



*matplotlib*



NumPy Matplotlib SciPy

For ML Techniques



Math

# Agenda

## **NumPy**

- Vectors
- Matrices
- NumPy arrays
- Sampling from Distributions

## **SciPy**

- Unconstrained optimisation
- Constrained optimisation

## **Matplotlib**

- Plotting simple curves
- Bar plots, histograms, scatter plots
- Contour plots, 3D plots

# Process

## Sequence

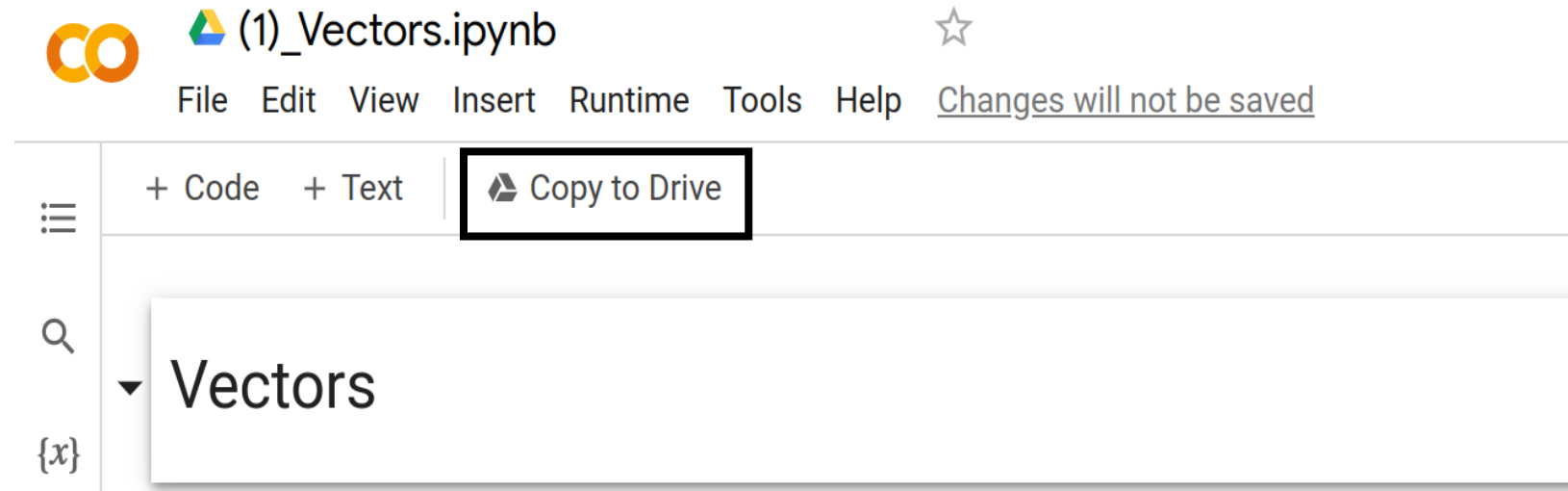
- (1) NumPy
- (2) Matplotlib
- (3) NumPy
- (4) Matplotlib
- (5) NumPy
- (6) Matplotlib, SciPy

## Type of Session

- Code with instructor
- Code demo + QA

## Colabs and Slides

- [http://tiny.cc/mlt\\_workshop](http://tiny.cc/mlt_workshop)
- We will go from (1) to (6)
- Open colab and copy it to your drive
- Slides will be uploaded tonight



# NumPy, Matplotlib, SciPy

NumPy

- Created in 2005
- Open source
- Current version: 1.23.0

Linear Algebra

- Vectors
- Matrices

Matplotlib

- Created in 2003
- Open source
- Current version: 3.6.2

Visualising

- data
- models

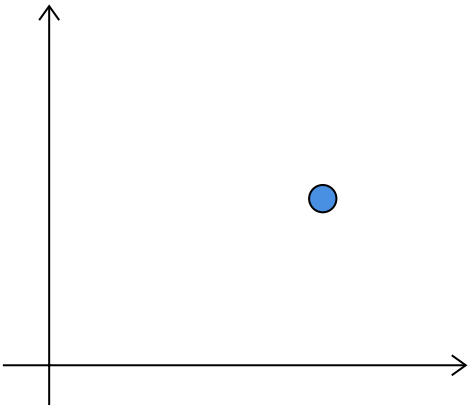
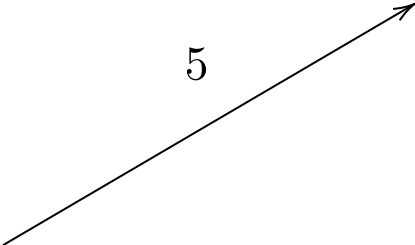
SciPy

- Created in 2001
- Open source
- Current version: 1.9.3

Optimisation

# Vectors

$$\mathbf{x} = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$$



```
np.array([3, 4])
```

sample

feature-vector

data-point

**array**

example

practical

abstract

geometric

algebraic

**computational**

Vector

1	2	3
---	---	---

# Broadcasting

1	2	3
---	---	---

\*

2
---

# Broadcasting

1	2	3
---	---	---

\*

2	2	2
---	---	---



# Broadcasting

1	2	3
---	---	---

\*

2	2	2
---	---	---

2	4	6
---	---	---

# Broadcasting

1	2	3
---	---	---

\*

2	2	2
---	---	---

2	4	6
---	---	---

- Conceptual explanation
- Not exactly what happens in NumPy

# Broadcasting

1	2	3
---	---	---

\*

2
---

→ More efficient

2	4	6
---	---	---

1	2	3
---	---	---

\*

2	2	2
---	---	---

2	4	6
---	---	---

# Math $\rightarrow$ NumPy (Vectors)

$\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$

$$\mathbf{x} + \mathbf{y}$$

$$\mathbf{x} + \mathbf{y}$$

$$\mathbf{x} \odot \mathbf{y}$$

$$\mathbf{x} * \mathbf{y}$$

$$\mathbf{x}^T \mathbf{y} \text{ or } \mathbf{x} \cdot \mathbf{y}$$

$$\mathbf{x} @ \mathbf{y}$$

$$a \cdot \mathbf{x} + b \cdot \mathbf{1}$$

$$a * \mathbf{x} + b$$

$$\mathbf{0}$$

$$\text{np.zeros}(d)$$

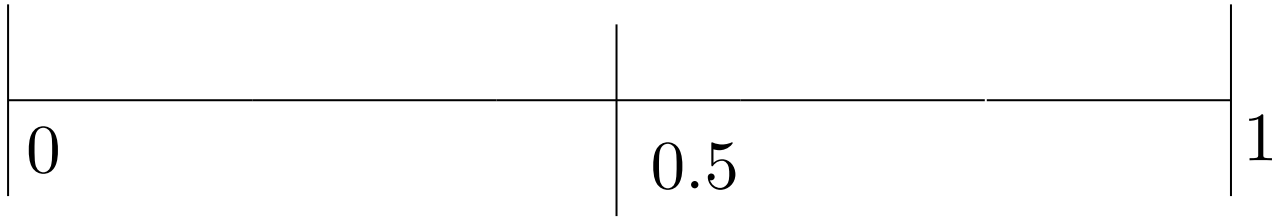
$$\mathbf{1}$$

$$\text{np.ones}(d)$$

$$\|\mathbf{x}\|_p$$

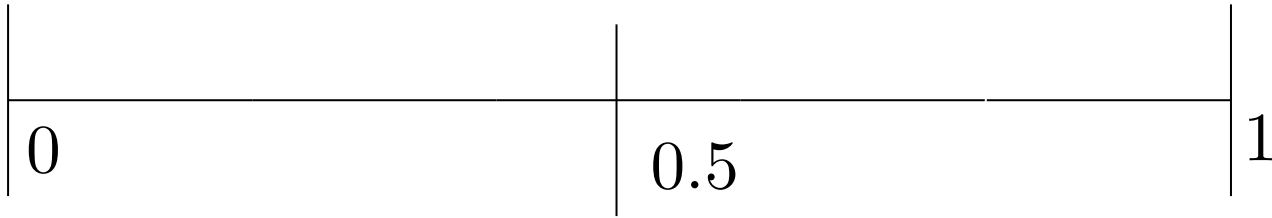
$$\text{np.linalg.norm}(\mathbf{x}, p)$$

# Linspace

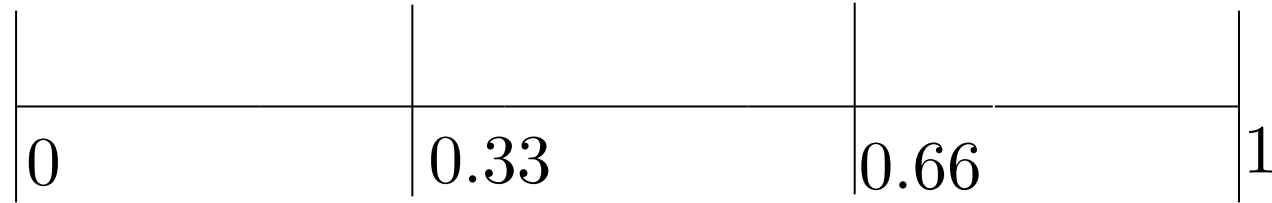


```
np.linspace(0, 1, 3)
```

# Linspace

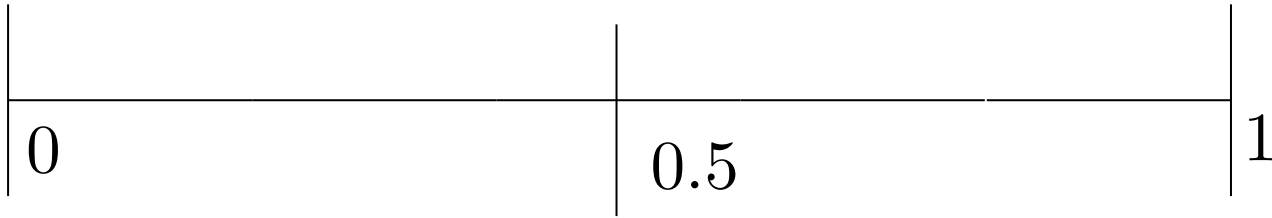


```
np.linspace(0, 1, 3)
```

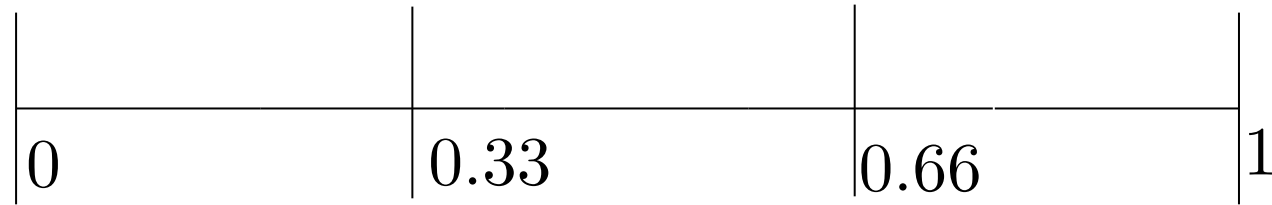


```
np.linspace(0, 1, 4)
```

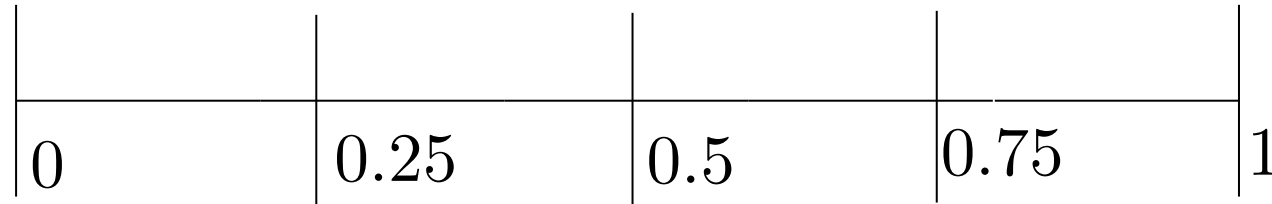
# Linspace



```
np.linspace(0, 1, 3)
```

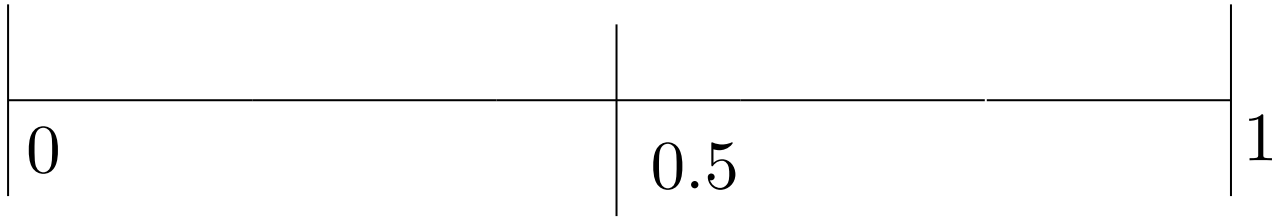


```
np.linspace(0, 1, 4)
```

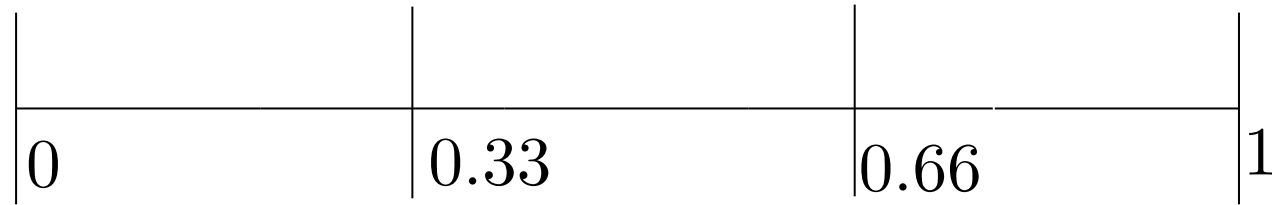


```
np.linspace(0, 1, 5)
```

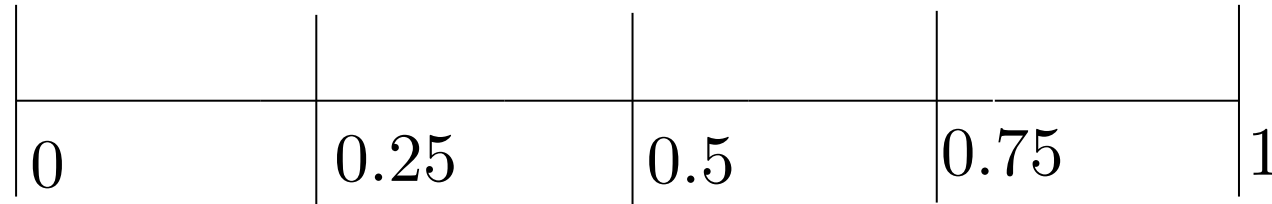
# Linspace



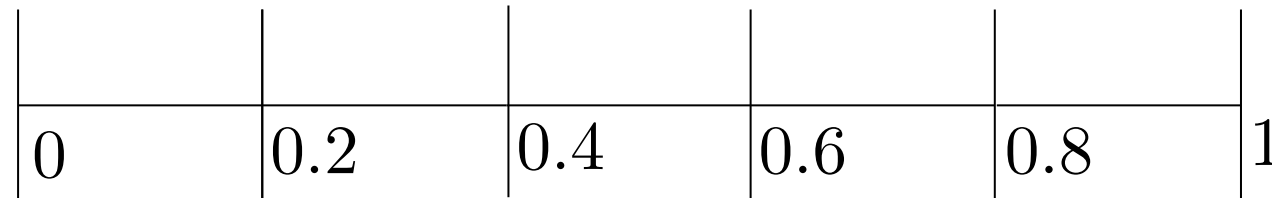
```
np.linspace(0, 1, 3)
```



```
np.linspace(0, 1, 4)
```



