**HIVSeqinR Documentation**

Last update: May 9, 2019

###########################################

PLEASE READ BEFORE USE: (added May 9, 2019)

Dear users,

1. This script is meant for analyzing linear HIV genomes WITH FLANKING PRIMER binding sites at 5' and 3' ends. This script is made specifically for the analysis of post de novo derived consensus sequences that came from the MGH DNA core de novo assembly pipeline Ultracycler v1.0 (H Wang & B Seed to be published).

2. This pipeline will NOT take plasmid sequences that contain HIV genomes. For this and other customized needs, please feel free to contact me at guinevere.q.lee@gmail.com.

Best,

Guin

###########################################

Objective: This pipeline takes a HIV full-genome fasta and process the genome for any defects. It is optimized to process HIV sequences that are produced this way:

1. Single-template, single-genome amplified from HIV DNA using primers that span 638-9632 (JCI 2017)
2. Shearing-based adaptor-ligation-based MiSeq library prep technology when median read depths of about 1000-2000 reads per base position (MGH DNA core)
3. Post de novo assembled using UltraCycler v1.0 (MGH DNA core, H Wang and B Seed, unpublished)
4. Pipeline input: Raw fasta of the consensus sequences, can include multiple contigs per sample

First, from the input fasta, the pipeline will generate of 3 files

1. Raw fasta: Output\_Concatenate.fasta
2. Reverse complement of the raw fasta: Output\_Concatenate\_RC.fasta
3. Blast mapping results of Output\_Concatenate.fasta against HXB2 1-9719
   1. Developed with blast+ version 2.5.0
   2. custom library construction: HXB2 1-9719
   3. blastn -query Output\_Concatenate.fasta -db HXB2.fasta -num\_alignments 1 -out Output\_20171211HXB2MEGA\_tabdelim.txt -outfmt "6 qseqid qlen sseqid sgi slen qstart qend sstart send evalue bitscore length pident nident btop stitle sstrand"
      1. Understanding E-value: “E value describes the random background noise. For example, an E value of 1 assigned to a hit can be interpreted as meaning that in a database of the current size one might expect to see 1 match with a similar score simply by chance. The lower the E-value, or the closer it is to zero, the more "significant" the match is.”
         1. blastn defaults E-value cutoff: 10
      2. Understanding blastn –task: “1.) “megablast”, for very similar sequences (e.g, sequencing errors), 2.) “dc-megablast”, typically used for inter-species comparisons, 3.) “blastn”, the traditional program used for inter-species comparisons, 4.) “blastn-short”, optimized for sequences less than 30 nucleotides.”
         1. MEGA blast default: only extends alignment when there is an exact match of 28bp

Framework: I will look at a list of things including “HIV/NonHIV”, “AnyInternalInversion”, “Any Hypermut” etc and produce a table summarizing these results. The final verdict will not be called until all pieces of evidence are collected.

Item 1. MyHIVTF\_FINAL. Is this sequence HIV or NonHIV?

1. Table: HXB2.MEGA mappings
2. For each uniq contig in the table, remove any mappings that did not have any parts that fell within 638-9719 (inclusive).
   1. MyDF <- subset(MyDF\_Pre,(MyDF\_Pre$sstart>=638 & MyDF\_Pre$sstart<=9632)|(MyDF\_Pre$send>=638 & MyDF\_Pre$send<=9632)|(MyDF\_Pre$sstart<=638 & MyDF\_Pre$send>=9632)|(MyDF\_Pre$send<=638 & MyDF\_Pre$sstart>=9632))
3. If step (ii) removed all rows (mappings) for this unique contig, it is NonHIV
4. If step (ii) didn’t remove all rows for this unique contig, test for overlapped-mappings (eg. 638-658 and 639-668)
   1. If there is no overlap, sum up the total mapped length by adding up mapped coordinates
   2. If there is overlap, only keep the longest fragment, then sum up the total mapped length
5. If (total mapped length) >=80% of (total contig length), this is HIV.
6. If (total mapped length) <80% of (total contig length), I call this NonHIV.
   1. \*\* I’m being very stringent here to minimizes false positives. The goal is to only let a contig pass this filter if it is purely HIV (no human DNA at all, or at 80% purity).
   2. Please contact me at [guinevere.q.lee@gmail.com](mailto:guinevere.q.lee@gmail.com) for validation notes for my detailed comparison of the performances of a few different blast algorithms in deciding HIV true/false.

