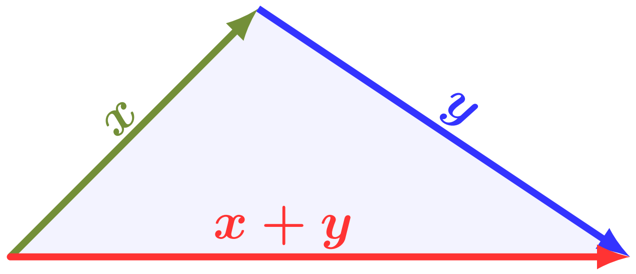


- **Vector operations**
  - Vector addition (subtraction): $\mathbf{a} + \mathbf{b}$
  - Scalar multiplication (division): $a\,\mathbf{b}$
  - Vector multiplication
    - Dot product (a.k.a. scalar product): $\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$ or $\mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\|\,\|\mathbf{b}\|\,\cos\theta$
    - Cross product (a.k.a. vector product): $\mathbf{a} \times \mathbf{b} = \|\mathbf{a}\|\,\|\mathbf{b}\|\,\sin\theta\;\mathbf{n}$

Image: Toward Data Science (an article by Kartik Chaudhary)

Vector

- **Norm** is a function from a vector space to the non-negative real numbers that behaves as like the distance from the origin (a.k.a. magnitude).
  - Notation: $\|\mathbf{a}\|$ for a vector $\mathbf{a}$
  - Properties
    - Triangle inequality: $\|\mathbf{x}+\mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|$
    - Absolute homogeneity: $\|a\,\mathbf{x}\| = |a|\,\|\mathbf{x}\|$
    - Positive definiteness: If $\|\mathbf{x}\| = 0$, then $\mathbf{x} = \mathbf{0}$



  - Examples
    - **Absolute-value norm**: A norm on 1-dimensional vector spaces

$$\|\mathbf{x}\| = |x_1|$$

    - **Euclidean norm** (a.k.a. L^2-norm): The most common norm in n-dimensional Euclidean spaces

$$\|\mathbf{x}\|_2 = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$$

    - **Manhattan norm** (a.k.a. L^1-norm)

$$\|\mathbf{x}\|_1 = |x_1| + |x_2| + \cdots + |x_n|$$

    - Note) A norm can be defined for a matrix.

Image: Wikipedia

Vector

- **Vector operations** (cont'd)
  - Vector multiplication
    - **Dot product** (a.k.a. scalar product, inner product)
      - Geometric definition: $\mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\|\ \|\mathbf{b}\|\ \cos\theta$
      - Algebraic definition: $\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$
        - As matrix multiplication: $\mathbf{a} \cdot \mathbf{b} = \mathbf{a}^\top \mathbf{b}$ where $\mathbf{a}$ and $\mathbf{b}$ are column vectors.
          $\mathbf{a} \cdot \mathbf{b} = \mathbf{a}\mathbf{b}^\top$ where $\mathbf{a}$ and $\mathbf{b}$ are row vectors.
        - e.g. $[3, 2] \cdot [5, 1] = 17$
        - e.g. $[3, 2, 9] \cdot [5, 1, -2] = -1$
      - Applications
        - Scalar projection
        - (Directional) similarity of two vectors (cosine similarity)

$$\cos\theta = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\|\ \|\mathbf{b}\|}$$

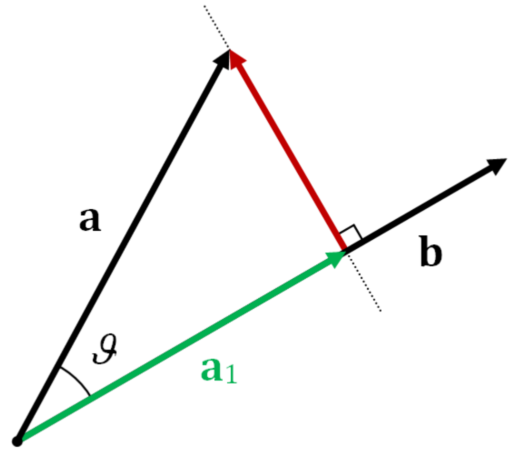


Image: Wikipedia

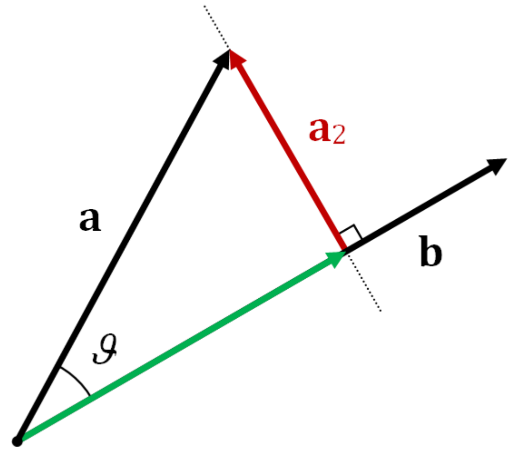
Vector

- **Vector operations** (cont'd)
  - Vector multiplication
    - **Cross product** (a.k.a. vector product) is defined only in a three-dimensional space.
      - Definition: $\mathbf{a} \times \mathbf{b} = \|\mathbf{a}\| \, \|\mathbf{b}\| \, \sin\theta \, \hat{\mathbf{n}}$

        where $\hat{\mathbf{n}}$ is a unit vector orthogonal (수직 in Korean) to a plane containing $\mathbf{a}$ and $\mathbf{b}$
      - Computing (using matrix determinant)

$$\mathbf{a} \times \mathbf{b} = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{vmatrix} = (a_2 b_3 - a_3 b_2)\mathbf{i} - (a_1 b_3 - a_3 b_1)\mathbf{j} + (a_1 b_2 - a_2 b_1)\mathbf{k}$$

        - e.g. $[1, 4, 1] \times [4, 2, 1] = [2, 3, -14]$
      - Applications
        - Relationship of angular velocity, angular momentum, and torque in a 3D space
        - Area of the parallelogram (평행사변형 in Korean) made by two vectors
        - More applications in computational geometry



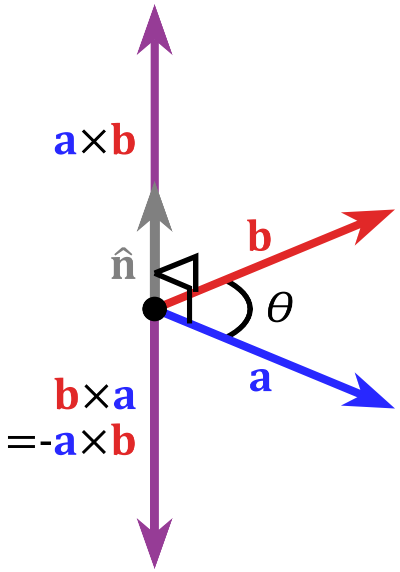


Image: Wikipedia

Vector

- **Vector operations** (cont'd)
  - Vector multiplication
    - **Cross product** (a.k.a. vector product) is defined only in a 3D space

