ggbio: visualization toolkits for genomic data

Tengfei Yin[1]

July 16, 2015

[1]tengfei.yin@sbgenomics.com

Contents

Chapter 1

Getting started

1.1 Citation

```
citation("ggbio")
```

1.2 Introduction

ggbio is a *Bioconductor* package building on top of *ggplot2*(), leveraging the rich objects defined by *Bioconductor* and its statistical and computational power, it provides a flexible genomic visualization framework, extends the grammar of graphics into genomic data, try to delivers high quality, highly customizable graphics to the users.

What it features

- `autoplot` function provides ready-to-use template for *Bioconductor* objects and different types of data.
- flexible low level components to use grammar of graphics to build you graphics layer by layer.
- layout transformation, so you could generate circular plot, grandlinear plot, stacked overview more easily.
- flexible tracks function to bind any *ggplot2*(), *ggbio* based plots.

Chapter 2

Case study: building your first tracks

In this chapter, you will learn

- how to add ideogram track.
- How to add gene model track.
- how to add track for bam files to visualize coverage and mismatch summary.
- how to add track for vcf file to visualize the variants.

2.1 Add an ideogram track

`Ideogram` provides functionality to construct ideogram, check the manual for more flexible methods. We build genome *hg19, hg18, mm10, mm9* inside, so you don't have download it on the fly. When embed with tracks, ideogram show zoomed region highlights automatically. `xlim` has special function here, is too changed highlighted zoomed region on the ideogram.

```
library(ggbio)
p.ideo <- Ideogram(genome = "hg19")
p.ideo
library(GenomicRanges)
## special highlights instead of zoomin!
p.ideo + xlim(GRanges("chr2", IRanges(1e8, 1e8+10000000)))
```

2.2 Add a gene model track

2.2.1 Introduction

Gene model track is one of the most frequently used track in genome browser, it is composed of genetic features CDS, UTR, introns, exons and non-genetic region. In *ggbio* we support three methods to make gene model track:

- *OrganismDb* object: recommended, support gene symbols and other combination of columns as label.
- *TxDb* object: don't support gene symbol labeling.
- *GRangesList* object: flexible, if you don't have annotation package available for the first two methods, you could prepare a data set parsed from gtf file, you can simply use it and plot it as gene model track.

2.2.2 Make gene model from OrganismDb object

OrganismDb object has a simpler API to retrieve data from different annotation resources, so we could label our transcripts in different ways

```
library(ggbio)
library(Homo.sapiens)
class(Homo.sapiens)
##
data(genesymbol, package = "biovizBase")
wh <- genesymbol[c("BRCA1", "NBR1")]
wh <- range(wh, ignore.strand = TRUE)


p.txdb <- autoplot(Homo.sapiens, which  = wh)
p.txdb

autoplot(Homo.sapiens, which  = wh, label.color = "black", color = "brown",
         fill = "brown")
```

To change the intron geometry, use `gap.geom` to control it, check out `geom_alignment` for more control parameters.

```
autoplot(Homo.sapiens, which  = wh, gap.geom = "chevron")
```

To collapse all features, use `stat` 'reduce'

```
autoplot(Homo.sapiens, which = wh, stat = "reduce")
```

Label could be turned off by setting it to `FALSE`, you could also use expression to make a flexible label combination from column names.

```
columns(Homo.sapiens)
autoplot(Homo.sapiens, which  = wh, columns = c("TXNAME", "GO"), names.expr = "TXNAME::GO")
```

2.2.3 Make gene model from TxDb object

TxDb doesn't contain any gene symbol information, so we use tx_id as default for label.

```
library(TxDb.Hsapiens.UCSC.hg19.knownGene)
txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene
autoplot(txdb, which = wh)
```

2.2.4 Make gene model from GRangesList object

Sometimes your gene model is not available as none of *OrganismDb* or *TxDb* object, it's may be stored in a table, you could simple parse it into a *GRangeList* object.

