Programación para *Data Science*

Unidad 4: Librerías científicas en Python

Esta Unidad se compone de 4 módulos: Matplotlib, Numpy, pandas y SciPy. A continuación, os proponemos una serie de ejercicios a realizar para cada uno de estos módulos.

Ejercicios para practicar

Los siguientes 4 ejercicios no puntuan para la PEC, pero os recomendamos que los intentéis resolver antes de pasar a los ejercicios propios de la PEC. También podéis encontrar las soluciones a estos ejercicios al final del Notebook.

Ejercicio 1

Calculad la norma y el determinante de la siguiente matriz: `[[1, 0], [2, -1]]`.

In [ ]:

```
# Respuesta
```

Ejercicio 2

Evaluad las funciones arcoseno y arcocoseno en el intervalo [0,1] y con paso (resolución) de 0.1 y guardadlas en dos *arrays*.

In [ ]:

```
# Respuesta
```

Ejercicio 3

Representad en una sola gráfica las funciones arcoseno y arcocoseno en el intervalo [-pi/4, pi/4]. Añadid la leyenda correspondiente a cada gráfico.

In [ ]:

```
# Respuesta
```

Ejercicio 4

Generad una lista de 100 valores enteros aleatorios de 0-9. Realizad los siguientes cálculos utilizando métodos de *numpy*:

- Media y desviación estándar de los valores de la lista
- Valor máximo y mínimo

- Sumad todos los valores de la lista
- Conseguid una lista de valores únicos

In [ ]:

```
# Respuesta
```

Ejercicios para la PEC

A continuación encontraréis los ejercicios que debéis completar en esta PEC y que forman parte de la evaluación de esta unidad.

NumPy - Ejercicios

Ejercicio 1

Ordenad la matriz bidimensional `[[5,1,7], [0,7,4], [7,23,1]]` por filas utilizando como algoritmo de ordenación el Merge sort (https://en.wikipedia.org/wiki/Merge_sort). **(0.5 puntos)**

Hint: No es necesario que implementéis el algoritmo de ordenación a mano, NumPy contiene métodos para realizar diferentes tipos de ordenación sobre diferentes estructuras de datos.

In [ ]:

```
# Respuesta
```

Ejercicio 2

Definid una función que dadas dos matrices, devuelva el valor absoluto de la multiplicación de los determinantes ambas, es decir, dadas A y B, nuestra función devolverá `|det(A) * det(B)|` . **(0.5 puntos)**

In [ ]:

```
# Respuesta
```

Ejercicio 3

Cread una matriz 10x10 que corresponda con la matriz identidad (https://es.wikipedia.org/wiki/Matriz_identidad) usando generadores básicos de arrays. Cread una matriz identidad 10x10 (esta vez usando generadores especificos de matrices identidad) y comprobad que ambas matrices son iguales **(0.5 puntos)**

Consideraciones:

- La primera matriz debe crearse usando constructores básicos de arrays, como los presentados en los Notebooks de teoría.
- La segunda matriz debe generarse utilizando el generador de matrices identidad de numpy.
- La comparación debe devolver True si las matrices son iguales (un único True), False de no ser así.

In [ ]:

```
# Respuesta
```

Ejercicio 4

Cread una matriz de 2x6 donde los valores de cada posición `(i, j)` correspondan a `i^2+j` para todo `i` par, `i/j^2` para todo `i` impar. **(0.5 puntos)**

In [ ]:

```
# Respuesta
```

Ejercicio 5

Cread dos matrices de tamaño 5x5 con números reales aleatorios. Obtened el resultado de multiplicar ambas matrices usando los dos métodos de multiplicación de matrices vistos en el Notebook de teoría. ¿Cuál es la diferencia entre ambos resultados? Cread ahora dos matrices de tamaño 4x5 y 5x5 respectivamente, repetid la operación. Describid cuál de los métodos de multiplicación podéis aplicar y porqué. **(0.5 puntos)**

In [ ]:

```
# Respuesta
```

Matplotlib - Ejercicios

Ejercicio 1

Representad en un único gráfico las funciones f1 y f2 definidas más abajo, evaluadas en el intervalo [0, 7] y con paso (resolución) 0.1. Definid una leyenda incluyendo el significado de cada función (2x, x^2) y posicionadla de tal forma que no cubra los graficos. **(1 punto)**

In [ ]:

```
import numpy as np

def f1(x):
    return np.power(x, 2)

def f2(x):
    return np.power(2,x)
```

In [2]:

```
# Respuesta
```

Ejercicio 2

El fichero *us_births.csv* contiene datos de los nacimientos en Estados Unidos durante los años 1994-2003. Cargad el fichero y representad de forma gráfica los nacimientos agrupados por mes y año. Utilizad un gráfico 2D de tipo *scatter*. **(1.5 puntos)**

Consideraciones:

- La figura debe contener 10 gráficos (1 por año)
- El valor de cada punto debe corresponder al total de nacimientos por año y mes.
- Incluid una leyenda que contenga los años. Podéis diferenciar cada gráfico como mejor os parezca (diferentes símbolos / colores / combinación de ambos)

In [ ]:

```
# Respuesta
```

pandas - Ejercicios

Ejercicio 1

Cargad los datos del fichero *got.csv* en un *dataframe*. Este conjunto de datos recoge información de la Guerra de los Cinco Reyes (https://awoiaf.westeros.org/index.php/War_of_the_Five_Kings) de las novelas de Cancion de Hielo y Fuego (https://es.wikipedia.org/wiki/Canci%C3%B3n_de_hielo_y_fuego) de George R.R Martin.