      - Line representation: $ax + by + c = 0$

        $(2x + 3y - 14 = 0)$

      - Definition: $\mathbf{a} \times \mathbf{b} = \|\mathbf{a}\| \, \|\mathbf{b}\| \, \sin\theta \, \hat{\mathbf{n}}$

        where $\hat{\mathbf{n}}$ is a unit vector orthogonal (수직 in Korean) to a plane containing $\mathbf{a}$ and $\mathbf{b}$

      - Computing (using matrix determinant)

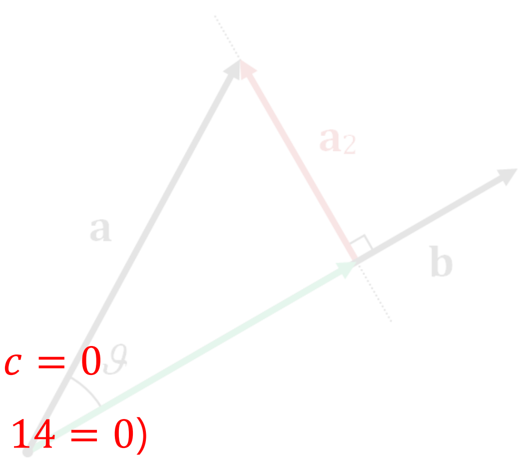
$$\mathbf{a} \times \mathbf{b} = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{vmatrix} = (a_2 b_3 - a_3 b_2)\mathbf{i} - (a_1 b_3 - a_3 b_1)\mathbf{j} + (a_1 b_2 - a_2 b_1)\mathbf{k}$$

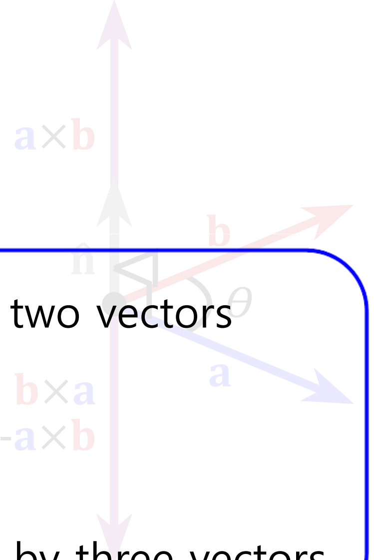
$(1, 4)$ $(4, 2)$

- e.g. $[1, 4, 1] \times [4, 2, 1] = [2, 3, -14]$

  - Applications
    - Relationship of angular velocity, angular momentum, ...
    - Area of the parallelogram (평행사변형 in Korean) made by two vectors
    - More applications in computational geometry

- Area of the parallelogram made by two vectors
- Line from two points
- Intersection point from two lines
- Distance between two *skew* lines
- Volume of the parallelepiped made by three vectors



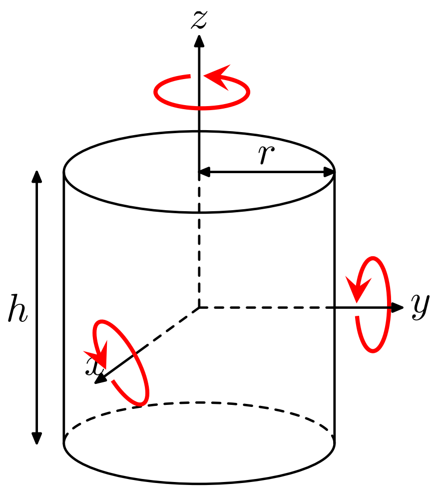


Matrix

- **Why a matrix?**
  - To represent physical quantities
    - e.g. Moment of inertia in the 3D space (a.k.a. inertia tensor)
  - To represent data and models
    - e.g. Images, multiple sound
    - e.g. State-space representation (control theory), camera projection
  - To represent geometric transformation
    - e.g. Rotation matrix $\mathrm{R}(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$
    - e.g. Rigid transformation, camera projection
  - To solve a system of linear equations (선형연립방정식 in Korean)
    - e.g. A system of equations for a line from (1, 4) and (4, 2)

$$4 = a \cdot 1 + b$$

$$2 = a \cdot 4 + b$$



$$I = \begin{bmatrix} \frac{1}{12}m(3r^2+h^2) & 0 & 0 \\ 0 & \frac{1}{12}m(3r^2+h^2) & 0 \\ 0 & 0 & \frac{1}{2}mr^2 \end{bmatrix}$$

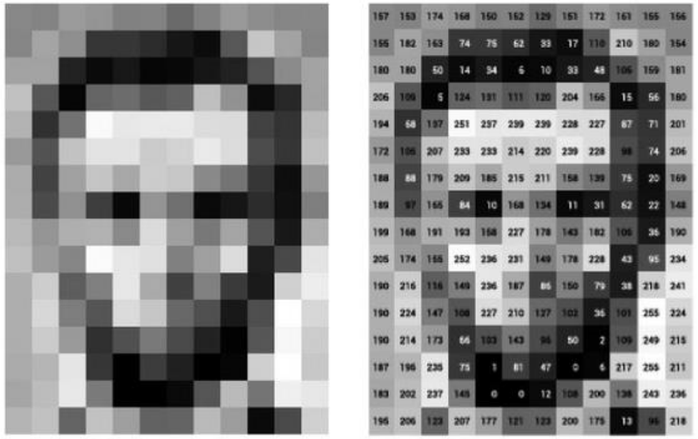


Image: Wikipedia and ResearchGate (a paper by Smits and Wevers).

Matrix

- **Matrix operations**
  - Matrix addition (subtraction): $\mathrm{A} + \mathrm{B}$
  - Scalar multiplication (division): $a\,\mathrm{B}$
  - Matrix multiplication (a.k.a. matrix product): AB
    - Not commutative (교환법칙 in Korean): $\mathrm{AB} \neq \mathrm{BA}$
      - Example) ~~Proof~~ Verification using SymPy

```
import sympy as sp

a11, a12, a21, a22 = sp.symbols('a11 a12 a21 a22')
b11, b12, b21, b22 = sp.symbols('b11 b12 b21 b22')
A = sp.Matrix([[a11, a12], [a21, a22]])
B = sp.Matrix([[b11, b12], [b21, b22]])

print(A+B == B+A) # True
print(A*B == B*A) # False
print(A*B - B*A)  # Check their difference
```

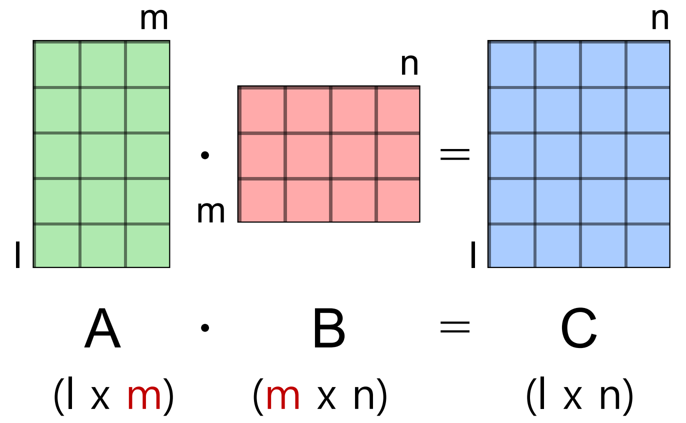


Image: Wikipedia

Matrix

- **Matrix operations** (cont'd)
  - **Matrix inverse** ($AA^{-1} = A^{-1}A = I$; ~ 역수 in Korean)
    - Invertible matrix (a.k.a. non-singular matrix, non-degenerate matrix) if A is
      - Square (A: $n \times n$ matrix)
      - Linearly independent rows/columns
        - Full rank ($\text{rank}(A) = n$)
        - Non-zero determinant ($\det(A) \neq 0$ or $|A| \neq 0$)
  - **Pseudo-inverse** (~ a generalized matrix inverse)
    - **Not necessarily square** (A: $m \times n$ matrix, $A^{\dagger}$: $n \times m$ matrix)
    - Left inverse ($A^{\dagger}A = I_n$): $A^{\dagger} = (A^{\intercal}A)^{-1}A^{\intercal}$
      - If A has linearly independent columns ($\text{rank}(A) = n$)
    - Right inverse ($AA^{\dagger} = I_m$): $A^{\dagger} = A^{\intercal}(AA^{\intercal})^{-1}$
      - If A has linearly independent rows ($\text{rank}(A) = m$)

