Chapter 2: Fundamental Mathematics for AI

Duration: 01 week

---

1. Linear Algebra

Essential Concepts :

- **Vectors**: Mathematical objects with magnitude and direction, used to represent data.

Learn NumPy

NumPy is a Python library.

NumPy is used for working with arrays.

NumPy is short for "Numerical Python".

Learning by Examples

In our "Try it Yourself" editor, you can use the NumPy module, and modify the code to see the result.

Example

Create a NumPy array:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

print(arr)

print(type(arr))
```

What is NumPy?

NumPy is a Python library used for working with arrays.

It also has functions for working in domain of linear algebra, fourier transform, and matrices.

NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely.

NumPy stands for Numerical Python.

Why Use NumPy?

In Python we have lists that serve the purpose of arrays, but they are slow to process.

NumPy aims to provide an array object that is up to 50x faster than traditional Python lists.

The array object in NumPy is called `ndarray`, it provides a lot of supporting functions that make working with `ndarray` very easy.

Arrays are very frequently used in data science, where speed and resources are very important.

Dimensions in Arrays

A dimension in arrays is one level of array depth (nested arrays).

nested array: are arrays that have arrays as their elements.

0-D Arrays

0-D arrays, or Scalars, are the elements in an array. Each value in an array is a 0-D array.

Example

Create a 0-D array with value 42

```
import numpy as np

arr = np.array(42)

print(arr)
```

1-D Arrays

An array that has 0-D arrays as its elements is called uni-dimensional or 1-D array.

These are the most common and basic arrays.

Example

Create a 1-D array containing the values 1,2,3,4,5:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

print(arr)
```

- **Matrices:** 2D arrays of numbers, fundamental for linear transformations.

2-D Arrays

An array that has 1-D arrays as its elements is called a 2-D array.

These are often used to represent matrix or 2nd order tensors.

NumPy has a whole sub module dedicated towards matrix operations called `numpy.mat`

Example

Create a 2-D array containing two arrays with the values 1,2,3 and 4,5,6:

```
import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6]])

print(arr)
```

3-D arrays

An array that has 2-D arrays (matrices) as its elements is called 3-D array.

These are often used to represent a 3rd order tensor.

Example

Create a 3-D array with two 2-D arrays, both containing two arrays with the values 1,2,3 and 4,5,6:

```
import numpy as np

arr = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])

print(arr)
```

Check Number of Dimensions?

NumPy Arrays provides the `ndim` attribute that returns an integer that tells us how many dimensions the array have.

Example

Check how many dimensions the arrays have:

```
import numpy as np

a = np.array(42)
b = np.array([1, 2, 3, 4, 5])
c = np.array([[1, 2, 3], [4, 5, 6]])
d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])

print(a.ndim)
print(b.ndim)
print(c.ndim)
print(d.ndim)
```

Access Array Elements

Array indexing is the same as accessing an array element.

You can access an array element by referring to its index number.

The indexes in NumPy arrays start with 0, meaning that the first element has index 0, and the second has index 1 etc.

Example

Get the first element from the following array:

```
import numpy as np

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

print(arr[0])
```

Example

Get third and fourth elements from the following array and add them.

```
import numpy as np

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

print(arr[2] + arr[3])
```

Access 2-D Arrays

To access elements from 2-D arrays we can use comma separated integers representing the dimension and the index of the element.

Think of 2-D arrays like a table with rows and columns, where the dimension represents the row and the index represents the column.

Example

Access the element on the first row, second column:

```
import numpy as np

arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])

print('2nd element on 1st row: ', arr[0, 1])
```

Access 3-D Arrays

To access elements from 3-D arrays we can use comma separated integers representing the dimensions and the index of the element.

