29회 투과전자현미경 워크샵

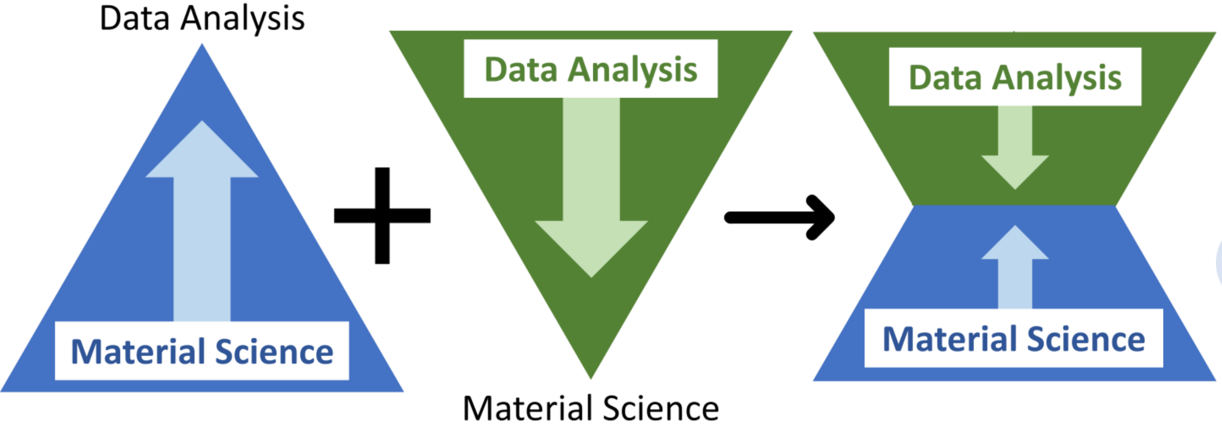
데이터 처리 기술 실습

2024.07.25 (목)

서울대학교 (김미영교수님 연구실)

유인규

Python Programming in TEM analysis



npj | Computational Materials

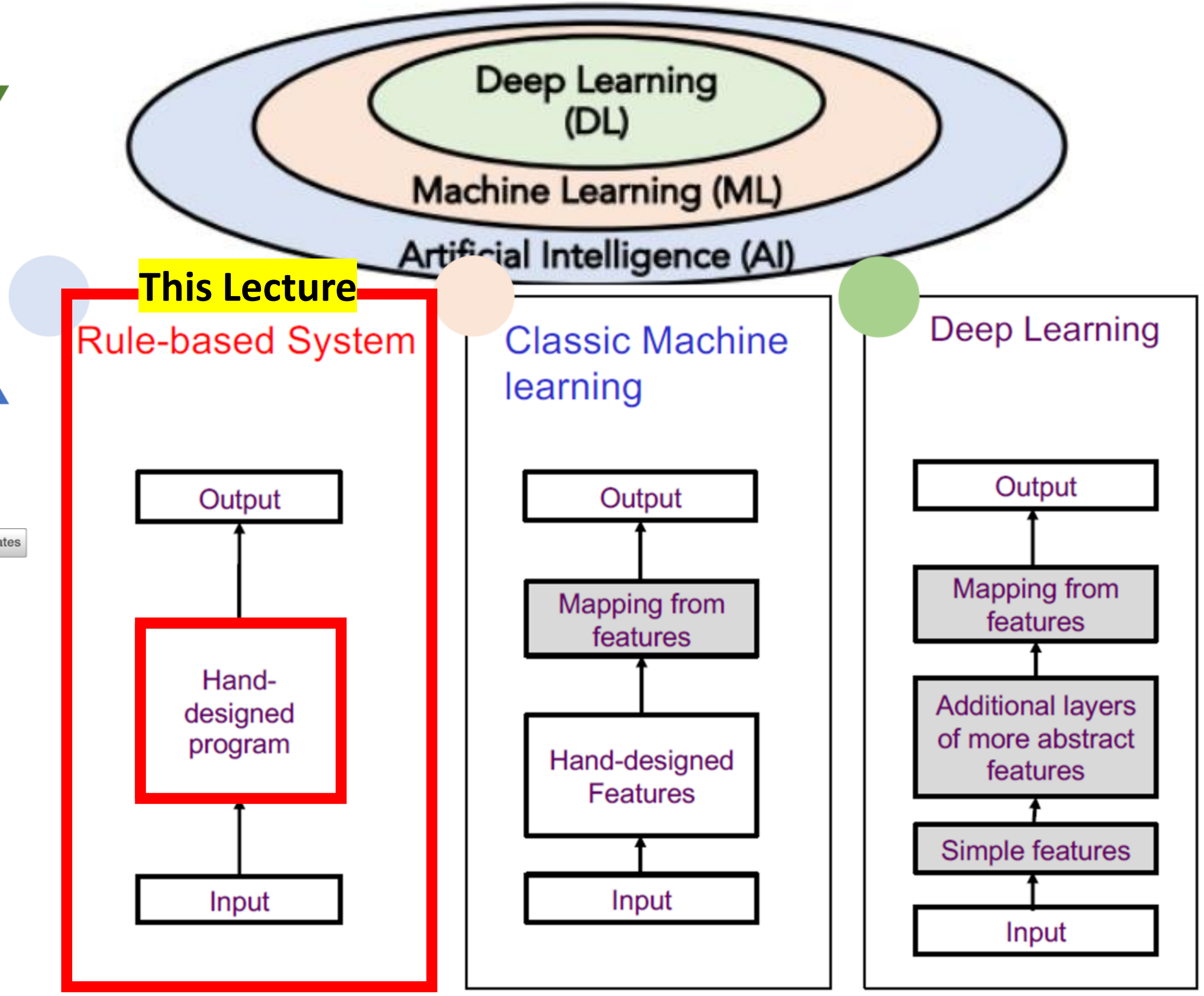
PERSPECTIVE OPEN

Check for updates

Off-the-shelf deep learning is not enough, and requires parsimony, Bayesianity, and causality

Rama K. Vasudevan[1], Maxim Ziatdinov[2], Lukas Vlcek[3,4] and Sergei V. Kalinin[1]

problems presented in those areas. In our opinion, the ①integration of human domain expertise and causal inference with deep learning will be the crucial link to correctly harnessing and exploiting the benefits that DL and ML can provide. Most importantly, the ②merger of machine learning with classical hypothesis driven science can bring ML beyond the current correlative paradigms into larger fields of Bayesian and causal learning and establish connections to the materials world via automated experiment[30–40] and open instrumental facilities, thus giving rise to fundamentally different ways of scientific research.

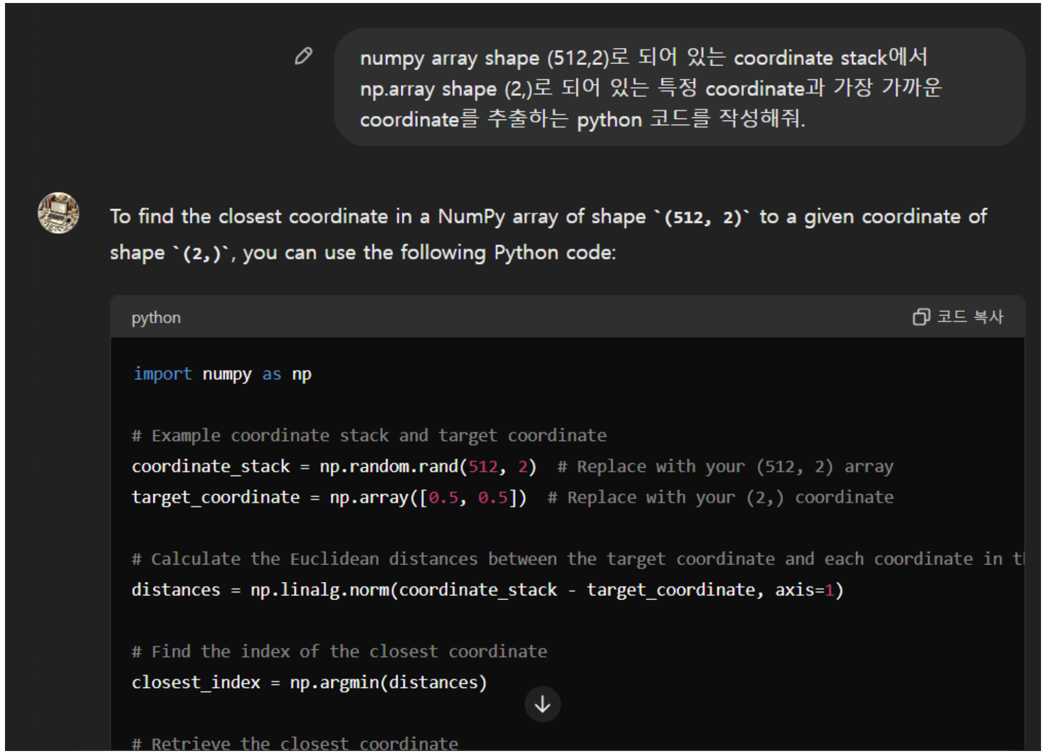


① Integration of **Human Domain Expertise** with Data Analysis

② Merger of Data Analysis with **Classical Hypothesis Driven Science**

Difficulties in analyzing TEM data via Python

- Necessity of understanding basic code for finding suitable open-source code (ex. ChatGPT generated code)



1) Resolve the reliability issues of Generative AI
2) Ask the right questions

- Flexibly resolve compatibility issues between existing open-source code and experimental data.

1) Data formats (e.g., .raw, .tiff, .emd, .h5py, .dat)
2) Version mismatch (e.g., documentation version)
3) Incorrect code usage (function, class)
4) Writing additional code to further develop the source code's functionality.

Content

PART 1

Basic Python structure for TEM data analysis using Python

1. List datatype
2. Numpy array datatype
3. Plot spectrum and image
4. For, If, While Statement
5. Function
6. Class

Targeted at Python beginners

PART 2

Applying the Python structure learned above to HAADF image analysis

7. Load HAADF image
8. Post-Processing HAADF image
9. Apply Gaussian fitting to HAADF image
10. Get atomic column coordinates

Targeted at Experienced Python Users

Before starting the code..

실습 파일 열기

<u>**준비사항 : 1) 와이파이 연결   2) 구글 계정 로그인**</u>

1. https://colab.research.google.com/ 접속

2. 노트 열기 창에서 GitHub Tab 클릭

3. GitHub URL 입력칸에 yig0222 입력

4. 경로에 나타난 TEMworkshop_demonstration.ipynb 파일 클릭

5. 파일 탭 Drive에 사본 저장 클릭하여 사본 저장

6. 새롭게 뜬 TEMworkshop_demonstration.ipynb의 사본 이름의 탭에서 작업.

