Sessió 2

Taula de continguts

Paquets Científics **1**
El paquet `Numpy` 1
El paquet `Pandas` 8
El paquet `Matplotlib` 11

Paquets Científics

> **Nota**
>
> Els documents d'aquest curs contenen adaptacions de diferents materials que s'han anat utilitzant a altres cursos i conté aportacions de varis membres del Departament de Matemàtiques de la UAB.

El paquet `Numpy`

`Python` és un llenguatge de programació interpretat. Això vol dir que hi ha un programa base, l'`intèrpret de Python`, que analitza i executa les instruccion que nosaltres programem directament a partir del codi font. Això permet, per exemple, poder tenir un entorn com el `Jupyter Notebook`, on anem veient en temps real el resultat de l'execució del codi `Python`.

Un altre aspecte on `Python` és flexible, també per facilitar la feina del programador, és en els tipus de les variables. Per exemple, una `llista` o un `diccionari` poden contenir elements de diferents tipus (`int`, `float`, `string`, etc).

Això fa que, normalment, un programa escrit en `Python` trigui més en executar-se que un programa equivalent escrit en un llenguatge compilat i fortament tipat (com `C` o `C++`).

Per millorar l'eficiència dels llenguatges escrits en `Python`, molts dels paquets que contenen funcions que podrien trigar en executar-se estan escrits en `C`. Llavors, aquests paquets escrits en `C` contenen una interfície que es pot utilitzar desde `Python`, de forma que podem tenir la comoditat d'escriure un programa en `Python` aprofitant l'eficiència de `C`.

Un dels paquets més importants de `Python` que permet fer càlculs numèrics de forma eficient és `Numpy`.

Per utilitzar el paquet `Numpy` primer l'hem d'importar. És molt comú importar el paquet `Numpy` assignant-li l'abreviatura `np`.

```
import numpy as np
```

El tipus de variable amb que el treballa `Numpy` és l'`array`. Els `arrays` son generalitzacions de matrius, on podem tenir valors organitzats en diferents dimensions. Per exemple, un vector és un `array` d'una dimensió i una matriu, amb valors organitzats en files i columnes, és un `array` amb dues dimensions. Els `arrays` a Numpy poden tenir un número de dimensions arbitrari, i cada dimensió pot tenir un número de components també arbitrari.

Per crear `arrays` de `Numpy` es pot fer directament a través de llistes de `Python`.

```
a = np.array([1, 2, 3])
a
```

Podem obtenir llavors el número de components de cada dimensió de l'array (en aquest cas només una dimensió):

```
a.shape
```

Per crear matrius (o `arrays` de vàries dimensions) es poden utilitzar llistes encaixades:

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

```
a.shape
```

`Numpy` proporciona diverses funcions per generar `arrays` amb valors predeterminats:

```
a = np.zeros(shape=(2, 3))
a
```

```
a = np.ones(shape=(3, 2))
a
```

```
a = np.eye(5)
a
```

En cas necessari es poden canviar les dimensions d'un `array` i els valors de cada component es reorganitzen automàticament.

```
a = np.arange(15)
a
```

```
a.reshape(3, 5)
```

```
a.reshape(5, 3)
```

Els arrays de `Numpy` de més de dues dimensions es poden generar de forma anàloga, però la visualització per pantalla és menys interpretable.

```
a = np.arange(30).reshape(5, 3, 2)
a
```

Per accedir als valors continguts en un array es pot fer servir un `slicing` equivalent al de les `llistes` de `Python` però en cada component.

```
a = np.arange(15).reshape(5, 3)
a[1:3, 1:]
```

Una opció que pot ser molt útil és indexar un `array` amb valors `True` i `False`.

```
a = np.arange(15).reshape(5, 3)
idx = (a % 4 == 0)
idx
```

```
a[idx]
```

Exercici

Definiu un `array` amb els números del 0 al 23 organitzats en una `matriu` de mida 6 x 4. Després utilitzeu l'`slicing` per seleccionar les components que corresponen a files i

columnes parells (la segona fila / segona columna, la segona fila / quarta columna, etc.)

Els arrays de `Numpy` es poden operar component a component amb els operadors habituals de `Python`.

```
a = np.ones(shape=(5,3))
b = np.arange(15).reshape(5, 3)
a+b
```

```
a = np.ones(shape=(5,3))
b = np.arange(15).reshape(5, 3)
a-b
```

Exercici

Definiu una matriu de mida 4x4 amb els números de l'1 al 16. Llavors, definiu una matriu de mida 4x4 que tingui el número 3 a la diagonal (la resta de valors a 0). Per últim, multipliqueu les components de les dues matrius per extraure la diagonal de la matriu original multiplicada per 3.

