Assignment 03 – Profiling and Numba

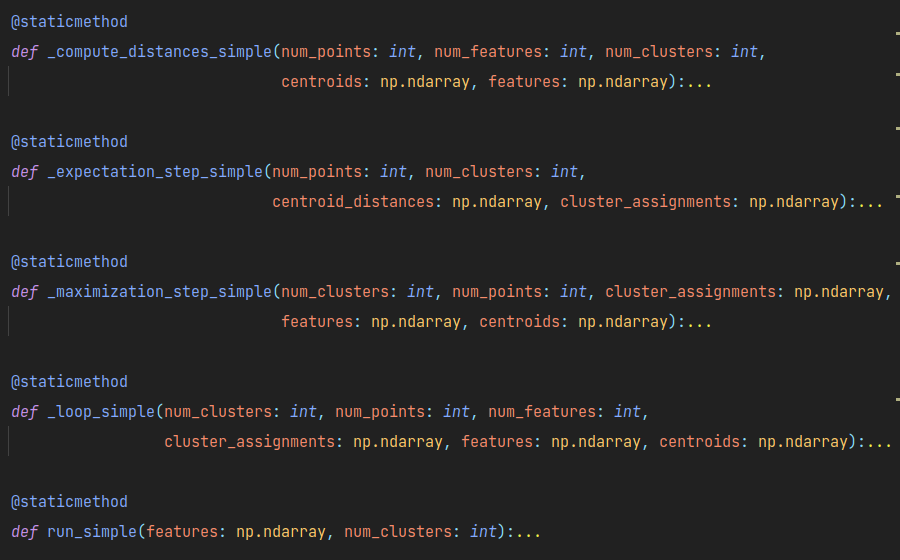
Konstantinos Georgiou

# General Info

* The code and the results for the assignment 3 is in this GitHub repo: <https://github.com/drkostas/DSE512-playground>
* The configuration I am using to run this assignment is this: <https://github.com/drkostas/DSE512-playground/blob/master/confs/assignment3_local_tcga.yml>
* Most of the code is in the *assignment3* folder: <https://github.com/drkostas/DSE512-playground/tree/master/assignment3>
  + There are 4 different *KMeans* implementations:
    - **simple**: Th non-vectorized *Kmeans* we created in class
    - **vectorized\_jacob**: The vectorized *Kmeans* we created in class
    - **vectorized**: My vectorized *Kmeans* version that I created in the previous assignment by improving Jacob’s vectorized.
    - **jitted vectorized\_jacob**: The jitted **vectorized\_jacob** implementation after modifying it a bit to make it able be jitted.
  + I didn’t jit my vectorized implementaiton because the use of **scipy.spatial.distance.cdist**, and **np.argmin** were messing with numba’s nopython mode.
  + **assignment3.py**: Loads the configuration, and runs the appropriate *KMeans* function for each subconfig (**simple**, **vectorized\_jacob**, **vectorized**) by calling either **kmeans.py** or **kmeans\_numba.py**
  + **kmeans.py**: It contains the KmeansRunner class which includes all the (non-numba) Kmeans implementations and the load\_dataset() function.
  + **kmeans\_numba.py**: Contains the **jitted vectorized\_jacob** implementation.
* I am also importing some other custom packages I’ve made, from which the most important ones are:
  + **profileit**: cProfile ContextManager-Decorator for profiling functions or code blocks - <https://github.com/drkostas/DSE512-playground/blob/master/playground/profiling_funcs/profileit.py>
  + **timeit**: ContextManager-Decorator for timing functions or code blocks - <https://github.com/drkostas/DSE512-playground/blob/master/playground/timing_tools/timeit.py>
* Profiling raw results & screenshots: <https://github.com/drkostas/DSE512-playground/tree/master/outputs/final/assignment3/profiling>
* Runtime results & Amdahl Plots: <https://github.com/drkostas/DSE512-playground/tree/master/outputs/final/assignment3/results>

# 1. Refactoring kmeans.py

In **kmeans.py**, there is a **run**() function which calls one of: **run\_simple()**, **run\_vectorized\_jacob(),** r**un\_vectorized()** inside a **profileit** *with* statement.

