Seminar III: R/Bioconductor

Leonardo Collado Torres
lcollado@lcg.unam.mx
Bachelor in Genomic Sciences
www.lcg.unam.mx/~lcollado/

August - December, 2009
Genomic Plots

Intro

GenomeGraphs

rtracklayer
About

- On this short class we’ll learn how to make some plots that are focused on *genomics*.
- The basic idea is a tool that enables you to visualize *tracks* of information.
Session packages

- Install commands:
  
  ```
  > install.packages("RMySQL")
  > source("http://bioconductor.org/biocLite.R")
  > biocLite("rtracklayer")
  > biocLite("humanStemCell")
  > biocLite("GenomeGraphs")
  ```
GenomeGraphs

- It uses grid graphics\(^1\) and works great with biomaRt.
- The syntax is different and longer from what we are used to.
- Much more flexible than other packages, and I find it to be more stable :)
- Who are the authors of the package?
- For more info, check this paper.

\(^1\)Just like lattice.
To start off, we’ll use the `gdPlot` function, which is the main one.

```r
library(GenomeGraphs)
```

```
? (gdPlot)
```

- What kind of object does it need as input?
- What determines the plotting order?
So we need to create a list with `gdObjects`.

How do we find them?
You can always look at the examples from the gdPlot help and find a few.

I would either browse the package help using:

```r
> help(package = GenomeGraphs)
```

Or thanks to some previous info, I know that the functions that create this kind of objects start with `make`. So we can use `apropos`.

```r
> apropos("make")
```
makeBaseTrack

- Lets create an object of class BaseTrack using `makeBaseTrack`.  
  ```r
  > args(makeBaseTrack)
  
  function (base, value, strand, segmentation, dp = NULL)
  NULL
  ```

- The first arguments are quite simple.
  1. `base` has the position values; the x coordinates.
  2. `value` is the analog for the y axis.
  3. `strand` is just a ”+” or ”-” character.

- Lets create a simple track for positions 1 to 100 with random log-normal values.
  ```r
  > makeBaseTrack(1:100, rlnorm(100))
  ```
### makeBaseTrack

Object of class 'BaseTrack':
- base position:
  [1] 1 2 3 4 5
- Values:
  [1] 0.8592530 1.8658355 0.4555190
  [4] 0.8784603 3.3690927

There are 95 more rows
- color = orange
- lty = solid
- lwd = 1
- size = 5
- type = p

\(^2\)Yes, it’s a long name :P
makeBaseTrack II

- The first lines print the head for base and value. The next ones inform us of the graphical parameters such as lwd (line width).
- Let's save our track into the object `a` assigning it to the positive strand.
  ```r
  > a <- makeBaseTrack(1:100, rlnorm(100),
  +                      strand = "+")
  ```
- Let's make the plot now :)
Simple gdPlot

> gdPlot(a)
gdPlot exercise

- Now create an object b using `makeBaseTrack` for the first 100 positions using random normal values and assign them to the negative strand.
- Then plot both a and b using `gdPlot`
Short solution

```r
> info <- list(makeBaseTrack(1:100, + rlnorm(100), strand = "+"), + makeBaseTrack(1:100, rnorm(100), + strand = "-"))
> gdPlot(info)
```
Short solution
What is the difference?

```r
> info2 <- list(makeBaseTrack(1:100, rlnorm(100), strand = ""),
+ makeBaseTrack(1:100, rnorm(100), strand = ""), makeGenomeAxis())
> gdPlot(info2)
```
What is the difference?
In GenomeGraphs, to change graphical arguments we need to use the `DisplayPars` function.

However, the arguments differ for every `gdObject`. So we need to check them before using them.

Let's go back to `makeBaseTrack` and change the color of the negative strand values.

```r
> b <- makeBaseTrack(1:100, rnorm(100),
+   strand = "-", dp = DisplayPars(color = "blue"))
```
Changing colors

```r
> gdPlot(list(a, b, makeGenomeAxis()))
```
Finding args

▶ In practice, it's better to find the arguments using `showDisplayOptions`

▶ For example:

```r
> showDisplayOptions("BaseTrack")

color = orange
lty = solid
lwd = 1
size = 5
type = p
```

▶ How many graphical arguments does the genome axis object have?
Names

- Say we want to add the strand names to our previous plots and to our axis.
- Any ideas? Remember that we are using a list.
gdPlot with names

> gdPlot(list(`+` = a, `−` = b, Base = makeGenomeAxis()))
Interaction with biomaRt

- GenomeGraphs can retrieve information from public databases using biomaRt.
- To do so, we use the function `makeGeneRegion`:
  ```r
  > args(makeGeneRegion)
  function (start, end, chromosome, strand, biomart, dp = NULL) {
    NULL
  }
  ```
  The `biomart` argument is a `mart` object. Let's create one:
  ```r
  > bsub <- useMart("bacterial_mart_54",
  + dataset = "bac_6_gene")
  ```
GeneRegion exercise

Using our bsub mart,

1. Create a GeneRegion object `c` with info from the genes from the bases 12000 to 20000 for the positive strand.
2. Create an object `d` for those on the negative strand.
3. Create a plot with the axis using `gdPlot`.

You will need to get the chromosome name. You might want to use `listAttributes` and/or do a simple `getBM`, or check on web biomart, or guess it ;)


Step by step solution

- To find the chromosome name, I simply checked the attributes list.

```r
> head(listAttributes(bsub))

      name
1 ensembl_gene_id
2 ensembl_transcript_id
3 ensembl_peptide_id
4 description
5 chromosome_name
6 start_position

   description
1      Ensembl Gene ID
2      Ensembl Transcript ID
3      Ensembl Protein ID
4      Description
5      Chromosome/plasmid
6      Gene Start (bp)
```
Step by step solution

► Of course, we can verify this with by using `getBM`:

```r
> res <- getBM(attributes = c("chromosome_name"),
+ filters = c("start", "end"),
+ values = list("1", "1000"),
+ mart = bsub)
> res

   chromosome_name
1         Chromosome
```

► Then we can get the info for the genes on the positive strand:

```r
> c <- makeGeneRegion(12000, 20000,
+ chromosome = "Chromosome",
+ strand = "+", biomart = bsub)
```
### Step by step solution

- For the `d` object, we use nearly the same code and we finish the job with `gdPlot`:

  ```r
  > d <- makeGeneRegion(12000, 20000,
  +                     chromosome = "Chromosome",
  +                     strand = "-", biomart = bsub)
  ```

\[I just modified one of the code lines we used on the `biomaRt` class.\]
Resulting plot

```r
> gdPlot(list(`+` = c, `-' = d, Bsub = makeGenomeAxis()))
```

![Plot diagram]

Bsub

<table>
<thead>
<tr>
<th></th>
<th>10000</th>
<th>12000</th>
<th>14000</th>
<th>16000</th>
<th>18000</th>
<th>20000</th>
<th>22000</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>11000</td>
<td>13000</td>
<td>15000</td>
<td>17000</td>
<td>19000</td>
<td>21000</td>
<td></td>
</tr>
</tbody>
</table>
Microarray data

- Lets make more complicated plots with microarray data from David et al.
- We’ll be using a different mart and the example dataset seqDataEx
  ```
  > mart <- useMart("ensembl", "scerevisiae_gene_ensembl"
  > data("seqDataEx")
  > head(seqDataEx$david)
  ```
Microarray data

<table>
<thead>
<tr>
<th>chr</th>
<th>location</th>
<th>strand</th>
<th>expr</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>-1</td>
<td>-0.20</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>1</td>
<td>0.61</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>-1</td>
<td>0.07</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>1</td>
<td>1.25</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>-1</td>
<td>-0.29</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>1</td>
<td>0.61</td>
</tr>
</tbody>
</table>

- Let's take a peak at chromosome IV:
Basic plot

```r
> gdPlot(makeGeneRegion(10000, 50000,
+          chr = "IV", strand = "+", biomart = mart),
+          10000, 50000)
```
Basic plot
Gene names

- Lets add the gene names (plot ids).
  
  ```R
  > showDisplayOptions("GeneRegion")
  ```

- What are the options to:
  1. Add the gene names?
  2. Change the rotation angle of the names?
  3. Change the letter size?
  4. Set the color? For example, to black.

- Re-make the previous plot with the names parallel to the $x$ axis, letter size 0.5 instead of 1, and in black.
Basic plot with names

```r
> gdPlot(makeGeneRegion(10000, 50000,
+   chr = "IV", strand = "+", biomart = mart,
+   dp = DisplayPars(plotId = TRUE,
+     idRotation = 0, cex = 0.5,
+     idColor = "black")), 10000,
+   50000)
```
Basic plot with names
Not much, right?

Let's make one with the microarray data using `makeGenericArray`:

```r
> args(makeGenericArray)
function (intensity, probeStart, probeEnd, segmentation, dp = NULL) NULL
```

For less typing, let's save the data into a shorter object:

```r
> david <- seqDataEx$david
```

We did a `head` earlier on the data, so let's use `location` for `probeStart` and `expr` for `intensity` arguments respectively.

