**Instructions for running Junction Analysis Codes**

1. Choose locus – remove the pound sign to “un-comment” each line and add pound signs in front of whichever locus definitions you wish to turn off
   1. OPTION 1 – select pre-entered locus and guide sequence (lines 36-49). I included two of our most common analysis sites (R26-MHD and R26-TINS)
   2. OPTION 2 – enter locus and guide sequences manually in either NGG PAM orientation (lines 51-56) or in CCN PAM orientation (lines 58-63). The locus sequence and raw reads must be in the same orientation. I have provided a separate small module titled “library\_revcom.py” that will convert all reads in a library to the reverse complement sequences if you need to do so to simplify analysis.
   3. OPTION 3 – Manual input of left and right flanks (lines 65-68). Again, the sequence orientation needs to match the orientation of the raw sequence reads
2. Generate csv with list of files you want to analyze in column 1, with no headers (example file “test.csv” provided). Place this .csv and all .csvs with sequence reads for analysis in the same folder as the script.
3. Make sure the code that reads sequences from the input files (lines 87-90) identifies your sequence reads in the correct column. Our output files from CLC Genomics Workbench place the sequence reads in column 3, which is reflected in the column names listed in line 89. If your output places reads in a different column of the .csv files, adjust line 89 accordingly.
4. Hit enter…and you’re done! When the program is complete, it will print:

fin fin

Process finished with exit code 0

If the code ever prints “Process finished with exit code 1”…something went wrong and it will give you an error code that should give you some idea of the issue. As long as you didn’t change any of the core code, the most common errors you might experience will likely be a failure to find the fastq files for analysis (they must be in the same folder as the code) or failure to write to the output file (if for some reason you have another file of the same name already open and you’re trying to perform reanalysis.

The program will print out a few lines per library as it is running that look like this (explanations in parentheses):

start file name

reads: ##### (reads in the complete library, unfiltered)

jxns with ambiguity: ## (jxns containing ambiguous base call, R, K, Y, S, W, etc)

failed matches: ## (reads that failed to match either right or left 10mer)

jxns with substitutions: ## (jxns with substitution in nts 3-10 on left/right, or adjacent to match in MH reads)

\*final junctions: ##### (jxns passing all filters: no jxns with ambiguous or substituted base calls)

\*These (final junctions) are the only junctions contributing to final analysis, junctions with failed flank matches, ambiguities, or substitutions are thrown out (into a file titled file\_name\_miscall\_analysis if you’re ever interested in looking at them. I promise they’re not very interesting).

Every junction that passes filters is classified with respect to left deletion, right deletion, microhomology size and sequence, insertion size and sequence, and direct or inverted TINS classification.

Once this is completed for every file in the list you provided, the code will compile a list of all the unique junctions across all the input files and then count frequency of each individual unique read in each library. This is the most taxing part of the code with respect to processing power and computer memory. If you don’t touch the computer during this time, it will run more efficiently.

The final output file will be a csv titled input file name &“\_full\_analysis\_freq” and will contain the compiled list of the unique junctions established in the last step, along with junction analysis and classifications and raw counts, raw frequencies, and repair-adjusted frequencies (normalized against the amount of germline sequence remaining) for each input library. Downstream analysis is best done in excel, so the code is essentially only optimized for junction analysis and frequency determinations, which tend to make Excel choke when libraries are of any substantial size…which they usually are.

Based on some limited testing, the entire code block should take under 1 minute per initial file if fewer than 10, between 1 and 2 minutes per file if 10-20 files are provided. This varies primarily on the length of the compiled unique junctions list because the bottleneck is certainly read counting (you can appreciate, for example, that a lot of computing power and memory is required to count the occurrences of 20,000 unique junctions across 4,000,000 library reads).