Mostrad el número de filas del *dataframe* y las etiquetas de los ejes. **(0.5 puntos)**

In [ ]:

```
# Respuesta
```

Ejercicio 2

Agrupad los datos cargados en el ejercicio 1 por el principal bando atacante (*attacker_1*). Para cada posición, mostrad el número de batallas y el resultado de la batalla (el resultado se encuentra en el campo *attacker_outcome*). **(0.5 puntos)**

In [ ]:

```
# Respuesta
```

Ejercicio 3

Mostrad los datos de las batallas donde el numero de participantes supera los 15000 "hombres" (contando ambos bandos: *attacker_size* y *defender_size*), el resultado haya sido favorable para el atacante, y la batalla se haya producido en invierno (*summer=0*). **(1 punto)**

In [ ]:

```
# Respuesta
```

Ejercicio 4

Contad el número de lugares que aparecen más de una vez en *dataframe* (campo *location*), utilizando las funciones de la librería *pandas*. ¿Existe algún lugar donde se haya realizado más de una batalla? Comprobad qué bandos estaban implicados. **(0.5 puntos)**

In [ ]:

```
# Respuesta
```

Ejercicio 5

Añadid una nueva columna al *dataframe* con un valor booleano indicando una predicción básica de si en una cierta batalla el bando atacante será vencedor. Definiremos el valor como *True* sí el bando atacante es mayor al defensor, y como *False* en caso contrario. Definid como *NaN* aquellos casos en los que no haya número de tropas en ningún bando. ¿En que casos, identificados por el número de batalla (*battle_number*), nuestra predicción coincide con el resultado real? **(1 punto)**

In [ ]:

```
# Respuesta
```

SciPy - Ejercicios

Hint: Podéis utilizar el módulo integrate (https://docs.scipy.org/doc/scipy/reference/integrate.html#module-scipy.integrate) de SciPy para resolver diferentes tipos de integrales.

Ejercicio 1

Resolved la siguiente integral definida: **(0.5 puntos)**

$$\int_0^5 x^2 + 4x + 3dx$$

In [1]:

```
# Respuesta
```

Ejercicio 2

Resolved la siguiente integral doble definida: **(1 punto)**

$$\int_{x=0}^{\pi} \int_{y=\pi}^{2\pi} 3x\,\sin(y^2) + 6y\,\sin(x^2) dydx$$

In [ ]:

```
# Respuesta
```

Soluciones a los ejercicios para practicar

Ejercicio 1

Calculad la norma y el determinante de la siguiente matriz: `[[1, 0], [2, -1]]`.

In [31]:

```
import numpy as np

# Creamos la matriz como un array bidimensional
m = np.array([[1, 0], [2, -1]])

# Y calculamos la normal y el determinante utilizando el módulo linalg
norm = np.linalg.norm(m)
det = np.linalg.det(m)

print "La matriz m es:", m
print "La norma de m es: ", norm
print "El determinante de m es:", det
```

```
La matriz m es: [[ 1  0]
 [ 2 -1]]
La norma de m es:  2.449489742783178
El determinante de m es: -1.0
```

Ejercicio 2

Evaluad las funciones arcoseno y arcocoseno en el intervalo [0,1] y con paso (resolución) de 0.1 y guardadlas en dos arrays.

In [11]:

```
import numpy as np

# Generamos un vector con valores entre 0 y 1 y con paso 0.1. Para ello utilizamos
# que funciona de forma análoga a la función range vista en anteriores unidades.
x = np.arange(0., 1.1, 0.1)

# Calculamos el valor del arcocoseno y del arcoseno por cada valor de x utilizando
arcoseno = np.arcsin(x)
arcocoseno = np.arccos(x)

print "Arcoseno de x para x entre 0 y 1, con paso 0.1:"
print arcoseno
print "Arcocoseno de x para x entre 0 y 1, con paso 0.1:"
print arcocoseno
```