- each group indicate one transcripts
- names of group are shown as labels
- this object must has a column contains following key word: cds, exon, intron, and it's not case senstitive. use `type` to map this column. By default, we will try to parse 'type' column.

Let's make a sample *GRangesList* object which contains all information, and fake some labels.

```
library(biovizBase)
gr.txdb <- crunch(txdb, which = wh)
```

```
## change column to 'model'
colnames(values(gr.txdb))[4] <- "model"
grl <- split(gr.txdb, gr.txdb$tx_id)
## fake some randome names
names(grl) <- sample(LETTERS, size = length(grl), replace = TRUE)
grl
```

We get our example data ready, it meets all requirements, to make it a gene model track it's pretty simple to use autoplot, but don't forget mapping because we changed our column names, asssume you store you model key words in column 'model'.

```
autoplot(grl, aes(type = model))
ggplot() + geom_alignment(grl, type = "model")
```

2.3 Add a reference track

To add a reference track, we need to load a *BSgenome* object from the annotation package. You can choose to plot the sequence as *text, rect, segment*.

2.3.1 Semantic zoom

Here we introduce semantic zoom in *ggbio*, for some plots like reference sequence, we use pre-defined zoom level threshold to automatically assign geom to the track, unless the geom is explicitly specified. In the example below, when your region is too wide we show text 'zoom in to see text', when you zoom into different level, it shows you different details. `zoom` is a function we will introduce more in chapter 3 when we introduce more about navigation.

You can pass a zoom in factor into `zoom` function, if it's over 1 it's zooming out, if it's smaller than 1 it's zooming in.

```
library(BSgenome.Hsapiens.UCSC.hg19)
bg <- BSgenome.Hsapiens.UCSC.hg19
p.bg <- autoplot(bg, which = wh)
## no geom
p.bg
## segment
p.bg + zoom(1/100)
## rectangle
p.bg + zoom(1/1000)
## text
p.bg + zoom(1/2500)
```

To override a zemantic zoom threshold, you simply provide a geom explicitly.

```
library(BSgenome.Hsapiens.UCSC.hg19)
bg <- BSgenome.Hsapiens.UCSC.hg19
## force to use geom 'segment' at this level
autoplot(bg, which = resize(wh, width = width(wh)/2000), geom = "segment")
```

2.4 Add an alignment track

ggbio supports visuaization of alignemnts file stored in bam, `autoplot` method accepts

- bam file path (indexed)

- *BamFile* object
- *GappedAlignemnt* object

It's simple to just pass a file path to `autoplot` function, you can stream a chunk of region by providing 'which' parameter. Otherwise please use method 'estiamte' to show overall estiamted coverage.

```
fl.bam <- system.file("extdata", "wg-brca1.sorted.bam", package  = "biovizBase")
wh <- keepSeqlevels(wh, "chr17")
autoplot(fl.bam, which = wh)
```

geom 'gapped pair' will show you alignments.

```
fl.bam <- system.file("extdata", "wg-brca1.sorted.bam", package  = "biovizBase")
wh <- keepSeqlevels(wh, "chr17")
autoplot(fl.bam, which = resize(wh, width = width(wh)/10), geom = "gapped.pair")
```

To show mismatch proportion, you have to provide reference sequence, the mismatched proportion is color coded in the bar chart.

```
library(BSgenome.Hsapiens.UCSC.hg19)
bg <- BSgenome.Hsapiens.UCSC.hg19
p.mis <- autoplot(fl.bam, bsgenome = bg, which = wh, stat = "mismatch")
p.mis
```

To view overall estimated coverage distribution, please use method 'estiamte'. 'which' parameter also accept characters. And there is a hidden value called '..coverage..' to let you do simple transformation in aes().

```
autoplot(fl.bam, method = "estimate")
autoplot(fl.bam, method = "estimate", which = paste0("chr", 17:18), aes(y = log(..coverage..)))
```

2.5 Add a variants track

This track is supported by semantic zoom.