Item 2. Internal Inversion.

1. Blast.HXB2.MEGA gives sstrand results (+ or -). It indicates whether a particular region maps to the + or – strand of HIV.
2. For each uniq contig, evaluate whether it is pure-plus, pure-minus, or plus&minus-mix based on NCBI-HXB2 blast+
3. If pure-plus, take the raw sequence as the final sequence for subsequent analysis and label this contig as “plus”
4. If pure-minus, take the RC of the raw sequence as the final sequence for subsequent analysis and label this contig as “minus”
5. If “mix”, label this is labeled “mix”

Item 3. Hypermut

1. To analyze for hypermutations, first, I need to align the sequences.
2. To ensure proper alignment, I export sequences for full-genome alignment based on the above item1 and 2. I export only sequences that are TRUE for being HIV, does not contain internal inversion, and have contig total length >=8000bp.
   1. MyDF\_Hypermut <- subset(DF\_FINAL,DF\_FINAL$MyHIVTF\_FINAL=="TRUE" & DF\_FINAL$MyLenFinal>=8000 & (DF\_FINAL$MyStrandBlastHXB2\_FINAL=="plus" | DF\_FINAL$MyStrandBlastHXB2\_FINAL=="minus"))
3. Add HXB2 and align with muscle
4. Then, I visually examine the alignment to make sure all regions are (roughly) properly aligned without unwelcomed surprises.
5. Then, the script automatically trim away the upstream block before 638, and downstream block after 9632, and save it as Output...ToHypermut...GLtrimmed.fasta
6. Then, the script automatically locate gag’s ATG, and trim away the downstream block after the G in gag’s ATG. This should contain all available bases in psi. Then I manually review for reasonable alignment and whether there are surprises that the pipeline isn’t designed for. Save file as Output...ToHypermut...GLtrimmed\_psigag.fasta
7. Put the file “” through an in-house re-construction of web Hypermut 2.0.
8. If p=values <=0.05, this sequence is TRUE for Hypermut

Item 4. Yes/No Primer

1. Check for the presence of 2nd-round PCR forward primer (U5-638F, GCGCCCGAACAGGGACCTGAAAGCGAAAG, 29bp (only look for positions 2-16, “CGCCCGAACAGGGAC”) within 100bp of contig start to ensure good sequence quality to make decisions regarding 5’DEFECTS.
2. I by-passed position 1 because I’ve seen a lot of sequences missing that base due to poor read quality
3. I don’t search too deep into 3’ to accommodate HIV sequence diversity (while having a clean 5’ is important for proper alignment).

Item 5. Count Indel

1. File “Output...ToHypermut...GLtrimmed\_psigag.fasta” is trimmed to HXB2 638-792 (gag’s ATG) and aligned to HXB2
2. Per contig, count the number of deletions relative to HXB2
3. Per contig, count the number of insertions relative to HXB2
4. Add the two numbers.
5. This represents the total “changes” this contig have, compared to HXB2. The larger this number, the more changes there are.

Item 6. Yes/No Gag ATG.

1. File “Output...ToHypermut...GLtrimmed\_psigag.fasta” is trimmed to HXB2 638-792 (gag’s ATG) and aligned to HXB2
2. In this file, check the last three bases of each contig. Is it an ATG?
3. If ATG, TRUE
4. If not ATG, FALSE
5. I do this check because sometimes I don’t see a start codon.

Up until this point, the process would create a table in which each row represents a unique contiq and all of its analysis results. I call this table Output\_MyBigSummary…DFFINAL.csv.

