Example code to call mutation densities

Verena Körber

2022-09-03

Example code to time chromosomal gains using mutation densities

This file contains example code to compute the mutation densities at chromosomal segments, stratified by copy number state and mutation multiplicity. The code is exemplarily run on a single tumor, requiring as input the .vcf file containing the SNVs and the copy number information as produced by ACEseq. In the first part, the number of clonal mutations per chromosomal segment is counted. Thereafter, mutation densities are computed by converting the mutation counts into "mutation time". Because chromosomal gains reduce the number of single-copy clonal mutations by copying the mutations to multiple copies, the number of single-copy clonal mutations is extrapolated to the expected count if there had been no gain. In this way, we render mutation densities between different segments comparable, allowing us to interpret them as "mutation time". From the normalized densities we will finally compute the densities at the MRCA of the tumor and test whether there is evidence for an early common ancestor preceding the MRCA. If you want to reproduce this code on your own computer, please download the scripts and the data from github and set the directory paths in the following accordingly.

We begin by loading the required libraries:

```
library("RColorBrewer")
library(ggplot2); theme_set(theme(panel.grid.major = element_blank(), panel.grid.minor =
↪  element_blank(), text=element_text(size=8, color="black"),
                                  panel.background = element_blank(), axis.line =
                                  ↪  element_line(colour = "black", size=0.75),
                                  axis.ticks = element_line(color = "black", size=0.75),
                                  axis.text = element_text(size=8, color="black")))
library(bedr)
```

```
##
##
## ######################
## #### bedr v1.0.7 ####
## ######################
##
## checking binary availability...
##   * Checking path for bedtools... FAIL
##   * Checking path for bedops... FAIL
##   * Checking path for tabix... FAIL
## tests and examples will be skipped on R CMD check if binaries are missing
```

```
library(NBevolution)
```

Set the directories

We then store the name of the folder containing the data,

```
data.directory <- "../Example_data/"
```

as well as the name of the subdirectory containing the SNV file,

```
snv.directory <- "SNVs/"
```

the subdirectory containing the ACEseq file,

```
cnv.directory <- "ACEseq/"
```

Eventually, we store the name of the directory containing custom scripts used in the analysis (this corresponds to the location of the folder "Analysis_and_plot" on github),

```
custom.script.directory <-"../Custom_scripts/"
```

and the name of the directory, where RData-objects are to be stored

```
rdata.directory <- "RData/"
```

Count the clonal mutations per segment stratified by mutation multiplicity

We exemplarily analyze the tumor NBE11:

```
tumor <- "NBE11"
```

Find the ACEseq file:

```
aceseq <- list.files(paste0(data.directory, "/", tumor, "/", cnv.directory, "/"),
↪  pattern="comb_pro_extra", full.names = T)[1]
```

Extract ploidy and purity estimates from ACEseq file

```
purity.ploidy <- Extract.purity.ploidy.from.ACEseq(aceseq)
```

Find the SNV file

```
mutations <- list.files(paste0(data.directory, "/", tumor, "/", snv.directory, "/"),
↪  pattern="somatic_snvs_conf_8_to_10", full.names = T)[1]
```

Count the clonal mutations on autosomal segments with CN between 1 and 4. The implemented function estimates the number of clonal mutations stratified by multiplicity using weighted binomial clustering.

```
clonal.mutations <- count.clonal.mutations(aceseq, mutations, chromosomes = c(1:22),
↪  ploidy. = purity.ploidy$ploidy, purity.=purity.ploidy$purity)
```

```
## READING VCF
##  * checking if file exists... PASS
##  * Reading vcf header...
##    Done
##  * Reading vcf body...
##    Done
##  * Parse vcf header...
##    Done
```

This returns a list object containing a matrix of clonal counts stratified by multiplicity (denoted by roman numbers):

```
print(clonal.mutations$clonal.mutation.matrix)
```

```
##                  chr1  chr2   chr3 chr4   chr5   chr6   chr7 chr8 chr9 chr10
## monosomic_I      0.00   0.0   0.00    0   0.00   0.00   0.00    0    0     0
## disomic_I        2.00 160.0 178.00  206   0.00   0.00   0.00  154   74   104
## disomic_II       0.00   0.0   0.00    0   0.00   0.00   0.00    0    0     0
## trisomic_I     227.36  84.6  11.70    0 209.52 267.30  35.72    2    0     0
## trisomic_II      2.32   2.7   7.15    0   3.24   1.35   1.14    0    0     0
## trisomic_III     0.00   0.0   0.00    0   0.00   0.00   0.00    0    0     0
## tetrasomic_I     0.00   0.0   0.00    0   0.00   0.00 120.28    0    0     0
## tetrasomic_II    0.00   0.0   0.00    0   0.00   0.00   1.86    0    0     0
## tetrasomic_III   2.00   0.0   0.00    0   0.00   0.00   0.00    0    0     0
## tetrasomic_IV    0.00   0.0   0.00    0   0.00   0.00   0.00    0    0     0
##                chr11 chr12 chr13 chr14 chr15 chr16 chr17 chr18 chr19 chr20
## monosomic_I        0     0   0.0     0   0.0   0.0  0.00     0  0.00     0
## disomic_I         70     0   0.0    84   0.0  46.0  0.00     0 40.00     0
## disomic_II         0     0   0.0     0   0.0   0.0  0.00     0  0.00     0
## trisomic_I        52   144 147.0     0 117.6  10.4  0.00     0 27.90    66
## trisomic_II        0     3   1.5     0   1.2  14.8  0.00     0  1.05     0
## trisomic_III       0     0   0.0     0   0.0   0.0  0.00     0  0.00     0
## tetrasomic_I       0     0   0.0     0   0.0   0.0 94.08     0  0.00     0
## tetrasomic_II      0     0   0.0     0   0.0   0.0  1.96     0  0.00     0
## tetrasomic_III     0     0   0.0     0   0.0   0.0  0.00     0  0.00     0
## tetrasomic_IV      0     0   0.0     0   0.0   0.0  0.00     0  0.00     0
##                chr21 chr22
## monosomic_I     0.00     0
## disomic_I       0.00    20
## disomic_II      0.00     0
## trisomic_I     39.90    10
## trisomic_II     1.05     0
## trisomic_III    0.00     0
## tetrasomic_I    0.00     0
## tetrasomic_II   0.00     0
## tetrasomic_III  0.00     0
## tetrasomic_IV   0.00     0
```

along with a second matrix that stores the corresponding segment lengths:

```
print(clonal.mutations$segment.length.matrix)
```

```
##                     chr1      chr2      chr3      chr4      chr5      chr6
## monosomic_I            0         0         0         0         0         0
## disomic_I       13594734 168601853 189221543 186534868    339959    171985
## disomic_II       1594766         0         0 186387966         0         0
## trisomic_I     205790673  65558780   5129218         0 174940443 166171786
## trisomic_II    205790673  65558780   5129218         0 174940443 166171786
## trisomic_III           0         0         0         0         0         0
## tetrasomic_I      209269       282         0         0         0     94813
## tetrasomic_II      77514       282         0         0         0         0
## tetrasomic_III    131755         0         0         0         0     94813
## tetrasomic_IV          0         0         0         0         0         0
##                    chr7      chr8      chr9     chr10    chr11     chr12
## monosomic_I           0         0         0         0        0         0
## disomic_I         24955 140512092 108239158 129246089 80683121         0
## disomic_II            0         0         0         0        0         0
## trisomic_I     24503732    617712         0         0 47744162 127359830
## trisomic_II    24503732    617712         0         0 47744162 127359830
## trisomic_III          0         0         0         0        0         0
## tetrasomic_I   71720484         0         0         0        0         0
## tetrasomic_II  71671725         0         0         0        0         0
## tetrasomic_III    48759         0         0         0        0         0
## tetrasomic_IV         0         0         0         0        0         0
##                   chr13    chr14    chr15    chr16    chr17 chr18    chr19
## monosomic_I           0        0        0        0        0     0   212759
## disomic_I             0 86622747        0 46129208        0     0 36111725
## disomic_II            0        0        0        0        0     0        0
## trisomic_I     95480252        0 76114216 28868691        0     0 18687132
## trisomic_II    95480252        0 76114216 28868691        0     0 18687132
## trisomic_III          0        0        0        0        0     0        0
## tetrasomic_I      79857        0    77616        0 61917812     0        0
## tetrasomic_II         0        0        0        0 61917812     0        0
## tetrasomic_III    79857        0    77616        0        0     0        0
## tetrasomic_IV         0        0        0        0        0     0        0
##                   chr20    chr21    chr22
## monosomic_I           0        0        0
## disomic_I             0        0 25249695
## disomic_II            0        0        0
## trisomic_I     58069078 32659909  9080279
## trisomic_II    58069078 32659909  9080279
## trisomic_III          0        0        0
## tetrasomic_I          0        0        0
## tetrasomic_II         0        0        0
## tetrasomic_III        0        0        0
## tetrasomic_IV         0        0        0
```

Compute the mutation density at each genomic segment by converting the mutation counts into mutation time

In the next step, we first translate mutation counts into "mutation time". Gained segments will have fewer single-copy clonal SNVs because some are copied to higher multiplicities during the gain. To compare single-copy mutation counts between gained and normal segments, we therefore normalize the data by extrapolating the measured number to the expected counts had there been no gain. This is done by the function "Mutation.time.converter".

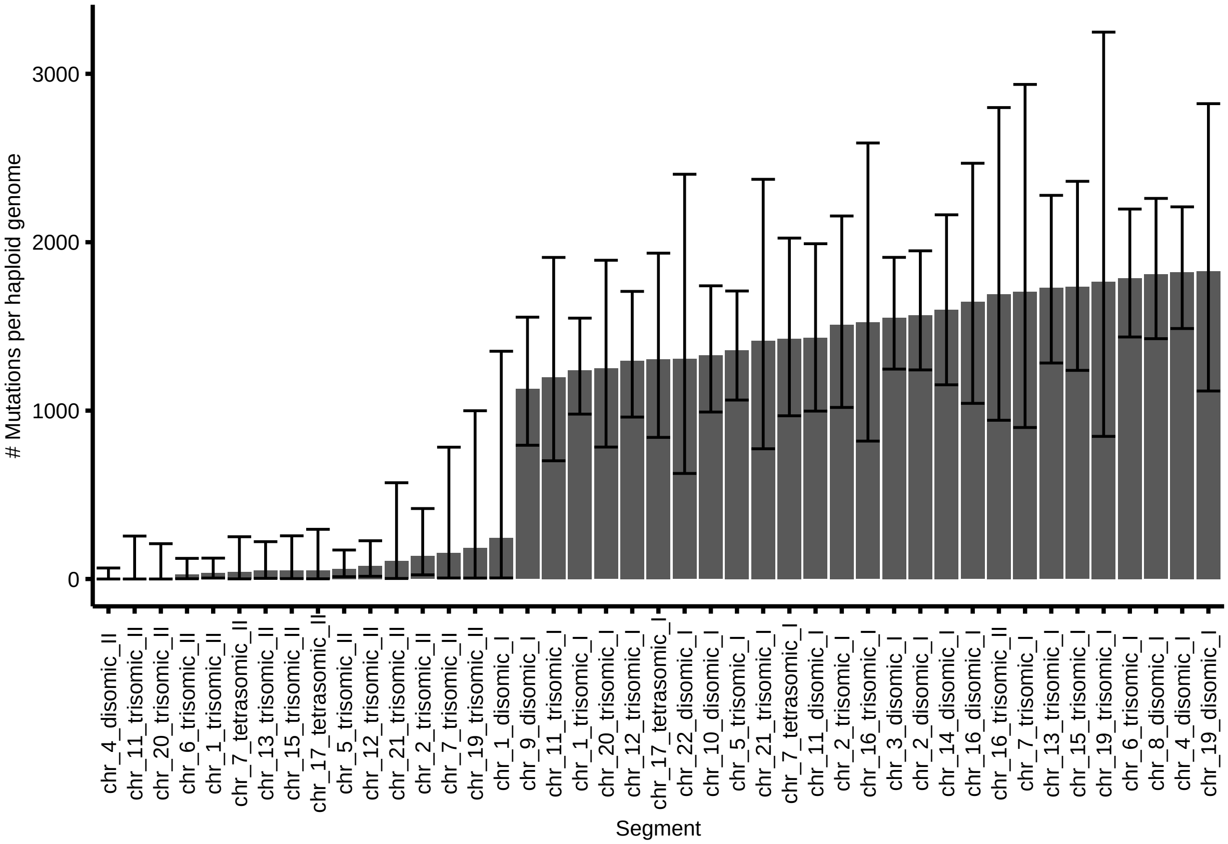
Thereafter, we model mutation accumulation as a Poisson process. The number of mutations acquired on a piece of DNA depends on the per-base mutation rate (at this time) and the length of the piece. Using a chisquare-approximation, we obtain confidence intervals for the mutation time.

We do this iteratively for each chromosome.

```
mutation.time <- data.frame()

## iterate through the chromosomes
for(j in 1:22){

  ## Get normalized mutation counts per copy and segment on this chromosome
  tmp.mut.count <- Mutation.time.converter(clonal.mutations$clonal.mutation.matrix[,j])

  tmp.genome.length <- clonal.mutations$segment.length.matrix[,j]
  ## Restrict analysis to fragments > 10^7 bp
  tmp.mut.count <- tmp.mut.count[which(tmp.genome.length>10^7)]
  tmp.genome.length <- tmp.genome.length[tmp.genome.length>10^7]

  if(length(tmp.mut.count)==0){next}

  ## Convert to mutations per haploid genome and store output

  mutation.time = rbind(mutation.time,
                        data.frame(Mean =  tmp.mut.count*3.3*10^9/tmp.genome.length,
                                   Min = 0.5*qchisq(0.025,
                                   ↪   tmp.mut.count*2)/tmp.genome.length*3.3*10^9,
                                   Max = 0.5*qchisq(0.975,
                                   ↪   (tmp.mut.count*2+2))/tmp.genome.length*3.3*10^9,
                                   Segment = paste("chr", j, names(tmp.mut.count),
                                   ↪   sep="_")))

}
mutation.time$Segment <- factor(mutation.time$Segment, levels =
↪   mutation.time$Segment[order(mutation.time$Mean)])
```

We visualize the mutation density per segment

```
ggplot(mutation.time,
            aes(x=Segment, y=Mean, ymin=Min, ymax=Max)) + geom_col() +
            ↪   geom_errorbar(aes(x=Segment, ymin=Min, ymax=Max)) +
  scale_y_continuous(limits=c(0, max(mutation.time$Max)), name = "# Mutations per haploid
  ↪   genome") +
  theme(axis.text.x = element_text(angle=90), panel.grid.major = element_blank(),
  ↪   panel.grid.minor = element_blank(),
        panel.background = element_blank(), axis.line = element_line(colour = "black"))
```

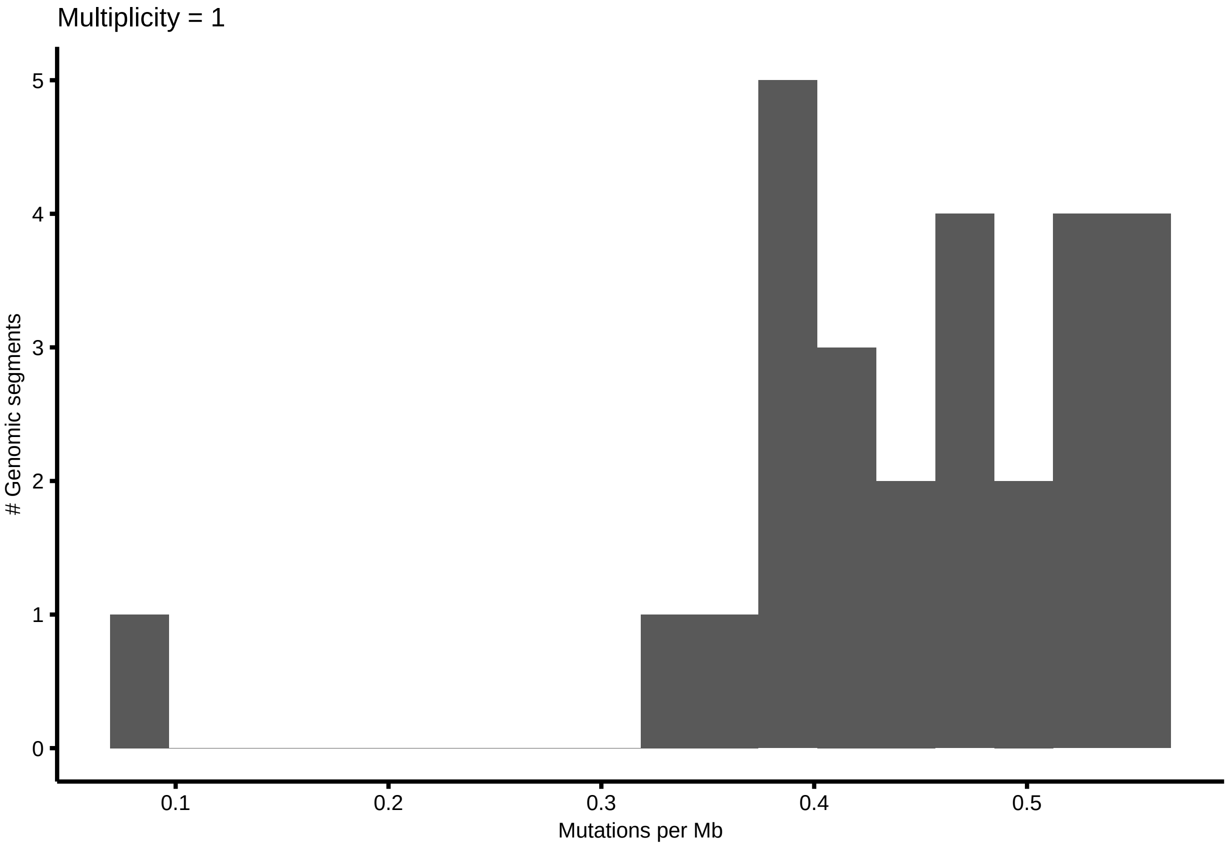


Next, we plot the densities as histograms, separately for lower- and higher-order SNVs. We here transform to #SNVs/Mb by dividing by 3.3*10^3. First, we prepare the input to plotting.

```
## set the binwidth to 20 bins
binwidth = (max(mutation.time$Mean/3.3/10^3) - min(mutation.time$Mean/3.3/10^3))/20

to.plot <- mutation.time
to.plot$Mean <- to.plot$Mean/3.3/10^3
to.plot$Max <- to.plot$Max/3.3/10^3
to.plot$Min <- to.plot$Min/3.3/10^3
## distinguish mutation multiplicity
to.plot$Timing <- sapply(as.character(to.plot$Segment), function(x){
  x <- strsplit(x, split="_")[[1]]
  if(is.na(x[4])){
    return("Multiplicity>1")
  }
  if(x[4]=="I"){
    return("Multiplicity=1")
  }else{
    return("Multiplicity>1")
  }
})
to.plot$Segment <- factor(to.plot$Segment, levels=unique(to.plot$Segment))
```

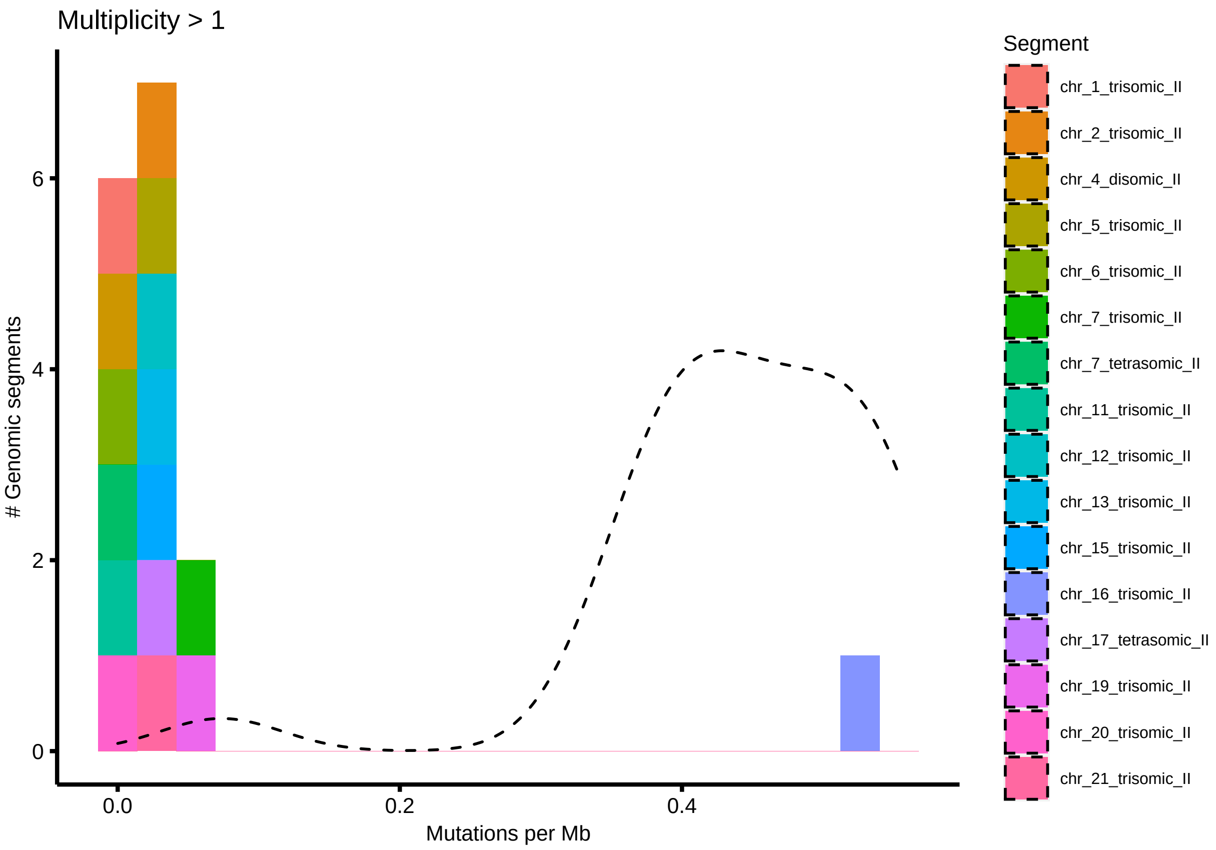
Now we plot the mutations at multiplicity = 1:

```
ggplot(to.plot[to.plot$Timing=="Multiplicity=1",], aes(x=Mean)) + geom_histogram(
↪  binwidth = binwidth)+
  scale_y_continuous(name="# Genomic segments") +
  scale_x_continuous(name="Mutations per Mb") + ggtitle("Multiplicity = 1")
```



Then, we plot the density of mutations at multiplicity > 1 and compare them to the (normalized) density (dashed line) of mutations with multiplicity $= 1$; we color-encode the gained segments.

```
ggplot(to.plot[to.plot$Timing=="Multiplicity>1",], aes(x=Mean, fill=Segment)) +
↪  geom_histogram( binwidth = binwidth)+
  geom_density(data=to.plot[to.plot$Timing=="Multiplicity=1",], linetype=2, aes(x=Mean),
  ↪  inherit.aes = F) +
  scale_y_continuous(name="# Genomic segments") +
  scale_x_continuous(name="Mutations per Mb") +
  ggtitle("Multiplicity > 1")
```



By visual inspection, the gain of chromosome 16 maps to the MRCA, while the others do not.

Estimate the mutational density at ECA and MRCA

We use negative binomial distributions as statistic model for the mutation distribution along the genome and correct for putative over-estimation of clonal variants due to sampling. To this end, we try to estimate false positive clonal SNVs from the 2 sample pairs available. Briefly, mutations called as clonal in the primary tumor but not identified at all in the relapsed tumors were likely false positive clonals. We assess their fraction in the 2 sample pairs and take the average as our empirical estimate of the false positive rate. You can re-run this analysis by downloading the data from the 2 tumor pairs NBE11/NBE66, NBE51/NBE78 and by running the folllowing code

```
##  primary tumor of tumor pairs
#primary.of.tumor.pairs <- c("NBE11", "NBE51")
## and relapse tumor of tumor pairs
#relapse.of.tumor.pairs <- c("NBE66", "NBE78")
#data.directory.discovery <- data.directory
#source(paste0(custom.script.directory, "Estimate_sampling_bias_from_tumor_pairs.R"))
```

Alternatively you can directly load the result

```
load(paste0(rdata.directory, "Assess_false_positives_due_to_sampling.RData"))
```

This reads in two vectors, “clonal.mutations.false.positives”, and “clonal.mutations.all”, referring the number of false positive clonals and the total number of clonal mutations for each of the tumor pairs.

```
print(clonal.mutations.false.positives)
```

```
## NBE11 NBE51
##   312    67
```

```
print(clonal.mutations.all)
```

```
## NBE11 NBE51
##  1695   566
```

Having done this, we now quantify the densities at ECA and MRCA and test whether individual gains map to the MRCA, or to the ECA, or to neither of them. This is done by the function MRCA.ECA.quantification. The function first estimates the density at MRCA as the average density of all normalized single-copy clonal mutations. The function then tests for each segment, whether the single-copy clonal mutations on this segment are consistent with the density at the MRCA using a negative binomial model. Next, the function tests for each segment, whether the density of multi-copy clonal mutations is consistent with the density at the MRCA. If yes, the segment is assigned to the MRCA. Thereafter, the function computes the average density of multi-copy clonal mutations on segments not mapping to the MRCA and classifies the result as the density at the ECA. It then iteratively tests for each segment using negative binomial distributions, whether the density of multi-copy clonal mutations is consistent with the density at the ECA. If yes, the segment is mapped to the ECA, if not, it is taken out and the mutation density at the ECA is recalculated. In the last step, the function tests whether any segment timed at MRCA is also consistent with the mutation density at the ECA.

```
mrca.eca <- MRCA.ECA.quantification(clonal.mutations$clonal.mutation.matrix,
↪  clonal.mutations$segment.length.matrix)
```

