Національний університет біоресурсів і
природокористування України

Організація наукових обчислень за допомогою пакета NumPy

Лектор: к.ф.-м.н., доцент Кириченко Віктор Вікторович

Вступ до NumPy

Що таке NumPy?

- Бібліотека для наукових обчислень на Python
- Основний об'єкт - багатовимірний масив
- Розроблена Travis Oliphant у 2005 році

Переваги

- Швидкі обчислення (використання C/C++)
- Ефективна робота з великими масивами даних
- Основа для багатьох наукових бібліотек

Основні можливості

- Векторизовані операції
- Широкий набір математичних функцій
- Інструменти лінійної алгебри

Застосування

- Обробка сигналів та зображень
- Статистичний аналіз
- Машинне навчання та наука про дані

Встановлення NumPy

Встановлення за допомогою pip

```
# Встановлення NumPy
pip install numpy

# Оновлення до останньої версії
pip install --upgrade numpy
```

pip - стандартний менеджер пакетів для Python

Встановлення за допомогою conda

```
# Встановлення NumPy
conda install numpy

# Оновлення до останньої версії
conda update numpy
```

conda - менеджер пакетів для Anaconda

Перевірка встановлення

```
import numpy as np

# Перевірка версії
print(np.__version__)
```

Якщо версія виведена без помилок, NumPy встановлено правильно

Рекомендації

- Використовуйте віртуальні середовища
- Перевіряйте версії залежностей
- Оновлюйте NumPy регулярно

Основні концепції NumPy

Об'єкт ndarray

- Багатовимірний масив однотипних даних
- Фіксований розмір при створенні
- Ефективна робота з пам'яттю

```
import numpy as np
a = np.array([1, 2, 3])
# ndarray з 3 елементами
```

Типи даних

- Цілі числа: int8, int16, int32, int64
- Дійсні числа: float16, float32, float64
- Логічні: bool
- Комплексні: complex64, complex128

Атрибути масиву

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

# Розмірність масиву
arr.ndim # 2

# Форма масиву
arr.shape # (2, 3)

# Кількість елементів
arr.size # 6
```

Переваги перед списками

- Компактне зберігання даних
- Швидкі векторизовані операції
- Ефективне використання пам'яті
- Широкий набір математичних функцій

Створення масивів (частина 1)

Зі списків

```python
import numpy as np

# Одновимірний масив
a = np.array([1, 2, 3, 4, 5])
# array([1, 2, 3, 4, 5])

# Двовимірний масив
b = np.array([[1, 2, 3], [4, 5, 6]])
# array([[1, 2, 3], [4, 5, 6]])
```

Функція array()

```python
import numpy as np

# Створення з кортежу
a = np.array((1, 2, 3))
# array([1, 2, 3])

# Створення з генератора
b = np.array([i**2 for i in range(5)])
# array([0, 1, 4, 9, 16])
```

Вказування типу даних

```python
import numpy as np

# Цілі числа
a = np.array([1, 2, 3], dtype=np.int32)

# Дійсні числа
b = np.array([1, 2, 3], dtype=np.float64)

# Комплексні числа
c = np.array([1, 2, 3], dtype=complex)
```

Важливі властивості

- Масиви NumPy є змінними об'єктами
- Всі елементи мають однаковий тип
- Розмір масиву визначається при створенні
- Пам'ять виділяється неперервно

Створення масивів (частина 2)

0 Масиви з нулями та одиницями

```
import numpy as np

# Масив з нулями
zeros_arr = np.zeros((2, 3))
# array([[0., 0., 0.],
# [0., 0., 0.]])

# Масив з одиницями
ones_arr = np.ones((2, 3))
# array([[1., 1., 1.],
# [1., 1., 1.]])
```

Пусті та випадкові масиви

```
import numpy as np

# Пустий масив (випадкові значення)
empty_arr = np.empty((2, 3))

# Випадкові значення [0,1)
random_arr = np.random.rand(2, 3)

# Випадкові цілі числа
randint_arr = np.random.randint(0, 10, (2, 3))
```

Масиви з послідовностями

```
import numpy as np

# Арифметична прогресія
arange_arr = np.arange(0, 10, 2)
# array([0, 2, 4, 6, 8])

# Рівномірно розподілені значення
linspace_arr = np.linspace(0, 1, 5)
# array([0. , 0.25, 0.5 , 0.75, 1. ])
```

Спеціальні масиви

```
import numpy as np

# Одинична матриця
eye_arr = np.eye(3)
# array([[1., 0., 0.],
# [0., 1., 0.],
# [0., 0., 1.]])

# Діагональна матриця
diag_arr = np.diag([1, 2, 3])
```

Індексація та зрізання масивів

Базова індексація

```
import numpy as np

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

# Доступ до елемента
element = arr[2] # 3

# Від'ємна індексація
last = arr[-1] # 5
```

Зрізання (slicing)

```
import numpy as np

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

# arr[start:stop:step]
slice1 = arr[1:4] # [2, 3, 4]
slice2 = arr[::2] # [1, 3, 5]
slice3 = arr[::-1] # [5, 4, 3, 2, 1]
```

Багатовимірні масиви

```
import numpy as np

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

# Доступ до елемента
element = arr[0, 1] # 2

# Зріз рядка
row = arr[0, :] # [1, 2, 3]
col = arr[:, 1] # [2, 5]
```

Логічна індексація

```
import numpy as np

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

# Умова
mask = arr > 2 # [F, F, T, T, T]

# Фільтрація
filtered = arr[mask] # [3, 4, 5]
filtered = arr[arr > 2] # [3, 4, 5]
```

Операції над масивами (частина 1)

Арифметичні операції

```python
import numpy as np

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

# Додавання
c = a + b # [5, 7, 9]

# Віднімання
d = a - b # [-3, -3, -3]

# Множення
e = a * b # [4, 10, 18]
```

Елементні операції

```python
import numpy as np

a = np.array([1, 4, 9])

# Квадратний корінь
b = np.sqrt(a) # [1, 2, 3]

# Показникова функція
c = np.exp(a) # [2.7, 54.6, 8103.1]

# Степінь
d = np.power(a, 2) # [1, 16, 81]
```

Трансляція (broadcasting)

```python
import numpy as np

# Масив і скаляр
a = np.array([1, 2, 3])
b = a + 5 # [6, 7, 8]

# Масиви різних розмірів
c = np.array([[1], [2], [3]])
d = np.array([4, 5, 6])
e = c + d # [[5,6,7], [6,7,8], [7,8,9]]
```

Правила трансляції

- Масиви повинні мати сумісні форми
- Розмірність повинна збігатися або бути 1
- Менший масив "розтягується" до більшого
- Оптимізована для швидких обчислень

Операції над масивами (частина 2)

Агрегаційні функції

```python
import numpy as np

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

# Сума елементів
total = np.sum(arr) # 15

# Середнє значення
mean = np.mean(arr) # 3.0

# Мінімум та максимум
min_val = np.min(arr) # 1
max_val = np.max(arr) # 5
```

Сортування

```python
import numpy as np

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

# Сортування масиву
sorted_arr = np.sort(arr)
# [1, 2, 3, 4, 5]

# Індекси відсортованих елементів
indices = np.argsort(arr)
# [1, 3, 0, 2, 4]
```

Зміна форми масиву

```python
import numpy as np

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

# Зміна форми
reshaped = arr.reshape(2, 3)
# [[1, 2, 3], [4, 5, 6]]

# Транспонування
transposed = reshaped.T
# [[1, 4], [2, 5], [3, 6]]
```

Об'єднання та розділення

```python
import numpy as np

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

# Об'єднання
concat = np.concatenate((a, b))
# [1, 2, 3, 4, 5, 6]

# Розділення
split = np.split(concat, 2)
# [array([1, 2, 3]), array([4, 5, 6])]
```

Статистичні операції з NumPy

Основні статистичні показники

```
import numpy as np

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

# Середнє значення
mean = np.mean(data) # 5.0

# Медіана
median = np.median(data) # 5.0

# Стандартне відхилення
std = np.std(data) # 2.58
```

Розподіли та дисперсія

```
import numpy as np

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

# Дисперсія
var = np.var(data) # 6.67

# Квантилі
q25 = np.quantile(data, 0.25) # 3.0
q75 = np.quantile(data, 0.75) # 7.0
```

Кореляція та коваріація

```
import numpy as np

x = np.array([1, 2, 3, 4, 5])
y = np.array([2, 4, 6, 8, 10])