To view your variants file, you could

- Import it using package *VariantAnntoation* as *VCF* object, then use `autoplot`
- Convert it into *VRanges* object and use `autoplot`.
- Simply provide vcf file path in `autoplot()`.

```
library(VariantAnnotation)
fl.vcf <- system.file("extdata", "17-1409-CEU-brca1.vcf.bgz", package="biovizBase")
vcf <- readVcf(fl.vcf, "hg19")
vr <- as(vcf[, 1:3], "VRanges")
vr <- renameSeqlevels(vr, value = c("17" = "chr17"))
## small region contains data
gr17 <- GRanges("chr17", IRanges(41234400, 41234530))
p.vr <- autoplot(vr, which = wh)
## none geom
p.vr
## rect geom
p.vr + xlim(gr17)
## text geom
p.vr + xlim(gr17) + zoom()
```

You can simply overide geom

```
autoplot(vr, which = wh, geom = "rect", arrow = FALSE)
```

2.6 Building your tracks

```
## tks <- tracks(p.ideo, mismatch = p.mis, dbSNP = p.vr, ref = p.bs, gene = p.txdb)
## tks <- tracks(fl.bam, fl.vcf, bs, Homo.sapiens) ## default ideo = FALSE, turned on
## tks <- tracks(fl.bam, fl.vcf, bs, Homo.sapiens, ideo = TRUE)
## tks + xlim(gr17)
gr17 <- GRanges("chr17", IRanges(41234415, 41234569))
tks <- tracks(p.ideo, mismatch = p.mis, dbSNP = p.vr, ref = p.bg, gene = p.txdb,
              heights = c(2, 3, 3, 1, 4)) + xlim(gr17) + theme_tracks_sunset()
tks
```

Chapter 3

Simple navigation

We try to provide a simple navigation API for your plot, so you could zoom in and zoom out, or go through view chunks one by one.

- `zoom`: put a factor inside and you can zoom in or zoom out
- `nextView`: switch to next view
- `prevView`: switch to previous view

Navigation function also works for tracks plot too.

```
## zoom in
tks + zoom()
```

Try following command yourself.

```
## zoom in with scale
p.txdb + zoom(1/8)
## zoom out
p.txdb + zoom(2)
## next view page
p.txdb + nextView()
## previous view page
p.txdb + prevView()
```

Don't forget `xlim` accept *GRanges* object (single row), so you could simply prepare a *GRanges* to store the region of interests and go through them one by one.

Chapter 4

Overview plots

Overview is a good way to show all events at the same time, give overall summary statiics for the whole genome.

In this chapter, we will introcue three different layouts that are used a lots in genomic data visualization.

4.1 how to make circular plots

4.1.1 Introduction

Circular view is a special layout in *ggbio* , this idea has been implemented in many different software, for example, the *Circos* project. However, we keep the grammar of graphics for users, so mapping varialbes to aesthetics is very easy, *ggbio* leverage the data structure defiend in *Bioconductor* to make this process as simple as possible.

4.1.2 Buidling circular plot layer by layer

Ok, let's start to process some raw data to the format we want. The data used in this study is from this a paper[1]. In this tutorial, We are going to

1. Visualize somatic mutation as segment.
2. Visualize inter,intro-chromosome rearrangement as links.
3. Visualize mutation score as point tracks with grid-background.
4. Add scale and ticks and labels.
5. To arrange multiple plots and legend. create multiple sample comparison.

All the raw data processed and stored in *GRanges* ready for use, you can simply load the sample data from *biovizBase*

```
data("CRC", package  = "biovizBase")
```

`layout_circle` is depreicated, because you have to set up radius and trackWidth manually with this function for creating circular plot.

We now present the new `circle` function, it accepts `Granges` object, and users don't have to specify radius, track width, you just add them one by one, it will be automatically created from innter circle to outside, unless you specify `trackWidth` and `radius` manually. To change default radius and trackWidth for all tracks, you simply put them in `ggbio` function.

- rule of thumb `seqlengths`, `seqlevels` and chromosomes names should be exactly the same.
- to use `circle`, you have to use `ggbio` constructor at the beginning instead of `ggplot`.

[1] http://www.nature.com/ng/journal/v43/n10/full/ng.936.html

You can use `autoplot` to create single track easily like

```
head(hg19sub)
autoplot(hg19sub, layout = "circle", fill  = "gray70")
```

Hoever, the low level `circle` function leave you more flexibility to build circular plot one by one. Let's start to add tracks one by one.

Let's use the same data to create ideogram, label and scale track, it layouts the circle by the order you created from inside to outside.

```
p <- ggbio() + circle(hg19sub, geom = "ideo", fill = "gray70") +
    circle(hg19sub, geom = "scale", size = 2) +
  circle(hg19sub, geom = "text", aes(label = seqnames), vjust = 0, size = 3)
p
```

To simply override the setting, you can do it globally in `ggbio` function or individually `circle` function by specifying parametters `trackWidth` and `radius`, you can also specify the global settin for buffer in between in `ggbio` like example below.

```
p <- ggbio(trackWidth = 10, buffer = 0, radius = 10) + circle(hg19sub, geom = "ideo", fill = "gray70") +
    circle(hg19sub, geom = "scale", size = 2) +
  circle(hg19sub, geom = "text", aes(label = seqnames), vjust = 0, size = 3)
p
```

Then we add a "rectangle" track to show somatic mutation, this will looks like vertical segments.

```
head(mut.gr)
p <- ggbio() + circle(mut.gr, geom = "rect", color = "steelblue") +
    circle(hg19sub, geom = "ideo", fill = "gray70") +
    circle(hg19sub, geom = "scale", size = 2) +
  circle(hg19sub, geom = "text", aes(label = seqnames), vjust = 0, size = 3)
p
```

Next, we need to add some "links" to show the rearrangement, of course, links can be used to map any kind of association between two or more different locations to indicate relationships like copies or fusions. To create a suitable structure to plot, please use another *GRanges* to represent the end of the links, and stored as elementMetadata for the "start point" *GRanges*. Here we named it as "to.gr" and will be used later.

```
head(crc.gr)
```

Here in this example, we use "intrachromosomal" to label rearrangement within the same chromosomes and use "interchromosomal" to label rearrangement in different chromosomes.

Get subset of links data for only one sample "CRC1"

```
gr.crc1 <- crc.gr[values(crc.gr)$individual == "CRC-1"]
```

Ok, add a "point" track with grid background for rearrangement data and map 'y' to variable "score", map 'size' to variable "tumreads", rescale the size to a proper size range.

```
## manually specify radius
p <- p + circle(gr.crc1, geom = "point", aes(y = score, size = tumreads),
                color = "red", grid = TRUE, radius = 30) + scale_size(range = c(1, 2.5))
p
```

Finally, let's add links and map color to rearrangement types. Remember you need to specify 'linked.to' parameter to the column that contain end point of the data.

```
## specify radius manually
p <- p + circle(gr.crc1, geom = "link", linked.to = "to.gr", aes(color = rearrangements),
```

```
                        radius = 23)
p
```

All those code could be simply constructed by following code

```
p <- ggbio() +
   circle(gr.crc1, geom = "link", linked.to = "to.gr", aes(color = rearrangements)) +
  circle(gr.crc1, geom = "point", aes(y = score, size = tumreads),
                color = "red", grid = TRUE) + scale_size(range = c(1, 2.5)) +
  circle(mut.gr, geom = "rect", color = "steelblue") +
    circle(hg19sub, geom = "ideo", fill = "gray70") +
    circle(hg19sub, geom = "scale", size = 2) +
  circle(hg19sub, geom = "text", aes(label = seqnames), vjust = 0, size = 3)
p
```

4.1.3 Complex arragnment of plots

In this step, we are going to make multiple sample comparison, this may require some knowledge about package *grid* and *gridExtra*. We will introduce a more easy way to combine your graphics later after this.