Matrix

- **Matrix operations** (cont'd)
  - **Matrix inverse**
    - Example) Line fitting from two points, $(1, 4)$ and $(4, 2)$
      - Line representation: $y = ax + b$
      - A system of equations: $a \cdot 1 + b = 4$

$$a \cdot 4 + b = 2$$

$$\begin{bmatrix} 1 & 1 \\ 4 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} 4 \\ 2 \end{bmatrix} \longrightarrow \mathbf{x} = \mathrm{A}^{-1}\mathbf{b}$$

$$(\mathrm{A}\mathbf{x} = \mathbf{b})$$

  - **Pseudo-inverse**
    - Example) Line fitting from more than two points such as $(1, 4)$, $(4, 2)$, and $(7, 1)$
      - $\mathrm{A} = \begin{bmatrix} 1 & 1 \\ 4 & 1 \\ 7 & 1 \end{bmatrix}$ and $\mathbf{b} = \begin{bmatrix} 4 \\ 2 \\ 1 \end{bmatrix}$

NumPy: A Matrix Library in Python

- NumPy supports multi-dimensional arrays and matrices with their high-level mathematical functions.
  - It is fundamental for scientific and numerical computing in Python.
- References: Documentation, Tutorials/Books/Talks, and **Cheatsheet** (made by DataCamp)

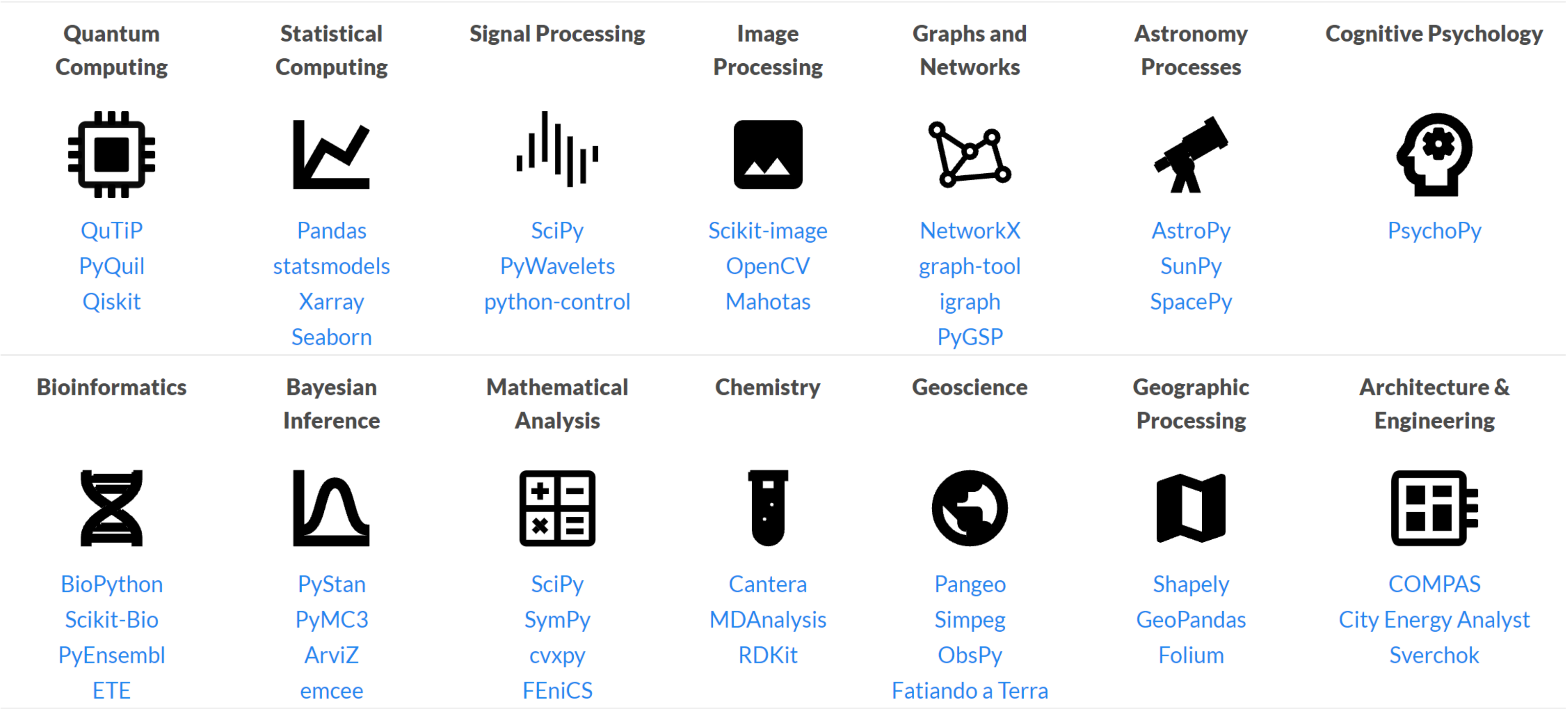


Image: NumPy (retrieved at October 14th, 2021)

NumPy: .

- NumPy
  - It is
- Referen

Python For Data Science *Cheat Sheet*

NumPy Basics

Learn Python for Data Science **Interactively** at www.DataCamp.com

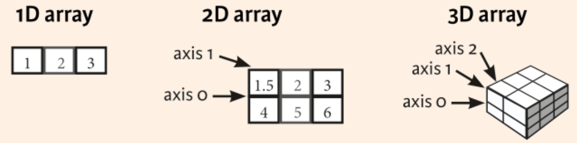
NumPy

The **NumPy** library is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.