# Коефіцієнт кореляції
corr = np.corrcoef(x, y)
# [[1. 1.] [1. 1.]]

# Коваріаційна матриця
cov = np.cov(x, y)
# [[2.5 5.] [5. 10.]]
```

Спеціалізовані функції

- `np.histogram()` - гістограма розподілу
- `np.percentile()` - процентилі
- `np.ptp()` - різниця між max та min
- `np.average()` - середнє вагове

Створення матриць

Двовимірні масиви

```
import numpy as np

# Створення зі списку списків
matrix = np.array([[1, 2, 3],
                   [4, 5, 6],
                   [7, 8, 9]])

# Перевірка розмірності
print(matrix.shape) # (3, 3)
```

Спеціальні матриці

```
import numpy as np

# Одинична матриця
identity = np.eye(3)
# [[1. 0. 0.]
# [0. 1. 0.]
# [0. 0. 1.]]

# Матриця з нулів
zeros = np.zeros((2, 3))
```

Діагональні матриці

```
import numpy as np

# Діагональна матриця
diag = np.diag([1, 2, 3])
# [[1 0 0]
# [0 2 0]
# [0 0 3]]

# Діагональ існуючої матриці
d = np.diag(matrix)
```

Перетворення типів

```
import numpy as np

# Вектор у матрицю-рядок
row = np.array([1, 2, 3]).reshape(1, -1)
# [[1 2 3]]

# Вектор у матрицю-стовпець
col = np.array([1, 2, 3]).reshape(-1, 1)
# [[1] [2] [3]]
```

Операції з матрицями (частина 1)

+ Додавання та віднімання

```
import numpy as np

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

# Додавання матриць
C = A + B
# [[ 6 8]
# [10 12]]

# Віднімання матриць
D = B - A
# [[4 4]
# [4 4]]
```

× Множення на скаляр

```
import numpy as np

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

# Множення на скаляр
B = k * A
# [[ 3 6]
# [ 9 12]]

# Ділення на скаляр
C = A / k
# [[0.333 0.666]
# [1. 1.333]]
```

Поелементні операції

```
import numpy as np

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

# Поелементне множення
C = A * B
# [[ 5 12]
# [21 32]]

# Поелементне ділення
D = B / A
# [[5. 3. ]
# [2.333 2. ]]
```

Σ Математичні функції

```
import numpy as np

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

# Піднесення до степеня
B = np.power(A, 2)
# [[ 1 4]
# [ 9 16]]

# Квадратний корінь
C = np.sqrt(A)
# [[1. 1.414]
# [1.732 2. ]]
```

Операції з матрицями (частина 2)

Σ Матричне множення

```
import numpy as np

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

# Матричне множення
C = np.dot(A, B)
# [[19 22]
# [43 50]]

# Альтернативний запис
D = A @ B
```

Транспонування

```
import numpy as np

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

# Транспонування матриці
B = A.T
# [[1 4]
# [2 5]
# [3 6]]

# Альтернативний запис
C = np.transpose(A)
```

Обернена матриця

```
import numpy as np

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

# Обернена матриця
A_inv = np.linalg.inv(A)
# [[-2.  1. ]
# [ 1.5 -0.5]]

# Перевірка: A @ A_inv ≈ I
result = np.round(A @ A_inv)
```

Визначник та слід

```
import numpy as np

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

# Визначник матриці
det = np.linalg.det(A)
# -2.0

# Слід матриці (сума діагоналі)
trace = np.trace(A)
# 5
```

Числові діапазони з NumPy

Функція arange()

```
import numpy as np

# Арифметична прогресія
a = np.arange(5)
# [0 1 2 3 4]

# Вказування початку та кінця
b = np.arange(2, 8)
# [2 3 4 5 6 7]

# З кроком
c = np.arange(0, 10, 2)
# [0 2 4 6 8]
```

Функція linspace()

```
import numpy as np

# Рівномірно розподілені значення
a = np.linspace(0, 1, 5)
# [0. 0.25 0.5 0.75 1. ]

# Без включення останнього значення
b = np.linspace(0, 1, 5, endpoint=False)
# [0. 0.2 0.4 0.6 0.8]

# Повернення кроку
c, step = np.linspace(0, 1, 5, retstep=True)
```

Функція logspace()

```
import numpy as np

# Логарифмічна шкала
a = np.logspace(0, 2, 5)
# [ 1. 3.16 10. 31.62 100. ]

# З вказаною основою
b = np.logspace(0, 2, 5, base=2)
# [1. 1.68 2.83 4.76 8. ]

# Від'ємні показники
c = np.logspace(-2, 0, 5)
```

Функція meshgrid()

```
import numpy as np

# Координатні сітки
x = np.array([0, 1, 2])
y = np.array([0, 1])

# Створення сітки
X, Y = np.meshgrid(x, y)

# X = [[0 1 2]
# [0 1 2]]
# Y = [[0 0 0]
# [1 1 1]]
```

Лінійна алгебра з NumPy

Σ Розв'язання рівнянь

```
import numpy as np

# Система Ax = b
A = np.array([[3, 1], [1, 2]])
b = np.array([9, 8])

# Розв'язання системи
x = np.linalg.solve(A, b)
# [2. 3.]

# Перевірка
np.allclose(np.dot(A, x), b)
```

Власні значення та вектори

```
import numpy as np

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

# Власні значення та вектори
eigenvalues, eigenvectors = np.linalg.eig(A)

# eigenvalues: [3.618 1.382]
# eigenvectors: [[ 0.850 -0.525]
# [ 0.525 0.850]]

# Перевірка
np.allclose(A @ eigenvectors[:,0], eigenvalues[0] * eigenvectors[:,0])
```

Матричні розклади

```
import numpy as np

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

# SVD розклад
U, s, Vh = np.linalg.svd(A)

# QR розклад
Q, R = np.linalg.qr(A)

# LU розклад
P, L, U = scipy.linalg.lu(A)
```

Інші функції

- `np.linalg.norm()` - норма вектора/матриці
- `np.linalg.matrix_rank()` - ранг матриці
- `np.linalg.cond()` - число обумовленості
- `np.linalg.pinv()` - псевдообернена матриця

Практичні приклади (частина 1)

Обробка даних

```
import numpy as np

# Зчитування даних з файлу
data = np.loadtxt('data.csv', delimiter=',')

# Нормалізація даних
normalized = (data - np.mean(data, axis=0)) / np.std(data, axis=0)

# Фільтрація аномалій
filtered = data[np.abs(data - np.mean(data)) < 3 * np.std(data)]
```

Математичні розрахунки

```
import numpy as np

# Обчислення полінома
x = np.linspace(-5, 5, 100)
y = x**3 - 2*x**2 + 5*x - 1

# Інтерполяція
from scipy.interpolate import interp1d
f = interp1d(x_data, y_data, kind='cubic')
y_new = f(x_new)
```

Статистичний аналіз

```
import numpy as np

# Кореляційний аналіз
correlation_matrix = np.corrcoef(data.T)

# Групування даних
groups = data[data[:, 0] < threshold]

# Регресійний аналіз
A = np.vstack([x, np.ones(len(x))]).T
slope, intercept = np.linalg.lstsq(A, y, rcond=None)[0]
```

Наукові обчислення

```
import numpy as np

# Розв'язання диф.рівнянь
def model(y, t):
  return -0.5 * y

# Метод скінченних різниць
dx = 0.1
x = np.arange(0, 10, dx)
dy_dx = np.gradient(y, dx)
```

Практичні приклади (частина 2)

Обробка зображень

```
import numpy as np
from PIL import Image

# Завантаження зображення
img = Image.open('image.jpg')
img_array = np.array(img)

# Зміна контрастності
contrast = 1.5
adjusted = img_array * contrast
adjusted = np.clip(adjusted, 0, 255)
```

Обробка сигналів

```
import numpy as np

# Генерація сигналу
t = np.linspace(0, 1, 1000)
signal = np.sin(2 * np.pi * 10 * t)

# Додавання шуму
noise = 0.2 * np.random.randn(len(t))
noisy_signal = signal + noise

# Фільтрація
from scipy.signal import butter, filtfilt
filtered = filtfilt(*butter(4, 0.1), noisy_signal)
```

Машинне навчання

