IRESM
Nivel Superior

Algoritmos y Estructura de Datos I

Grupo N° 3

Tema: NumPy

Profesor: Federico Bosack

Fecha de entrega: 10/11/2021

Integrantes:

- Escobosa, Gonzalo Gastón
- Ludueña, María Emilia
- Martinez Maria, Candela
- Pavan Destru, Camila

Índice

NumPy

NumPy es una biblioteca para el lenguaje de programación Python que da soporte para crear vectores y matrices grandes multidimensionales, es un tipo de dato estructurado muy utilizado en análisis de datos, en informática científica y en el área del aprendizaje automático; junto con una gran colección de funciones matemáticas de alto nivel para operar con ellas. También tiene funciones para trabajar en el dominio del álgebra lineal, la transformada de Fourier además las matrices. NumPy es un software de código abierto, cuenta con muchos colaboradores y puede usarse libremente.
La nomenclatura “NumPy” significa Python numérico.

Historia

El lenguaje de programación Python no fue diseñado originalmente para computación numérica, pero atrajo la atención de la comunidad científica y de ingeniería desde el principio. En 1995 se fundó el grupo de interés especial (SIG) matrix-sig con el objetivo de definir un paquete de computación de vectores.
Jim Fulton completó la implementación de un paquete para matrices, luego generalizado por Jim Hugunin y llamado Numeric .

A mediados de los 90, dos paquetes principales estaban presentes en el mundo científico: Numarray y Numeric.

- **Numarray** era un paquete de procesamiento de matrices diseñado para manipular de manera eficiente matrices multidimensionales grandes.
- **Numeric** era eficiente para el manejo de arreglos pequeños y tenía una API C rica.

Se escribió un nuevo paquete llamado Numarray como reemplazo más flexible de Numeric. Al igual que Numeric, actualmente también está obsoleto.

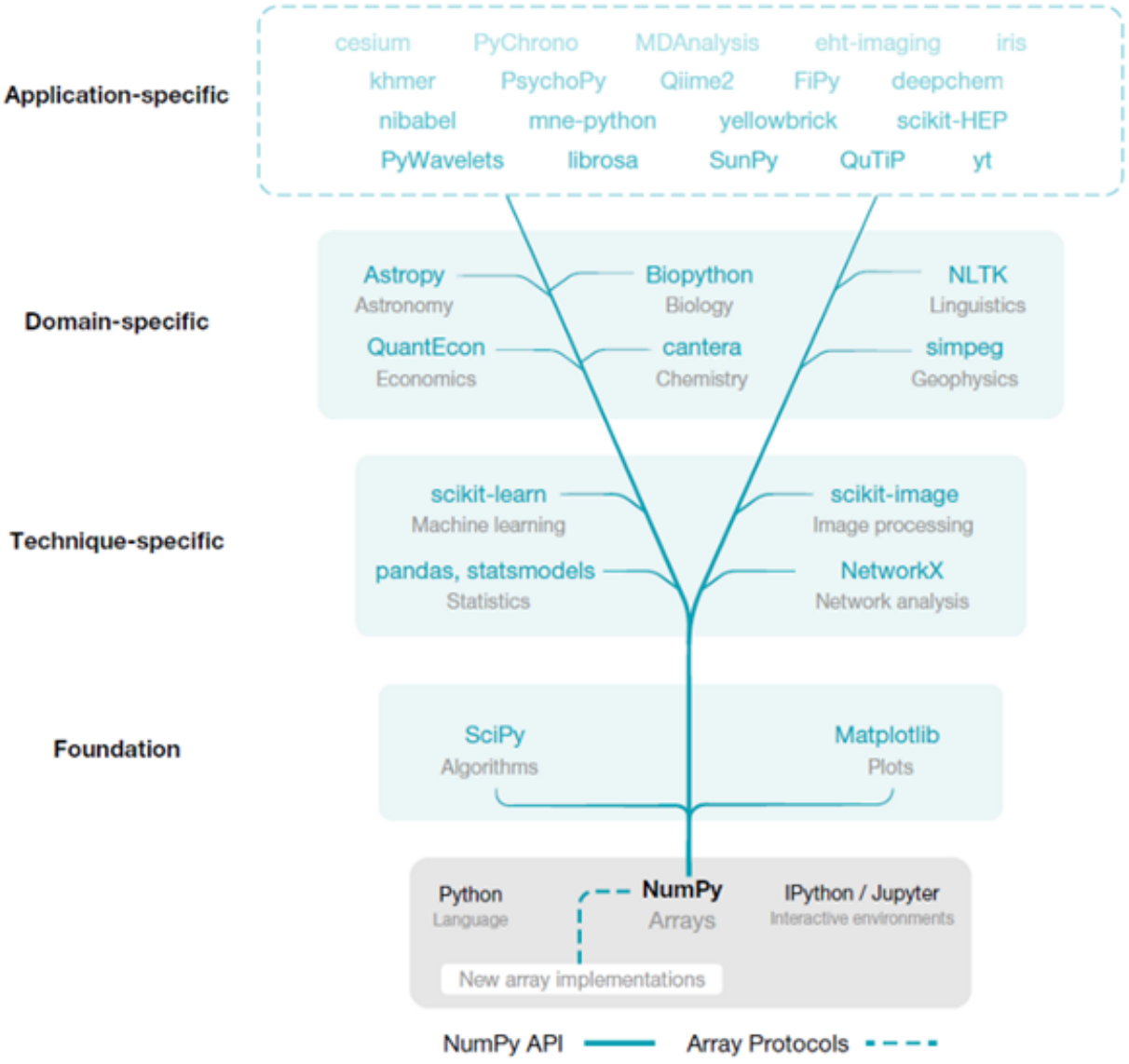
Había un deseo de incluir Numeric en la biblioteca estándar de Python, pero Guido van Rossum decidió que el código no se podía mantener en su estado en ese momento. A principios de 2005, el desarrollador de NumPy, Travis Oliphant, quería unificar la comunidad en torno a un solo paquete de arreglo y portó las características de Numarray a Numeric, lanzando el resultado como NumPy 1.0 en 2006.

¿Qué hace que NumPy sea tan bueno?

NumPy tiene una sintaxis que es a la vez compacta, potente y expresiva. Permite a los usuarios gestionar datos en vectores, matrices y matrices de mayor dimensión. Dentro de esas estructuras de datos, permite a los usuarios:

- Acceso,
- Manipular,
- Calcular

Hoy en día NumPy es compatible con la mayoría de las bibliotecas de Python que realizan cálculos científicos o numéricos, esto incluye SciPy, Matplotlib, pandas, scikit-learn y scikit-image. A modo más ilustrativo se puede observar la siguiente imágen:



En Python tenemos listas que sirven para el propósito de las matrices, pero son lentas de procesar.

NumPy tiene como objetivo proporcionar un objeto que funciona como matriz, que es hasta 50 veces más rápido que las listas tradicionales de Python.

¿En qué idioma está escrito NumPy?

NumPy es una biblioteca de Python y está escrito parcialmente en Python, pero la mayoría de las partes que requieren un cálculo rápido están escritas en C o C ++.

¿Dónde está la base de código NumPy?

El código fuente de NumPy se encuentra en este repositorio de github https://github.com/numpy/numpy

La librería Numpy

NumPy es una biblioteca que facilita el manejo de matrices. Pero, ¿qué es una matriz?
En informática, una matriz es una estructura de datos. Esta estructura de datos contiene elementos del mismo tipo de datos. Los tipos de datos pueden ser números (entero, flotante,…), cadenas (texto, cualquier cosa almacenada entre comillas “”), marcas de tiempo (fechas) o punteros a objetos de Python. Las matrices son útiles porque ayudan a organizar los datos. Con esta estructura, los elementos se pueden ordenar o buscar fácilmente. Cada matriz tiene una forma que se define por (n , m) con n el número de filas y m el número de columnas.
Como mencionamos anteriormente, NumPy es una librería de Python especializada en el cálculo numérico y el análisis de datos. (Esta oración la podríamos sacar para no repetir tanto)
NumPy incorpora una nueva clase de objetos llamados arrays que permite representar colecciones de datos de un mismo tipo en varias dimensiones, y funciones muy eficientes para su manipulación.

Instalación de NumPy

Para instalar NumPy en un ordenador se realiza insertando el siguiente comando en la consola: *C:\Users\Your Name>pip install numpy*[1]

[1] Tener en cuenta que para realizarlo de esta manera previamente hay que tener instalado Python y PIP.

Si este comando falla, use una distribución de Python que ya tenga NumPy instalado como, Anaconda, Spyder, etc.

Importar NumPy

Una vez instalado NumPy, impórtelo en sus aplicaciones agregando la palabra clave: import

```
import numpy
```

Ahora NumPy está importado y listo para usar.

Ejemplo

```
import numpy

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

print(arr)
```

NumPy como np

NumPy generalmente se importa bajo el alias.np
alias: En Python alias son un nombre alternativo para referirse a lo mismo.
Cree un alias con la palabra clave mientras importa:as

```
import numpy as np
```

Ahora el paquete NumPy se puede denominar como en lugar de .npnumpy

Ejemplo

```
import numpy as np

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

print(arr)
```

Comprobación de la versión de NumPy

La cadena de versión se almacena en el atributo.`__version__`

Ejemplo

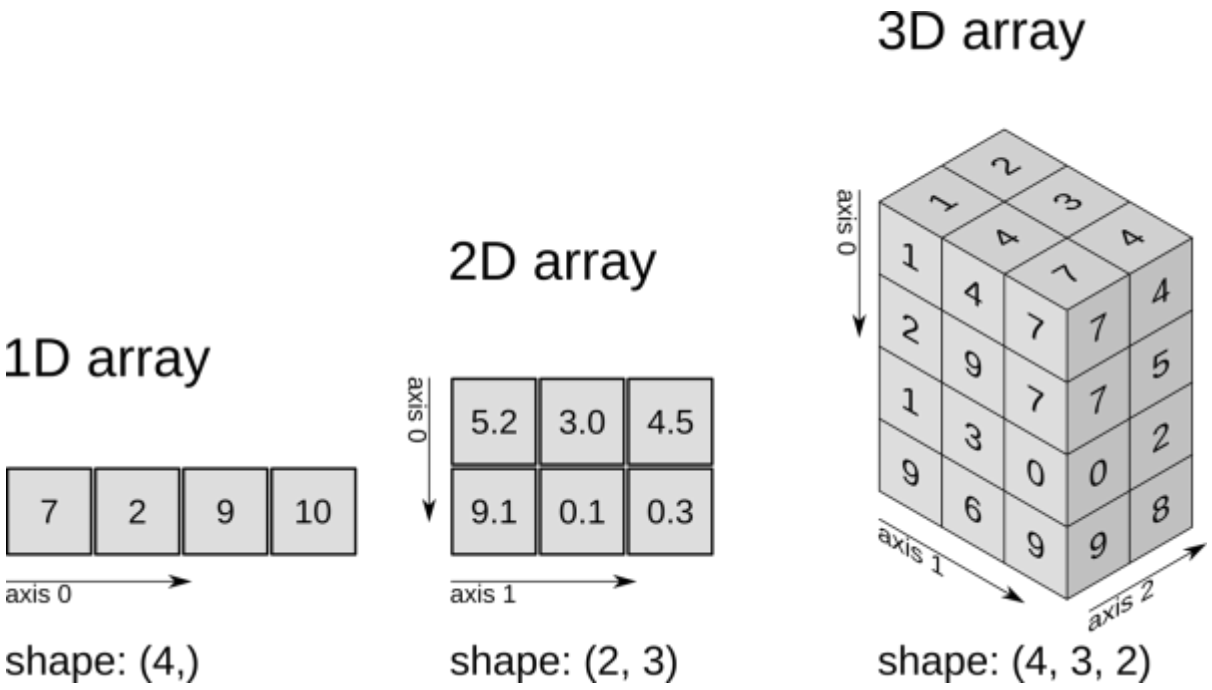
```
import numpy as np

print(np.__version__)
```

La clase de objetos `array`

Un array es una estructura de datos de un mismo tipo organizada en forma de tabla o cuadrícula de distintas dimensiones.

Las dimensiones de un array también se conocen como ejes.



Creación de arrays

Para crear un array se utiliza la siguiente función de NumPy `np.array(lista)` : Crea un array a partir de la lista o tupla `lista` y devuelve una referencia a él. El número de dimensiones del array dependerá de las listas o tuplas anidadas en `lista`:

- Para una lista de valores se crea un array de una dimensión, también conocido como vector.

- Para una lista de listas de valores se crea un array de dos dimensiones, también conocido como matriz.
- Para una lista de listas de listas de valores se crea un array de tres dimensiones, también conocido como cubo.
- Y así sucesivamente. No hay límite en el número de dimensiones del array más allá de la memoria disponible en el sistema.

Los elementos de la lista o tupla deben ser del mismo tipo.

```
>>> # Array de una dimensión
>>> a1 = np.array([1, 2, 3])
>>> print(a1)
[1 2 3]
>>> # Array de dos dimensiones
>>> a2 = np.array([[1, 2, 3], [4, 5, 6]])
>>> print(a2)
[[1 2 3]
 [4 5 6]]
>>> # Array de tres dimensiones
>>> a3 = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
>>> print(a3)
[[[ 1  2  3]
  [ 4  5  6]]

 [[ 7  8  9]
  [10 11 12]]]
```

Otras funciones útiles que permiten generar arrays son:

- `np.empty(dimensiones)` : Crea y devuelve una referencia a un array vacío con las dimensiones especificadas en la tupla `dimensiones`.