We just want 9 single circular plots put together in one page, since we cannot keep too many tracks, we only keep ideogram and links. Here is one sample.

```
grl <- split(crc.gr, values(crc.gr)$individual)
## need "unit", load grid
library(grid)
crc.lst <- lapply(grl, function(gr.cur){
  print(unique(as.character(values(gr.cur)$individual)))
  cols <- RColorBrewer::brewer.pal(3, "Set2")[2:1]
  names(cols) <- c("interchromosomal", "intrachromosomal")
  p <- ggbio() + circle(gr.cur, geom = "link", linked.to = "to.gr",
                        aes(color = rearrangements)) +
                 circle(hg19sub, geom = "ideo",
                        color = "gray70", fill = "gray70") +
                 scale_color_manual(values = cols)  +
                 labs(title = (unique(values(gr.cur)$individual))) +
                 theme(plot.margin = unit(rep(0, 4), "lines"))
})
```

We wrap the function in grid level to a more user-friendly high level function, called `arrangeGrobByParsingLegend`. You can pass your ggplot2 graphics to this function , specify the legend you want to keep on the right, you can also specify the column/row numbers. Here we assume all plots we have passed follows the same color scale and have the same legend, so we only have to keep one legend on the right.

```
arrangeGrobByParsingLegend(crc.lst, widths = c(4, 1), legend.idx = 1, ncol = 3)
```

4.2 How to make grandlinear plots

4.2.1 Introduction

Let's use a subset of *PLINK* output (https://github.com/stephenturner/qqman/blob/master/plink.assoc.txt.gz) as our example test data.

```
snp <- read.table(system.file("extdata", "plink.assoc.sub.txt", package = "biovizBase"),
                  header = TRUE)
require(biovizBase)
gr.snp <- transformDfToGr(snp, seqnames = "CHR", start = "BP", width = 1)
head(gr.snp)
## change the seqname order
require(GenomicRanges)
gr.snp <- keepSeqlevels(gr.snp, as.character(1:22))
seqlengths(gr.snp)
## need to assign seqlengths
data(ideoCyto, package = "biovizBase")
seqlengths(gr.snp) <- as.numeric(seqlengths(ideoCyto$hg18)[1:22])
## remove missing
gr.snp <- gr.snp[!is.na(gr.snp$P)]
## transform pvalue
values(gr.snp)$pvalue <- -log10(values(gr.snp)$P)
head(gr.snp)
## done
```

The data is ready, we need to pay attention

- if seqlengths is missing, we use data range, so the chromosome length is not accurate
- use seqlevel to control order of chromosome

4.2.2 Corrdinate genome

In `autoplot`, argument `coord` is just used to transform the data, after that, you can use it as common `GRanges`, all other geom/stat works for it.

```
autoplot(gr.snp, geom = "point", coord = "genome", aes(y = pvalue))
```

However, we recommend you to use more powerful function `plotGrandLinear` to generate manhattan plot introduced in next section.

4.2.3 Convenient `plotGrandLinear` function

For *Manhattan plot*, we have a function called `plotGrandLinear`. aes(y =) is required to indicate the y value, e.g. p-value.

Color mapping is automatically figured out by *ggbio* following the rules

- if `color` present in `aes()`, like `aes(color = seqnames)`, it will assume it's mapping to data column called 'seqnames'.
- if `color` is not wrapped in `aes()`, then this function will **recylcle** them to all chromosomes.
- if `color` is single character representing color, then just use one arbitrary color.

Let's test some examples for controling colors.

```
plotGrandLinear(gr.snp, aes(y = pvalue), color = c("#7fc97f", "#fdc086"))
```

Let's add a cutoff line

```
plotGrandLinear(gr.snp, aes(y = pvalue), color = c("#7fc97f", "#fdc086"),
                cutoff = 3, cutoff.color = "blue", cutoff.size = 0.2)
```

Sometimes you use color to mapping other varibles so you may need a different to separate chromosomes.

```
plotGrandLinear(gr.snp, aes(y = pvalue, color = OR), spaceline = TRUE, legend = TRUE)
```

4.2.4 How to highlight some points?

You can provide a highlight *GRanges*, and each row highlights a set of overlaped snps, and labeled by rownames or certain columns, there is more control in the function as parameters, with prefix highlight.*, so you could control color, label size and color, etc.

```
gro <- GRanges(c("1", "11"), IRanges(c(100, 2e6), width = 5e7))
names(gro) <- c("group1", "group2")
plotGrandLinear(gr.snp, aes(y = pvalue), highlight.gr = gro)
```

4.3 How to make stacked karyogram overview plots

4.3.1 Introduction

A karyotype is the number and appearance of chromosomes in the nucleus of a eukaryotic cell[2]. It's one kind of overview when we want to show distribution of certain events on the genome, for example, binding sites for certain protein, even compare them across samples as example shows in this section.

`GRanges` and `Seqinfo` objects are an ideal container for storing data needed for karyogram plot. Here is the strategy we used for generating ideogram templates.

- Althouth `seqlengths` is not required, it's highly recommended for plotting karyogram. If a `GRanges` object contains `seqlengths`, we know exactly how long each chromosome is, and will use this information to plot genome space, particularly we plot all levels included in it, **NOT JUST** data space.
- If a `GRanges` has no `seqlengths`, we will issue a warning and try to estimate the chromosome lengths from data included. This is **NOT** accurate most time, so please pay attention to what you are going to visualize and make sure set `seqlengths` before hand.

4.3.2 Create karyogram temlate

Let's first introduce how to use `autoplot` to generate karyogram graphic.

The most easy one is to just plot Seqinfo by using `autoplot`, if your *GRanges* object has seqinfo with seqlengths information. Then you add data layer later.

```
data(ideoCyto, package = "biovizBase")
autoplot(seqinfo(ideoCyto$hg19), layout = "karyogram")
```

To show cytoband, your data need to have cytoband information, we stored some data for you, including *hg19, hg18, mm10, mm9*.

```
## turn on cytoband if it exists
biovizBase::isIdeogram(ideoCyto$hg19)
autoplot(ideoCyto$hg19, layout = "karyogram", cytoband = TRUE)
```

To change order or only show a subset of the karyogram, you have to manipulate `seqlevels`, please check out manual for `keepSeqlevels, seqlevels` in *GenomicRanges* package for more information. Or you could read the example below.

[2] http://en.wikipedia.org/wiki/Karyotype

4.3.3 Add data on karyogram layout

If you have single data set stored as *GRanges* to show on a karyogram layout, `autoplot` function is enough for you to plot the data on it.

We use a default data in package *biovizBase*, which is a subset of RNA editing set in human. The data involved in this `GRanges` is sparse, so we cannot simply use it to make karyogram template, otherwise, the estimated chromosome lengths will be very rough and inaccurate. So what we need to do first is to *add seglength information to this object.*

```
data(darned_hg19_subset500, package = "biovizBase")
dn <- darned_hg19_subset500
library(GenomicRanges)
seqlengths(dn)
## add seqlengths
## we have seqlegnths information in another data set
seqlengths(dn) <- seqlengths(ideoCyto$hg19)[names(seqlengths(dn))]
## then we change order
dn <- keepSeqlevels(dn, paste0("chr", c(1:22, "X")))
seqlengths(dn)
autoplot(dn, layout = "karyogram")
```