I refactored these 3 functions, to call a **\_loop()** sub-function which in turn calls the **\_compute\_distances()**, **\_expectation\_step()**, **\_maximization\_step()** functions to run each individual step of the algorithm. Each implementation has different functions for these steps, for example, **run\_vectorized\_jacob()** calls **\_loop\_vectorized\_jacob()** etc. If this is not clear enough, feel free to ask me and I can elaborate more. Example:

# 2. Profile kmeans.py

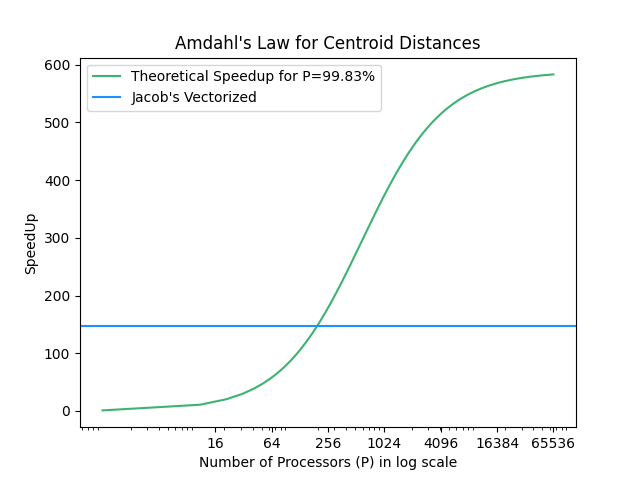
Inside **kmeans.py**, the **run()** functions, calls the appropriate implementation inside a **profileit** context manager. The times are the following:

| Algorithmic Step | Kmeans Simple | Jacob’s Kmeans Vectorized | My Kmeans Vectorized |
| --- | --- | --- | --- |
| **Compute Distances** | 627.4(s) – 99.83% | 3.27(s) – 76.16% | 1.336(s) – 52.22% |
| **Expectation** | 0.0311(s) – 0.00005% | 0.0324(s) – 0.75% | 0.0004(s) – 0.02% |
| **Maximaziation** | 1.02(s) - 0.16% | 0.9901(s) – 23.06% | 1.221(s) – 47.72% |
| **Total** | 628.5(s) – 100% | 4.294(s) – 100% | 2.559(s) – 100% |

*The speedup from my Kmeans implementation is not relevant because I attempted to improve all three functions.*

The total speedup of Jacob’s Kmeans Vectorized compared to Kmeans simple is:

Jac\_vec\_speedup = 628.5(s)/4.294(s) = 146.367

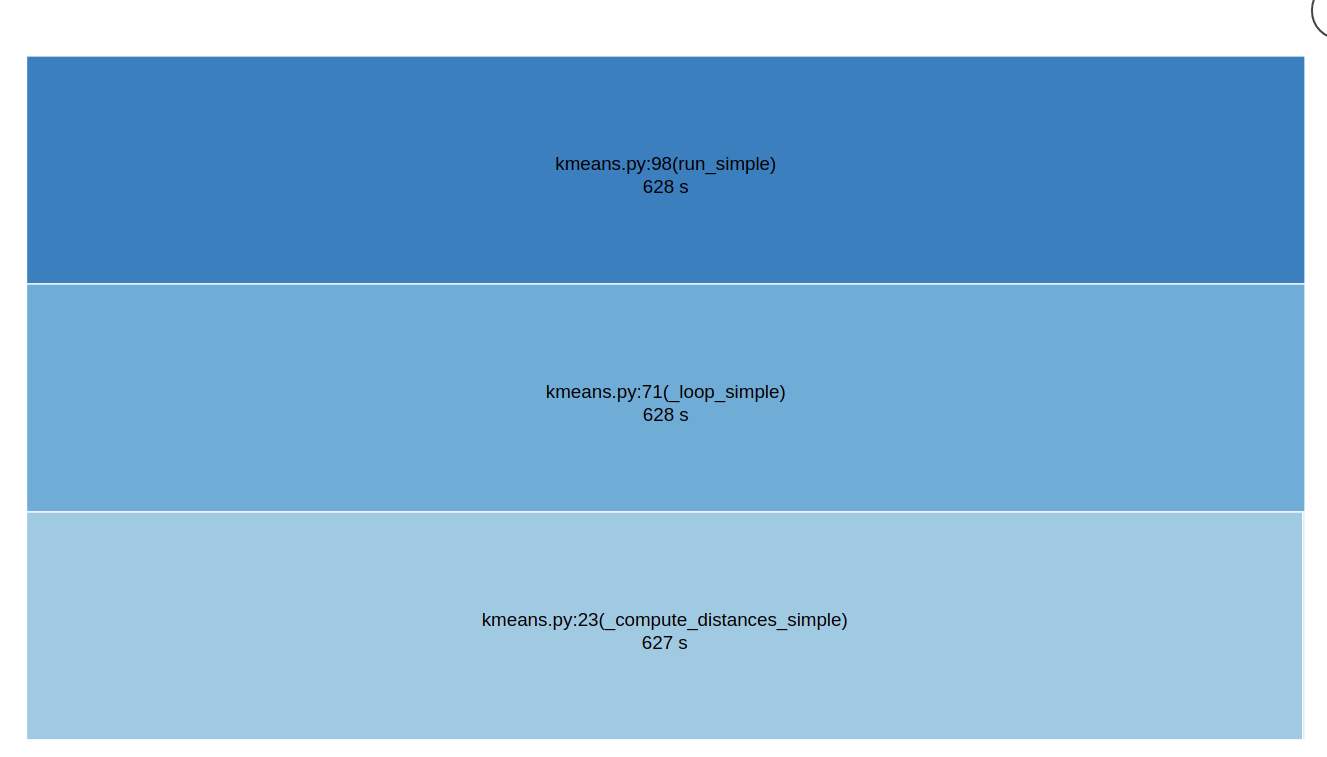
Plotting Amdahl’s Law with this speedup yields the following figure:

The maximum theoretical speedup that centroid distances can give if parallelized according to Amdahl’s law is **~583 times**. Jacob’s vectorized Kmeans achieved **~25.1%** of that maximum.