- `np.zeros(dimensiones)` : Crea y devuelve una referencia a un array con las dimensiones especificadas en la tupla `dimensiones` cuyos elementos son todos ceros.
- `np.ones(dimensiones)` : Crea y devuelve una referencia a un array con las dimensiones especificadas en la tupla `dimensiones` cuyos elementos son todos unos.
- `np.full(dimensiones, valor)` : Crea y devuelve una referencia a un array con las dimensiones especificadas en la tupla `dimensiones` cuyos elementos son todos `valor`.
- `np.identity(n)` : Crea y devuelve una referencia a la matriz identidad de dimensión `n`.
- `np.arange(inicio, fin, salto)` : Crea y devuelve una referencia a un array de una dimensión cuyos elementos son la secuencia desde `inicio` hasta `fin` tomando valores cada `salto`.
- `np.linspace(inicio, fin, n)` : Crea y devuelve una referencia a un array de una dimensión cuyos elementos son la secuencia de `n` valores equidistantes desde `inicio` hasta `fin`.
- `np.random.random(dimensiones)` : Crea y devuelve una referencia a un array con las dimensiones especificadas en la tupla `dimensiones` cuyos elementos son aleatorios.

```
>>> print(np.zeros(3,2))
[[0. 0. 0.]
 [0. 0. 0.]]
>>> print(np.idendity(3))
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
>>> print(np.arange(1, 10, 2))
[1 3 5 7 9]
>>> print(np.linspace(0, 10, 5))
[ 0.   2.5  5.   7.5 10. ]
```

Atributos de un array

Existen varios atributos y funciones que describen las características de un array.

`-a.ndi` : Devuelve el número de dimensiones del array `a`.
`-a.shape` : Devuelve una tupla con las dimensiones del array `a`.
`-a.size` : Devuelve el número de elementos del array `a`.
`-a.dtype`: Devuelve el tipo de datos de los elementos del array `a`.

Acceso a los elementos de un array

Para acceder a los elementos contenidos en un array se usan índices al igual que para acceder a los elementos de una lista, pero indicando los índices de cada dimensión separados por comas.

Al igual que para listas, los índices de cada dimensión comienzan en 0.

También es posible obtener subarrays con el operador dos puntos `:` indicando el índice inicial y el siguiente al final para cada dimensión, de nuevo separados por comas.

```
>>> a = np.array([[1, 2, 3], [4, 5, 6]])
>>> print(a[1, 0])  # Acceso al elemento de la fila 1 columna 0
4
>>> print(a[1][0])  # Otra forma de acceder al mismo elemento
4
>>> print(a[:, 0:2])
[[1 2]
 [4 5]]
```

Filtrado de elementos de un array

Una característica muy útil de los arrays es que es muy fácil obtener otro array con los elementos que cumplen una condición.

- `a[condicion]` : Devuelve una lista con los elementos del array `a` que cumplen la condición `condicion`.

```
>>> a = np.array([[1, 2, 3], [4, 5, 6]])
>>> print(a[(a % 2 == 0)])
[2 4 6]
>>> print(a[(a % 2 == 0) &  (a > 2)])
[2 4]
```

Operaciones matemáticas con arrays

Existen dos formas de realizar operaciones matemáticas con arrays: a nivel de elemento y a nivel de array.

Las operaciones a nivel de elemento operan los elementos que ocupan la misma posición en dos arrays. Se necesitan, por tanto, dos arrays con las mismas dimensiones y el resultado es una array de la misma dimensión.

Los operadores matemáticos `+`, `-`, `*`, `/`, `%`, `**` se utilizan para la realizar suma, resta, producto, cociente, resto y potencia a nivel de elemento.

```
>>> a = np.array([[1, 2, 3], [4, 5, 6]])
>>> b = np.array([[1, 1, 1], [2, 2, 2]])
>>> print(a + b )
[[2 3 4]
 [6 7 8]]
>>> print(a / b)
[[1.  2.  3. ]
 [2.  2.5 3. ]]
>>> print(a ** 2)
[[ 1  4  9]
 [16 25 36]]
```

Operaciones matemáticas a nivel de array

Para realizar el producto matricial se utiliza el método `a.dot(b)` : Devuelve el array resultado del producto matricial de los arrays `a` y `b` siempre y cuando sus dimensiones sean compatibles.

