Laboratorio de Programación Científica

Asistente Graduado: Julio Nicolás Reyes Torres

PRÁCTICA 3: *NUMPY* Y *MATPLOTLIB*

Descripción: Se presenta una guía-tutorial para el uso de las librerías *Numpy* y *Matplotlib* del lenguaje de programación Python. Estas librerías constituyen paquetes de gran utilidad en Python, la funcionalidad y soporte de estas dos librerías las constituye como las más usadas y poderosas para este lenguaje de programación. En esta guía se explican las bases de sintaxis y arquitectura para realizar diferentes operaciones matemáticas con arreglos en *Numpy* y se presenta una introducción a la librería gráfica *Matplotlib.* La tutoría está desarrollada fundamentalmente para Python, sin embargo, las diferentes funciones matemáticas y gráficas presentadas son extendidas para Matlab.

Objetivos:

- Establecer las bases del manejo de Numpy y Matplotlib, dando a conocer su gran funcionalidad y versatilidad para el procesamiento de computación científica y análisis gráfico, respectivamente.

- Identificar las funciones más útiles de Numpy para el procesamiento matemático y operaciones entre arreglos.

- Conocer cómo se implementa con Numpy las operaciones entre números binarios y enteros. Así como, el formato de representación numérica para números negativos, el complemento a 2.

- Implementar diferentes tipos de gráficas usando la librería Matplotlib. Se explica cómo hacer un plot básico (análogo en Matlab), subplot, bars, histogramas y la gráfica de una función en 3D.

Instalación de *Numpy* y *Matplotlib*:

1. Settings (Windows) / Preferences (MacOs, Linux)

2.Project interpreter

3. (+)

4. Nombre del paquete (Numpy, Matplotlib)

5. Install package

Numpy

Es la librería principal para la computación científica en Python, provee un procesamiento multidimensional de alto rendimiento para el uso de **arreglos.** Numpy cuenta con un amplio set de funciones que permiten realizar diversas operaciones entre arreglos. Su fácil implementación y versatilidad la constituyen como la librería más importante para computación científica y manejo de datos. Numpy fue creada por Travis Oliphant en el 2005 y es la evolución de Numarray.

Características:

- Realiza operaciones matemáticas y lógicas entre matrices
- Integra diversas rutinas de transformación como las transformadas de Fourier
- Implementa operaciones del álgebra lineal y generación de números aleatorios.

Se considera que Numpy junto con Scipy y Matplotlib, la constituyen una herramienta poderosa en el manejo de arreglos y es análogo a Matlab. Estructuralmente tiene

varias similitudes con Matlab. A continuación, se listan algunas funciones equivalentes para cada lenguaje.

Nota: Para ver más diferencias funciones equivalentes, revisar la documentación: *NumPy for Matlab users.* https://docs.scipy.org/doc/numpy/user/numpy-for-matlab-users.html

Matlab	Numpy	
`a([1:end 1],:)`	`a[r_[:len(a),0]]`	`a` with copy of the first row appended to the end
`a.'`	`a.transpose()` or `a.T`	transpose of `a`
`a'`	`a.conj().transpose()` or `a.conj().T`	conjugate transpose of `a`
`a * b`	`a @ b`	matrix multiply
`a .* b`	`a * b`	element-wise multiply
`a./b`	`a/b`	element-wise divide
`a.^3`	`a**3`	element-wise exponentiation
`(a>0.5)`	`(a>0.5)`	matrix whose i,jth element is (a_ij > 0.5). The Matlab result is an array of 0s and 1s. The NumPy result is an array of the boolean values `False` and `True`.
`find(a>0.5)`	`nonzero(a>0.5)`	find the indices where (`a` > 0.5)
`a(:,find(v>0.5))`	`a[:,nonzero(v>0.5)[0]]`	extract the columms of `a` where vector v > 0.5
`a(:,find(v>0.5))`	`a[:,v.T>0.5]`	extract the columms of `a` where column vector v > 0.5
`a(a<0.5)=0`	`a[a<0.5]=0`	`a` with elements less than 0.5 zeroed out
`a .* (a>0.5)`	`a * (a>0.5)`	`a` with elements less than 0.5 zeroed out
`a(:) = 3`	`a[:] = 3`	set all values to the same scalar value
`y=x`	`y = x.copy()`	numpy assigns by reference
`ndims(a)`	`ndim(a)` or `a.ndim`	get the number of dimensions of an array
`numel(a)`	`size(a)` or `a.size`	get the number of elements of an array
`size(a)`	`shape(a)` or `a.shape`	get the "size" of the matrix
`size(a,n)`	`a.shape[n-1]`	get the number of elements of the n-th dimension of array `a`. (Note that MATLAB® uses 1 based indexing while Python uses 0 based indexing, See note INDEXING)