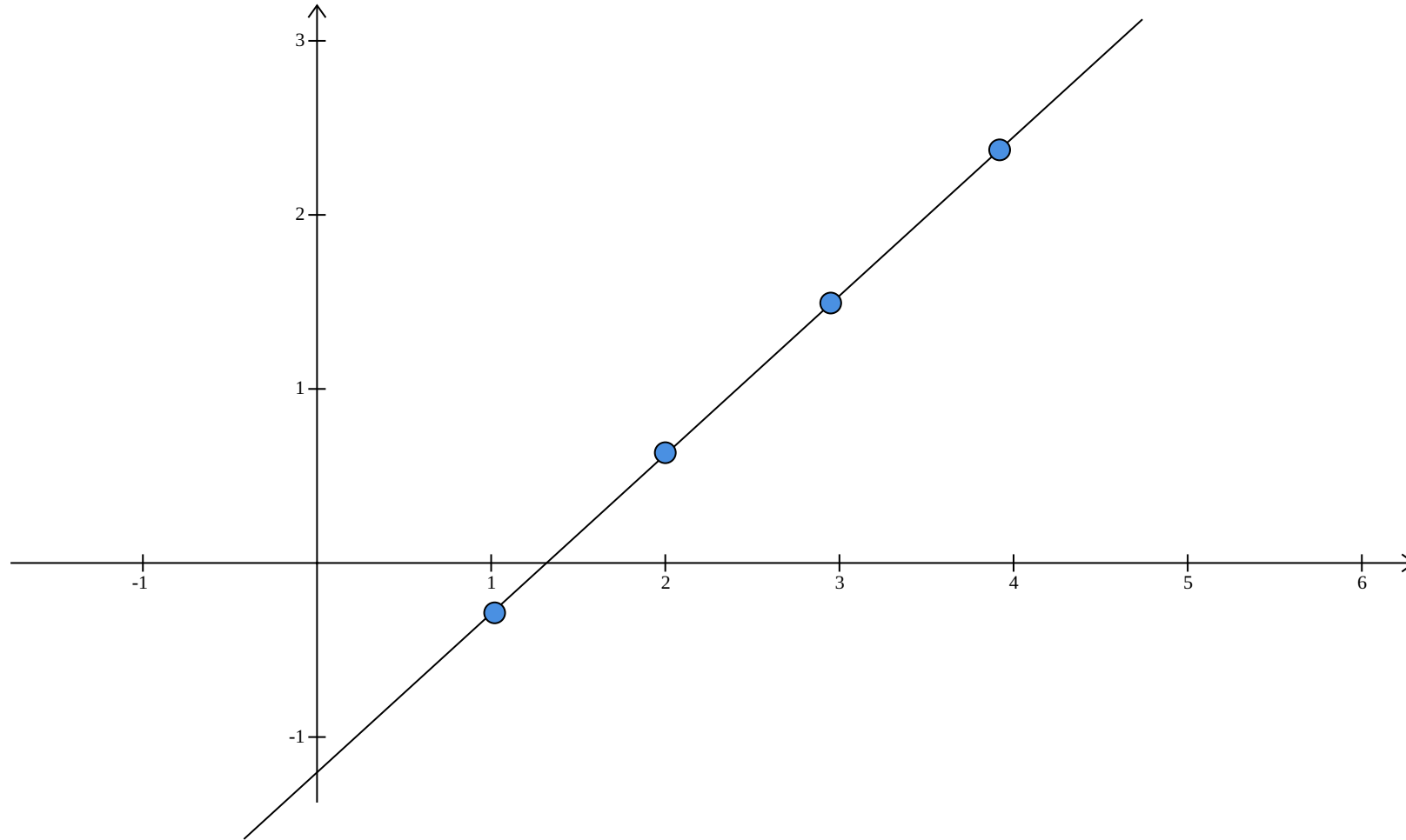
```
np.linspace(0, 1, 5)
```



```
np.linspace(0, 1, 6)
```



# Plotting in Matplotlib



$[x_1, x_2, x_3, x_4]$

$[y_1, y_2, y_3, y_4]$

# Subplots

```
plt.subplot(2, 2, 1)
```

```
plt.subplot(2, 2, 2)
```

```
plt.subplot(2, 2, 3)
```

```
plt.subplot(2, 2, 4)
```