Example

Access the third element of the second array of the first array:

```
import numpy as np

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

print(arr[0, 1, 2])
```

Slice elements from index 1 to index 5 from the following array:

```
import numpy as np

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

print(arr[1:5])
```

Slice elements from index 4 to the end of the array:

```
import numpy as np

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

print(arr[4:])
```

Slice elements from the beginning to index 4 (not included):

```
import numpy as np

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

print(arr[:4])
```

Use the minus operator to refer to an index from the end:

Example

Slice from the index 3 from the end to index 1 from the end:

```
import numpy as np

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

print(arr[-3:-1])
```

Return every other element from the entire array:

```
import numpy as np

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

print(arr[::2])
```

Return every other element from index 1 to index 5:

```
import numpy as np

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

print(arr[1:5:2])
```

From the second element, slice elements from index 1 to index 4 (not included)

```
import numpy as np

arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])

print(arr[1, 1:4])
```

From both elements, return index 2:

```
import numpy as np

arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])

print(arr[0:2, 2])
```

From both elements, slice index 1 to index 4 (not included), this will return a 2-D array:

```
import numpy as np

arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])

print(arr[0:2, 1:4])
```

Make a copy, change the original array, and display both arrays:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])
x = arr.copy()
arr[0] = 42

print(arr)
print(x)
```

Sort the array:

```
import numpy as np

arr = np.array([3, 2, 0, 1])

print(np.sort(arr))
```

Create an array from the elements on index 0 and 2:

```
import numpy as np

arr = np.array([41, 42, 43, 44])

x = [True, False, True, False]

newarr = arr[x]

print(newarr)
```

Sort a 2-D array:

```
import numpy as np

arr = np.array([[3, 2, 4], [5, 0, 1]])

print(np.sort(arr))
```

Iterate on the elements of the following 2-D array:

```
import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6]])

for x in arr:
  print(x)
```

Iterate on each scalar element of the 2-D array:

```
import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6]])

for x in arr:
  for y in x:
    print(y)
```

Iterate down to the scalars:

```
import numpy as np

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

for x in arr:
  for y in x:
    for z in y:
      print(z)
```

Iterate through the following 3-D array:

```
import numpy as np

arr = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])

for x in np.nditer(arr):
  print(x)
```

- **Products**:

1 Dot product (vector · vector)

Let

$$\mathbf{a} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix}$$

Definition:

$$\mathbf{a} \cdot \mathbf{b} = a_1 b_1 + a_2 b_2 + a_3 b_3$$

Calculation:

$$(1)(4) + (2)(5) + (3)(6) = 4 + 10 + 18 = \boxed{32}$$

1 Dot product (vector · vector)

Definition:

The dot product of two vectors is a **scalar** (a single number).

Example

$$\mathbf{a} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix}$$

Calculation

$$\mathbf{a} \cdot \mathbf{b} = (1 \times 4) + (2 \times 5) + (3 \times 6) = 4 + 10 + 18 = \boxed{32}$$

2 Matrix × Vector product

Let

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} 5 \\ 6 \end{bmatrix}$$

Calculation:

$$A\mathbf{x} = \begin{bmatrix} (1)(5) + (2)(6) \\ (3)(5) + (4)(6) \end{bmatrix} = \begin{bmatrix} 17 \\ 39 \end{bmatrix}$$

📌 Result is a **vector**.

---

3 Matrix × Matrix product

Let

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \quad B = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$$

Calculation:

$$AB = \begin{bmatrix} (1)(5) + (2)(7) & (1)(6) + (2)(8) \\ (3)(5) + (4)(7) & (3)(6) + (4)(8) \end{bmatrix} = \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}$$

✅ Summary Table

Operation	Input	Output
Dot product	Vector · Vector	Scalar
Matrix × Vector	Matrix, Vector	Vector
Matrix × Matrix	Matrix, Matrix	Matrix

2 Matrix × vector product

Definition:

A matrix multiplied by a vector gives anotl

vector.

Example

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} 5 \\ 6 \end{bmatrix}$$

Calculation

$$A\mathbf{x} = \begin{bmatrix} (1 \times 5 + 2 \times 6) \\ (3 \times 5 + 4 \times 6) \end{bmatrix} = \begin{bmatrix} 17 \\ 39 \end{bmatrix}$$

3 Matrix × matrix product

Definition:

Matrix multiplication produces another **matrix.**

Example

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

Calculation

$$AB = \begin{bmatrix} (1 \times 2 + 2 \times 1) & (1 \times 0 + 2 \times 5) \\ (3 \times 2 + 4 \times 1) & (3 \times 0 + 4 \times 5) \end{bmatrix} = \begin{bmatrix} 4 & 10 \\ 10 & 20 \end{bmatrix}$$

1 Dot product (vector · vector)

