BMEG 310 Assignment 3 - Group 28

Maximilien Filion - 20598645, Aneesh Kaura - 66832304,
Monica Mihailescu - 47409289

2025-10-23

Introduction

We have taken a tissue sample from a genetically modified mouse and performed RNA sequencing with the intention of exploring the expression levels. Attached to this assignment is a BAM file for one of those individual cells (in '.sam' format to be readable). This file contains reads which have been aligned to the mouse genome.

A BAM file is the standard data template for all of the genetic information in a library. As such, it is the starting point for many analyses approaches. We would like you to become familiar with the contents of a BAM file and their biological implications by investigating the contents of this file.

Use the tables linked to in the appendix to help you interpret the contents of the file.

In your answers, please list all relevant code used and results found.

Data and Requirements

Data for Q1-Q2: single_cell_RNA_seq_bam.sam
Data for Q3: RNA_seq_annotated_variants.vcf

Questions

Q1. Sequencing technologies

Why areas of the genome with high GC content are hard to sequence?

It is important to understand the structural biology of GC regions of DNA. Cytosine and Guanine for three hydrogen bonds, unlike adenine and thymine that form only two. This increase the electrostatic force and pi staking, actively attracting the two molecules together. That being said, when performing biochemical assays such as a polymerase chain reaction, the original strands need to be separated. Rich region will require more energy to "unzip". Furthermore, they will tend to coil up into hairpins and other complex tertiary structures, rendering assays difficult to perform successfully.When thinking about the next generation sequencing workflow, we know that the fragmentation, conversion and adapter ligation steps are not affected by those GC rich regions. The issue comes at the library amplification step, when DNA has to be denatured and cloned - GC rich regions can be underrepresented due to incomplete denaturation or polymerase stalling. Cluster generation on Illumina flow cells can also be affected by the same phenomenon. Even if the clonal expansion of the libraries is performed properly, it is difficult to bind fluorescent probes properly to those GC rich regions due to the DNA secondary structures that prevent proper hybridization or extension during sequencing by synthesis.

Sources:

Most sequencing techniques are biased against sequencing GC rich regions. This mainly comes down to the fact that the GC pairing involves three hydrogen bonds, while AT pairs only have two. That makes GC rich regions more stable and less likely to break apart, which is necessary for all sequencing methods, and therefore have low to no representation in sequencing output [1].

In DNA pairing, A and T exhibit relatively weaker bonding compared to G and C. Higher GC ratios intensify the stability of DNA strand, raising the likelihood of non-disassociation between strands or nonspecific binding of probes to unintended sites. Additionally, high-GC regions might form secondary structures that can interfere with capture or sequencing. Conversely, extremely low GC ratios weaken probe-target DNA binding, potentially leading to probe detachment [2].

[1] https://www.reddit.com/r/askscience/comments/7l27a7/what_makes_whole_genome_sequencing_so_difficult/ [2] https://www.celemics.com/resources/blogs/blog-ngs-hard-to-capture-regions-en/

Q2. Global alignment exercise

Similar to the approach for Needleman–Wunsch algorithm, find the best global alignment between the two following sequences:
ATTCGAC
ATCAC
Use a gap penalty of -2 and the following scoring matrix:

Transition
Transversion
Matrix

	A	T	C	G
A	1	–5	–5	–1
T	–5	1	–1	–5
C	–5	–1	1	–5
G	–1	–5	–5	1

Figure 1: Alt text for the image

In your answer, please include the grid table (used for storing the scores and traceback) and also include how you calculated the first top-left 9 elements of the table.

```
# create the two DNA sequence objects
seq1 <- strsplit("ATTCGAC", split = "")[[1]]
seq2 <- strsplit("ATCAC", split = "")[[1]]

# This function performs the needleman-wunsch alignment on two DNA character strings. It will apply a p
needleman_wunsch_storing_matrix <- function(s1, s2) {

  # convert the DNA sequence to a numeric vector
  s1 <- ifelse(s1 == "A", 1, ifelse(s1 == "T", 2, ifelse(s1 == "C", 3, ifelse(s1 == "G", 4, -1))))
  s2 <- ifelse(s2 == "A", 1, ifelse(s2 == "T", 2, ifelse(s2 == "C", 3, ifelse(s2 == "G", 4, -1))))
```

```
  # create a transition transversion matrix
  penalty_matrix <- rbind(c(1,-5,-5,-1),c(-5,1,-1,-5),c(-5,-1,1,-5), c(-1,-5,-5,1))

  # create gap penalty
  gap_penalty <- -2

  # Needleman-Wunsch algorithm
  # Initiate the matrix
  storing_table <- matrix(0, nrow = (length(s2) + 1), ncol = (length(s1) + 1))
  # fill in with gap penalty on the x-axis
  for (i in 2:ncol(storing_table)){
    storing_table[1,i] <- (-2) * (i-1)
  }
  # fill in with gap penalty on the y-axis
  for (i in 2:nrow(storing_table)){
    storing_table[i,1] <- (-2) * (i-1)
  }

  # fill the table
  for (i in 2:nrow(storing_table)){
    for (j in 2:ncol(storing_table)){
      match_score <- storing_table[i - 1, j - 1] + penalty_matrix[s2[i - 1], s1[j - 1]]
      delete_score <- storing_table[i - 1, j] + gap_penalty
      insert_score <- storing_table[i, j - 1] + gap_penalty

      storing_table[i, j] <- max(match_score, delete_score, insert_score)
    }
  }

  print (storing_table)
}

# call the function on our two DNA strings
seq1_seq2_score_matrix <- needleman_wunsch_storing_matrix(seq1, seq2)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## [1,]    0   -2   -4   -6   -8  -10  -12  -14
## [2,]   -2    1   -1   -3   -5   -7   -9  -11
## [3,]   -4   -1    2    0   -2   -4   -6   -8
## [4,]   -6   -3    0    1    1   -1   -3   -5
## [5,]   -8   -5   -2   -1   -1    0    0   -2
## [6,]  -10   -7   -4   -3    0   -2   -2    1
```

```
# This function performs traceback on a needleman-wunsch score matrix. It returns a vector containing t
needleman_wunsch_traceback_vector <- function(storing_table, s1, s2) {

  # initialize the number of steps that it will take to reach the first cell
  i <- nrow(storing_table)
  j <- ncol(storing_table)
  traceback_vector <- c()

  # convert the DNA sequence to a numeric vector
  s1 <- ifelse(s1 == "A", 1, ifelse(s1 == "T", 2, ifelse(s1 == "C", 3, ifelse(s1 == "G", 4, -1))))
```

```
  s2 <- ifelse(s2 == "A", 1, ifelse(s2 == "T", 2, ifelse(s2 == "C", 3, ifelse(s2 == "G", 4, -1))))

  # create a transition transversion matrix
  penalty_matrix <- rbind(c(1,-5,-5,-1),c(-5,1,-1,-5),c(-5,-1,1,-5), c(-1,-5,-5,1))

  # create gap penalty
  gap_penalty <- -2

  while (i > 1 && j > 1) {
    score <- storing_table[i, j]
    diag_score <- storing_table[i - 1, j - 1] + penalty_matrix[s2[i - 1], s1[j - 1]]
    up_score   <- storing_table[i - 1, j] + gap_penalty
    left_score <- storing_table[i, j - 1] + gap_penalty

    # check which direction produced the current score
    if (score == diag_score) {
      traceback_vector <- c(traceback_vector, "diag")
      i <- i - 1
      j <- j - 1
    } else if (score == up_score) {
      traceback_vector <- c(traceback_vector, "up")
      i <- i - 1
    } else if (score == left_score) {
      traceback_vector <- c(traceback_vector, "left")
      j <- j - 1
    }
  }

  print(traceback_vector)
}

# call the function on the score matrix found precedently
needleman_wunsch_traceback_vector(seq1_seq2_score_matrix, seq1, seq2)
```

```
## [1] "diag" "diag" "left" "diag" "diag" "left" "diag"
```

The Needleman–Wunsch algorithm was used to perform the alignment. A scoring matrix was initialized based on the lengths of the two sequences, with the first row and column filled with gap penalties. At each step, the algorithm compared bases to determine matches, mismatches, or gaps, and filled the matrix accordingly. Finally, a traceback process identified the sequence of steps leading to the optimal alignment. The first 9 elements of the table were computed as discussed above.

Q3. Looking at the Metadata of an alignment (SAM) file

Q3.1.

