Лабораторна Робота 10 – NumPy.

Мета роботи – вивчити та засвоїти навички роботи з базовим функціоналом модуля NumPy. Ознайомитися з принципами використання масивів, матричної та векторної математики, а також операцій над многочленами та засвоїти практичні навички їх застосування.

1.1 Введення в NumPy

NumPy це open-source модуль для python, який надає загальні математичні і числові операції у вигляді пре-скомпільованих, швидких функцій. Вони об'єднуються в високорівневі пакети. Вони забезпечують функціонал, який можна порівняти з функціоналом MatLab. NumPy (Numeric Python) надає базові методи для маніпуляції з великими масивами і матрицями. SciPy (Scientific Python) розширює функціонал numpy величезною колекцією корисних алгоритмів, таких як мінімізація, перетворення Фур'є, регресія, і інші прикладні математичні техніки.

Є кілька шляхів імпорту. Стандартний метод це - використовувати простий вислів:

```
[26]: import numpy
```

Проте, для великої кількості викликів функцій numpy, ставати утомливо писати numpy.X знову і знову. Замість цього набагато легше зробити це так:

```
[ ]: import numpy as np
```

Цей вислів дозволяє нам отримувати доступ до numpy об'єктів використовуючи np.X замість numpy.X.

1.2 Масиви в NumPy

Головною особливістю numpy є об'єкт array. Масиви схожі зі списками в python, виключаючи той факт, що елементи масиву повинні мати однаковий тип даних, як float і int. З масивами можна проводити числові операції з великим обсягом інформації в рази швидше і, головне, набагато ефективніше ніж зі списками. Створення масиву зі списку:

```
[3]: a = np.array([1, 4, 5, 8], float)
     print(a)
     print(type(a))

     [1. 4. 5. 8.]
     <class 'numpy.ndarray'>
```

Тут функція array приймає два аргументи: список для конвертації в масив і тип для кожного елемента. До всіх елементів можна отримати доступ і маніпулювати ними також, як ви б це робили зі звичайними списками:

```
[4]: print(a[:2])
     print(a[3])
     a[0] = 5.
     print(a)

     [1. 4.]
     8.0
     [5. 4. 5. 8.]
```

Масиви можуть бути і багатовимірними. На відміну від списків можна задавати команди в дужках. Ось приклад двовимірного масиву (матриця):

```
[5]: a = np.array([[1, 2, 3], [4, 5, 6]], float)
     print(a)
     print(a[0,0])
     print(a[0,1])

     [[1. 2. 3.]
      [4. 5. 6.]]
     1.0
     2.0
```

Array slicing працює з багатовимірними масивами аналогічно, як і з одновимірними, застосовуючи кожен зріз, як фільтр для встановленого

вимірювання. Використовуйте ":" у вимірі для вказування використання всіх елементів цього виміру:

```
[6]: a = np.array([[1, 2, 3], [4, 5, 6]], float)
     print(a[1,:])
     print(a[:,2])
     print(a[-1:, -2:])
```

```
[4. 5. 6.]
[3. 6.]
[[5. 6.]]
```

Метод shape повертає кількість рядків і стовпців в матриці. Метод dtype повертає тип змінних, що зберігаються в масиві. Метод len повертає довжину першого виміру (осі). Метод in використовується для перевірки на наявність елемента в масиві:

```
[7]: print(a.shape)
     print(a.dtype)
     print(len(a))
     print(2 in a)
     print(0 in a)
```

```
(2, 3)
float64
2
True
False
```

Масиви можна переформувати за допомогою методу, який задає новий багатовимірний масив. Дотримуючись такого прикладу, ми переформатуємо одновимірний масив з десяти елементів у двовимірний масив, що складається з п'яти рядків і двох стовпців:

```
[8]: a = np.array(range(10), float)
     print(a)
     a = a.reshape((5, 2))
     print(a)
     print(a.shape)
```

```
[0. 1. 2. 3. 4. 5. 6. 7. 8. 9.]
[[0. 1.]
 [2. 3.]
 [4. 5.]
 [6. 7.]
 [8. 9.]]
(5, 2)
```