```python
import numpy as np

a = np.array([1, 2, 3])
b = np.array([4, 5, 6])

dot_product = np.dot(a, b)
print(dot_product)
```

2 Matrix × Vector product

```python
A = np.array([[1, 2],
              [3, 4]])

x = np.array([5, 6])

result = np.dot(A, x)
print(result)
```

3 Matrix × Matrix product

```python
B = np.array([[5, 6],
              [7, 8]])

matrix_product = np.dot(A, B)
print(matrix_product)
```

🔷 Or using the `@` operator:

```python
matrix_product = A @ B
```

🔑 Important Notes

- `np.dot()` works for vectors and matrices
- `@` is the **recommended modern syntax** for matrix multiplication
- `*` performs **element-wise multiplication**, not matrix product

- **Norms**:

🟦 What is a norm?

A **norm** measures the **size (length)** of a vector.

Let the vector:

$$x = [3, -4, 1]$$

1 L2 norm (Euclidean norm)

📐 Math

$$\|x\|_2 = \sqrt{3^2 + (-4)^2 + 1^2}$$
$$= \sqrt{9 + 16 + 1}$$
$$= \sqrt{26} \approx 5.10$$

🐍 Python

```python
import numpy as np

x = np.array([3, -4, 1])

l2 = np.linalg.norm(x, 2)
print("L2 norm:", l2)
```

📌 **Meaning**:

The **real geometric length** of the vector in space.

2 L1 norm (Manhattan norm)

📐 Math

$$\|x\|_1 = |3| + |-4| + |1|$$

$$= 3 + 4 + 1 = 8$$

🐍 Python

```python
l1 = np.linalg.norm(x, 1)
print("L1 norm:", l1)
```

📌 **Meaning:**

Distance if you move **only horizontally and vertically** (like city streets).

3 Infinity norm (Max norm)

📐 Math

$$\|x\|_\infty = \max(|3|, |-4|, |1|)$$

$$= 4$$

🐍 Python