This returns a list object containing the estimates for the mutation density at MRCA (per haploid genome):

```
print(mrca.eca$mutation.time.mrca)
```

```
## [1] 1275.558
```

along with the lower and upper bounds of a bootstrapped 95% CI:

```
print(mrca.eca$mutation.time.mrca.lower)
```

```
##     2.5%
## 1141.867
```

```
print(mrca.eca$mutation.time.mrca.upper)
```

```
##    97.5%
## 1446.675
```

Note that, due to the randomness in the bootstrapping, lower and upper bounds may vary slightly between runs. To convert the estimates to #SNVs/Mb, divide by 3.3*10^3:

```
print(mrca.eca$mutation.time.mrca/3.3/10^3)
```

```
## [1] 0.3865326
```

Similarly, the list contains the estimates for the (mean and 95%CI) density at ECA:

```
print(mrca.eca$mutation.time.eca)
```

```
## [1] 44.01231
```

```
print(mrca.eca$mutation.time.eca.lower)
```

```
##     2.5%
## 23.85312
```

```
print(mrca.eca$mutation.time.eca.upper)
```

```
##    97.5%
## 56.07753
```

Moreover, the object contains the segments mapping to the MRCA,

```
print(mrca.eca$gains.at.mrca)
```

```
## [1] "chr_7_trisomic_II"  "chr_16_trisomic_II" "chr_19_trisomic_II"
```

In this example, some of them are also consistent with the density at the ECA:

```
print(mrca.eca$gains.at.mrca.conforming.eca)
```

```
## [1] "chr_7_trisomic_II"  "chr_19_trisomic_II"
```

while other segments uniquely map to the ECA:

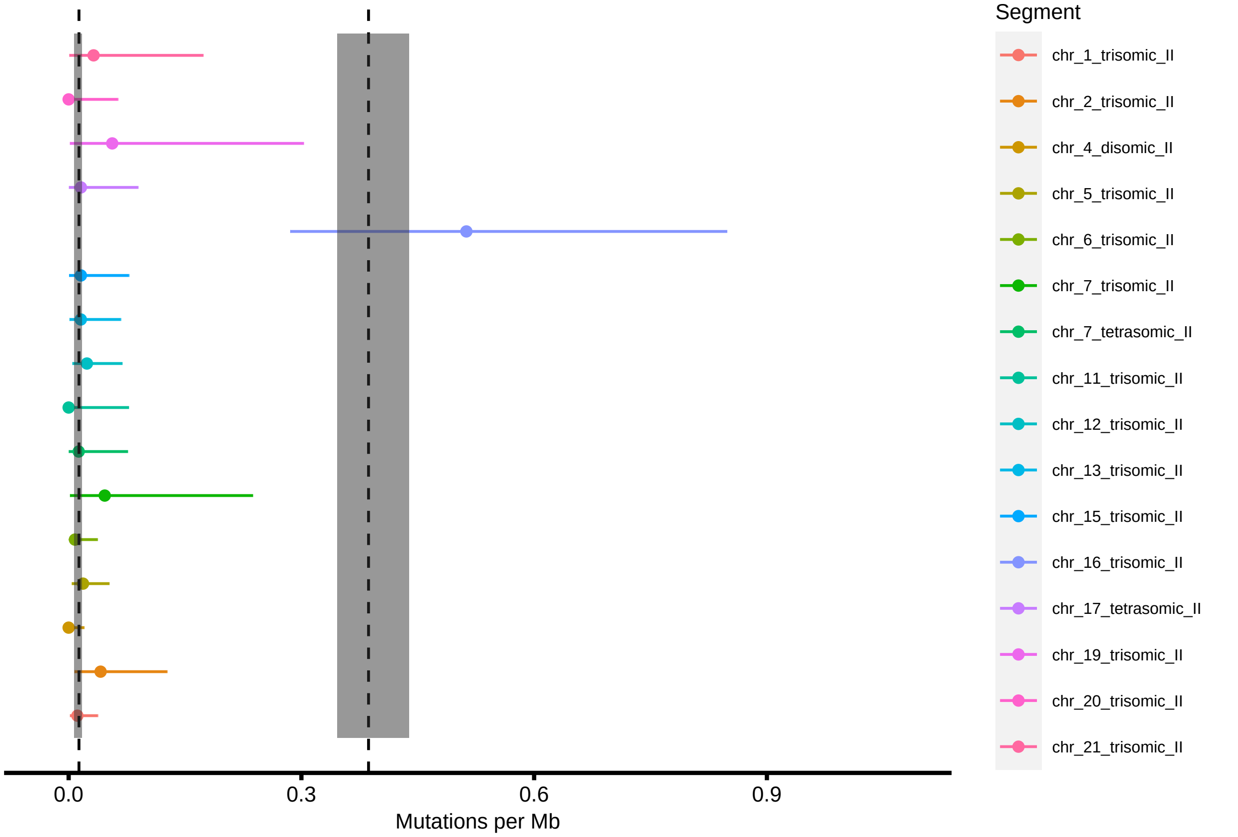
```
print(mrca.eca$gains.uniquely.mapped.to.eca)
```

```
##  [1] "chr_1_trisomic_II"    "chr_2_trisomic_II"    "chr_4_disomic_II"
##  [4] "chr_5_trisomic_II"    "chr_6_trisomic_II"    "chr_7_tetrasomic_II"
##  [7] "chr_11_trisomic_II"   "chr_12_trisomic_II"   "chr_13_trisomic_II"
## [10] "chr_15_trisomic_II"   "chr_17_tetrasomic_II" "chr_20_trisomic_II"
## [13] "chr_21_trisomic_II"
```