Зверніть увагу, метод reshape створює новий масив, а не модифікує оригінальний. Майте на увазі, зв'язування імен в python працює і з масивами. Метод copy використовується для створення копії існуючого масиву в пам'яті:

```
[9]: a = np.array([1, 2, 3], float)
     b = a
     c =  a.copy()
     a[0] = 0
     print(a)
     print(b)
     print(c)
```

```
[0. 2. 3.]
[0. 2. 3.]
[1. 2. 3.]
```

Списки можна теж створювати з масивів:

```
[10]: a = np.array([1, 2, 3], float)
      print(a.tolist())
      print(list(a))
```

```
[1.0, 2.0, 3.0]
[1.0, 2.0, 3.0]
```

Заповнення масиву однаковим значенням.

```
[13]: a = np.array([1, 2, 3], float)
      print(a)
      a.fill(0)
      print(a)
```

```
[1. 2. 3.]
[0. 0. 0.]
```

Транспонування масивів також можливо, при цьому створюється новий масив:

```
[15]: a = np.array(range(6), float).reshape((2, 3))
      print(a)
      print(a.transpose())
```

```
[[0. 1. 2.]
 [3. 4. 5.]]
[[0. 3.]
 [1. 4.]
 [2. 5.]]
```

Багатовимірний масив можна переконвертувати в одновимірний за допомогою методу flatten:

```
[16]: a = np.array([[1, 2, 3], [4, 5, 6]], float)
      print(a)
      print(a.flatten())
```

```
[[1. 2. 3.]
 [4. 5. 6.]]
[1. 2. 3. 4. 5. 6.]
```

Два або більше масивів можна об'єднати за допомогою методу concatenate:

```
[17]: a = np.array([1,2], float)
      b = np.array([3,4,5,6], float)
      c = np.array([7,8,9], float)
      print(np.concatenate((a, b, c)))
```

```
[1. 2. 3. 4. 5. 6. 7. 8. 9.]
```

У висновку, розмірність масиву може бути збільшена при використанні константи newaxis в квадратних дужках:

```
[18]: a = np.array([1, 2, 3], float)
      b = np.array([3,4,5,6], float)
      print(a)
      print(a[:,np.newaxis])
      print(a[:,np.newaxis].shape)
      print(b[np.newaxis,:])
      print(b[np.newaxis,:].shape)
```

```
[1. 2. 3.]
[[1.]
 [2.]
 [3.]]
(3, 1)
[[3. 4. 5. 6.]]
(1, 4)
```

Зауважте, тут кожен масив двовимірний; створений за допомогою newaxis має розмірність один. Метод newaxis підходить для зручного створення належномірних масивів в векторній і матричній математиці.

1.3 Базові та математичні операції над масивами

Коли для масивів ми використовуємо стандартні математичні операції, повинен дотримуватися принцип: елемент - елемент. Це означає, що масиви повинні бути однакового розміру під час додавання, віднімання і тому подібних операцій:

```
[20]: a = np.array([1,2,3], float)
      b = np.array([5,2,6], float)
      print(a + b)
      print(a - b)
      print(a * b)
      print(a / b)
```

```
[6. 4. 9.]
[-4.  0. -3.]
[ 5.  4. 18.]
[0.2 1.  0.5]
```

Для двомірних масивів, множення залишається поелементний і не відповідає множенню матриць. Для цього існують спеціальні функції. До того ж до стандартних операторів, в numpy включена бібліотека стандартних математичних функцій, які можуть бути застосовані поелементно до масивів. Власне функції: abs, sign, sqrt, log, log10, exp, sin, cos, tan, arcsin, arccos, arctan, sinh, cosh, tanh, arcsinh, arccosh, і arctanh.

```
[21]: a = np.array([1, 4, 9], float)
      print(np.sqrt(a))
```

```
[1. 2. 3.]
```

Функції floor, ceil і rint повертають нижні, верхні або найближчі (округлене) значення:

```
[22]: a = np.array([1.1, 1.5, 1.9], float)
      print(np.floor(a))
      print(np.ceil(a))
      print(np.rint(a))
```

```
[1. 1. 1.]
[2. 2. 2.]
[1. 2. 2.]
```

Також в numpy включені дві важливі математичні константи:

```
[23]: print(np.pi)
      print(np.e)
```

```
3.141592653589793
2.718281828459045
```

Проводити ітерацію масивів можна аналогічно списками. Для багатовимірних масивів ітерація буде проводитися по першій осі, так, що кожен прохід циклу буде повертати «рядок» масиву:

```
[25]: a = np.array([[1, 2], [3, 4], [5, 6]], float)
      for x in a:
          print(x)

      [1. 2.]
      [3. 4.]
      [5. 6.]
```

Для отримання будь-яких властивостей масивів існує багато функцій. Елементи можуть бути підсумовані або перемножити:

```
[26]: print(np.sum(a))
      print(np.prod(a))

      21.0
      720.0
```

Деякі функції дають можливість оперувати статистичними даними. Це функції mean (середнє арифметичне), варіація і девіація:

```
[27]: a = np.array([2, 1, 9], float)
      print(a.mean())
      print(a.var())
      print(a.std())

      4.0
      12.666666666666666
      3.559026084010437
```

Можна знайти мінімум і максимум в масиві. Функції argmin і argmax повертають індекс мінімального або максимального елемента:

```
[28]: a = np.array([2, 1, 9], float)
      print(a.min())
      print(a.max())
      print(a.argmin())
      print(a.argmax())

      1.0
      9.0
      1
      2
```

Як і списки, масиви можна відсортувати:

```
[34]: a = np.array([6, 2, 5, -1, 0], float)
      a.sort()
      print(a)

      [-1.  0.  2.  5.  6.]
```

Значення в масиві можуть бути «скорочені», щоб належати заданому діапазону. Це те ж саме що застосовувати min (max (x, minval), maxval) до кожного елементу x:

```
[35]: a = np.array([6, 2, 5, -1, 0], float)
      print(a.clip(0, 5))

      [5. 2. 5. 0. 0.]
```

Унікальні елементи можуть бути вилучені ось так:

```
[36]: a = np.array([1, 1, 4, 5, 5, 5, 7], float)
      print(np.unique(a))

      [1. 4. 5. 7.]
```

Для двомірних масивів діагональ можна отримати так:

```
[37]: a = np.array([[1, 2], [3, 4]], float)
      print(a.diagonal())

      [1. 4.]
```

1.4 Оператори порівняння і тестування значень

Булево порівняння може бути використано для поелементного порівняння масивів однакових довжин. Значення, що повертається це масив булевих True / False значень, а результат порівняння може бути збережений в масиві:

```
[39]: a = np.array([1, 3, 0], float)
      b = np.array([0, 3, 2], float)
      print(a > b)
      print(a == b)
      c = a <= b
      print(c)

      [ True False False]
      [False  True False]
      [False  True  True]
```

Масиви можуть бути порівняні з одиночним значенням:

```
[40]: a = np.array([1, 3, 0], float)
      print(a > 2)

      [False  True False]
```

Оператори any і all можуть бути використані для визначення істинні хоча б один або всі елементи відповідно:

```
[41]: c = np.array([ True, False, False], bool)
      print(any(c))
      print(all(c))
```

```
True
False
```

Комбіновані булеві вирази можуть бути застосовані до масивів за принципом елемент - елемент використовуючи спеціальні функції logical_and, logical_or і logical_not:

```
[42]: a = np.array([1, 3, 0], float)
      print(np.logical_and(a > 0, a < 3))
      b = np.array([True, False, True], bool)
      print(np.logical_not(b))
      c = np.array([False, True, False], bool)
      print(np.logical_or(b, c))
```

```
[ True False False]
[False  True False]
[ True  True  True]
```

Функція where створює новий масив з двох інших масивів однакових довжин використовуючи логічний фільтр для вибору межу двома елементами. Базовий синтаксис: where (boolarray, truearray, falsearray):

```
[45]: a = np.array([1, 3, 2], float)
      print(np.where(a != 1, 1 / a, a))
```

```
[1.         0.33333333 0.5       ]
```

1.5 Вибір елементів масиву і маніпуляція з ними

Ми вже бачили, як і у списків, елементи масиву можна отримати використовуючи операцію доступу за індексом. Однак, на відміну від списків, масиви також дозволяють робити вибір елементів використовуючи інші масиви. Це означає, що ми можемо використовувати масив для фільтрації специфічних підмножин елементів інших масивів. Булеві масиви можуть бути використані як масиви для фільтрації:

```
[47]: a = np.array([[6, 4], [5, 9]], float)
      print (a >= 6)
      print(a[a >= 6])
```

```
[[ True False]
 [False  True]]
[6. 9.]
```

