PythonNotebook5_solution_2023

December 13, 2023

[1]:

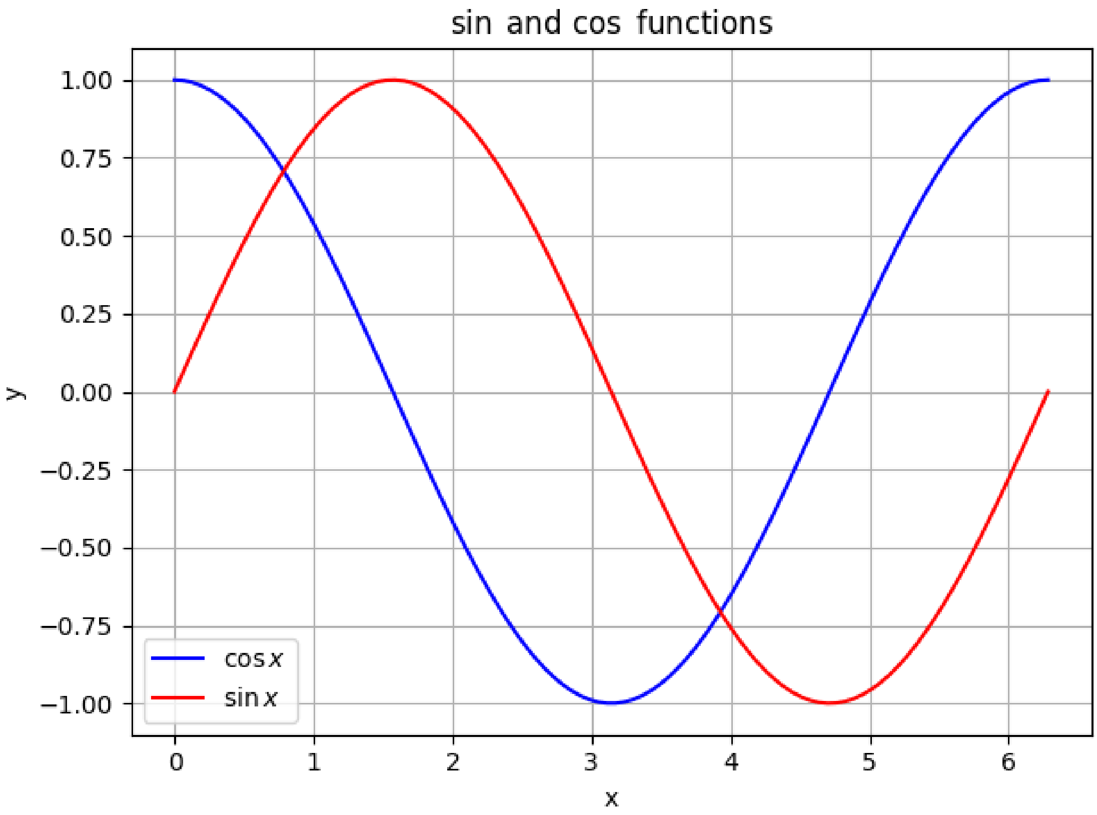
```
import numpy as np
import matplotlib.pyplot as plt
```

0.1 Exercise 5.2.1 Function plots

Copy the code above, modify it to add a sine function to the plot. Give the two curves distinct colors (choose colors you like) and add a legend showing which of the curves is which.

[2]:

```
x = np.linspace(0, 2*np.pi, 100)
y1 = np.cos(x)
y2 = np.sin(x)

plt.plot(x, y1, label="$\cos x$", c='b')
plt.plot(x, y2, label="$\sin x$", c='r')
plt.legend()
plt.title("$\sin$ and $\cos$ functions")
plt.ylabel("y")
plt.xlabel("x")
plt.grid()
plt.tight_layout()
plt.show()
```



0.2 Exercise 5.2.2 Bar graph revisited

In the last notebook you made a bar graph of monthly rainfall amounts. Perhaps you wondered if it's possible to place the bars side by side so they don't overlap.

You can do this by shifting the x-coordinates of your bars, and setting the bar width using the `width=` argument to the bar graph function.

