gTrack Documentation

Release 1

Marcin Imielienski, Jeremiah Wala

Sep 27, 2018

Contents

Contents:

CHAPTER 1

How to Graph CNV Data

```
opts_chunk$set(fig.width=8, fig.height=13)
```

```
#### make sure to load gUtils
library(gTrack)
#### load in SCNA data
#### these are level 3 segments from over 1600 TCGA breast cancer cases
#### downloaded from the TCGA and converted to a single GRanges object
scna = seg2gr(readRDS(gzcon(url("https://data.broadinstitute.org/snowman/gTrack/inst/
↪extdata/files"))))

#### we define amplification events and deletion events
#### and then do a simple recurrence analysis on amplifications

#### amplification events are defined using %Q% gUtils operator to filter
#### on scna metadata for segments with seg.mean greater than 1
#### greater than 50 markers and width less than 1MB
 amps = scna %Q% (seg.mean>1 & num.mark > 50 & width < 1e7)

#### apply a similar filter to define deletions
 dels = scna %Q% (seg.mean<(-1) & num.mark > 50 & width < 1e7)

#### compute the amplification "score" as the total number of amplification
#### events in a given region
 amp.score = as(coverage(amps), 'GRanges')

#### define the peaks of amplification as the 100 regions with
#### the highest amplification score
 amp.peaks = amp.score %Q% (rev(order(score))) %Q% (1:100)

#### "reduce" or merge the top peaks to find areas of recurrent amplification
amp.peaks = reduce(amp.peaks+1e5) %$% amp.peaks %Q% (rev(order(score)))

### do a similar analysis for dels
del.score = as(coverage(dels), 'GRanges')
```

(continues on next page)

(continued from previous page)

```
del.peaks = del.score %Q% (rev(order(score))) %Q% (1:100)
del.peaks = reduce(del.peaks+1e5) %$% del.peaks %Q% (rev(order(score)))

#### load in GRanges of GENCODE genes
genes = readRDS(system.file("extdata", 'genes.rds', package = "gTrack"))

#### use %$% operator to annotate merged amp and del peaks with "gene name" metadata
amp.peaks = amp.peaks %$% genes[, 'gene_name']
del.peaks = del.peaks %$% genes[, 'gene_name']

### now that we've computed scores and annotated peaks
### we want to inspect these peaks and plot them with gTrack

### load in precomputed gTrack of hg19 GENCODE annotation
### (note this is different from the GENCODE genes which is a GRanges
### we loaded in a previous line .. this is purely for visualization)
ge = track.gencode()
```

```
## Pulling gencode annotations from /data/research/mski_lab/Software/R/gTrack/extdata/
↪gencode.composite.collapsed.rds
```

```
#### build a gTrack of amps colored in red with black border
#### and one of dels colored in blue
gt.amps = gTrack(amps,  col = 'red', name = 'Amps')
gt.dels = gTrack(dels, col = 'blue', name = 'Dels')

#### build a gTrack of amp and del score as a line plot
gt.amp.score = gTrack(amp.score, y.field = 'score',
    col = 'red', name = 'Amp score', line = TRUE)
gt.del.score = gTrack(del.score, y.field = 'score',
    col = 'blue', name = 'Amp score', line = TRUE)

#### build a gTrack of peaks of amp and del peaks
gt.amp.peaks = gTrack(amp.peaks, gr.labelfield = 'gene_name',
    col = 'pink', border = 'black', name = 'Amp peaks', height = 5)
gt.del.peaks = gTrack(del.peaks, gr.labelfield = 'gene_name',
    col = 'lightblue', border = 'black', name = 'Amp peaks', height = 5)

### let's look at the top amplification peak
amp.peaks[1]
```

```
## GRanges object with 1 range and 2 metadata columns:
##       seqnames              ranges strand |     score
##          <Rle>           <IRanges>  <Rle> | <numeric>
##   [1]        8 [39254760, 39606122]      * |  253.9448
##                                                gene_name
##                                              <character>
##   [1] RP11-122L4.1, AC123767.1, CTD-2024D23.1, ADAM18, ADAM2
##   -------
##   seqinfo: 24 sequences from an unspecified genome
```

```
### interesting! this looks like a novel peak with genes that have
### not previously been associated with breast cancer
### ("RP11-122L4.1, AC123767.1, CTD-2024D23.1, ADAM18, ADAM2")
```

(continues on next page)

(continued from previous page)

```
### let's look at the data supporting this peak - including
### the underlying amp events, amp score, and peak region boundary
```

```
plot(c(ge, gt.amps, gt.amp.peaks, gt.amp.score), amp.peaks[1]+1e6)
```

```
### hmm, something looks suspicious since all the segments have the same
### start and end.  These could be copy number artifacts that often arise
### in segmentation of array data, sometimes due to germline copy number
### polymorphisms.


### to see this pattern more clearly, let's enlarge the
### amplification track, also add the deletion data, and replot
my.gt = c(ge, gt.dels, gt.del.peaks, gt.del.score,
            gt.amps, gt.amp.peaks, gt.amp.score)
```

```
plot(my.gt, amp.peaks[1]+1e6)
```

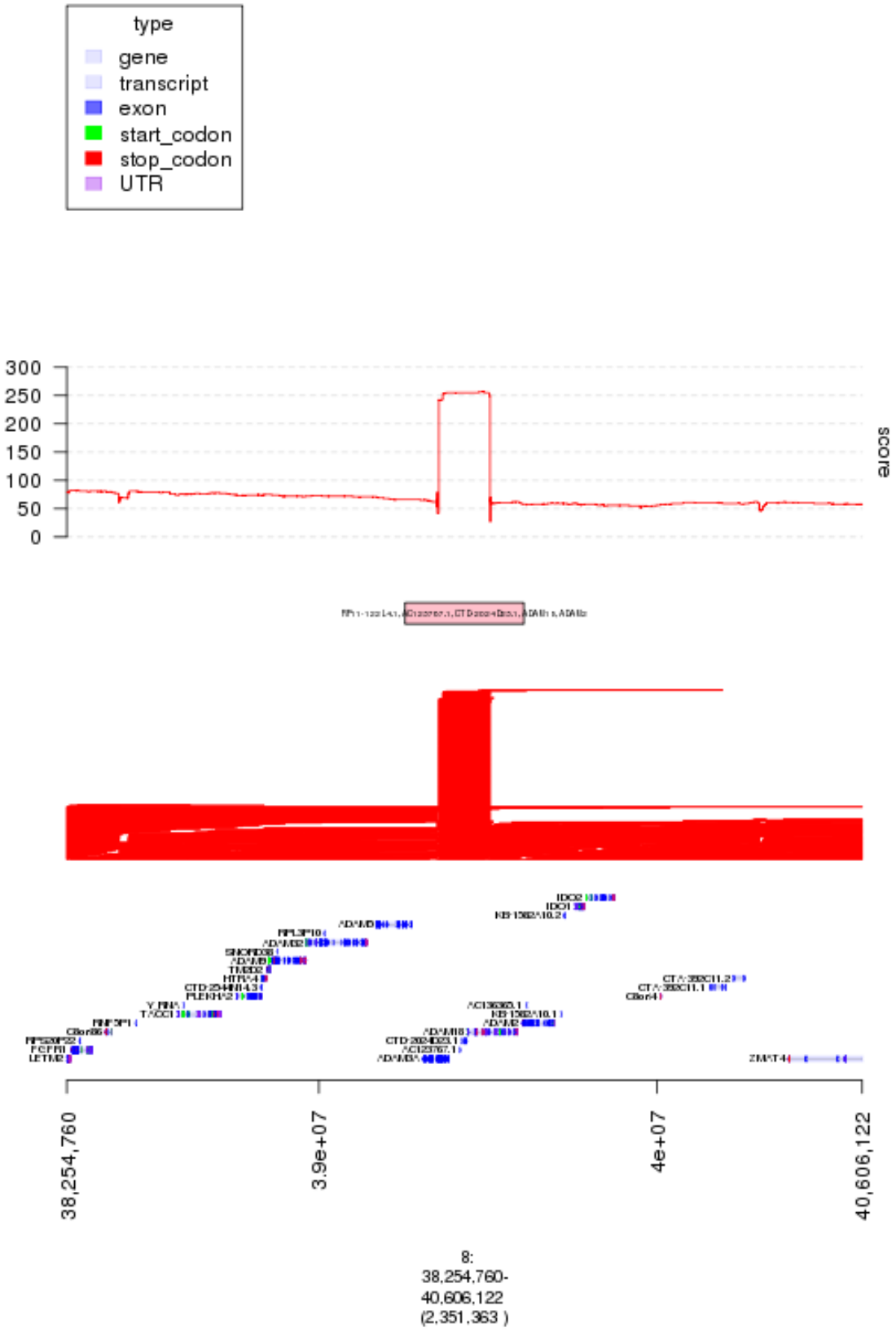
```
### interesting so this appears to also be a peak in the deletion analysis
### and a region that accumulates both amplification and deletion calls in
### many tumor samples.  This could either be a copy number polymorphism
### or an artifact.

### let's load in a track of copy events from the Database of Germline Variation
### which catalogues common copy changes in human populations
dgv = readRDS(system.file(c('extdata/files'), 'dgv.rds', package = 'gTrack'))
```

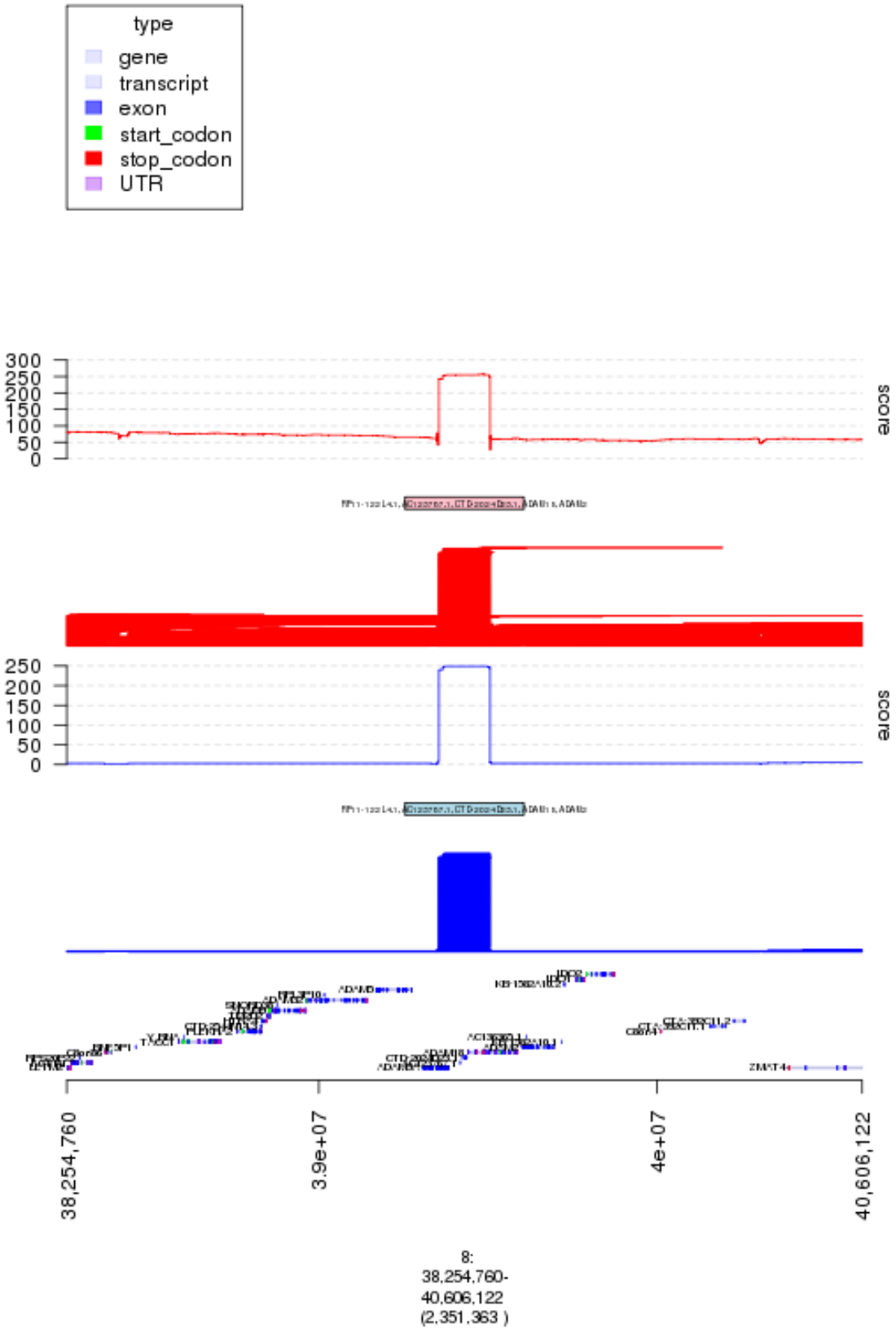
```
plot(c(ge, gt.amps, gt.amp.peaks, gt.amp.score), amp.peaks[1]+1e6)
```

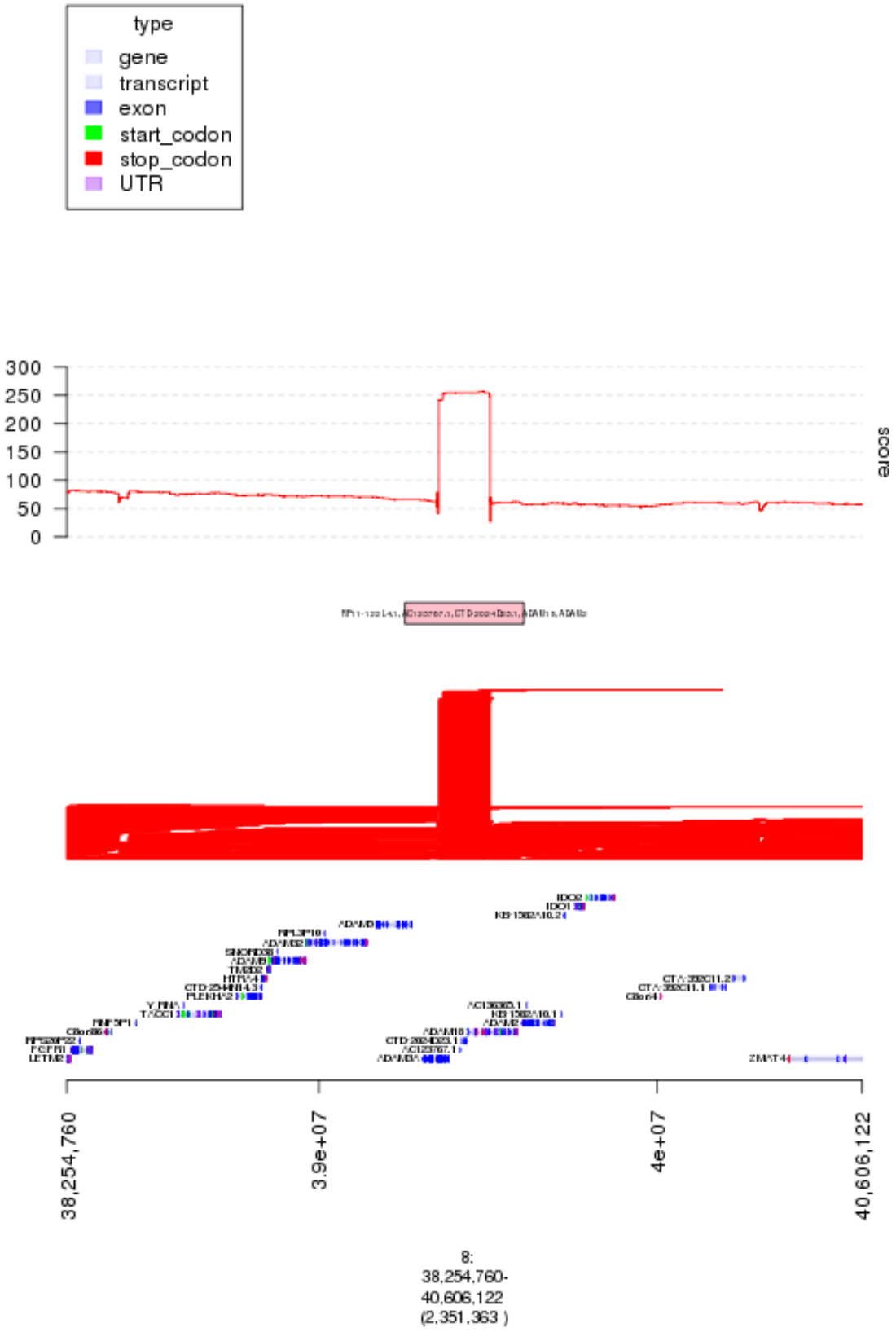
```
### indeed looks like this is a region around which people have previously
### seen germline copy number variations, so it's likely an artifact

### let's look at the next amp peak
print(amp.peaks[2])
```

```
## GRanges object with 1 range and 2 metadata columns:
##       seqnames               ranges strand |     score
##          <Rle>            <IRanges>  <Rle> | <numeric>
##   [1]       11 [68809874, 69577804]      * |  102.4002
##
↪                                                                  gene_name
##
↪                                                                <character>
##   [1] TPCN2, MIR3164, RP11-554A11.7, RP11-554A11.8, MYEOV, RP11-211G23.2, RP11-
↪211G23.1, AP000439.1, AP000439.2, AP000439.5, AP000439.3, CCND1, ORAOV1, FGF19
##   -------
##   seqinfo: 24 sequences from an unspecified genome
```

```
### this peak includes  CCND1 in addition to other genes
### this peak is known to be a target of amplification in breast cancer
### and so likely real

### let's plot it:
```


type
gene
transcript
exon
start_codon
stop_codon
UTR
300
250
200
150
100
50
0
score
38,254,760
3.9e+07
4e+07
40,606,122
8:
38,254,760-
40,606,122
(2,351,363)


type
gene
transcript
exon
start_codon
stop_codon
UTR
300
250
200
150
100
50
0
score
250
200
150
100
50
0
score
38,254,760
3.9e+07
4e+07
40,606,122
8:
38,254,760-
40,606,122
(2,351,363)


type
gene
transcript
exon
start_codon
stop_codon
UTR
300
250
200
150
100
50
0
score
38,254,760
3.9e+07
4e+07
40,606,122
8:
38,254,760-
40,606,122
(2,351,363)

```
plot(my.gt, amp.peaks[2]+1e6)
```

```
## budget ..
```

```
## Error in (function (...) : all elements in '...' must be GRanges objects
```

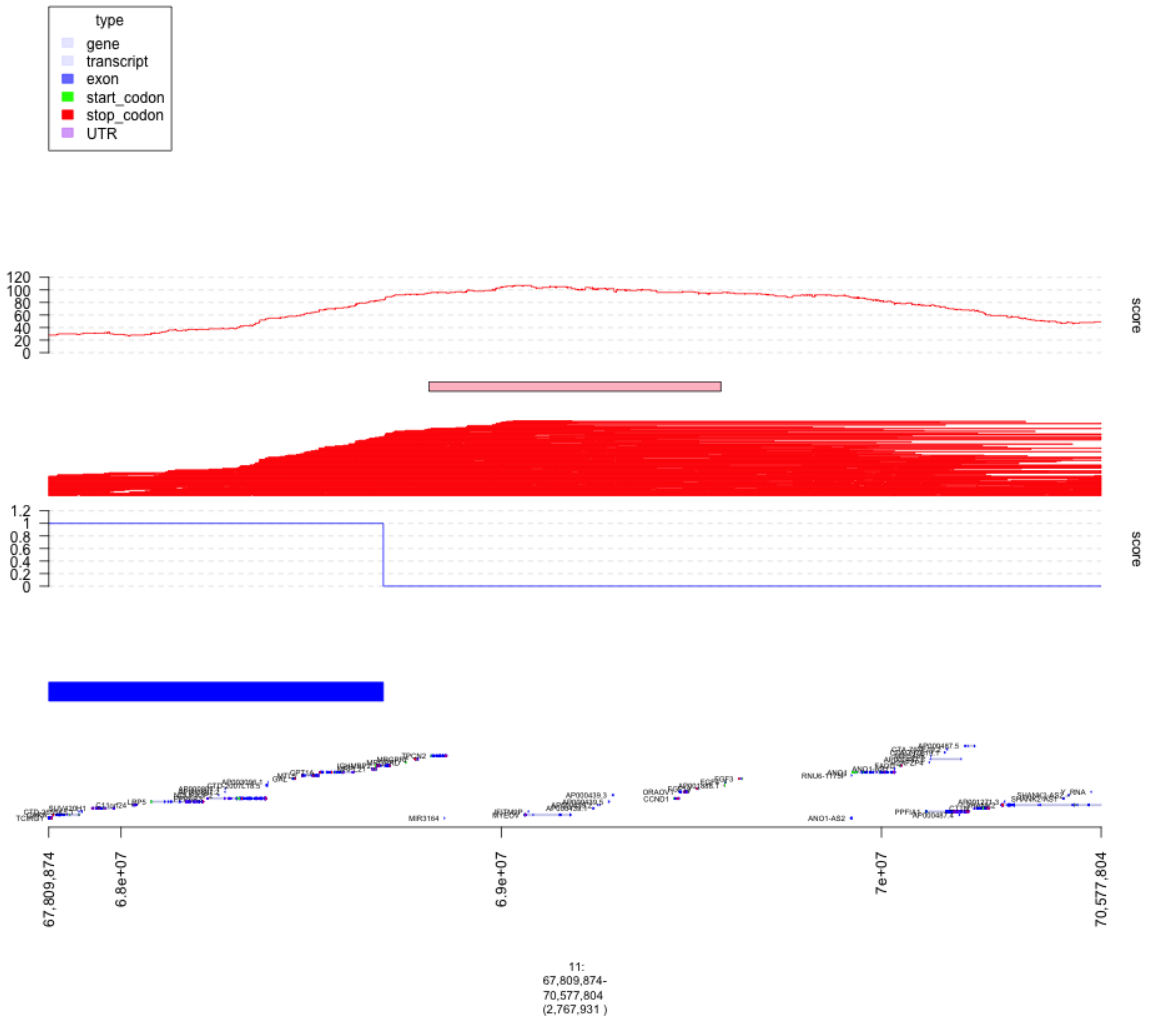


Fig. 4: plot of chunk plot4

```
### unlike the previous peak this has an enrichment of amplifications vs deletions
### not known have a bunch of germline copy number changes in the DGV

### let's zoom in on the individual events, getting rid of the other tracks
```

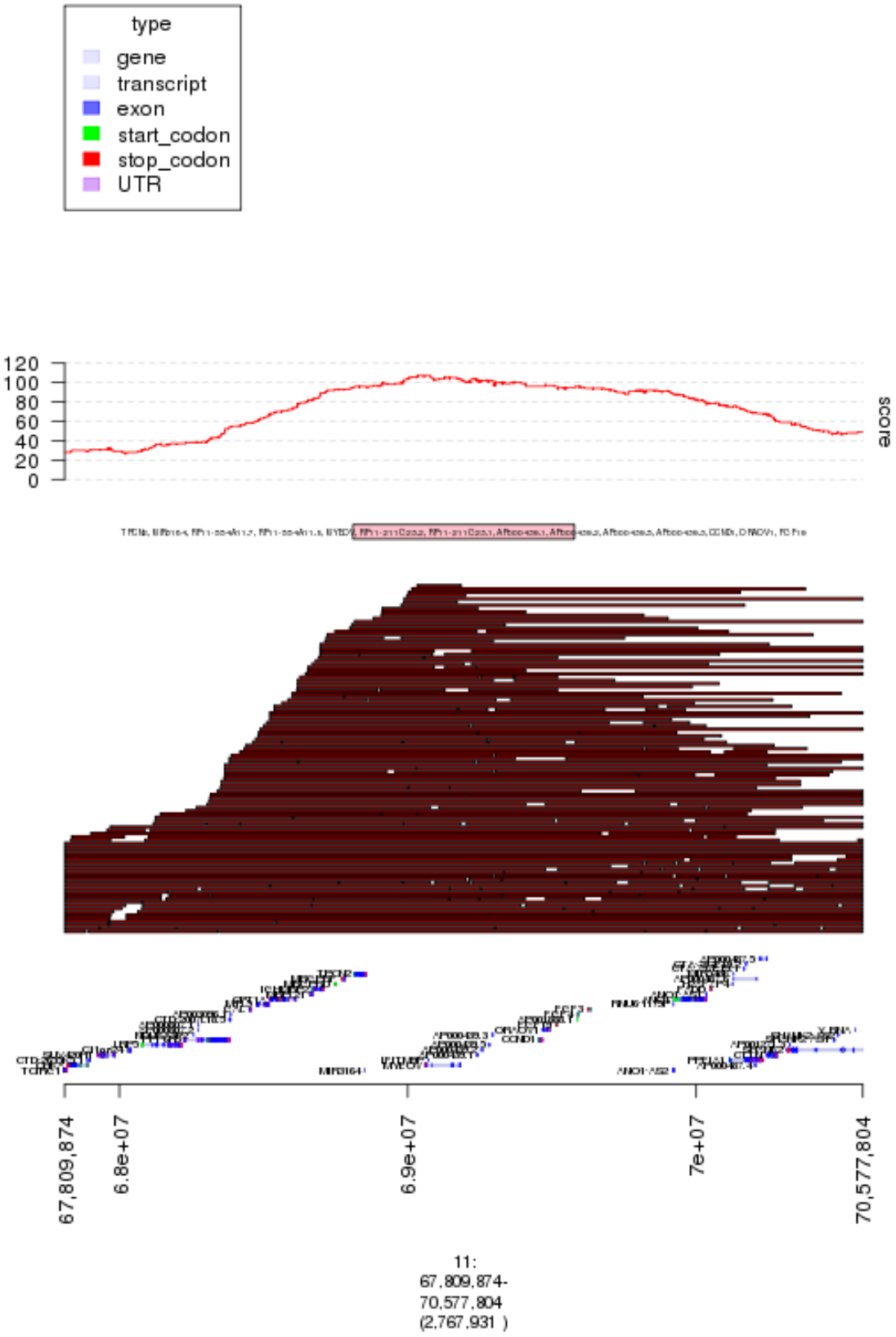
(continues on next page)

(continued from previous page)

```
### increase the height of the amp track
### and adding a black border to better define event boundaries
gt.amps$border = 'black'
gt.amps$height = 30
my.gt = c(ge, gt.amps, gt.amp.peaks, gt.amp.score)
```

```
plot(my.gt, amp.peaks[2]+1e6)
```

```
### here each red segment is a somatic amplification or gain in a different patietn
### the peak looks real, in that the events have relatively random starts
### and ends and cluster around this target gene.
```


type
gene
transcript
exon
start_codon
stop_codon
UTR
120
100
80
60
40
20
0
score
67,809,874
6.8e+07
6.9e+07
7e+07
70,577,804
11:
67,809,874-
70,577,804
(2,767,931)

CHAPTER 2

Customizing a Small Data Set

In this vignette, examples of how to segment a data set such as a single GRanges object, how to specify the y-axis of a graph, how to color that same graph, how to add a color to each unique value will be shown.

2.1 gr.tile(gr , w) - Divide GRanges into tiles of length "w"

```
## DO NOT FORGET TO LOAD gUtils library.
library(gUtils)

#The only interval in this GRanges object has a range of length 100, it'll be divided by 5 and thus, 20 tiles of length 5 will be returned.
gr <- gr.tile(GRanges(1, IRanges(1,100)), w=5)
```

```
## Plot tiles
plot(gTrack(gr))
```

2.2 gTrack(gr + n) - Extend each range by "n" base pairs

```
plot(gTrack(gr+5))
```

2.3 stack.gap - Specify degree of spacing(in x-direction) between ADJACENT tiles.

```
gr <- GRanges(seqnames = Rle(c("chr1" , "chr2" , "chr1" , "chr3") ,
  c(1,3,2,4)), ranges = IRanges(c(1,3,5,7,9,11,13,15,17,19) , end =
    c(2,4,6,8,10,12,14,16,18,20), names = head(letters,10)), GC=seq(1,10,length=10), name=seq(5,10,length=10))
```

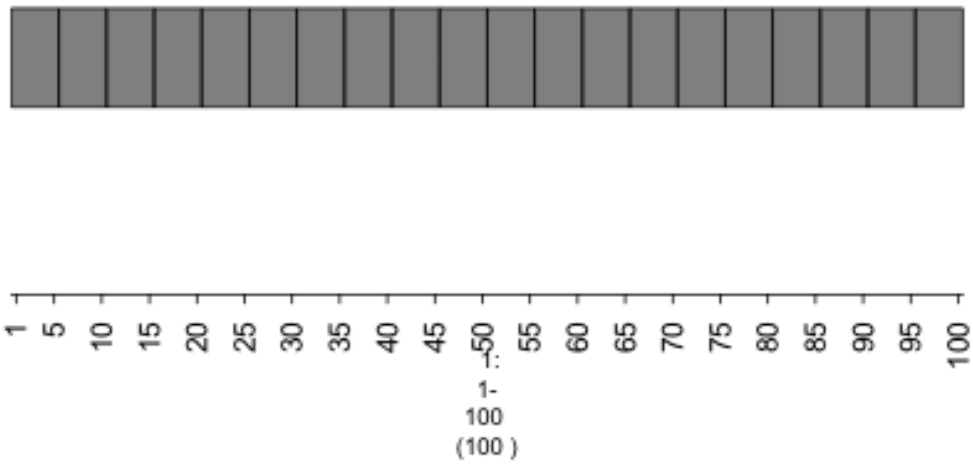


Fig. 1: plot of chunk plot-tiles

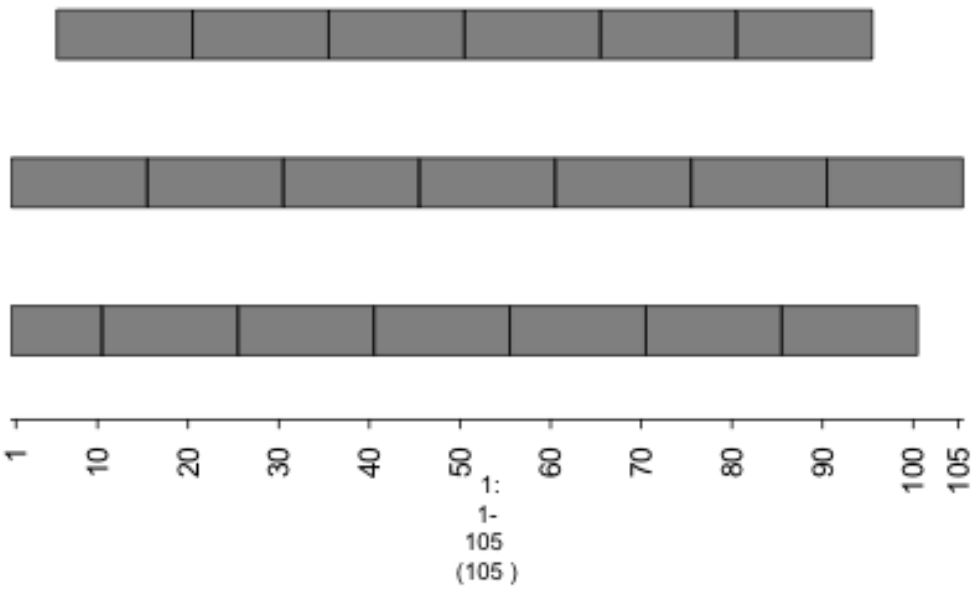


Fig. 2: plot of chunk plot-overlappingtiles

```
plot(gTrack(gr))
```

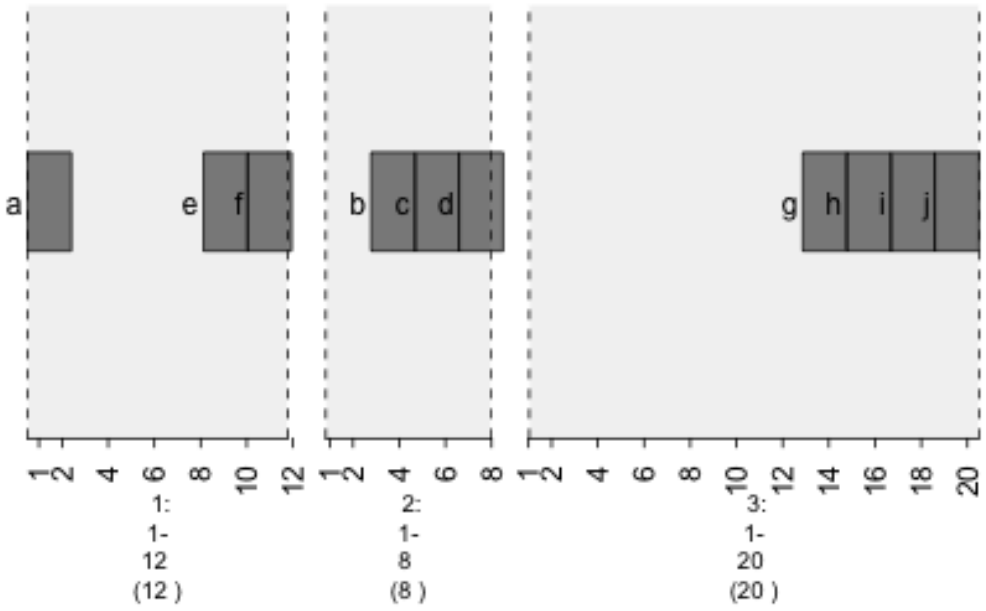


Fig. 3: plot of chunk plot-gr

```
plot(gTrack(gr , stack.gap = 2))
```

```
plot(gTrack(gr , stack.gap = 3))
```

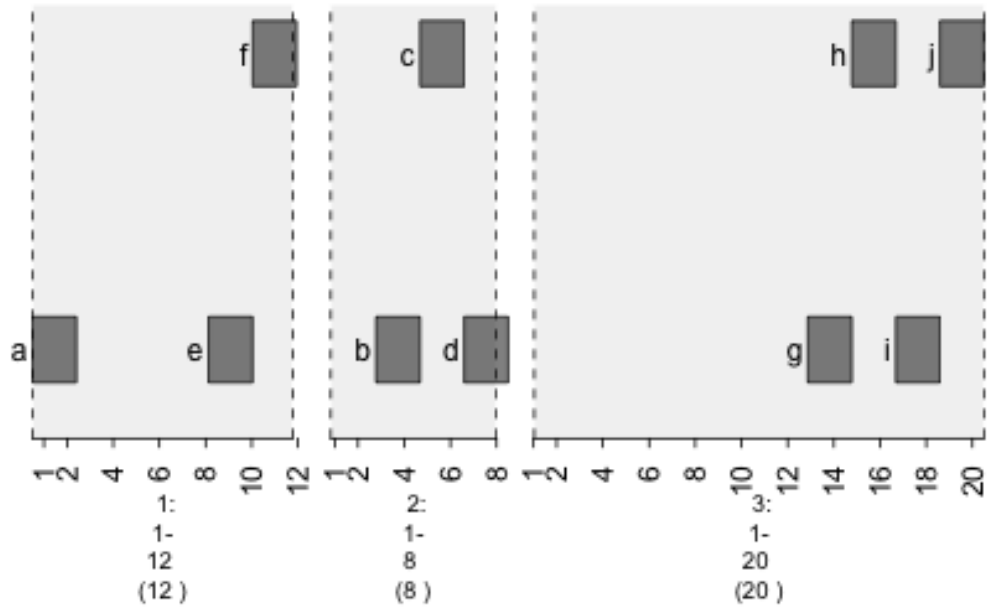


Fig. 4: plot of chunk plot-stack.gap2

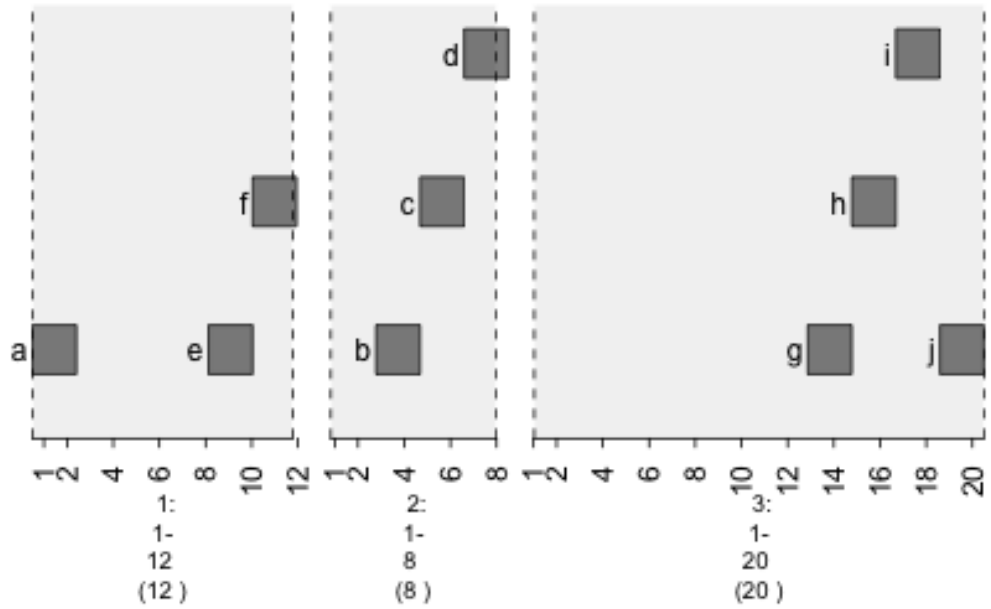


Fig. 5: plot of chunk plot-stack.gap3

2.4 y.field - Specify y-axis of graph

```
plot(gTrack(gr , y.field = 'GC'))
```

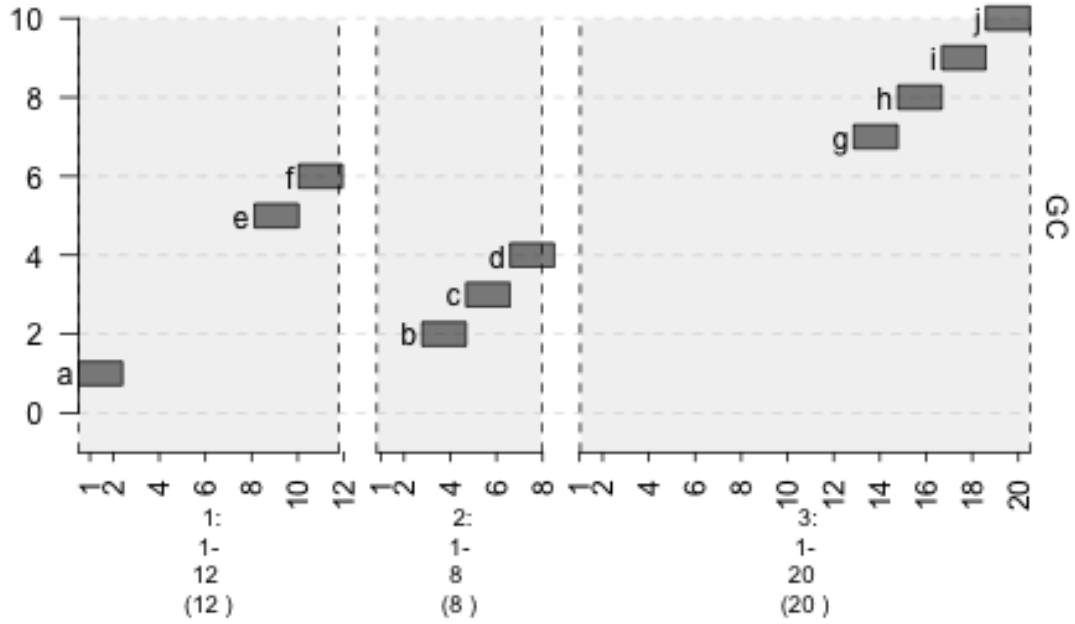


Fig. 6: plot of chunk plot-y.fieldGC

2.5 bars - Plot data points as vertical bars

gTrack(gr , bars = TRUE/FALSE)

```
plot(gTrack(gr , y.field = 'GC' , bars = TRUE , col = 'light blue'))
```

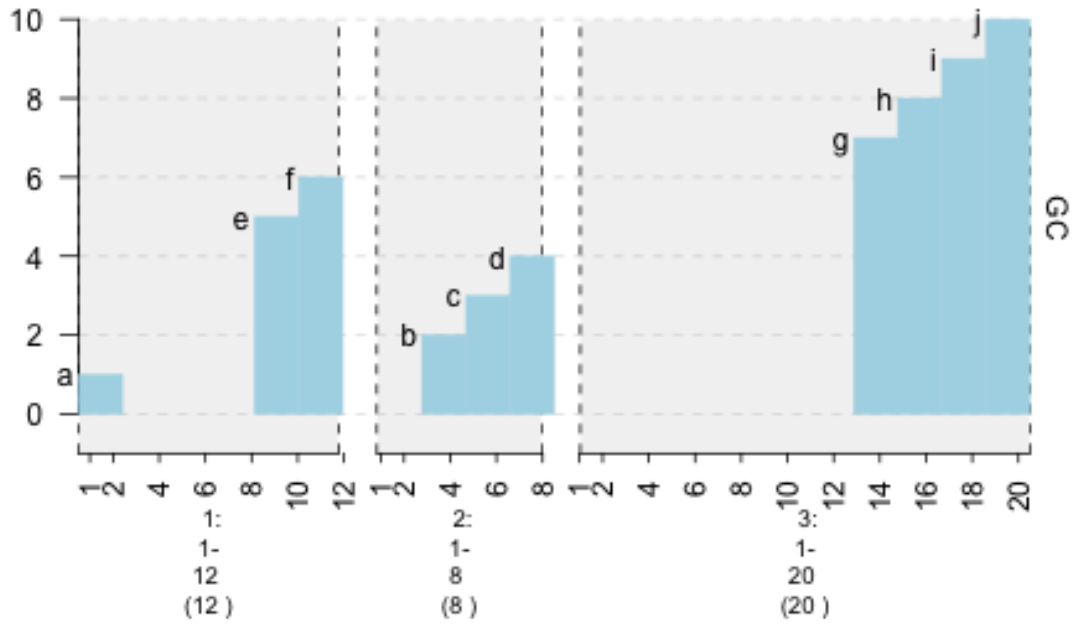


Fig. 7: plot of chunk plot-bars

2.6 lines - Plot data points as lines.

gTrack(gr , lines = TRUE/FALSE)

```
plot(gTrack(gr , y.field = 'GC' , lines = TRUE , col = 'purple'))
```

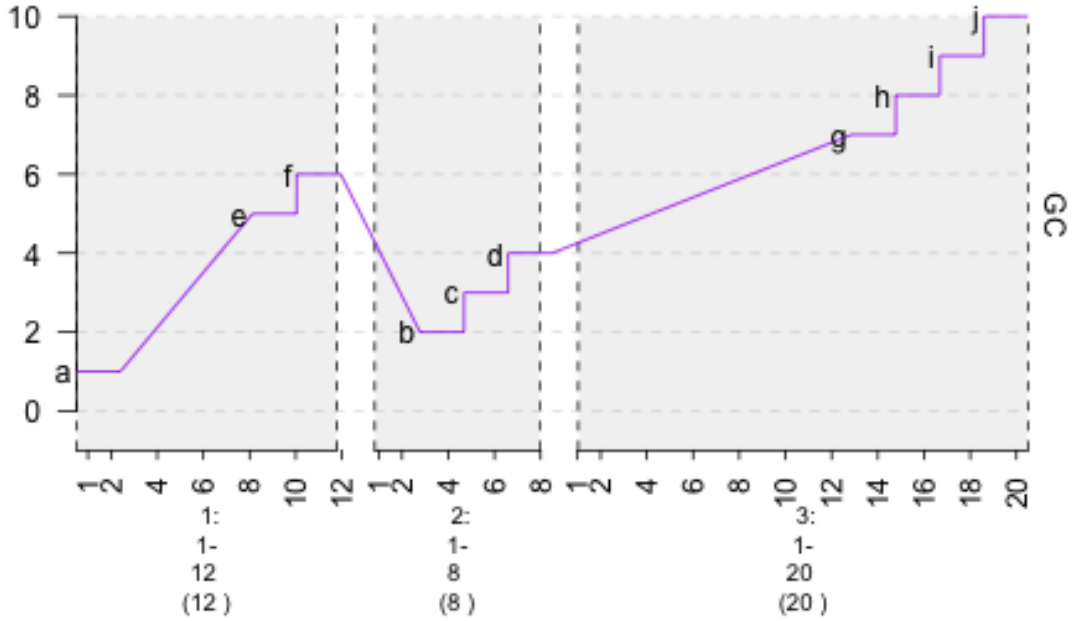


Fig. 8: plot of chunk plot-lines

2.7 circles - Plot data points as circles.

gTrack(gr , circles = TRUE/FALSE)

```
plot(gTrack(gr , y.field = 'GC' , circles = TRUE , col = 'magenta' , border = '60'))
```

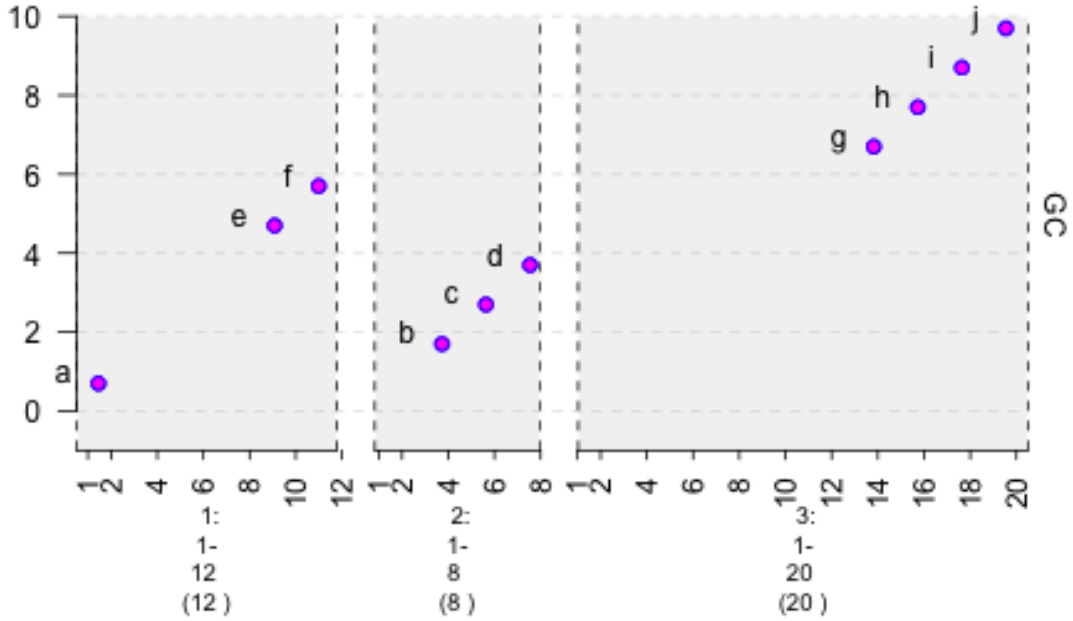


Fig. 9: plot of chunk plot-circles

2.8 colormap - Specify mapping of colors to values.

```
plot(gTrack(gr , y.field = 'GC' , bars = TRUE , col = NA , colormaps = list(GC = c("1
↪"="red" , "2" = "blue" , "3"="magenta", "4"="light blue" ,"5"="black" , "6"="green",
↪ "7"="brown" , "8"="pink", "9"="yellow", "10" = "orange")) ))
```

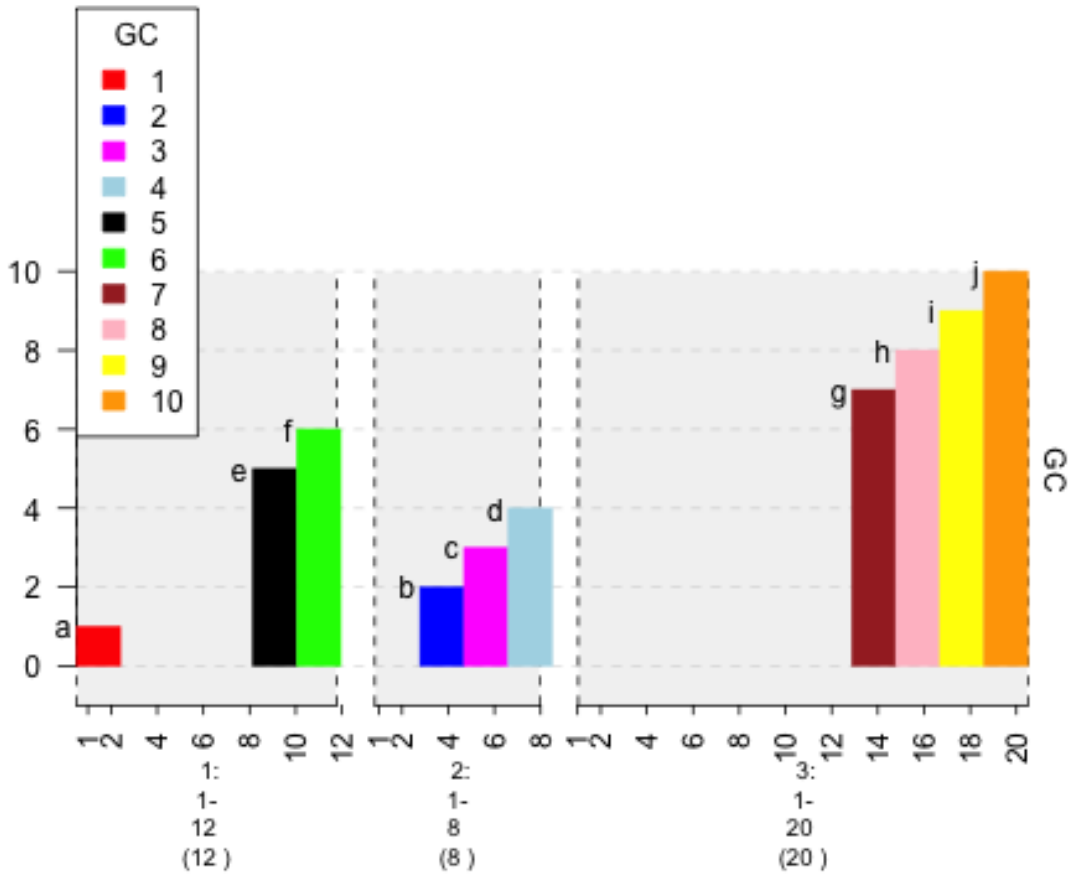


Fig. 10: plot of chunk plot-colormap

2.9 gr.colorfield - Automatically specify mapping of colors to values.

```
plot(gTrack(gr , y.field = 'GC' , bars = TRUE , col = NA , gr.colorfield = 'GC'))
```

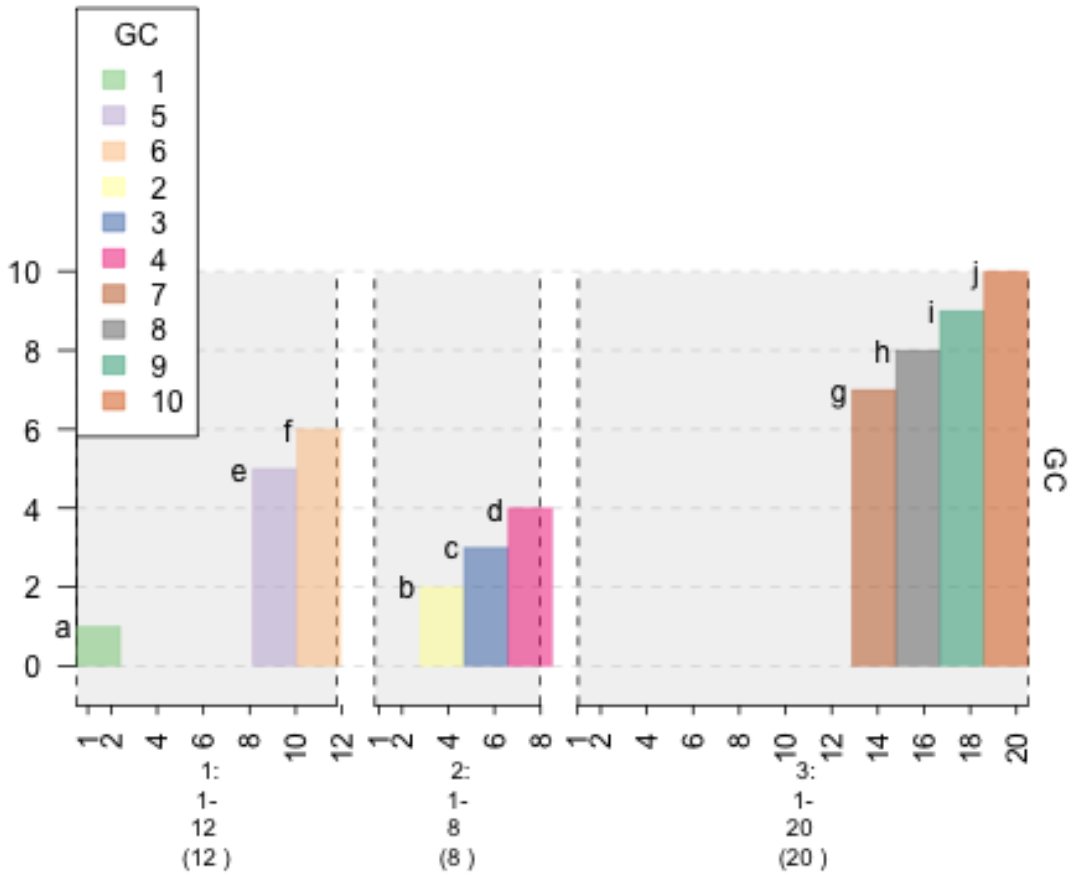


Fig. 11: plot of chunk plot-gr.colorfield

2.10 gr.labelfield - Plot values for each data point.

```
plot(gTrack(gr , y.field = 'GC' , bars = TRUE , col = NA , gr.colorfield = 'GC' , gr.
↪labelfield = 'name'))
```

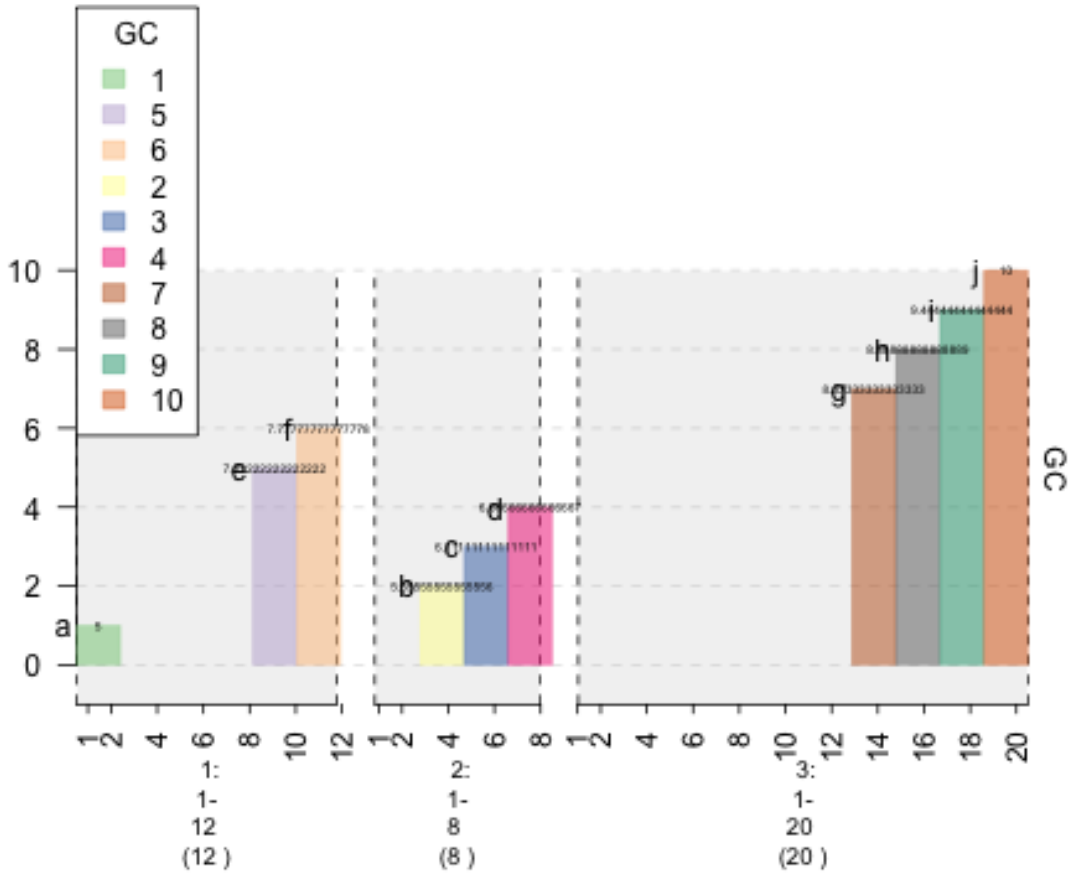


Fig. 12: plot of chunk plot-labelfield

CHAPTER 3

How to Create Graphs

During chromosomal rearrangements, translocations may occur and graphing such a phenomenon is possible with gTrack. However, the **edges** parameter must be used. This vignette will explain how to prepare a graph.

3.1 Edges Parameter

In order to create a connected graph in gTrack, the **edges** parameter of gTrack must be supplied a matrix or a data frame of connections.

```
##create a GRanges object storing 10 sequences. These sequences will serve as nodes␣
↪for the graph.
gr <- GRanges(seqnames = Rle(c("chr1" , "chr2" , "chr1" , "chr3") ,
  c(1,3,2,4)), ranges = IRanges(c(1,3,5,7,9,11,13,15,17,19) ,
  end = c(2,4,6,8,10,12,14,16,18,20),
  names = head(letters,10)),
  GC=seq(1,10,length=10),
  name=seq(5,10,length=10))

gr
```

```
## GRanges object with 10 ranges and 2 metadata columns:
##     seqnames    ranges strand |        GC             name
##        <Rle> <IRanges>  <Rle> | <numeric>        <numeric>
##   a     chr1   [ 1,  2]      * |         1                5
##   b     chr2   [ 3,  4]      * |         2 5.55555555555556
##   c     chr2   [ 5,  6]      * |         3 6.11111111111111
##   d     chr2   [ 7,  8]      * |         4 6.66666666666667
##   e     chr1   [ 9, 10]      * |         5 7.22222222222222
##   f     chr1   [11, 12]      * |         6 7.77777777777778
##   g     chr3   [13, 14]      * |         7 8.33333333333333
##   h     chr3   [15, 16]      * |         8 8.88888888888889
##   i     chr3   [17, 18]      * |         9 9.44444444444444
```

(continues on next page)

(continued from previous page)

```
##   j     chr3  [19, 20]      * |        10               10
##   -------
##   seqinfo: 3 sequences from an unspecified genome; no seqlengths
```

```
## Specify links between nodes using a matrix. Numeric 1s refer to a connection while␣
↪conversely with 0s.

##create an N*N matrix filled with 0s.
graph = matrix(0 , nrow = 10 , ncol = 10)

##set certain indices to 1.
graph[1,3]=1
graph[1,10]=1
graph[2,5]=1
graph[2,8]=1
graph[3,5]=1
graph[4,1]=1
graph[4,2]=1
graph[4,6]=1
graph[4,9]=1
graph[5,1]=1
graph[5,2]=1
graph[5,4]=1
graph[8,1]=1
graph[8,2]=1
graph[9,1]=1
graph[10,1]=1
```

```
##use edges parameter to create graph.
plot(gTrack(gr , edges = graph , stack.gap = 5))
```

3.2 col Column

If a **matrix** is used to create a graph, color and style of edges cannot be specified.Instead of using a **matrix**, a data frame can be used to specify those attributes.

```
##the "from" column specifies the beginning node (range).
##the "to" column specifies the end node (range).
##the "col" specifies the color of the edge.
graph = data.frame(from = 1:9, to = c(6,9,7,2,4,10,8,5,3) , col = c('red', 'blue',
↪'green'))
```

```
plot(gTrack(gr , edges = graph , stack.gap = 5))
```

3.3 lwd Column

To change the width of the edges, use the **lwd** parameter.

```
##the "lwd" column specifies the width of the edge.
 graph$lwd = 1.844941
 graph
```

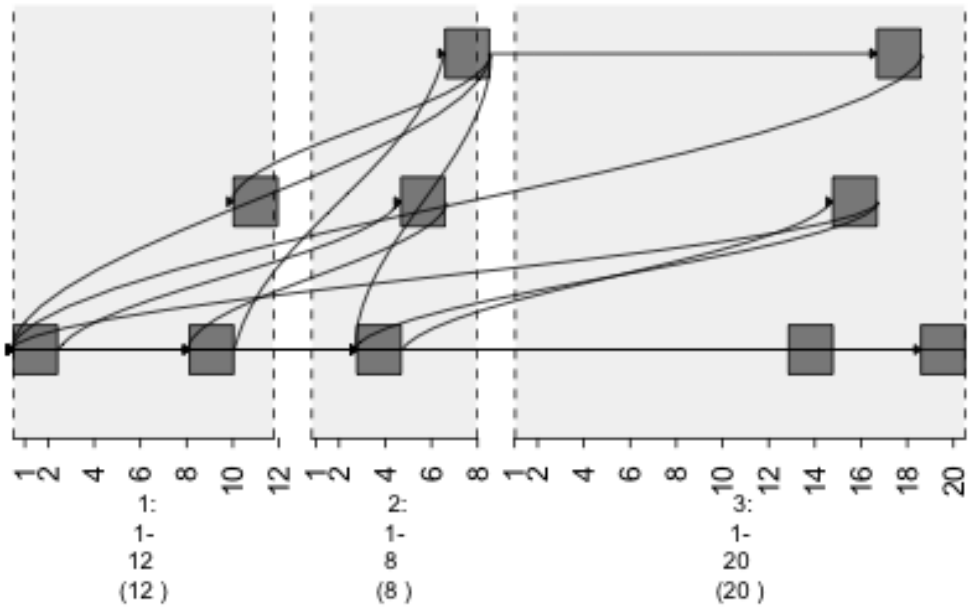


Fig. 1: plot of chunk plot1

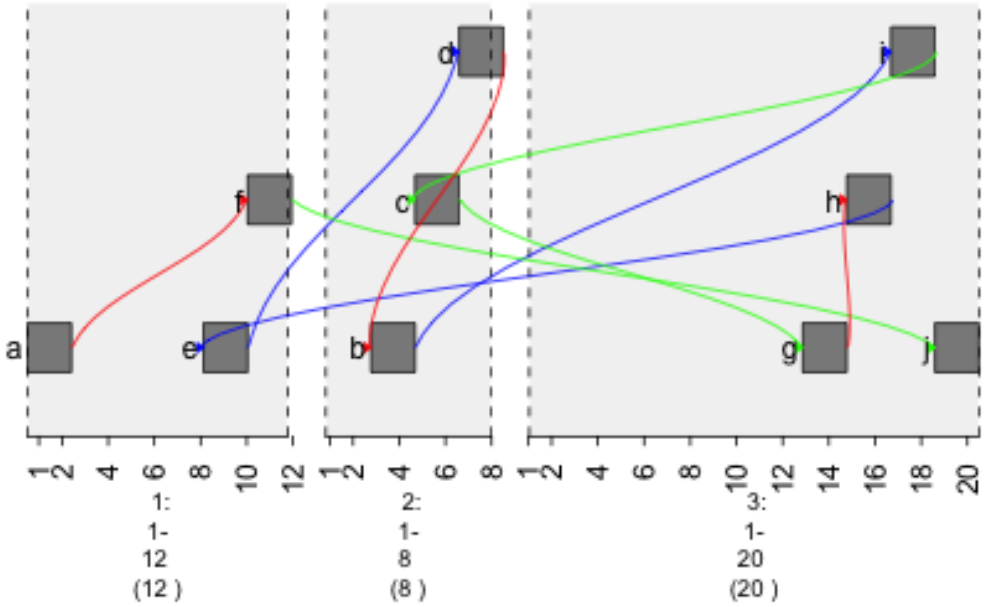


Fig. 2: plot of chunk colored-graph

```
##   from to   col      lwd
## 1    1  6   red 1.844941
## 2    2  9  blue 1.844941
## 3    3  7 green 1.844941
## 4    4  2   red 1.844941
## 5    5  4  blue 1.844941
## 6    6 10 green 1.844941
## 7    7  8   red 1.844941
## 8    8  5  blue 1.844941
## 9    9  3 green 1.844941
```

```
plot(gTrack(gr, edges = graph, stack.gap = 5))
```

3.4 lty Column

Change style of edge by **lty** parameter.

```
## lty specifies the style of the edge (no dashes, big dashes, little dashes)
graph$lty = c(1,2,3)
```

```
plot(gTrack(gr , edges = graph , stack.gap = 5))
```

3.5 h Column

Increase **"curviness"** of the edges by adding **h** column.

```
graph$h = 10
```

```
plot(gTrack(gr , edges = graph , stack.gap = 5))
```

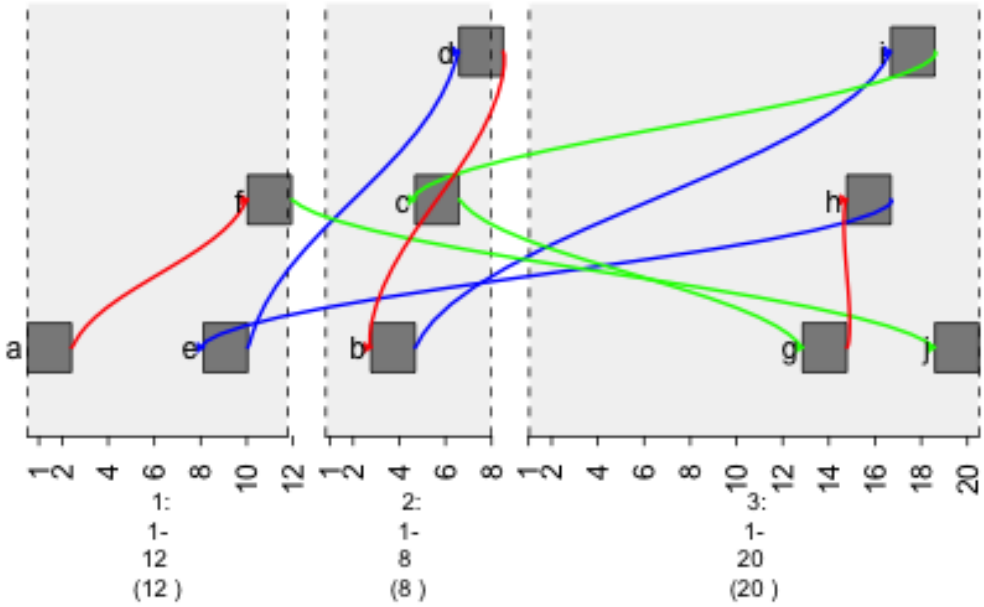


Fig. 3: plot of chunk width-graph

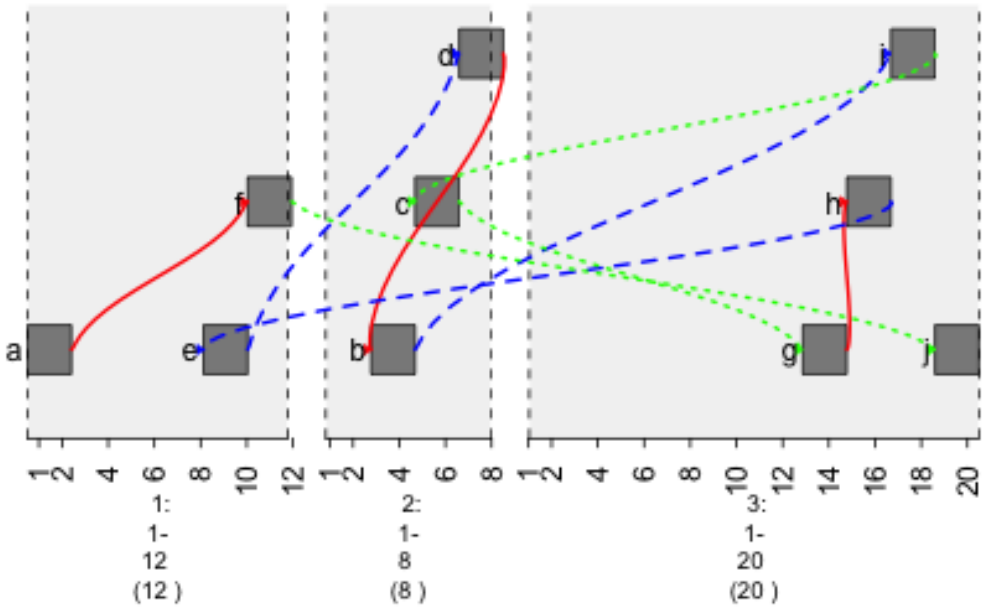


Fig. 4: plot of chunk style-graph

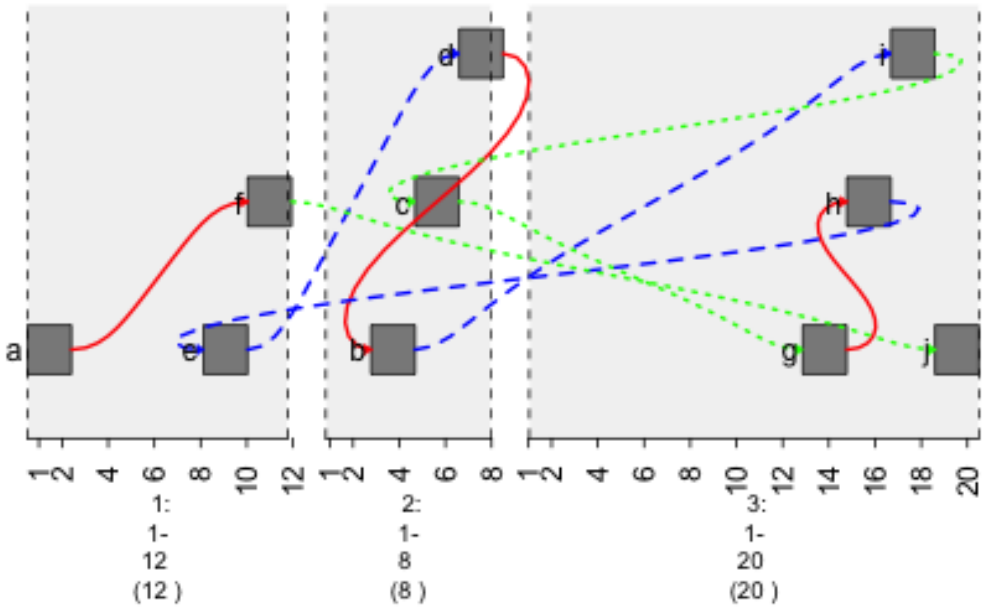


Fig. 5: plot of chunk curviness-graph

CHAPTER 4

How to Create Heat Maps

gTrack is able to create heat graphs and in this vignette, the steps to make one will be shown. In order to create one the **mdata** parameter of gTrack must be supplied a matrix.

4.1 mdata Parameter

```
## DO NOT FORGET TO LOAD gUtils library.
library(gUtils)


## In order to create a heat map for each node in the matrix, color intensity needs␣
↪to be specified.
## To illustrate, a random N*N matrix filled with values from 1:100 is used, but␣
↪first a GRanges object is made.

##create an N lengthed GRanges object that you want to have connections.
gr <- GRanges(seqnames = Rle(c("chr1" , "chr2" , "chr1" , "chr3") ,c(1,3,2,4)),␣
↪ranges = IRanges(c(1,3,5,7,9,11,13,15,17,19) ,end = c(2,4,6,8,10,12,14,16,18,20),␣
↪names = head(letters,10)),GC=seq(1,10,length=10), name=seq(5,10,length=10))
heatMap = matrix(runif(length(gr)^2), nrow = 10, ncol = 10)
```

```
plot(gTrack(gr, mdata = heatMap, stack.gap = 5))
```

```
## It is also possible to add multiple plots to the same window. Use the␣
↪concatenation operator.
 plot(c(gTrack(gr, edges = graph, stack.gap = 5), gTrack(gr, mdata = heatMap, stack.
↪gap = 5)))
```

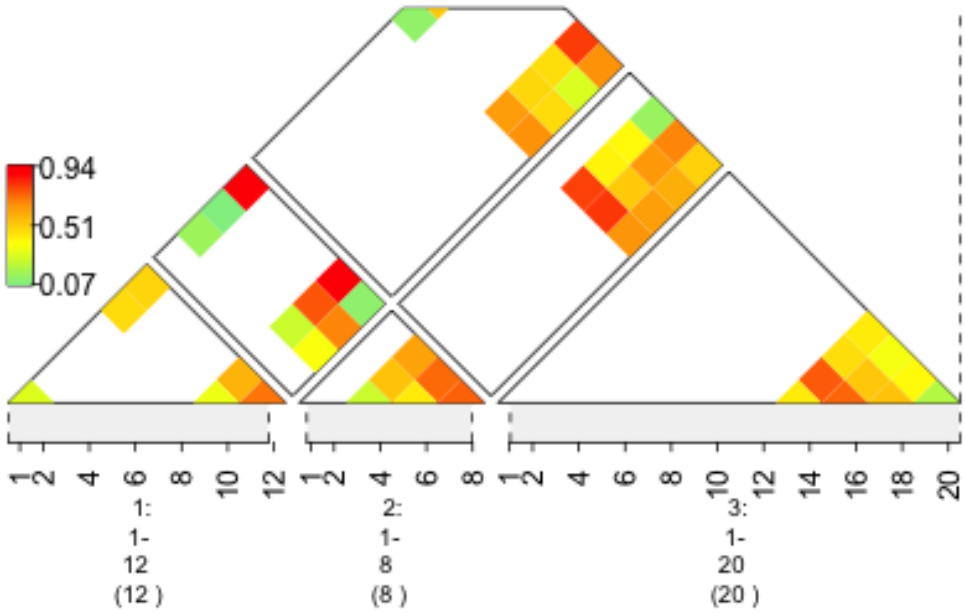


Fig. 1: plot of chunk plotheatmap

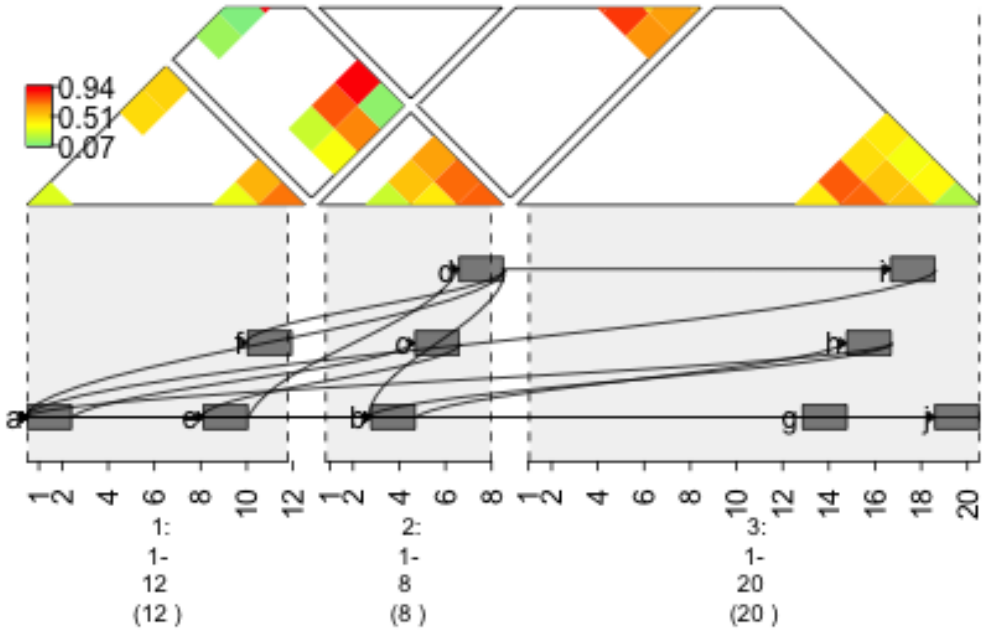


Fig. 2: plot of chunk plotheatmap

4.2 Zooming In and Out of a Graph

```
## In order to zoom in and out, essentially the ranges of the sequences need to be␣
↪stretched or shrinked.
## Firstly, in upstream code, the "seqnames" field of the GRanges object was supplied␣
↪character vectors. The problem that arises is that the seqlengths of the GRanges␣
↪object cannot be implicitly determined.
## gUtils can easily fix a GRanges object so that the seqlengths is properly set.
## Use the gr.fix function. It will find the largest coordinate for each seqname and␣
↪subsequently save those values in the seqlengths parameter.
 gr <- gr.fix(gr)

## create the window for the plot.
 si = si2gr(seqinfo(gr.fix(gr)))
```

```
plot(c(gTrack(gr , edges = graph, stack.gap = 5) , gTrack(gr , mdata = heatMap, stack.
↪gap = 5)) , gr.sub(si , 'chr' , '' )+20)
```

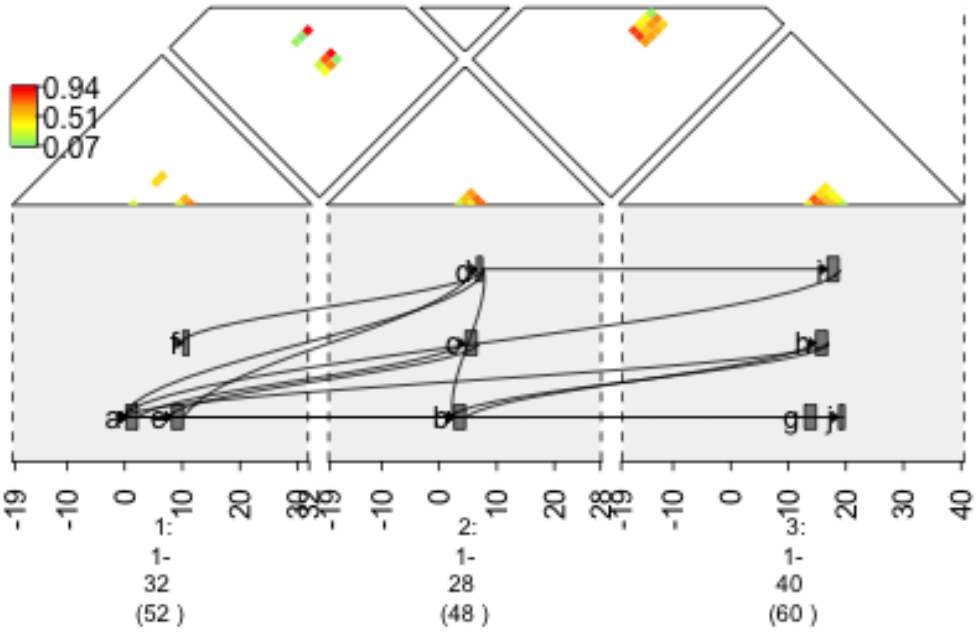


Fig. 3: plot of chunk zoomedOutGraph

CHAPTER 5

How To Graph Relationships In The Genome

Genes interact with each other and gTrack is capable of graphing them.

In this vignette, the **draw.paths** and **circle** parameters of gTrack will aide in illustrating gene interactions. Specifically, they will be used in graphing variants in random sequences.

5.1 Using Draw.paths Parameter

To prepare a data set that illustrates the draw.paths parameter, a GRangesList storing RANDOM sequences in chromosomes 1,2, and 3 is created. Then, two graphs, one with and one without the draw.paths parameter will be made. The difference in the two show the affect the draw.paths parameter has on graphs.

```
gene1 = sort(sample(gUtils::gr.tile(gUtils::parse.gr('1:1-5e3+'), 50), 5))
gene2 = rev(sort(sample(gUtils::gr.tile(gUtils::parse.gr('2:1-5e3-'), 50), 12)))
gene3 = sort(sample(gUtils::gr.tile(gUtils::parse.gr('3:1-5e3+'), 50), 8))

##Create a column that keeps a counter of the exon number.

gene1$exon = 1:length(gene1)
gene2$exon = 1:length(gene2)
gene3$exon = 1:length(gene3)

## Combine into GRangesList
grl = GRangesList(gene1 = gene1, gene2 = gene2, gene3 = gene3)

gt.genes = gTrack(grl)

## Plot two graphs, one with and one without the draw.paths parameter.
fusion = GRangesList(c(grl$gene1[1:3], grl$gene2[5:9], grl$gene3[7:8]))
gt.fusion = gTrack(fusion, draw.paths = FALSE)
gt.fusion.o = gTrack(fusion, draw.paths = TRUE)

## separating the windows for the graph.
win = gUtils::parse.gr(c('1:1-1e4', '2:1-1e4', '3:1-1e4'))
```

```
plot(c(gt.genes, gt.fusion, gt.fusion.o), win +1e3)
```

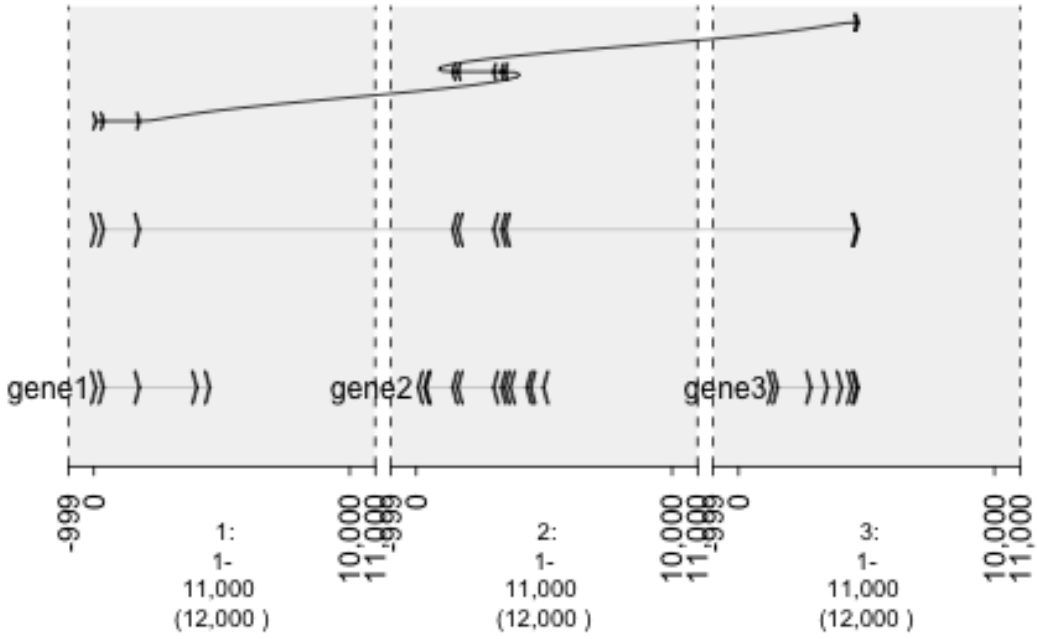


Fig. 1: plot of chunk -plotList

CHAPTER 6

Graphing Point Mutations

To illustrate gTrack's functionality in graphing point mutations, a data set of sequences is created and a few of them will are picked as variants. This data will be graphed and because there are outliers (variants), they will be easily visable. This vignette also exemplifies how/when to use the gTrack **name** parameter.

6.1 name Parameter

```
## create sequences from chromosomes 1-3.
fake.genome = c('1'=1e4, '2'=1e3, '3'=5e3)
tiles = gr.tile(fake.genome, 1)

## Choose 5 random indices. These indices will store the variants.
hotspots = sample(length(tiles), 5)

## for each sequence, calculate the shortest distance to one of the hotspots.
d = pmin(Inf, values(distanceToNearest(tiles, tiles[hotspots]))$distance, na.rm =␣
↪TRUE)
## for sequences near the hotspots, the "prob" will be a higher positive number. It␣
↪becomes smaller as it moves farther from the hotspot.
prob = .05 + exp(-d^2/10000)
```

```
## sample 2000 of the sequences. the one nearer to the hotspots will "probably" be␣
↪selected.
mut = sample(tiles, 2000, prob = prob, replace = TRUE)
```

```
## Error in sample.int(length(x), size, replace, prob): incorrect number of␣
↪probabilities
```

```
## graph with different degrees of stack.gap. The higher numeric supplied to stack.
↪gap helps separate the data, visually.
gt.mut0 = gTrack(mut, circle = TRUE, stack.gap = 0, name = "Track 0")
```

(continues on next page)

(continued from previous page)

```
gt.mut2 = gTrack(mut, circle = TRUE, stack.gap = 2, name = "Track 2")
gt.mut10 = gTrack(mut, circle = TRUE, stack.gap = 10, name = "Track 10")
gt.mut50 = gTrack(mut, circle = TRUE, stack.gap = 50, name = "Track 50")
```

```
win = si2gr(fake.genome)
plot(c(gt.mut0, gt.mut2, gt.mut10, gt.mut50), win)
```

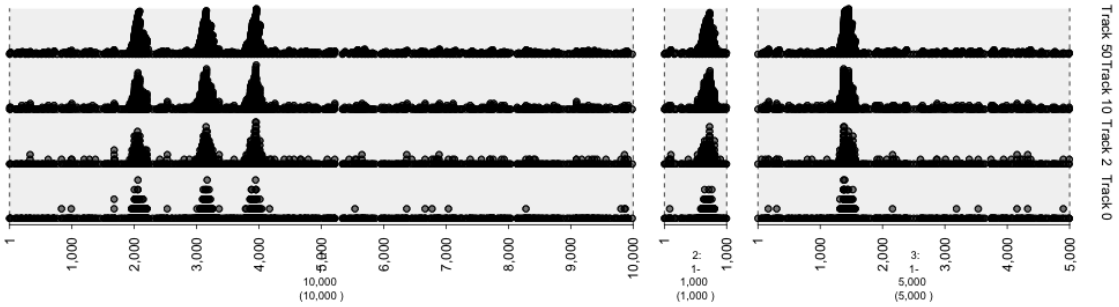


Fig. 1: plot of chunk mutations2-plot

CHAPTER 7

How to Graph Structural Variations

```
## make sure to load in these libraries
library(gTrack)
library(bamUtils) ## to use read.bam function
```

```
options(warn=-1)
## this load sequences that have had coverage calculated from cancer cell lines␣
↪(GRanges object, have to make into a gTrack)
cov = readRDS(gzcon(url('https://data.broadinstitute.org/snowman/gTrack/inst/extdata/
↪coverage.rds')))

## wrap a gTrack around this, draw with blue circles, and label the track "Cov" and␣
↪sets 0 as lower bound for all views
gt.cov = gTrack(cov, y.field = 'mean', circles = TRUE, col = 'blue', name = 'Cov')

## this loads the junctions data from the cell line (GRangesList, where each item is␣
↪a length 2 GRanges
## with strand information specifying the two locations and strands that are being␣
↪fused)
junctions = readRDS('../../inst/extdata/junctions.rds')

## loading the GENCODE gene model gTrack (hg19 pre-loaded comes with gTrack,
## but can be made from any .gff file from GENCODE (http://www.gencodegenes.org/
↪releases/19.html)
gt.ge = track.gencode()
```

```
## Pulling gencode annotations from /Library/Frameworks/R.framework/Versions/3.3/
↪Resources/library/gTrack/extdata/gencode.composite.collapsed.rds
```

```
## this loads a gTrack object of a genome graph i.e. network of the same cancer cell␣
↪line (generated by JaBba)
graph = readRDS('../../inst/extdata/graph.rds')

## pick an interesting junction and plot the genes, coverage, and genome graph around␣
↪it
```

(continues on next page)

(continued from previous page)

```
## the links argument specifies the junctions that are being drawn
window = junctions[[290]] + 1e5
```

```
plot(c(gt.ge, gt.cov, graph), window, links = junctions)
```

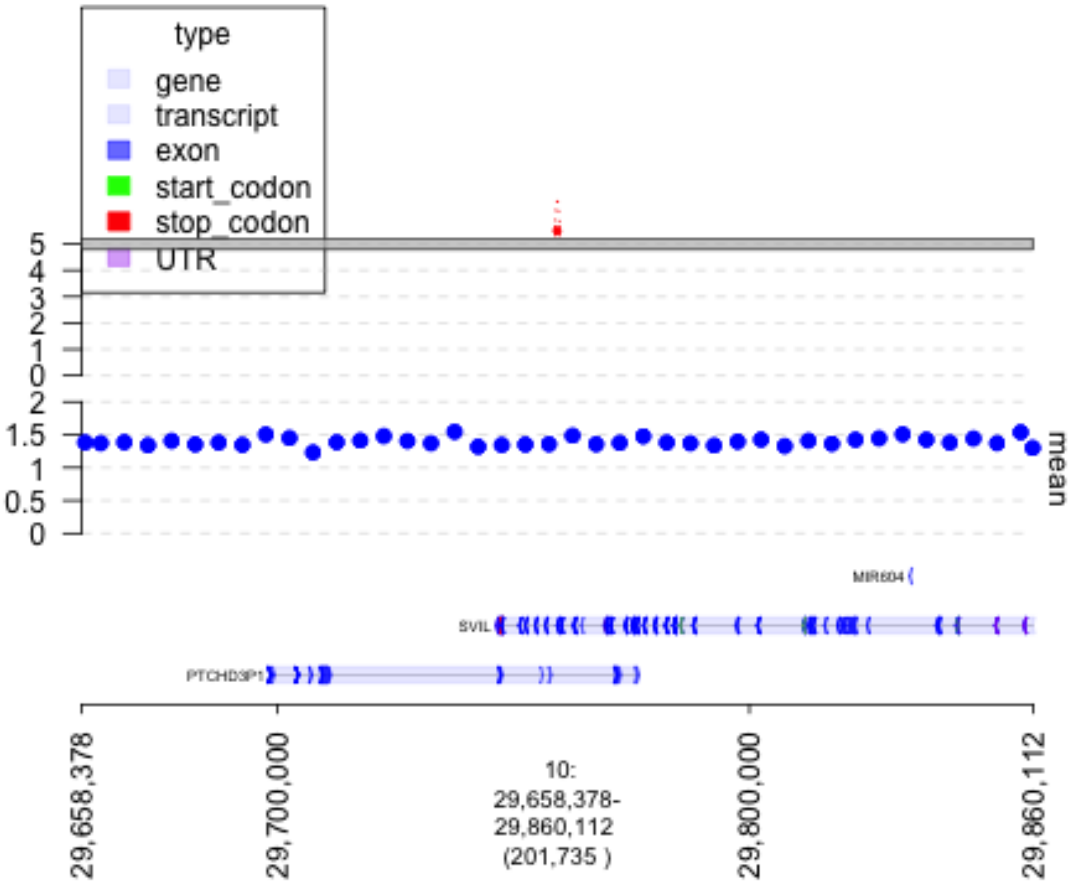


Fig. 1: plot of chunk plot-firstSV

```
ix = 194
cwindow = junctions[[ix]]

jix = c(582, 583)
window = unlist(junctions[jix]) + 3e5
```

(continues on next page)

(continued from previous page)

```
## convert junctions to a data frame. values() returns values from the hash which is␣
↪the junctions object, in this example.
values(junctions)$col = 'gray'
values(junctions)$lwd = 1
values(junctions)$lty = 2 ## dashed instead of dotted line style
values(junctions)$col[jix] = 'red'
values(junctions)$lwd[jix] = 3 ## thicker line width
values(junctions)$lty[jix] = 1 ## solid line style for junction of interest
```

```
plot(c(gt.ge, gt.cov, graph), window, links = junctions)
```

7.1 Graping BAM data

```
## 4 windows corresponding to the 4 breakpoints involved in these two rearrangements.
window = unlist(junctions[jix]) + 250

## pull the reads out in these windows from the tumor and normal bam file.
treads = read.bam("../../inst/extdata/files/tumor.bam", window)
nreads = read.bam("../../inst/extdata/files/normal.bam", window)

## make them into gTracks
td.treads = gTrack(treads, draw.var = TRUE, name = 'Tumor reads')
td.nreads = gTrack(nreads, draw.var = TRUE, name = 'Normal reads')
```

```
plot(c(gt.ge, td.nreads, td.treads), window, links = junctions)
```

```
## dividing tumor read pairs between those that support a rearrangement (i.e. hit␣
↪multiple windows)
## and concordant read pairs that lie only within a single window
treadsr = treads[grl.in(treads, window, logical = FALSE)>1]
treadsc = treads[grl.in(treads, window, logical = FALSE)==1]

## isolating normal
nreadsr = nreads[grl.in(nreads, window, logical = FALSE)>1]
nreadsc = nreads[grl.in(nreads, window, logical = FALSE)==1]

td.treadsr = gTrack(treadsr, draw.var = TRUE, name = 'Tumor reads R', height = 30,␣
↪angle = 45)
td.nreadsr = gTrack(nreadsr, draw.var = TRUE, name = 'Normal reads')
td.treadsc = gTrack(treadsc, draw.var = TRUE, name = 'Tumor reads')
td.nreadsc = gTrack(nreadsc, draw.var = TRUE, name = 'Normal reads C')
```

```
plot(c(gt.ge, td.nreadsc, td.nreadsr, td.treadsc, td.treadsr), window, links =␣
↪junctions)
```

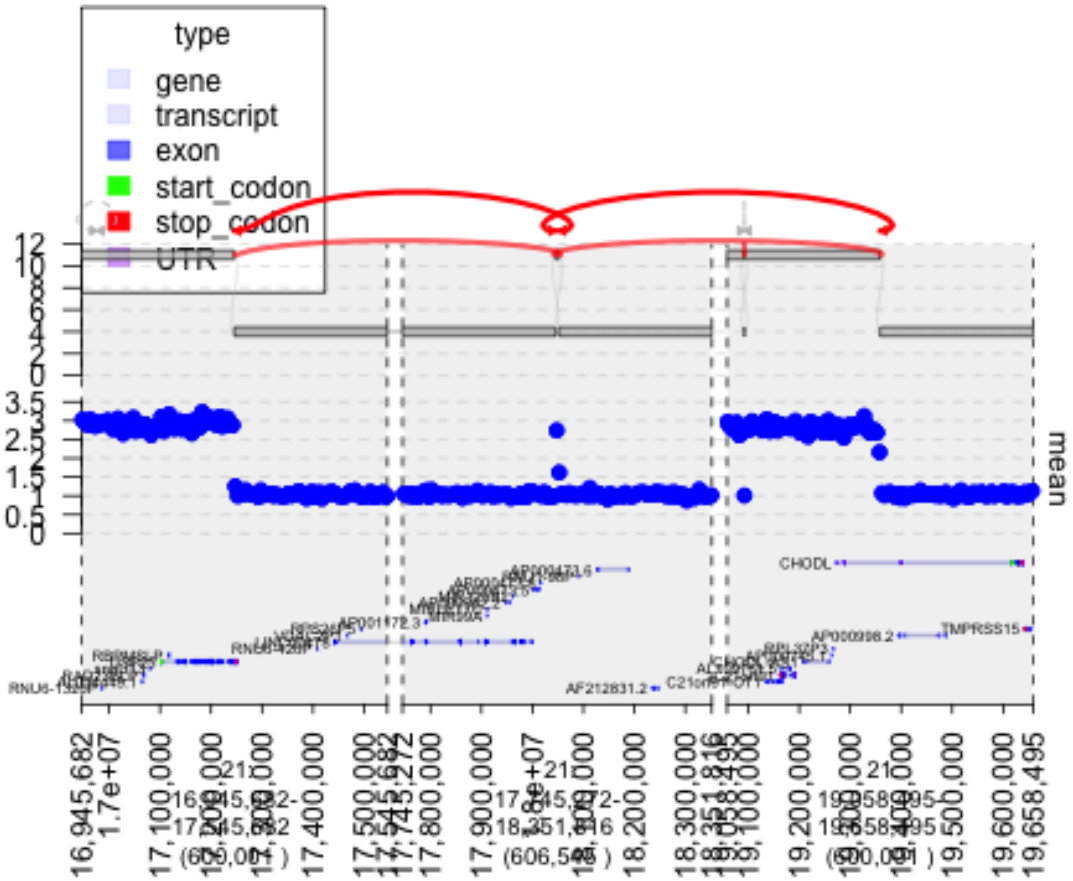


Fig. 2: plot of chunk plot2ndgraph

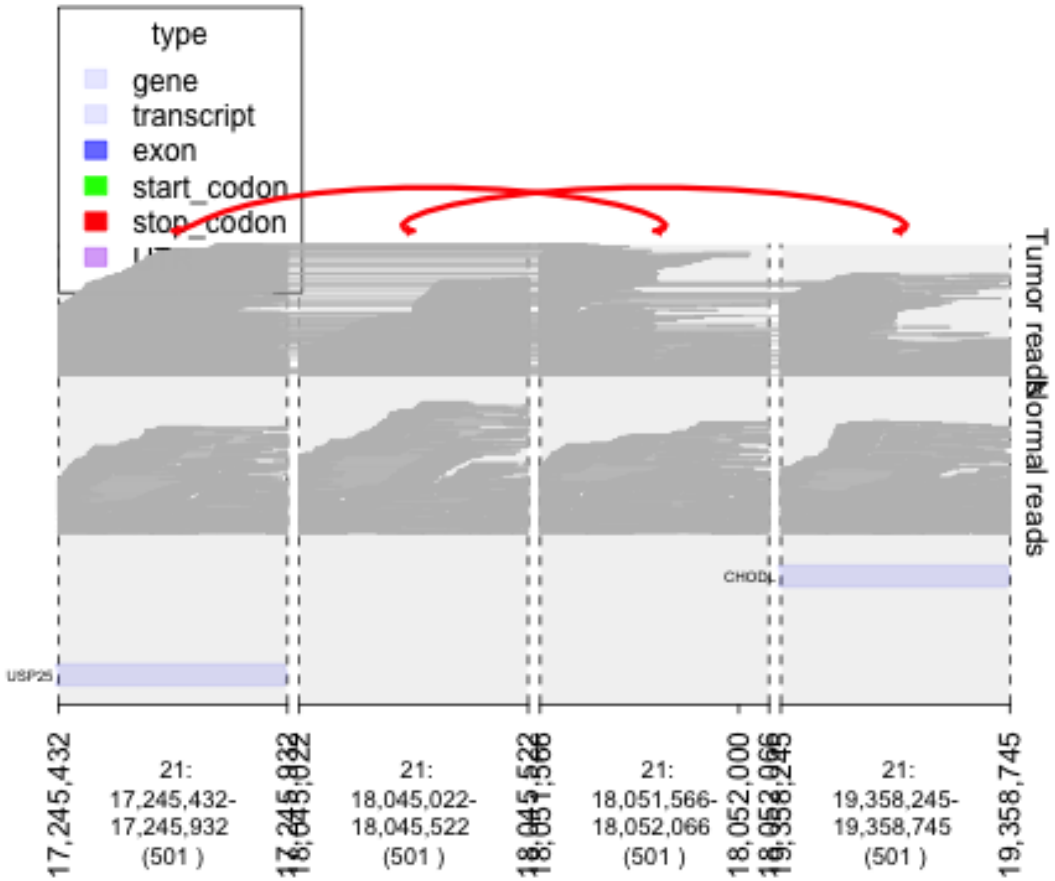


Fig. 3: plot of chunk graph_BAM_data

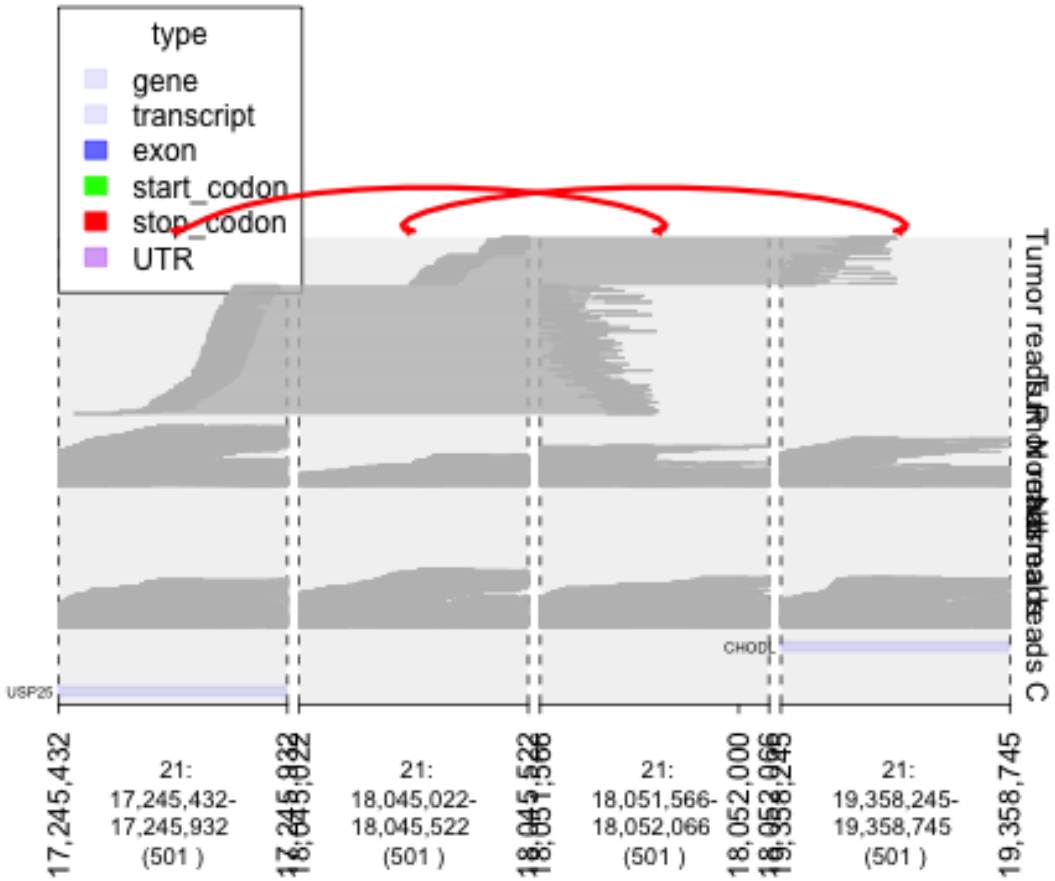


Fig. 4: plot of chunk plotingTumors

CHAPTER 8

How to Graph Super Enhancers

To illustrate gTrack's capability in exploring data sets such as ChIP-Seq data as well as reading BigWig data sets. This entire vignette attemps to replicate Zhang, X. et al (2016) which identified focally amplified super enhancers in epithelial cancers.

```
## The methods section of http://www.nature.com/ng/journal/v48/n2/full/ng.3470.html␣
↪stated
## that GISTIC (used to find copy number variations) analyses were performed on␣
↪available TCGA data
## TCGA data was found on the TCGA copy number portal which is created by the Broad␣
↪Institute of
## MIT and Harvard.

## After finding version 3.0 of the SNP pipeline on 22 October 2014, clicking on the␣
↪IGV
## session returned an XML document (http://portals.broadinstitute.org/tcga/gisticIgv/
↪session.xml?analysisId=21&tissueId=548&type=.xml)
## which stored the web path to the *.seg.gz file. I downloaded that and found
## that it stored the log2 ratios (tumor coverage / normal coverage).

## wget http://www.broadinstitute.org/igvdata/tcga/tcgascape/141024_update/all_
↪cancers.seg.gz
## wget http://www.broadinstitute.org/igvdata/tcga/tcgascape/141024_update/sample_
↪info.txt

## gzip -d all_cancers.seg.gz
```

8.1 Using dt2gr (gUtils) and y.field (gTrack) and gr.colorfield and colormaps

```
##############################                    ##############################
```

(continues on next page)

(continued from previous page)

```
##############################    Starting Analysis    ##############################

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

library(gTrack) ## main sauce.
library(gUtils) ## for dt2gr

## Load coverage data into data.table. Very fast, thanks data.table.
## seg_data <- fread('../../inst/extdata/files/all_cancers.seg')

seg_data[Log2.Ratio <= 0, data_sign := "deletion"]
seg_data[Log2.Ratio > 0, data_sign := "insertion"]

## Coerce into GRanges from data.table because gTrack operates on GRanges.
seg_ranges <- dt2gr(seg_data)

## first glimpse (gotta know what the data looks like). Probably should zoom in␣
↪before even starting.
```

```
## if you want the colors to be chosen automatically.
plot(gTrack(seg_ranges, y.field = 'Log2.Ratio', gr.colorfield = 'data_sign'))
```

```
## if you want to manually set the colors. Better because red/blue can be chosen␣
↪instead of some random colors.
plot(gTrack(seg_ranges, y.field = 'Log2.Ratio', colormaps = list('data_sign' =␣
↪c(insertion = "blue", deletion = "red"))))
```

```
## Subset to MYC enhancer amplification regions.
seg_data_chrom8 <- seg_data[ Chromosome == 8]

## coerce into GRanges from data.table because gTrack operates on GRanges.
seg_ranges_chrom8 <- dt2gr(seg_data_chrom8)
```

```
## if you want to manually set the colors. Better because red/blue can be chosen␣
↪instead of some random colors.
plot(gTrack(seg_ranges_chrom8, y.field = 'Log2.Ratio', colormaps = list('data_sign' =␣
↪c(insertion = "blue", deletion = "red"))), win = seg_ranges_chrom8)
```

8.2 Using parse.gr (gUtils)

```
##############################                         ##############################

##############################    Plot MYC Enhancers   ##############################

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

## first MYC(myc) (s)uper-(e)nhancer.
myc_se <- parse.gr(c('8:129543949-129554294'))
## zoom into that region to view CNA.
win <- myc_se
plot(gTrack(seg_ranges_chrom8, y.field = 'Log2.Ratio', colormaps = list('data_sign' =␣
↪c(insertion = "blue", deletion = "red"))), win)
```

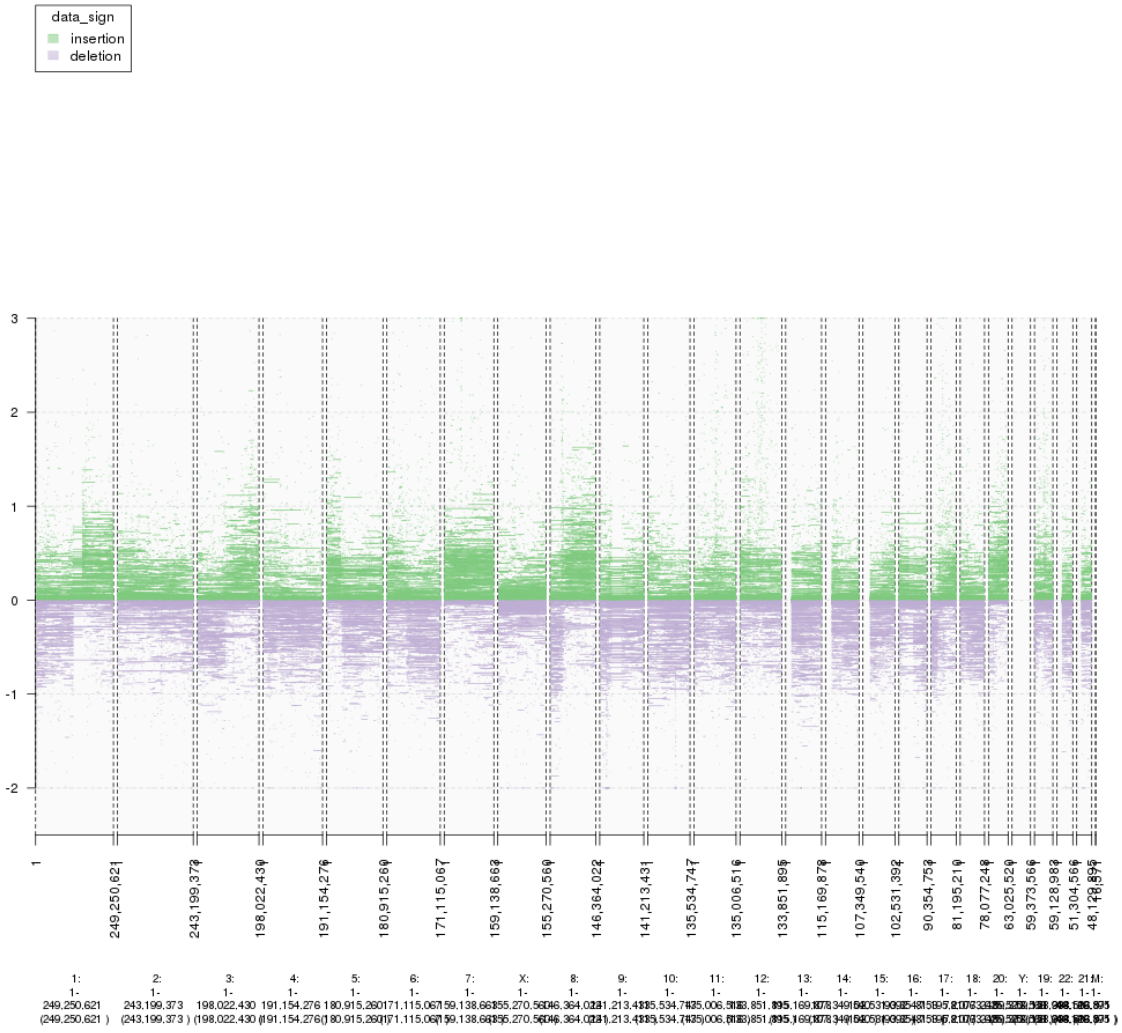


Fig. 1: plot of chunk starting_analysis_plot

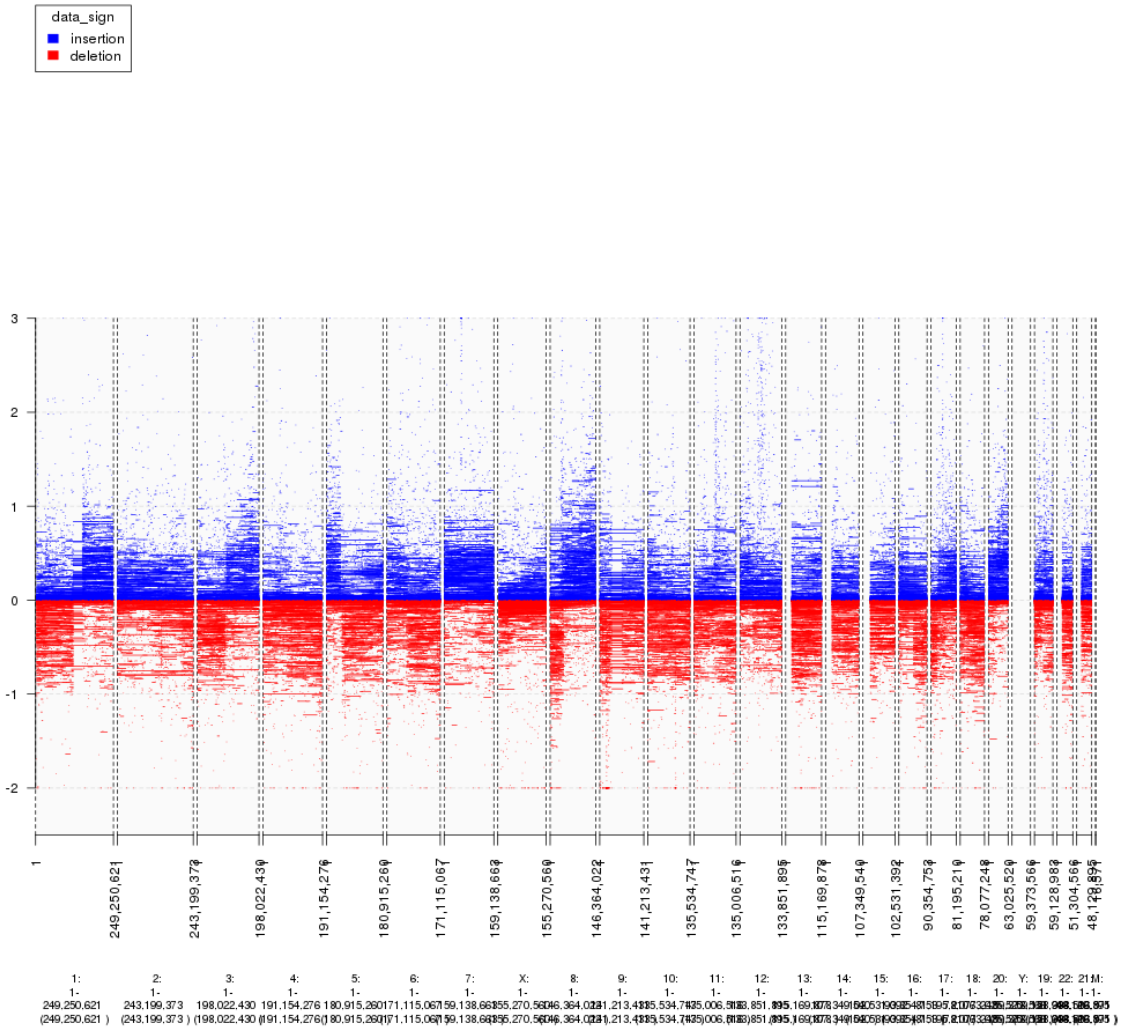


Fig. 2: plot of chunk starting_analysis_plot2

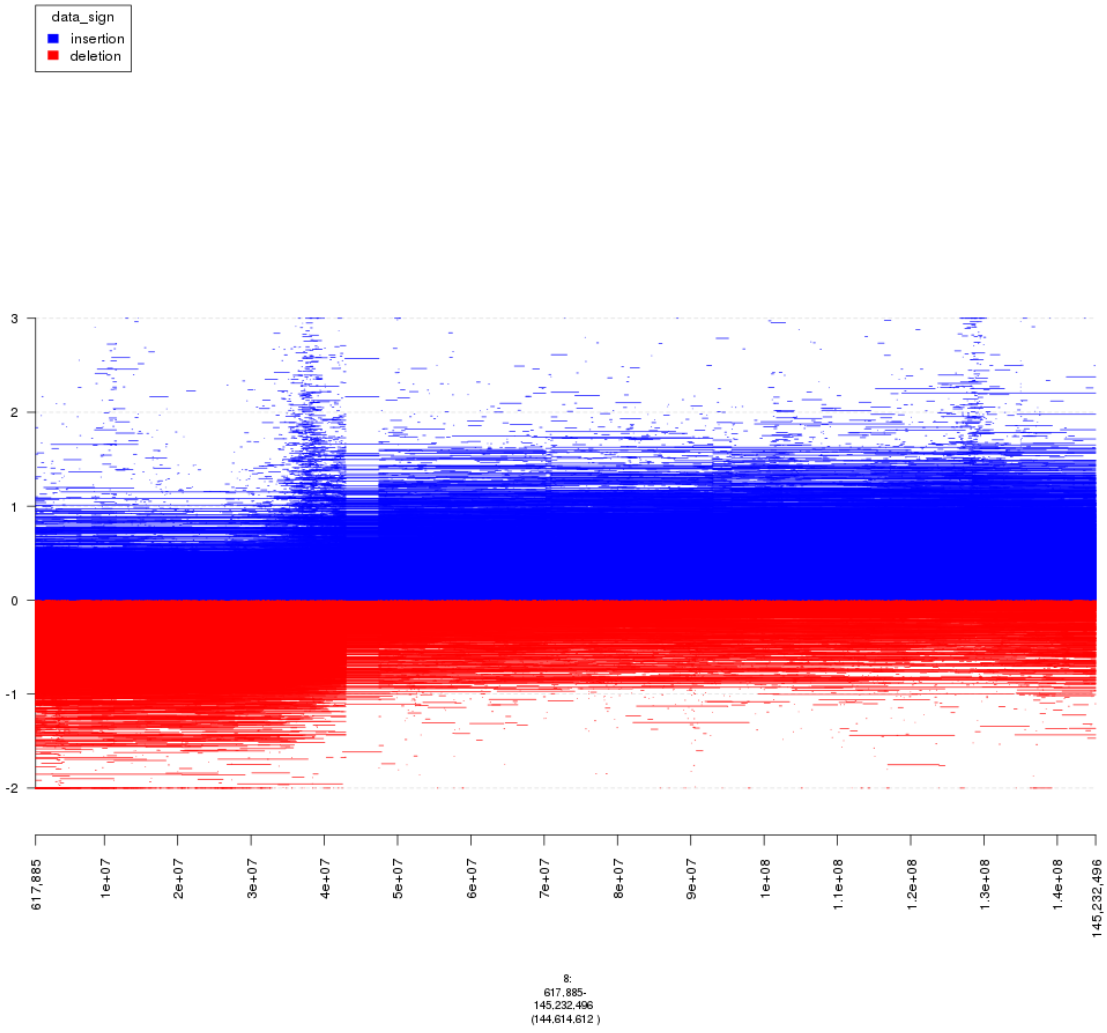


Fig. 3: plot of chunk starting_analysis_plot3

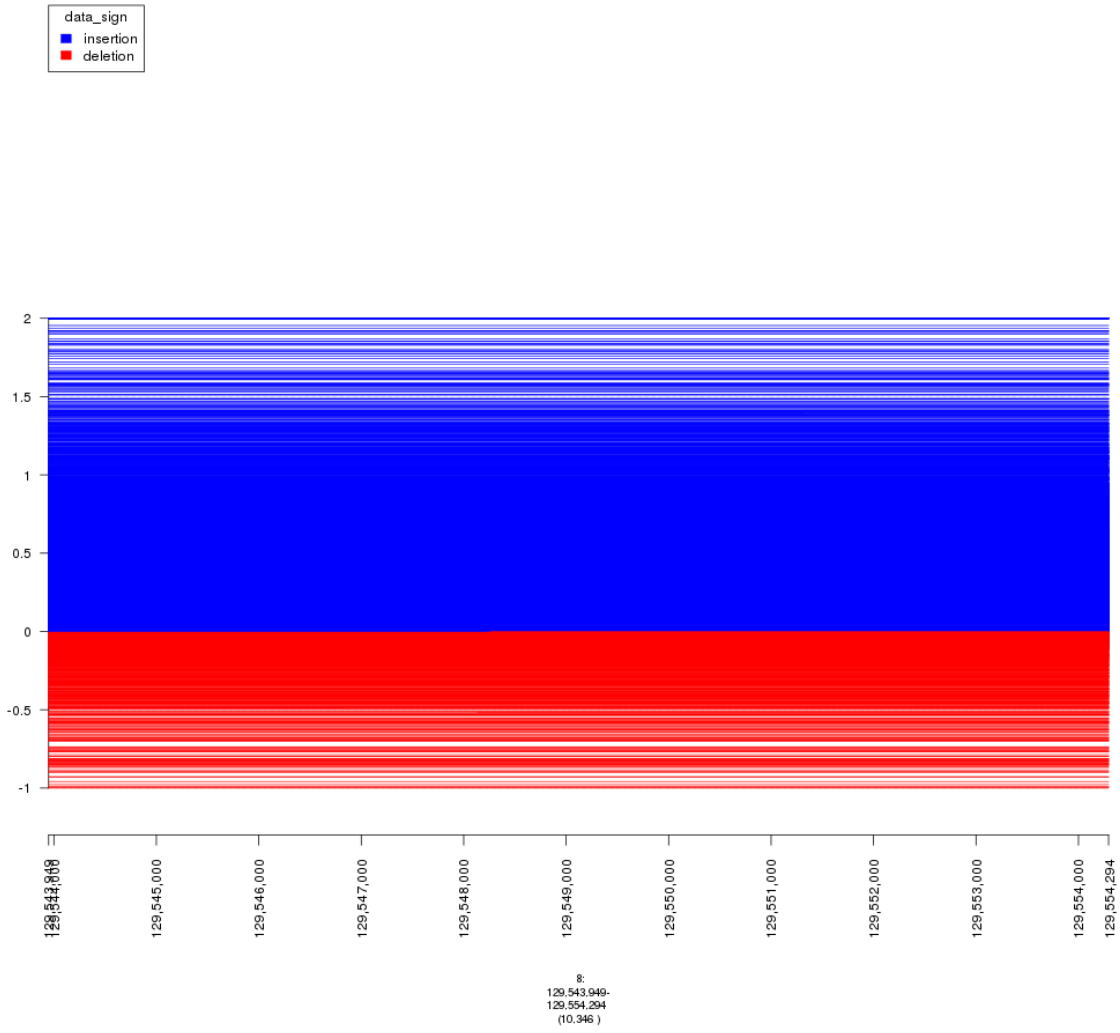


Fig. 4: plot of chunk plot_MYC_enhancers

```
## second MYC super-enhancer
myc_se <- parse.gr(c('8:129166547-129190290'))
win <- myc_se
plot(gTrack(seg_ranges_chrom8, y.field = 'Log2.Ratio', colormaps = list('data_sign' =
↪c(insertion = "blue", deletion = "red"))), win)
```

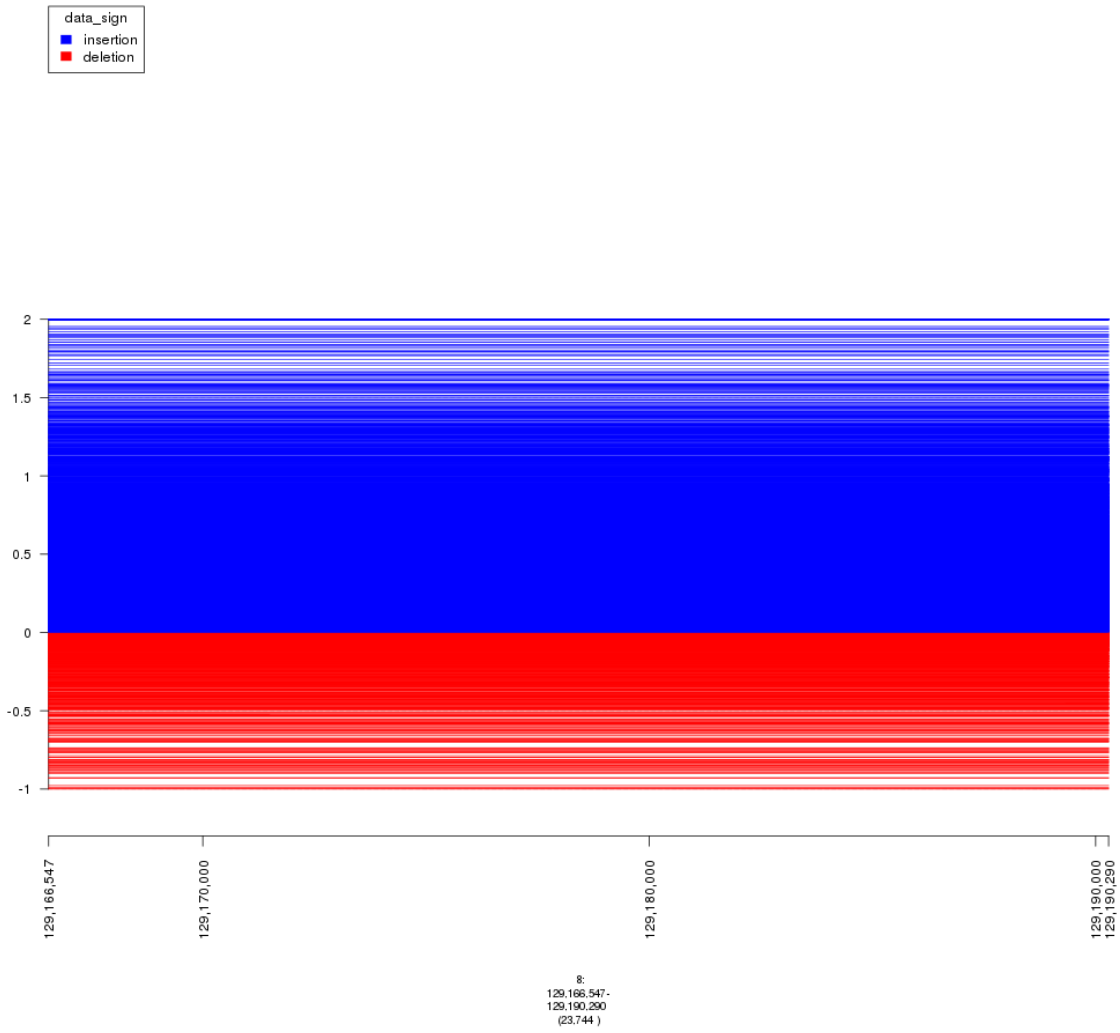


Fig. 5: plot of chunk plot_MYC_enhancers

```
## it looks like both regions have focal insertions and deletions.
plot(gTrack(seg_ranges_chrom8, colormaps = list('data_sign' = c(insertion = "blue",
↪deletion = "red"))), win = seg_ranges_chrom8+10e6)
```

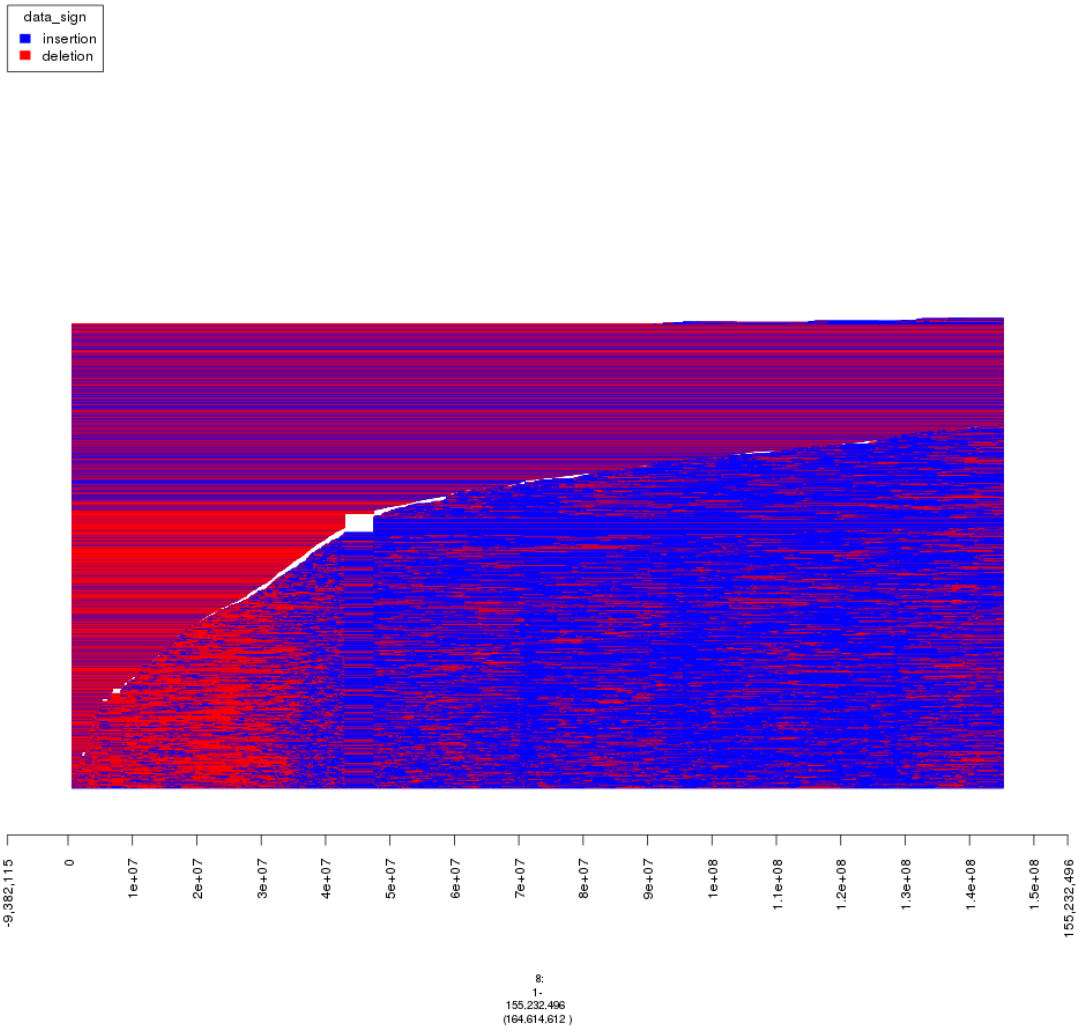


Fig. 6: plot of chunk plot_MYC_enhancers

```
##############################                              ##############################

##############################    Setting Thresholds    ##############################

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

## max width is not 50 MB (actually 30KB) to remove very broad copy number changes.
## min width is not 20KB to exclude artifacts.

seg_data_chrom8 <- seg_data_chrom8[End.bp - Start.bp <= 30e3]
seg_ranges_chrom8 <- dt2gr(seg_data_chrom8)
plot(gTrack(seg_ranges_chrom8, colormaps = list('data_sign' = c(insertion = "blue",␣
↪deletion = "red"))), win = seg_ranges_chrom8+10e6)
```

```
## explore data set for determining threshold for log2 ratio.

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

##############################       Random Fact        ##############################

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

## There are more insertions than deletions.
sorted_ratios <- sort(seg_data_chrom8$'Log2.Ratio')
length(sorted_ratios)
```

```
## [1] 4458
```

```
#### -1 and 2
seg_data_chrom8_2 <- seg_data_chrom8[Log2.Ratio >= -1 & Log2.Ratio <= 2]
seg_ranges_chrom8_2 <- dt2gr(seg_data_chrom8_2)

plot(gTrack(seg_ranges_chrom8_2, colormaps = list('data_sign' = c(insertion = "blue",␣
↪deletion = "red"))), win = seg_ranges_chrom8_2+10e6)
```

```
##############################                              ##############################
↪#
            # Not much of a change, will ignore setting thresholds for Log2.Ratio
##############################                              ##############################
↪#
```

8.3 Reading bigWig in gTrack

```
## bigWig downloaded from https://www.encodeproject.org/experiments/ENCSR000AUI/

## fold change.
plot(gTrack('~/my_git_packages/super_enhancers/db/ENCFF038AQV.bigWig'), win = parse.
↪gr('8:128635434-128941434'))
```

```
### store gencode genes.
ge = track.gencode()
```

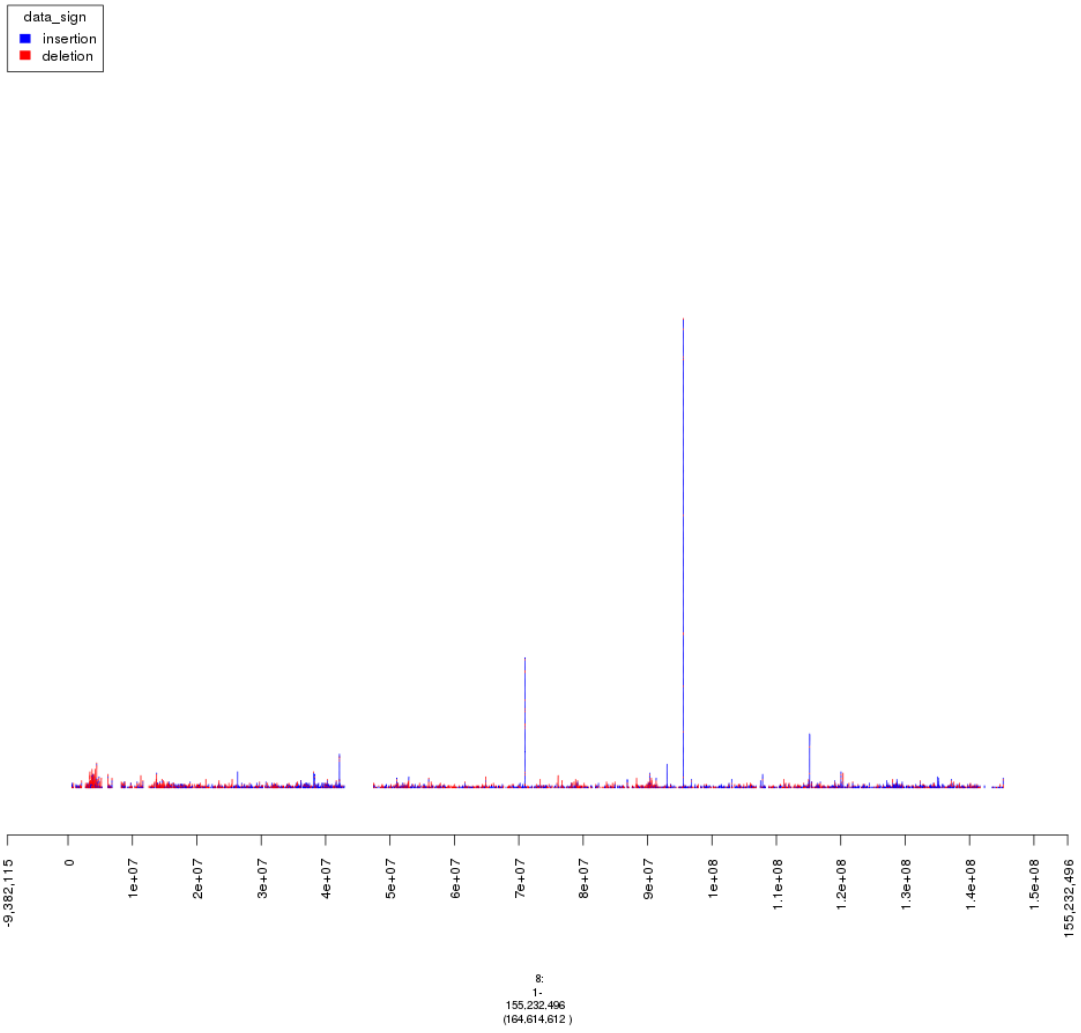


Fig. 7: plot of chunk setting_thresholds

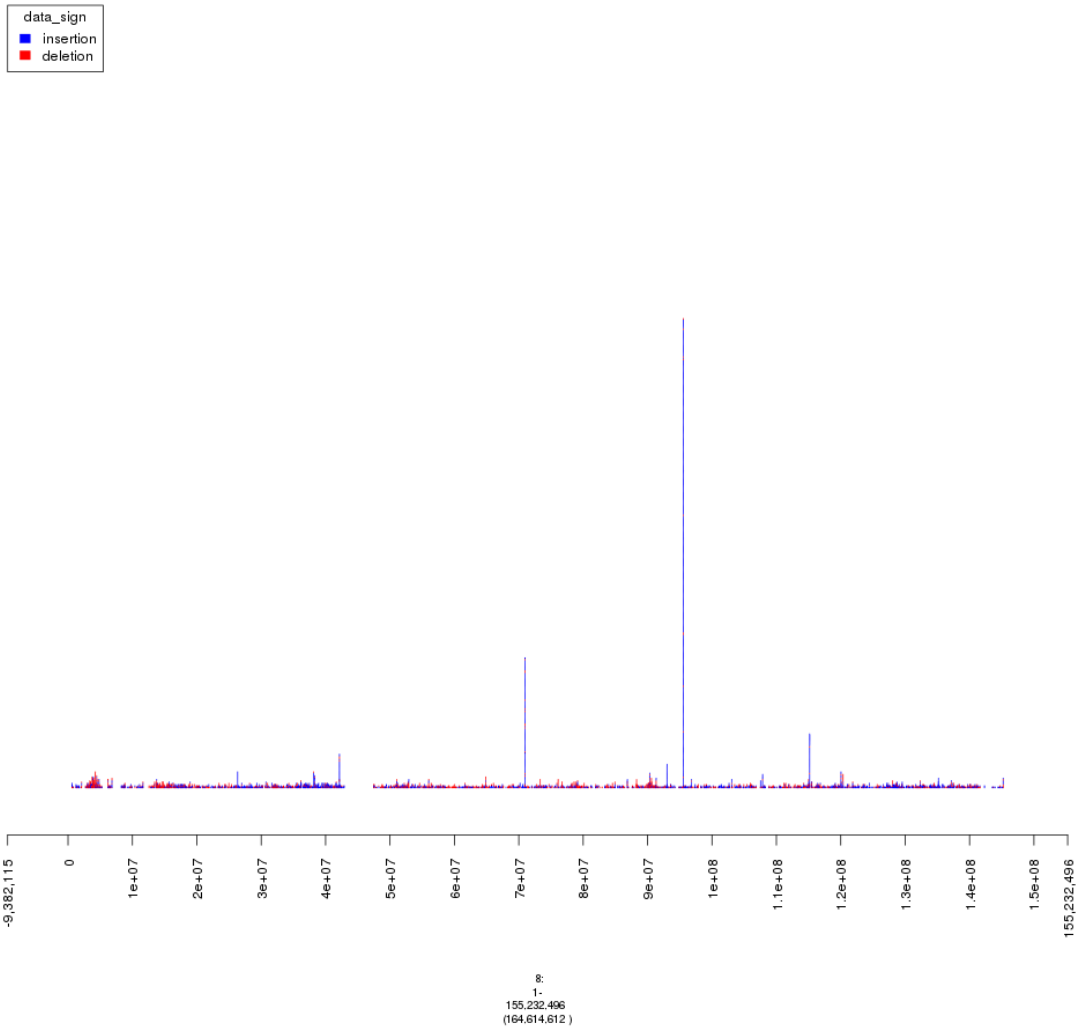


Fig. 8: plot of chunk random_fact

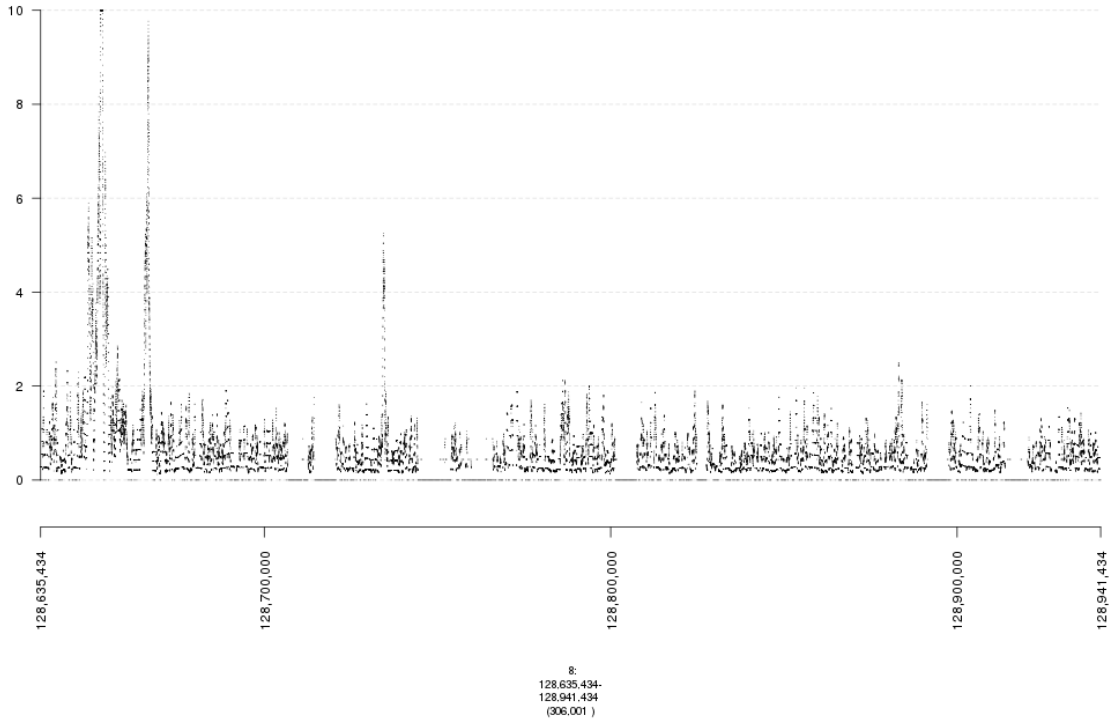


Fig. 9: plot of chunk bigWig

```
## Pulling gencode annotations from /gpfs/commons/groups/imielinski_lab/lib/R-3.3.0/
↪gTrack/extdata/gencode.composite.collapsed.rds
```

```
### Plot ENCODE, peak super-enhancer, and copy number data.
### without super-enhancers.

plot(c(gTrack('~/my_git_packages/super_enhancers/db/ENCFF038AQV.bigWig', color =
↪'green'), gTrack(seg_ranges_chrom8, colormaps = list('data_sign' = c(insertion =
↪"blue", deletion = "red"))), ge), win = parse.gr('8:128635434-128941434'))
```

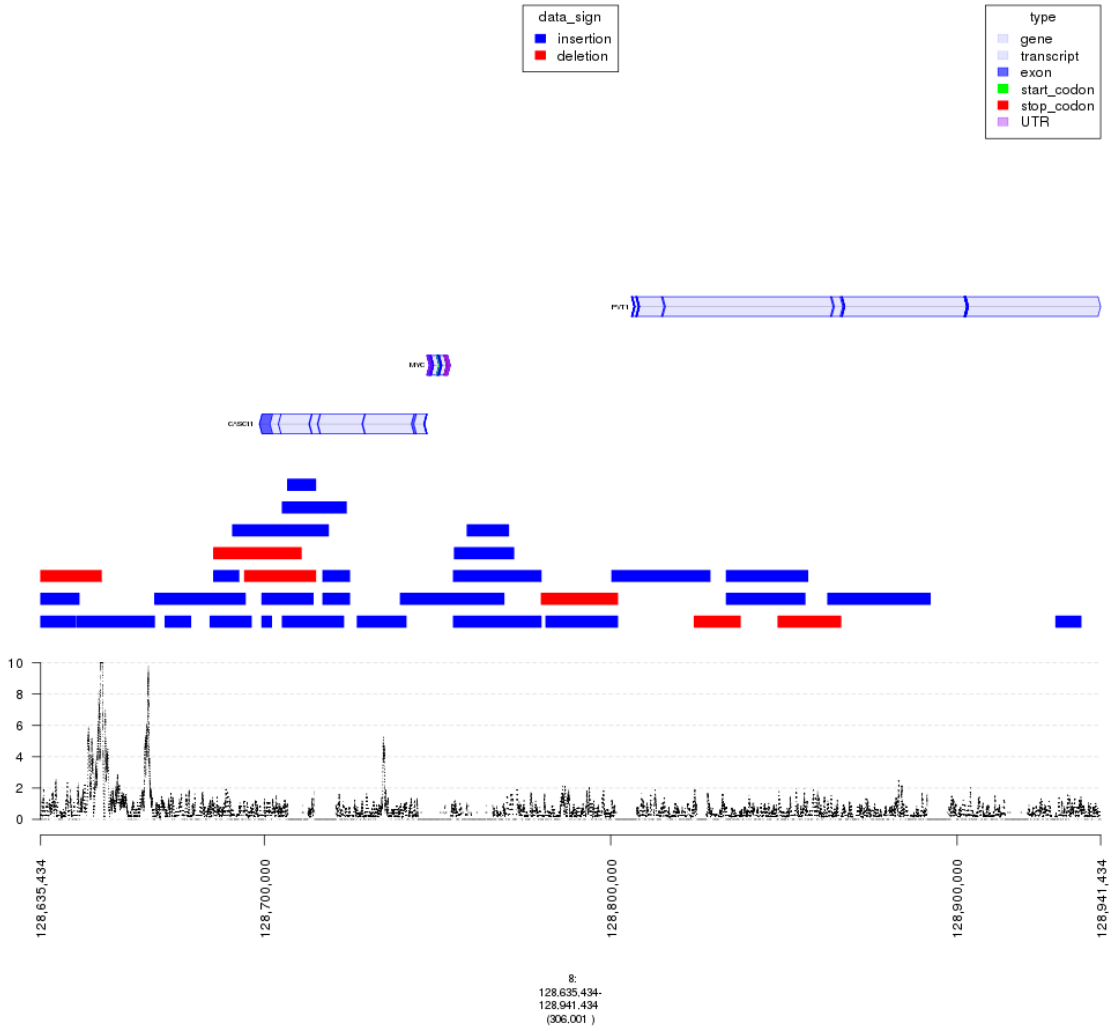


Fig. 10: plot of chunk bigWig

```
### plotting regions witout super-enhancers.
plot(c(gTrack('~/my_git_packages/super_enhancers/db/ENCFF038AQV.bigWig', color =
↪'green', bar = TRUE), gTrack(seg_ranges_chrom8, colormaps = list('data_sign' =␣
↪c(insertion = "blue", deletion = "red"))), ge), win = parse.gr('8:128735434-
↪129641434'))
```

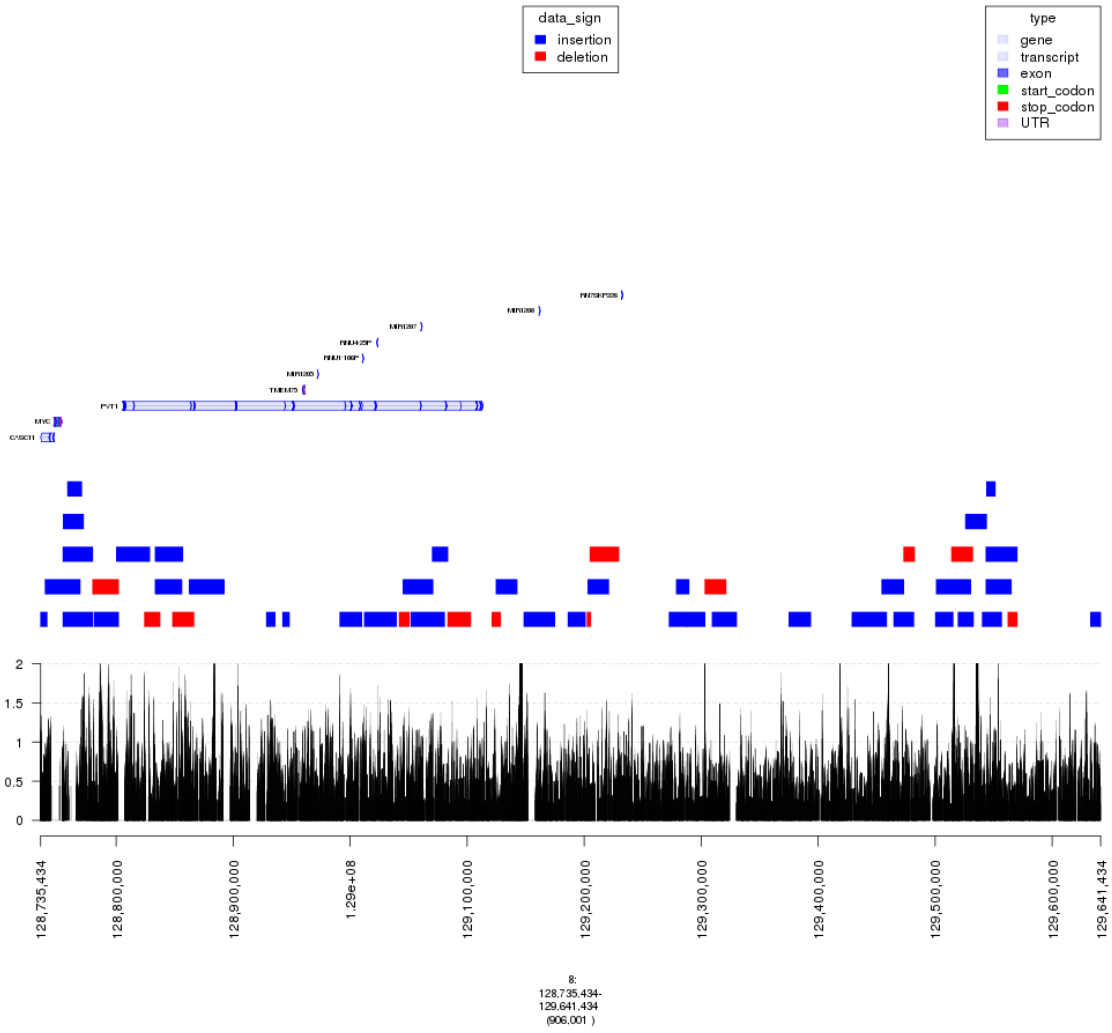


Fig. 11: plot of chunk bigWig

```
### Split the copy number data into two objects - one for insertions & other for␣
↪deletions.

seg_data_chrom8_insertions <- seg_data_chrom8[data_sign == "insertion"]
seg_data_chrom8_deletions <- seg_data_chrom8[data_sign == "deletion"]
```

(continues on next page)

(continued from previous page)

```
seg_ranges_chrom8_insertions <- dt2gr(seg_data_chrom8_insertions)
seg_ranges_chrom8_deletions <- dt2gr(seg_data_chrom8_deletions)

### with super-enhancers & gencode & ChIP-seq & insertions/deletions split.
plot(c(gTrack('~/my_git_packages/super_enhancers/db/ENCFF038AQV.bigWig', bar = TRUE),␣
↪gTrack(seg_ranges_chrom8_insertions, col = "blue"), gTrack(seg_ranges_chrom8_
↪deletions, col = "red"), ge), win = parse.gr('8:128735434-129641434'))
```