7. 첫번째 git clone 셀 실행 (실행은 Ctrl+Enter)

8. 이후 순차적으로 셀 실행

Colab 단축키

- **Esc and Enter** : Cell 진입하고 나오기
- **Ctrl + 방향키** : 단어별로 커서 이동
- **Shift + 방향키** : 블록 설정하며 이동
- **Alt + 좌클릭** : 여러 개의 커서 설정
- **a (above), b (below)** : 새로운 셀 추가
- **Ctrl+ enter** : 해당 cell의 코드 실행
- **Shift + enter** : 해당 cell 코드 실행 후 다음 cell로 이동

Datatype (List, Array) : List

List : Storage to store everything (string, numbers, list, array, variable etc..)

Ex1. ["a", "b", "c"] Store Strings

"a"	"b"	"c"

Ex2. [1,2,3] Store Numbers

1	2	3

Ex3. [[1,2,3], [4,5,6], [7,8,9]] Store Lists

1	2	3	4	5	6	7	8	9

Ex4. Can store different data types in one

"a"	2	1	2	3

Create, Append, Length

```
#Create list
tmp_list= []
tmp_list.append("a")
tmp_list.append("b")
tmp_list.append("c")

print(tmp_list)
print(len(tmp_list))
```


= [ ]

"a"	"b"	"c"

Indexing in List (ex. ["a", "b", "c"])

```
#Index
print("Index 0: ", tmp_list[0])
print("Index 1: ", tmp_list[1])
print("Index 2: ", tmp_list[2])
print("Index 0 to 1:", tmp_list[0:2])
print("application:", tmp_list[0:2][1])
```

"a"	"b"	"c"
0	1	2

Application : List in List

```
#Application
tmp_list2 = []
tmp_list2.append(tmp_list)
tmp_list2.append(tmp_list)
print(len(tmp_list2))
print(tmp_list2[0])
print(tmp_list2[0][0:2][1])
```

"a"	"b"	"c"	"a"	"b"	"c"

Datatype (List, Array) : Array

Package

matplotlib

Module

Matplotlib.
pyplot

Import

```python
#Import Numpy and Pyplot Module
import numpy as np
import matplotlib.pyplot as plt
```

- 원하는 이름으로 import (import ~ as ~)
- Import numpy to operate numpy array
- Import pyplot to plot and show image

```python
from scipy.optimize import curve_fit #
```

```python
from numpy import *
```

- from 구문으로 원하는 것만 import (from ~ import ~)
- from 에 있는 모든 것들을 import (from ~ import *)

Usage of Import Statement

```python
import matplotlib.pyplot as plt

plt.figure(figsize=(6,6))
plt.imshow(tmp_img, cmap="inferno")
plt.title("Image")
plt.colorbar()
plt.axis("off")
plt.show()
```

```python
import numpy as np

arr1 = np.zeros((128,128), dtype="int8")
print("Arr1 : ", np.min(arr1),"to", np.max(arr1))
```

```python
from numpy import *
tmp = zeros((128,128))
print(tmp.shape)
```

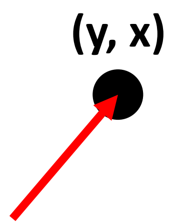
```python
from scipy.optimize import curve_fit

popt, pcov = curve_fit(gaussian_2d, np.array([x, y]), data, p0=initial_guess)
amplitude, center_x, center_y, sigma = popt
```

Datatype (List, Array) : Array

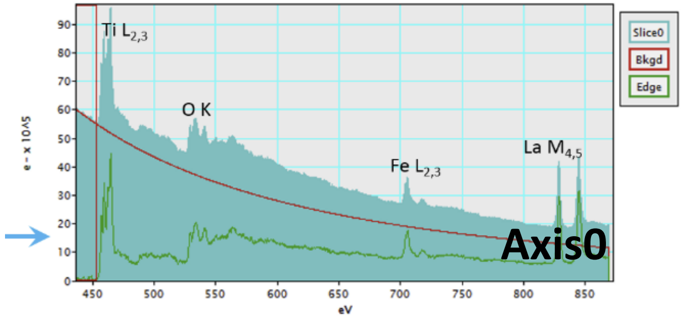
Usage of Numpy array

```
"""Usage of numpy array"""
vector = np.array([2,3])
spectrum = np.array([1,2,3,4,5,6,7,8,9])
image = np.array([[1,2,3],[4,5,6],[7,8,9]])
```

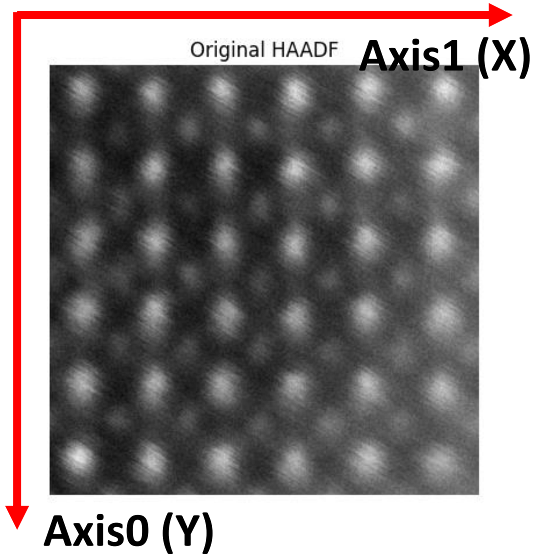
- Coordinate (1-dim)



- Spectrum (EELS, EDS, 1-dim)

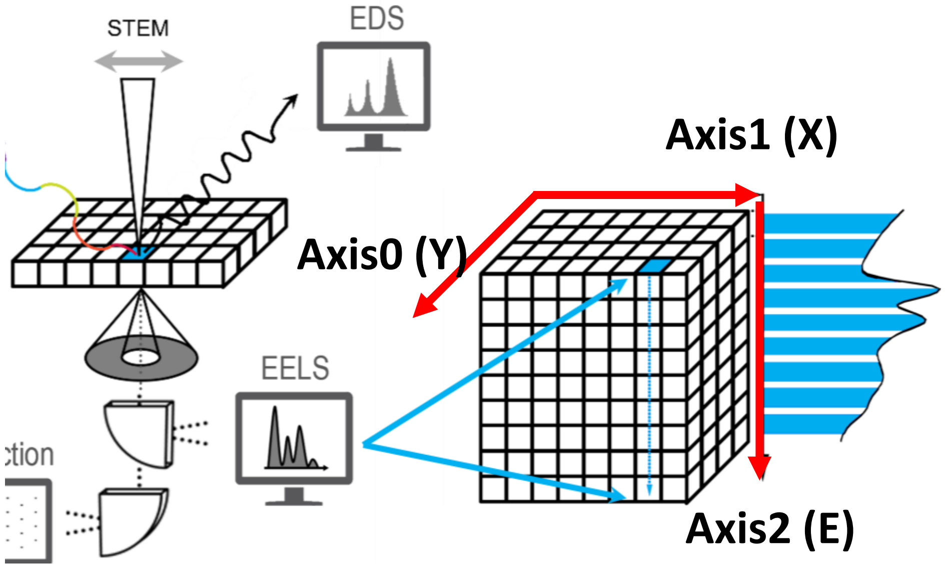


- Image (HAADF, 2-dim)

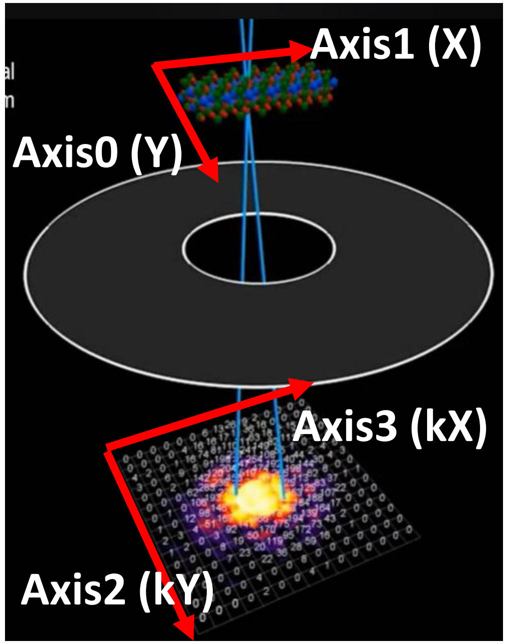


- Multi-dimension dataset

- STEM-EELS, STEM-EDS : 3-dim



- 4D-STEM : 4-dim



All types of TEM data can be converted to Numpy arrays.

Ref) https://www.gatan.com/techniques/

Datatype (List, Array) : Array

Create Numpy array

```
"""Create Numpy array"""
#Create numpy array template
arr1 = np.zeros((128,128), dtype="int8")
print("Arr1 : ", np.min(arr1),"to", np.max(arr1))

arr2 = np.ones((128,128), dtype="float32")
print("Arr2 : ", np.min(arr2),"to", np.max(arr2))

arr3 = np.random.random((128,128))
print("Arr3 : ", np.min(arr3),"to", np.max(arr3))

arr4 = np.random.randint(0,10, (128,128))
print("Arr3 : ", np.min(arr4),"to", np.max(arr4))

#Create numpy array manually
arr4 = np.array([[1.,2.],[3.,4.]]).astype("float32")
print("Arr4 : ", arr4)
print("Arr4 shape:",arr4.shape, "| Arr4 dtype:", arr4.dtype)
```

Usage of Numpy array

```
"""Usage of numpy array"""
vector = np.array([2,3])
spectrum = np.array([1,2,3,4,5,6,7,8,9])
image = np.array([[1,2,3],[4,5,6],[7,8,9]])
```

- Vector coordinate (1-dim)
- Spectrum (ex. EELS, EDS, 1-dim)
- Image (ex. STEM image, 2-dim)
- Multi-dimension dataset

(ex. STEM-EELS, STEM-EDS : 3-dim, 4D-STEM : 4-dim)

- Matrix operation (n-dim)

Difference between Numpy and List

```
"""Basic operations"""
vec1 = np.array([1,2,3])
vec2 = np.array([3,2,1])

list1 = [1,2,3]
list2 = [3,2,1]

mat_add = vec1+vec2
list_add = list1+list2
mat_concat = np.concatenate((vec1, vec2))
```

- List : Storage for everything
- Array : Storage and Operation for numbers (data)

Array shape conversion

```
spec_to_img = spectrum.reshape((3,3))
img_to_spec = image.flatten()
spec_to_2d = spectrum[np.newaxis,:]
```

- 1D (9,) to 2D (3,3) : reshape(3,3)
- 2D (3,3) to 1D (9,) : flatten
- 1D (9,) to 2D (1,9) : Usage of np.newaxis()

Datatype (List, Array) : Array

Index in Numpy array

```
print("[0,0]: ", tmp_arr[0,0])
print("[0,-1]: ", tmp_arr[0,-1])
print("[0]: ", tmp_arr[0])
print("[1,2:4]: ", tmp_arr[1,2:4])
print("[1:3,2:-1]: ", tmp_arr[1:3,2:-1])
```

Numpy 2D-array

Axis 1

(X axis)

Axis 0

(Y axis)

1	2	3	4	5
2	3	4	5	6
3	4	5	6	7

Ex. np.array([[1,2,3,4,5], [2,3,4,5,6], [3,4,5,6,7]])

np.where

```
tmp_where = np.where(tmp_img<4, 1, 0)
tmp_argwhere = np.argwhere(tmp_img<4)
```

1	2	3	4
4	5	6	7
7	8	9	10
9	10	11	12

np.where result

1	1	1	0
0	0	0	0
0	0	0	0
0	0	0	0

Index : [0,0], [0,1], [0,2]

np.max

```
tmp_max = np.max(tmp_img)
tmp_argmax = np.argmax(tmp_img) #Flattened index
tmp_maxind = np.unravel_index(tmp_argmax, tmp_img.shape)
```

1	2	3	4
4	5	6	7
7	8	9	10
9	10	11	12

np.max : 12

np.argmax : 15 – Flattened index

Index : [0,1], [0,2], [0,3]

Ex. np.array([[1,2,3,4], [4,5,6,7], [7,8,9,10], [9,10,11,12]])

Combination of list and numpy

List to numpy array : np.asarray(list)

```
list = []
arr1 = np.array([1,2,3])
arr2 = np.array([4,5,6])
arr3 = np.array([7,8,9])

list.append(arr1)
list.append(arr2)
list.append(arr3)
```

```
spec_stack = np.asarray(list)
print("Shape of numpy spectrum stack:", spec_stack.shape)
```

```
list = []
img1 = np.random.randint(0,10, (4,4))
img2 = np.random.randint(0,10, (4,4))
img3 = np.random.randint(0,10, (4,4))
```

```
list.append(img1)
list.append(img2)
list.append(img3)
```

```
img_stack = np.asarray(list)
print("")
print("Shape of numpy image stack:", img_stack.shape)
```

Numpy array to List : numpy_array.tolist()

```
list_stack = img_stack.tolist()
print("Numpy array to list :", type(list_stack))
print(len(list_stack))
print("List index 0:", list_stack[0])
print("Numpy index 0:", img_stack[0])
```

Recap

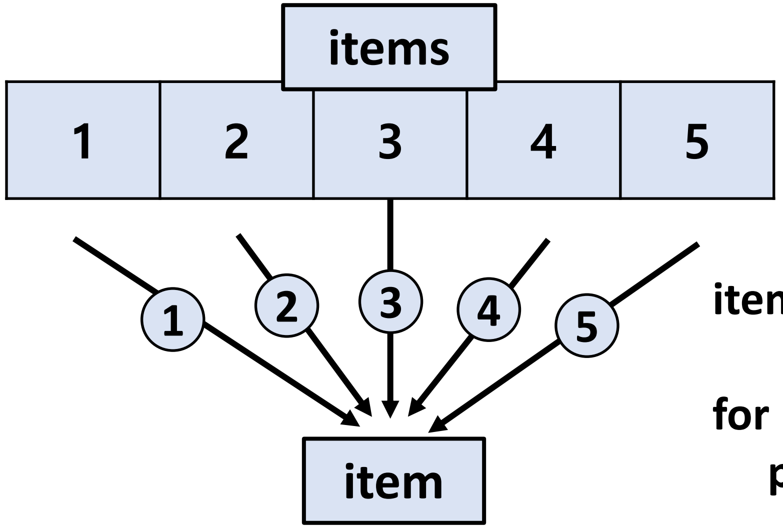
- List : Storage for everything
- Array : Storage and Operation for numbers (data)

Application

- 하나의 Multi-dimensional Stack으로 관리: Numpy array로 변환
- 여러 자료형 및 여러 shape의 데이터를 한 Storage에 보관하고 싶을 때 : List로 변환

For, If, While Statement : For statement

Usage of for statement



items = [1,2,3,4,5]

for item in items:
print(item)

Application1.

```
for item in tmp_img:
    print("Original Result: ", item)
```

1	2	3	4	→ 1st item
4	5	6	7	→ 2nd item
7	8	9	10	→ 3rd item
9	10	11	12	→ 4th item

Application2.

```
for count, item in enumerate(tmp_img):
    print("Result", count, ": ", item)
```

1	2	3	4	→ 1st item, count : 0
4	5	6	7	→ 2nd item, count : 1
7	8	9	10	→ 3rd item, count : 2
9	10	11	12	→ 4th item, count : 3

Application3.

```
for count, flat_item in enumerate(tmp_img.flatten()):
    print("Flat Result", count, ":", flat_item)
```

1	2	3	4	4	5	6	7	7	8	9	10	9	10	11	12

1st item, count : 0

2nd item, count : 1

For, If, While Statement : For statement

Nested For Loop

```
for i in range(5):
    for j in range(5):
        print("i,j :", i,j)
```

for j in range(5):

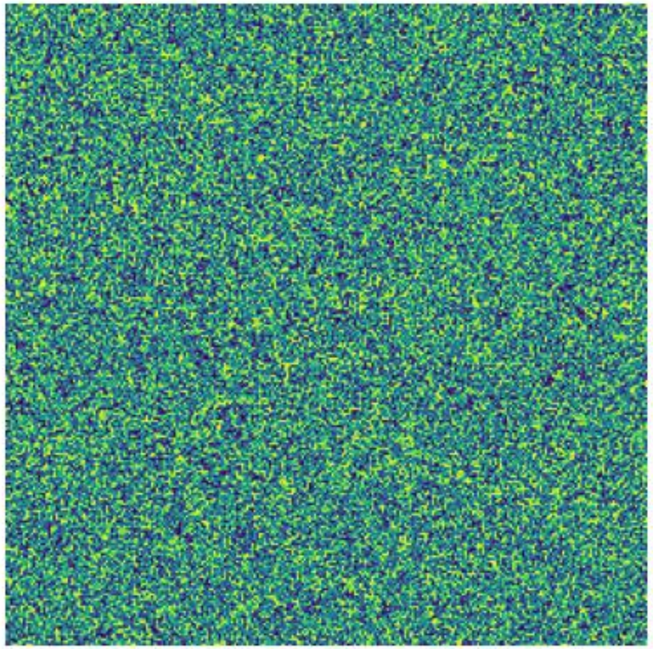
0,0	0,1	0,2	0,3	0,4
1,0	1,1	1,2	1,3	1,4
2,0	2,1	2,2	2,3	2,4
3,0	3,1	3,2	3,3	3,4
4,0	4,1	4,2	4,3	4,4

for i in range(5):

Scanning System

```
STEM_image = np.zeros((256,256))
for i in range(STEM_image.shape[0]):
    for j in range(STEM_image.shape[1]):
        STEM_image[i,j] = np.random.randint(0,10)
```

for j in range(256):



for i in range(256):

<u>Scanning System</u>

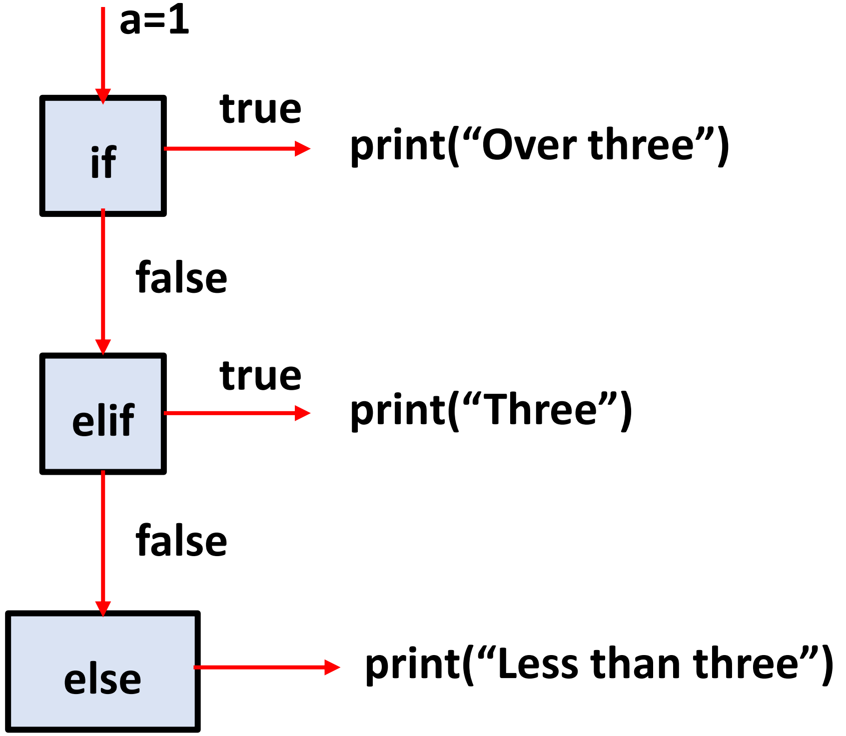
Slow scan axis : Y axis (Axis 0)

Fast scan axis : X axis (Axis 1)

For, If, While Statement : If and While

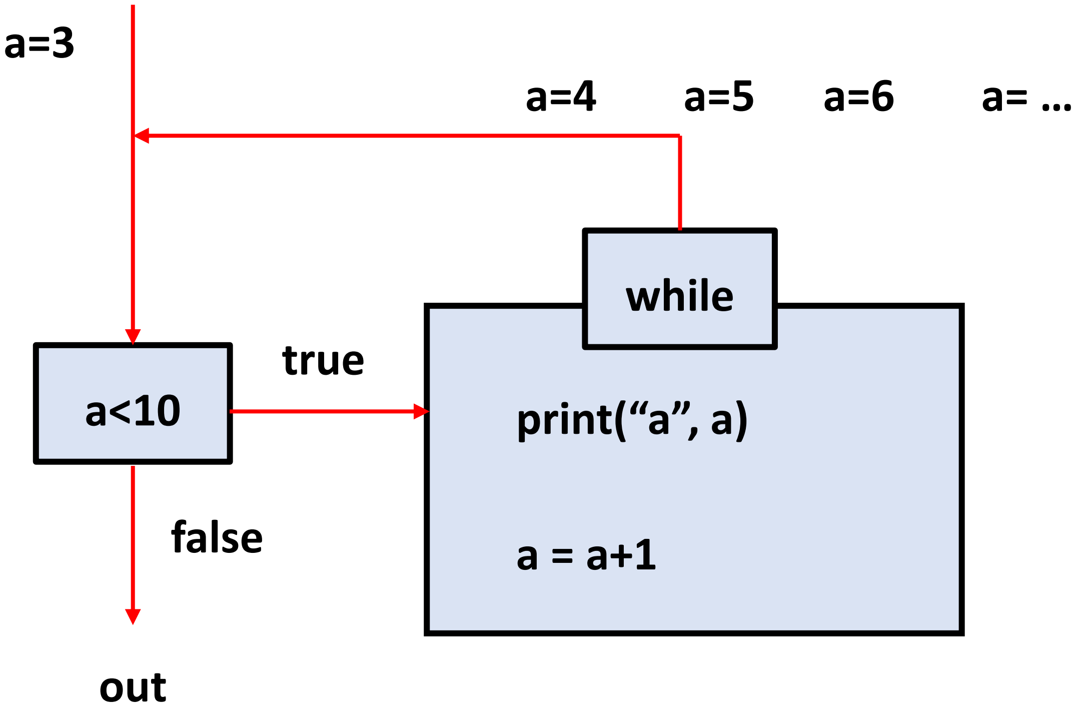
If statement

```python
a=1
if a>3:
    print("Over three")
elif a==3:
    print("Three")
else:
    print("Less than three")
```