Y para trasponer una matriz se utiliza el método `a.T` : Devuelve el array resultado de trasponer el array `a`.

```
>>> a = np.array([[1, 2, 3], [4, 5, 6]])
>>> b = np.array([[1, 1], [2, 2], [3, 3]])
>>> print(a.dot(b))
[[14 14]
 [32 32]]
>>> print(a.T)
[[1 4]
 [2 5]
 [3 6]]
```

La estructura de datos ndarray

La funcionalidad principal de NumPy es su estructura de datos "ndarray" para una matriz de *n* dimensiones, que proporciona muchas funciones de soporte que hacen que trabajar con él sea muy fácil. Las matrices se utilizan con mucha frecuencia en la ciencia de datos, donde la velocidad y los recursos son muy importantes.

Las matrices NumPy se almacenan en un lugar continuo en la memoria a diferencia de las listas, por lo que los procesos pueden acceder a ellas y manipularlas de manera muy eficiente. Este comportamiento se denomina cercanía de referencias en informática. Esta es la razón principal por la que NumPy es más rápido que las listas. También está optimizado para trabajar con las últimas arquitecturas de CPU.

A diferencia de la estructura de datos de lista incorporada de Python, estas matrices se escriben de forma homogénea: todos los elementos de una única matriz deben ser del mismo tipo.

Limitaciones de NumPy

Insertar o agregar entradas a una matriz no es sencillo como lo es con las listas de Python. La `rutina np.pad(...)` para extender matrices crea nuevas matrices con la forma deseada y los valores de relleno, copia la matriz dada en la nueva y la devuelve. La `operación np.concatenate([a1, a2])` de NumPy no vincula realmente las dos matrices, sino que devuelve una nueva, llena con las

entradas de ambas matrices dadas en secuencia. La remodelación de las dimensiones de una matriz con `np.reshape(...)` solo es posible siempre que el número de elementos de la matriz no cambie. Estas circunstancias se originan en el hecho de que las matrices de NumPy deben ser vistas en búferes de memoria contiguos.

Existen estas y otras limitaciones, sin embargo, existen paquetes adicionales que permiten sortearlas.

Ejercicio propuesto para la clase

Vamos a calcular el salario de tres personas después de aplicar el impuesto a las ganancias:

1) El ejercicio deberá tener:

   ➔ Sueldos para los años 2018,2019,2020 que contenga los siguientes salarios anuales:

   -laura= [200,350,400]

   -viviana = [220,370,390]

   -pablo = [240,356,370]

   ➔ Impuesto a las ganancias para los años 2018, 2019 y 2020 con una matriz que contenga los impuestos de:

   - Laura: para el año 2018 fue del 12%, en el segundo de 14% y el tercero de 15%.

   - Viviana: para el año 2018 fue del 13%, para el 2019 fue del 10% y para 2020, 12%.

   - Pablo: para el año 2018 fue del 11%, en el segundo de 12% y el tercero de 10%.

2) Obtener la matriz con los salarios netos de cada persona para cada año (salario neto = salario - impuesto) Profe usted considera que dejemos la fórmula o que la piensen los chicos?
3) Obtener el salario máximo de la matriz obtenida (con los salarios netos).

Solución:

En nuestro ejemplo obtenemos en base a una matriz de salarios para 3 años y una matriz de impuestos el salario obtenido. Finalmente obtenemos el mayor salario.

```
import numpy as np

#Sueldos para los años 2018,2019,2020
```

```
laura= [200,350,400]
viviana= [220,370,390]
pablo= [240,356,370]

#Creamos la matriz para las listas de sueldos
salarios = np.array([laura, viviana, pablo])

#Creamos directamente la matriz de impuestos
impuestos = np.array([
    [0.12,0.14,0.15],
    [0.13,0.10,0.12],
    [0.11,0.12,0.10]
    ])

#Obtenemos el salario
salario_neto = salarios - (salarios*impuestos)

#Matriz de salarios
print('Matriz de salarios')
print(salario_neto)

#Obtener el salario máximo
print('Salario Máximo: ',np.max(salario_neto))
```

Salida:

```
Matriz de salarios
[[176.   301.   340.  ]
 [191.4  333.   343.2 ]
 [213.6  313.28 333.  ]]
Salario Máximo:  343.2
```

Bibliografía

- La librería NumPy. Extraído de: https://aprendeconalf.es/docencia/python/manual/numpy/
- NumPy. Extraído de: https://es.wikipedia.org/wiki/NumPy
- Por qué NumPy es tan fundamental. Extraído de: https://ichi.pro/es/por-que-numpy-es-tan-fundamental-8293086950144