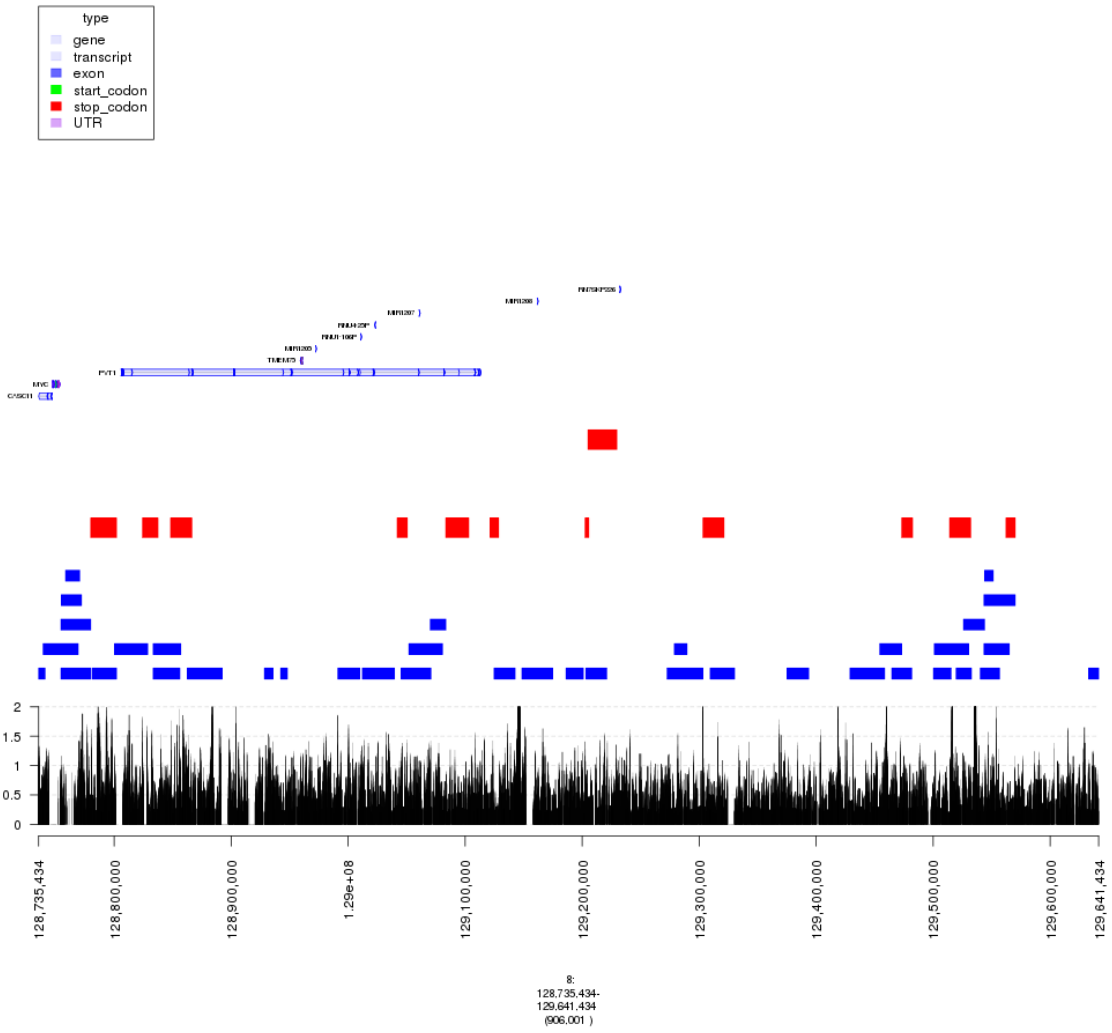


Fig. 12: plot of chunk bigWig

```
## view the density of insertions.
plot(gTrack(seg_ranges_chrom8_insertions, y.field = "Log2.Ratio", col = "blue"), win
↪= parse.gr('8:128735434-129641434'))
```

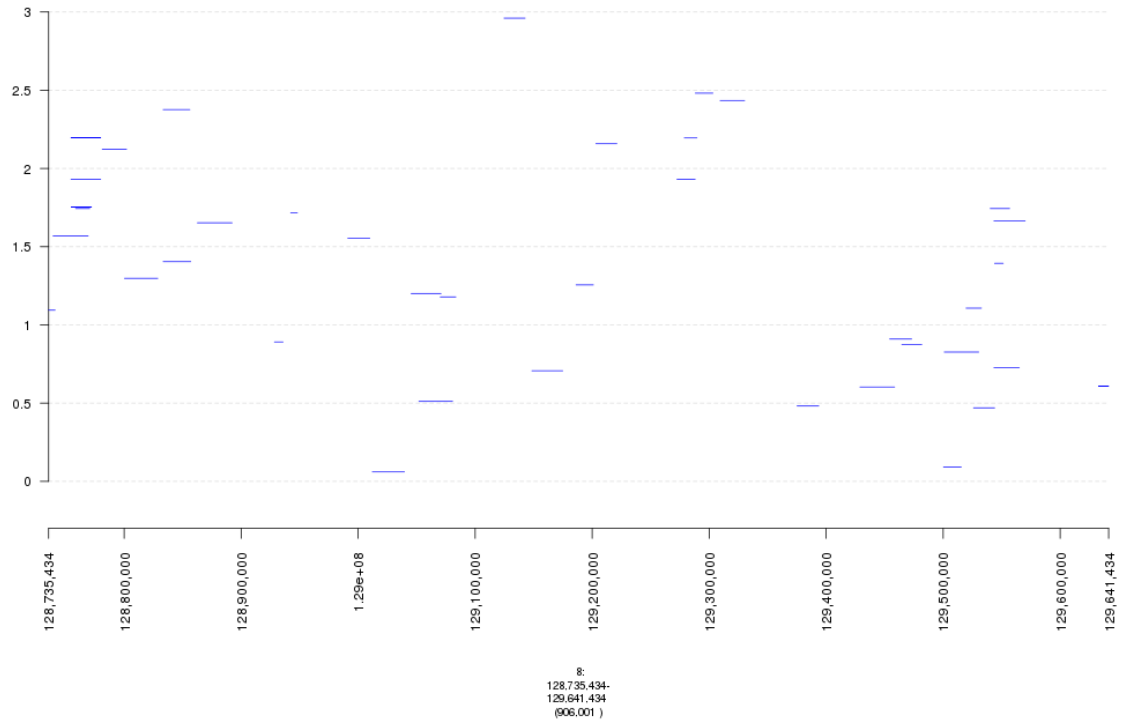


Fig. 13: plot of chunk bigWig

```
### Filtering broad events
seg_data_chrom8_deletions2 <- seg_data_chrom8_deletions[Log2.Ratio >= -0.6]
seg_data_chrom8_insertions2 <- seg_data_chrom8_insertions[Log2.Ratio >= 0.6]

seg_ranges_chrom8_insertions <- dt2gr(seg_data_chrom8_insertions)
seg_ranges_chrom8_deletions <- dt2gr(seg_data_chrom8_deletions)
```

(continues on next page)

(continued from previous page)

```
plot(c(gTrack('~/my_git_packages/super_enhancers/db/ENCFF038AQV.bigWig', color =
↪'green', bar = TRUE), gTrack(seg_ranges_chrom8_insertions, col = "blue"),␣
↪gTrack(seg_ranges_chrom8_deletions, col = "red"), ge), win = parse.gr('8:128735434-
↪129641434'))
```

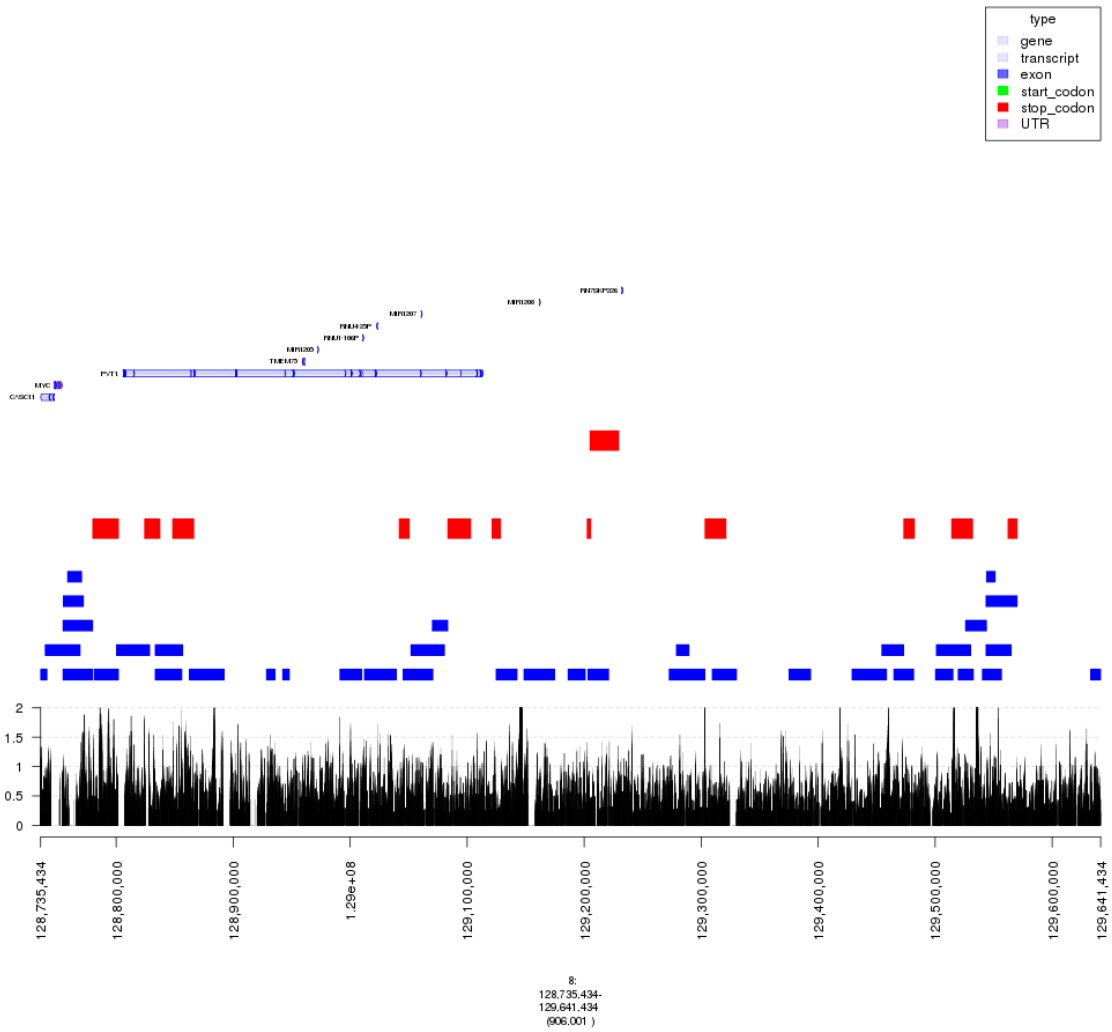


Fig. 14: plot of chunk filter_broad_events

```
### Replicable pipeline

## Subset to MYC enhancer amplifications regions.
seg_data_chrom8 <- seg_data[ Chromosome == 8]
## coerce data.table into GRanges because gTrack operates on GRanges.
```

(continues on next page)

(continued from previous page)

```
seg_ranges_chrom8 <- dt2gr(seg_data_chrom8)

## max width is 10MB.
seg_data_chrom8 <- seg_data_chrom8[End.bp - Start.bp <= 10e6]

seg_data_chrom8_deletions <- seg_data_chrom8[Log2.Ratio <= 0, data_sign := "deletion"]
seg_data_chrom8_insertions <- seg_data_chrom8[Log2.Ratio > 0, data_sign := "insertion
↪"]


seg_data_chrom8_insertions <- seg_data_chrom8[data_sign == "insertion"]
seg_data_chrom8_deletions <- seg_data_chrom8[data_sign == "deletion"]

gray = 'gray20'
gt.h3k36 = gTrack('~/DB/Roadmap/consolidated//E114-H3K36me3.pval.signal.bigwig', name␣
↪= 'H3K36me3', bar = TRUE, col = gray)
gt.h3k4 = gTrack('~/DB/Roadmap/consolidated//E114-H3K4me3.pval.signal.bigwig', name =
↪'H3K4me3', bar = TRUE, col = gray)
gt.enh = gTrack('~/DB/Roadmap/consolidated//E114-H3K27ac.pval.signal.bigwig', name =
↪'H3K27Ac', bar = TRUE, col = gray)
gt.open = gTrack('~/DB/Roadmap/consolidated//E114-DNase.pval.signal.bigwig', name =
↪'DNAase', bar = TRUE, col = gray)
gt.rnapos = gTrack('~/DB/Roadmap/consolidated/E114.A549.norm.pos.bw', name = 'RNAseq+
↪', bar = TRUE, col = gray)
gt.rnaneg = gTrack('~/DB/Roadmap/consolidated/E114.A549.norm.neg.bw', name = 'RNAseq-
↪', bar = TRUE, col = gray, y0 = 0, y1 = 1200)

THRESH = 1
seg_data_chrom8_deletions <- seg_data_chrom8_deletions[Log2.Ratio >= -THRESH]
seg_data_chrom8_insertions <- seg_data_chrom8_insertions[Log2.Ratio >= THRESH]
seg_ranges_chrom8_insertions <- dt2gr(seg_data_chrom8_insertions)
seg_ranges_chrom8_deletions <- dt2gr(seg_data_chrom8_deletions)
plot(c(gTrack('~/my_git_packages/super_enhancers/db/ENCFF038AQV.bigWig', color =
↪'green', bar = TRUE), gTrack(seg_ranges_chrom8_insertions, col = "blue"),␣
↪gTrack(seg_ranges_chrom8_deletions, col = "red"), ge), win = parse.gr('8:128735434-
↪129641434'))
```

```
acov = as(coverage(seg_ranges_chrom8_insertions), 'GRanges')
dcov = as(coverage(seg_ranges_chrom8_deletions), 'GRanges')
plot(c(gt.rnapos, gt.enh, gTrack('~/my_git_packages/super_enhancers/db/ENCFF038AQV.
↪bigWig', color = 'green', bar = TRUE), gTrack(acov, 'score', bar = TRUE),␣
↪gTrack(dcov, 'score', bar = TRUE),  gTrack(seg_ranges_chrom8_insertions, col = "blue
↪"), gTrack(seg_ranges_chrom8_deletions, col = "red"), ge), win = parse.gr(
↪'8:128735434-129641434'))+1e6
```

```
## numeric(0)
```

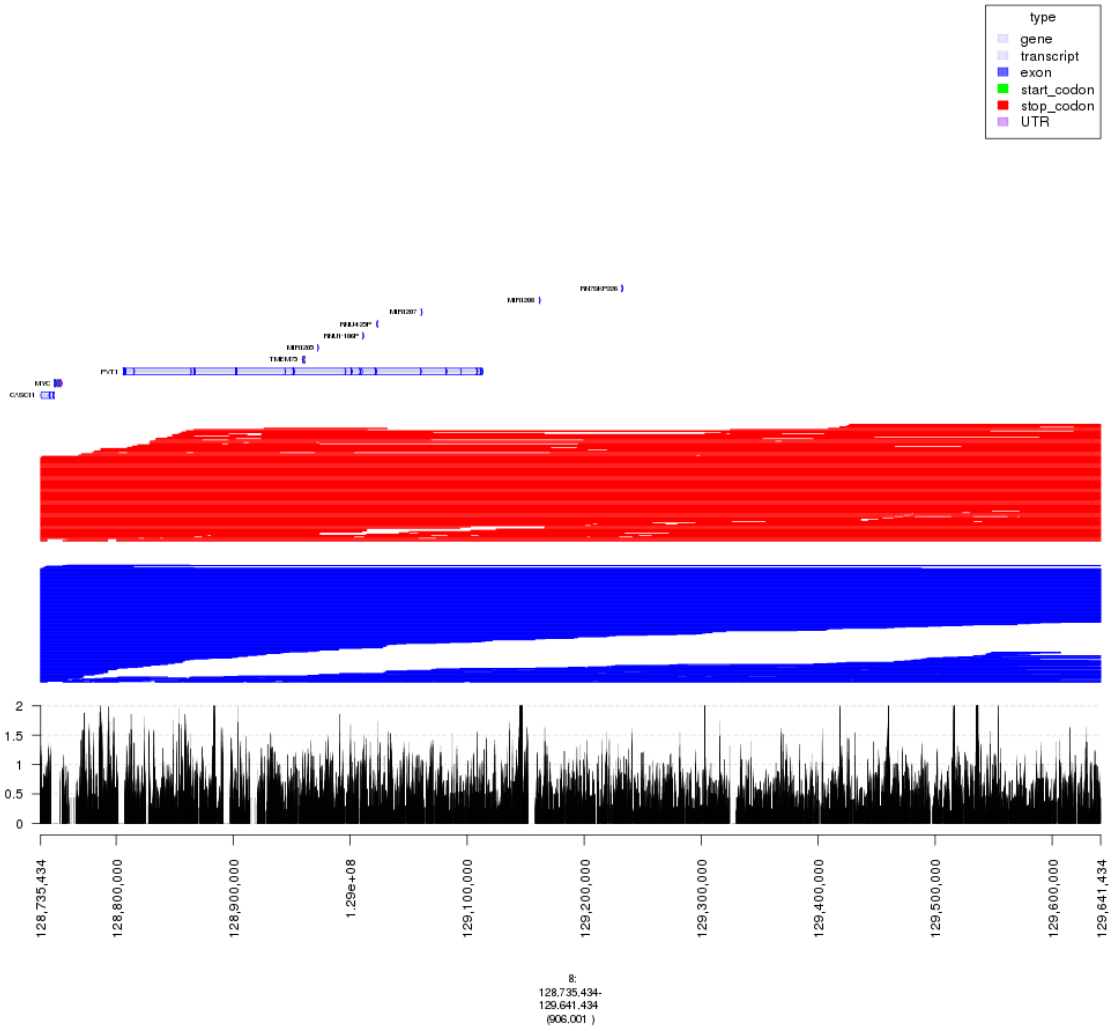


Fig. 15: plot of chunk filter_broad_events

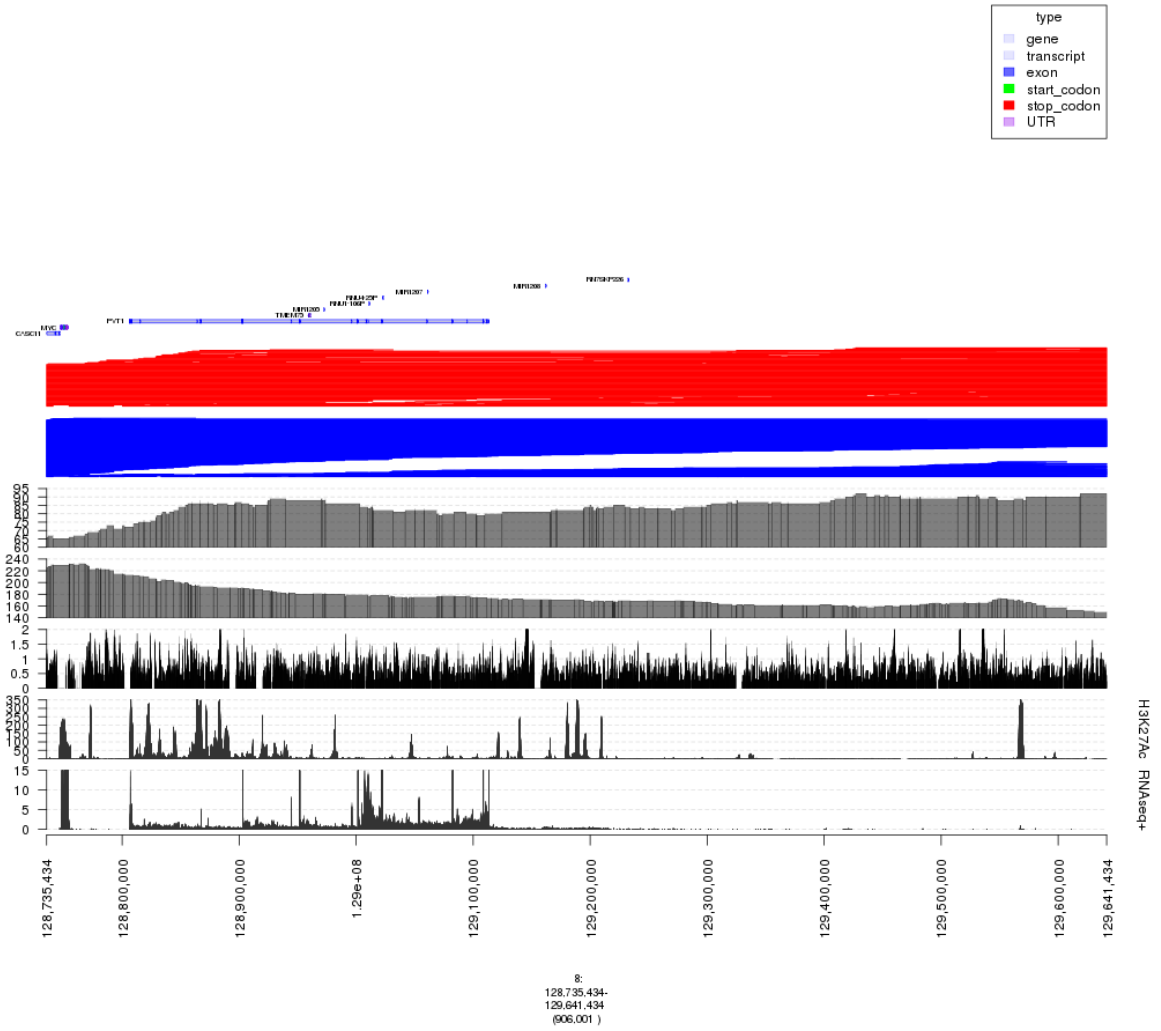


Fig. 16: plot of chunk filter_broad_events

CHAPTER 9

Indices and tables

- genindex
- modindex
- search