Item 7. HIVSeqinR ver2.6

1. From the above script, I export the RC-fixed final sequence to be translated and analyzed using HIVSeqinR
2. 20171214 RAM usage requirements reduced and optimized
3. First, prepare a fasta file containing the trimmed reference sequence from HXB2 for each gene and genomic regions of interest
4. I will use gag as an example below
5. First, ensure the input query sequence has no “N” (historic). If there is no “N”, QC01 Passed.
6. Then, map gag to the input query sequence (sample #1)
7. Index the start and end of gag in query sequence’s coordinate
8. If %identity against HXB2 is >=70%, QC02 Passed, otherwise fail
9. Translation method 1 – “Untrimmed.” This method locates the real stop codon without having alignment artifacts and won’t miss N-terminal insertions. Translate from the unaligned file, from query’s indexed start coordinate to the end of the entire query length, then index the first stop codon.
   1. If the length of translated aa \* (AA\_TooShort) is > My expected aa length (HXB2 Gag is 500aa), AND if the length of translated aa \* (AA\_TooLong) is < My expected aa length (HXB2 Gag is 500aa), QC03 Passed.
   2. AA\_TooShort is 95% globally except for gag (97%)
   3. AA\_TooLong is 120% globally
10. Translation method 2 – “Trimmed.” This method exports the aa ranged from HXB2 start to stop, for the purpose of future alignment in blocks. Translate from the unaligned file, from query’s indexed start coordinate to query’s stop coordinate
11. For genes that have ATG: Check to see if the first aa of the trimmed translation starts with M. If not, fail QC04.
    1. if (mydf$Output\_01\_MyWorkingRefNuc\_Name[q]=="HXB2\_gag\_790-2289" | mydf$Output\_01\_MyWorkingRefNuc\_Name[q]=="HXB2\_vif\_5041-5616" | mydf$Output\_01\_MyWorkingRefNuc\_Name[q]=="HXB2\_vpr\_5559-5847mod" | mydf$Output\_01\_MyWorkingRefNuc\_Name[q]=="HXB2\_vpu\_6062-6307" | mydf$Output\_01\_MyWorkingRefNuc\_Name[q]=="HXB2\_env\_6225-8792" | mydf$Output\_01\_MyWorkingRefNuc\_Name[q]=="HXB2\_nef\_8797-9414" | mydf$Output\_01\_MyWorkingRefNuc\_Name[q]=="HXB2\_TAT\_slab" | mydf$Output\_01\_MyWorkingRefNuc\_Name[q]=="HXB2\_REV\_slab") {

The final table contains these columns:

Output\_01\_MyWorkingRefNuc\_Name,

Output\_01a\_MyWorkingRefNuc\_UnAlign\_Seq,

Output\_02\_MyWorkingPreAlnNuc\_Name,

Output\_02a\_QC0\_TargetSeqATCG,

Output\_03a\_MyWorkingAln\_NucRef,

Output\_03b\_MyWorkingAln\_NucTarget,

Output\_03a1\_MyWorkingAln\_NucRef\_Len,

Output\_03a2\_MyWorkingAln\_NucTarget\_Len,

Output\_04a\_MyIndex\_start,

Output\_04b\_MyIndex\_end,

Output\_04c\_MyIndex\_ATCG\_start,

Output\_04d\_MyIndex\_ATCG\_end,

Output\_05a\_QC1\_lentrim1vs2,

Output\_05b\_QC2a\_PercIdentity,

Output\_05c\_QC2b\_PerIden\_Cutoff\_Nuc,

Output\_06a\_MyFinal\_AA,

Output\_06b\_MyFinal\_AA\_len,

Output\_06b2\_MyAA\_Expected\_len,

Output\_06c\_MyFinal\_Nuc,

Output\_06d\_MyFinal\_Nuc\_len,

Output\_06e\_MyFinal\_Stop\_4bp,

Output\_06f\_QC3\_AA\_TooShort,

Output\_07a\_AA\_trimmed\_HXB2,

Output\_07b\_AA\_trimmed\_sample,

Output\_08a\_ATG

Output\_08b\_QC4\_ATG

This table is different from “MyBigSummary.” Each row presents a unique combination of contiq and gene/gene-region. Therefore, the next step is to collapse this table to make it report one unique contig per row so that Item 7 can be merged into “MyBigSummary” table.