Q3.1. Use `read.csv("single_cell_RNA_seq_bam.sam", nrows=73, sep="\t", header=FALSE, fill=TRUE)` to load the first 73 lines of the header of the file and print the contents. These lines contain tabulated information about the BAM file and the circumstances of its data collection. According to the header table in section 1.3 of the BAM/SAM document in the appendix, what do the `SN` and `LN` tags indicate?

```
single_cell_RNA_seq_bam <- read.csv("single_cell_RNA_seq_bam.sam", nrows=73, sep="\t", header=FALSE,
fill=TRUE)
```

Q3.2.

A sequence is any template string of bases to which we can align a read. This includes chromosomes (which are continuous sequences of bases) and new strings resulting from genetic modifications. What is the length of the X chromosome, in bp, for our alignment?

Fun fact (not tested): One of the sequences in this BAM file titled Cre_ERT2 is a Cre-recombinase variant. Cre recombinase, when combined with the loxP sequence (see cre-lox recombination) is the primary element used in many experiments (such as this one) to induce a genetic modification to certain cells in vivo. This allows us to study the effect of changing a gene at any time during the life cycle of an organism, and has allowed us to make discoveries in many areas including stem cell research.

Q4. Looking at the Reads of an alignment (SAM) file

Q4.1.

Use the code below to load the reads into an R dataframe. Each row contains one read. How many reads are there in this BAM file?

```
sam <- read.csv("single_cell_RNA_seq_bam.sam", sep="\t", header=FALSE,
comment.char="@", col.names = paste0("V",seq_len(30)), fill=TRUE)
sam <- sam[paste0("V",seq_len(11))]
```

Q4.2.

Print out the 10th row of the dataframe to look at the format of a read. Compare it to the mandatory BAM fields table in section 1.4 of the SAM/BAM documentation in the appendix. **The order of columns in the bam file have been preserved in the dataframe**. Which column of your dataframe should you look at to find the chromosome to which the read was aligned? To which BAM data field does the dataframe column "V11" correspond?

Q4.3. How many reads in this file align to chromosome X?

Hint: You can compare a column vector to a constant using logical symbols (==, <, >, etc.) to get a column vector of TRUE or FALSE. Remember, when summing, a true symbol is worth "1" while a false symbol is worth "0".

Q4.4. What is the mean base quality (BQ) for reads aligning to chromosome X?

Q4.5. Plot the distribution of BQs across all bases and reads as a boxplot. Comment on your observation.

Hint: This is similar to a boxplots that was provided in the lecture related to primary analysis.

Q4.6. Referring to section 1.4 of the SAM/BAM documentation, what column contains the leftmost mapping position of the reads?

Q4.7.

In order to transform a BAM file into expression levels for each gene, we need to count the number of reads covering a particular location or gene. The protein Hspa8 is located on chromosome 9 at bases 40801273 - 40805199. How many reads have their leftmost mapping position aligned within these coordinates?

Hint: you can implement AND logic on two column vectors with "&".

Q4.8. Mapping quality is an indication of how well a read aligned to the reference genome during

the alignment step of processing our library data. It is reported as an integer between 0 and 255. How many reads have mapping quality less than 50?

Q4.9. What is the mean mapping quality of the reads which have mapping quality less than 50?

Hint: you can obtain a subset of a dataframe by using df[bool_vec,] where bool_vec contains TRUE/FALSE elements and bool_vec and df have the same number of rows.

Q4.10. (bonus): The genome of the mouse used in this experiment has been edited to include the

DNA sequence for the protein 'tdTomato', which is a fluorophore. Count the number of reads which align to the tdTomato sequence. Assuming that these reads are accurate, would you expect this cell to emit fluorescently? What might be the purpose of modifying a genome to include a fluorophore?

Hint: Think about studying cell populations under a microscope.

Q5. Investigating the Variants

We have used Strelka, which is a variant-calling tool, to find all of the SNPs and short indels in the genome of this cell using the BAM file. The variants were then annotated using snpEff to label them with information such as which gene they affect and the type of modification they result in once the RNA undergoes translation to a protein. The results are in a VCF file (extension '.vcf') which is attached.

Q5.1.

Use the following lines of code to obtain the header of the file and a dataframe where each row is a variant. As you can see, information in the VCF file is organised by multiple levels of character-separated data, so it will take multiple rounds of parsing to extract relevant information. For the first variant (row) in the dataframe, what is the reference allele base at the site, and what is the alternative allele called by Strelka?

```
vcf_con <- file("RNA_seq_annotated_variants.vcf", open="r")
vcf_file <- readLines(vcf_con)
close(vcf_con)
vcf <- data.frame(vcf_file)
header <- vcf[grepl("##", vcf$vcf_file), ]
factor(header)
```

```
##   [1] ##fileformat=VCFv4.1
##   [2] ##fileDate=20200930
##   [3] ##source=strelka
##   [4] ##source_version=2.9.2
##   [5] ##startTime=Wed Sep 30 13:12:59 2020
##   [6] ##contig=<ID=1,length=195471971>
##   [7] ##contig=<ID=10,length=130694993>
##   [8] ##contig=<ID=11,length=122082543>
##   [9] ##contig=<ID=12,length=120129022>
##  [10] ##contig=<ID=13,length=120421639>
##  [11] ##contig=<ID=14,length=124902244>
##  [12] ##contig=<ID=15,length=104043685>
##  [13] ##contig=<ID=16,length=98207768>
##  [14] ##contig=<ID=17,length=94987271>
##  [15] ##contig=<ID=18,length=90702639>
##  [16] ##contig=<ID=19,length=61431566>
##  [17] ##contig=<ID=2,length=182113224>
##  [18] ##contig=<ID=3,length=160039680>
##  [19] ##contig=<ID=4,length=156508116>
##  [20] ##contig=<ID=5,length=151834684>
##  [21] ##contig=<ID=6,length=149736546>
##  [22] ##contig=<ID=7,length=145441459>
##  [23] ##contig=<ID=8,length=129401213>
##  [24] ##contig=<ID=9,length=124595110>
##  [25] ##contig=<ID=MT,length=16299>
##  [26] ##contig=<ID=X,length=171031299>
##  [27] ##contig=<ID=Y,length=91744698>
##  [28] ##contig=<ID=JH584299.1,length=953012>
##  [29] ##contig=<ID=GL456233.1,length=336933>
##  [30] ##contig=<ID=JH584301.1,length=259875>
##  [31] ##contig=<ID=GL456211.1,length=241735>
##  [32] ##contig=<ID=GL456350.1,length=227966>
##  [33] ##contig=<ID=JH584293.1,length=207968>
##  [34] ##contig=<ID=GL456221.1,length=206961>
##  [35] ##contig=<ID=JH584297.1,length=205776>
##  [36] ##contig=<ID=JH584296.1,length=199368>
##  [37] ##contig=<ID=GL456354.1,length=195993>
##  [38] ##contig=<ID=JH584294.1,length=191905>
##  [39] ##contig=<ID=JH584298.1,length=184189>
##  [40] ##contig=<ID=JH584300.1,length=182347>
##  [41] ##contig=<ID=GL456219.1,length=175968>
##  [42] ##contig=<ID=GL456210.1,length=169725>
##  [43] ##contig=<ID=JH584303.1,length=158099>
##  [44] ##contig=<ID=JH584302.1,length=155838>
##  [45] ##contig=<ID=GL456212.1,length=153618>
##  [46] ##contig=<ID=JH584304.1,length=114452>
##  [47] ##contig=<ID=GL456379.1,length=72385>
##  [48] ##contig=<ID=GL456216.1,length=66673>
##  [49] ##contig=<ID=GL456393.1,length=55711>
##  [50] ##contig=<ID=GL456366.1,length=47073>
##  [51] ##contig=<ID=GL456367.1,length=42057>
##  [52] ##contig=<ID=GL456239.1,length=40056>
##  [53] ##contig=<ID=GL456213.1,length=39340>
##  [54] ##contig=<ID=GL456383.1,length=38659>
```

```
##  [55] ##contig=<ID=GL456385.1,length=35240>
##  [56] ##contig=<ID=GL456360.1,length=31704>
##  [57] ##contig=<ID=GL456378.1,length=31602>
##  [58] ##contig=<ID=GL456389.1,length=28772>
##  [59] ##contig=<ID=GL456372.1,length=28664>
##  [60] ##contig=<ID=GL456370.1,length=26764>
##  [61] ##contig=<ID=GL456381.1,length=25871>
##  [62] ##contig=<ID=GL456387.1,length=24685>
##  [63] ##contig=<ID=GL456390.1,length=24668>
##  [64] ##contig=<ID=GL456394.1,length=24323>
##  [65] ##contig=<ID=GL456392.1,length=23629>
##  [66] ##contig=<ID=GL456382.1,length=23158>
##  [67] ##contig=<ID=GL456359.1,length=22974>
##  [68] ##contig=<ID=GL456396.1,length=21240>
##  [69] ##contig=<ID=GL456368.1,length=20208>
##  [70] ##contig=<ID=JH584292.1,length=14945>
##  [71] ##contig=<ID=JH584295.1,length=1976>