Варто зауважити, що коли ми передаємо логічний масив a> = 6 як індекс для операції доступу за індексом масиву a, що повертається масив буде зберігати тільки True значення. Також ми можемо записати масив для фільтрації в змінну:

```
[48]: a = np.array([[6, 4], [5, 9]], float)
      sel = (a >= 6)
      print(a[sel])
```

```
[6. 9.]
```

Більш хитромудра фільтрація може бути досягнута використанням булевих виразів:

```
[49]: print(a[np.logical_and(a > 5, a < 9)])
```

```
[6.]
```

На додачу до булевого вибору, також можна використовувати цілочисельні масиви. В цьому випадку, цілочисельний масив зберігає індекси елементів, які будуть взяті з масиву. Розглянемо наступний одновимірний приклад

```
[50]: a = np.array([2, 4, 6, 8], float)
      b = np.array([0, 0, 1, 3, 2, 1], int)
      print(a[b])
```

```
[2. 2. 4. 8. 6. 4.]
```

1.6 Векторна і матрична математика

NumPy забезпечує багато функцій для роботи з векторами і матрицями. Функція dot повертає скалярний добуток векторів:

```
[51]: a = np.array([1, 2, 3], float)
      b = np.array([0, 1, 1], float)
      print(np.dot(a, b))
```

```
5.0
```

Функція dot також може множити матриці:

[55]:
```
a = np.array([[0, 1], [2, 3]], float)
b = np.array([2, 3], float)
c = np.array([[1, 1], [4, 0]], float)
print(a, end="\n\n")
print(np.dot(b, a), end="\n\n")
print(np.dot(a, b), end="\n\n")
print(np.dot(c, a), end="\n\n")
```

```
[[0. 1.]
 [2. 3.]]

[ 6. 11.]

[ 3. 13.]

[[2. 4.]
 [0. 4.]]
```

Також можна отримати скалярний, тензорний і зовнішній добуток матриць і векторів.

```
a = np.array([1, 4, 0], float)
b = np.array([2, 2, 1], float)
print(np.outer(a, b))
print(np.inner(a, b))
print(np.cross(a, b))
```

```
[[2. 2. 1.]
 [8. 8. 4.]
 [0. 0. 0.]]
10.0
[ 4. -1. -6.]
```

NumPy також надає набір вбудованих функцій і методів для роботи з лінійної алгеброю. Це все можна знайти в під-модулі linalg. Цими модулями також можна оперувати з виродженими і невиродженими матрицями. Визначник матриці шукається таким чином:

[57]:
```
a = np.array([[4, 2, 0], [9, 3, 7], [1, 2, 1]], float)
print(a, end="\n\n")
print(np.linalg.det(a))
```

```
[[4. 2. 0.]
 [9. 3. 7.]
 [1. 2. 1.]]

-48.00000000000003
```

Також можна знайти власний вектор і власне значення матриці:

```
[59]: vals, vecs = np.linalg.eig(a)
      print(vals,vecs, sep="\n\n")
```

```
[ 8.85591316  1.9391628  -2.79507597]

[[-0.3663565  -0.54736745  0.25928158]
 [-0.88949768  0.5640176  -0.88091903]
 [-0.27308752  0.61828231  0.39592263]]
```

1.7 Математика многочленів

NumPy надає методи для роботи з поліномами. Передаючи список коренів, можна отримати коефіцієнти рівняння:

```
[60]: print(np.poly([-1, 1, 1, 10]))
```

```
[  1. -11.   9.  11. -10.]
```

Тут, масив повертає коефіцієнти відповідні рівнянню:

$$x^4 - 11x^3 + 9x^2 + 11x - 10$$

Може бути проведена і зворотна операція: передаючи список коефіцієнтів, функція root поверне все коріння многочлена:

```
[61]: print(np.roots([1, 4, -2, 3]))
```

```
[-4.5797401 +0.j          0.28987005+0.75566815j  0.28987005-0.75566815j]
```