`[ 1 2 3; 4 5 6 ]`	`array([[1.,2.,3.], [4.,5.,6.]])`	2x3 matrix literal
`[ a b; c d ]`	`block([[a,b], [c,d]])`	construct a matrix from blocks `a`, `b`, `c`, and `d`
`a(end)`	`a[-1]`	access last element in the 1xn matrix `a`
`a(2,5)`	`a[1,4]`	access element in second row, fifth column
`a(2,:)`	`a[1]` or `a[1,:]`	entire second row of `a`
`a(1:5,:)`	`a[0:5]` or `a[:5]` or `a[0:5,:]`	the first five rows of `a`
`a(end-4:end,:)`	`a[-5:]`	the last five rows of `a`
`a(1:3,5:9)`	`a[0:3][:,4:9]`	rows one to three and columns five to nine of `a`. This gives read-only access.

Crear un arreglo en Numpy

La clase arreglo se define en Numpy como ***ndarray*** (n-dimensional array). Cada elemento en ndarray es un objeto de tipo de dato (llamado dtype). La forma para definir un ndarray en Python es:

```
numpy.array(object, dtype = None, copy = True, order = None, subok = False, ndmin = 0)
```

Sr.No.	Parameter & Description
1	**object** Any object exposing the array interface method returns an array, or any (nested) sequence.
2	**dtype** Desired data type of array, optional
3	**copy** Optional. By default (true), the object is copied
4	**order** C (row major) or F (column major) or A (any) (default)
5	**subok** By default, returned array forced to be a base class array. If true, sub-classes passed through
6	**ndmin** Specifies minimum dimensions of resultant array

Ejemplos

I. Definir un arreglo unidimensional:

```
import numpy as np
a = np.array([1,2,3])
print (a)

b = np.array([1.2, 3.5, 5.1])
b.dtype
```

II. Definir un arreglo Bidimensional:

```
import numpy as np
a = np.array([[1, 2], [3, 4]])
print (a)
```

III. Definiendo el tipo de dato en el array

```
import numpy as np
a = np.array([1, 2, 3], dtype = complex)
print (a)

c = np.array( [ [1,2], [3,4] ], dtype=complex )
print (c)
```

IV. Array de ceros y unos

```
np.zeros( (3,4) )
np.ones( (2,3,4), dtype=np.int16 )
np.empty( (2,3) ) #Es random depende de la memoria
```

V. Definiendo una secuencia de números

a) arange

```
np.arange( Valor inicial, Valor Final, Pasos )
```

Ejemplo:

```
np.arange( 10, 30, 5 )
np.arange( 0, 2, 0.3 )
```

b) linspace: Cuando se ***arange*** con argumentos de punto flotante, generalmente no es posible predecir el número de elementos obtenidos, esto debido a la precisión finita del punto flotante. Por lo tanto, generalmente es mejor usar la función

linspace que recibe como argumento el número de elementos que queremos, en lugar del paso:

```
np.linspace( Valor inicial, Valor Final, # Elementos )
```

Ejemplo:

```
np.linspace( 0, 2, 9 )

from numpy import pi
x = np.linspace( 0, 2*pi, 100 )
f = np.sin(x)
```

VI. Función ***reshape*** para crear un arreglo con n-dimensiones

```
b = np.arange(12).reshape(4,3)            # 2d array
c = np.arange(24).reshape(2,3,4)          # 3d array

print(np.arange(10000).reshape(100,100))
```

Operaciones básicas con *Arrays*

```
a = np.array( [20,30,40,50] )
b = np.arange( 4 )
c = a-b

>>> c
array([20, 29, 38, 47])

>>> b**2
array([0, 1, 4, 9])

>>> 10*np.sin(a)
array([ 9.12945251, -9.88031624,  7.4511316 ,
-2.62374854])

>>> a<35
array([ True, True, False, False])
```

Nota: En Python la multiplicación de matrices se hace con: el operador @ (en python >=3.5) o con la función *dot*. El operador (*) sólo multiplica elemento por elemento

```
A = np.array( [[1,1], [0,1]] )
B = np.array( [[2,0], [3,4]] )

>>> A * B                          # elementwise product
```

```
array([[2, 0],[0, 4]])

>>> A @ B                          # matrix product
array([[5, 4],[3, 4]])

>>> A.dot(B)                       # another matrix product
array([[5, 4],[3, 4]])
```