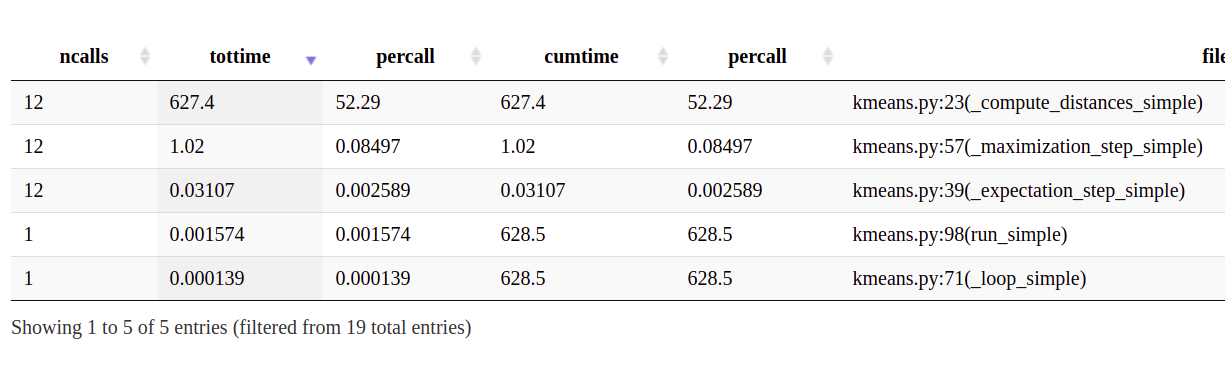
# 3. Visualize Icicle Plots

## Simple Kmeans

**Icicle Plot:**

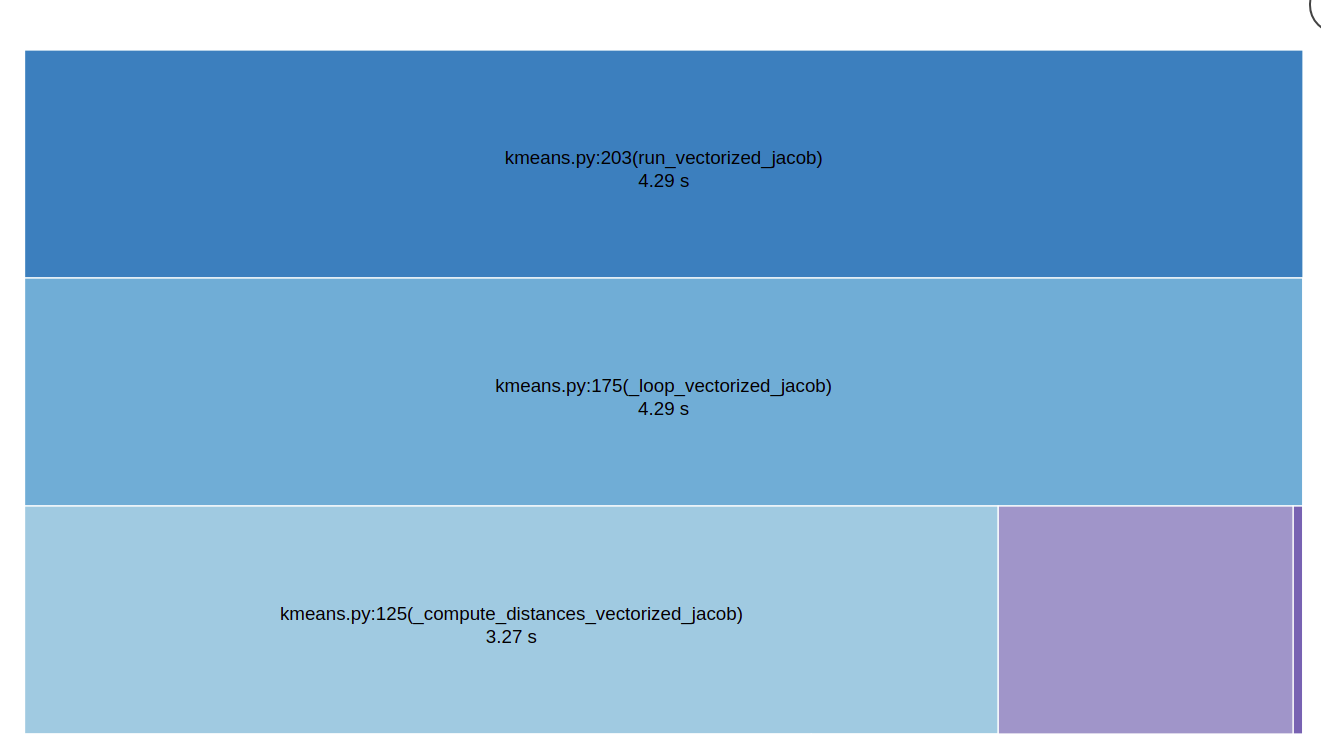


**Table with times and calls:**

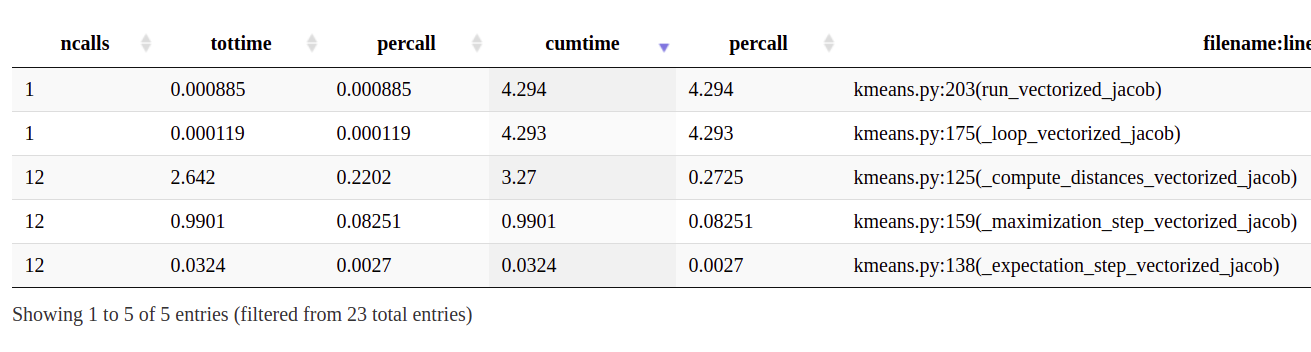


## Jacob’s Vectorized Kmeans

**Icicle Plot:**

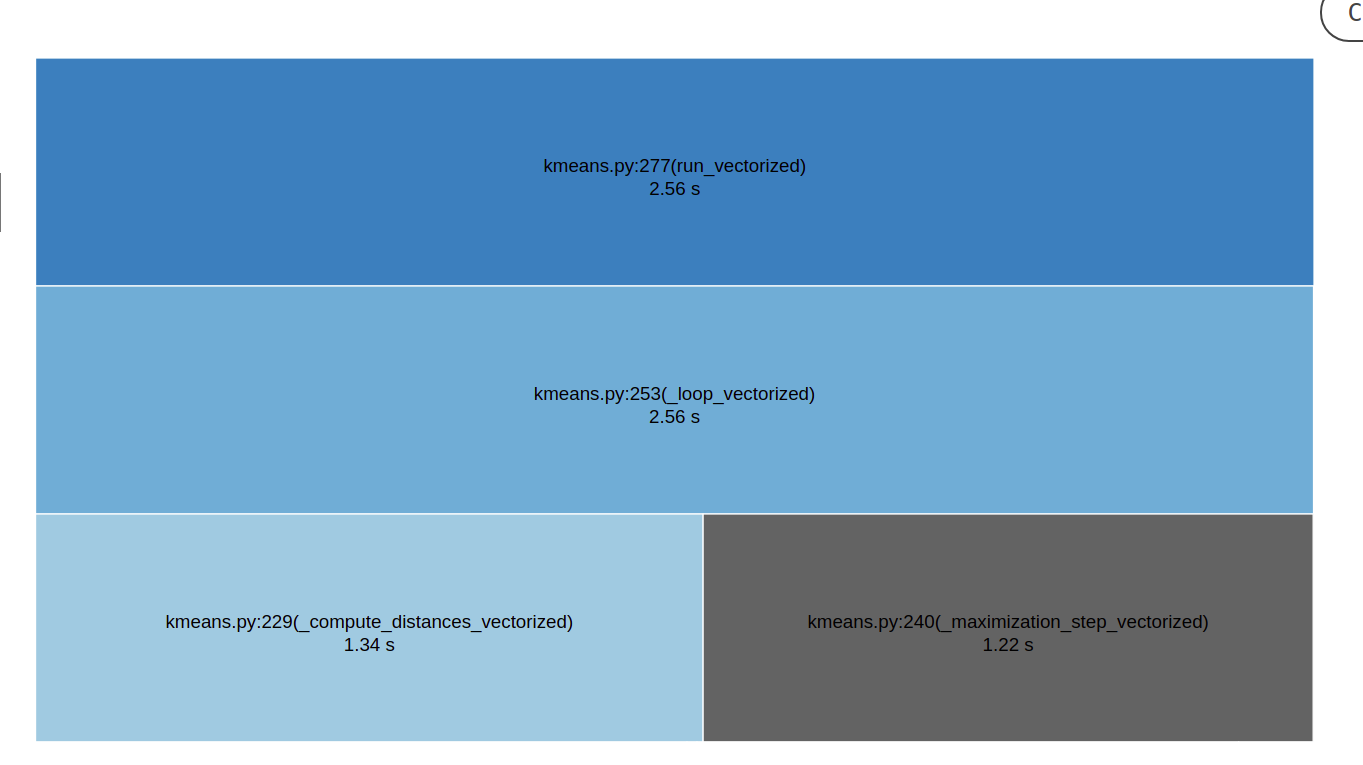


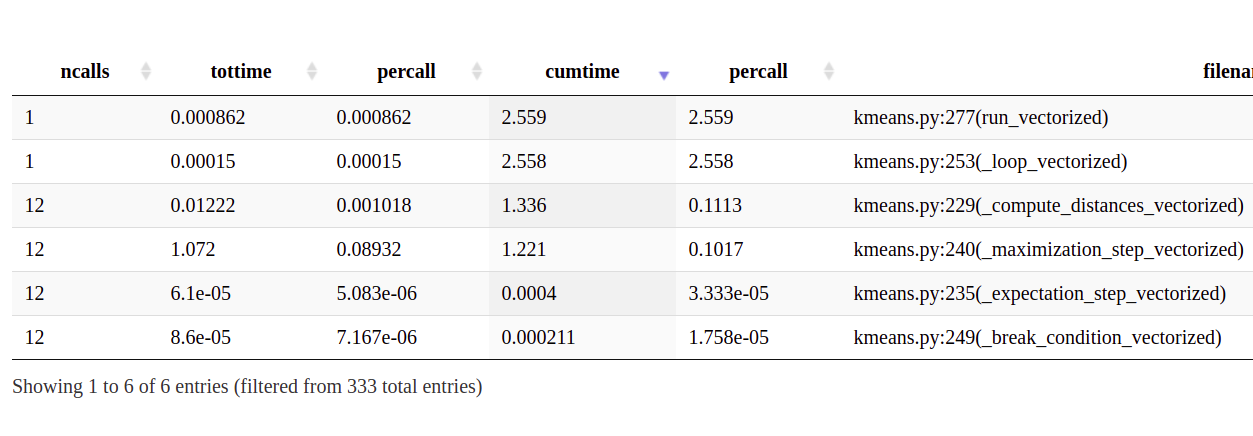
**Table with times and calls:**



## My Vectorized Kmeans

**Icicle Plot:**

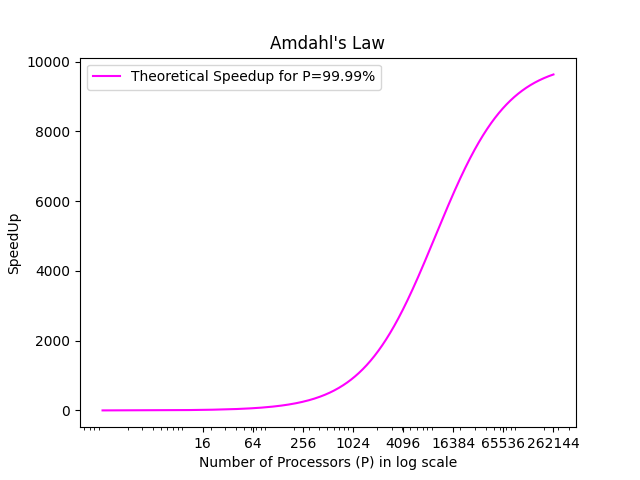