Коефіцієнти многочлена можуть бути інтегровані, а також можуть бути взяті похідні:

```
[63]: print(np.polyint([1, 1, 1, 1]),np.polyder([1./4., 1./3., 1./2., 1., 0.]), sep="\n\n")
```

```
[0.25       0.33333333 0.5        1.         0.        ]

[1. 1. 1. 1.]
```

Функції polyadd, polysub, polymul і polydiv також підтримують підсумовування, віднімання, множення і ділення коефіцієнтів многочлена, відповідно. Функція polyval підставляє в многочлен задане значення.

```
[64]: print(np.polyval([1, -2, 0, 2], 4))
```

```
34
```

1.8 Пакет SciPy

Пакет SciPy дуже добре розширює функціонал NumPy.

Розглянемо деякі можливості SciPy. Більшість функцій SciPy доступні після імпорту модуля пакету. Функції в кожному модулі добре задокументовані у внутрішніх docstring`ах і в офіційній документації. Більшість з них безпосередньо надає функції для роботи з числовими алгоритмами і вони дуже прості у використанні. Таким чином, SciPy зберігає гігантську кількість часу в наукових обчисленнях, тому що він забезпечує вже написані і тестовані функції.

Модуль	Для чого використовується
scipy.constants	Набір математичних і фізичних констант
scipy.special	Багато спеціальних функцій для математичної фізики, такі як: Ейрі, еліптичні, Бесселя, гамма, бета, гіпергеометричні, параболічного циліндра, Матьє, кулястої хвилі, Струве, Кельвіна.
scipy.integrate	Функції для роботи з чисельним інтеграцією використовуючи методи трапеції, Сімпсона, Ромберга та інші. Також надає методи для роботи з повними диференціальними рівняннями.
scipy.optimize	Стандартні методи пошуку максимуму / мінімуму для роботи з узагальненими для користувача функціями. Включені алгоритми: Нелдера - Міда, Поулла (Powell's), сполучених градієнтів, Бройде - Флетчера - Гольдфарба - Шанно, найменших квадратів, умовної оптимізації, імітації відпалу, повного перебору, Брента, Ньютона, бисекции, Бройде, Андерсона і лінійного пошуку.
scipy.linalg	Більш широкий функціонал для роботи з лінійної алгеброю ніж в NumPy. Надає більше можливостей для використання спеціальних і швидких функцій, для специфічних об'єктів (Наприклад: трёхдіагональная матриця). Включені методи: пошук невироджених матриці, пошук визначника, рішення лінійних систем рівнянь, розрахунку норм і псевдообернена матриця, спектрального розкладання, сингулярного розкладання, LU-розкладання, розкладання Холецкого, QR-розкладання, розкладання Шура і багато інших математичних операцій для роботи з матрицями.
scipy.sparse	Функції для роботи з великими розрідженими матрицями
scipy.interpolate	Методи і класи для інтерполяції об'єктів, які можуть бути використані для дискретних числових даних. Лінійна і сплайнова (Прим. Перекладача: математичне уявлення плавних кривих) інтерполяція доступна для одно- і двох-мірних наборів даних.

scipy.fftpack	Методи для обробки перетворень Фур'є.
scipy.signal	Методи для обробки сигналів, наприклад: згортка функцій, кореляція, дискретне перетворення Фур'є, що згладжує B-сплайн, фільтрація, і т.д і т.п.
scipy.stats	Велика бібліотека статистичних функцій і розподілів для роботи з наборами даних.

2. Варіанти завдання

Використовуючи засоби NumPy виконати завдання відповідно до таблиці варіантів (*таб. 10.1*), усі значення вихідних масивів задаються випадковим чином, всі інші невідомі (значення розмірності масивів, значення обмежень), вводяться користувачем з клавіатури.