Transformation of item 7. PrematureStops or TooLong for GAG, POL, PR, RT, RNaseH, INT, ENV, GP41

1. for each uniq contig, for each of the above genes, export whether QC03 Passed or Failed
2. export the length of the actual (untrimmed) amino acid for each of these genes
3. export whether any of the genes that were expected to have ATG actually had an M as their first amino acid based on the trimmed version (this column is called “MyAnyATGFailed”

Bonus from item 7. Pdf map

1. For each uniq contig that went through item 7, a pdf genome map is generated to give a visual color show of defects in the genome
2. If a gene fails QC02 (low %identity against HXB2), it means the region is likely missing or has diverged a lot from HXB2. This is colored “grey”
3. If a gene fails QC03, there is something wrong with the translation and gene is shown in color “red” in the map.

## FINAL VERDICT ##

For each sequence input, this pipeline will label it with a single verdict. There are 31 possible verdicts highlighted in yellow below (including “Intact,” “PrematureStop,” “Hypermut” etc). When a verdict contains the words “Check,” it is a flag to alert users that there is an abnormality. In these cases, I suggest users to manually validate that the pipeline’s verdict is correct.

## R script for VERDICT; order matters ##

MyVerdict <- c()

for (m in 1:length(DF\_FINAL$MyHIVTF\_FINAL)){

if ((is.na(DF\_FINAL$MyHIVTF\_FINAL[m])==FALSE & DF\_FINAL$MyHIVTF\_FINAL[m]!=TRUE)){

MyVerdict <- append(MyVerdict,"NonHIV")

} else if (is.na(DF\_FINAL$MyLenFinal[m])==FALSE & DF\_FINAL$MyLenFinal[m]<8000){

MyVerdict <- append(MyVerdict,"LargeDeletion")

} else if (is.na(DF\_FINAL$MyStrandBlastHXB2\_FINAL[m])==FALSE & DF\_FINAL$MyStrandBlastHXB2\_FINAL[m]=="mix"){

MyVerdict <- append(MyVerdict,"InternalInversion")

} else if (is.na(DF\_FINAL$MyStrandBlastHXB2\_FINAL[m])==FALSE & DF\_FINAL$MyStrandBlastHXB2\_FINAL[m]=="plusScrambleCheck") {

MyVerdict <- append(MyVerdict,"ScramblePlus")

} else if (is.na(DF\_FINAL$MyStrandBlastHXB2\_FINAL[m])==FALSE & DF\_FINAL$MyStrandBlastHXB2\_FINAL[m]=="minusScrambleCheck") {

MyVerdict <- append(MyVerdict,"ScrambleMinus")

} else if (is.na(DF\_FINAL$MyStrandBlastHXB2\_FINAL[m])==FALSE & DF\_FINAL$MyStrandBlastHXB2\_FINAL[m]=="Check") {

MyVerdict <- append(MyVerdict,"ScrambleCheck")

} else if (is.na(DF\_FINAL$MyHypermutTF[m])==FALSE & DF\_FINAL$MyHypermutTF[m]=="TRUE"){

MyVerdict <- append(MyVerdict,"Hypermut")

#For everything without primers: InferredXXX#

} else if (is.na(DF\_FINAL$MyGAGPassFail[m])==FALSE & (DF\_FINAL$MyPrimerTF[m]=="NoPrimer") ){