Otras operaciones

a) Random

```
a = np.random.random((2,3))
```

b) Array constante

```
c = np.full((2,2), 7)
```

c) Matriz identidad

```
d = np.eye(3)
```

d) Otras

```
b = np.arange(12).reshape(3,4)
e = np.arange(12)**3

b.sum()                                 # sum all elements
b.min()                                 # min of array

b.max()                                # max of array


b.sum(axis=0)                                 # sum of each column
>>> array([12, 15, 18, 21])

b.min(axis=1)                                 # min of each row
>>> array([0, 4, 8])

b.cumsum(axis=1)                              # cumulative sum

np.exp(e)                                       #Euler exponent

np.sqrt(e)                                       # Square root

np.add(b, e)                                   # sum matrix b and e
```

Indexación, corte e iteración

a) Arreglos

```
a = np.arange(10)**3

a = np.arange(10)**3
>>> array([  0,   1,   8,  27,  64, 125, 216, 343, 512, 729])

a[2]
>>> 8

a[2:5]
>>> array([ 8, 27, 64])

a[:6:2] = -1000    # equivalent to a[0:6:2] = -1000; from start to
position 6, exclusive, set every 2nd element to -1000
>>> array([-1000,     1, -1000,    27, -1000,   125,   216,   343,
512,   729])

a[ : :-1]                                    # reversed a
>>> array([  729,   512,   343,   216,   125, -1000,    27, -1000,
1, -1000])

for i in a:
    print(i**(1/3.))


x = np.arange(0,10,2)                        # x=([0,2,4,6,8])
y = np.arange(5)                             # y=([0,1,2,3,4])
m = np.vstack([x,y])                         # m=([[0,2,4,6,8],
                                             #     [0,1,2,3,4]])
xy = np.hstack([x,y])                  # xy =([0,2,4,6,8,0,1,2,3,4])
```

b) Matrices

```
f = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])


f[2,3]
f[0:5, 1]                    # each row in the second column of b
f[ : ,1]                     # equivalent to the previous example
f[1:3, : ]  # each column in the second and third row of b

f[-1]                      # the last row. Equivalent to b[-1,:]

f.transpose()
```

Números binarios

a) Binario a entero

```
int('00010001', 2)
```

b) Entero a Binario

```
bin ( Número )

np.binary_repr(3)
>>> '11'

np.binary_repr(-3)
>>> '-11'

np.binary_repr(28, width=8)
>>> '00011100'
```

c) Complemento a 2:

Nota: El complemento a 2 se obtiene **SOLO** cuando se ingresa el número negativo y el número de bits (width)

```
np.binary_repr(-3, width=3)
>>>  '101'

np.binary_repr(-10, width=4)
>>> '10110'
```

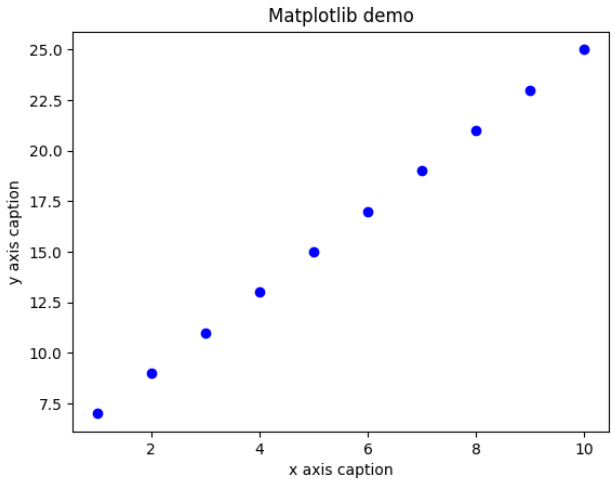
Matplotlib (http://www.matplotlib.org/)

Es una librería de graficación para Python, provee diversos formatos y ambientes interactivos para crear todo tipo de gráficas. La página oficial de Matplotlib ofrece tutoriales y ejemplos de diferentes tipos de gráficas (https://matplotlib.org/gallery/index.html).

El paquete más importante de Matplotlib es ***pyplot()*** que permite visualmente de forma similar a Matlab hacer gráficas 2D.