Таблиця 10.1 – Варіанти завдань

№	*Завдання*
1.	Підрахувати добуток ненульових елементів на головній діагоналі прямокутної матриці Z = X(NxM) * Y(MxK). Для Z = np.array ([[1, 0, 1], [2, 0, 2], [3, 0, 3], [4, 4, 4]]) = 3.
2.	Дано два вектори X[N] і Y[M]. Перевірити, задають вони одну і ту ж мультимножину. Для X = np.array ([1, 2, 2, 4]), Y = np.array ([4, 2, 1, 2, 1]) відповідь True.
3.	Дан двовимірний масив X(Nx3). Знайти всі неповторювані рядки кожен елемент яких відповідає умові (x>n or x<=k)
4.	Знати добуток матриць Z = X(Nx4) * Y(4xM) і виконати нормалізацію стовпців (з кожного стовпчика відняти середнє цього стовпчика). В отриманій матриці знайти дисперсію і визначник матриці.
5.	Видаліть з вектору X всі значення які зустрічаються в матриці Y(Nx4) і знайдіть N максимальних значень в отриманому векторі
6.	Знайти максимальний елемент у векторі x серед елементів, перед якими стоїть нульовий. Для X = np.array ([8, 3, 0, 5, 0, 0, 7, 6, 0])
7.	Реалізувати кодування довжин серій (Run-length encoding). Даний вектор X. Необхідно повернути кортеж з двох векторів однакової довжини. Перший містить числа, а другий - скільки разів їх потрібно повторити. Приклад: X = np.array ([2, 2, 2, 3, 3, 3, 5]). Відповідь: (np.array ([2, 3, 5]), np.array ([3, 3, 1])).
8.	Дан двовимірний масив X(20x3). Знайти всі рядки що повторюються хоча б 1 раз, та кожен елемент яких відповідає умові (x>n or x<=k)
9.	Перетворити випадковий вектор X[N^2] в матрицю Y(NxN), провести нормалізацію (з кожного стовпчика відняти середнє цього стовпчика). В отриманій матриці знайти медіану та стандартне відхилення.
10.	Знайти мінімальний елемент у векторі x серед елементів, перед якими не стоїть нульовий. Для X = np.array ([6, 2, 0, 3, 0, 0, 5, 7, 0])

№	*Завдання*
11.	Дана матриця X (Nx3), знайти рядки що складаються з нерівних між собою значень (наприклад [2,2,3])
12.	Підрахувати добуток ненульових елементів на побічній діагоналі прямокутної матриці. Для X = np.array ([[1, 0, 1], [2, 0, 2], [3, 0, 3], [4, 4, 4]]) = 8.
13.	Для заданого числа знайдіть найближчий до нього елемент у векторі
14.	Знайти максимальний елемент у векторі x серед елементів, перед якими стоїть нульовий. Для X = np.array ([6, 2, 0, 3, 0, 0, 5, 7, 0])
15.	Знайти мінімальний елемент у векторі x серед елементів, перед якими стоїть нульовий. Для X = np.array ([8, 3, 0, 5, 0, 0, 7, 6, 0])
16.	Дано 2 масиви A (15x3) і B (2x2). Знайти рядки в A, які містять елементи з кожного рядка в B, незалежно від порядку елементів в B

3. Приклад

Варіант -16.

Файл програми

[3]:
```
import numpy as np
A = np.random.randint(0,5,(15,3))
print(A,"\n")
B = np.random.randint(0,5,(2,2))
print(B,"\n")
C = (A[..., np.newaxis, np.newaxis] == B)
rows = (C.sum(axis=(1,2,3)) >= B.shape[1]).nonzero()[0]
print(rows)
```

```
[[2 1 1]
 [0 0 2]
 [0 2 2]
 [4 2 4]
 [2 4 2]
 [3 4 0]
 [3 0 0]
 [3 2 2]
 [4 1 2]
 [1 1 1]
 [4 4 1]
 [3 0 3]
 [1 1 0]
 [2 1 2]
 [0 1 0]]

[[2 2]
 [4 0]]

[ 0  1  2  3  4  5  6  7  8 10 13 14]
```

4. Контрольні запитання

1. Модуль NumPy. Призначення та базові принципи застосування. Наведіть приклади.
2. Опишіть базові принципи та механізми роботи з масивами за допомогою NumPy.
3. Наведіть приклади математичних операцій з масивами у NumPy.
4. Опишіть механізми вибору окремих елементів масивів та методи їх обробки.
5. Опишіть існуючий у NumPy інструментарій для векторної та матричної математики.
6. Яким чином у NumPy реалізовано роботу з многочленами, наведіть приклади.
7. Модулі SciPy, призначення та особливості використання.