Structured Arrays & Record Arrays in NumPy

28 novembre 2025

Résumé

Ce document explore les **Structured Arrays** et **Record Arrays** de NumPy, des structures de données avancées permettant de manipuler des données hétérogènes de manière efficace. Nous verrons leur intérêt par rapport aux approches traditionnelles et leurs applications pratiques.

Table des matières

1 Introduction aux Tableaux Structurés

1.1 Définition

Un **tableau structuré** (Structured Array) est un tableau NumPy dont chaque élément est une structure composée de plusieurs champs de types différents. C'est l'équivalent NumPy d'une table de base de données ou d'un DataFrame simple.

1.2 Avantages Principaux

— Stockage de données hétérogènes dans un seul tableau
— Accès efficace par champ
— Opérations vectorisées sur des sous-ensembles de données
— Meilleure performance que les structures Python pures

2 Création et Manipulation de Base

2.1 Définition des Types

```python
import numpy as np

# D finition d'un type structur
dtype = [('nom', 'U10'), ('age', 'i4'), ('salaire', 'f8')]

# Cr ation du tableau structur
employes = np.array([
    ('Alice', 30, 55000.0),
    ('Bob', 25, 45000.0),
    ('Charlie', 35, 60000.0),
    ('Diana', 28, 52000.0),
    ('Eve', 40, 48000.0)
], dtype=dtype)

print("Tableau structur  complet:")
print(employes)
print(f"\nShape: {employes.shape}, Dtype: {employes.dtype}")
```

Sortie :

```
Tableau structuré complet:
[('Alice', 30, 55000.) ('Bob', 25, 45000.) ('Charlie', 35, 60000.)
```

```
 ('Diana', 28, 52000.) ('Eve', 40, 48000.)]

Shape: (5,), Dtype: [('nom', '<U10'), ('age', '<i4'), ('salaire', '<f8')]
```

2.2 Accès aux Données

```python
# Acc s    un champ sp cifique
print("Noms:", employes['nom'])
print(" ges :", employes['age'])
print("Salaires:", employes['salaire'])

# Acc s    un   lment   sp cifique avec tous ses champs
print("\nPremier employ :", employes[0])
print("Nom du premier employ :", employes[0]['nom'])
```

Sortie :

```
Noms: ['Alice' 'Bob' 'Charlie' 'Diana' 'Eve']
Âges: [30 25 35 28 40]
Salaires: [55000. 45000. 60000. 52000. 48000.]

Premier employé: ('Alice', 30, 55000.)
Nom du premier employé: Alice
```

3 Filtrage Avancé vs Filtrage Simple

3.1 Problèmes du Filtrage Simple

```python
# MAUVAISE APPROCHE: Tableaux s par s
noms = ['Alice', 'Bob', 'Charlie', 'Diana', 'Eve']
ages = [30, 25, 35, 28, 40]
salaires = [55000, 45000, 60000, 52000, 48000]

# Filtrage complexe et error-prone
jeunes_well_paid_indices = [
    i for i, (age, salaire) in enumerate(zip(ages, salaires))
    if age < 35 and salaire > 50000
]

resultats_noms = [noms[i] for i in jeunes_well_paid_indices]
resultats_ages = [ages[i] for i in jeunes_well_paid_indices]
resultats_salaires = [salaires[i] for i in jeunes_well_paid_indices]

print("R sultats (approche simple):")
print("Noms:", resultats_noms)
print(" ges :", resultats_ages)
print("Salaires:", resultats_salaires)
```

3.2 Solution avec Structured Arrays

```
# BONNE APPROCHE: Structured Array
jeunes_well_paid = employes[
    (employes['age'] < 35) & (employes['salaire'] > 50000)
]

print("R sultats (Structured Array):")
print(jeunes_well_paid)
print(f"\nType: {type(jeunes_well_paid)}")
print(f"Shape: {jeunes_well_paid.shape}")
```

Sortie :

```
Résultats (Structured Array):
[('Alice', 30, 55000.) ('Diana', 28, 52000.)]

Type: <class 'numpy.ndarray'>
Shape: (2,)
```

4 Valeur Ajoutée des Structured Arrays

4.1 Comparaison Détaillée

Aspect	Approche Simple	Structured Arrays
Intégrité des données	Risque de désynchronisation	Données toujours synchronisées
Performance	Boucles Python lentes	Opérations vectorisées NumPy
Mémoire	Stockage fragmenté	Stockage contigu
Code	Complexe et répétitif	Concise et expressive
Maintenance	Difficile	Facile

TABLE 1 – Comparaison des approches

4.2 Exemples Avancés de Filtrage

```
# Filtres complexes avec Structured Arrays
print("Employ s entre 25 et 35 ans:")
print(employes[(employes['age'] >= 25) & (employes['age'] <= 35)])

print("\nEmploy s avec salaire > moyenne:")
salaire_moyen = employes['salaire'].mean()
print(employes[employes['salaire'] > salaire_moyen])

print("\nEmploy s tri s par  ge :")
print(np.sort(employes, order='age'))
```

```
11
12 print("\nEmploy s  tri  s  par  salaire  d croissant:")
13 print(np.sort(employes, order='salaire')[::-1])
```

5 Record Arrays

5.1 Définition et Création

Les **Record Arrays** sont une extension des Structured Arrays qui permettent l'accès aux champs via la notation pointée.

```
1  # Cr ation d'un Record Array
2  employes_rec = np.rec.array([
3      ('Alice', 30, 55000.0),
4      ('Bob', 25, 45000.0),
5      ('Charlie', 35, 60000.0)
6  ], dtype=[('nom', 'U10'), ('age', 'i4'), ('salaire', 'f8')])
7
8  # Alternative: conversion depuis Structured Array
9  employes_rec2 = employes.view(np.recarray)
10
11 print("Record Array:")
12 print(employes_rec)
13 print(f"\nType: {type(employes_rec)}")
```

5.2 Accès avec Notation Pointée

```
1  # Acc s avec notation point e (Record Arrays seulement)
2  print("Noms (notation point e):", employes_rec.nom)
3  print(" ges  (notation point e):", employes_rec.age)
4  print("Salaires (notation point e):", employes_rec.salaire)
5
6  # Comparaison avec Structured Array
7  print("\nAvec Structured Array (notation crochet):", employes['nom'])
8  print("Avec Record Array (notation point e):", employes_rec.nom)
```

6 Applications Pratiques

6.1 Gestion de Données Complexes

```
1  # Exemple: Donn es scientifiques complexes
2  dtype_complexe = [
3      ('id', 'i8'),
4      ('timestamp', 'datetime64[s]'),
5      ('position', 'f8', (3,)),  # x, y, z
6      ('vitesse', 'f8', (3,)),   # vx, vy, vz
```

```
    ('masse', 'f8'),
    ('categorie', 'U20')
]

donnees_scientifiques = np.array([
    (1, np.datetime64('2023-01-01T10:00:00'), [1.0, 2.0, 3.0], [0.1, 0.2,
        0.3], 5.5, 'planete'),
    (2, np.datetime64('2023-01-01T10:00:01'), [4.0, 5.0, 6.0], [0.4, 0.5,
        0.6], 7.2, 'etoile'),
    (3, np.datetime64('2023-01-01T10:00:02'), [7.0, 8.0, 9.0], [0.7, 0.8,
        0.9], 3.1, 'asteroide')
], dtype=dtype_complexe)

print("Donn es scientifiques:")
print(donnees_scientifiques)
print("\nPositions seulement:")
print(donnees_scientifiques['position'])
```

6.2 Analyse de Performances

```
import time

# Test de performance
def test_performance():
    n = 100000

    # Donn es de test
    ages = np.random.randint(20, 65, n)
    salaires = np.random.uniform(30000, 100000, n)
    noms = np.array([f'Employe_{i}' for i in range(n)])

    # Approche Structured Array
    dtype = [('nom', 'U20'), ('age', 'i4'), ('salaire', 'f8')]
    employes = np.array(list(zip(noms, ages, salaires)), dtype=dtype)

    # Test de filtrage
    start = time.time()
    resultat_sa = employes[(employes['age'] < 35) & (employes['salaire']
       > 50000)]
    temps_sa = time.time() - start

    # Approche Python pure
    data_python = list(zip(noms, ages, salaires))
    start = time.time()
    resultat_python = [(nom, age, salaire) for nom, age, salaire in
       data_python
                       if age < 35 and salaire > 50000]
    temps_python = time.time() - start

```

```
    print(f"Structured Arrays: {temps_sa:.4f}s, {len(resultat_sa)} r sultats")
    print(f"Python pur: {temps_python:.4f}s, {len(resultat_python)} r sultats")
    print(f"Acc l ration: {temps_python/temps_sa:.1f}x")

test_performance()
```

7 Meilleures Pratiques

7.1 Choix des Types de Données

Type Python	Type NumPy	Description
str	'U<n>'	Chaîne Unicode (n caractères)
int	'i<n>'	Entier signé (n bytes)
float	'f<n>'	Flottant (n bytes)
bool	'?'	Booléen
datetime	'datetime64[unit]'	Date/Heure

TABLE 2 – Correspondance des types de données

7.2 Conseils d'Utilisation

1. **Choisir les types appropriés** pour minimiser l'usage mémoire
2. **Utiliser des noms de champs significatifs**
3. **Préférer Structured Arrays** pour la performance pure
4. **Utiliser Record Arrays** pour la lisibilité du code
5. **Valider les données** avant création du tableau

8 Conclusion

Points Clés à Retenir

- Les **Structured Arrays** permettent de stocker des données hétérogènes dans un tableau NumPy unique
- Ils offrent une **meilleure performance** que les approches Python pures
- Le **filtrage avancé** est naturel et expressif
- Les **Record Arrays** ajoutent la notation pointée pour plus de lisibilité
- Idéal pour les **données tabulaires simples** avant de passer à Pandas

8.1 Quand Utiliser Structured Arrays ?

- Données structurées avec schéma fixe

— Performance critique
— Opérations NumPy avancées nécessaires
— Éviter la dépendance à Pandas
— Données très dynamiques
— Opérations complexes de jointure/groupby
— Manipulations de données avancées

9 Exercices Pratiques

9.1 Challenge 1 : Gestion d'Inventaire

Créez un système d'inventaire avec Structured Arrays pour gérer des produits avec : nom, catégorie, prix, quantité en stock, et date d'ajout. Implémentez des fonctions pour :

— Ajouter/supprimer des produits
— Filtrer les produits low-stock (< 10 unités)
— Trouver les produits les plus chers par catégorie

9.2 Challenge 2 : Analyse de Données Étudiantes

Créez un système pour gérer les notes d'étudiants avec : nom, matière, note, date d'examen. Implémentez des analyses pour :

— Calculer la moyenne par matière
— Trouver les étudiants en échec ($< 50\%$)
— Identifier les améliorations entre deux examens