##  [72] ##contig=<ID=tdTomato,length=2250>
##  [73] ##contig=<ID=SSM2_GFP,length=1619>
##  [74] ##contig=<ID=CreERT2,length=1983>
##  [75] ##content=strelka germline small-variant calls
##  [76] ##INFO=<ID=END,Number=1,Type=Integer,Description="End position of the region described in this
##  [77] ##INFO=<ID=BLOCKAVG_min30p3a,Number=0,Type=Flag,Description="Non-variant multi-site block. Non
##  [78] ##INFO=<ID=SNVHPOL,Number=1,Type=Integer,Description="SNV contextual homopolymer length">
##  [79] ##INFO=<ID=CIGAR,Number=A,Type=String,Description="CIGAR alignment for each alternate indel al
##  [80] ##INFO=<ID=RU,Number=A,Type=String,Description="Smallest repeating sequence unit extended or c
##  [81] ##INFO=<ID=REFREP,Number=A,Type=Integer,Description="Number of times RU is repeated in referen
##  [82] ##INFO=<ID=IDREP,Number=A,Type=Integer,Description="Number of times RU is repeated in indel al
##  [83] ##INFO=<ID=MQ,Number=1,Type=Integer,Description="RMS of mapping quality">
##  [84] ##FORMAT=<ID=GT,Number=1,Type=String,Description="Genotype">
##  [85] ##FORMAT=<ID=GQ,Number=1,Type=Integer,Description="Genotype Quality">
##  [86] ##FORMAT=<ID=GQX,Number=1,Type=Integer,Description="Empirically calibrated genotype quality sc
##  [87] ##FORMAT=<ID=DP,Number=1,Type=Integer,Description="Filtered basecall depth used for site genoty
##  [88] ##FORMAT=<ID=DPF,Number=1,Type=Integer,Description="Basecalls filtered from input prior to sit
##  [89] ##FORMAT=<ID=MIN_DP,Number=1,Type=Integer,Description="Minimum filtered basecall depth used fo
##  [90] ##FORMAT=<ID=AD,Number=.,Type=Integer,Description="Allelic depths for the ref and alt alleles
##  [91] ##FORMAT=<ID=ADF,Number=.,Type=Integer,Description="Allelic depths on the forward strand">
##  [92] ##FORMAT=<ID=ADR,Number=.,Type=Integer,Description="Allelic depths on the reverse strand">
##  [93] ##FORMAT=<ID=FT,Number=1,Type=String,Description="Sample filter, 'PASS' indicates that all fil
##  [94] ##FORMAT=<ID=DPI,Number=1,Type=Integer,Description="Read depth associated with indel, taken fr
##  [95] ##FORMAT=<ID=PL,Number=G,Type=Integer,Description="Normalized, Phred-scaled likelihoods for ge
##  [96] ##FORMAT=<ID=PS,Number=1,Type=Integer,Description="Phase set identifier">
##  [97] ##FORMAT=<ID=SB,Number=1,Type=Float,Description="Sample site strand bias">
##  [98] ##FILTER=<ID=IndelConflict,Description="Indel genotypes from two or more loci conflict in at l
##  [99] ##FILTER=<ID=SiteConflict,Description="Site is filtered due to an overlapping indel call filte
## [100] ##FILTER=<ID=LowGQX,Description="Locus GQX is below threshold or not present">
## [101] ##FILTER=<ID=HighDPFRatio,Description="The fraction of basecalls filtered out at a site is grea
## [102] ##FILTER=<ID=HighSNVSB,Description="Sample SNV strand bias value (SB) exceeds 10">
## [103] ##FILTER=<ID=LowDepth,Description="Locus depth is below 3">
## [104] ##FILTER=<ID=NotGenotyped,Description="Locus contains forcedGT input alleles which could not b
## [105] ##FILTER=<ID=PloidyConflict,Description="Genotype call from variant caller not consistent with
## [106] ##FILTER=<ID=NoPassedVariantGTs,Description="No samples at this locus pass all sample filters
## [107] ##SnpEffVersion="4.3t (build 2017-11-24 10:18), by Pablo Cingolani"
## [108] ##INFO=<ID=ANN,Number=.,Type=String,Description="Functional annotations: 'Allele | Annotation
```

```
## [109] ##INFO=<ID=LOF,Number=.,Type=String,Description="Predicted loss of function effects for this va
## [110] ##INFO=<ID=NMD,Number=.,Type=String,Description="Predicted nonsense mediated decay effects for
## 110 Levels: ##content=strelka germline small-variant calls ...
```

```
variants <- read.csv("RNA_seq_annotated_variants.vcf", skip=length(header),
header=TRUE, sep="\t")
```