Use the following import convention:

```
>>> import numpy as np
```

NumPy Arrays



Creating Arrays

```
>>> a = np.array([1,2,3])
>>> b = np.array([(1.5,2,3), (4,5,6)], dtype = float)
>>> c = np.array([[(1.5,2,3), (4,5,6)], [(3,2,1), (4,5,6)]],
                 dtype = float)
```

Initial Placeholders

`>>> np.zeros((3,4))`	Create an array of zeros
`>>> np.ones((2,3,4),dtype=np.int16)`	Create an array of ones
`>>> d = np.arange(10,25,5)`	Create an array of evenly spaced values (step value)
`>>> np.linspace(0,2,9)`	Create an array of evenly spaced values (number of samples)
`>>> e = np.full((2,2),7)`	Create a constant array
`>>> f = np.eye(2)`	Create a 2X2 identity matrix
`>>> np.random.random((2,2))`	Create an array with random values
`>>> np.empty((3,2))`	Create an empty array

I/O

Saving & Loading On Disk

```
>>> np.save('my_array', a)
>>> np.savez('array.npz', a, b)
>>> np.load('my_array.npy')
```

Saving & Loading Text Files

```
>>> np.loadtxt("myfile.txt")
>>> np.genfromtxt("my_file.csv", delimiter=',')
>>> np.savetxt("myarray.txt", a, delimiter=" ")
```

Data Types

`>>> np.int64`	Signed 64-bit integer types
`>>> np.float32`	Standard double-precision floating point
`>>> np.complex`	Complex numbers represented by 128 floats
`>>> np.bool`	Boolean type storing `TRUE` and `FALSE` values
`>>> np.object`	Python object type
`>>> np.string_`	Fixed-length string type
`>>> np.unicode_`	Fixed-length unicode type

Inspecting Your Array

`>>> a.shape`	Array dimensions
`>>> len(a)`	Length of array
`>>> b.ndim`	Number of array dimensions
`>>> e.size`	Number of array elements
`>>> b.dtype`	Data type of array elements
`>>> b.dtype.name`	Name of data type
`>>> b.astype(int)`	Convert an array to a different type

Asking For Help

```
>>> np.info(np.ndarray.dtype)
```

Array Mathematics

Arithmetic Operations

`>>> g = a - b`
`array([[-0.5, 0. , 0. ],`
`[-3. , -3. , -3. ]])`
	Subtraction

`>>> np.subtract(a,b)`	Subtraction

`>>> b + a`
`array([[ 2.5, 4. , 6. ],`
`[ 5. , 7. , 9. ]])`
	Addition

`>>> np.add(b,a)`	Addition

`>>> a / b`
`array([[ 0.66666667, 1. , 1. ],`
`[ 0.25 , 0.4 , 0.5 ]])`
	Division

`>>> np.divide(a,b)`	Division

`>>> a * b`
`array([[ 1.5, 4. , 9. ],`
`[ 4. , 10. , 18. ]])`
	Multiplication

`>>> np.multiply(a,b)`	Multiplication
`>>> np.exp(b)`	Exponentiation
`>>> np.sqrt(b)`	Square root
`>>> np.sin(a)`	Print sines of an array
`>>> np.cos(b)`	Element-wise cosine
`>>> np.log(a)`	Element-wise natural logarithm

`>>> e.dot(f)`
`array([[ 7., 7.],`
`[ 7., 7.]])`
	Dot product

Comparison

`>>> a == b`
`array([[False, True, True],`
`[False, False, False]], dtype=bool)`
	Element-wise comparison

`>>> a < 2` `array([True, False, False], dtype=bool)`	Element-wise comparison
`>>> np.array_equal(a, b)`	Array-wise comparison

Aggregate Functions

`>>> a.sum()`	Array-wise sum
`>>> a.min()`	Array-wise minimum value
`>>> b.max(axis=0)`	Maximum value of an array row
`>>> b.cumsum(axis=1)`	Cumulative sum of the elements
`>>> a.mean()`	Mean
`>>> b.median()`	Median
`>>> a.corrcoef()`	Correlation coefficient
`>>> np.std(b)`	Standard deviation

Copying Arrays

`>>> h = a.view()`	Create a view of the array with the same data
`>>> np.copy(a)`	Create a copy of the array
`>>> h = a.copy()`	Create a deep copy of the array

Sorting Arrays

`>>> a.sort()`	Sort an array
`>>> c.sort(axis=0)`	Sort the elements of an array's axis

Subsetting, Slicing, Indexing

Also see Lists

Subsetting

`>>> a[2]` `3`	Select the element at the 2nd index
`>>> b[1,2]` `6.0`	Select the element at row 1 column 2 (equivalent to `b[1][2]`)

Slicing

`>>> a[0:2]` `array([1, 2])`	Select items at index 0 and 1
`>>> b[0:2,1]` `array([ 2., 5.])`	Select items at rows 0 and 1 in column 1
`>>> b[:1]` `array([[1.5, 2., 3.]])`	Select all items at row 0 (equivalent to `b[0:1, :]`)

`>>> c[1,...]`
`array([[[ 3., 2., 1.],`
`[ 4., 5., 6.]]])`
	Same as `[1,:,:]`

`>>> a[ : :-1]` `array([3, 2, 1])`	Reversed array `a`

Boolean Indexing

`>>> a[a<2]` `array([1])`	Select elements from `a` less than 2

Fancy Indexing

`>>> b[[1, 0, 1, 0],[0, 1, 2, 0]]` `array([ 4. , 2. , 6. , 1.5])`	Select elements `(1,0)`,`(0,1)`,`(1,2)` and `(0,0)`

`>>> b[[1, 0, 1, 0]][:,[0,1,2,0]]`
`array([[ 4. ,5. , 6. , 4. ],`
`[ 1.5, 2. , 3. , 1.5],`
`[ 4. , 5. , 6. , 4. ],`
`[ 1.5, 2. , 3. , 1.5]])`
	Select a subset of the matrix's rows and columns

Array Manipulation

Transposing Array

`>>> i = np.transpose(b)`	Permute array dimensions
`>>> i.T`	Permute array dimensions

Changing Array Shape

`>>> b.ravel()`	Flatten the array
`>>> g.reshape(3,-2)`	Reshape, but don't change data

Adding/Removing Elements

`>>> h.resize((2,6))`	Return a new array with shape (2,6)
`>>> np.append(h,g)`	Append items to an array
`>>> np.insert(a, 1, 5)`	Insert items in an array
`>>> np.delete(a,[1])`	Delete items from an array

Combining Arrays

`>>> np.concatenate((a,d),axis=0)` `array([ 1, 2, 3, 10, 15, 20])`	Concatenate arrays

`>>> np.vstack((a,b))`
`array([[ 1. , 2. , 3. ],`
`[ 1.5, 2. , 3. ],`
`[ 4. , 5. , 6. ]])`
	Stack arrays vertically (row-wise)

`>>> np.r_[e,f]`	Stack arrays vertically (row-wise)

`>>> np.hstack((e,f))`
`array([[ 7., 7., 1., 0.],`
`[ 7., 7., 0., 1.]])`
	Stack arrays horizontally (column-wise)

`>>> np.column_stack((a,d))`
`array([[ 1, 10],`
`[ 2, 15],`
`[ 3, 20]])`
	Create stacked column-wise arrays

`>>> np.c_[a,d]`	Create stacked column-wise arrays

Splitting Arrays

`>>> np.hsplit(a,3)` `[array([1]),array([2]),array([3])]`	Split the array horizontally at the 3rd index

`>>> np.vsplit(c,2)`
`[array([[[ 1.5, 2. , 1. ],`
`[ 4. , 5. , 6. ]]]),`
`array([[[ 3., 2., 3.],`
`[ 4., 5., 6.]]])]`
	Split the array vertically at the 2nd index

DataCamp
Learn Python for Data Science **Interactively**

Image: DataCamp

NumPy: A Matrix Library in Python

- Usage example) Creating n-dimensional arrays (1/2)
  - `numpy.array` can contain a **homogenous** (~ same) data type. (vs. `list` and `tuple`)

```
import numpy as np

# 1. Create an array from a composite data (list or tuple, not set and dictionary)
A = np.array([3, 29, 82])
B = np.array(((3., 29, 82), (10, 18, 84)))
C = np.array([[3, 29, 82], [10, 18, 84]], dtype=float)
D = np.array([3, 29, 'Choi'])
E = np.array([[3], [29], [82]])
print(A.ndim, A.size, A.shape, A.dtype) # 1 3 (3,) int32
                                        # Note) np.array_equal(A, A.T) == True
print(B.ndim, B.size, B.shape, B.dtype) # 2 6 (2, 3) float64
print(C.ndim, C.size, C.shape, C.dtype) # 2 6 (2, 3) float64
print(D.ndim, D.size, D.shape, D.dtype) # 1 3 (3,) <U11 (Unicode)
                                        # Note) array(['3', '29', 'Choi'])
print(E.ndim, E.size, E.shape, E.dtype) # 2 3 (3,1) int32
                                        # Note) np.array_equal(E, E.T) == False
                                        #       because E.T == array([[3, 29, 82]])
```

NumPy: A Matrix Library in Python

- Usage example) Creating n-dimensional arrays (2/2)
  - `numpy.array` can contain a **homogenous** (~ same) data type. (vs. `list` and `tuple`)

```
import numpy as np

# 2. Create an array using initializers
F = np.zeros((3, 2))                # Create a 3x2 array filled with 0 (default: float64)
G = np.ones((3, 2))                 # Create a 2x3 array filled with 1
H = np.eye(3, dtype=np.float32) # Create a 3x3 identity matrix (single-precision)
I = np.empty((3, 2))                # == np.zeros((3, 2))
J = np.empty((0, 9))                # [ ] with size of (0, 9)
K = np.arange(0, 1, 0.2)            # Step 0.2: array([0., 0.2, 0.4, 0.6, 0.8])
L = np.linspace(0, 1, 5)            # Number 5: array([0., 0.25, 0.50, 0.75, 1.])
M = np.random.random((3, 2))        # == np.random.uniform(size=(3, 2))
                                    # Note) np.random.normal()
```

NumPy: A Matrix Library in Python

- Usage example) Indexing and slicing
  - `numpy.array` can access elements of a n-dim array not only with `[r][c]` but also with **`[r,c]`**.
  - `numpy.array` can access elements with a Boolean array (called **logical indexing**).

	0	1	2
0	3	29	82
1	10	18	84

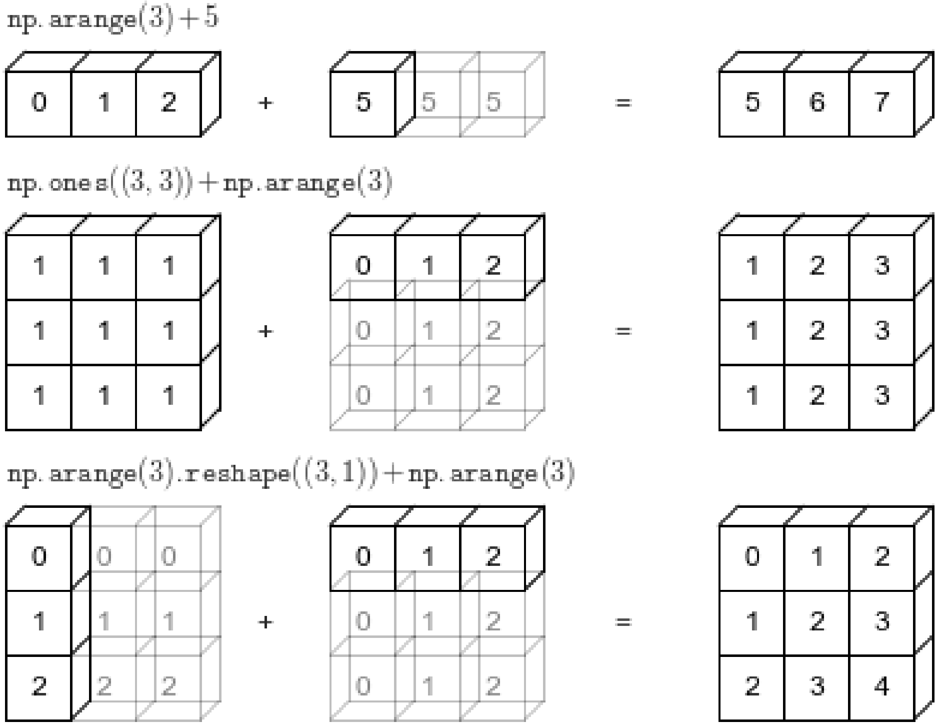
```python
import numpy as np
A = np.array(((3., 29, 82), (10, 18, 84)))

# 1. Indexing and slicing
A[1][1]                 # 18.0
A[1, 1]                 # 18.0  Note) list[1, 1] does not work!
A[1,1:2]                # array([18.])
A[1,:]                  # Get a row:       array([10., 18., 84.])
A[:,2]                  # Get a column:    array([82., 84.])
A[0:2,0:2]              # Get a submatrix: array([[3., 29.], [10., 18.]])

# 2. Logical indexing
A > 80                  # array([[False, False, True], [False, False, True]])
A[A > 80]               # array([82., 84.])
A[A > 80] = 80          # Masked operations are possible!

# 3. Fancy indexing
A[(1, 0, 0), (1, 0, 2)] # array([18., 3., 82.])
                        # Get items at (1, 1), (0, 0), (0, 2)
```

Image: Jake VanderPlas's book

NumPy: A Matrix Library in Python

- Usage example) Arithmetic operations (vs. `list` and `tuple`)

```
import numpy as np
A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])

# 1. Element-wise arithmetic operations
print(A + B)                        # np.add(A, B)
print(A - B)                        # np.subtract(A, B)
print(A * B)                        # np.multiply(A, B)
print(A / B)                        # np.divide(A, B)

# 2. Matrix operations
print(A.T)                          # A.transpose()
print(A @ B)                        # np.matmul(A, B)
print(np.linalg.norm(A))            # 5.48 (default: L2-norm)
print(np.linalg.matrix_rank(A))     #  2, full rank
print(np.linalg.det(A))             # -2, non-zero determinant
print(np.linalg.inv(A))             # Matrix inverse
print(np.linalg.pinv(A))            # Matrix pseudo-inverse

# 3. Broadcasting
print(A + 1)                        # [[2, 3], [4, 5]]
print(A + [0, -1])                  # [[1, 1], [3, 3]]
print(A + [[1], [-1]])              # [[2, 3], [2, 3]]
```



