prog_datasci_4_scilib_entrega

November 1, 2019

1 Programación para *Data Science*

1.1 Unidad 4: Librerías científicas en Python

En este Notebook encontraréis dos conjuntos de ejercicios: un primer conjunto de **ejercicios para practicar** y un segundo conjunto de **actividades evaluables** como PEC de la asignatura.

En cuanto al conjunto de ejercicios para practicar, éstos no puntúan para la PEC, pero os recomendamos que los intentéis resolver como parte del proceso de aprendizaje. Encontraréis ejemplos de posibles soluciones a los ejercicios al propio notebook, pero es importante que intentéis resolverlos vosotros antes de consultar las soluciones. Las soluciones os permitirán validar vuestras respuestas, así como ver alternativas de resolución de las actividades. También os animamos a preguntar cualquier duda que surja sobre la resolución de los **ejercicios para practicar** en el foro del aula.

En relación a las actividades evaluables, veréis que cada una de ellas tiene asociada una puntuación que indica el peso que tiene la actividad sobre la nota de la PEC. Adicionalmente, hay un ejercicio opcional, que no tiene puntuación dentro de la PEC, pero que se valora al final del semestre de cara a conceder las matrículas de honor y redondear las notas finales. Podéis sacar la máxima nota de la PEC sin necesidad de hacer este ejercicio! El objetivo de este ejercicio es que sirva como pequeño reto para los estudiantes que quieran profundizar en el contenido de la asignatura.

Además, veréis que todas las actividades tienen una etiqueta que indica los recursos necesarios para llevarla a cabo. Hay tres posibles etiquetas:

- NM **Sólo materiales**: las herramientas necesarias para realizar la actividad se pueden encontrar en los materiales de la asignatura.
- EG **Consulta externa guiada**: la actividad puede requerir hacer uso de herramientas que no se encuentran en los materiales de la asignatura, pero el enunciado contiene indicaciones de dónde o cómo encontrar la información adicional necesaria para resolver la actividad.
- EI **Consulta externa independente**: la actividad puede requerir hacer uso de herramientas que no se encuentran en los materiales de la asignatura, y el enunciado puede no incluir la descripción de dónde o cómo encontrar esta información adicional. Será necesario que el estudiante busque esta información utilizando los recursos que se han explicado en la asignatura.

Es importante notar que estas etiquetas no indican el nivel de dificultad del ejercicio, sino únicamente la necesidad de consulta de documentación externa para su resolución. Además, recordad que las **etiquetas son informativas**, pero podréis consultar referencias externas en cualquier

momento (aunque no se indique explícitamente) o puede ser que podáis hacer una actividad sin consultar ningún tipo de documentación. Por ejemplo, para resolver una actividad que sólo requiera los materiales de la asignatura, puedéis consultar referencias externas si queréis, ya sea tanto para ayudaros en la resolución como para ampliar el conocimiento!

En cuanto a la consulta de documentación externa en la resolución de los ejercicios, recordad **citar siempre la bibliografía utilizada** para resolver cada actividad.

1.2 Ejercicios para practicar

Los siguientes 3 ejercicios no puntúan para la PEC, pero os recomendamos que los intentéis resolver antes de pasar a los ejercicios propios de la PEC. También encontraréis las soluciones a estos ejercicios al final del Notebook.

1.2.1 Ejercicio 1

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)|`.

Nota: Podéis consultar la documentación del módulo linalg para resolver este ejercicio.

EG

[ ]:
```
# Respuesta
```

1.2.2 Ejercicio 2

Cread una matriz 10x10 que corresponda con la 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.

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.

Nota: Podéis consultar la documentación sobre creación de matrices de Numpy para resolver este ejercicio.