# Math $\rightarrow$ NumPy (Matrices)

$$\mathbf{A} + \mathbf{B}$$

$$\mathbf{A} + \mathbf{B}$$

$$\mathbf{A}, \mathbf{B} \in \mathbb{R}^{m \times n}$$

$$\mathbf{I} \in \mathbb{R}^{d \times d}$$

$$\mathbf{A} \odot \mathbf{B}$$

$$\mathbf{A} * \mathbf{B}$$

$$\mathbf{A}\mathbf{B}$$

$$\mathbf{A} @ \mathbf{B}$$

$$\mathbf{A}^T$$

$$\mathbf{A.T}$$

$$\mathbf{I}$$

$$\text{np.eye}(d)$$

$$\text{diag}(a_1, \dots, a_d)$$

$$\text{np.diag}(a_1, \dots, a_d)$$

$$\mathbf{0}$$

$$\text{np.zeros}((m, n))$$

$$\mathbf{1}$$

$$\text{np.ones}((m, n))$$

# n-Dimensional Arrays

1	2	3
---	---	---

```
shape = (3, )
```

```
ndim = 1
```

```
row
```

# n-Dimensional Arrays

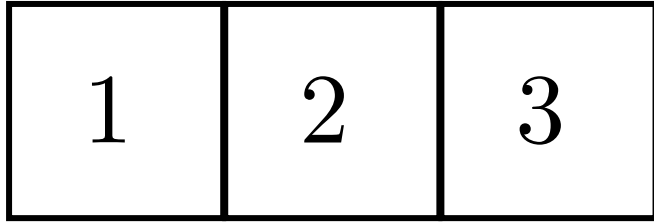
1	2	3
---	---	---

shape = (3, )  
ndim = 1  
row

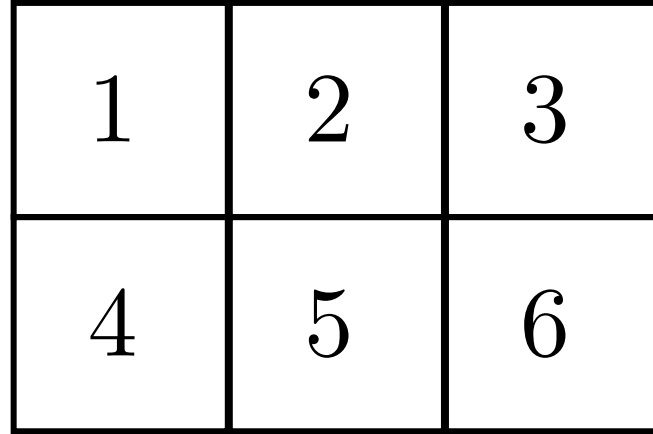
1	2	3
4	5	6

shape = (2, 3)  
ndim = 2  
row x col

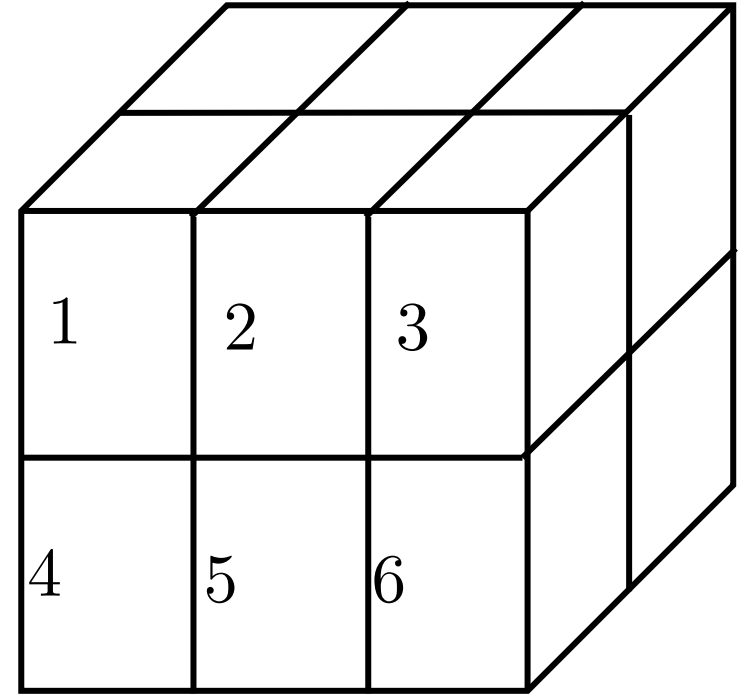
# n-Dimensional Arrays



shape = (3, )  
ndim = 1  
row

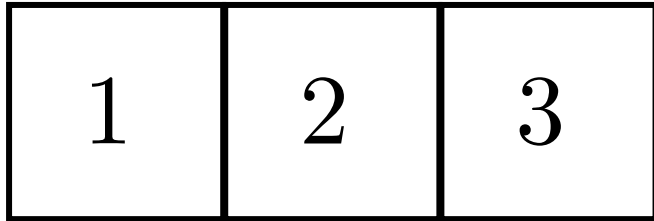


shape = (2, 3)  
ndim = 2  
row x col

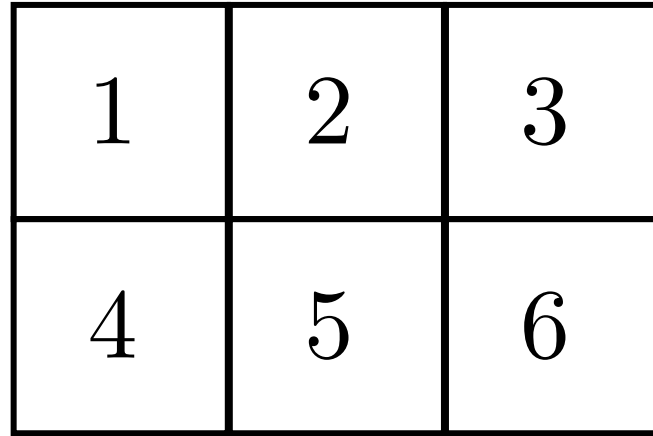


shape = (2, 2, 3)  
ndim = 3  
depth X row x col

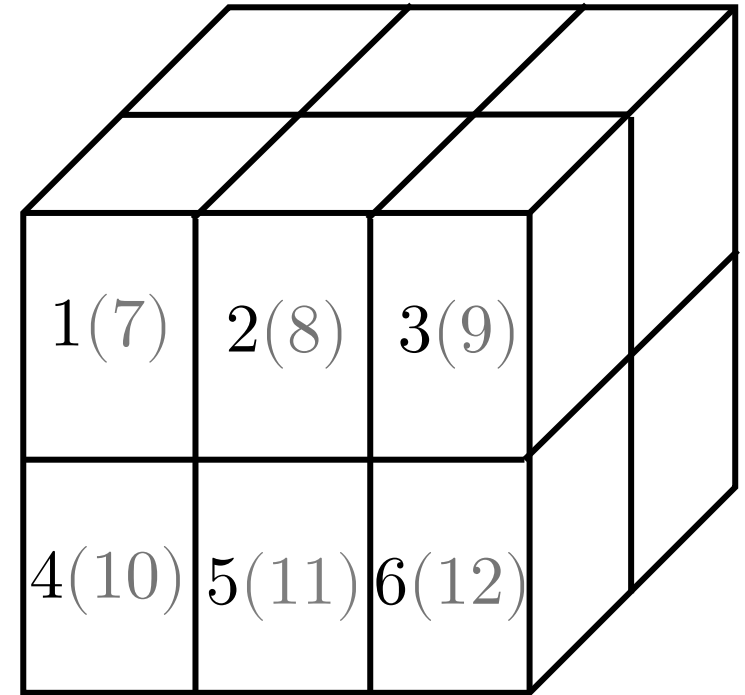
# n-Dimensional Arrays



shape = (3, )  
ndim = 1  
row



shape = (2, 3)  
ndim = 2  
row x col



shape = (2, 2, 3)  
ndim = 3  
depth X row x col

# n-Dimensional Arrays: Use Cases

ndim	shape	Object
1	100	Feature-vector
2	10 x 10	Grayscale image
3	3 x 10 x 10	RGB image
4	50 x 3 x 10 x 10	50 RGB images
5	100 x 50 x 3 x 10 x 10	100 videos 50 frames each Each frame is a 10 x 10 RGB image



# Reshape

1	2	3	4	5	6
---	---	---	---	---	---

(6, )

**default**

1	2	3
4	5	6

(2, 3)

1	3	5
2	4	6

(2, 3)

# Reshape

default

1	2	3	4	5	6
---	---	---	---	---	---

(6, )

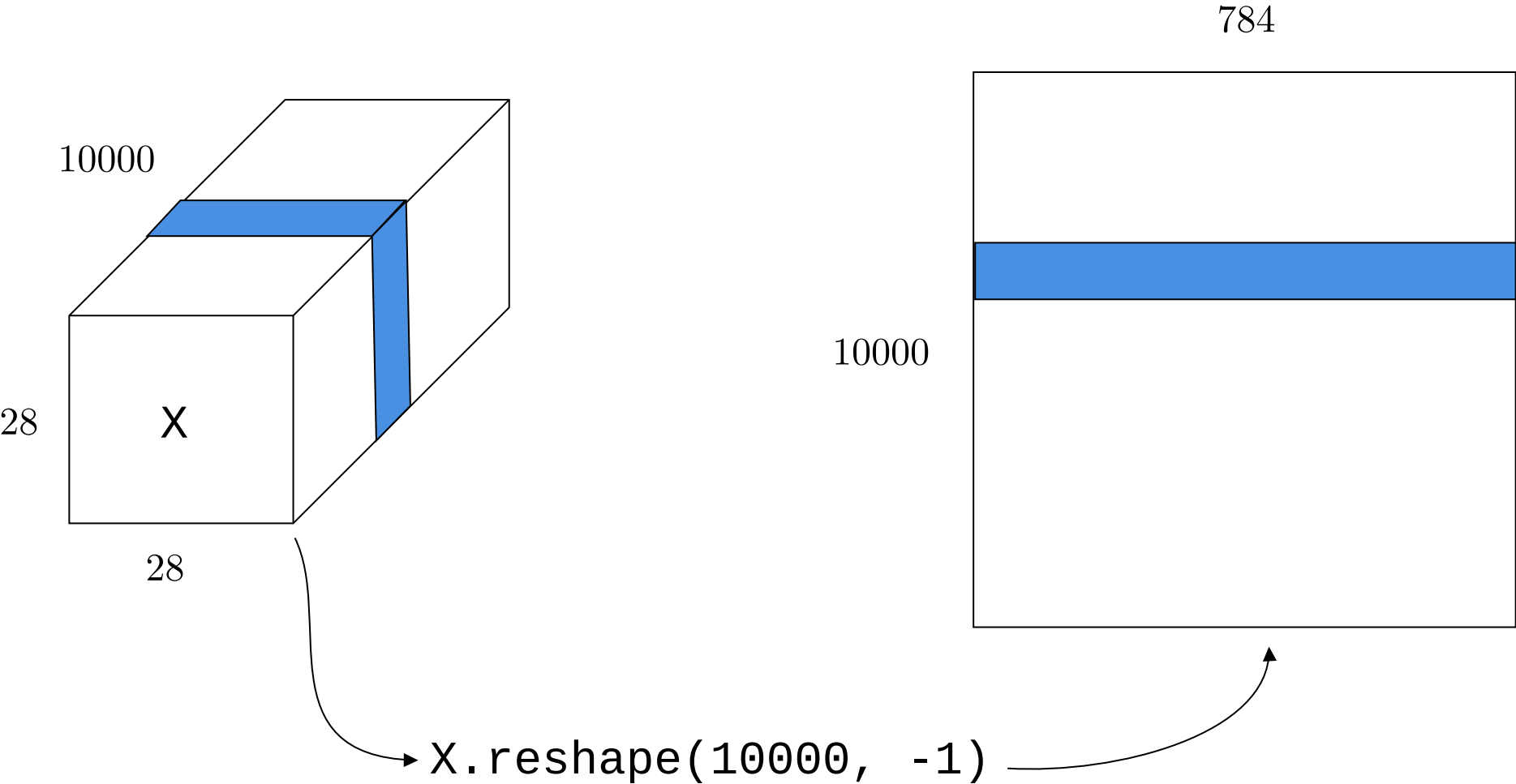
1	2	3
4	5	6

(2, 3)

(6, )

1	4	2	5	3	6
---	---	---	---	---	---

# Reshape: Use Case



Matrix-vector Addition: Row vector

1	2	3
4	5	6

+

1	2	3
---	---	---

=

2	4	6
5	7	9

Matrix-vector Addition: Column vector

1	2	3
4	5	6

+

1	2
---	---

?

Matrix-vector Addition: Column vector

1	2	3
4	5	6

+

1
2

=

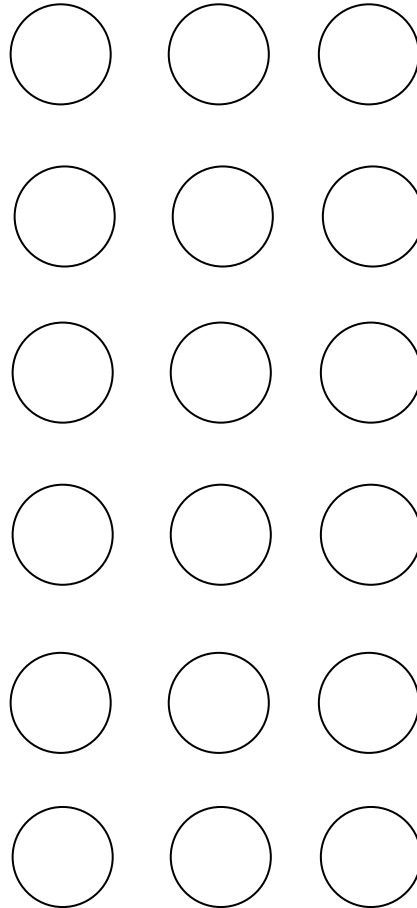
2	3	4
6	7	8

# Matrix-vector Addition: Use Case

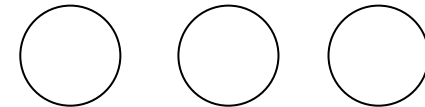
**$XW$**

**$b$**

6 samples



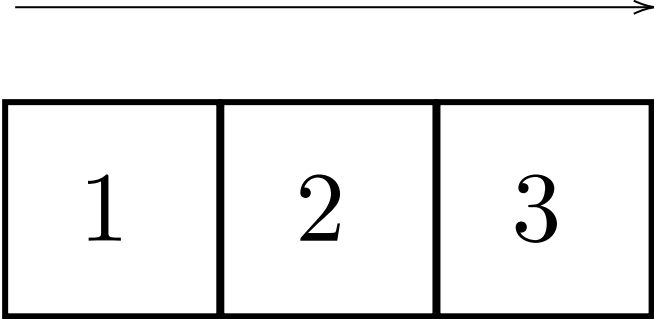
+



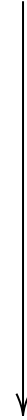
3 neurons