Matrix

- Example) Line fitting from two points, $(1, 4)$ and $(4, 2)$
  - Line representation: $y = ax + b$
  - A system of equations: $a \cdot 1 + b = 4$

$$a \cdot 4 + b = 2$$

$$\downarrow$$

$$\begin{bmatrix} 1 & 1 \\ 4 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} 4 \\ 2 \end{bmatrix} \longrightarrow \mathbf{x} = \mathrm{A}^{-1}\mathbf{b}$$

$$(\mathrm{A}\mathbf{x} = \mathbf{b})$$

- Example) Line fitting from two points, $(1, 4)$ and $(4, 2)$, using NumPy

```
import numpy as np

A = np.array([[1., 1.], [4., 1.]])
b = np.array([[4.], [2.]])
A_inv = np.linalg.inv(A)
print(A_inv * b) # [[-1.33333333  1.33333333]
                 #  [ 2.66666667 -0.66666667]]  Note) broadcast
print(A_inv @ b) # [[-0.66666667]
                 #  [ 4.66666667]]
```

Matrix

- Example) Line fitting from two points, $(1, 4)$ and $(4, 2)$, using NumPy

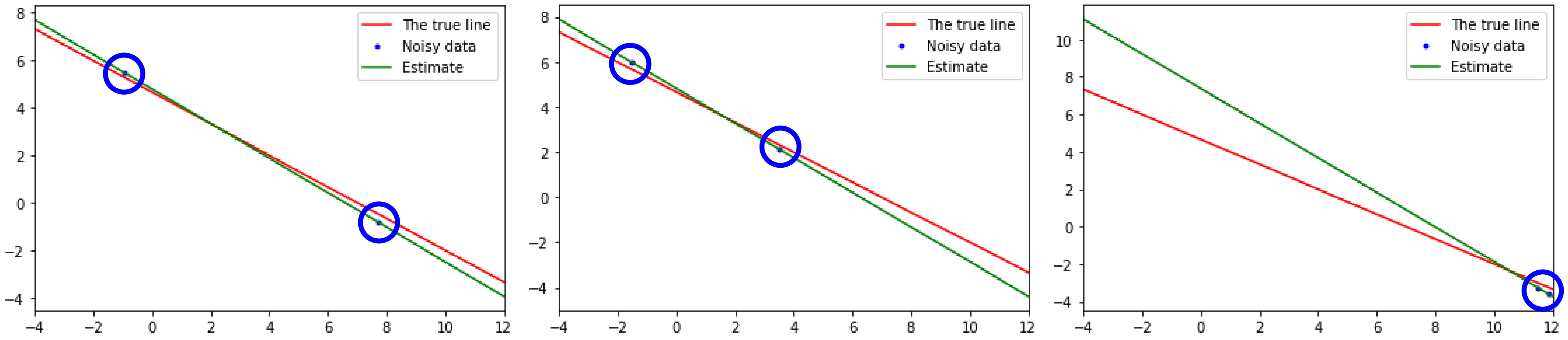
```python
import numpy as np

A = np.array([[1., 1.], [4., 1.]])
b = np.array([[4.], [2.]])
A_inv = np.linalg.inv(A)
print(A_inv @ b) # [[-0.66666667]
                 #  [ 4.66666667]]
```

- Example) Line fitting from more than two points such as $(1, 4)$, $(4, 2)$, and $(7, 1)$ using NumPy
  - $\mathrm{A} = \begin{bmatrix} 1 & 1 \\ 4 & 1 \\ 7 & 1 \end{bmatrix}$ and $\mathbf{b} = \begin{bmatrix} 4 \\ 2 \\ 1 \end{bmatrix}$

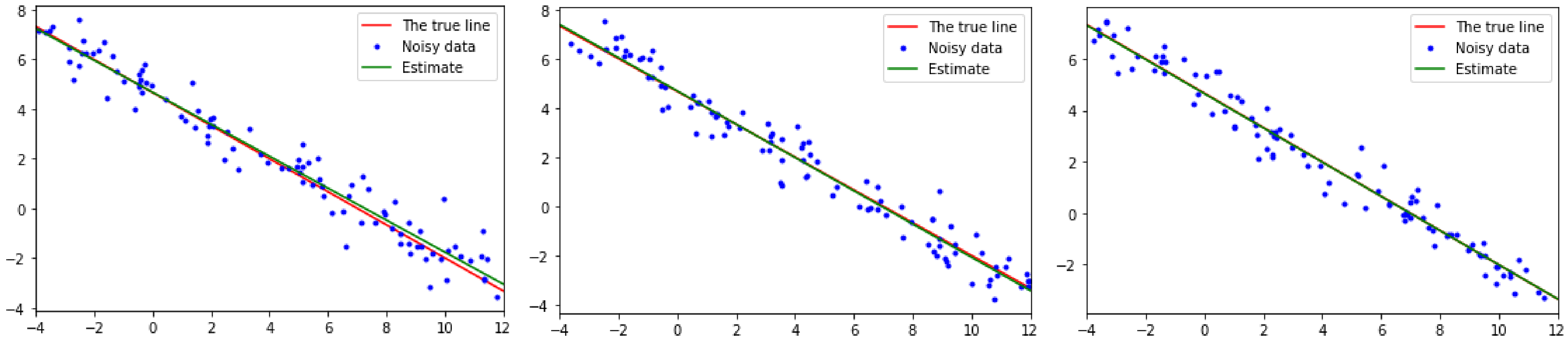
```python
import numpy as np

A = np.array([[1., 1.], [4., 1.], [7., 1.]])
b = np.array([[4.], [2.], [1.]])
A_inv = np.linalg.pinv(A) # Left inverse
print(A_inv @ b) # [[-0.5]
                 #  [ 4.33333333]]
```

Matrix

- Example) Line fitting with noisy but more data (1/2)
  - `data_num = 2`



  - `data_num = 100`



Matrix

- Example) Line fitting with noisy but more data (2/2)

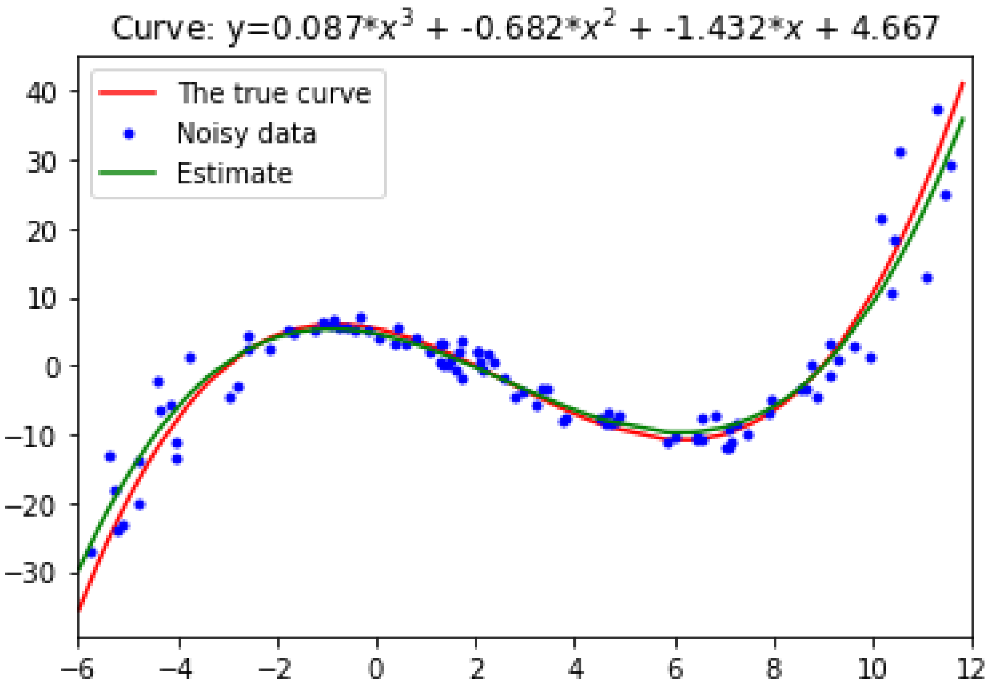
```
import numpy as np
import matplotlib.pyplot as plt

true_line = lambda x: -2/3*x + 14/3
data_range = np.array([-4, 12])
data_num = 100
noise_std = 0.5

# Generate the true data
x = np.random.uniform(data_range[0], data_range[1], size=data_num)
y = true_line(x) # y = -2/3*x + 10/3

# Add Gaussian noise
xn = x + np.random.normal(scale=noise_std, size=x.shape)
yn = y + np.random.normal(scale=noise_std, size=y.shape)

# Solve the system of equations
A = np.vstack((xn, np.ones(xn.shape))).T
b = yn
line = np.linalg.pinv(A) @ b

# Plot the data and result
plt.title(f'Line: y={line[0]:.3f}*x + {line[1]:.3f} ')
plt.plot(data_range, true_line(data_range), 'r-', label='The true line')
plt.plot(xn, yn, 'b.', label='Noisy data')
plt.plot(data_range, line[0]*data_range + line[1], 'g-', label='Estimate')
plt.xlim(data_range)
plt.legend()
plt.show()
```