EG

[ ]:
```
# Respuesta
```

1.2.3 Ejercicio 3

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. NM

[ ]:
```
import numpy as np

def f1(x):
```

```
    return np.power(x, 2)

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

[ ]:
```
# Respuesta
```

1.3 Ejercicios y preguntas teóricas para la PEC

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

NumPy - Ejercicios

1.3.1 Ejercicio 1

Cread dos matrices, A y B, de mida 5x5 con números enteros aleatorios entre 1 y 10. A continuación cread una tercera matriz, C, de mida 2x5 a partir de la última fila de la matriz A y la primera fila de la matriz B: la primera fila de la matriz C será la última de la matriz A, y la segunda fila de la matriz C será la primera fila de la matriz B. Mostrad por pantalla el resultado obtenido.

(0.5 punts) NM

[ ]:
```
# Respuesta
```

1.3.2 Ejercicio 2

Dadas las matrices A y B del ejercicio anterior, mostrad por pantalla el resultado de las siguientes operaciones:

- Sumar las dos matrices
- Restar la matriz B de la matriz A
- Multiplicar las dos matrices

Explicad brevemente cuál es el método de multiplicación escogido y porque.

(0.5 punts) NM

[ ]:
```
# Respuesta
```

1.3.3 Ejercicio 3

Ordenad la matriz bidimensional `[[5,1,7], [0,7,4], [7,23,1]]` por columnas utilizando como algoritmo de ordenación el Heapsort.

Nota: Para resolver este ejercicio os puede ser de ayuda la función sort de Numpy que nos permite ordenar los elementos de un array N-dimensional.

(0.5 punts) EG

[ ]:
```
# Respuesta
```

1.3.4 Ejercicio 4

Cread una matriz, I, de mida 10x10 donde todos sus elementos tendrán el valor `0` utilizando generadores de matrices de numpy. Modificad la matriz anterior de tal forma que `I[i, j] = i * j` para todo `i` y `j` pares, es decir, para todas las posiciones donde tanto `i` como `j` son pares.

(0.5 punts) NM

[ ]:
```
# Respuesta
```

1.3.5 Ejercicio 5

Una técnica que se aplica en muchos algoritmos de compresión de imagenes, como fase previa a la compresión, es asignar a cada píxel de la imagen el valor medio de sus N píxeles vecinos. Nosotros implementaremos una versión simplificada de esta técnica.

Una imagen se puede ver como una matriz bidimensional donde cada elemento de la matriz corresponde a un valor de un píxel de la imagen. Por ejemplo, si generamos una imágen a partir de la matriz que os proporcionamos más abajo, veremos una imagen en escala de grises. Para probarlo podéis ejecutar la siguiente intrucción: `generate_png(pixels)`. Esto os generará un fichero, `generated.png`, en el directorio de la PAC.

Cread una nueva matriz, `pixels_transform`, a partir de la matriz `pixels` donde el valor de cada elemento de `pixels_transform` se calculará como la media de los valores de los 4 vecinos más próximos del correspondiente píxel en la matriz `pixels`. Veámoslo formalmente:

```
pixels_transform[i, j] = (pixels[i, j-1] + pixels[i-1, j] + pixels[i, j+1] + pixels[i+1, j]) /
```

Consideraciones:

- Consideramos como los 4 vecinos más próximos los píxeles anterior (izquierda), superior (arriba), posterior (derecha) e inferior (abajo) del píxel que estamos analizando.
- Si no podemos acceder a alguno de estos vecinos (e.g. el píxel (0, 0) no tiene vecinos anterior y superior), asumimos que los valores que nos faltan son 0.

Mostrad por pantalla el resultado obtenido.

Nota: Para resolver este ejercicio necesitamos recorer todos los elementos de la matriz `pixels` y para cada elemento tenemos que comprobar los valores de sus 4 vecinos. Necesitaremos un bloque condicional (`if`) para cada uno de los elementos vecinos.

(1.5 punts) NM

[ ]:
```
from PIL import Image
import numpy as np

pixels = [
    [54, 93, 71, 168],
    [122, 54, 167, 93],
    [71, 167, 54, 122],
    [167, 82, 23, 54]
]

# Podéis utilizar la siguiente función para generar una imagen a partir de un␣
 ↪array de píxeles.
```

```python
# Por ejemplo, generate_png(pixels)
def generate_png(matriz):
    # Nos aseguramos que la matriz es un array de numpy
    pixels = np.array(matriz, dtype=np.uint8)

