Count reads in regions

```
#source("https://bioconductor.org/biocLite.R")
#biocLite("Rsamtools")
library(Rsamtools)
```

Read in all input files

```
#GFF file information
gffFileInfo <- read.csv(file = "un_bowtie_gffFileInfo.csv",
                        stringsAsFactors = FALSE,
                        check.names = FALSE)

#Sample information
sampleInfo <- read.csv("un_bowtie_sampleInfo.csv",
                       stringsAsFactors = FALSE,
                       check.names = FALSE)

#Complete path to output folder
outFolderP <- "/Users/ls483/Documents/SRA.GEO/RProj_RNAseq/output_gene_cds"

#If bams file have a common extension,you can input it here.Eg.SRR1946637_un.bowtie2.sorted.
#So sample name = "SRR1946637" and samplExt = "_un.bowtie2.sorted""
sampleExt = "_un.bowtie2.sorted.bam"
```

This is the main function that is called:

```
funcBatchProcess <- function(gff.folder.path, gff.file.names, ref,
                             sample.folder.paths, sample.names, sampleNameExt,
                             outFolderPath, map.q.threshold = 0){
    #browser()
    #org.path = system('getwd()', intern = TRUE)
    org.path = getwd()
    #counts.matrix.list only saves in.region column from function 'gene.level.counts'
    counts.matrix.list = vector('list', length(gff.file.names)) #initialize empty vector
    names(counts.matrix.list) = gsub('.gff', '', gff.file.names)
    #counts.list only saves all 3 columns from function 'gene.level.counts'
    counts.list = counts.matrix.list

    #Loop for every GFF file
    for(g in 1:length(gff.file.names)){
        #browser()
        print(g)
        #setwd(gff.folder.path)

        #Read GFF file
        gg <- paste(gff.folder.path[g],"/",gff.file.names[g], sep="")


        if(file.exists(gg)) {
```

```
        gff = funcReadGffFile(gg)
        #       cat(g, ' ', gff.file.names[g], '\n')

        region.pos = funcGetRegionPosFromGff(gff)
        #       last row is unannotated parts of genome
        counts.matrix.list[[g]] = matrix(0, nrow(region.pos) + 1, length(sample.names))
        rownames(counts.matrix.list[[g]]) = c(rownames(region.pos), 'unannotated parts of genome')
        colnames(counts.matrix.list[[g]]) = paste('col', 1:length(sample.names), sep='_') # list of
        counts.list[[g]] = vector('list', length(sample.names))
        names(counts.list[[g]]) = paste('list', 1:length(sample.names), sep='_')

        #intended to store subset of sam/bam file for one specific virus
        viral.sam.file = gsub('.gff', '.sam', gff.file.names[g])

        #loop for every sample
        for(s in 1:length(sample.folder.paths)){
            print(s)
            cat(g, gff.file.names[g], '\t', s, sample.names[s], '\n')
            setwd(sample.folder.paths[s])

            b = paste(sample.names[s],sampleNameExt,sep="") #name of bam file

            if(file.exists(b)) {

                #call function to get rows from bam file corres to one gff
               # browser()
                sam = funcSubsetBam(sample.names[s],sample.name.sampleExt = sampleNameExt, refName =

                counts = matrix(-1, nrow(region.pos), 3)
                colnames(counts) = c('in.region','on.boundary','in.gaps')
                rownames(counts) = rownames(region.pos)

                if(is.null(sam)){
                    reads.pos = NULL
                }else{
                    # Function to get Positions
                    reads.pos = funcGetReadPosFromSam(sam, map.q.threshold)
                    if(!is.null(reads.pos))
                        #Function to get Region Level Counts
                        counts = funcRegionLevelCounts(reads.pos, region.pos)
                }
                g1 <- gsub('.gff', '', gff.file.names[g])
                fileName1 <- paste(outFolderPath,"/",g1,".RData" ,sep="")
                save(region.pos, reads.pos, counts, file = fileName1)

                counts.matrix.list[[g]][, s] = c(counts[,1], counts[1,3])
                colnames(counts.matrix.list[[g]])[s] = sample.names[s]
                counts.list[[g]][[s]] = counts
                names(counts.list[[g]])[s] = sample.names[s]
                #browser()

                fileNameA <- paste(outFolderPath,"/",g1,"..",sample.names[s],"..counts.RData" ,sep=
                fileNameB <- paste(outFolderPath,"/",g1,"..",sample.names[s],"..counts.csv" ,sep=""
```

```
                    save(counts, file = fileNameA)
                    write.csv(counts, file = fileNameB)

                } else {
                    print(paste(b, ": BAM file does not exist", sep=""))
                }
            } #end of sample for loop

        } else {
            print(paste(gg,"GFF file does not exist",sep=""))
        }

    } #End of For gff loop
    setwd(org.path)
    save(counts.matrix.list, file='counts.matrix.list.Rdata')
    save(counts.list, file='counts.list.Rdata')

} #end of FuncBatchProcess
```