En el cas en que les dimensions dels arrays no siguin iguals, `Numpy` intenta fer `broadcasting`, de forma que els valors dels arrays es repeteixen les vegades que siguin necessàries per igualar les dimensions.

```
a = np.arange(3)
b = np.array(5)
print("a:\n{}\n".format(a))
print("b:\n{}\n".format(b))
print("a + b:\n{}\n".format(a+b))
```

```
a = np.ones(shape=(3,3))
b = np.arange(3)
print("a:\n{}\n".format(a))
print("b:\n{}\n".format(b))
print("a + b:\n{}\n".format(a+b))
```

```
a = np.ones(shape=(3,3))
b = np.arange(3).reshape(3,1)
print("a:\n{}\n".format(a))
print("b:\n{}\n".format(b))
print("a + b:\n{}\n".format(a+b))
```

```
a = np.arange(3).reshape(3,1)
b = np.arange(3)
print("a:\n{}\n".format(a))
print("b:\n{}\n".format(b))
print("a + b:\n{}\n".format(a+b))
```

`Numpy` ens permet també calcular diversos estadístics donat un `array`.

```
a = np.arange(15).reshape(3,5)
a
```

```
print("Mitjana de tots els elements: {}".format(a.mean()))
print("Mitjana dels elements de cada columna: {}".format(a.mean(axis=0)))
print("Mitjana dels elements de cada fila: {}".format(a.mean(axis=1)))
```

```
print("Suma de tots els elements: {}".format(a.sum()))
print("Suma dels elements de cada columna: {}".format(a.sum(axis=0)))
print("Suma dels elements de cada fila: {}".format(a.sum(axis=1)))
```

Exercici

Definiu una funció `covariancia` amb dos paràmetres `x` i `y` que calculi la covariància entre els dos vectors.
Recordeu que la covàriancia entre els dos vectors ve definida per:

$$c = \sum (x_i - \bar{x})(y_i - \bar{y})$$

On $\bar{x}$ i $\bar{y}$ reprensenten la mitjana dels vectors x (`x`) i y (`y`) respectivament.

Exercici

El model lineal simple ve definit per la següent equació

$$y = mx + b$$

Donat un conjunt de dades $\{(x_i, y_i)\}_I$, podem posar tots els valors de la variable independent en un vector x i tots els valors de la variable resposta en un vector y. Llavors, el valors de m i b que millor aproximen les dades venen donats per

$$m = \frac{cov(x, y)}{cov(x, x)}$$
$$b = \bar{y} - m\bar{x}$$

Calculeu els valors m i b que millor aproximen el següent conjunt de dades:

$$\{(1, 5.1), (2, 7.8), (3, 11.0), (4, 13.9)\}$$

```
x = np.array([1, 2, 3, 4])
y = np.array([5.1, 7.8, 11.0, 13.9])
```

Els `arrays` de `Numpy` es poden concatenar indicant la dimensió que es vol utilitzar. Per una matriu, seria triar si es volen concatenar horitzontalment o verticalment.

```
a = np.array([[1, 2], [3, 4]])
b = np.array([[5, 6], [7, 8]])
c = np.concatenate((a, b), axis=0)

print("a:\n{}\n".format(a))
print("b:\n{}\n".format(b))
print("Contatenació eix 0:\n{}\n".format(c))
```

```
a = np.array([[1, 2], [3, 4]])
b = np.array([[5, 6], [7, 8]])
c = np.concatenate((a, b), axis=1)

print("a:\n{}\n".format(a))
print("b:\n{}\n".format(b))
print("Contatenació eix 1:\n{}\n".format(c))
```

Donat un `array` que correspongui a una matriu, podem utilitzar `Numpy` per generar la matriu transposada.

```
a = np.arange(15).reshape(3, 5)
a
```

```
np.transpose(a)
```

És important observar que si utilitzem la notació de `Python` per multiplicar dos `arrays` de `Numpy`, la multiplicació es farà component a component. En particular, aquesta operació no correspon al producte habitual de matrius. Per calcular el producte de matrius s'ha de demanar específicament a `Numpy`.

```
a = np.arange(15).reshape(3, 5)
np.matmul(a, np.transpose(a)) # Diferent d'utilitzar l'operació * (producte component a compo
```