    # Utilizamos la librería PIL para crear una imagen a partir de una array de␣
 ↪píxeles
    imagen = Image.fromarray(pixels)
    imagen.save('generated.png')
```

[ ]:
```python
# Respuesta
```

Matplotlib - Ejercicios

1.3.6 Ejercicio 1

Representad en una única gráfica las funciones f1 y f2 definidas más abajo, evaluadas en el intervalo [0, 2] y con paso (resolución) 0.01. Haced que los rangos de valores mostrados sean `[0.25, 2.25]` para el eje de las X y `[-1, 1]` para el eje de las Y. Adicionalmente, mostrad la gráfica correspondiente a `f1` con color verde y la gráfica correspondiente a `f2` con color rojo.

(1 punt) NM

[ ]:
```python
import numpy as np

def f1(x):
    return np.exp(-x)

def f2(x):
    return np.exp(-x) * np.sin(2 * np.pi * x)
```

[ ]:
```python
# Respuesta
```

1.3.7 Ejercicio 2

A continuación os facilitamos los resultados de las elecciones generales, que tuvieron lugar en España el pasado mes de abril del 2019, para los 10 partidos más votados. Mostrad dos gráficas de barras con los resultados del número de votos y del número de diputados para cada partido respectivamente.

Consideraciones:

- Debéis mostrar las dos gráficas una al lado de la otra (distribución horizontal).
- Cada gráfica tiene que mostrar los datos de manera horizontal (barras horizontales).
- El eje de las Y debe incluir las etiquetas con el nombre de los partidos.
- Debeís incluir un texto con el número de votos para cada partido dentro del gráfico. El texto debería estar situado a la derecha de cada barra.

Nota: En el Notebook de teoria hemos visto como mostrar gráficas lineares simples. Matplotlib nos ofrece muchos otros tipos de gráficas, siendo las gráficas de barra uno de ellos. Para resolver este ejercicio podéis consultar la documentación oficial de Matplotlib para ver como trabajar con

este tipo de gráficas. También os pueden resultar útiles las funciones text o annotate para mostrar el nombre de votos.

(1.5 punts) EG

Partido	Votos	Diputados
PSOE	7513142	123
PP	4373653	66
Cs	4155665	57
PODEMOS-IU-EQUO	2897419	33
VOX	2688092	24
ERC	1020392	15
ECP-GUANYEM	615665	7
JxCAT	500787	7
EAJ-PNV	395884	6
PACMA	328299	0

[ ]:
```
# Respuesta
```

pandas - Ejercicios

1.3.8 Ejercicio 1

Cargad los datos del fichero *amazon.csv*, que podéis encontrar en la carpeta `data`, en un *dataframe*. Este conjunto de datos recoge información sobre el número de incendios en bosques de Brasil entre el 1998 y el 2017.

Mostrad el número de filas del *dataframe* y el nombre de las columnas.

Consideraciones:

- Debéis mostrar únicamente la información que os pedimos al enunciado.

(0.5 punts) NM

[ ]:
```
# Respuesta
```

1.3.9 Ejercicio 2

Agrupad los datos cargados al ejercicio 1 por **año** y, para cada año, mostrad el número total de incendios que tuvierón lugar en todo el país.

Nota: Al Notebook de teoria hemos visto como calcular la media de una agrupación de datos. Para resolver este ejercicio tendréis que investigar como sumar todos los valores de una columna en una agrupación. Os puede ser de utilidad la función aggregate de pandas, la cual nos permite aplicar diferentes funciones a una columna en un `groupby`.

(0.5 punts) EG

[ ]:
```
# Respuesta
```

1.3.10 Ejercicio 3

Mostrad los estados de Brasil donde se registraron más de *800* incendios al mes, contando los meses de *junio*, *julio* y *agosto*. Debéis mostrar como resultado una lista de los estados **sin repetir**.

Nota: Al Notebook de teoria hemos visto operaciones básicas de filtrado. Para resolver este ejercicio deberéis investigar como aplicar condiciones más complejas. Para extraer una lista de valores *únicos* de una columna de un dataframe os puede ser útil la función unique de pandas.

(1 punt) EG

[ ]:
```
# Respuesta
```

1.3.11 Ejercicio 4

Queremos buscar aquellos estados donde, durante el año *2017*, tuvierón lugar *más de 800* incendios al mes, a lo largo de cualquier mes del año. Debéis mostrar por pantalla el **estado**, el **mes** (o los meses si la condición se cumple en varios meses) y el **nombre total de incendios** registrados en el mes correspondiente.

Nota: Además del contenido de los Notebook de teoria, podéis utilizar lo que habéis aprendido haciendo el ejercicio 2.

(0.5 punts) NM

[ ]:
```
# Respuesta
```

1.3.12 Ejercicio 5

Cread una copia del dataframe original añadiendo una nueva columna que nos indicará para cada fila el nivel de riesgo de dicha fila. Definimos tres niveles de riesgo:

- **high** : si se ha registrado una fila con más de 800 incendios
- **medium** : si se ha registrado una fila con un nombre de incendios entre 400 y 800
- **low** : si se ha registrado una fila con menos de 400 incendios

Mostrad por pantalla aquellas filas que correspondan a un nivel de riesgo alto (*high*).

Nota: pandas nos ofrece principalmente tres maneras de añadir nuevas columnas a un dataframe existente. Cualquiera de las tres formas son válidas para resolver este ejercicio. Podéis buscar en Internet como añadir columnas a un dataframe de pandas, ya que la documentación presenta los tres métodos de forma separada.

(1.5 punts) EG

[ ]:
```
# Respuesta
```

1.4 Soluciones ejercicios para practicar

1.4.1 Ejercicio 1

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)|`.

Nota: Podéis consultar la documentación del módulo linalg para resolver este ejercicio.

EG

[ ]:
```
# El enunciado nos pide realizar dos operaciones: el determinante de matrices y␣
 ↪el valor absoluto.
# Ambas funciones se encuentran disponibles a numpy.
from numpy.linalg import det
from numpy import absolute