# Axis

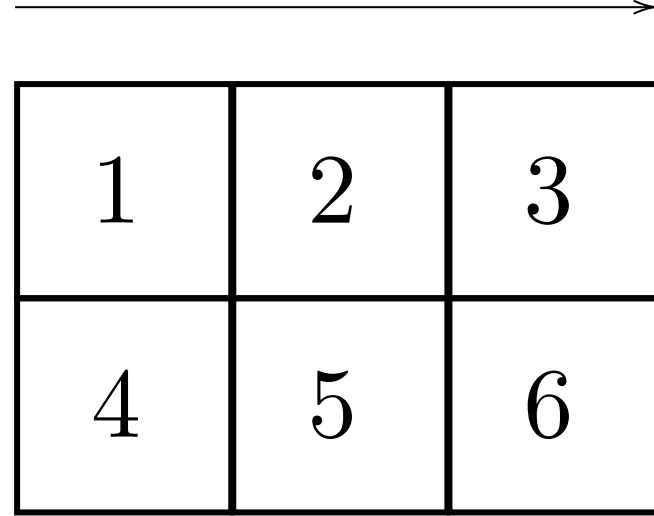
axis = 0



axis = 0

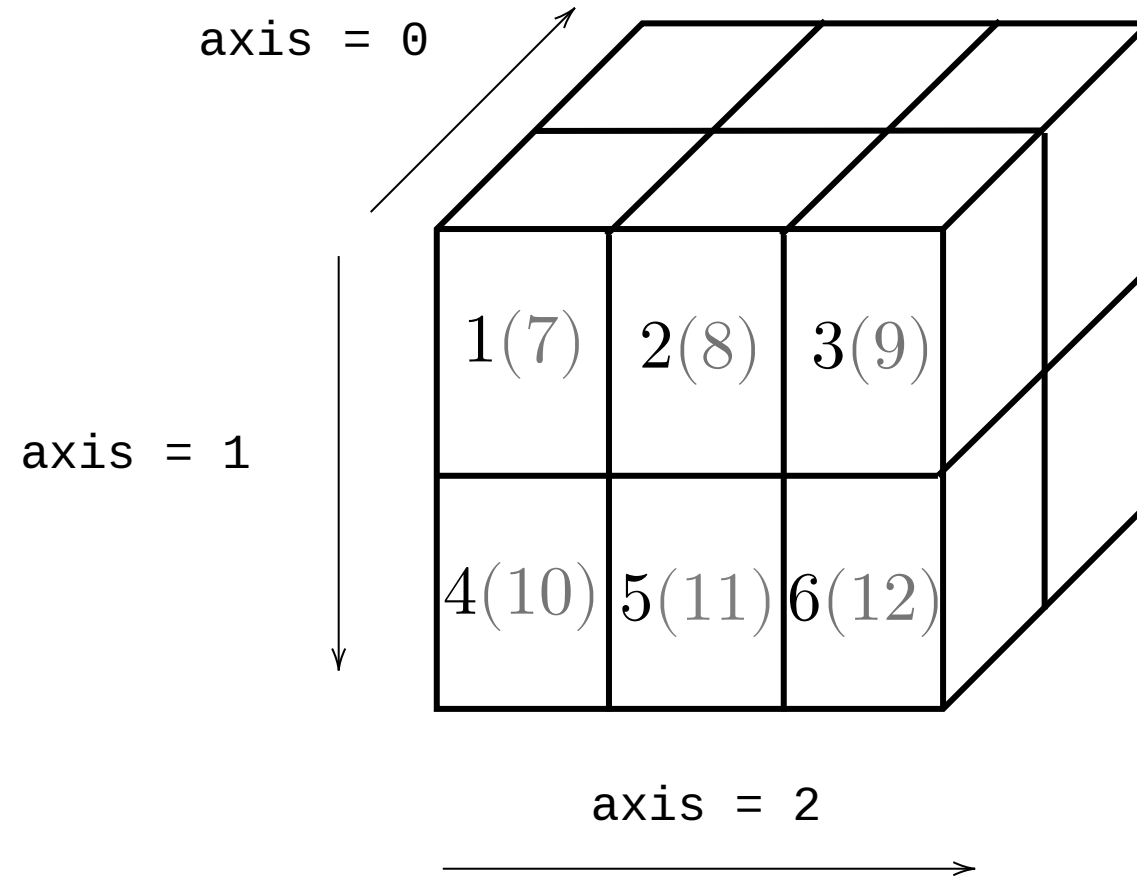


axis = 1

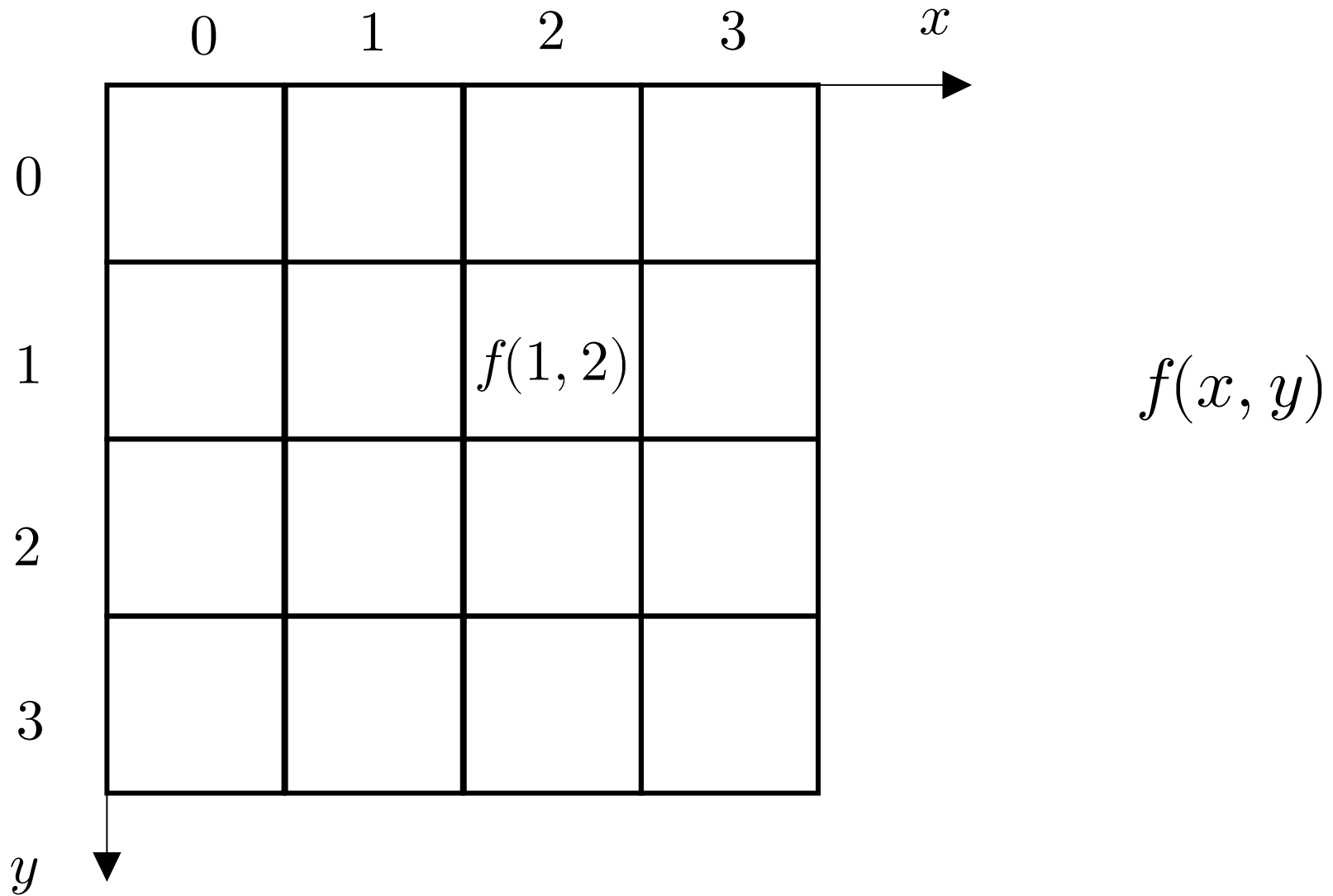




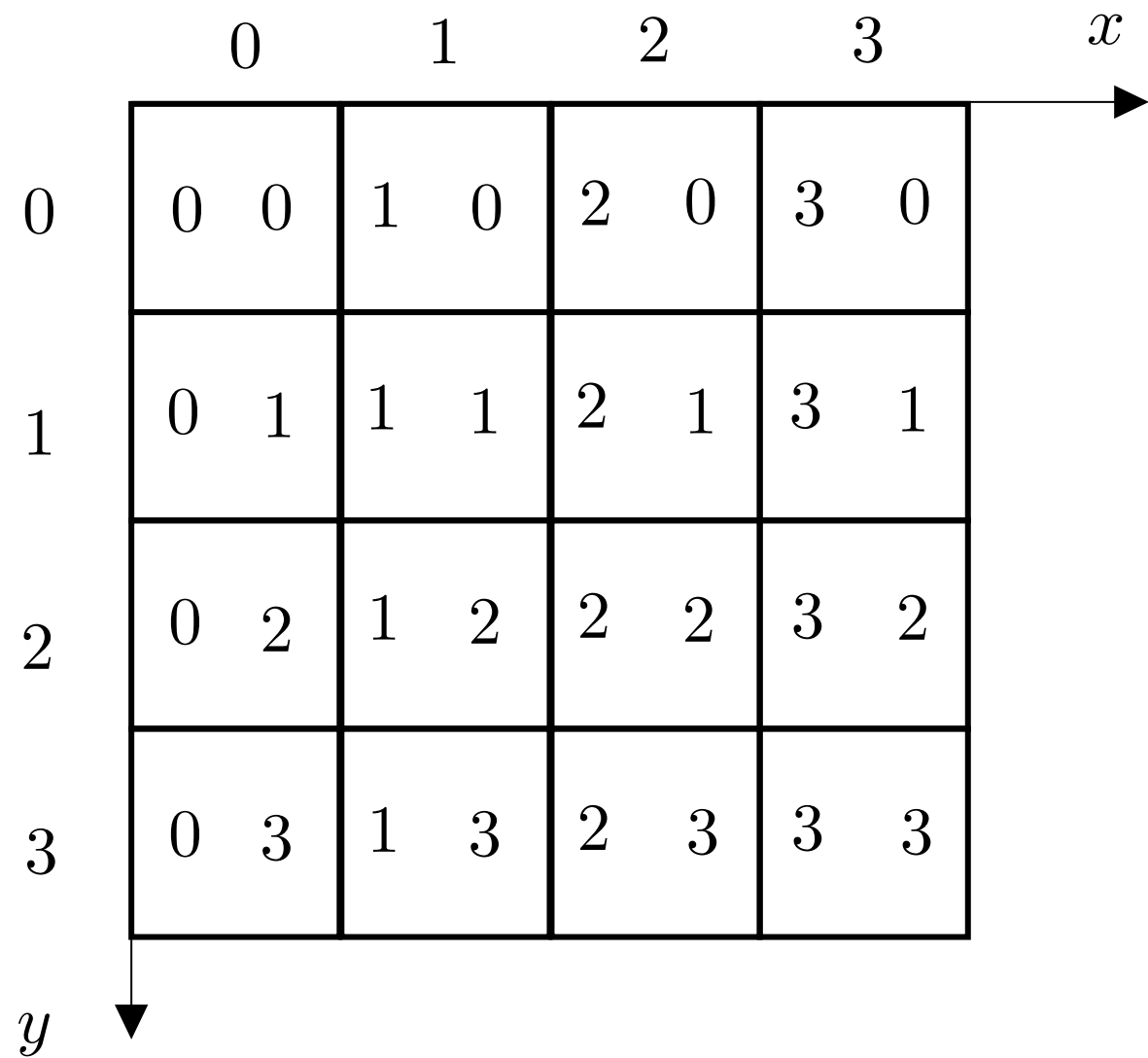
# Axis



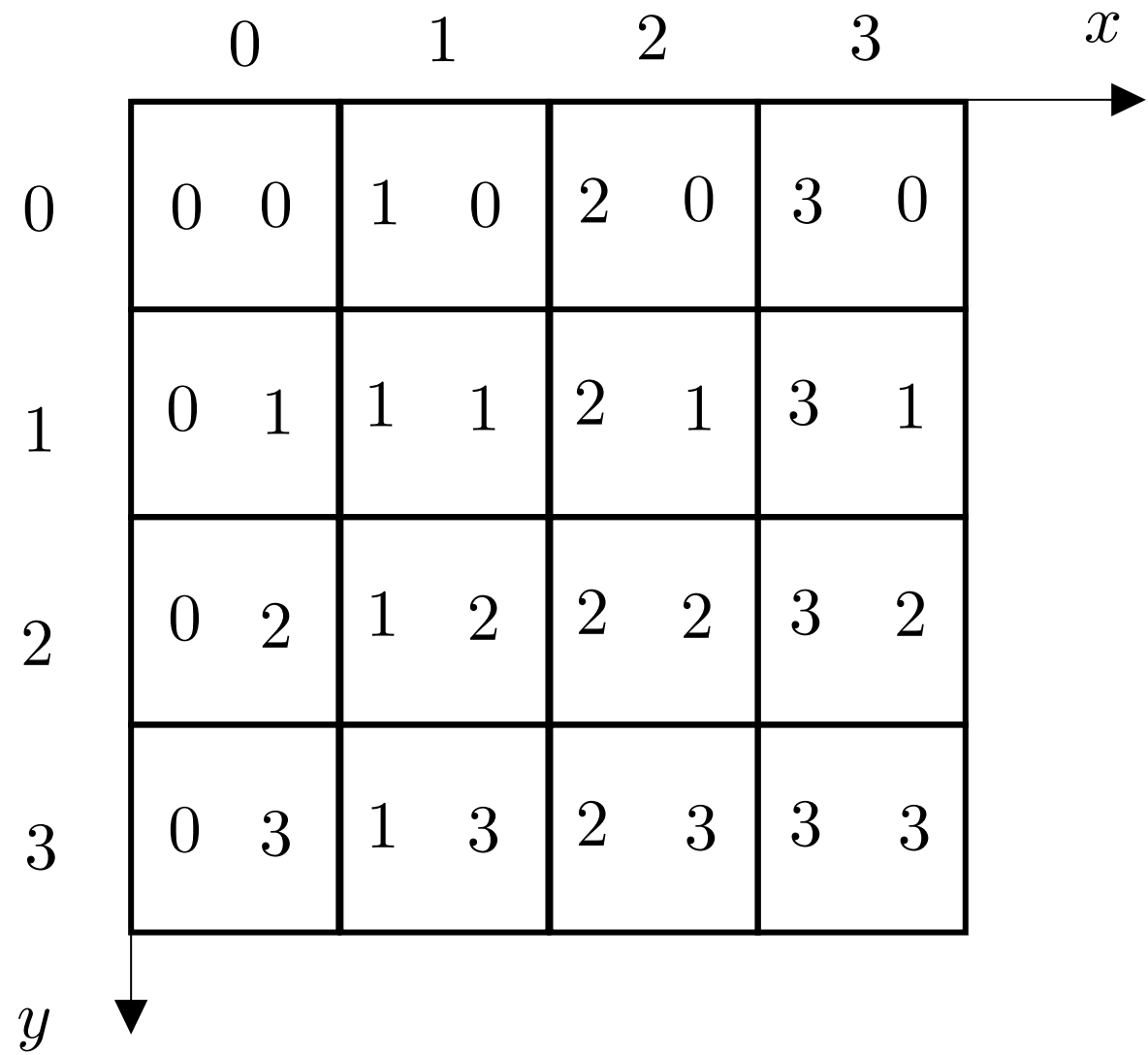
# Meshgrid



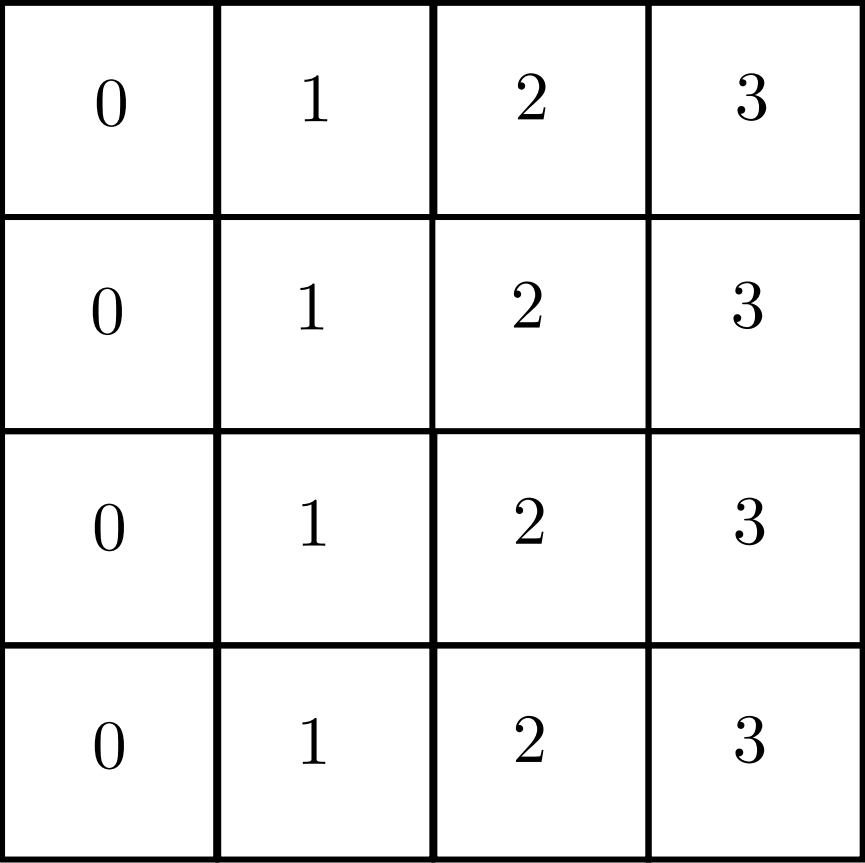
Meshgrid



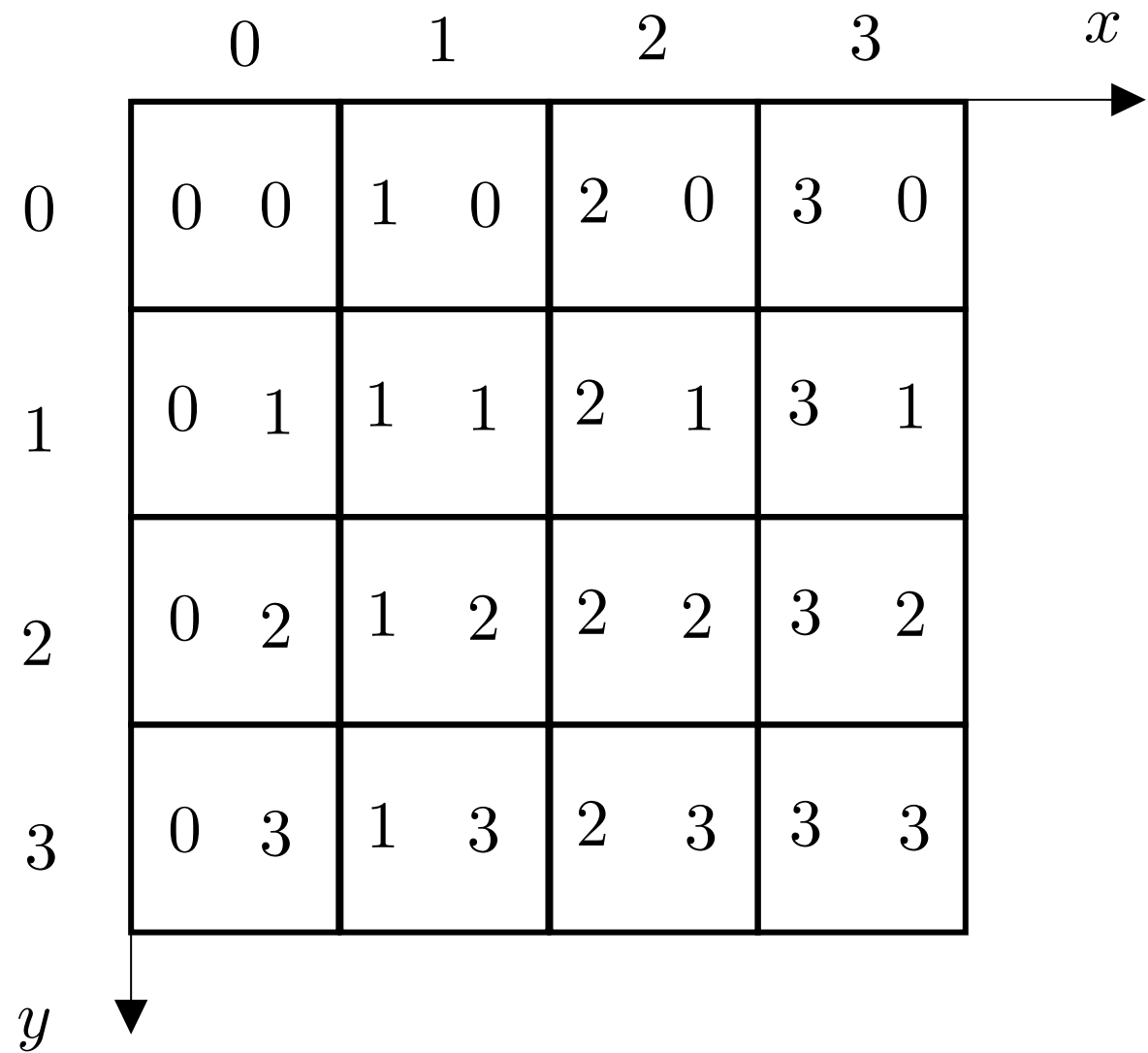
