Numpy & Matplotlib

Nicolas Jouvin

2022

This exercise sheet gets you accointed with NumPy and Matplotlib modules.

1 NumPy

Lecture notes

First, take a look at the NumPy warmup notebook on the website.

From the notebook, we now that NumPy is an implementation of arrays in Python (vector, matrices, tensor). It comes with many built-in methods functions for the `np.array` type. It also allows for random number generation via the `np.random` submodule and linear algebra via the cousin `scipy` module.

```
import numpy as np

x = np.array([1,2,3]) # vecteur 1D
A = np.array([[1,2,3],
              [4,5,6]]
             ) # matrice
```

1.1 Some basics

Question 1 (Operations)

Compute

1. the sum of all elements of `x` and `A` using the dedicated method.
2. the row-wise and column-wise sum of elements of `A`.
3. the product Ax. What is the expected shape ?

Solution

```
x.sum()
A.sum(axis=0)
A.sum(axis=1)
y = A @ x
y.shape == (2,)
```

Question 2 (Array creation)

Create an array with

1. integers from 0 to 9

2. a linearly spaced sequence of the following 13 floats between 0 and 2π : $\{0, \pi/6, \pi/3, ..., 2\pi\} = \{k\pi/6,\ k = 0, \ldots, 12\}$
3. A 5×5 diagonal matrix with only 23 on its diagonal.

Solution

```
a = np.arange(10)
b = np.linspace(0, 2 * np.pi, num=13)
B = 23 * np.eye(5)
```

Question 3 (Indexing & slicing)

Create

```
X = np.arange(100).reshape(50, 2)
```

Use slicing `:` to get

1. the whole second column of `X` & the whole 5th of `X`
2. The first 7 rows
3. The rows from 3rd to 7th (included)
4. The last 5 rows
5. One every 3 rows of `X` starting from the second one

Use boolean indexing to

1. get the elements of `X` greater or equal to 30.
2. get the elements of `X` in $[25, 45[$
3. fill `C = np.zeros((5, 5))` of 1 on every two rows and one every two columns (une colonne sur deux et une ligne sur deux).

```
[[1. 0. 1. 0. 1.]
[0. 0. 0. 0. 0.]
[1. 0. 1. 0. 1.]
[0. 0. 0. 0. 0.]
[1. 0. 1. 0. 1.]]
```

Solution

```
X = np.arange(100).reshape(50, 2)
X[:, 1]  # 2nd column
X[4, :]  # 5th row
X[:7, :]  # first 7 rows
X[2:7, :]  # rows from 3rd to 7th
X[45:, :]  # last 5 rows
X[1::3, :]  # One every 3 rows starting from the second one

X[np.where(X >= 30)]
X[np.where((X >= 25) & (X < 45))]

C = np.zeros((5, 5))
```

```
C[::2, ::2] = 1
print(C)
```

Question 4 (Random number generation)
Create a NUmPy random number generator with seed set to 42. Then, draw

1. 100 samples from a $\mathcal{N}(-3, 2)$
2. 50 samples uniformly from the list `['a', 'b', 'c']`
3. a 85×5 matrix of samples from $\mathcal{U}([-3.5, 6e^2])$
4. a 42×3 matrix of samples from a Poisson $\mathcal{P}(3)$

Solution

```
rng = np.random.default_rng(seed=42)
x_norm = rng.normal(loc=3, scale=2, size=(100,))
x_dunif = rng.choice(["a", "b", "c"], size=(50,))
x_cunif = rng.uniform(low=-3.5, high=6 * np.exp(2), size=(85, 5))
x_poiss = rng.poisson(lam=3, size=(42, 3))
```

Question 5 (Universal function (ufunc))
Compute

1. $\sin(x_i)$ for $x \in \{0, \pi/6, \pi/3, ..., 2\pi\}$
2. $\exp(-x^2)$ for $x \in \{-4, -3, \ldots, 3, 4\}$
3. $c_i = \tan(a_i) \times b_i - a_i^{b_i}$ with `a = rng.random(100)` and `b = rng.random(100)`

Solution

```
b = np.linspace(0, 2 * np.pi, num=13)
np.sin(b)
c = np.arange(-4, 5, 1)
np.exp(-(c**2))
a = rng.random(100)
b = rng.random(100)
c = np.tan(a) * b - a ** b
```

> **Lecture notes**
>
> NumPy has built-in function and methods that operates on arrays. For example, the `np.min()` and `np.max()` function return the maximum element of an array. In addition, the `np.argmin()` and `np.argmax()` return the indice(s) of the minimum (resp. maximum) element of the array.
>
> They have an argument axis that allow to specify the axis among which the min (resp. max) should be computed.

Question 6 (Advanced operations)
Let

```
a = rng.random(size=(100,))
target = 0.23
```

1. find the maximum element of `a`
2. find the closest value to `target` in array `a`. What if `a` is 2-D ? *e.g.* `a = rng.random(size=(100,10))`. **Hint:** look at the `np.unravel_index` function documentation.
3. when `a` is 2-D, find the maximum for each column and the maximum for each row.
4. Read the documentation of `np.unique()` and its arguments. Then, count the number of unique elements in `x = rng.choice(np.arange(10), size = 100)`.

Solution

```
a.max()

closest_idx = np.argmin(np.abs(a - target))
a[closest_idx]


a = rng.random(size=(100, 10))
closest_idx = np.argmin(np.abs(a - target))
# error : a[closest_idx]
closest_idx = np.unravel_index(closest_idx, a.shape)  # convert to tuple of ind
a[closest_idx]

a.max(axis=0)  # max / column
a.max(axis=1)  # max / row

help(np.unique)
x = rng.choice(np.arange(10), size = 100)
val, counts = np.unique(x, return_counts=True)
```

Lecture notes

The SciPy module is built for NumPy arrays. It is organized in many submodules for signal processing, statistics, etc. Its `scipy.linalg` submodule provides advanced linear algebra tools such as norm computations (possibly with `axis` specification) and eigen decomposition.

```
import scipy.linalg as spla
help(spla.norm)
help(spla.eig)
help(spla.inv)
```

Question 7 (Linear algebra)

Let `X = rng.random(shape=(20, 5))`

1. compute the two symmetric matrices $A = XX^\top$ and $B = X^\top X$
2. check that the euclidean norms of A and B (seen a vectors in $\mathbb{R}^{100}$) are equal
3. compute the row-wise l_1 norm of A
4. get P and D from the eigen decomposition of $A = PDP^{-1}$
5. compute P^{-1}

6. use `np.allclose` to check if PDP^{-1} is equal (up to a 10^{-7} absolute tolerance) to A
7. use `np.isclose` to extract the non-zero eigenvalues of A.
8. check that the non-zero eigenvalues of A and B are the same
9. check that the sum of eigenvalues of A (or B) is (up to a 10^{-7} absolute tolerance) equal to the sum of the diagonal element of A (or B).

Solution

```
X = rng.random(size=(20, 5))
A = X @ X.T
B = X.T @ X
spla.norm(A, ord=2) == spla.norm(B, ord=2)
spla.norm(A, ord=1, axis=0)
eigval, P = spla.eig(A)  # eig() return a tuple of length 2 with (D, P)
D = np.diag(eigval)
P_inv = spla.inv(P)
atol = 1e-7
if not np.allclose(A, P @ D @ P_inv, atol=atol):
    print("There's in eigen decomposition !")

eigval[~np.isclose(eigval, 0)]  # non zero eigenvalues of A
eigvalB, _ = spla.eig(B)
if not np.allclose(
    np.sort(eigval[~np.isclose(eigval, 0)]),
    np.sort(eigvalB[~np.isclose(eigvalB, 0)]),
):
    print("The non-zeros eigenvalues should be the same for A and B !")

eigval.real.sum() - np.diag(A).sum() < atol
```

Lecture notes

NumPy uses array broadcasting to simplify operation on arrays like $X + 1$ or $x * y$. In order to work, shapes must match and the operation is done element by element. Otherwise, NumPy will automatically try to detect in which way the operation can be done by extending the shape of the smaller array. We say that the smaller array is broadcast across the bigger array.

For example the following code tries to normalize the row of X by its rowsum. **It will return a broadcasting error.**

```
X = rng.random(size=(20, 5)
X = X / X.sum(axis=1) # return a broadcasting error
```

Indeed, the shape `(20, 5)` and shape `(20,)` are not compatible. The shape must be `(20, 1)` to be extended. In this example it can be achieved vie the `keepdims` arguments or the `np.newaxis` trick to created a new "dimension" for our 1D array

```
X = rng.random(size=(20, 5)
```

```
X = X / X.sum(axis=1, keepdims=True) # the column dimension is preserved with keepdims
X = X / X.sum(axis=1)[:, np.newaxis] # equivalent
```

Question 8 (Broadcasting)

Let X be as before

```
X = rng.random(size=(20, 5))
```

Use array broadcasting to

1. Normalize the rows of X in the sense of euclidean norm. Sanity check : $diag(XX^\top) = I_{20}$ up to an small absolute tolerance.
2. Normalize the columns of X in the sense of euclidean norm. Sanity check : $diag(X^\top X) = I_5$ up to an small absolute tolerance.
3. multiply column j by $j+1$, *i.e.* column 1 by 2, column 2 by 3, etc.
4. multiply row i by $i+2$, *i.e.* row 1 by 3, row 2 by 4, etc.