# Definimos la función
def abs_det(A, B):
    return absolute(det(A) * det(B))
```

1.4.2 Ejercicio 2

Cread una matriz 10x10 que corresponda con la 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.

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.

Nota: Podéis consultar la documentación sobre creación de matrices de Numpy para resolver este ejercicio.

EG

[ ]:
```
import numpy as np

# Podemos definir la matriz de forma manual como una lista de listas
m0 = np.array([[1, 0, 0, 0, 0, 0, 0, 0, 0, 0,], [0, 1, 0, 0, 0, 0, 0, 0, 0, 0,],
               [0, 0, 1, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
               [0, 0, 0, 0, 1, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
               [0, 0, 0, 0, 0, 0, 1, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 1, 0, 0],
               [0, 0, 0, 0, 0, 0, 0, 0, 1, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 1]])

# O empezando por una 10x10 con todos los elementos iguales a 0 y asignar 1s a␣
 ↪la diagonal
# recorriendo la matriz por filas y columnas
m1 = np.zeros((10,10), dtype=int)
for i, r in enumerate(m1):
    r[i] = 1

# Por otro lado, definimos la identidad 10x10 utilizando el constructor␣
 ↪built-in"
m2 = np.identity(10, dtype=int)

# Y realizamos la comparación. Para hacerlo podemos utilizar el método␣
 ↪array_equal de numpy
```

```
# que nos compara si dos arrays son iguales
print(np.array_equal(m0, m2))
print(np.array_equal(m1, m2))

# También podemos comprobar que la comparación falla si modificamos cualquier␣
 ↪valor de la matriz
m0[1, 2] = 1
print(np.array_equal(m0, m2))
```

1.4.3 Ejercicio 3

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. NM

[ ]:
```
import numpy as np

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

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

[ ]:
```
%matplotlib inline
import matplotlib.pyplot as plt

# El enunciado nos indica que el punto inicial es 0, el final es 7 y la␣
 ↪resolució es 0.1
start = 0
end = 7
res = 0.1


# Definimos los arrays de valores con arange. Tenemo que sumar 0.1 al final␣
 ↪dado que arange
# funciona de forma análoga a range [inicio, final)

# Ambas gráficas comparten el mismo eje x
x = np.arange(start, end+res, res)

# Y los arrays a y b corresponden a los ejes y para f1 y f2 respectivamente.␣
 ↪Para calcularlos
# simplemente evaluamos las dos funciones con los valores de x
a = [f1(val) for val in x]
b = [f2(val) for val in x]

plt.plot(x, a)
plt.plot(x, b)
```

```
plt.legend(["f1", "f2"])
plt.show()
```