#If gag has ATG, all GAG/POL/ENV has to pass to be inferred intact

if (is.na(DF\_FINAL$MyGAGPassFail[m])==FALSE & (DF\_FINAL$MyGagATG[m]==TRUE)){

if (is.na(DF\_FINAL$MyGAGPassFail[m])==FALSE & (substring(DF\_FINAL$MyGAGPassFail[m],1,6)=="Passed" & substring(DF\_FINAL$MyPOLPassFail[m],1,6)=="Passed" & substring(DF\_FINAL$MyINTPassFail[m],1,6)=="Passed" & substring(DF\_FINAL$MyENVPassFail[m],1,6)=="Passed" & substring(DF\_FINAL$MyGP41PassFail[m],1,6)=="Passed")){

MyVerdict <- append(MyVerdict,"Inferred\_Intact")

} else if (is.na(DF\_FINAL$MyGAGPassFail[m])==FALSE & (substring(DF\_FINAL$MyGAGPassFail[m],1,6)=="Failed" | substring(DF\_FINAL$MyPOLPassFail[m],1,6)=="Failed" | substring(DF\_FINAL$MyINTPassFail[m],1,6)=="Failed" | substring(DF\_FINAL$MyENVPassFail[m],1,6)=="Failed" | substring(DF\_FINAL$MyGP41PassFail[m],1,6)=="Failed")) {

MyVerdict <- append(MyVerdict,"Inferred\_PrematureStopORInframeDEL")

} else {

MyVerdict <- append(MyVerdict,"Inferred\_Check#1\_NoPrimerGagATGNoInferredIntactNoInferredPrematureStop")

}

#if gag has no ATG

} else if (is.na(DF\_FINAL$MyGAGPassFail[m])==FALSE & (DF\_FINAL$MyGagATG[m]==FALSE)) {

if (is.na(DF\_FINAL$MyGAGPassFail[m])==FALSE & (substring(DF\_FINAL$MyGAGPassFail[m],1,6)=="Passed" & substring(DF\_FINAL$MyPOLPassFail[m],1,6)=="Passed" & substring(DF\_FINAL$MyINTPassFail[m],1,6)=="Passed" & substring(DF\_FINAL$MyENVPassFail[m],1,6)=="Passed" & substring(DF\_FINAL$MyGP41PassFail[m],1,6)=="Passed")){

MyVerdict <- append(MyVerdict,"Inferred\_Intact\_GagNoATG")

} else if (is.na(DF\_FINAL$MyGAGPassFail[m])==FALSE & (substring(DF\_FINAL$MyGAGPassFail[m],1,6)=="Failed" & substring(DF\_FINAL$MyPOLPassFail[m],1,6)=="Passed" & substring(DF\_FINAL$MyINTPassFail[m],1,6)=="Passed" & substring(DF\_FINAL$MyENVPassFail[m],1,6)=="Passed" & substring(DF\_FINAL$MyGP41PassFail[m],1,6)=="Passed") & (as.numeric(as.character(DF\_FINAL$MyPsiDELrelative[m])) + as.numeric(as.character(DF\_FINAL$MyPsiINrelatiive[m])))>=15 ) {

MyVerdict <- append(MyVerdict,"Inferred\_Intact\_NoGag")

} else if (is.na(DF\_FINAL$MyGAGPassFail[m])==FALSE & (substring(DF\_FINAL$MyGAGPassFail[m],1,6)=="Failed" & (substring(DF\_FINAL$MyPOLPassFail[m],1,6)=="Passed" & substring(DF\_FINAL$MyINTPassFail[m],1,6)=="Passed" & substring(DF\_FINAL$MyENVPassFail[m],1,6)=="Passed" & substring(DF\_FINAL$MyGP41PassFail[m],1,6)=="Passed"))) {

MyVerdict <- append(MyVerdict,"Inferred\_PrematureStopORInframeDEL\_GagNoATGandFailed")

} else if (is.na(DF\_FINAL$MyGAGPassFail[m])==FALSE & (substring(DF\_FINAL$MyPOLPassFail[m],1,6)=="Failed" | substring(DF\_FINAL$MyINTPassFail[m],1,6)=="Failed" | substring(DF\_FINAL$MyENVPassFail[m],1,6)=="Failed" | substring(DF\_FINAL$MyGP41PassFail[m],1,6)=="Failed")) {