# Meshgrid



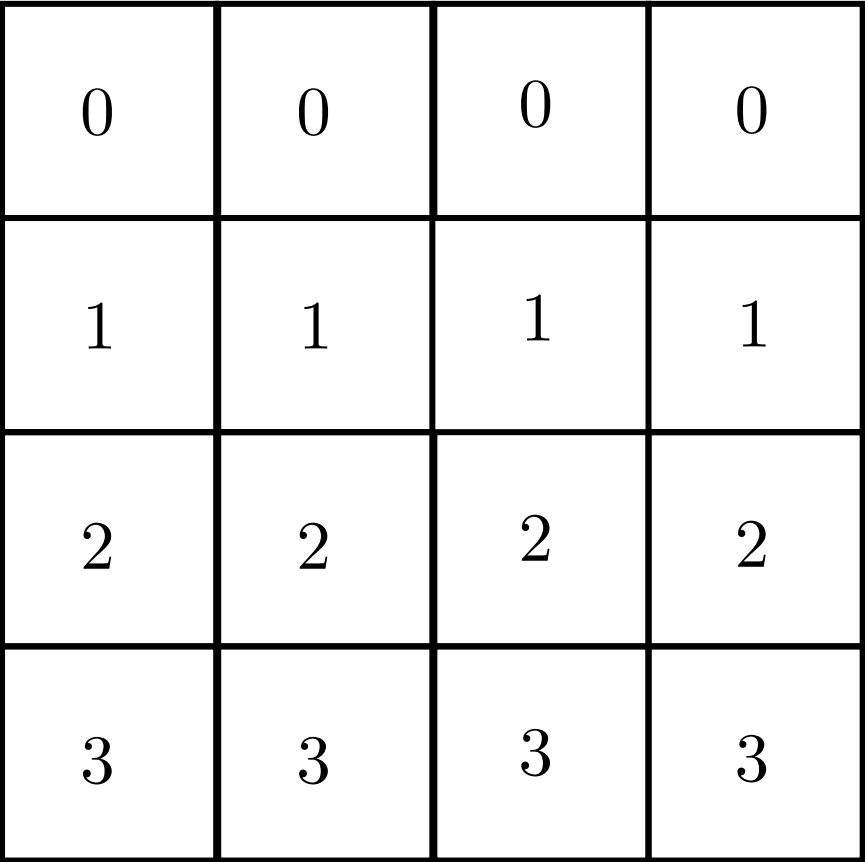
```
np.meshgrid([0, 1, 2, 3],  
            [0, 1, 2, 3])[0]
```



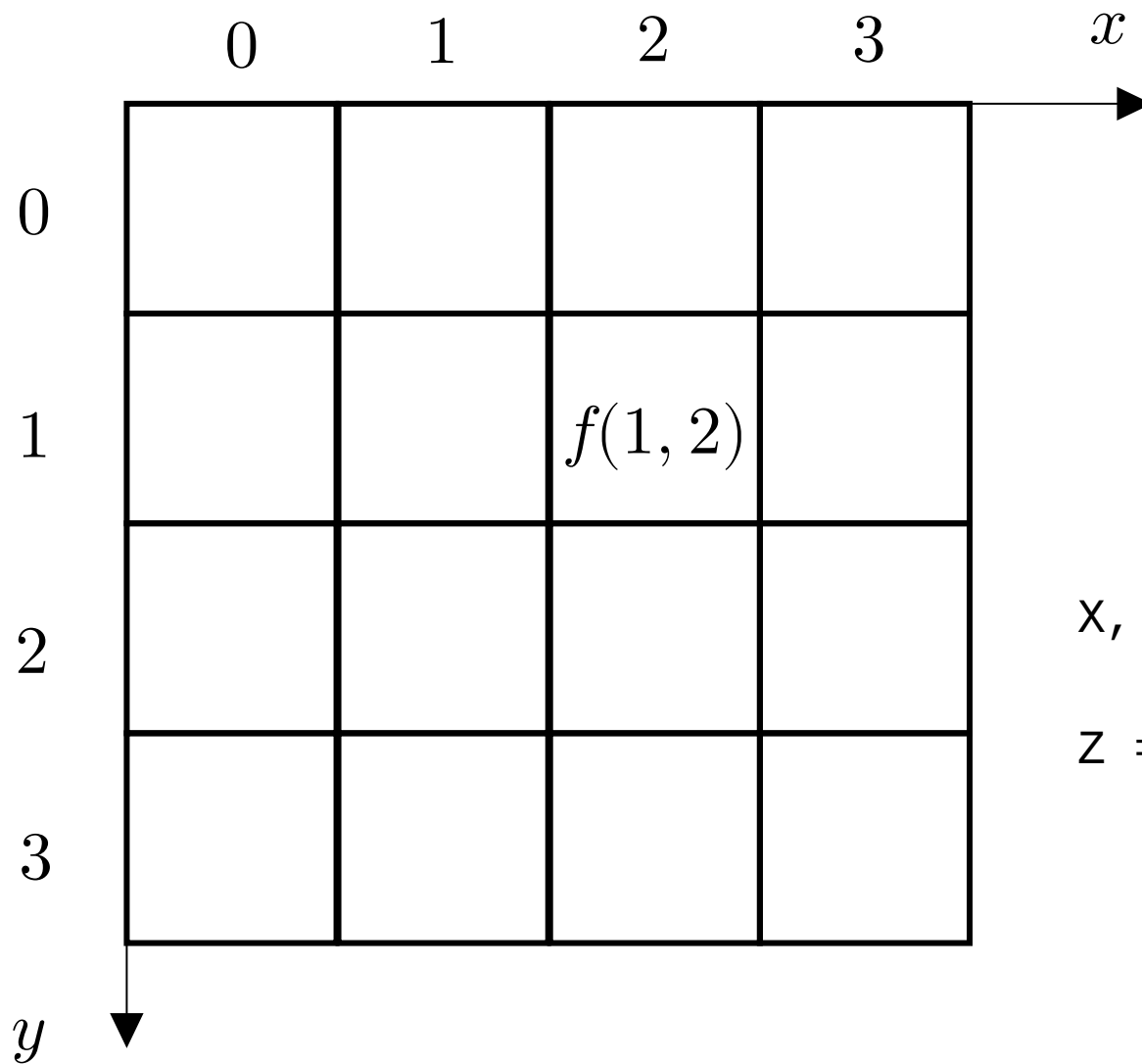
# Meshgrid



```
np.meshgrid([0, 1, 2, 3],  
            [0, 1, 2, 3])[1]
```



# Meshgrid



$$f(x, y)$$

```
X, Y = np.meshgrid([0, 1, 2, 3],  
                    [0, 1, 2, 3])
```

```
Z = f(X, Y)
```

Math and NumPy

NumPy is math done using Python

# References

- Logos for [NumPy](#), [SciPy](#) and [Matplotlib](#) from official pages
- NumPy doc on [Broadcasting](#)
- SciPy doc on [minimize](#) API
- [Code](#) for 3D plotting
- The slides were prepared using <https://www.mathcha.io/>