- copy your plotting code from the last notebook.
- make a numpy array of the month list.
- now you can easily add or subtract a number from the whole array.
- make a plot where the bars for the different regions don't overlap.

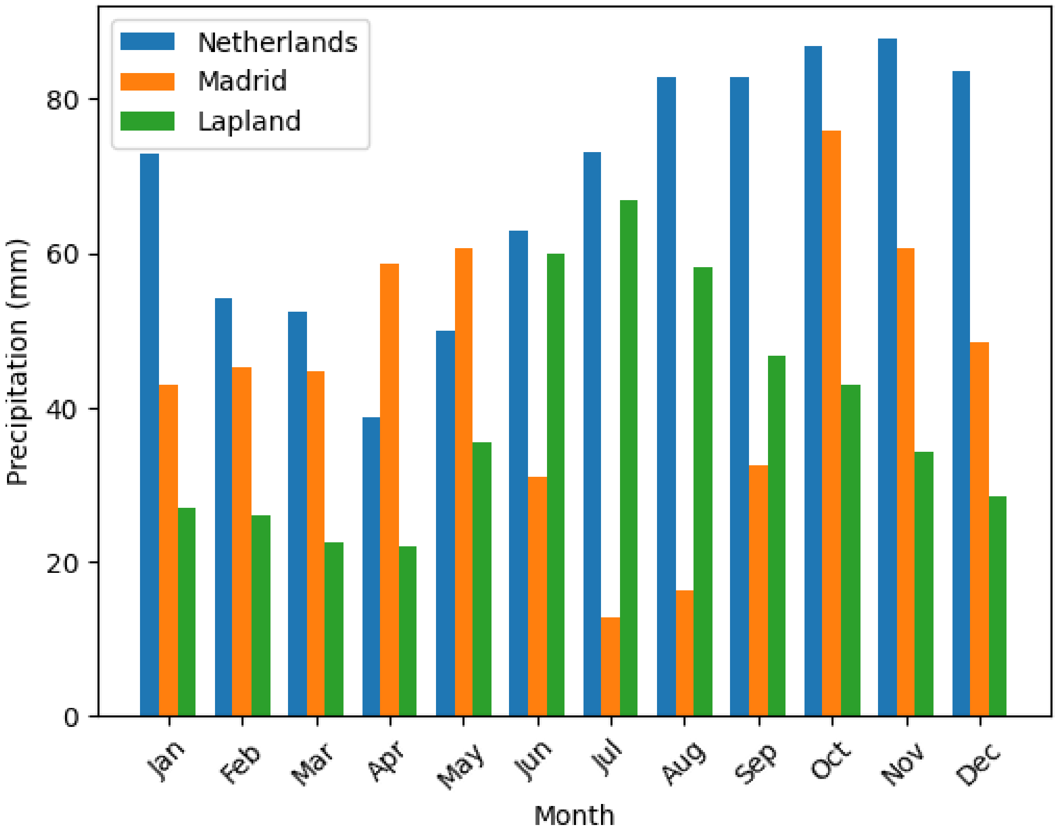
[3]:

```
# just run this cell to define the data

# Monthly mean precipitation (mm)
# Data from https://climatecharts.net/

month = np.arange(1,13)
Pr_NL      = [72.8, 54.1, 52.5, 38.7, 50.0, 63.0, 73.1, 82.9, 82.9, 86.8, 87.7,␣
 ↪83.6] # Netherlands
```

```
Pr_Madrid  = [43.0, 45.1, 44.8, 58.6, 60.7, 31.0, 12.7, 16.3, 32.5, 75.8, 60.7,␣
 ↪48.5] # Spain, around Madrid
Pr_Lapland = [27.0, 25.9, 22.5, 22.1, 35.6, 60.0, 66.9, 58.1, 46.6, 42.9, 34.2,␣
 ↪28.4] # Finland, Lapland, around Sodankylä
#             Jan   Feb   Mar   Apr   May   Jun   Jul   Aug   Sep   Oct   Nov  ␣
 ↪Dec
```