This is a sub-function (an internal function)

```
## sam.file:    NC_022518.sam
## sample.name: sample.name without .bwa.bam/sam
funcSubsetBam <- function(sample.name, sample.name.sampleExt = ".bam", refName){
    #browser()

    bam.file = paste(sample.name,sample.name.sampleExt,sep="")

    #source("https://bioconductor.org/biocLite.R")
    #biocLite("Rsamtools")
    #library("Rsamtools")
    bfl = Rsamtools::BamFile(bam.file)
    which1 <- as(seqinfo(bfl)[refName], "GRanges")
    which1

    params <- ScanBamParam(which = which1, what = c("qname", "flag","rname","pos","mapq","cigar"))

    temp <- Rsamtools::scanBam(file =bam.file , param = params)
    temp1 <- as.data.frame(temp)

    colnames(temp1) <- c("qname", "flag","rname","pos","mapq","cigar")

    #subset for specific virus
    #temp2 <- dplyr::filter(temp1, rname == refName)
    temp2 <- temp1


    return(temp2)
}
```

This is a sub-function (an internal function)

```
###################
## cigar parsing functions
## usage: sapply(cigars = c('26S2I5D36M', '28S22M', '39M10S'), funcCigarSums)

funcMatcher <- function(pattern, x) {
    ind = gregexpr(pattern, x)[[1]]
    start = as.numeric(ind)
    end = start + attr(ind, "match.length")- 2
    apply(cbind(start,end), 1, function(y) substr(x, start=y[1], stop=y[2]));
}
```

This is a sub-function (an internal function)

```
funcDoOne <- function(c, cigar) {
    pat <- paste("\\d+", c , sep="")
    sum(as.numeric(funcMatcher(pat, cigar)), na.rm=T)
}
```

This is a sub-function (an internal function)

```
## function
funcCigarSums <- function(cigar, chars=c("M","N","D","I","S","H", "P", "X", "=")) {
    sapply (X = chars, funcDoOne, cigar)
}
```

This is a sub-function (an internal function)

```
###################
funcGetReadPosFromSam <- function(sam, map.q.threshold = 40){
    #browser()
    map.q.flag = sam[, 5] >= map.q.threshold
    star.cigar.flag = sam[, 6] != '*'
    idx = map.q.flag & star.cigar.flag
    if(length(which(idx)) < 1)
        return(NULL)
    cigars = as.character(sam[which(idx), 6])

    cigar.summary = sapply(cigars, funcCigarSums)

    reads.length = cigar.summary[1,] + cigar.summary[3,] - cigar.summary[4,]
    reads.pos = cbind(as.numeric(sam[which(idx), 4]), as.numeric(sam[which(idx), 4]) + reads.length - 1
    colnames(reads.pos) = c('start', 'end')
    rownames(reads.pos) = NULL
    return(reads.pos)
}
```

This is a sub-function (an internal function)

```
### read gff from gff file, skip first 5 and the last comment line
funcReadGffFile <- function(gff.file, head.skip = 5, tail.ignored = 1){
    #browser()
    print(gff.file)
```

```
    gff = read.delim(gff.file, header=F, skip = head.skip)
    gff = gff[1:(nrow(gff)-tail.ignored), ]
    return(gff)
}
```

This is a sub-function (an internal function)

```
### get region position, n * 2, start, end
funcGetRegionPosFromGff <- function(gff){
    #browser()
    region.pos = as.matrix(gff[,c(4,5)])
    colnames(region.pos) = c('start', 'end')
    rownames(region.pos) = paste(as.character(gff[,1]), #NC_022518.1
                                 as.character(gff[,3]), #long_terminal_repeat/region/gene/CDS
                                 as.numeric(gff[,4]),   #start  1112
                                 as.numeric(gff[,5]),   #end    6746
                                 sep='_')
    return(region.pos)
}
```

This is a sub-function (an internal function)

```
# reads.pos and genes.pos are same format, N * 2 matrix, with column 'start' and 'end'
funcRegionLevelCounts <- function(reads.pos, region.pos){
    #browser()
    n.counts = matrix(0,nrow(region.pos),3)
    colnames(n.counts) = c('in.region','on.boundary','in.gaps')
    rownames(n.counts) = rownames(region.pos)
    low = region.pos[, 1]
    high = region.pos[, 2]
    for(i in 1:nrow(reads.pos)){
        start = reads.pos[i, 1]
        end = reads.pos[i, 2]
        idx.w = ((start >= low) & (end <= high))
        if(sum(idx.w) > 0)
            n.counts[idx.w, 1] = n.counts[idx.w, 1] + 1

        idx.r = ((start >= low) & (start < high) & (end > high))    # on right boundary
        idx.l = ((start < low) & (end > low) & (end <= high))   # on left boundary
        idx.c = ((start < low) & (end > high))  # cover region
        idx.b = idx.r | idx.l | idx.c
        if(sum(idx.b) > 0)
            n.counts[idx.b, 2] = n.counts[idx.b, 2] + 1


        if(sum(idx.w | idx.b) == 1)
            n.counts[1, 3] = n.counts[1, 3] + 1
    }
    return(n.counts)
}
```

Calling main function `funcBatchProcess`

```
#calling func
funcBatchProcess(gffFileInfo$gff.folder.path,
                 gffFileInfo$gff.file.names,
                 gffFileInfo$ref,
                 sampleInfo$sample.folder.paths,
                 sampleInfo$sample.names,
                 sampleNameExt = sampleExt,
                 outFolderPath = outFolderP)
```