```
import numpy as np

# Генерація даних
X = np.random.rand(100, 2)
y = (X[:, 0] + X[:, 1] > 1).astype(int)

# Нормалізація
X_norm = (X - X.mean(axis=0)) / X.std(axis=0)

# Розподіл на вибірки
indices = np.random.permutation(len(X))
train_idx, test_idx = indices[:80], indices[80:]
```

Числове моделювання

```
import numpy as np

# Метод Ейлера для ДР
def euler(f, y0, t):
  y = np.zeros(len(t))
  y[0] = y0
  for i in range(1, len(t)):
    y[i] = y[i-1] + f(y[i-1]) * (t[i] - t[i-1])
  return y

# Модель популяції
f = lambda y: 0.1 * y * (1 - y/100)
```

Переваги продуктивності NumPy

Векторизація

```python
import numpy as np

# Звичайний Python
a = [1, 2, 3, 4, 5]
b = [6, 7, 8, 9, 10]
c = [a[i] + b[i] for i in range(len(a))]

# NumPy
a_np = np.array([1, 2, 3, 4, 5])
b_np = np.array([6, 7, 8, 9, 10])
c_np = a_np + b_np
```

Векторизовані операції виконуються на рівні компільованого коду, а не через інтерпретатор Python

Ефективність пам'яті

```python
import numpy as np

# Розмір об'єкта
a = list(range(1000))
b = np.arange(1000)

# Порівняння розміру
# a займає ~80000 байт
# b займає ~8000 байт

# Тип даних
c = np.array([1, 2, 3], dtype=np.int32)
```

NumPy масиви зберігають дані компактно без додаткової інформації про тип кожного елемента

Оптимізовані реалізації

- Базові функції написані на C/Fortran
- Використання оптимізованих бібліотек (BLAS, LAPACK)
- Паралельні обчислення для великих масивів
- Спеціалізовані інструкції процесора (SIMD)

Продуктивність NumPy близька до мов компіляції (C, Fortran), але з простотою Python

Порівняння швидкості

```python
import numpy as np
import time

# Підготовка даних
size = 1000000
a = np.random.rand(size)
b = np.random.rand(size)

# Вимірювання часу
start = time.time()
c = a + b
end = time.time()
print(f"Час: {end-start:.5f} сек")
```

NumPy операції зазвичай у 10-100 разів швидші за аналогічні операції зі списками Python

Поширені функції NumPy

Математичні функції

- `np.sin()`, `np.cos()`, `np.tan()` - тригонометричні
- `np.exp()`, `np.log()` - експонента та логарифм
- `np.sqrt()`, `np.power()` - корінь та степінь
- `np.abs()`, `np.round()` - модуль та округлення

Маніпуляції з масивами

- `np.reshape()`, `np.ravel()` - зміна форми
- `np.transpose()`, `np.T` - транспонування
- `np.concatenate()`, `np.split()` - об'єднання та розділення
- `np.sort()`, `np.argsort()` - сортування

Статистичні функції

- `np.mean()`, `np.median()` - середнє та медіана
- `np.std()`, `np.var()` - відхилення та дисперсія
- `np.min()`, `np.max()` - мінімум та максимум
- `np.corrcoef()`, `np.cov()` - кореляція та коваріація

Лінійна алгебра

- `np.dot()`, `np.matmul()` - множення матриць
- `np.linalg.inv()` - обернена матриця
- `np.linalg.det()` - визначник матриці
- `np.linalg.eig()` - власні значення та вектори

Висновки

Ефективність обчислень

NumPy забезпечує значне прискорення обчислень порівняно зі стандартними структурами даних Python завдяки векторизації операцій та оптимізованим реалізаціям на C/Fortran.

Універсальність

Пакет NumPy є основою для багатьох наукових бібліотек Python, включаючи Pandas, SciPy, Matplotlib, що робить його незамінним інструментом для наукових обчислень.

Зручність роботи з багатовимірними даними

Масиви NumPy дозволяють ефективно працювати з багатовимірними даними, що особливо важливо для задач лінійної алгебри, обробки зображень та наукових симуляцій.

Широкий функціонал

NumPy надає величезну кількість функцій для математичних операцій, статистичних розрахунків, лінійної алгебри та роботи з масивами даних різних типів.

Оптимізація пам'яті

Компактне зберігання даних у масивах NumPy дозволяє ефективно використовувати оперативну пам'ять при роботі з великими обсягами інформації.