[4]:

```
# your plotting here
from copy import deepcopy

Pr_NL = np.array(Pr_NL)
Pr_Madrid = np.array(Pr_Madrid)
Pr_Lapland = np.array(Pr_Lapland)

# Width of a bar
width = 0.25

month_NL = month - width
month_Madrid = deepcopy(month) # <= learn more about copy and deepcopy here:␣
 ↪https://docs.python.org/3/library/copy.html
month_Lapland = month + width

plt.bar(month_NL, Pr_NL, width=width, label='Netherlands')
plt.bar(month_Madrid, Pr_Madrid, width=width, label='Madrid')
plt.bar(month_Lapland, Pr_Lapland, width=width, label='Lapland')

plt.xlabel('Month')
plt.ylabel('Precipitation (mm)')

# Month names and setting x-ticks
month_names = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep',␣
 ↪'Oct', 'Nov', 'Dec']
plt.xticks(month, month_names, rotation=45)  # Rotate labels by 45 degrees

plt.legend()
plt.show()
```



0.3 Exercise 5.3.1

- Create a 4x2 matrix `m` filled with different values

4x2 means four rows and two columns. You can use `np.array()`, and pass it nested lists, or you can use `np.zeros((rows, columns))` and fill in the values afterwards. Here `rows` and `columns` are the number of rows and columns you want. Note the double parentheses. The reason is that `(rows, columns)` is a tuple, and the `np.zeros()` function takes this tuple as it's argument.

- Check the shape of your matrix, using `m.shape`
- Try the transpose() function on your array (hint: `m.transpose()`). What did it do?

[5]:

```
### OPTION 1: Filling with random numbers using numpy functions; note that here and on Option 2 we round the random numbers to 2 decimals.

n_rows, n_columns = 4, 2

m = np.round(np.random.random((n_rows, n_columns)), 2)

print("Matrix m and its transpose (generated using numpy functions)")
print(m)
```

```python
print("Transpose:")
print(m.transpose())
print("=======================================")

### OPTION 2: Filling with random numbers using a for loop (note that using numpy is always faster than using for loops)

from random import random

m = np.zeros((n_rows, n_columns))

for i in range(m.shape[0]):
    for j in range(m.shape[1]):
        m[i,j] = round(random(), 2)

print("Matrix m and its transpose (generated using for loops)")
print(m)
print("Transpose:")
print(m.transpose())
print("=======================================")

### OPTION 3: Filling with fixed numbers

m = np.array([[0, 1],
              [2, 3],
              [4, 5],
              [6, 7]])

print("Matrix m and its transpose (generated using fixed numbers)")
print(m)
print("Transpose:")
print(m.transpose())
print("=======================================")
```

```
Matrix m and its transpose (generated using numpy functions)
[[0.91 0.97]
 [0.58 0.38]
 [0.82 0.66]
 [0.79 0.89]]
Transpose:
[[0.91 0.58 0.82 0.79]
 [0.97 0.38 0.66 0.89]]
=======================================
Matrix m and its transpose (generated using for loops)
[[0.6  0.52]
 [0.36 0.99]
 [0.42 0.97]
 [0.73 0.92]]
```

```
Transpose:
[[0.6  0.36 0.42 0.73]
 [0.52 0.99 0.97 0.92]]
========================================
Matrix m and its transpose (generated using fixed numbers)
[[0 1]
 [2 3]
 [4 5]
 [6 7]]
Transpose:
[[0 2 4 6]
 [1 3 5 7]]
========================================
```

0.4 Exercise 5.3.2 Slicing matrices

As you have seen for Python lists and strings, numpy arrays can also be sliced using `[start:end:step]`.

- Use slicing to cut out the part

```
11, 12
15, 16
```

from matrix A defined below.

```
[6]: # Just run this cell to define A
     A = np.array([[ 1,  2,  3,  4],
                   [ 5,  6,  7,  8],
                   [ 9, 10, 11, 12],
                   [13, 14, 15, 16],
                   [17, 18, 19, 20]])
```

```
[7]: A_sliced = A[2:4, 2:4]
     print(A_sliced)
```

```
[[11 12]
 [15 16]]
```