ENGIN 604 Introducción a Python para las Finanzas
Introducción a NumPy - Pauta

Profesor: *Gabriel E. Cabrera*
Ayudante: *Alex Den Braber*

1. Arrays

Los objetos como las listas (`list`) son muy convenientes debido a su flexibilidad (mutables), sin embargo, este tipo de estructura tiene un costo, ocupan mucha memoria y su desempeño es bajo. Las aplicaciones financieras y/o cientificas necesitan operaciones de alto desempeño en estructuras especiales. Una de las estructura de datos más importantes son los *array*. Los *arrays* suelen estructurar otros objetos del mismo tipo de dato en las filas como en las columnas.

Si se asume que el tipo de dato más relevante es `float` e `int` (la idea se puede extender a otros tipos de datos), un `array` de una dimensión representaría, matemáticamente hablando, un vector fila o columna (depende como se defina). Los casos más comunes son las matrices que sería un `array` de dos dimensiones $i \times j$ (fila y columna).

Se necesita entonces una clase capaz de trabajar eficientemente con *arrays*. La solución es la librería **NumPy** con su clase `ndarray`. Si se quiere crear la matriz X:

$$X = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

Con la listas (`list`):

```
x = list([[1,2,3],[4,5,6],[7,8,9]])
x
```

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

Donde cada elemento anidado a la lista sería una fila de la matriz X y cada elemento del elemento anidado una columna. Si se desa ver el valor de la posición 2×2:

```
x[1][1]
```

```
## 5
```

Sin embargo, python interpreta a la variable X como una lista y no como un `array`. Utilizando `numpy`

```
import numpy as np

mat_x = np.array(x) # se genera el array
mat_x
```

```
## array([[1, 2, 3],
##        [4, 5, 6],
##        [7, 8, 9]])
```

Ahora la variable `mat_x` es un *array* y como tal posee los siguientes atributos:

- Dimensiones:

```
mat_x.ndim # posee dos dimensiones
```

```
## 2
```

- Los valores de las dimensiones:

```
m, n = mat_x.shape # número de filas y columnas
m, n
```

```
## (3, 3)
```

- Todos los elementos:

```
mat_x.size
```

```
## 9
```

- Nueva clase:

```
type(mat_x)
```

```
## <class 'numpy.ndarray'>
```

la versión `range()` que se encuentra *built-in* en Python, existe en **NumPy** como `arange()`.

```
np.arange(1,10).reshape(-1,3)
```

```
## array([[1, 2, 3],
##        [4, 5, 6],
##        [7, 8, 9]])
```

La documentación de la librería se encuentra en https://numpy.org/doc/stable/reference/.

2. NumPy

1. Genere la siguiente matriz:

$$A = \begin{pmatrix} 3 & 0 & 2 \\ 2 & 0 & 2 \\ 0 & 1 & 1 \end{pmatrix}$$

   a. Encuentre la transpuesta de la matriz A.

```
num = [3, 0, 2, 2, 0, 2, 0, 1, 1]

list_to_array = np.array(num)
a = list_to_array.reshape((3,3))

# forma 1
a.T

# forma
```

```
## array([[3, 2, 0],
##        [0, 0, 1],
##        [2, 2, 1]])
```

```
a.transpose()
```

```
## array([[3, 2, 0],
##        [0, 0, 1],
##        [2, 2, 1]])
```

b. Encuentre la matriz inversa de A.

```
b = np.linalg.inv(a)
```

c. Muestre que se cumple:

$$AB = BA = I$$

Donde B es la matriz inversa de A e I la matriz indentidad.

```
a.dot(b)

# para generar una matriz identidad
```

```
## array([[1.00000000e+00, 0.00000000e+00, 0.00000000e+00],
##        [2.22044605e-16, 1.00000000e+00, 0.00000000e+00],
##        [0.00000000e+00, 0.00000000e+00, 1.00000000e+00]])
```

```
np.eye(3)
```

```
## array([[1., 0., 0.],
##        [0., 1., 0.],
##        [0., 0., 1.]])
```

d. Mediante matrices calcule el promedio de A.

```
mean_a = np.ones((1,3)).dot(a).dot(np.ones((3,1))) / a.size
mean_a[0][0]
```

```
## 1.2222222222222223
```

