Results:

The runtime for the equivalence class based model was much faster than the full model. The number of iterations in the first part of the EM algorithm was much smaller for the equivalence based model since duplicate reads were grouped into classes. In addition to that, not having to calculate P3 values for each alignment made the runtime much faster. The downside to the equivalence class based model was that the results were less accurate compared to the full model. Although the full model was slower, it was a better representation of the actual data.

Design:

In my design I decided to break apart the different steps for the full EM model and the equivalence based model. Breaking up the steps in each algorithm allowed me to debug and find any unnecessary computation. I kept the computation methods as close to the class slides as possible. If I had to do the project from scratch, I would implement the equivalence based model first and then implement the full EM model. I would also implement the code in C or Java in order to run loops faster. Python was a bit sluggish at certain loop iterations and using Numba to compile some functions was a bit difficult to use with Numpy. This is because certain Numpy functions are not supported by Numba. Another point in my design that differed the full EM algorithm from the equivalence based model algorithm is that I had to user a larger epsilon value to check for convergence.

Roadblocks:

The first major roadblock I had was getting my mind around what it means to run the expectation part of the EM algorithm and what it means to run the maximization part of the EM algorithm when it relates to transcript quantification from RNA alignments. Once I was able to understand what each part of the EM algorithm did, I was able to implement, debug, and better understand how to improve my code. The second major roadblock I had was using Numba with Python in order to optimize and speed up the runtime of my full EM algorithm. Some Numpy functions are not supported by Numba, so I had to find ways to compute what I wanted without Numpy.

Graphed Results: