Plasmid copy number

Plasmid copy number from sequence mapping

My previous blog entry was about numerous types of analysis you can do from just looking at basic mapping statistics, without even having to look at the actual alignments or anything yet. Another neat thing we did a while back using just that, was determine the copy number of secondary chromosomal contigs by comparing the coverage of these contigs to that of the chromosome.

This script is not intended as an automatic script that is usable on your dataset, although it is possible to automate it more. For this particular problem i did 2 years ago, both the file extentions and the contig labels were hardcoded, and are dataset specific, and as an example script i think it still serves it's purpose to illustrate the problem and sollution, so i won't be ammending it. So just to reitterate

#This script is intended as a template only!

First off we'll start with some basic functions to keep our working area clean and tidy (although we don't need them)

```
# A random function found on stackoverflow to remove all currently loaded pacakges... need this to prev
detachAllPackages <- function() {
  basic.packages <- c("package:stats","package:graphics","package:grDevices","package:utils","package:da
  package.list <- search()[ifelse(unlist(gregexpr("package:",search()))==1,TRUE,FALSE)]
  package.list <- setdiff(package.list,basic.packages)
  if (length(package.list)>0)  for (package in package.list) detach(package, character.only=TRUE)
}
detachAllPackages()

# A random function to remove all objects from workspace
clear_workspace <- function() {
  rm(list=ls())
}
clear_workspace()

# Function to check if package is already installed, if not, installs it. After installation it loads i
install_load <- function(Required_Packages) {
  for(package in Required_Packages){
    if (!package %in% installed.packages()) install.packages(package, character.only = TRUE)
    library(package, character.only = TRUE)
  }
}
```

For this example we don't need much, we'll use the count function in plyr, we'll plot using ggplot2 and scales to transform the figure somewhat.

```
#Install and load the following packages
install_load(c("ggplot2","scales","plyr"))
```

We used PacBio reference for this, so it only has 2 contigs, the chromosome and the plasmid, which we know the name of. For your example this might be different and might need additional lines to define each "level" or "factor" what type they are off.

Our input data looked something like this

```
##                   V1 V2 V3      V4          V5
## 1 tig00000001_pilon  0 10 5211076 1.91899e-06
## 2 tig00000001_pilon  1 34 5211076 6.52456e-06
## 3 tig00000001_pilon  2  4 5211076 7.67596e-07
## 4 tig00000001_pilon  3  5 5211076 9.59495e-07
## 5 tig00000001_pilon  4  2 5211076 3.83798e-07
## 6 tig00000001_pilon  5  1 5211076 1.91899e-07
```

This data contains several contig labels

```
## [1] "tig00000001_pilon" "tig00000097_pilon" "genome"
```

tig000000001__pilon and tig000000097__pilon being our chromosome and plasmid, but as you can see, when running 'bedtools genomecov -d' we also obtained a global overview coverage, labeled "genome". I forgot what option was used that generated this, but in this example we removed it from our analaysis. Your version probably won't have these lines in your 'bedtools genomecov -d' file and this is why this script does require some additional labour.

```
# Load the data
# Using file_choose we'll pick the output from "bedtools genomecov -d"
table_input_142 <- read.table(sep = "\t", header = FALSE, file.choose(), na.strings=c("","NA"))
# Now remove "genome" entries, as this is a summary of all seperate contigs, not the entire assembly
table_input_142 <- table_input_142[table_input_142$V1!="genome",c(1,2,3)]
# Add 142 to the list so that we can differentiate the individual sets later on
table_input_142 <- cbind(strain = "142", table_input_142)
# Count the number of contigs in the table (in this case it's going to be 2... i checked manually)
sumContigs_142 <- count(table_input_142, "V1")
# So this is a manual thing... not automatic... i know there are two contigs, and i know the first (and
# So i'll rename level 1 to chr and level 2 to pADAP-type plasmid... this replaces the contig names
table_input_142[which(table_input_142$V1==sumContigs_142[1,1]),2] <- "Chr"
table_input_142[which(table_input_142$V1==sumContigs_142[2,1]),2] <- "pADAP-type plasmid"
```

For my particular dataset i had 6 PacBio samples in total, so we loaded another 5 in the same way. One of them had two secondary chromosomes, so we just added a third descriptor called "Secondary plasmid"

```
table_input_143 <- read.table(sep = "\t", header = FALSE, file.choose(), na.strings=c("","NA"))
table_input_143 <- table_input_143[table_input_143$V1!="genome",c(1,2,3)]
table_input_143 <- cbind(strain = "143", table_input_143)
sumContigs_143 <- count(table_input_143, "V1")
table_input_143[which(table_input_143$V1==sumContigs_143[1,1]),2] <- "Chr"
table_input_143[which(table_input_143$V1==sumContigs_143[2,1]),2] <- "pADAP-type plasmid"
table_input_143[which(table_input_143$V1==sumContigs_143[3,1]),2] <- "Secondary plasmid"
```

etc… for 440, 626, 1048, and RM5

The generated table_inputs have now been reduced to nucleotides per coverage, with a strain name factor. So we can easily merge them together now.

```
# Now because we added the strain names to a seperate column, we can rbind all the data without issues
coverage_table <- rbind(table_input_142,
                        table_input_143,
                        table_input_440,
                        table_input_626,
                        table_input_1048,
                        table_input_RM5)
```

As some samples have better quality than other, the depth of coverage might be a little different. To keep the plots informative, we therefore converted the nucleotide frequency (per x coverage) to a log10 scale, that way it's easier to plot the coverage on one axis, and the frequency of nucleotides (that have that coverage) on the other axis without having weird axis

Additionally, to make this more visual, we converted plasmid related contigs into negative values so they mirror the chromosomal contig coverage

```
# All "Chr" levels will remain positive values with a log scale to keep the coverage in a range we can
# All plasmid levels will be converted to negative so they will mirror the chr coverage plot nicely
coverage_table$V3[coverage_table$V1=="Chr"] <- sapply(coverage_table$V3[coverage_table$V1=="Chr"],log)
coverage_table$V3[coverage_table$V1=="pADAP-type plasmid"] <- sapply(coverage_table$V3[coverage_table$V
coverage_table$V3[coverage_table$V1=="Secondary plasmid"] <- sapply(coverage_table$V3[coverage_table$V1
```

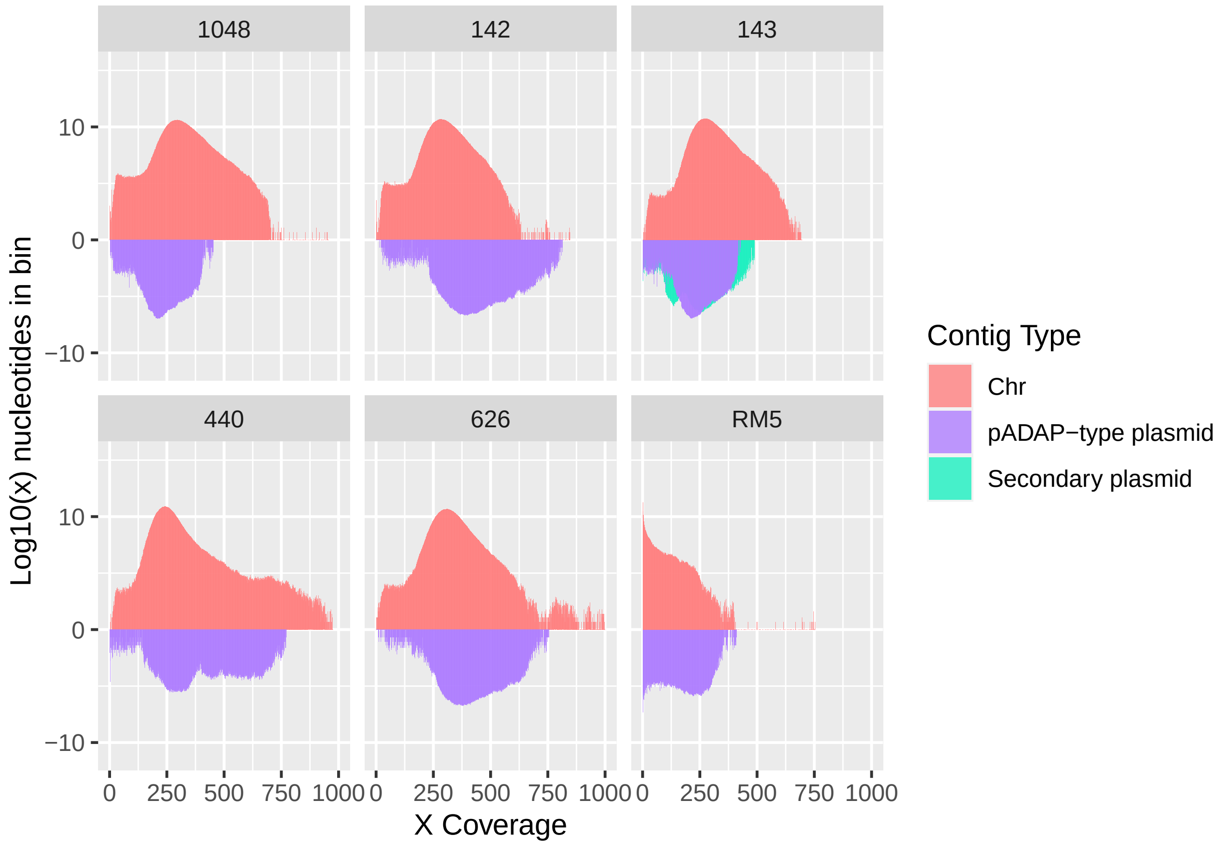
Lastly, we just clean up the table a bit, convert it to a data.frame, give it proper column names for the plot to work with, and make sure we didn't sneak any NAs or other non numeric values in there. You should always double check before coorecing data into a new type that you aren't introducing false data or any mistakes into your final output though!

```
coverage_table <- as.data.frame(coverage_table)
colnames(coverage_table) <- c("Strain","Type","Cov","Freq")
coverage_table$Cov <- as.numeric(as.character(coverage_table$Cov))
coverage_table$Freq <- as.numeric(as.character(coverage_table$Freq))
```

Now plotting will be as easy as followed

```
# Plot that bad boy!!!!
p <- ggplot(coverage_table, aes(Cov, Freq, fill=Type)) +
  labs(y = "Log10(x) nucleotides in bin", x = "X Coverage", fill ="Contig Type") +
  geom_bar(stat="identity", position="identity") +
  facet_wrap(~Strain) +
  xlim(0,1000) +
#  scale_y_continuous(label=ytick_formatter) +
#  theme(axis.ticks = element_blank(), axis.text.y = element_blank()) +
  scale_fill_manual(values=alpha(c("#ff8080", "#af7eff","#1bf0c0", "#178bad", "#cd3700"),0.8))
p
```

```
## Warning: Removed 47 rows containing missing values (geom_bar).
```



Now the plot will contain negative bin numbers on the y-axis for the plasmids… this is too much effort to change in the R code, much easier to import into inkscape, adobe illustrator or CorelDraw and just remove the little dash '-' character… takes 5 seconds, or you can spend 2 hours programming it in R for a script you probably will only use once haha.

Have fun, and thanks for flying Lesley airlines!