MyVerdict <- append(MyVerdict,"Inferred\_PrematureStopORInframeDEL\_GagNoATG")

} else {

MyVerdict <- append(MyVerdict,"Inferred\_Check#2\_NoPrimerGagNoATGNoInferredIntactNoInferredPrematureStop")

}

} else {

MyVerdict <- append(MyVerdict,"Inferred\_Check#3\_NoPrimer")

}

#For everything with primers #

} else if (is.na(DF\_FINAL$MyGAGPassFail[m])==FALSE & (DF\_FINAL$MyPrimerTF[m]=="YesPrimer") ){

#if gag has ATG

if (is.na(DF\_FINAL$MyGAGPassFail[m])==FALSE & (DF\_FINAL$MyGagATG[m]==TRUE)) {

if (is.na(DF\_FINAL$MyGAGPassFail[m])==FALSE & (substring(DF\_FINAL$MyGAGPassFail[m],1,6)=="Failed" | substring(DF\_FINAL$MyPOLPassFail[m],1,6)=="Failed" | substring(DF\_FINAL$MyINTPassFail[m],1,6)=="Failed" | substring(DF\_FINAL$MyENVPassFail[m],1,6)=="Failed" | substring(DF\_FINAL$MyGP41PassFail[m],1,6)=="Failed")) {

MyVerdict <- append(MyVerdict,"PrematureStop")

} else if (is.na(DF\_FINAL$MyGAGPassFail[m])==FALSE & (substring(DF\_FINAL$MyGAGPassFail[m],1,6)=="Passed" & substring(DF\_FINAL$MyPOLPassFail[m],1,6)=="Passed" & substring(DF\_FINAL$MyINTPassFail[m],1,6)=="Passed" & substring(DF\_FINAL$MyENVPassFail[m],1,6)=="Passed" & substring(DF\_FINAL$MyGP41PassFail[m],1,6)=="Passed")) {

if ((as.numeric(as.character(DF\_FINAL$MyPsiDELrelative[m])) + as.numeric(as.character(DF\_FINAL$MyPsiINrelatiive[m])))>=15) {

MyVerdict <- append(MyVerdict,"5DEFECT")

} else if ( (as.numeric(as.character(DF\_FINAL$MyPsiDELrelative[m])) + as.numeric(as.character(DF\_FINAL$MyPsiINrelatiive[m])))<15 ) {

MyVerdict <- append(MyVerdict,"Intact")

} else {

MyVerdict <- append(MyVerdict,"Check#4\_YesPrimerYesGagATGNoPrematureStopNot5DEFECTNotIntact")

}

} else {

MyVerdict <- append(MyVerdict,"Check#5\_YesPrimerYesGagATG")

}

#if gag has no ATG

} else if (is.na(DF\_FINAL$MyGAGPassFail[m])==FALSE & (DF\_FINAL$MyGagATG[m]==FALSE) ) {

if (is.na(DF\_FINAL$MyGAGPassFail[m])==FALSE & (substring(DF\_FINAL$MyPOLPassFail[m],1,6)=="Failed" | substring(DF\_FINAL$MyINTPassFail[m],1,6)=="Failed" | substring(DF\_FINAL$MyENVPassFail[m],1,6)=="Failed" | substring(DF\_FINAL$MyGP41PassFail[m],1,6)=="Failed") & (as.numeric(as.character(DF\_FINAL$MyPsiDELrelative[m])) + as.numeric(as.character(DF\_FINAL$MyPsiINrelatiive[m])))>=15) {

MyVerdict <- append(MyVerdict,"5DFECT\_IntoGag")

} else if (is.na(DF\_FINAL$MyGAGPassFail[m])==FALSE & (substring(DF\_FINAL$MyPOLPassFail[m],1,6)=="Failed" | substring(DF\_FINAL$MyINTPassFail[m],1,6)=="Failed" | substring(DF\_FINAL$MyENVPassFail[m],1,6)=="Failed" | substring(DF\_FINAL$MyGP41PassFail[m],1,6)=="Failed")) {