Solution

```
X = rng.random(size=(20, 5))

X_rownorm = X / spla.norm(
X, ord=2, axis=1, keepdims=True
)  # normalize rows of X
np.allclose(np.diag(X_rownorm @ X_rownorm.T), np.ones(X.shape[0]))


X_colnorm = X / spla.norm(
X, ord=2, axis=0, keepdims=True
)  # normalize cols of X
np.allclose(np.diag(X_colnorm.T @ X_colnorm), np.ones(X.shape[1]))

X * np.arange(2, X.shape[1]+2)[np.newaxis, :] # column j * (j+1), j=1, ..., 5
X * np.arange(3, X.shape[0]+3)[:, np.newaxis] # row i * (i+2), i=1, ..., 20
```

2 Matplotlib

Lecture notes

The `matplotlib.pyplot` module provides basic plotting functionalities to draw points, lines with titles.

The basic object is a Figure. We open a new Figure with `plt.figure()` where we can provide argument such as the size, etc.

```
import matplotlib.pyplot as plt
plt.figure()
plt.plot([1, 3, 4, 6], [2, 3, 5, 1])
plt.scatter([1, 3, 4, 6], np.array([2, 3, 5, 1]) + 1)
plt.show()
```

When a figure is open, matplotlib will draw on it by default. There are several plotting functions such as

- `plt.plot()` : line plot (draw a line between points)

- `plt.scatter()` : scatter plot (draw only the points)

Important: However, in order to show the figure in the screen we need to call `plt.show()`.

Note: `plt.show()` closes the current figure, the latter being "lost". However, figures may be saved with `plt.savefig()`. It is important to keep them when running script with `python my_script.py` in bash or anaconda prompt, otherwise they will be lost.

```
plt.figure()
plt.plot([1, 3, 4, 6], [2, 3, 5, 1])
plt.scatter([1, 3, 4, 6], np.array([2, 3, 5, 1]) + 1)
plt.savefig('my-first-plot.pdf')
```

Question 1 (Basic plotting)
Plot $\sin(t)$ for $t \in [-2\pi, 2\pi]$ (discretized in 100 linearly spaced points). Add label "t" and "sin(t)" for x-axis and y-axis. Finally add the title "Sinus function". Save the plot to a file named 'sin-plot.pdf'. Your plot should look like Figure 1.

Solution

```
t = np.linspace(-2* np.pi, 2 * np.pi, 100)   # 100 points between
plt.figure()
plt.plot(t, np.sin(t))
plt.xlabel('t')
plt.ylabel('sin(t)')
plt.title('Sinus function')
plt.savefig("sin-plot.pdf")
```

Question 2 (Setting x and y limits)
Change the x and y axis limits to $[-8, 10]$ and $[-0.9, 2]$ respectively.

Solution

```
plt.figure()
plt.plot(t, np.sin(t))
plt.xlabel("t")
plt.ylabel("sin(t)")
plt.title("Sinus function")
plt.xlim(-8, 10)
plt.ylim(-0.9, 2)
plt.show()
```

Question 3 (Playing with graphical arguments)
Look at the `plt.plot` documentation or this webpage and try changing the plot to have

1. a red, dashed line
2. a green, dash-dot line with width 4

Solution

```
# red. dashed line
plt.figure()

plt.plot(t, np.sin(t), linestyle="--", color="red")

plt.show()



# green,  dashed-dot line

plt.figure()

plt.plot(t, np.sin(t), linestyle="-.", color="green", linewidth=4)

plt.show()
```

Question 4 (Adding another plot)

On the same plot, add $\cos(t)$ as a orange scatter plot with `x`-shaped point markers of size 12.

Solution

```
# Add cos scatter plot
plt.figure()
plt.plot(t, np.sin(t))
plt.scatter(t, np.cos(t), color="orange", marker="x", s=12)
plt.show()
```

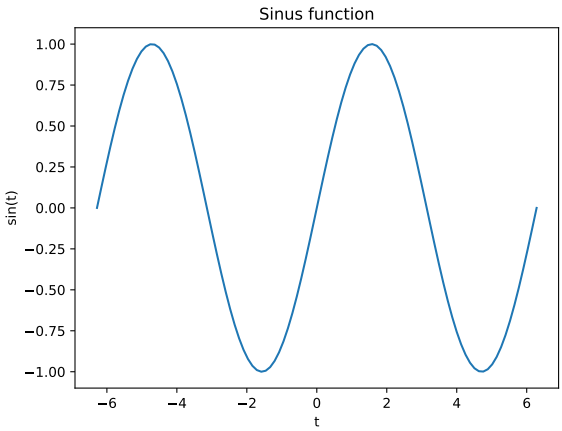

Figure 1: Question 1