

# Fondations Mathématiques & Outils Python

Jour 1 — Matin

Julien Rolland

Formation M2 Développement Fullstack

Jour 1

- 1 Algèbre linéaire
- 2 Probabilités (express)
- 3 Pourquoi Python pour l'IA ?
- 4 NumPy & Vectorisation
- 5 Pandas — Introduction pratique

## Scalaire $x \in \mathbb{R}$

Un nombre unique.

Ex : taux d'apprentissage

$\alpha = 0.01$

## Vecteur $\mathbf{x} \in \mathbb{R}^n$

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

Ex : features d'un exemple.

## Matrice $\mathbf{X} \in \mathbb{R}^{m \times n}$

$$\mathbf{X} = \begin{pmatrix} x_{11} & \cdots & x_{1n} \\ \vdots & \ddots & \vdots \\ x_{m1} & \cdots & x_{mn} \end{pmatrix}$$

Ex : dataset ( $m$  exemples,  $n$  features).

## Convention de notation

Scalars :  $x$  (italique minuscule)    Vectors :  $\mathbf{x}$  (gras minuscule)    Matrices :  $\mathbf{X}$  (gras majuscule)

## Produit scalaire $\langle \mathbf{u}, \mathbf{v} \rangle$

$$\mathbf{u} \cdot \mathbf{v} = \sum_{i=1}^n u_i v_i = \mathbf{u}^\top \mathbf{v}$$

*Mesure de similarité / projection.*

## Norme euclidienne

$$\|\mathbf{x}\| = \sqrt{\mathbf{x}^\top \mathbf{x}} = \sqrt{\sum_i x_i^2}$$

## Produit matriciel $\mathbf{X} \in \mathbb{R}^{m \times k}$ , $\mathbf{Y} \in \mathbb{R}^{k \times n}$

$$(\mathbf{XY})_{ij} = \sum_{l=1}^k X_{il} Y_{lj}$$

### Attention

$\mathbf{XY} \neq \mathbf{YX}$  en général !

Les dimensions intérieures doivent correspondre.

## Transposée $\mathbf{X}^\top$

$$(\mathbf{X}^\top)_{ij} = X_{ji}$$

Propriété utile :

$$(\mathbf{XY})^\top = \mathbf{Y}^\top \mathbf{X}^\top$$

## Inverse $\mathbf{X}^{-1}$ (matrice carrée)

$$\mathbf{XX}^{-1} = \mathbf{X}^{-1}\mathbf{X} = \mathbf{I}$$

### Application directe

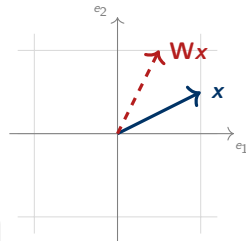
Solution du système  $\mathbf{X}\mathbf{w} = \mathbf{y}$  :

$$\mathbf{w} = \mathbf{X}^{-1}\mathbf{y}$$

Utilisé dans la régression linéaire.

Multiplier par une matrice **transforme l'espace** :

- **Rotation** — change l'orientation
- **Mise à l'échelle** — étire ou compresse
- **Projection** — réduit la dimension



## Pourquoi ça nous importe ?

Un réseau de neurones = **composition** de transformations linéaires entrecoupées de non-linéarités.

$$h = \sigma(Wx + b)$$

## Variable aléatoire $X$

Prend des valeurs selon une **distribution**  $p(x)$ .

**Espérance** — valeur moyenne

$$\mathbb{E}[X] = \sum_x x p(x)$$

**Variance** — dispersion autour de la moyenne

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

## Exemple concret

$X$  = erreur de prédiction d'un modèle.

$\mathbb{E}[X] \approx 0 \Rightarrow$  modèle non-biaisé.

$\text{Var}(X)$  faible  $\Rightarrow$  prédictions stables.

**Écart-type**  $\sigma = \sqrt{\text{Var}(X)}$

Dans la même unité que  $X$  — plus interprétable.

## Bernoulli $X \sim \mathcal{B}(p)$

$$P(X = 1) = p, \quad P(X = 0) = 1 - p$$

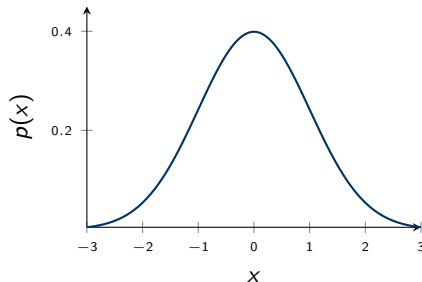
*Utilisée pour* : classification binaire (sortie d'un neurone sigmoïde).

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

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

*Utilisée pour* : initialisation des poids, bruit, régression.

Gaussienne  $\mathcal{N}(0, 1)$





## Idée fondatrice du ML supervisé

On cherche les paramètres  $\mathbf{w}$  qui rendent les données **les plus probables** :

$$\mathbf{w}^* = \arg \max_{\mathbf{w}} p(\mathcal{D} \mid \mathbf{w}) \iff \mathbf{w}^* = \arg \min_{\mathbf{w}} \mathcal{L}(\mathbf{w})$$

**Log-vraisemblance** (plus pratique)

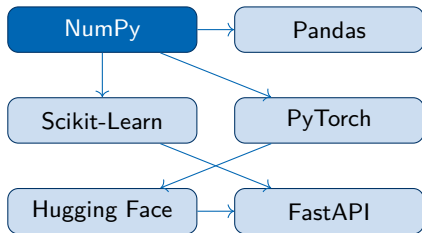
$$\ell(\mathbf{w}) = \log p(\mathcal{D} \mid \mathbf{w}) = \sum_i \log p(y_i \mid \mathbf{x}_i, \mathbf{w})$$

Maximiser  $\ell$  = minimiser une **loss**.

## Lien avec la MSE

Si le bruit est gaussien, maximiser la vraisemblance revient à minimiser :

$$\mathcal{L}(\mathbf{w}) = \frac{1}{m} \sum_{i=1}^m (y_i - \hat{y}_i)^2$$



## Pourquoi Python ?

- Syntaxe accessible — focus sur les idées
- Bibliothèques C/CUDA sous le capot  
⇒ performance native
- Standard de facto en recherche et industrie

## Ce cours

NumPy → Pandas → Sklearn → PyTorch  
→ Hugging Face → FastAPI

Un notebook = **document interactif** mêlant code, résultats et texte.

## Deux types de cellules :

- **Code** — exécuté par le *kernel* Python, affiche le résultat juste en dessous
- **Markdown** — texte formaté, formules  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ , titres, explications

## Workflow

Écrire → Shift+Enter → voir le résultat  
Les variables persistent entre les cellules.

## Attention — ordre d'exécution

Les cellules peuvent être exécutées dans n'importe quel ordre. Toujours relancer **Kernel** → **Restart & Run All** avant de rendre un travail.

In

```
import numpy as np
```

In

```
x = np.arange(5)  
print(x ** 2)
```

```
[ 0  1  4  9 16 ]
```

Md

```
# cellule Markdown  
*Résultat **: vecteur au carré
```

## Approche naïve Python

```
1 def dot_loop(u, v):  
2     result = 0.0  
3     for u_i, v_i in zip(u, v):  
4         result += u_i * v_i  
5     return result
```

## Approche vectorisée NumPy

```
1 import numpy as np  
2  
3 # Produit scalaire -- vectorise  
4 def dot_numpy(u, v):  
5     return np.dot(u, v)
```

### Problème

Python interprète chaque itération.

Pour  $n = 10^6$  : **~200 ms**

### Résultat

Exécuté en C/BLAS.

Pour  $n = 10^6$  : **~1 ms**

**≈ 200× plus rapide**

## Règle d'or

En NumPy/PyTorch : **aucune boucle for sur les éléments**. Toujours chercher l'opération matricielle équivalente.

```
1 import numpy as np
2
3 x = np.array([[1, 2, 3],
4               [4, 5, 6]])
5
6 print(x.shape)      # (2, 3)
7 print(x.ndim)       # 2  <- rank
8 print(x.dtype)      # int64
9 print(x.size)       # 6
```

Attribut	Signification
shape	dimensions ( $m, n, \dots$ )
ndim	nombre d'axes (rank)
dtype	type des éléments
size	nombre total d'éléments

## Rangs courants

<b>0D</b> scalaire	<b>1D</b> vecteur
<b>2D</b> matrice	<b>3D</b> batch d'images

## Slicing

```
1 X = np.arange(12).reshape(3, 4)
2 # [[ 0  1  2  3]
3 #   [ 4  5  6  7]
4 #   [ 8  9 10 11]]
5
6 X[0]          # 1e ligne -> [0,1,2,3]
7 X[:, 2]       # 3e colonne
8 X[1:, 1:3]    # sous-matrice
```

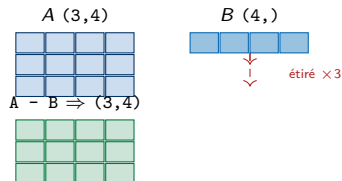
## Reshape / Vue

```
1 v = np.arange(6)          # shape (6,)
2 M = v.reshape(2, 3)       # shape (2,3)
3 F = M.flatten()           # retour (6,)
4
5 # -1 = "calcule toi-meme"
6 v.reshape(3, -1)          # (3, 2)
```

## Attention

reshape retourne une **vue** (pas de copie).  
Modifier la vue modifie l'original.

NumPy aligne les shapes **depuis la droite**.  
Toute dimension de taille **1** est automatiquement étirée.



```
1 X = np.random.randn(100, 5)
2 # shape (100, 5)
3
4 mean = X.mean(axis=0)
5 # shape (5,) -- une moyenne par feature
6
7 X_centered = X - mean
8 # (100,5) - (5,) => (100,5)
9 # soustrait la meme moyenne
10 # sur chaque ligne automatiquement
11
12 std = X.std(axis=0)
13 X_norm = (X - mean) / std
```

## NumPy ndarray

- Un seul type (dtype)
- Accès par indices entiers
- Ultra-rapide pour le calcul
- Pas de métadonnées

```
1 x[0, 2]           # ligne 0, col 2
```

## Pandas DataFrame

- Colonnes hétérogènes (int, float, str...)
- Accès par **noms de colonnes**
- Index temporel possible
- Idéal pour l'exploration / nettoyage

```
1 df['age']          # colonne 'age'  
2 df.loc[0, 'age']  # ligne 0, col age
```

## Flux de travail typique

Charger avec **Pandas** → Nettoyer → Convertir en **NumPy** / Tensor pour le modèle.



```
1 import pandas as pd
2
3 df = pd.read_csv('data.csv')
4
5 df.shape           # (nb_lignes, nb_colonnes)
6 df.dtypes          # types de chaque colonne
7 df.head(5)         # 5 premières lignes
8 df.describe()      # stats : mean, std, min, max...
9 df.info()          # types + valeurs manquantes
```

## Sélection

```
1 df[['col1', 'col2']]      # sous-df
2 df[df['age'] > 18]         # filtre
3 df.groupby('ville').mean() # agregation
```

## Passage à NumPy

```
1 X = df[features].values   # ndarray
2 y = df['target'].values   # ndarray 1D
```

## Gérer les valeurs manquantes (NaN)

```
1 df.isna().sum() # nb NaN par colonne
2 df.dropna() # supprimer les lignes avec
   NaN
3 df.fillna(df.mean()) # remplacer par la
   moyenne
4 df['col'].fillna('N/A') # valeur par
   défaut
```

## One-Hot Encoding

```
1 pd.get_dummies(df, columns=['ville'])
2 # 'ville' -> ville_Paris, ville_Lyon...
```

## Pourquoi c'est crucial

La majorité du temps en ML est passée ici.

Un modèle entraîné sur des données mal nettoyées sera systématiquement biaisé — **garbage in, garbage out.**

## Règle pratique

Toujours inspecter `df.info()` et `df.describe()` **avant** de toucher aux modèles.

### Mathématiques

- Vecteurs, matrices, produit, transposée
- Espérance, variance, distributions  $\mathcal{B}$ ,  $\mathcal{N}$
- Vraisemblance  $\leftrightarrow$  fonction de coût

### Python

- Boucles `for`  $\rightarrow$  opérations vectorisées
- `ndarray` : `shape`, `rank`, `dtype`, `slicing`, `broadcasting`
- `Pandas` : chargement, exploration, nettoyage