While statement

```python
a=3
while a<10:
    print("a:",a)
    a= a+1
```



Combination of For, If and While Statement

Plot Spectrum and Image using pyplot

Plot Spectrum

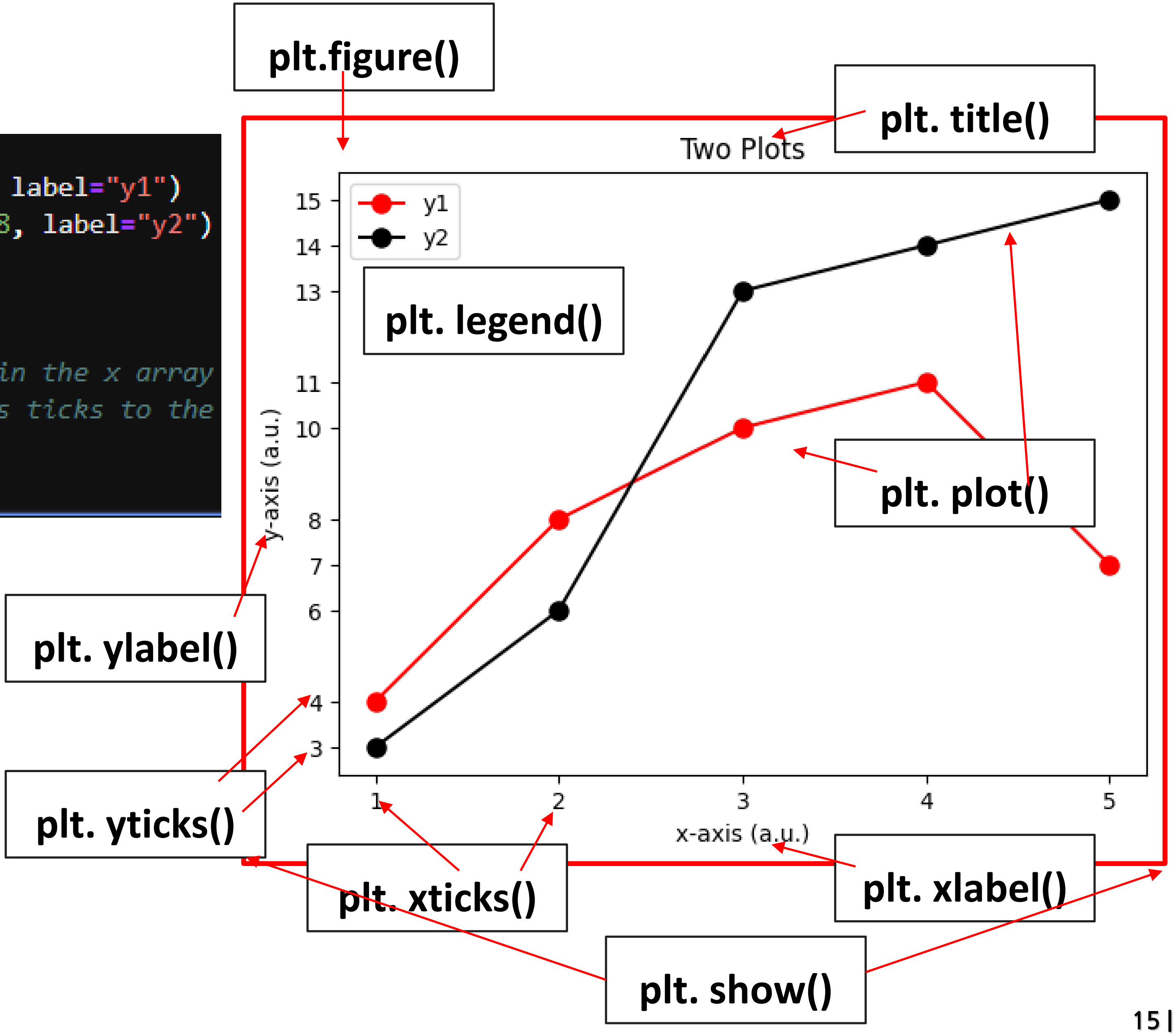
```python
plt.figure()
plt.plot(x,y1, color="red", marker='o', markersize=8, label="y1")
plt.plot(x,y2, color="black", marker='o', markersize=8, label="y2")
plt.title("Two Plots")
plt.xlabel("x-axis (a.u.)")
plt.ylabel("y-axis (a.u.)")
plt.xticks(x)  # Set the x-axis ticks to the numbers in the x array
plt.yticks(np.concatenate((y1, y2)))  # Set the y-axis ticks to the
plt.legend() #y1,y2 legend mark
plt.show()
```

Example of Spectrum y1,y2

x : np.array([[1,2,3,4,5])

y1 : np.array([4,8,10,11,7])

y2 : np.array([3,6,13,14,15])



Plot Spectrum and Image using pyplot

Plot Image

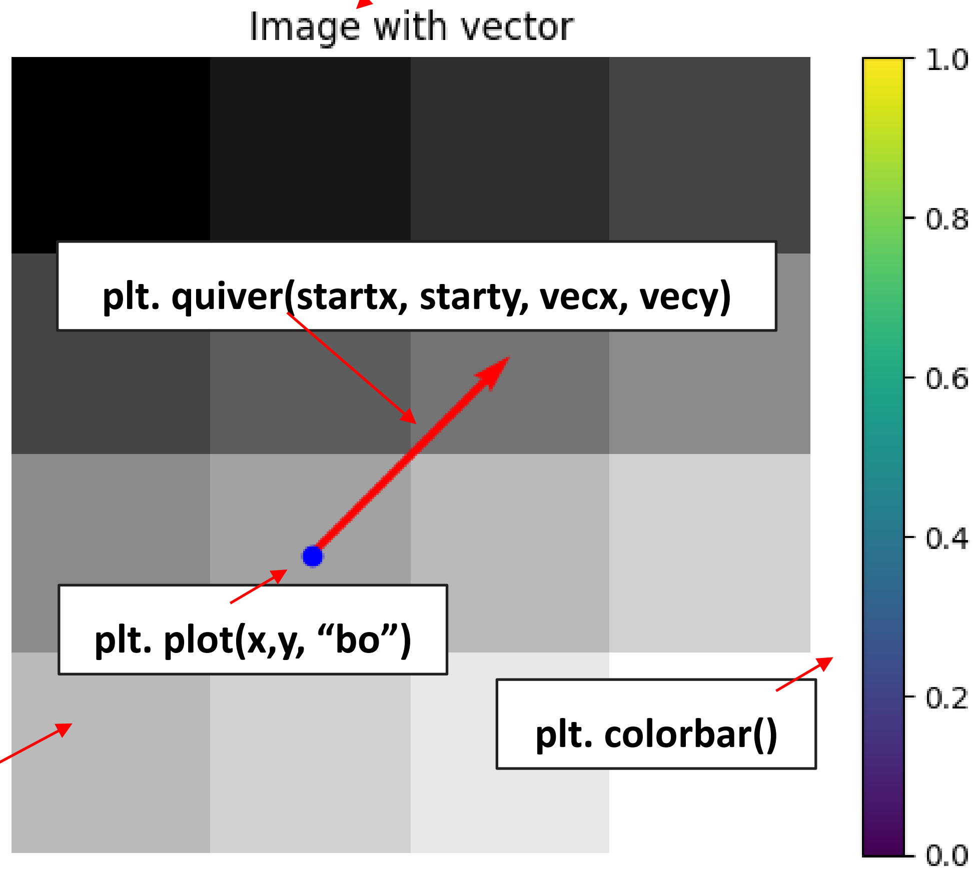
```python
plt.figure()
plt.imshow(tmp_img, cmap="gray")
plt.plot(1,2, 'bo')  # Mark the starting point with a blue circle, Note the axis definit
# (1,-1) vector starting at (1,2) -> Note the axis definition
plt.quiver(1, 2, 1, -1, angles='xy', scale_units='xy', scale=1, color='red', width=0.01)
plt.title("Image with vector")
plt.axis("off")
plt.colorbar()
plt.show()
```

1	2	3	4
4	5	6	7
7	8	9	10
9	10	11	12

Ex. np.array([[1,2,3,4], [4,5,6,7], [7,8,9,10],[9,10,11,12]])

plt.figure()

plt. title()



plt. quiver(startx, starty, vecx, vecy)

plt. axis("off")

plt. plot(x,y, "bo")

plt. colorbar()

plt. imshow()

plt. show()

Plot Spectrum and Image using pyplot

Making Multiple Plots

plt.subplots(yaxis, xaxis, figsize = (xaxis, yaxis))

```python
fig, ax = plt.subplots(1,3, figsize=(15, 5))

ax[0].plot()
ax[0].plot(x,y1, color="red", marker='o', markersize=8)
ax[0].plot(x,y2, color="black", marker='o', markersize=8)
ax[0].set_title("Two plots")
ax[0].set_xlabel("x-axis (a.u.)")
ax[0].set_ylabel("y-axis (a.u.)")
ax[0].set_xticks(x)
ax[0].set_yticks(np.concatenate((y1, y2)))

ax[1].imshow(tmp_img, cmap="inferno") #colormap : gray, viridis, inferno, jet etc..
ax[1].set_title("Image")
ax[1].axis("off")

ax[2].imshow(tmp_img, cmap="gray")
ax[2].plot(1,2, 'bo')  # Mark the starting point with a blue circle
ax[2].set_title("Image with  vector")
ax[2].quiver(1, 2, 1, -1, angles='xy', scale_units='xy', scale=1, color='red', width=0.01)
ax[2].axis("off")

plt.tight_layout()
plt.show()
```

```python
plt.title("Two Plots")
plt.xlabel("x-axis (a.u.)")
plt.ylabel("y-axis (a.u.)")
plt.xticks(x)  # Set the x-axis ticks
plt.yticks(np.concatenate((y1, y2)))
```

Add "set_"

```python
fig2, ax2 = plt.subplots(2,2)
print(type(ax2))
print(ax2.shape)
```

ax : numpy array with shape of (3,)

ax2 : (2,2) shape numpy array

Application

```python
"""Combination of For statement and pyplot subplots"""
#Plot all images in numpy image stack
num_img = img_stack.shape[0]
colormap = ["gray", "jet", "viridis"]
fig, ax = plt.subplots(1, num_img, figsize=(5*num_img, 5))

for i,img in enumerate(img_stack):
    ax[i].imshow(img, cmap=colormap[i])
    ax[i].axis("off")
    ax[i].set_title("Image"+ str(i+1))

plt.tight_layout()
plt.show()
```

- Usage of "img_stack"
- Usage of "shape"
- Usage of list
- Usage of "enumerate"
- Usage of "subplots"