Hint: Take a look at the VCF Variant Call Format document in the appendix for details on each column name.

Q5.2. The INFO field is organised into variables by the form 'TAG=value' (see the VCF Variant Call

Format document). Write code to obtain the entirety of the ANN info value contents from the INFO field for the first variant.

Hint: You will need strsplit() and grep()/grepl() to accomplish this. Take a look at https://www.math.ucla.edu/~anderson/rw1001/library/base/html/strsplit.html and https://stackoverflow.com/questions/21311386/using-grep-to-help-subset-a-data-frame-in-r for how to make use of them. With which character should you split the string?

Hint: Make sure to convert the INFO field entry to string format using as.character() so that it can be passed into strsplit().

Q5.3.

Each INFO tag-value pair is detailed in a line of the header, beginning with the tag '##INFO=<ID=VARIABLE, ...'. Look for the header entry starting with '##INFO=<ID=ANN, ...' which details the format of the ANN value contents. This tag-value pair contains the results of the annotations found by snpEff. Based on the ANN value of the first variant, what does the 'Annotation' field tell us about this variant?

Hint: snpEff can return multiple annotation entries for the same variant because some variants may have multiple possible effects. The first annotation entry is the most confident/important and, resultantly, you should only look at the first entry to answer this and all subsequent question. You can use strsplit() again with ',' separation character if you wish to look at each of the ANN entries separately.

Hint: Refer to the snpEff documentation in the appendix for a list of snpEff annotation label names and summaries of their effects.

Q5.4. Perform the parsing done in Q5.1-3 again on variant line 683. What gene would this variant affect?

Q5.5. Within the entire VCF file, how many HIGH impact variants we see in total?

Q5.6. What is a frameshift variant? Does it have a greater or lesser effect on the resultant protein than a missense variant? Why?

Q5.7. We can divide variants into two broad categories: intronic/intergenic and exonic. Count the

number of potential intronic variants. What do you notice about the number of intronic variants (compared to overall number of variants)?

Hint: Use grepl() on the INFO field to look for tell-tale tags.

Hint: assume no overlap between exonic and intronic tags within a variant entry.

Q5.8. List all the genes that have been affected by coding mutations and have high impact. What do you find that is interesting?

Hint: You can use SNPeff HIGH/MODERATE impact field to help you finding those genes.

Q5.9. (bonus):

Using Strelka on our data, we can detect indels, but only to a limited extent. Most of the reads in our BAM file have read lengths around 60bp long. Why might this have consequences for the detection of insertions that are longer than 60bp?

Q5.10.

Variant Allele Frequency (VAF) is an important metric that helps us to measure how many DNA molecules in a given sample are carrying a given variant. It also helps to identify potential false-positive situations caused by incorrect base calls or alignment. VAF is calculated by

The number of variant alleles /(The number of Variant alleles + The number of Reference alleles)

In the form of a boxplot, plot the distribution of the VAFs across all the variants. How many variants have VAF $> 5\%$? How many of those variants (the ones with $>5\%$ VAF) are in coding regions?

Hint: You will need to parse the genotype encoding field (GT:GQ:GQX:DP:DPF...) to get allele counts and then get VAF. To understand that column, look at the VCF Variant Call Format Document (GATK) section 5.

APPENDIX

SAM/BAM Format Specification Document: https://samtools.github.io/hts-specs/SAMv1.pdf VCF Variant Call Format Document (GATK): https://gatk.broadinstitute.org/hc/en-us/articles/36003 5531692-VCF-Variant-Call-Format snpEff Annotations Document: http://snpeff.sourceforge.net/VCFannotationformat_ v1.0.pdf