MyVerdict <- append(MyVerdict,"PrematureStop\_GagNoATG")

} else if (is.na(DF\_FINAL$MyGAGPassFail[m])==FALSE & (substring(DF\_FINAL$MyPOLPassFail[m],1,6)=="Passed" & substring(DF\_FINAL$MyINTPassFail[m],1,6)=="Passed" & substring(DF\_FINAL$MyENVPassFail[m],1,6)=="Passed" & substring(DF\_FINAL$MyGP41PassFail[m],1,6)=="Passed")) {

if ( (substring(DF\_FINAL$MyGAGPassFail[m],1,6)=="Passed") & (as.numeric(as.character(DF\_FINAL$MyPsiDELrelative[m])) + as.numeric(as.character(DF\_FINAL$MyPsiINrelatiive[m])))>=15 ) {

MyVerdict <- append(MyVerdict,"5DEFECT\_GagNoATGGagPassed")

} else if ( (substring(DF\_FINAL$MyGAGPassFail[m],1,6)=="Passed") & (as.numeric(as.character(DF\_FINAL$MyPsiDELrelative[m])) + as.numeric(as.character(DF\_FINAL$MyPsiINrelatiive[m])))<15 ) {

MyVerdict <- append(MyVerdict,"Intact\_GagNoATG")

} else if ((substring(DF\_FINAL$MyGAGPassFail[m],1,6)=="Failed") & (as.numeric(as.character(DF\_FINAL$MyPsiDELrelative[m])) + as.numeric(as.character(DF\_FINAL$MyPsiINrelatiive[m])))>=15 ) {

MyVerdict <- append(MyVerdict,"5DEFECT\_GagNoATGGagFailed")

} else if ((substring(DF\_FINAL$MyGAGPassFail[m],1,6)=="Failed") & (as.numeric(as.character(DF\_FINAL$MyPsiDELrelative[m])) + as.numeric(as.character(DF\_FINAL$MyPsiINrelatiive[m])))<15 ) {

MyVerdict <- append(MyVerdict,"Check#6a\_YesPrimerGagNoATGGagFailed<15bp")

} else {

MyVerdict <- append(MyVerdict,"Check#6\_YesPrimerGagNoATGNo5DEFECTNotIntact")

}

} else {

MyVerdict <- append(MyVerdict,"Check#7\_YesPrimerGagNoATG")

}

} else {

MyVerdict <- append(MyVerdict,"Check#8\_YesPrimer")

}

} else {

MyVerdict <- append(MyVerdict,"Check#9\_NoCategory")

}

}

##################################

HIVSeqinR also has an optional deep sequencing quality control arm. This arm is only relevant for users using the Massachusetts General Hospital (MGH) DNA core for deep-sequencing services.

https://dnacore.mgh.harvard.edu/new-cgi-bin/site/pages/index.jsp

MGH DNA core posts results of deep sequencing runs with compressed \*.sit files containing the sequence quality data in \*.csv format. In this arm, HIVSeqinR receives the \*.csv quality files and evaluates: (i) QC1: percentage of total contig length that has read depth >=10 reads, and would flag a sample as “QC1 failed” if <80% of the bases have low read depth, (ii) QC2: every base position within the contig (with exceptions to the first and last bases of the primer binding sites) must have at least three reads to support the call, otherwise would be flagged as “QC2 failed.” Finally (iii) to control for single HIV genome template input (to exclude sequences derived from the amplification of multiple templates), we flag each base position that has >=5% discordant base calls ,where discordance is defined as having non-random base mixtures that are over 80% biased (e.g. a position with 1000 read-depth will pass QC3 if (1) all 1000 reads calls for A, or (2) if 950 reads calls for A, 16 for T, 16 for C, and 18 for G, but (3) would fail QC3 if 950 reads calls for A, 45 for T, 3 for C and 2 for G (90% biased). A sample that fails any of QC1, QC2 or QC3 are included from subsequent analyses. These QC settings represents a very stringent approach and can be relaxed at users’ discretion.