Function

```
def operation(number):
    return 2*number+1

input = 5
output = operation(input)
print("Input:",input,"Output", output)
```

def function_name(variable):

code using variables

return output (*Optional)

output = function_name(input_variable)

Previous Code Example

```
list = []
img1 = np.array([[1,2,3], [4,5,6], [7,8,9]])
img2 = np.array([[2,0,4], [5,6,7], [8,5,3]])
img3 = np.array([[3,4,8], [9,3,8], [9,10,7]])

list.append(img1)
list.append(img2)
list.append(img3)
img_stack = np.asarray(list)
```

```
plt.figure(figsize=(6,6))
plt.imshow(tmp_img, cmap="inferno")
plt.title("Image")
plt.colorbar()
plt.axis("off")
plt.show()
```

Convert to Definition

```
def list_to_array(one, two, three):
    list = []

    list.append(one)
    list.append(two)
    list.append(three)

    img_stack = np.asarray(list)
    print("Stack Shape: ", img_stack.shape)

    return img_stack
```

```
img_stack_tmp = list_to_array(img1,img2,img3)
```

Positional argument

```
def display_image(title, image, axis=False):
    plt.figure()
    plt.title(title)
    plt.imshow(image, cmap='gray')
    if not axis:
        plt.axis('off')
    plt.show()
```

If statement

```
display_image("Image", tmp_img)
```

Class

```python
class ArrayProcessor:
    def __init__(self, array):
        #Attribute
        self.array = array
        self.array_sorted = self.sort_array()
        self.maxind = self.find_max_index()

    #Method
    def show_image(self, cmap='viridis'):
        plt.figure()
        plt.imshow(self.array, cmap=cmap)
        plt.colorbar()
        plt.title("Original Image")
        plt.show()

    def sort_array(self):
        return np.sort(self.array, axis=None).reshape(self.array.shape)

    def find_max_index(self):
        max_value = np.max(self.array)
        max_indices = np.where(self.array == max_value)
        return np.array(max_indices).reshape(2)
```

Parameter

Attribute

Method

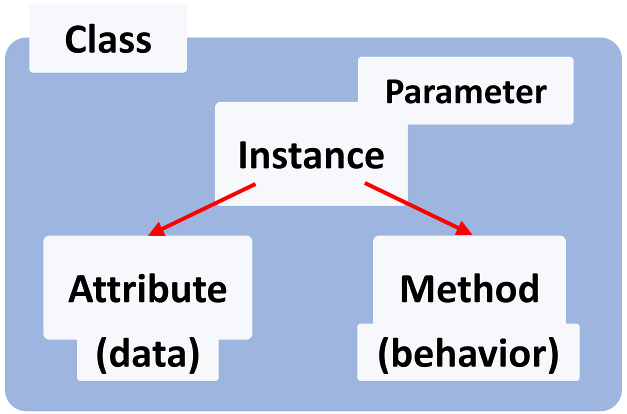
```python
# Sample 2D numpy array
tmp_arr = np.array([[1,2,3,4,5],[2,3,4,5,6],[3,4,5,6,7]])
# Create an instance of ArrayProcessor
processor = ArrayProcessor(tmp_arr)

# Show the image of the array
processor.show_image(cmap="jet") #can change cmap

# Print the maximum value indices
print("")
print("Indices of the maximum value:")
print(processor.maxind)

# Get the argsorted array
sorted_array = processor.sort_array()
display("Sorted Image", sorted_array)
```

Instance

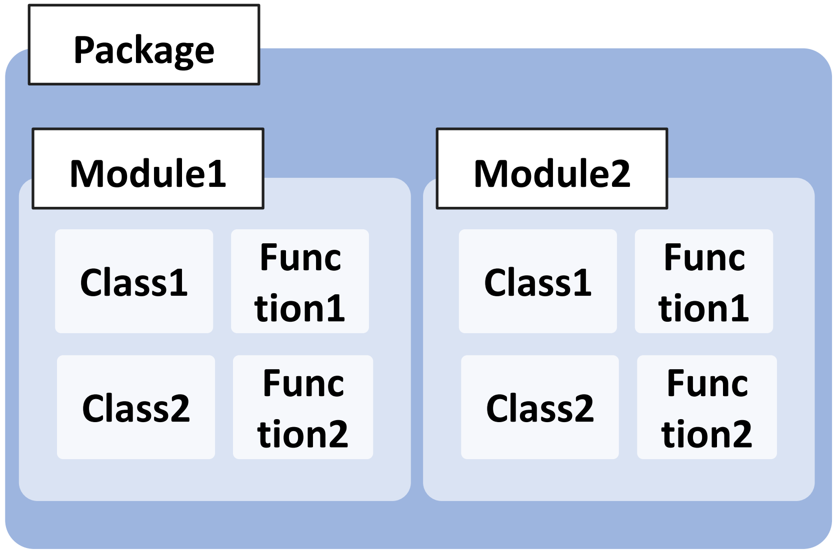
Attribute

Method

Class concept



Class, Function in Module



Example of Class, Function in numpy package

- tmp_arr = np.array([1,2,3])

: list를 받아서 ndarray를 반환하는 function

: ndarray Class의 Instance를 반환

- tmp_arr.shape

: ndarray class의 attribute인 shape을 tmp_arr instance에 적용

- output = np.max(tmp_arr)

: ndarray를 받아서 output으로 반환하는 function

Basic Python Structure

List, Numpy Array

Plot using Pyplot

For, If, While Statement

Function, Class

END of PART 1

Break Time

PART 2

TEM Application

Loading, Saving Tiff file

Filtering using cv2

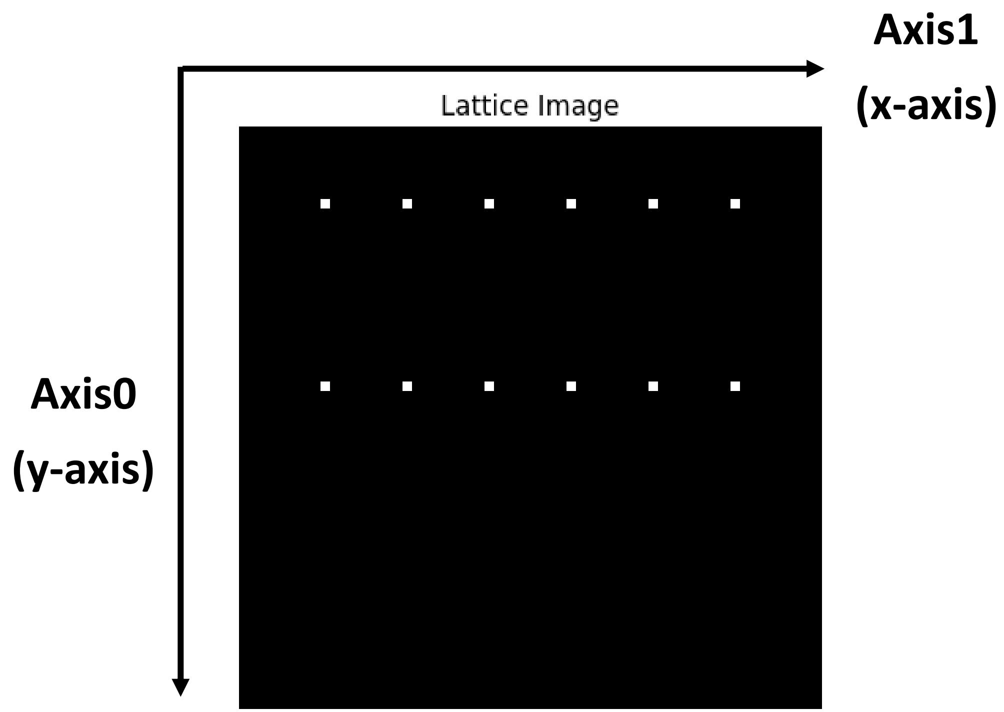
Manually Creating Image

Fitting Atomic column center

PART2. Application to HAADF image

1. Create Image Manually

```python
"""Create Image Manually"""

lattice_img = np.zeros((64,64))
pts = np.array([[8,9], [8,18], [8,27], [8,36], [8,45], [8,54],
                [28,9], [28,18], [28,27], [28,36], [28,45], [28,54]])
for pt in pts:
    lattice_img[pt[0], pt[1]] = 1

display("Lattice Image", lattice_img)
```



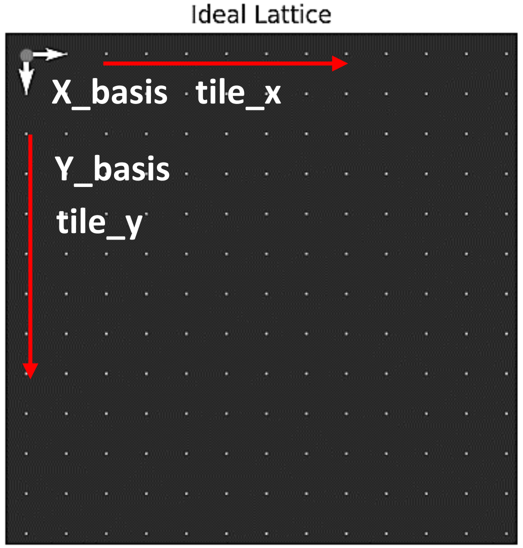
2. Create Lattice Image Manually

Parameters to define lattice

1) Start Point
2) Y_basis, X_basis vectors
3) Gaussian Template
4) Number of atomic columns (tile_y, tile_x)
5) Image Shape

Set coordinates of atomic columns

```python
lattice_img = np.zeros((256,256))
start = np.array([10,10])
y_basis = np.array([0,20]) #Change according to crystal
x_basis = np.array([20,0]) #Change according to crystal

tile_y = 30
tile_x = 30

for y in range(tile_y):
    for x in range(tile_x):
        coord = start + y*y_basis + x*x_basis
        if coord[0]<lattice_img.shape[0] and coord[1]<lattice_img.shape[1]:
            lattice_img[coord[0], coord[1]] = 1
```

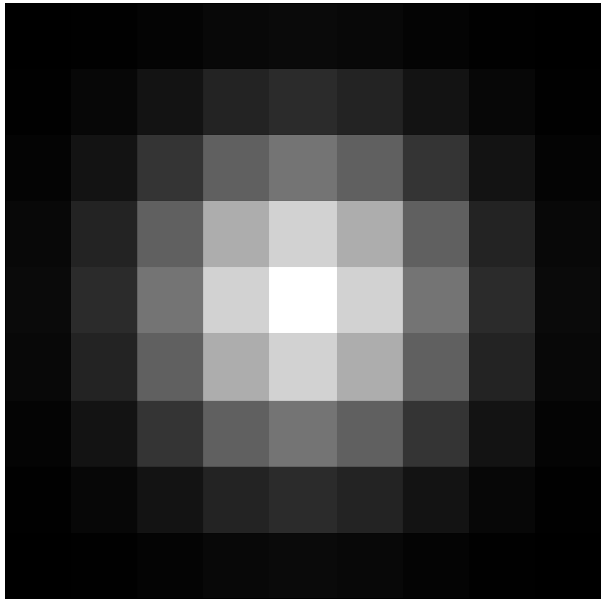


2. Create Lattice Image Manually

Create Gaussian Template

```python
def gaussian_kernel(size, A, sigma=1):
    """Creates a 2D Gaussian kernel."""
    x, y = np.meshgrid(np.linspace(-size//2, size//2+1, size), np.linspace(-size//2, size//2+1, size))
    kernel = A*np.exp(-(x**2 + y**2) / (2 * sigma**2)) #2D Gaussian Function
    return kernel

"""Example of creating Gaussian Template"""
size, amplitude, sig = 9, 5, 2
gaussian_template = gaussian_kernel(size, amplitude, sigma=sig)
display("Gaussian template", gaussian_template)
```

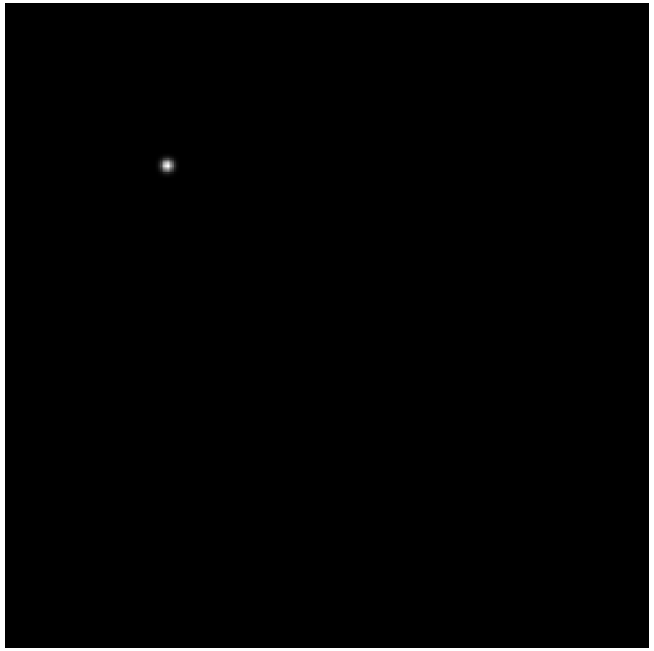
Gaussian template



Overlay Gaussian Template to atomic column

```python
atomic_img = np.zeros((256,256))

pos = np.array([64,64])
atomic_img[pos[0]-size//2: pos[0]+size//2+1, pos[1]-size//2:pos[1]+size//2+1] = gaussian_template
```

Overlaid gaussian template to one atomic column



2. Create Lattice Image Manually

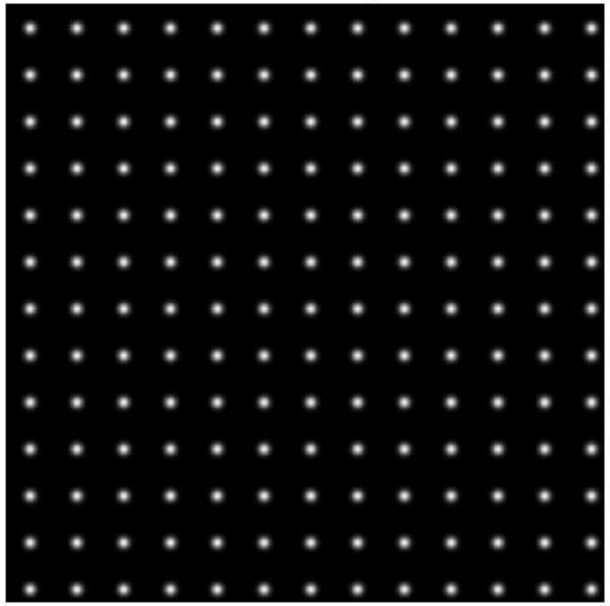
Make Lattice Image Generation function

```python
"""Function to generate lattice image based on parameters"""
def generate_lattice_image(start, y_basis, x_basis, gaussian_template, tile_y, tile_x, size, shape=(512,512)):
    lattice_img = np.zeros(shape)

    for y in range(tile_y):
        for x in range(tile_x):
            coord = start + y * y_basis + x * x_basis #Coordinate of each atomic column
            coord = coord.astype(int)  # Ensure coord is integer
            if coord[0] + size // 2 + 1 < lattice_img.shape[0] and coord[1] + size // 2 + 1 < lattice_img.shape[1]:
                lattice_img[coord[0] - size // 2: coord[0] + size // 2 + 1, coord[1] - size // 2: coord[1] + size // 2 + 1] = gaussian_template

    return lattice_img
```

Image shape limit

```python
"""Example of generating lattice image"""
start = np.array([10,10])
y_basis = np.array([0,20]) #Change according to crystal structure
x_basis = np.array([20, 0]) #Change according to crystal structure
tile_y = 30
tile_x = 60
size=9
gaussian_template = gaussian_kernel(9, 5, 2) #Change according to atom element

lattice_img =  generate_lattice_image(start, y_basis, x_basis, gaussian_template, tile_y, tile_x, size, shape=(256,256))
display("Lattice Image", lattice_img)
```

Lattice Image

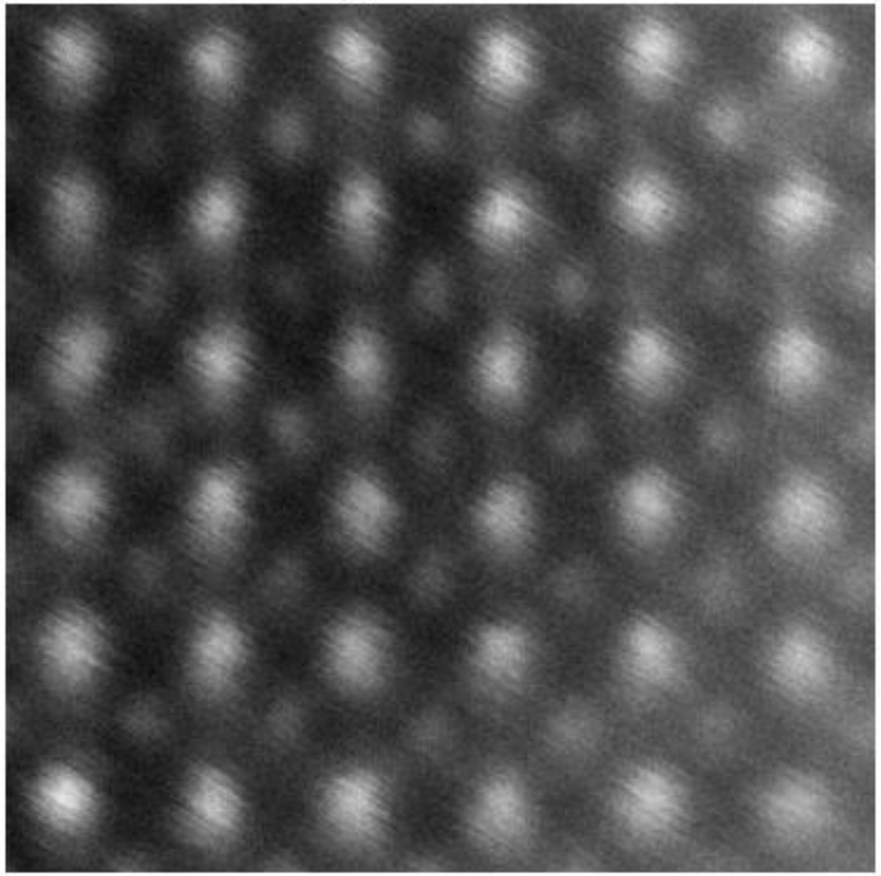


3. Load Experimental HAADF image

```
import tifffile #Import tifffile to load tif file
img_adr = "TEM_workshop/data/atomic_image.tiff"
```

```
HAADF = tifffile.imread(img_adr)
display_image("Original HAADF", HAADF)
```

Original HAADF

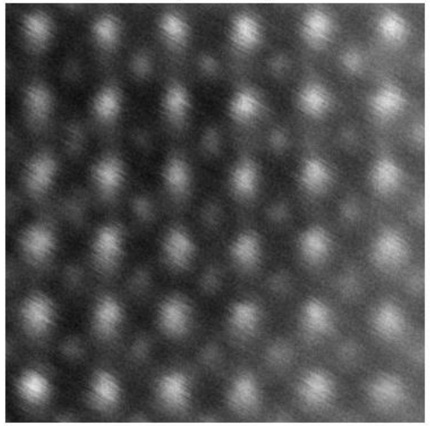


4. Filtering original image

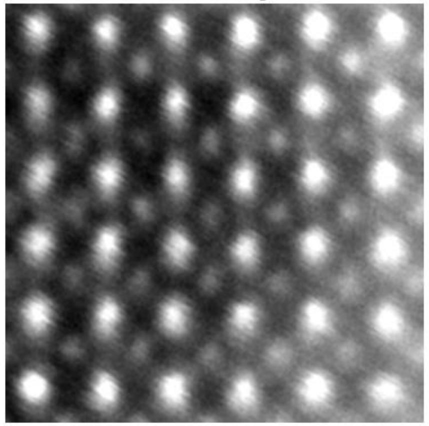
```
import cv2
HAADF_uint8 = cv2.normalize(HAADF, None, 0, 255, cv2.NORM_MINMAX).astype(np.uint8)
# Set Parameters for filtering
custom_kernel = gaussian_kernel(5, 0.1, sigma=3) #As a example
print("Kernel")
print(custom_kernel)
print("")

# Apply Custom Filter
filtered_HAADF = cv2.filter2D(HAADF_uint8, -1, custom_kernel)
display("Original Image", HAADF_uint8)
display("Filtered Image", filtered_HAADF)
```

- **Convert “float32” numpy array to normalized “uint8” numpy array**
- **Set kernel and apply filter(kernel) to filter2D function**
- **Can do various filtering using various kernels and methods**
- **Can save outputs using tifffile.imwrite(file_adr, output)**

Original Image



Filtered Image

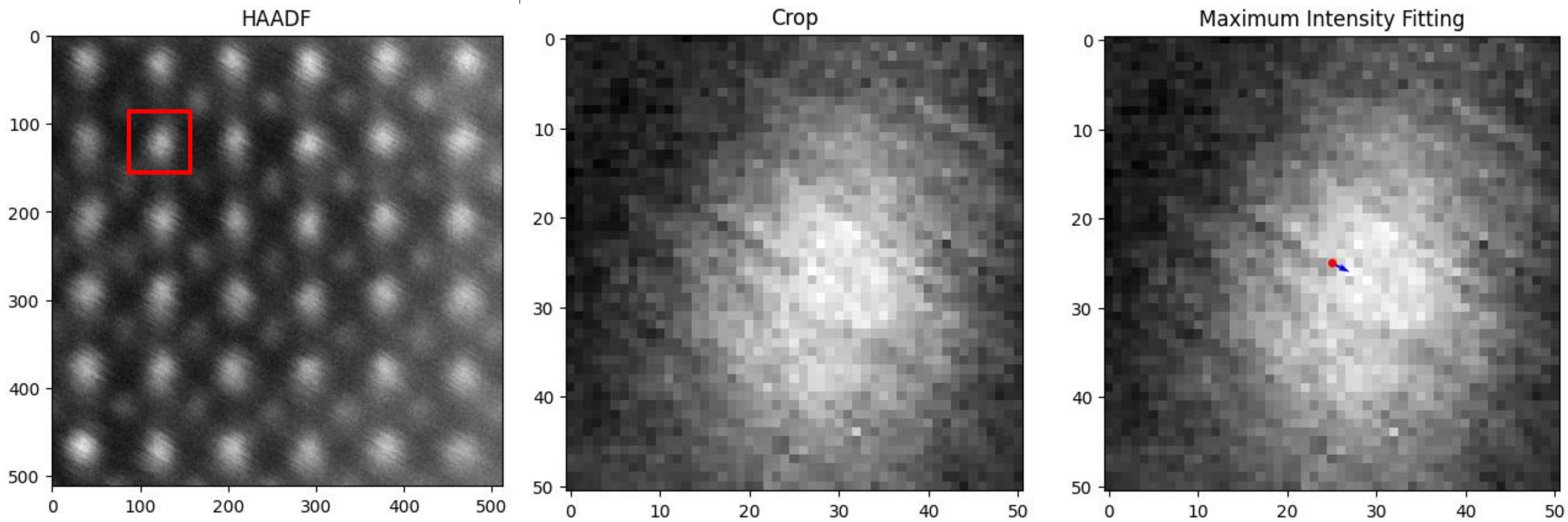


5. Crop Region of Interest

```
atom_center = [120,120]
crop_size = 51 #Should be odd number
crop = HAADF[atom_center[0]-crop_size//2:atom_center[0]+crop_size//2+1, atom_center[1]-crop_size//2:atom_center[1]+crop_size//2+1]
```

6. Atomic Column center fitting (Maximum Intensity Fitting)

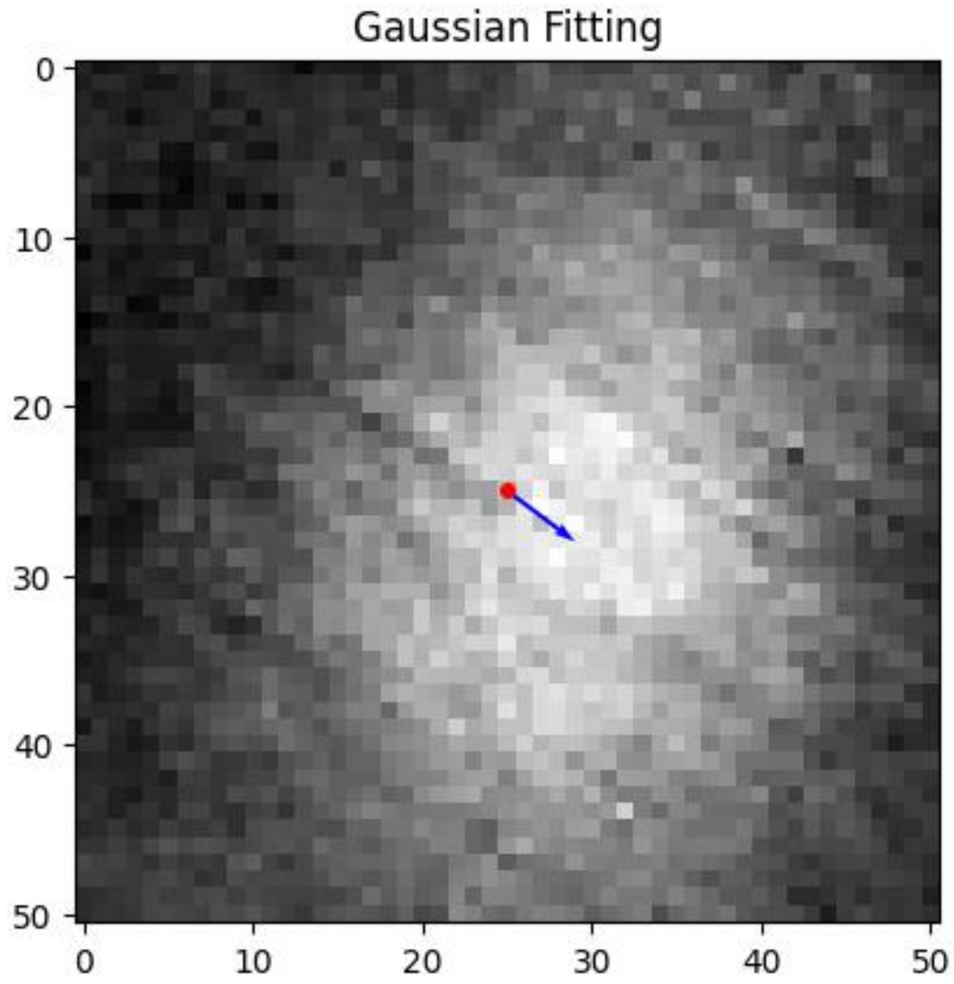
```
crop_argmax = np.argmax(crop) #Flattened index
crop_maxind = np.unravel_index(crop_argmax, crop.shape)
```



6. Atomic Column center fitting (Gaussian Fitting)

```
from scipy.optimize import curve_fit #Import curve_fit for guassian fitting

def gaussian_2d(xy, amplitude, xo, yo, sigma):
    x = xy[0]
    y = xy[1]
    xo = float(xo)
    yo = float(yo)
    g = amplitude * np.exp(-(((x-xo)**2)/(2*sigma**2) + ((y-yo)**2)/(2*sigma**2)))
    return g.ravel()

def gaussian_fitting(crop):
    data = crop.flatten()
    y_,x_ = crop.shape
    y = np.linspace(0, y_-1, y_) #y_grid
    x = np.linspace(0, x_-1, x_) #x_grid
    x,y = np.meshgrid(x,y) #convert to meshgrid

    initial_guess = [np.max(crop), int(x_/2), int(y_/2), 0.5]
    popt, pcov = curve_fit(gaussian_2d, np.array([x, y]), data, p0=initial_guess)
    amplitude, center_x, center_y, sigma = popt

    return amplitude, int(np.round(center_x)), int(np.round(center_y)), sigma
```

```
"""Using Gaussian Fitting to get refined atomic center"""
amplitude, center_x, center_y, sigma = gaussian_fitting(crop)
```



- **Apply 2D Gaussian function to Gaussian fitting**
- **Use curve_fit module to get parameters from fitted function**
- **Get amplitude, centerx, centery and sigma value of 2D Gaussian**

6. Atomic Column center fitting

```python
"""Make function for refining center"""
def atom_center_fitting(img, initial_guess, crop_size):
    assert crop_size %2 !=0, "Crop_size should be odd number"
    crop = img[initial_guess[0]-crop_size//2:initial_guess[0]+crop_size//2+1, initial_guess[1]-crop_size//2:initial_guess[1]+crop_size//2+1]
    _, center_x, center_y, _ = gaussian_fitting(crop)
    refined_center = np.array(initial_guess)+np.array([center_y-crop_size//2, center_x-crop_size//2])
    return refined_center

def atom_center_fitting_2(img, initial_guess, crop_size):
    assert crop_size %2 !=0, "Crop_size should be odd number"
    crop = img[initial_guess[0]-crop_size//2:initial_guess[0]+crop_size//2+1, initial_guess[1]-crop_size//2:initial_guess[1]+crop_size//2+1]
    center_y, center_x = np.unravel_index(np.argmax(crop), crop.shape)
    refined_center = np.array(initial_guess)+np.array([center_y-crop_size//2, center_x-crop_size//2])
    return refined_center
```

1) Gaussian fitting

2) Maximum Intensity

7. Atom Center Fitting in HAADF image

- **Know Image shape, Tile_y, Tile_x as a prior knowledge**
- **Find start point, y_basis, x_basis vector, crop_size for lattice generation**
- **Create Gaussian Template**

Use generate_lattice_image function to generate lattice image

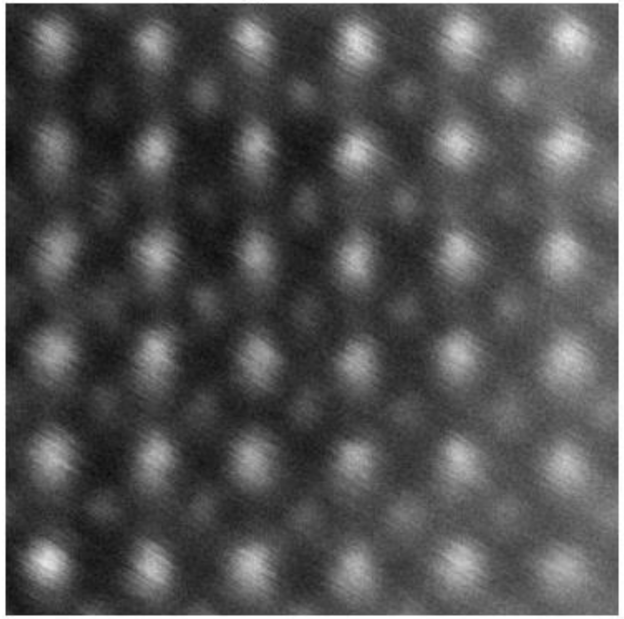
Recap

Parameters to define lattice

1) Start Point
2) Y_basis, X_basis vectors
3) Gaussian Template
4) Number of atomic columns (tile_y, tile_x)
5) Image Shape

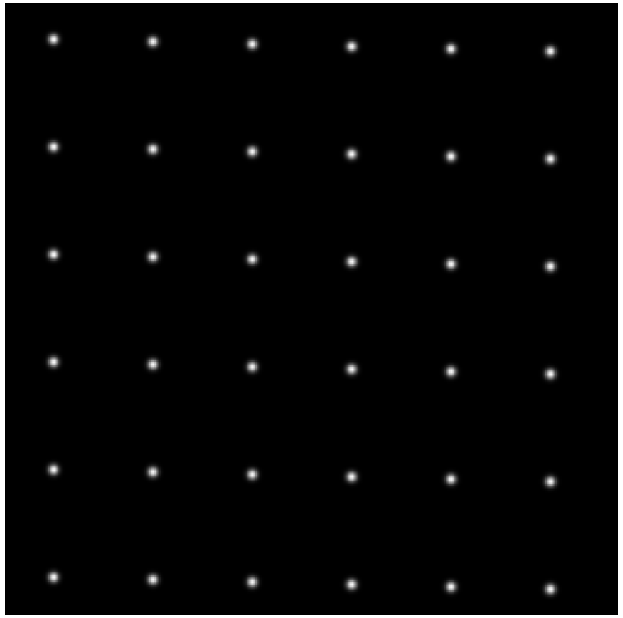
7. Atom Center Fitting in HAADF image

Original HAADF



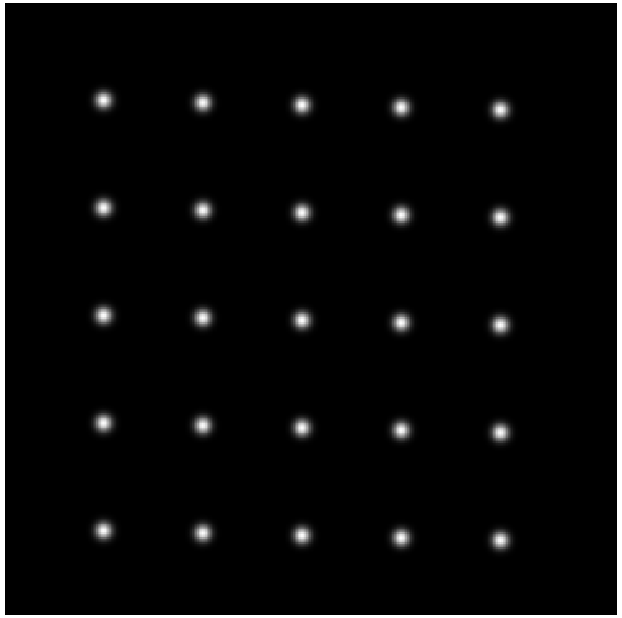
1. Bright columns

Lattice Image



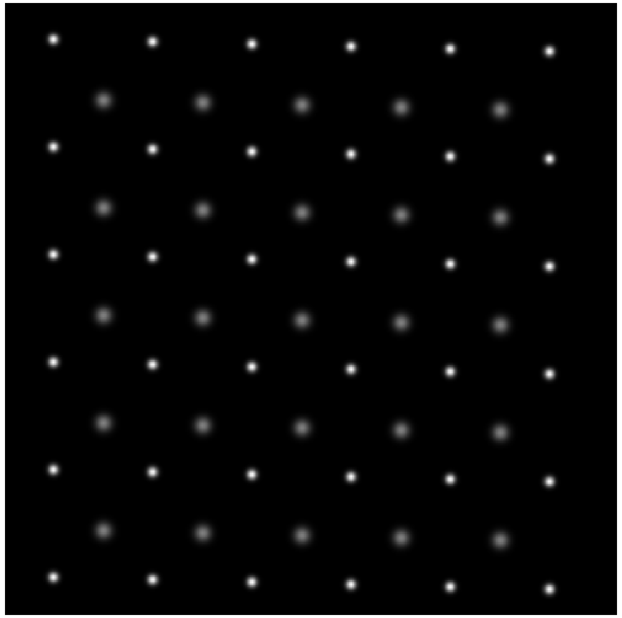
2. Dark columns

Lattice Image



3. Sum

Lattice Image



4. Get All Atom column center coordinates

Bright, Dark column coordinate 저장

```
#Positions of Bright Columns
bright_coord = []
for y in range(6):
    for x in range(6):
        coord1 = fitted_start_center1+y*y_basis1+x*x_basis1
        bright_coord.append(coord1)
```

Coordinate data csv파일로 저장

```
#Save coordinate file (Optional)
save_folder = "C:/Users/user/Downloads/"
bright_array = np.asarray(bright_coord).astype(np.uint8)
dark_array = np.asarray(dark_coord).astype(np.uint8)

np.savetxt(save_folder+"bright_coord.csv", bright_array, delimiter = ",")
np.savetxt(save_folder+"dark_coord.csv", dark_array, delimiter = ",")
```

Summary

Basic Python Structure

List, Numpy Array

Plot using Pyplot

For, If, While Statement

Function, Class

TEM Application

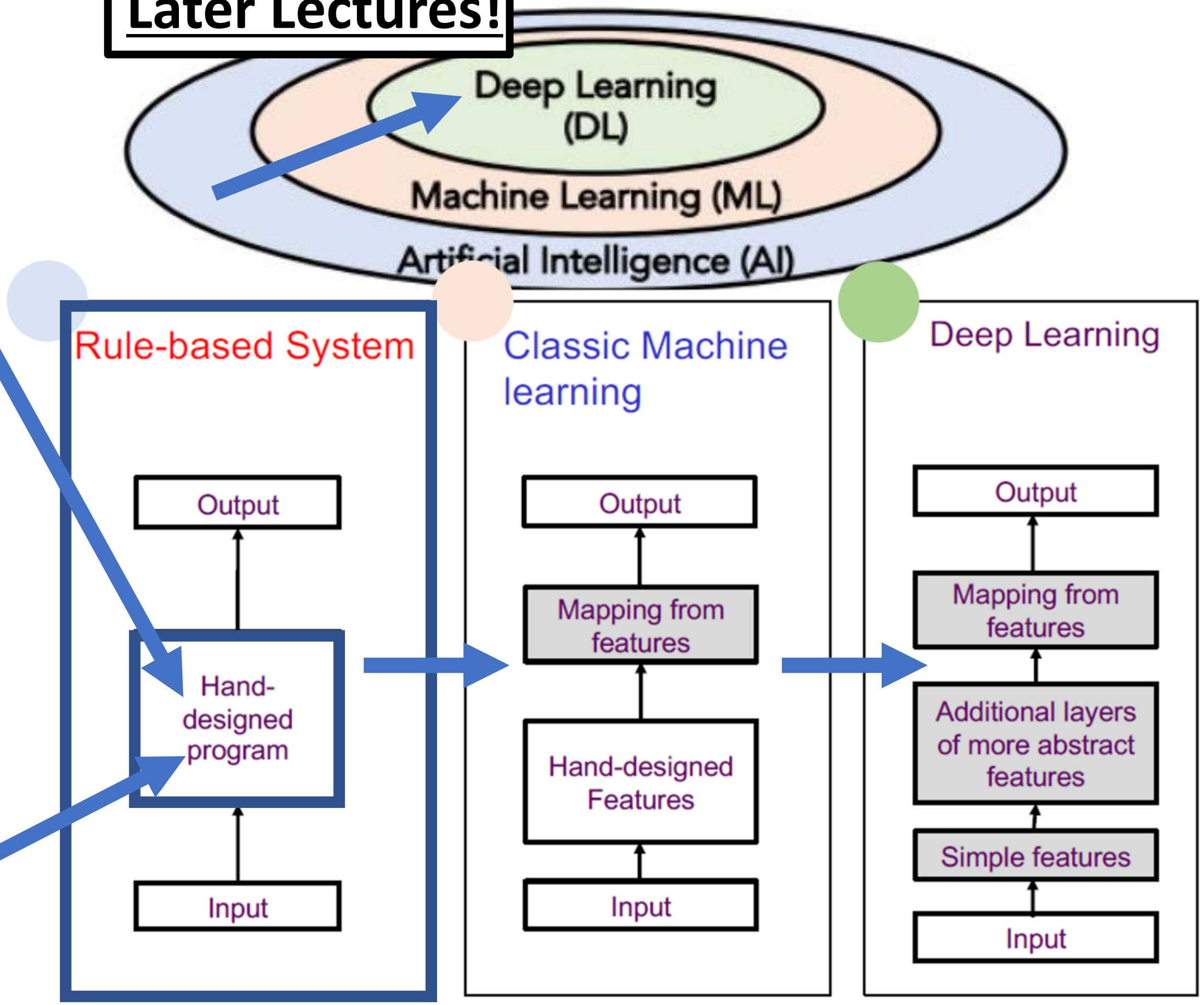
Loading, Saving Tiff file

Filtering using cv2

Manually Creating Image

Fitting Atomic column center

Later Lectures!



29회 투과전자현미경 워크샵

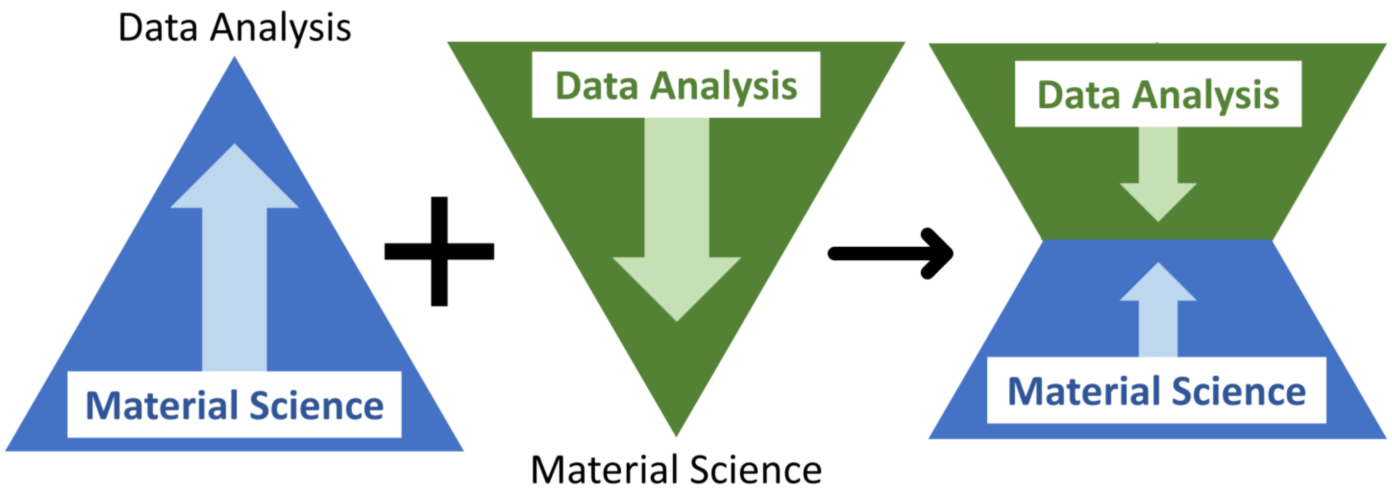
데이터 처리 기술 실습

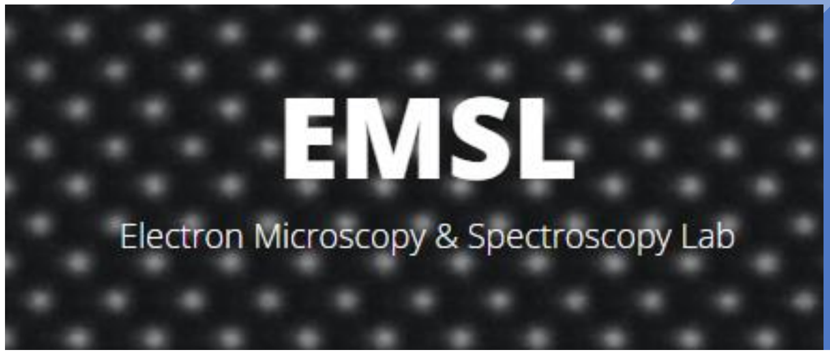
Ingyu Yoo (Email: yig0222@snu.ac.kr)

Code Link: https://github.com/yig0222/

Research Interest : Structural Analysis on TMDs, Machine Learning & 4D-STEM postprocessing

Thank you for listening





서 울 대 학 교
SEOUL NATIONAL UNIVERSITY

서울대학교 재료공학부