$$\begin{bmatrix} x_1 & x_2 & \cdots & x_n \\ 1 & 1 & \cdots & 1 \end{bmatrix} \xrightarrow{\text{transpose}} \begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

$(\mathbf{A}\mathbf{x} = \mathbf{b})$

Matrix

- Example) Curve fitting (1/2)
  - Curve representation: $y = ax^3 + bx^2 + cx + d$ (a 3rd-order polynomial equation)
    - e.g. $y = 0.1x^3 - 0.8x^2 - 1.5x + 5.4$



$$\begin{bmatrix} x_1^3 & x_1^2 & x_1 & 1 \\ x_2^3 & x_2^2 & x_2 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ x_n^3 & x_n^2 & x_n & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

$(\mathrm{A}\mathbf{x} = \mathbf{b})$

Matrix

- Example) Curve fitting (2/2)

```python
import numpy as np
import matplotlib.pyplot as plt

true_curve = lambda x: 0.1*x**3 - 0.8*x**2 - 1.5*x + 5.4
data_range = (-6, 12)
data_num = 100
noise_std = 0.5

# Generate the true data
x = np.random.uniform(data_range[0], data_range[1], size=data_num)
y = true_curve(x)

# Add Gaussian noise
xn = x + np.random.normal(scale=noise_std, size=x.shape)
yn = y + np.random.normal(scale=noise_std, size=y.shape)

# Solve the system of equations
A = np.vstack((xn**3, xn**2, xn, np.ones(xn.shape))).T
b = yn
curve = np.linalg.pinv(A) @ b

# Plot the data and result
plt.title(f'Curve: y={curve[0]:.3f}*$x^3$ + {curve[1]:.3f}*$x^2$ + {curve[2]:.3f}*$x$ + {curve[3]:.3f}')
xc = np.linspace(*data_range, 100)
plt.plot(xc, true_curve(xc), 'r-', label='The true curve')
plt.plot(xn, yn, 'b.', label='Noisy data')
plt.plot(xc, curve[0]*xc**3 + curve[1]*xc**2 + curve[2]*xc + curve[3], 'g-', label='Estimate')
plt.xlim(data_range)
plt.legend()
plt.show()
```

$$\begin{bmatrix} x_1^3 & x_1^2 & x_1 & 1 \\ x_2^3 & x_2^2 & x_2 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ x_n^3 & x_n^2 & x_n & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

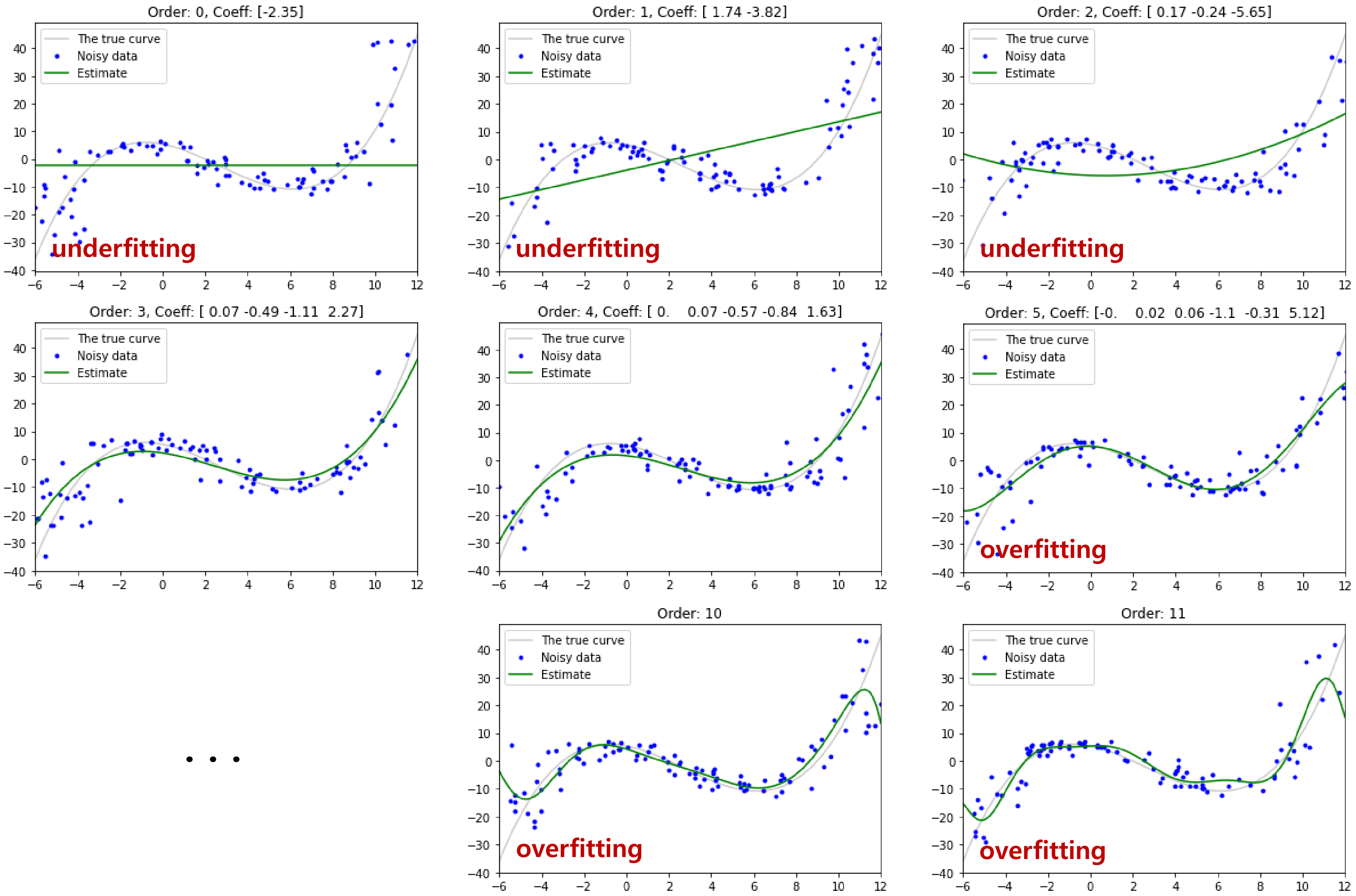
$$(\mathbf{A}\mathbf{x} = \mathbf{b})$$

Matrix

- Example) Curve fitting with model selection (1/2)