```python
linf = np.linalg.norm(x, np.inf)
print("Infinity norm:", linf)
```

📌 **Meaning:**

The **largest absolute component** of the vector.

⚙️ Why norms are important

- 🔔 Signal magnitude
- 😈 Machine learning (regularization)
- ⚡ Control systems (error size)
- 📐 Geometry & optimization

2. Probability & Statistics

2 Types of Random Variables

A. Discrete Random Variables

- Takes **countable values** (like 0, 1, 2...).
- Often comes from counting things.

Example 1: Tossing a coin 3 times.

Let X = number of heads.

Outcome	X
HHH	3
HHT	2
HTH	2
THH	2
HTT	1
THT	1
TTH	1
TTT	0

So $X = 0, 1, 2, 3$ → discrete.

B. Continuous Random Variables

- Takes **any value in a range**.
- Often comes from measuring things.

Example 2: Height of a randomly selected person.

- Can be 160.1 cm, 160.15 cm, 160.151 cm... infinitely many possibilities.

3 Probability Distributions

A **random variable** is useless without knowing how likely its values are.

A. For Discrete RVs: Probability Mass Function (PMF)

- Gives the probability of each possible value.
- Must satisfy:
  1. $0 \leq P(X = x) \leq 1$
  2. $\sum P(X = x_i) = 1$

Example: Coin toss 3 times, X = number of heads.

X	P(X)
0	1/8
1	3/8
2	3/8
3	1/8

B. For Continuous RVs: Probability Density Function (PDF)

- $P(a \leq X \leq b) = \int_a^b f(x)dx$
- The **area under the curve** gives probabilities.
- Must satisfy:
  1. $f(x) \geq 0$ for all x
  2. $\int_{-\infty}^{\infty} f(x)dx = 1$

Example: Height X of people ~ Normal(μ=170, σ=10)

- $f(x) = \frac{1}{10\sqrt{2\pi}} e^{-\frac{(x-170)^2}{200}}$

---

4 Expectation (Mean) and Variance

A. Expectation (Mean)

- **Average value you expect in the long run**
- **Discrete:** $E[X] = \sum x \cdot P(X = x)$
- **Continuous:** $E[X] = \int_{-\infty}^{\infty} xf(x)dx$

Example (Discrete): Coin toss 3 times

$$E[X] = 0 \cdot \frac{1}{8} + 1 \cdot \frac{3}{8} + 2 \cdot \frac{3}{8} + 3 \cdot \frac{1}{8} = 1.5$$

B. Variance

- Measures **spread around the mean**
- Formula: $Var(X) = E[(X - E[X])^2]$

Example (Discrete):

$$Var(X) = (0 - 1.5)^2 \cdot \frac{1}{8} + (1 - 1.5)^2 \cdot \frac{3}{8} + \cdots = 1.125$$

---

5 Common Discrete Distributions

Distribution	Description	Example
Bernoulli	0/1 outcome	Coin toss
Binomial	Sum of n Bernoullis	# of heads in 10 tosses
Poisson	# of events in fixed time	# of calls in an hour

6 Common Continuous Distributions

Distribution	Description	Example
Uniform	All values equally likely	Random # between 0 and 1
Normal	Bell curve	Heights, errors
Exponential	Time until event	Time until a call arrives

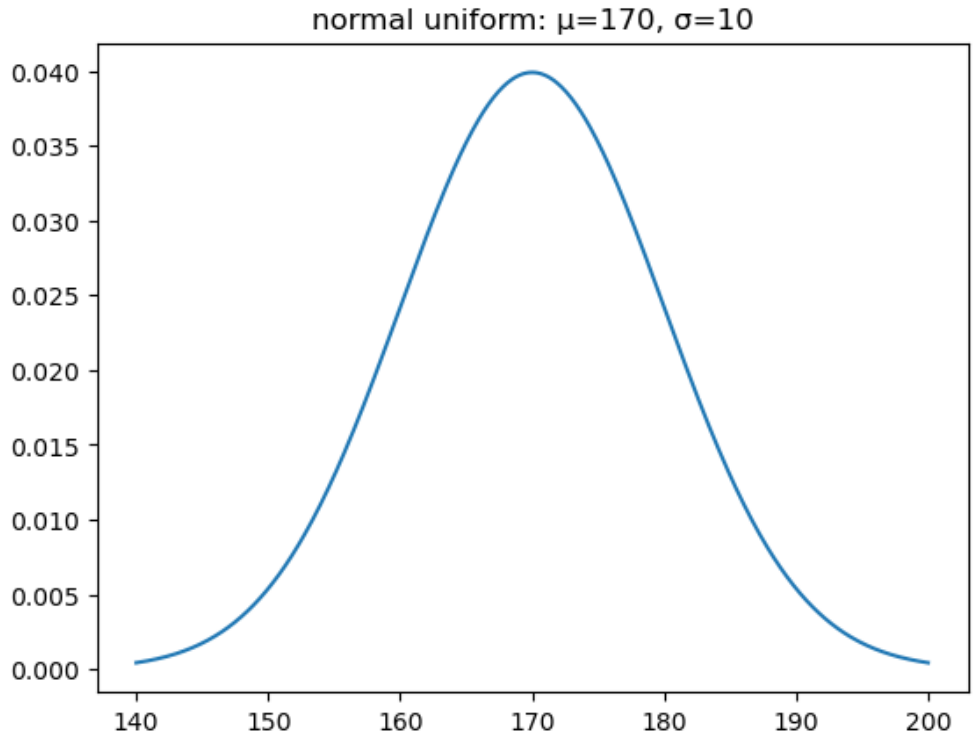
7 Python Example

python

```python
import numpy as np

# Discrete: number of heads in 3 coin tosses
X = np.array([0,1,2,3])
P = np.array([1,3,3,1])/8  # PMF

# Expected value
E_X = np.sum(X * P)
Var_X = np.sum((X - E_X)**2 * P)

print("E[X] =", E_X)   # 1.5
print("Var[X] =", Var_X) # 1.125
```

```
E[X] = 1.5
Var[X] = 0.75
```

```python
# Continuous: Normal random variable
import matplotlib.pyplot as plt
from scipy.stats import norm

x = np.linspace(140, 200, 500)
f = norm.pdf(x, loc=170, scale=10)

plt.plot(x, f)
plt.title("Normal PDF: μ=170, σ=10")
plt.show()
```



📌 Expectation (Mean) of a Random Variable

The **expectation** (or mean) is the **"average" value** you would expect if you repeated an experiment many times.

It is **denoted by** $E[X]$.

1️⃣ Discrete Random Variable

Formula:

$$E[X] = \sum x \cdot P(x)$$

- x = possible value of the random variable
- $P(x)$ = probability of that value

Example 1: Fair Dice 🎲

- Random variable X = number on the dice
- Values: $1, 2, 3, 4, 5, 6$
- Each has probability $P(x) = 1/6$

$$E[X] = 1 * (1/6) + 2 * (1/6) + 3 * (1/6) + 4 * (1/6) + 5 * (1/6) + 6 * (1/6) = 21/6 = 3.5$$

Python Example:

```python
import numpy as np

x = np.array([1,2,3,4,5,6])
p = np.array([1/6]*6)  # probabilities

E_X = np.sum(x * p)
print("Expected value (discrete):", E_X)
```

Example 2: Biased Coin

- Random variable $X = 1$ if heads, 0 if tails
- Probability of heads $P(H) = 0.7$, tails $P(T) = 0.3$

$$E[X] = 0 * 0.3 + 1 * 0.7 = 0.7$$

✅ Interpretation: Average outcome per toss = 0.7 (closer to head because coin is biased)

```python
x = np.array([0, 1])
p = np.array([0.3, 0.7])
E_X = np.sum(x * p)
print("Expected value of biased coin:", E_X)
```

2 Continuous Random Variable

Formula:

$$E[X] = \int_{-\infty}^{\infty} x \cdot f(x)\, dx$$

- $f(x)$ = **probability density function (PDF)**
- The integral sums the "weighted values" over all possible x

---

Example: Uniform Distribution $U(0, 1)$

- PDF: $f(x) = 1$ for $0 \leq x \leq 1$
- Expectation:

$$E[X] = \int_0^1 x \cdot 1\, dx = \frac{x^2}{2}\Big|_0^1 = 0.5$$

✅ Interpretation: Average value between 0 and 1 = 0.5

Python Simulation:

```python
samples = np.random.uniform(0, 1, 100000)
E_X = samples.mean()
print("Expected value (continuous):", E_X)
```

🔑 Intuition

- **Expectation = weighted average**
- Each value is weighted by its probability
- Discrete → sum of probabilities × values
- Continuous → integral of values × PDF

Think of it as the **"center of mass"** of the probability distribution.

📌 Variance — Measure of Spread

Variance tells us **how much the values of a random variable deviate from the mean.**

It is **denoted by** $\mathrm{Var}(X)$.

$$\mathrm{Var}(X) = E[(X - E[X])^2]$$

- X = random variable
- $E[X]$ = mean (expectation)
- Squaring ensures deviations are **always positive**

1️⃣ Discrete Random Variable

Formula:

$$\mathrm{Var}(X) = \sum (x - E[X])^2 \cdot P(x)$$

- Each deviation $(x - E[X])$ is weighted by its probability

Example 1: Fair Dice 🎲

- Values: $X = 1, 2, 3, 4, 5, 6$
- Probability: $P(x) = 1/6$
- Mean: $E[X] = 3.5$

$$\mathrm{Var}(X) = \sum (x - 3.5)^2 * \frac{1}{6}$$

Step by step:

x	x - E[X]	$(x-E[X])^2$	$(x-E[X])^2 * P(x)$
1	-2.5	6.25	1.0417
2	-1.5	2.25	0.375
3	-0.5	0.25	0.0417
4	0.5	0.25	0.0417
5	1.5	2.25	0.375
6	2.5	6.25	1.0417

$$\text{Var}(X) = 2.9167 \approx 2.92$$

Python Example:

```python
import numpy as np

x = np.array([1,2,3,4,5,6])
p = np.array([1/6]*6)
E_X = np.sum(x * p)
Var_X = np.sum((x - E_X)**2 * p)
print("Variance (discrete):", Var_X)
```

Example 2: Biased Coin

- X = 1 (head), 0 (tail)
- P(1)=0.7, P(0)=0.3
- Mean: E[X]=0.7

$$\text{Var}(X) = (0-0.7)^2 * 0.3 + (1-0.7)^2 * 0.7 = 0.21$$

```python
x = np.array([0,1])
p = np.array([0.3,0.7])
E_X = np.sum(x*p)
Var_X = np.sum((x - E_X)**2 * p)
print("Variance (biased coin):", Var_X)
```

---

2 Continuous Random Variable

Formula:

$$\text{Var}(X) = \int_{-\infty}^{\infty} (x - E[X])^2 \cdot f(x)\, dx$$

- $f(x)$ = probability density function (PDF)

Example: Uniform Distribution $U(0,1)$

- PDF: $f(x) = 1$
- Mean: $E[X] = 0.5$

$$\text{Var}(X) = \int_0^1 (x - 0.5)^2 dx = \left[\frac{(x-0.5)^3}{3}\right]_0^1 = \frac{1}{12} \approx 0.0833$$

Python Simulation:

```python
samples = np.random.uniform(0,1,100000)
Var_X = ((samples - samples.mean())**2).mean()
print("Variance (continuous):", Var_X)
```

Intuition

1. **Variance = average squared deviation from the mean**
2. Small variance → values are close to mean
3. Large variance → values are widely spread

Quick Summary

Random Variable	Formula	Interpretation
Discrete	Var(X) = Σ (x-E[X])² P(x)	Weighted average of squared deviations
Continuous	Var(X) = ∫ (x-E[X])² f(x) dx	Area under curve of squared deviations
Dice	2.92	Moderate spread around 3.5
Biased Coin	0.21	Small spread around 0.7
Uniform(0,1)	1/12 ≈ 0.0833	Values evenly spread between 0 and 1

📊 Common Probability Distributions

🔵 1) Normal Distribution — $\mathcal{N}(\mu, \sigma^2)$

🔷 What it is

A **continuous**, **bell-shaped** distribution centered at the mean μ.

$$X \sim \mathcal{N}(\mu, \sigma^2)$$

- μ: mean (center)
- σ^2: variance (spread)

🔷 Why it is important

✅ **Error modeling**
Most measurement errors are small and symmetric around 0.

✅ **Central Limit Theorem (CLT)**
The **sum or average of many independent random variables** tends to follow a normal distribution, **even if the original variables are not normal.**

🔷 Probability Density Function (PDF)

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

🔷 Intuition

- Values near μ are **very likely**
- Values far from μ are **rare**
- About:
  - 68% of data in $\mu \pm \sigma$
  - 95% in $\mu \pm 2\sigma$

🔷 Example: Sensor Error

- Mean error = 0
- Standard deviation = 0.2

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

errors = np.random.normal(0, 0.2, 10000)
plt.hist(errors, bins=50)
plt.title("Normal Distribution: Sensor Error")
plt.show()
```

📌 Applications

- Measurement noise
- Control systems
- Kalman filters
- Machine learning loss errors

🟢 2) Binomial Distribution — $B(n, p)$

🔷 What it is

A **discrete** distribution that counts the **number of successes** in n independent trials.

$$X \sim B(n, p)$$

- n: number of trials
- p: probability of success

🔷 Probability Mass Function (PMF)

$$P(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$$

🔷 Mean and Variance

$$E[X] = np \quad , \quad \mathrm{Var}(X) = np(1-p)$$

🔷 Intuition

- Each trial = **success or failure**
- You count how many successes occurred

🔷 Example: Binary Classification

- A classifier predicts correctly with probability $p = 0.9$
- Test set size $n = 20$

```python
samples = np.random.binomial(20, 0.9, 10000)
print("Mean:", samples.mean())
print("Variance:", samples.var())
```

📌 Applications

- Pass/fail tests
- Quality control
- Binary classification accuracy
- Communication systems (bit errors)

🟠 3) Uniform Distribution — $U(a, b)$

🔷 What it is

A **continuous** distribution where **all values in [a, b] are equally likely.**

$$X \sim U(a, b)$$

🔷 Probability Density Function (PDF)

$$f(x) = \frac{1}{b-a}, \quad a \le x \le b$$

🔷 Mean and Variance

$$E[X] = \frac{a+b}{2}$$

$$\mathrm{Var}(X) = \frac{(b-a)^2}{12}$$

🔷 Intuition

- No value is preferred
- Pure randomness with equal chance

🔷 Example: Weight Initialization (Neural Networks)

```python
weights = np.random.uniform(-0.5, 0.5, 1000)
print("Mean:", weights.mean())
print("Variance:", weights.var())
```

📌 Applications

- Random sampling
- Monte Carlo simulations
- Neural network initialization
- Optimization algorithms

🔁 Comparison Summary

Distribution	Type	Mean	Variance	Typical Use
Normal	Continuous	μ	σ^2	Noise, errors
Binomial	Discrete	np	$np(1-p)$	Success/failure
Uniform	Continuous	$\frac{a+b}{2}$	$\frac{(b-a)^2}{12}$	Sampling