Data Science Survival Skills

Homework 11

Homework 11:

In this homework, we will use Numba and Cython to make our code faster.

- **Numba** is a just-in-time compiler that generates fast machine code from our existing Python code.

- **Cython** allows us to call C functions and declare C types on variables and class attributes, making high-performance code (GitHub).

Homework 11: Task 1/2

- We provide you with the following code:

```
def movingAverage(y, window=7):

    result = np.zeros_like(y)

    for i in range(window, y.size):
        result[i] = y[i-window:i].mean()
    return result

x = np.linspace(0, 20, 10000)
y = np.sin(x)+np.random.randn(x.size)/4
```

Use the **nopython** mode of numba to run **movingAverage()** in machine code.

- Calculate the moving average of y. Therefore use the previous function with **and** without the **nopython** mode. Compare the speed of both results using *%timeit*.

➜ **Slide: Screenshot of your results (time spent using each method)**

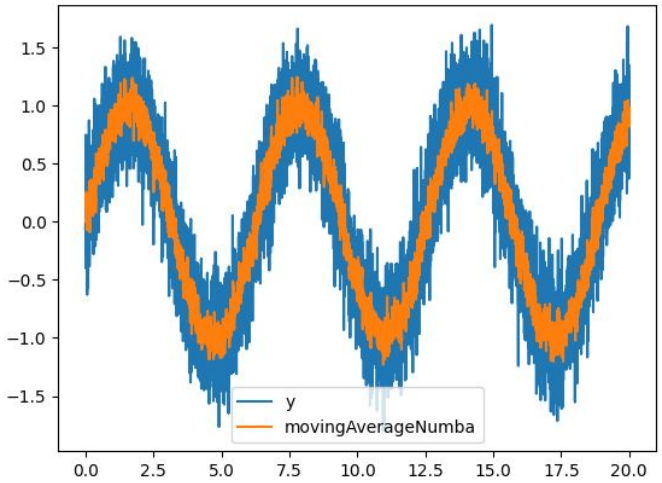
➜ **Slide Question A: What is the difference between the decorators njit and jit?**

Homework 11: Task 1/2

➜ **Slide Question B:** **The first time using the function in machine code takes longer than the original function. Why?**

- Use the *%%timeit* and analyze the difference.

➜ **Slide:** **plot your results.**



Homework 11: Task 2/2

- We provide you with the following functions:

```
def py_dot(v1, v2):
   return sum(x*y for x, y in zip(v1, v2))

def np_dot(v1, v2):
   #TODO
```

- Use the **cythonmagic** command interface to interactively work with Cython.

```
%load_ext Cython
```

```
%%cython

def fast_dot(v1, v2):
   cdef double result = 0.0
   # TODO
   return result
```

- Complete the function **np_dot()** that returns the dot product of two arrays without using for loops.
- Complete the function **fast_dot()** using C code. Here, you are allowed to use a for loop.

Homework 11: Task 2/2

- Test your functions with the following code:

```
n = 1000

print("With numpy arrays: \n")
v1 = np.random.uniform(size=n)
v2 = np.random.uniform(size=n)

%timeit py_dot(v1, v2)
%timeit np_dot(v1, v2)
%timeit fast_dot(v1, v2)

print("\n With Python lists: \n")
v1 = list(v1)
v2 = list(v2)

%timeit py_dot(v1, v2)
%timeit np_dot(v1, v2)
%timeit fast_dot(v1, v2)
```

Note: If it doesn't work with Python lists, please try the array library that was also used in the exercise.

➜ **Slide:** **Screenshot of your code and time results.**

Homework 11: Task 2/2

Run your previous defined functions with different values for n (e.g. [1_000, 1_000_000, 10_000_000]). Interpret the results you observe!

➜ **Slide Question C: Which one of the previous functions performs the fastest calculation using numpy arrays and which one using Python lists? Why?**

➜ **Slide Question D: How does the value for n affect the measurement results? Explain why the behavior is like this.**

Homework 11: Example

Tasks 1 and 2:

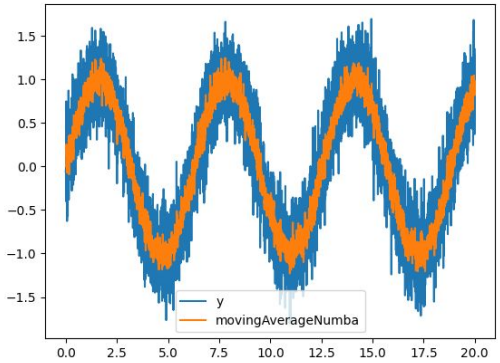
```
## Task 1
YOUR CODE


%time movingAverageNumba(y) #First shot
CPU times: user 884 ms, sys: 200 ms, total: 1.08 s
Wall time: 4.3 s

%timeit movingAverageNumba(y) #Timeit

235 µs ± 1.79 µs per loop (mean ± std. dev. of 7 runs, 1,000 loops each)
```

```
## Task 2
YOUR Functions
```



- Answer to question A: A very good answer
- Answer to question B: ...
- Answer to question C:...
- Answer to question D:...

Homework: Requirements

You must complete **all** homework assignments (**unless otherwise specified**) following these guidelines:

- **One** slide/page.
- **PDF** file format only.
- It has to contain your **name** and **student** (**matriculation**) **number** in the down-left corner.
- Font: **Arial,** Font-size: **> 10 Pt.**
- Answer **all** the questions and solve all the tasks requested.
- Be careful with **plagiarism**. Repeated solutions will not be accepted!