1) Plots

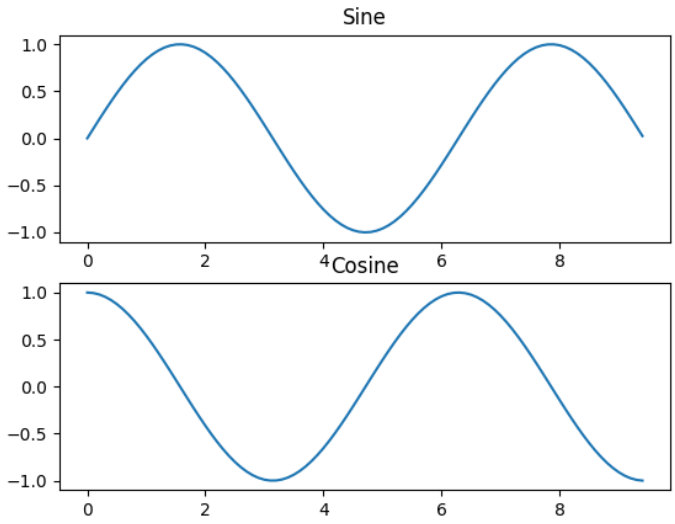
```
import numpy as np
from matplotlib import pyplot as plt

x = np.arange(1,11)
y = 2 * x + 5
plt.title("Matplotlib demo")
plt.xlabel("x axis caption")
plt.ylabel("y axis caption")
plt.plot(x,y,"ob")
plt.show()
```



2) Subplots

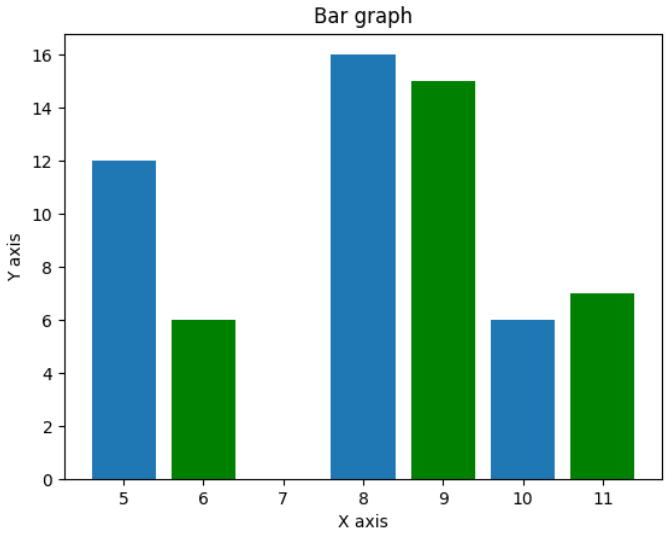
```
import numpy as np
import matplotlib.pyplot as plt

# Compute the x and y coordinates for points on sine and cosine
curves
x = np.linspace(0, 3 * np.pi, 100)

y_sin = np.sin(x)
y_cos = np.cos(x)

# Set up a subplot grid that has height 2 and width 1,
# and set the first such subplot as active.
plt.subplot(2, 1, 1)

# Make the first plot
```

```
plt.plot(x, y_sin)
plt.title('Sine')

# Set the second subplot as active, and make the second plot.
plt.subplot(2, 1, 2)
plt.plot(x, y_cos)
plt.title('Cosine')

# Show the figure.
plt.show()
```



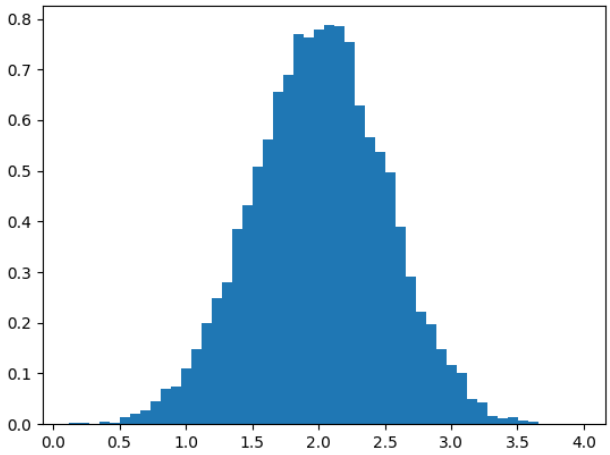
3) Barras

```
from matplotlib import pyplot as plt
x = [5,8,10]
y = [12,16,6]

x2 = [6,9,11]
y2 = [6,15,7]
plt.bar(x, y, align = 'center')
plt.bar(x2, y2, color = 'g', align =
'center')
plt.title('Bar graph')
plt.ylabel('Y axis')
plt.xlabel('X axis')

plt.show()
```



4) Histogramas

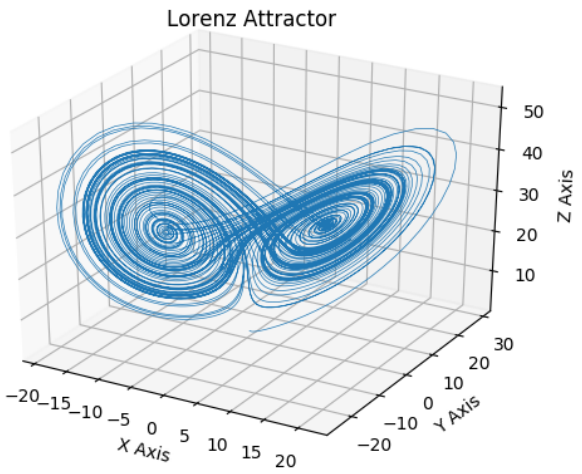
```
import numpy as np
import matplotlib.pyplot as plt
# Build a vector of 10000 normal deviates with variance 0.5^2
and mean 2
mu, sigma = 2, 0.5
v = np.random.normal(mu,sigma,10000)
# Plot a normalized histogram with 50 bins
plt.hist(v, bins=50, density=1)    # matplotlib version (plot)
plt.show()
```



5) Gráficas más elaboradas (3D plotting). Ejemplo del atractor de Lorenz

```
import numpy as np
import matplotlib.pyplot as plt
# This import registers the 3D projection, but is otherwise
unused.
from mpl_toolkits.mplot3d import Axes3D  # noqa: F401 unused
import


def lorenz(x, y, z, s=10, r=28, b=2.667):
    '''
    Given:
       x, y, z: a point of interest in three dimensional space
       s, r, b: parameters defining the lorenz attractor
    Returns:
       x_dot, y_dot, z_dot: values of the lorenz attractor's
partial
           derivatives at the point x, y, z
```

```
    '''
    x_dot = s*(y - x)
    y_dot = r*x - y - x*z
    z_dot = x*y - b*z
    return x_dot, y_dot, z_dot


dt = 0.01
num_steps = 10000

# Need one more for the initial values
xs = np.empty(num_steps + 1)
ys = np.empty(num_steps + 1)
zs = np.empty(num_steps + 1)

# Set initial values
xs[0], ys[0], zs[0] = (0., 1., 1.05)

# Step through "time", calculating the partial derivatives at the
current point
# and using them to estimate the next point
for i in range(num_steps):
    x_dot, y_dot, z_dot = lorenz(xs[i], ys[i], zs[i])
    xs[i + 1] = xs[i] + (x_dot * dt)
    ys[i + 1] = ys[i] + (y_dot * dt)
    zs[i + 1] = zs[i] + (z_dot * dt)

# Plot
fig = plt.figure()
ax = fig.gca(projection='3d')
ax.plot(xs, ys, zs, lw=0.5)
ax.set_xlabel("X Axis")
ax.set_ylabel("Y Axis")
ax.set_zlabel("Z Axis")
ax.set_title("Lorenz Attractor")

plt.show()
```



Bibliografía

[1]. Matplotlib. (s/f). Python plotting — Matplotlib 3.1.1 documentation. Recuperado el 30 de agosto de 2019, de https://matplotlib.org/

[2]. Tutorialspoint. (s/f). NumPy - Matplotlib. Recuperado el 30 de agosto de 2019, de https://www.tutorialspoint.com/numpy/numpy_matplotlib.htm

[3]. Machine Learning Plus. (s/f). NumPy Tutorial. Recuperado el 30 de agosto de 2019, de https://www.machinelearningplus.com/python/numpy-tutorial-part1-array-python-examples/

[4]. Numpy. (s/f). Quickstart tutorial — NumPy v1.18.dev0 Manual. Recuperado el 30 de agosto de 2019, de https://numpy.org/devdocs/user/quickstart.html

[5]. Tutorialspoint. (s/f). NumPy - Ndarray Object. Recuperado el 30 de agosto de 2019, de https://www.tutorialspoint.com/numpy/numpy_ndarray_object.htm

[6]. Johnson, J. (s/f). Python Numpy Tutorial. Recuperado el 30 de agosto de 2019, de http://cs231n.github.io/python-numpy-tutorial/#numpy

[7]. Scipy.org. (s/f). NumPy for Matlab users — NumPy v1.17 Manual. Recuperado el 30 de agosto de 2019, de https://docs.scipy.org/doc/numpy/user/numpy-for-matlab-users.html