Consistent with our visual impression when looking at the densities, most segments map to the ECA, while chromosome 16 uniquely maps to the MRCA. However, we cannot say with certainty whether Chr 7 and 19 were gained at MRCA or at ECA.

A different way to visualize this result is to plot the densities at ECA and MRCA with 95% CI together with the densities at the individual segments with 95% CI

```
to.plot. <- to.plot[to.plot$Timing=="Multiplicity>1",]
to.plot.$Segment <- factor(to.plot.$Segment, levels=unique(to.plot.$Segment))

ggplot(to.plot., aes(x=Mean, xmin=Min, xmax=Max, y=as.numeric(Segment) -
                       0.5, col=Segment)) +
  geom_point() + geom_errorbarh(height=0)+
  geom_vline(data=data.frame(x=mrca.eca$mutation.time.mrca), aes(xintercept=x/3.3/10^3,
  ↪  y=1), inherit.aes = F,
             linetype=2) +
  geom_ribbon(data=data.frame(xmin=rep(mrca.eca$mutation.time.mrca.lower,2),
                              xmax=rep(mrca.eca$mutation.time.mrca.upper,2),
                              y=c(0, nrow(to.plot.))),
              aes(xmin=xmin/3.3/10^3, xmax=xmax/3.3/10^3,y=y),
              inherit.aes = F, col=NA, alpha=0.5) +
  geom_vline(data=data.frame(x=mrca.eca$mutation.time.eca), aes(xintercept=x/3.3/10^3,
  ↪  y=1), inherit.aes = F,
             linetype=2) +
  geom_ribbon(data=data.frame(xmin=rep(mrca.eca$mutation.time.eca.lower, 2),
                              xmax=rep(mrca.eca$mutation.time.eca.upper,2),
                              y=c(0, nrow(to.plot.))),
              aes( xmin=xmin/3.3/10^3, xmax=xmax/3.3/10^3, y=y),
              inherit.aes = F, col=NA, alpha=0.5) +
  scale_x_continuous(name="Mutations per Mb", limits =  c(-0.05*(max(to.plot$Mean)),
  ↪  max(to.plot$Max)*1.1)) +
scale_y_continuous(limits=c(0, nrow(to.plot.))) + theme(axis.line.y=element_blank(),
↪  axis.text.y=element_blank(),
                            axis.ticks.y=element_blank(),
                            ↪  axis.title.y=element_blank())
```



You can see that the densities at Chr 7 and 19 have large CIs. This is because the gained segments are small and thus the uncertainty in mutation density is large. As a consequent, their densities are consistent with both the density at ECA and at MRCA and we cannot say exactly when the gains occurred (you may note that the 95%CIs of the two gains do not overlap with the 95%CIs of the MRCA. They were nevertheless not called as significantly different from the MRCA because we corrected the p values for multiple sampling and used a threshold of 0.01).