Una altra operació típica amb matrius que podem fer amb `Numpy` és calcular la inversa.

```
a = np.array([[1, 1, 1], [1, 1, 2], [1, 2, 3]])
a
```

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

Podem utilitzar el càlcul d'inversa d'una matriu per resoldre sistemes d'equacions lineals amb `Numpy`. Per exemple, considerem el següent sistema:

$$x + y + z = 1$$
$$x + y + 2z = 0$$
$$x + 2y + 3z = -1$$

Llavors, podem calcular

```
a = np.array([[1, 1, 1], [1, 1, 2], [1, 2, 3]])
b = np.array([1, 0, -1]).reshape(3, 1)
```

```
np.matmul(np.linalg.inv(a), b)
```

Exercici

Si tenim un model de regressió lineal múltiple, la matriu de disseny és una matriu X on cada fila conté un registre i cada columna representa una variable explicativa. Si denotem el vector de la variable resposta com Y i β el vector de paràmetres, tenim que els valors òptims dels paràmetres s'obtenen amb la següent fórmula:

$$\beta = (X^T X)^{-1} X^T Y$$

On hem assumit que la primera columna de la matriu X conté sempre el valor 1.
Donada la matriu X i el vector y a continuació, calculeu els paràmetres òptims β.

```
X = np.array([[1, 1, 1], [4, 5, 6], [9, 8, 7], [0, 3, 3], [2, 1, 1], [6, 7, 8]])
y = np.array([2, 3, 4, 5, 6, 7])
```

El paquet `Pandas`

El paquet `Numpy` ens permet treballar amb `arrays` de dades, i en particular amb matrius, de forma molt eficient, però els `arrays` no tenen gaire informació de context, en particular no tenim cap índex a les files ni cap nom que ens permeti distingir les columnes.

El paquet `Pandas` precisament afegeix aquesta informació de context (índex i noms de columnes) als `arrays` de `Numpy`.

Per utilitzar el paquet `Pandas` primer s'ha d'importar, i és molt habitual importar el paquet `Pandas` assignant-li l'abreviatura `pd`.

```
import pandas as pd
```

El tipus de variable principal de `Pandas` és el `DataFrame`, que es correspon amb el que seria una `matriu` de dades, on cada fila i cada columna tenen un nom assignat.

Per crear un `DataFrame` de `Pandas` ho podem fer a partir de llistes de `Python`, o a partir d'`arrays` de `Numpy` o a partir de diccionaris.

```
df = pd.DataFrame(np.ones(shape=(3,3)))
df
```

```
d = {"id": [0, 1, 2], "nom": ["a", "b", "c"], "edat": [20, 21, 22]}
df = pd.DataFrame(d)
df
```

També es pot crear un `DataFrame` de `Pandas` a partir d'un índex i una llista de columnes i omplir-lo amb un valor per defecte.

```
df = pd.DataFrame(0, index=np.arange(5), columns=["id", "nom", "edat"])
df
```

`Pandas` ofereix moltes utilitats per treballar amb `índex` definits a partir de dates.

```
date_range = pd.date_range("2024-01-01", "2024-12-31", freq="D")
df = pd.DataFrame(0, index=date_range, columns=["id", "nom", "edat"])
df
```

Amb els mètodes `set_index` i `reset_index` es pot fixar quina columna ha de ser l'índex del `DataFrame`.

```
date_range = pd.date_range("2024-01-01", "2024-12-31", freq="D")
df = pd.DataFrame(0, index=date_range, columns=["id", "nom", "edat"])
df.reset_index()
```

```
date_range = pd.date_range("2024-01-01", "2024-12-31", freq="D")
df = pd.DataFrame(0, index=date_range, columns=["id", "nom", "edat"])
df = df.reset_index()
df = df.rename(columns={"index": "data"})
df.set_index("id")
```

Per accedir als valors d'un `DataFrame`, `Pandas` implementa dos mètodes, `loc` i `iloc`, que permeten fer un `slicing` similar al de `Numpy`.

El mètode `loc` selecciona els registres a partir del valor de l'índex, mentre que el mètode `iloc` fa la selecció a partir de la posició dels registres.

```
date_range = pd.date_range("2024-01-01", "2024-12-31", freq="D")
df = pd.DataFrame(0, index=date_range, columns=["id", "nom", "edat"])
df
```

```
df.loc["2024-06-10":"2024-06-17"] # Observeu que Pandas inclou el registre corresponent a la
```

```
df.iloc[161:167] # Mentre que si accedim amb iloc, Pandas no inclou el registre corresponent
```

Per accedir a les columnes ho podem fer directament especificant el nom de la columna entre claudàtors.

```
df["id"]
```

Si volem accedir a més d'una columna, hem de posar entre claudàtors una llista de noms, de forma que, si ho escribim directament, queden dos claudàtors junts.

```
llista_columnes = ["nom", "edat"]
df[llista_columnes]
```

```
df[["nom", "edat"]]
```

Els mètodes `loc` i `iloc` permeten també filtrar a la vegada una sèrie de files i columnes.

```
df.loc["2024-01-01", "nom"]
```

També podem fer servir la notació per extreure files i columnes en l'assignació de nous valors, igual que fèiem amb les llistes de `Python` i els `arrays` de `Numpy`.

```
df.loc["2024-01-01", "nom"] = "Toni"
df
```

Per afegir columnes al `DataFrame` simplement hem d'assignar un valor a cada fila i anomenar la nova columna.

```
df["num"] = range(len(date_range))
df
```

El mètode `loc` accepta que li passem una sèrie de valors `True` i `False`, on els valors `True` indiquen les files que volem extreure.

```
idx = (df["num"] % 100 == 0)
idx
```

```
df.loc[idx]
```

Aquest tipus de selecció s'acostuma a escriure directament en una línia, sense definir explícitament la variable `idx`.

```
df.loc[df["num"] % 100 == 0]
```

Exercici

Utilitzeu el següent `DataFrame` per extreure les files on la variable `id` és múltiple de 5 i la variable edat és més gran que 20.

```
date_range = pd.date_range("2024-01-01", "2024-12-31", freq="D")
df = pd.DataFrame(0, index=date_range, columns=["id", "nom", "edat"])
df["id"] = range(len(date_range))
df["edat"] = range(len(date_range))
df["edat"] = df["edat"] % 40
df
```

De forma anàloga a `Numpy`, `Pandas` també proporciona mètodes per extreure estadístiques d'un `DataFrame`.

```
df = pd.DataFrame(np.random.randint(0, 100, size=(100, 4)), columns=["A", "B", "C", "D"])
df
```

```
df.describe()
```

```
df.mean()
```

```
df.sum()
```

El paquet `Matplotlib`

El paquet `Matplotlib` és un dels paquest amb els que es poden generar visualitzacions en `Python`. Per utilitzar-ho, importarem el submòdul `pyplot`, que s'acostuma a abreviar com `plt`.

```
import matplotlib.pyplot as plt
```

El paquet `Matplotlib` està molt integrat amb `Numpy`, pel que norlament es fan servir arrays per indicar a `Matplotlib` les dades que volem visualitzar.

Per visualitzar la gràfica d'una funció f només hem d'indicar els valors de x que volem fer servir acompanyats dels valors $f(x)$. `Matplotlib` s'encarregarà llavors d'unir els valors que li hem donat amb una linia.

```
x = np.linspace(0, 8*np.pi, 100)
y = np.sin(x)
```

```
plt.plot(x, y)
plt.show()
```

Podem modificar fàcilment atributs de la visualització, com el color o el gruix de linia.

```
plt.plot(x, y, color="orange", linewidth=5)
plt.show()
```

El paquet `Matplotlib` ens permet també mostrar diferents gràfiques en una mateixa visualització. La forma més simple de fer-ho és repetir la crida a la instrucció `plot`.

```
x = np.linspace(0, 8*np.pi, 100)
y1 = np.sin(x)
y2 = np.cos(x)
```

```
plt.plot(x, y1)
plt.plot(x, y2)
plt.show()
```

Per tenir clar quina gràfica fa referència a cada funció podem afegir una etiqueta a cada `plot` i llavors demanar a `Matplotlib` que afegeixi una llegenda a la visualització.

```
plt.plot(x, y1, label="sinus")
plt.plot(x, y2, label="cosinus")
plt.legend()
plt.show()
```

`Matplotlib` ens permet generar visualitzacions de diferents tipus. Per exemple, si volem mostrar directament els punts sense cap linia que els uneixi podem utilitzar el mètode `scatter`.

```
plt.scatter(x, y1, label="sinus")
plt.scatter(x, y2, label="cosinus")
plt.legend()
plt.show()
```

Exercici

Genereu una visualització amb els punts definits en l'exercici de la regressió lineal simple. Afegiu també a la visualització la recta definida pels paràmetres que millor aproximen les dades.

```
x1 = np.array([1, 2, 3, 4])
y1 = np.array([5.1, 7.8, 11.0, 13.9])
```