2. Genere las siguientes matrices:

$$C = \begin{pmatrix} 2 & 4 \\ 5 & -6 \end{pmatrix} \quad \text{y} \quad D = \begin{pmatrix} 9 & -3 \\ 3 & 6 \end{pmatrix}$$

a. Obtenga la suma entre C y D.

```
c = np.array([[2,4],[5, -6]])
d = np.array([[9,-3],[3,6]])

c + d # suma
```

```
## array([[11,  1],
##        [ 8,  0]])
```

```
c - d # resta
```

```
## array([[ -7,   7],
##        [  2, -12]])
```

```
c * d # multiplicación
```

```
## array([[ 18, -12],
##        [ 15, -36]])
```

```
c / d # división
```

```
## array([[ 0.22222222, -1.33333333],
##        [ 1.66666667, -1.        ]])
```

b. Obtenga el producto punto entre C y D.

```
c.dot(d)
```

```
## array([[ 30,  18],
##        [ 27, -51]])
```

3. Genere una función que mutliplique (no producto punto) dos matrices.

```
mat_a = np.array([[1,2,3],[4,5,6]])
mat_b = np.array([[1,2,3],[4,5,6]])

def manual_mult(mat_a, mat_b):
  '''
  multiplica el elemento ij de la matriz a con el elemento ij de la matriz b
  '''
  m,n = mat_a.shape

  zero_mat = np.zeros((m,n))

  for i in range(m):
    for j in range(n):
      zero_mat[i][j] = mat_a[i][j] * mat_b[i][j]

  return(zero_mat)

# probamos la función
manual_mult(mat_a, mat_b)
```

```
## array([[ 1.,  4.,  9.],
##        [16., 25., 36.]])
```

3. Números Aleatorios

NumPy Posee su propio generador de números seudo-aleatorios (Mersenne Twister). El generador es una función matemática que genera una secuencia de "números aleatorios". Considera un parámetro para comenzar la secuencia de números, llamada semilla (`seed`). La función es determinística, es decir, con la misma semilla se producirá siempre la misma secuencia de números (la elección de la semilla no importa).

1. Genere 10000 valores con una distribución normal estandar $\sim N(\mu = 0, \sigma = 1)$.

   a. A partir de los valores creados genere una matriz E cuya dimensiones sea 100×100. ¿Cuál es la diferencia entre `reshape()` y `resize()`?

   ```
   np.random.seed(10)

   array_norm = np.random.randn(10000)

   # reshape
   e = array_norm.reshape((100,100))

   # resize
   array_norm.resize((100,10), refcheck=False)
   array_norm # in-place, pero no debe haber sido referenciado por ejemplo con reshape
   ```

   b. "Aplane" la matriz E y compruebe que efectivamente los datos se distribuyen con $\mu = 0$ y $\sigma = 0$.

   ```
   e_flatten_col = e.flatten(order='C') # aplanado por columna
   e_flatten_row = e.flatten(order='F') # aplanado por fila

   e_flatten_col.mean() # promedio
   e_flatten_col.std() # desviación estandar

   array_norm
   ```

4. Sistema de Ecuaciones

1. Use matrices para resolver los siguientes sistemas de ecuaciones:

   a.

$$\begin{aligned} a + b + c &= 6 \\ 3a - 2b + c &= 2 \\ 2a + b - c &= 1 \end{aligned}$$

```
# se genera las matrices (incógnita + resultado)
mat1 = np.array([[1, 1, 1], [3, -2, 1], [2, 1, -1]])
mat1_res = np.array([6, 2, 1])

# se resuelve el sistema de ecuaciones
np.linalg.solve(mat1, mat1_res)
```

```
## array([1., 2., 3.])
```

   b.

$$\begin{aligned} 3a + 4b - 5c + d &= 10 \\ 2a + 2b + 2c - d &= 5 \\ a - b + 5c - 5d &= 7 \\ 5a + d &= 4 \end{aligned}$$

```
# se genera las matrices (incógnita + resultado)
mat2 = np.array([[3, 4, -5, 1], [2, 2, 2, -1], [1, -1, 5, -5], [5, 0, 0, 1]])
mat2_res = np.array([10, 5, 7, 4])

# se resuelve el sistema de ecuaciones
np.linalg.solve(mat2, mat2_res)
```

```
## array([ 1.24778761,  1.01769912, -0.88495575, -2.23893805])
```