Problem1

February 24, 2021

0.1 Jeremy Kazimer

5018-1732

Assignment #1

For the first assignment of this class, I would like to replicate Assignment #5, Problem #1 of PHY 410. The question supposes that we have code to generate a vector of size $N = 1,000,000$ of five numbers $\vec{x}_i$ such that they're randomly-distributed by some mean and standard deviation. Then, the goal is to calculate the weighted average for each of the five numbers with non-normalized weights $\vec{W} = (5, 15, 30, 100, 400)$.

The catch is that we're not supposed to use loops. On a normal computer, this is easy. However, on a Raspberry Pi Zero W it's a bit different. Namely, the processing power is severly reduced. For example, my current laptop has 8 GB of RAM with a 3-core 1.2 GHz processor. The Raspberry Pi Zero W, on the other hand, has 512 MB of RAM with a 1-core 1 GHz processor. Thus, the problem must be modified to meets the demands of this device.

For example, it is not feasible to import the entirety of `numpy`. Instead, we'll have to important only the sub-libraries that we need:

[1]:
```
from numpy import random
from numpy import array
```

We can, however, keep $N = 1,000,000$, but the runtime will be atrocious in producing the array (but also operations on said array) due to technical limitations. Therefore, it would be sensical to decrease $N = 1,000,000 \rightarrow N = 100,000$. Consider that size$(x) = 100,000 \times 5$ so it's not quite 1-dimensional.

[2]:
```
# Vector of random x's
x = random.normal(size = (100000, 5),
                  loc = [20, 30, 20, 50, 30],
                  scale = [5, 5, 5, 5, 5])
```

Now, the weights:

[3]:
```
# Unnormalized weights
W = array([5, 15, 30, 100, 400])
```

```
# Normalized weights
nW = W/W.sum()
```

Then, we can calculate the weighted average via the formula

$$\bar{x} = \frac{1}{N}\sum_{i=0}^{N-1} \vec{x}_i \cdot \vec{W}_i$$

which translates roughly to this code. I opt to use my own definition (in comparison to the original assignment) because it saves on importing functions from `numpy`.

[4]:
```
# Dot product
dp = x*nW
```

[5]:
```
# Weighted average
wa = dp.sum()/x.shape[0]
```

[9]:
```
print('The weighted average: {:.4f}'.format(wa))
```

```
The weighted average: 33.0084
```

And something simple: we can calculate the average runtime using the `timeit` command. The best way to do this is by making a function that takes in no inputs, but does exactly what we did above for random inputs each time. This is because calculating for the same input n number of times should produce basically the same runtime, with nearly zero standard deviation. As such, it's necessary to vary the conditions such that a bigger picture can be seen:

[8]:
```
def random_average():
    # We declared this earlier, so it doesn't really matter all that much if␣
 ↪it's global.
    # This just saves me on re-declaring it since it already exists in the␣
 ↪global scope.
    # Python would just default to the global nW without declaring it, anyway.
    global nW

    x = random.normal(size = (100000, 5),
                      loc = [20, 30, 20, 50, 30],
                      scale = [5, 5, 5, 5, 5])

    dp = x*nW
    wa = dp.sum()/x.shape[0]
```

This doesn't need to return anything because the return statement is $\mathcal{O}(1)$ since it's non-recursive. The contribution to runtime therefore comes most from vector operators and declaring the original `x` array. Furthermore, since we are not using this outside the scope of this assignment, it is fine to just leave it like that. Otherwise, of course, we would want initial parameters as well as a return statement. Regardless, here is the time it takes on the Raspberry Pi Zero W:

[12]:
```
# -r: number of runs, as in how many times to compute random_average() for␣
 ↪different initial conditions
# -n: number of loops, as in how many times to compute random_average() for the␣
 ↪same initial conditions
# -r introduces variance to the system, whereas -n reduces variance in each␣
 ↪entry in that system

%timeit -r 10 -n 10 random_average()
```

```
549 ms ± 18.4 ms per loop (mean ± std. dev. of 10 runs, 10 loops each)
```

Really, it's not that bad - if this were consuming resources on my laptop I would be frustrated. However, since this is ran remotely and at most is using my laptop as a mobile hotspot, I'm not all that concerned. I can easily leave this to run for some larger project and checking its internal temperature sporadically.

Ultimately this assignment reproduction is rather simple, I think it captures in essence the issues I will be facing with the Raspberry Pi Zero W: computational power. Not only does it lack RAM, but it also does not have the processor for some of the assignments seen in the previous semester. As such, compromises will have to be made (importing only portions of a library or reducing the size of an assignment) in order for this to work. In a way, it's a self-imposed challenge, as any other Raspberry Pi model is definitely more powerful. Regardless, I think I captured the tone of the Raspberry Pi Zero W moving forward.