Then we take one step further, the power of *ggplot2* or *ggbio* is the flexible multivariate data mapping ability in graphics, make data exploration much more convenient. In the following example, we are trying to map a categorical variable 'exReg' to color, this variable is included in the data, and have three levels, '3' indicate 3' utr, '5' means 5' utr and 'C' means coding region. We have some missing values indicated as `NA`, in default, it's going to be shown in gray color, and keep in mind, since the basic geom(geometric object) is rectangle, and genome space is very large, so change both color/fill color of the rectangle to specify both border and filled color is necessary to get the data shown as different color, otherwise if the region is too small, border color is going to override the fill color.

```
## since default is geom rectangle, even though it's looks like segment
## we still use both fill/color to map colors
autoplot(dn, layout = "karyogram", aes(color = exReg, fill = exReg))
```

Or you can set the missing value to particular color yo u want (NA values is not shown on the legend).

```
## since default is geom rectangle, even though it's looks like segment
## we still use both fill/color to map colors
autoplot(dn, layout = "karyogram", aes(color = exReg, fill = exReg), alpha  = 0.5) +
  scale_color_discrete(na.value = "brown")
```

Well, sometimes we have too many values, we want to separate them by groups and show them at diffent height, below is a hack for that purpose and in next section, we will introduce a more flexible and general way to add data layer by layer.

Template chromosome y limits is [0, 10], that's why this hack works

```
## let's remove the NA value
dn.nona <- dn[!is.na(dn$exReg)]
## compute levels based on categories
dn.nona$levels <- as.numeric(factor(dn.nona$exReg))
## do a trcik show them at different height
p.ylim <- autoplot(dn.nona, layout = "karyogram", aes(color = exReg, fill = exReg,
                                      ymin = (levels - 1) * 10/3,
                                      ymax = levels * 10 /3))
```

4.3.4 Add more data using layout_karyogram function

In this section, a lower level function `layout_karyogram` is going to be introduced. This is convenient API for constructing karyogram plot and adding more data layer by layer. Function `ggplot` is just to create blank object to add layer on.

You need to pay attention to

- when you add plots layer by layer, seqnames of different data must be the same to make sure the data are mapped to the same chromosome. For example, if you name chromosome following schema like *chr1* and use just number *1* to name other data, they will be treated as different chromosomes.
- cannot use the same aesthetics mapping multiple time for different data. For example, if you have used aes(color =), for one data, you cannot use aes(color =) anymore for mapping variables from other add-on data, this is currently not allowed in *ggplot2*, even though you expect multiple color legend shows up, this is going to confuse people which is which. HOWEVER, `color` or `fill` without `aes()` wrap around, is allowed for any track, it's set single arbitrary color.
- Default rectangle y range is [0, 10], so when you add on more data layer by layer on existing graphics, you can use `ylim` to control how to normalize your data and plot it relative to chromosome space. For example, with default, chromosome space is plotted between y [0, 10], if you use `ylim = c(10 , 20)`, you will stack data right above each chromosomes and with equal width. For geom like 'point', which you need to specify 'y' value in `aes()`, we will add 5% margin on top and at bottom of that track.

Many times we overlay different datas sets, so let's break down the previous samples into 4 groups and treat them as different data and build them layer by layer, assign the color by hand. You could use ylim to control where they are ploted.

```
## prepare the data
dn3 <- dn.nona[dn.nona$exReg == '3']
dn5 <- dn.nona[dn.nona$exReg == '5']
dnC <- dn.nona[dn.nona$exReg == 'C']
dn.na <- dn[is.na(dn$exReg)]
## now we have 4 different data sets
autoplot(seqinfo(dn3), layout = "karyogram") +
  layout_karyogram(data = dn3, geom = "rect", ylim = c(0, 10/3), color = "#7fc97f") +
  layout_karyogram(data = dn5, geom = "rect", ylim = c(10/3, 10/3*2), color = "#beaed4") +
  layout_karyogram(data = dnC, geom = "rect", ylim = c(10/3*2, 10), color = "#fdc086") +
  layout_karyogram(data = dn.na, geom = "rect", ylim = c(10, 10/3*4), color = "brown")
```