$$\begin{bmatrix} x_1^r & x_1^{r-1} & \cdots & x_1^2 & x_1 & 1 \\ x_2^r & x_2^{r-1} & \cdots & x_2^2 & x_2 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_n^r & x_n^{r-1} & \cdots & x_n^2 & x_n & 1 \end{bmatrix} \begin{bmatrix} c_r \\ c_{r-1} \\ \vdots \\ c_2 \\ c_1 \\ c_0 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

$(A\mathbf{x} = \mathbf{b})$



```python
import numpy as np
import matplotlib.pyplot as plt

def buildA(order, xs):
    A = np.empty((0, len(xs)))
    for i in range(order + 1):
        A = np.vstack((xs**i, A))
    return A.T

true_coeff = [0.1, -0.8, -1.5, 5.4]
poly_order = 3 # Try other integer (>= 0)
data_range = (-6, 12)
data_num = 100
noise_std = 1

# Generate the true data
x = np.random.uniform(data_range[0], data_range[1], size=data_num)
y = buildA(len(true_coeff) - 1, x) @ true_coeff

# Add Gaussian noise
xn = x + np.random.normal(scale=noise_std, size=x.shape)
yn = y + np.random.normal(scale=noise_std, size=y.shape)

# Solve the system of equations
A = buildA(poly_order, xn)
b = yn
coeff = np.linalg.pinv(A) @ b

# Plot the data and result
plt.title(f'Order: {poly_order}, Coeff: ' + np.array2string(coeff, precision=2, suppress_small=True))
xc = np.linspace(*data_range, 100)
plt.plot(xc, np.matmul(buildA(len(true_coeff) - 1, xc), true_coeff), 'k-', label='The true curve', alpha=0.2)
plt.plot(xn, yn, 'b.', label='Noisy data')
plt.plot(xc, np.matmul(buildA(poly_order, xc), coeff), 'g-', label='Estimate')
plt.xlim(data_range)
plt.legend()
plt.show()
```

$$\begin{bmatrix} x_1^r & x_1^{r-1} & \cdots & x_1^2 & x_1 & 1 \\ x_2^r & x_2^{r-1} & \cdots & x_2^2 & x_2 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_n^r & x_n^{r-1} & \cdots & x_n^2 & x_n & 1 \end{bmatrix} \begin{bmatrix} c_r \\ c_{r-1} \\ \vdots \\ c_2 \\ c_1 \\ c_0 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

$$(A\mathbf{x} = \mathbf{b})$$

Matrix

- Example) Line fitting from two points, $(1, 4)$ and $(1, 2)$
  - Line representation: $y = ax + b$
  - A system of equations: $a \cdot 1 + b = 4$

    $a \cdot 1 + b = 2$

$$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} 4 \\ 2 \end{bmatrix} \longrightarrow \mathbf{x} = \mathrm{A}^{-1}\mathbf{b}$$

$$(\mathrm{A}\mathbf{x} = \mathbf{b})$$

- Example) Line fitting from two points, $(1, 4)$ and $(1, 2)$, using NumPy

```
import numpy as np

A = np.array([[1., 1.], [1., 1.]])
b = np.array([[4.], [2.]])
A_inv = np.linalg.inv(A) # Error! (singular matrix)
print(A_inv @ b)
```

Matrix

- **Null space** (a.k.a. kernel)
  - A set of vectors which map A (m-by-n matrix) to the zero vector

    $N(\mathrm{A}) = \{ \mathbf{v} \in K^n \mid \mathrm{A}\mathbf{v} = \mathbf{0} \}$
  - Rank-nullity theorem: $\mathrm{rank}(\mathrm{A}) + \mathrm{nullity}(\mathrm{A}) = n$
  - Application) Solving a **homogenous** system of linear equations, $\mathrm{A}\mathbf{x} = \mathbf{0}$

- Practice) Line fitting from two points, $(1, 4)$ and $(1, 2)$, using NumPy
  - Line representation: $ax + by + c = 0$
  - In a matrix form, $\begin{bmatrix} 1 & 4 & 1 \\ 1 & 2 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

How about $(1, 4)$ and $(4, 2)$?

```python
import numpy as np
from scipy import linalg

A = np.array([[1., 4., 1.], [1., 2., 1.]])
x = linalg.null_space(A)
print(x / x[0]) # [[1.], [0.], [-1.]]  Note) Line: x - 1 = 0
```

Summary

- NumPy: N-dimensional array representation and numerical analysis (수치해석 in Korean)
  - `numpy.array` vs. `list/tuple`
    - \- Homogeneous data type
    - \+ Indexing/slicing and arithmetic operations
- ~~Linear algebra~~ **Vector**
  - Why? 1) To represent physical quantities / 2) To represent data and models
  - Vector operations
    - Note) Norm (~ the distance from the origin, magnitude)
    - Vector multiplication
      - Dot product: $\mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\| \, \|\mathbf{b}\| \, \cos\theta$
        - Applications: Cosine similarity
      - Cross product: $\mathbf{a} \times \mathbf{b} = \|\mathbf{a}\| \, \|\mathbf{b}\| \, \sin\theta \, \hat{\mathbf{n}}$ ($\hat{\mathbf{n}}$: the unit orthogonal vector of $\mathbf{a}$ and $\mathbf{b}$) defined in 3D spaces
        - Applications: Physics in a 3D space, …, computational geometry

Summary

- ~~Linear algebra~~ **Matrix**
  - Why? + 3) To represent geometric transformation / 4) **To solve a system of linear equations**
  - Matrix operations
    - Matrix multiplication (not commutative)
    - Matrix inverse (square + full rank), pseudo-inverse
  - Examples) **Line and curve fitting** (regression analysis; 회귀분석 in Korean)
    - What is a model and its parameters?
      - e.g. Line: $y = ax + b$ (or $ax + by + c = 0$)
      - e.g. Curve: nth-order polynomial equations (e.g. $y = ax^3 + bx^2 + cx + d$)
    - How to formulate the problem using matrices: $\mathrm{A}\mathbf{x} = \mathbf{b}$ (or $\mathrm{A}\mathbf{x} = \mathbf{0}$)
    - How to solve the equations using matrix operations: Pseudo-inverse (or finding null vectors)
    - Lesson #1) **More (over-constrained) data is beneficial.**
      - The estimation result from *more data* becomes *more robust* (~ strong) against noise.
    - Lesson #2) **Model complexity can lead underfitting and overfitting.**
      - *Too complex (flexible) models* can be too sensitive to noise, which can cause *overfitting*.