As a short parenthesis, look at this neat trick:

```r
> head(david[, "expr", drop = FALSE])
```
GenericArray

expr
1  -0.20
2   0.61
3   0.07
4   1.25
5  -0.29
6   0.61

> head(david[, "expr", drop = TRUE])

   1    2    3    4    5    6
-0.20  0.61  0.07  1.25 -0.29  0.61

▶ Neat eh? :) Let's use `makeGenericArray` now:
> e <- makeGenericArray(david[, "expr",
+   drop = FALSE], david[, "location"])
GenericArray plot

> gdPlot(list(e, makeGenomeAxis()))
Something... complicated :)

Now, let's make a complicated plot

- One GeneRegion for each strand
- One GenericArray for each strand
Code:

```r
> df <- as.data.frame(seqDataEx$david)
> lst <- lapply(c("+", "-"), function(s) {
  + a <- as.matrix(subset(df, strand ==
  + ifelse(s == "+", 1, -1)))
  + c(makeGenericArray(a[, "expr",
    + drop = FALSE], a[, "location"]),
    + makeGeneRegion(start = min(df[, 
      "location"]), end = max(df[, 
        "location"]), chr = "IV",
    + strand = s, biomart = mart,
    + dp = DisplayPars(plotId = TRUE,
      + idRotation = 0,
      + cex = 0.5, idColor = "black")))
```
Code:

```r
+ })
> yeastLst <- c(unlist(lst), makeGenomeAxis())
```
A great plot!

> `gdPlot(yeastLst)`
Overlays

- We can also add some rectangles and text to highlight interesting parts of the plot.
- To do so, we use `makeRectangleOverlay` and `makeTextOverlay`:

  ```r
  > args(makeRectangleOverlay)
  function (start, end, region = NULL, coords = c("genomic", "absolute"),
            dp = NULL)
  NULL

  > args(makeTextOverlay)
  function (text, xpos, ypos, region = NULL, coords = c("genomic",
            "absolute"), dp = NULL)
  NULL
  ```
Rectangle overlay exercise

- Lets add a rectangle overlay to the previous plot.
- Which display argument enables us to make the rectangle semi-transparent? Use `showDisplayOptions`:

  ```
  > showDisplayOptions("RectangleOverlay")
  
  alpha = 0
color = black
fill = black
lty = solid
lwd = 1
  ```

- Add a rectangle overlay starting at 1301500, ending at 1302500, covering the first 2 panels and at exactly mid-transparency.
Solution :) 

```r
> ovlay <- makeRectangleOverlay(1301500, 1302500, region = c(1, 2), dp = DisplayPars(alpha = 0.5))
> gdPlot(yeastLst, overlays = c(ovlay))
```
Solution :)

[Graph showing data points and labeled genomic regions]
With text

- Now, let's add some text using `makeTextOverlay`

```r
> tovlay <- makeTextOverlay("SpecialRegion",
+ 1303500, 0.75, region = c(1,
+ 1), dp = DisplayPars(color = "black"))
```
End result

> gdPlot(yeastLst, overlays = c(ovlay, tovlay))
End result
For those of you who love splicing, `makeTranscript` will be most useful :)  

Let's take a look at gene ENSG00000168309:

```r
> args(makeTranscript)

function (id, type, biomart, dp = NULL)
NULL

> hMart <- useMart("ensembl", "hsapiens_gene_ensembl")
> trans <- makeTranscript("ENSG00000168309",
+   biomart = hMart)
```
Alternative splicing

\[ \text{gdPlot(list(trans, makeGenomeAxis()))} \]
Exons and gene models

▶ Visualizing data can be troublesome when you have mixed ranges. Say a small exon, then a large intron, a medium exon, etc.