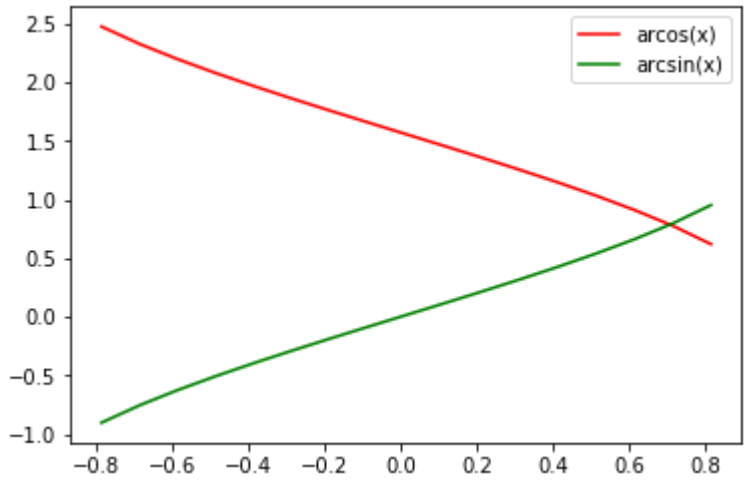
```
Arcoseno de x para x entre 0 y 1, con paso 0.1:
[0.         0.10016742 0.20135792 0.30469265 0.41151685 0.52359878
 0.64350111 0.7753975  0.92729522 1.11976951 1.57079633]
Arcocoseno de x para x entre 0 y 1, con paso 0.1:
[1.57079633 1.47062891 1.36943841 1.26610367 1.15927948 1.04719755
 0.92729522 0.79539883 0.64350111 0.45102681 0.        ]
```

Ejercicio 3

Representad en una sola gráfica las funciones arcoseno y arcocoseno en el intervalo [-pi/4, pi/4]. Añadid la leyenda correspondiente a cada gráfico.

In [12]:

```
%matplotlib inline

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

# Empezamos por calcular los valores del intervalo -pi/4 a pi/4. No se ha especifica
# elegiremos uno nosotros mismos. Para dar una resolución suficientemente alta, eleg
# mas bajo el paso, más elementos en la lista y por tanto, mayor resolución).

paso = 0.1
x = np.arange(-np.pi/4., np.pi/4. + paso, paso)

# A continuación representamos ambas funciones en el mismo gráfico
# Fijaos que los parámetros de la función plot corresponde a:
# grafico_1_eje_x, grafico_1_eje_y, color_grafico_1, grafico_2_eje_x, grafico_2_eje_

plt.plot(x, np.arccos(x), 'r', x, np.arcsin(x), 'g')

# Añadimos la leyenda al gráfico
plt.legend(["arcos(x)", "arcsin(x)"])

# Y lo mostramos por pantalla
plt.show()
```



Ejercicio 4

Generad una lista de 100 valores enteros aleatorios de 0-9. Realizad los siguientes cálculos utilizando métodos de *numpy*:

- Media y desviación estándar de los valores de la lista
- Valor máximo y mínimo
- Sumad todos los valores de la lista
- Conseguid una lista de valores únicos

In [33]:

```
from random import randint
import numpy as np

# Utilizamos la función randint del módulo random y list comprehensions para genera
# unidades anteriores
rand_ints = [randint(0, 9) for _ in range(100)]

# Para calcular la media y desviación estándar utilizamos las funciones mean y std
m = np.mean(rand_ints)
std = np.std(rand_ints)

print "Nuestra lista de valores aleatorios es:", rand_ints
print "Su valor medio es %f, y su desviación estándar es %f" %(m, std)

# Para valores máximo y mínimo, utilizamos max y min
max_val = np.max(rand_ints)
min_val = np.min(rand_ints)

print "Sus valores máximo y mínimo son, respectivamente: %d y %d" %(max_val, min_va

# Para sumar todos los elementos de la lista, la función sum
sum_values = np.sum(rand_ints)

print "El resultado de la suma de todos los valores de la lista es", sum_values

# Y finalmente para conseguir una lista de valores únicos, la función unique
unique_values = np.unique(rand_ints)

print "Y la lista de valores únicos es", unique_values
```

```
Nuestra lista de valores aleatorios es: [1, 9, 3, 6, 1, 0, 0, 1, 7, 7,
5, 8, 7, 6, 2, 8, 7, 7, 0, 6, 9, 9, 0, 0, 2, 3, 6, 7, 2, 5, 9, 2, 5,
1, 5, 6, 9, 5, 4, 1, 2, 0, 3, 0, 7, 4, 2, 0, 0, 7, 5, 5, 7, 3, 2, 1,
8, 9, 3, 9, 9, 4, 0, 6, 3, 6, 9, 6, 9, 1, 4, 0, 9, 0, 1, 8, 2, 9, 4,
5, 8, 2, 9, 7, 2, 1, 4, 1, 1, 4, 8, 9, 0, 7, 3, 0, 1, 3, 6, 4]
Su valor medio es 4.330000, y su desviación estándar es 3.095335
Sus valores máximo y mínimo son, respectivamente: 9 y 0
El resultado de la suma de todos los valores de la lista es 433
Y la lista de valores únicos es [0 1 2 3 4 5 6 7 8 9]
```