What's more, you could even chagne the geom for those data

```
dn$pvalue <- runif(length(dn)) * 10
p <- autoplot(seqinfo(dn)) + layout_karyogram(dn, aes(x = start, y = pvalue),
                     geom = "point", color = "#fdc086")
p
```

4.3.5 More flexible layout of karyogram

```
p.ylim + facet_wrap(~seqnames)
```

Chapter 5

Link ranges to your data

Plot GRanges object structure and linked to a even spaced paralell coordinates plot which represting the data in elementeMetadata.

```
library(TxDb.Hsapiens.UCSC.hg19.knownGene)
library(ggbio)
data(genesymbol, package = "biovizBase")
txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene
model <- exonsBy(txdb, by = "tx")
model17 <- subsetByOverlaps(model, genesymbol["RBM17"])
exons <- exons(txdb)
exon17 <- subsetByOverlaps(exons, genesymbol["RBM17"])
## reduce to make sure there is no overlap
## just for example
exon.new <- reduce(exon17)
## suppose
values(exon.new)$sample1 <- rnorm(length(exon.new), 10, 3)
values(exon.new)$sample2 <- rnorm(length(exon.new), 10, 10)
values(exon.new)$score <- rnorm(length(exon.new))
values(exon.new)$significant <- sample(c(TRUE,FALSE), size = length(exon.new),replace = TRUE)
## data ready
exon.new
```

Make the plots, you can pass a list of annotation tracks too.

```
p17 <- autoplot(txdb, genesymbol["RBM17"])
plotRangesLinkedToData(exon.new, stat.y = c("sample1", "sample2"), annotation = list(p17))
```

For more information, check the manual.

Chapter 6

Miscellaneous

Every plot object produced by *ggplot2* is essentially a *ggplot2* object, so you could use all the tricks you know with *ggplot2* on *ggbio* plots too, including scales, colors, themes, etc.

6.1 Themes

In *ggbio*, we developed some more themes to make things easier.

6.1.1 Plot theme

Plot level themes are like any other themes defined in *ggplot2*, simply apply it to a plot.

```
p.txdb
p.txdb + theme_alignment()
p.txdb + theme_clear()
p.txdb + theme_null()
```

When you have multiple chromosomes encoded in seqnames, you could use theme_genome to make a 'fake' linear view of genome coordinates quickly by applying this theme, because it's not equal to chromosome lengths, it's simply

```
library(GenomicRanges)
set.seed(1)
N <- 100
gr <- GRanges(seqnames = sample(c("chr1", "chr2", "chr3"),
                      size = N, replace = TRUE),
              IRanges(start = sample(1:300, size = N, replace = TRUE),
                      width = sample(70:75, size = N,replace = TRUE)),
              strand = sample(c("+", "-"), size = N, replace = TRUE),
              value = rnorm(N, 10, 3), score = rnorm(N, 100, 30),
              sample = sample(c("Normal", "Tumor"),
                size = N, replace = TRUE),
              pair = sample(letters, size = N,
                replace = TRUE))
seqlengths(gr) <- c(400, 1000, 500)
autoplot(gr)
autoplot(gr) + theme_genome()
```

6.1.2 Track theme

Track level themes are more complex, it controls whole looking of the tracks, it's essentially a theme object with some attributes controlling the tracks appearance.

See how we make a template, you could customize in the same way

```
theme_tracks_sunset
```

The attributes you could control is basically passed to tracks() constructor, including

label.bg.color	character
label.bg.fill	character
label.text.color	character
label.text.cex	numeric
track.plot.color	characterORNULL
track.bg.color	characterORNULL
label.width	unit

Table 6.1: tracks attributes

Chapter 7

Session Information

```
sessionInfo()
```