▶ If you have exon microarray data, then `makeExonArray` and `makeGeneModel` will be useful to you :)⁴

> `args(makeExonArray)`

```r
function (intensity, probeStart, probeEnd, probeId, nProbes,
        displayProbesets = FALSE, dp = NULL)
NULL
```

> `args(makeGeneModel)`

```r
function (start, end, chromosome, dp = NULL)
NULL
```

▶ Here is an example using the `unrData` dataset:
Exons and gene models

```r
> data("unrData", package = "GenomeGraphs")
> class(unrData)
[1] "matrix"
> dim(unrData)
[1] 117  33
> head(unrPositions)
```
## Exons and gene models

<table>
<thead>
<tr>
<th>probesetId</th>
<th>chromosome</th>
<th>start</th>
<th>stop</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2429278</td>
<td>115061081</td>
<td>115061119</td>
</tr>
<tr>
<td>2</td>
<td>2429279</td>
<td>115061152</td>
<td>115061198</td>
</tr>
<tr>
<td>3</td>
<td>2429280</td>
<td>115061275</td>
<td>115061409</td>
</tr>
<tr>
<td>4</td>
<td>2429281</td>
<td>115061486</td>
<td>115061528</td>
</tr>
<tr>
<td>5</td>
<td>2429282</td>
<td>115061888</td>
<td>115062089</td>
</tr>
<tr>
<td>6</td>
<td>2429283</td>
<td>115062185</td>
<td>115062218</td>
</tr>
</tbody>
</table>
Exons and gene models

First we create the **exon track** which zooms into every exon, and then a **gene model** so we don’t lose the forest :)

```r
> exon <- makeExonArray(intensity = unrData, 
+ probeStart = unrPositions[, 3], probeEnd = unrPositions[, 4], 
+ probeId = as.character(unrPositions[, 1]), nProbes = unrNProbes, 
+ dp = DisplayPars(color = "blue", 
+ mapColor = "dodgerblue2"), 
+ displayProbesets = FALSE)
> geneModel <- makeGeneModel(start = unrPositions[, 3], end = unrPositions[, 4])
```

For the curious ones, you can make custom annotation tracks using `makeAnnotationTrack`. 
Example plot

```r
> gdPlot(list(exon, geneModel, makeGenomeAxis()))
```
Conclusions

▶ Fast! Which is great for a quick exploration of your data by regions.
▶ Once you get the basics, it’s easy to use :) 
▶ Very flexible!
▶ Has several handy functions for making genomic plots.
▶ Has the same limitations as other R plots.
Credits

Nearly all the GenomeGraphs examples and exercises are from James Bullard’s recent lab at BioC2009 available here. I modified some and expanded the explanations so it’d be easier to understand :)

---

Seminar III: R/Bioconductor
GenomeGraphs
I suppose that quite a few of you have heard about genome browsers, mainly the UCSC Genome Browser: http://genome.ucsc.edu/

It's probably the most popular web tool for exploring genomes, specially eukaryotic genomes.

Nowadays, they added quite a few nice tools and you can easily access data from a lot of information tracks: conservation, repeats, SNPs, etc.

Other tools are BLAT (similar to BLAST), a PCR in silico tool, ...
GBs

- Genome Browsers (GBs) are great for biologists. You don’t need to write any code to use it.
- It simplifies the process of gathering data for you (aka, getting the tracks).
- You don’t need any special software to use it, just a browser like Firefox.
- However, the formats are tedious to work with...
- If you have custom tracks, you’ll need to do lots of uploading.
rtracklayer

Michael Lawrence developed rtracklayer which lets you:

1. Import and export data to a GB (there are several formats).
2. Create tracks using IRanges. And of course, manipulate them :)
3. Browsing the data on the genome browser.

- However, I ran into problems when using the package recently :P
- It's not as easy to understand as GenomeGraphs.
- So why learn about it? Michael is VERY active on the mailing lists and I’m sure that he’ll fix whatever is wrong right now. Also, GBs are very popular so you might need to use them in the future.
To learn

- If you want to learn about the package, check the vignette and the R code file available at [here](#).
- Note that your computer might freeze on page 9 of the vignette file.
> sessionInfo()

R version 2.10.0 Under development (unstable) (2009-07-25 r48998)
i686-pc-linux-gnu

locale:
[1] LC_CTYPE=en_US.UTF-8
[2] LC_NUMERIC=C
[3] LC_TIME=en_US.UTF-8
[4] LC_COLLATE=en_US.UTF-8
[5] LC_MONETARY=C
[6] LC_MESSAGES=en_US.UTF-8
[7] LC_PAPER=en_US.UTF-8
[8] LC_NAME=C
[9] LC_ADDRESS=C
[10] LC_TELEPHONE=C
[12] LC_IDENTIFICATION=C

attached base packages:
SessionInfo

[1] grid stats graphics
[4] grDevices utils datasets
[7] methods base

other attached packages:
[1] GenomeGraphs_1.5.1
[2] biomaRt_2.1.0

loaded via a namespace (and not attached):
[1] RCurl_0.98-1 tools_2.10.0
[3] XML_2.6-0