****

**Table with times and calls:**

# **4. Numba**

According to the table from Problem 2, the three functions (**compute\_distances**, **expectation\_step**, and **maximization\_step**) take up 99.83%+0.00005%+0.16% = 99.99% of the code.

For the maximum speedup let’s plot Amdahl’s Law again:

The maximum theoretical speedup that all three can give if parallelized according to Amdahl’s law is **~9632.57 times**.

Running **Jacob’s Vectorized Kmeans** using numba took this time:

| Algorithmic Step | Kmeans Simple | My Kmeans Vectorized | Jacob’s Vectorized Kmeans with Numba |
| --- | --- | --- | --- |
| **Compute Distances** | 627.4(s) – 99.83% | 1.336(s) – 52.22% | 3.3935(s) – 68.55% |
| **Expectation** | 0.0311(s) – 0.00005% | 0.0004(s) – 0.02% | 0.1570(s) – 3.17% |
| **Maximaziation** | 1.02(s) - 0.16% | 1.221(s) – 47.72% | 1.3980(s) – 28.24% |
| **Total** | 628.5(s) – 100% | 2.559(s) – 100% | 4.9504(s) – 100% |

**Jacob’s Vectorized Kmeans with Numba** speedup: 628.5/4.9504 = 126.95 → 1.3%

**My Vectorized Kmeans** speedup: 628.5/4.9504 = 245.6 → 2.5%

Plotting Amdahl’s Law along with the two speed ups:

