R and Stats - PDCB topic
Genome Graphs

LCG Leonardo Collado Torres
lcollado@wintergenomics.com – lcollado@ibt.unam.mx

March 31st, 2011

BioC Overview

GenomeGraphs

Quick review

- Main site: `http://www.bioconductor.org/`
- Finding packages: BiocViews and/or Workflows
- Installing BioC:

```
> source("http://bioconductor.org/biocLite.R")
> biocLite()
```

- Installing a specific BioC package:

```
> source("http://bioconductor.org/biocLite.R")
> biocLite("PkgName")
```

- Browsing your local Vignettes:

```
> browseVignettes(package = "PkgName")
```

- So, how many vignettes do you have locally?

Vignettes

- Just use the browseVignettes function without any arguments:

  `> browseVignettes()`
- The result is a html page with links to all the PDFs and R files.
- The whole idea behind a vignette is to exemplify how you can combine multiple functions from the same package.

Experimental Data Pkgs

- Using the BioCViews, which experimental data packages are related to high throughput sequencing?
- Having a broader diversity of exp. data pkgs has been one of the goals for some time: **you can contribute!**

Session package

- Install commands:

```
> source("http://bioconductor.org/biocLite.R")
> 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.

gdPlot

- To start off, we'll use the gdPlot function, which is the main one.

  ```
  > library(GenomeGraphs)
  ```

  ```
  > `?`(gdPlot)
  ```

- What kind of object does it need as input?
- What determines the plotting order?

gdObjects

- So we need to create a `list` with gdObjects.
- How do we find them?

gdObjects II

- You can always look at the examples from the `gdPlot` help and find a few.
- I would either browse the package help using:

  ```
  > 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.

  ```
  > apropos("make")
  ```

makeBaseTrack

- Lets create an object of class `BaseTrack` using makeBaseTrack[2].

```
> args(makeBaseTrack)

function (base, value, strand, trackOverlay, 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.

```
> makeBaseTrack(1:100, rlnorm(100))
```

makeBaseTrack

```
Object of class 'BaseTrack':
 base position:
[1] 1 2 3 4 5
Values:
[1] 0.8173010 2.0153349 0.5943839
[4] 3.6994237 0.8042467

 There are 95 more rowscolor  =  orange
lty  =  solid
lwd  =  1
size  =  5
type  =  p
```

[2] Yes, it's a long name :P

makeBaseTrack II

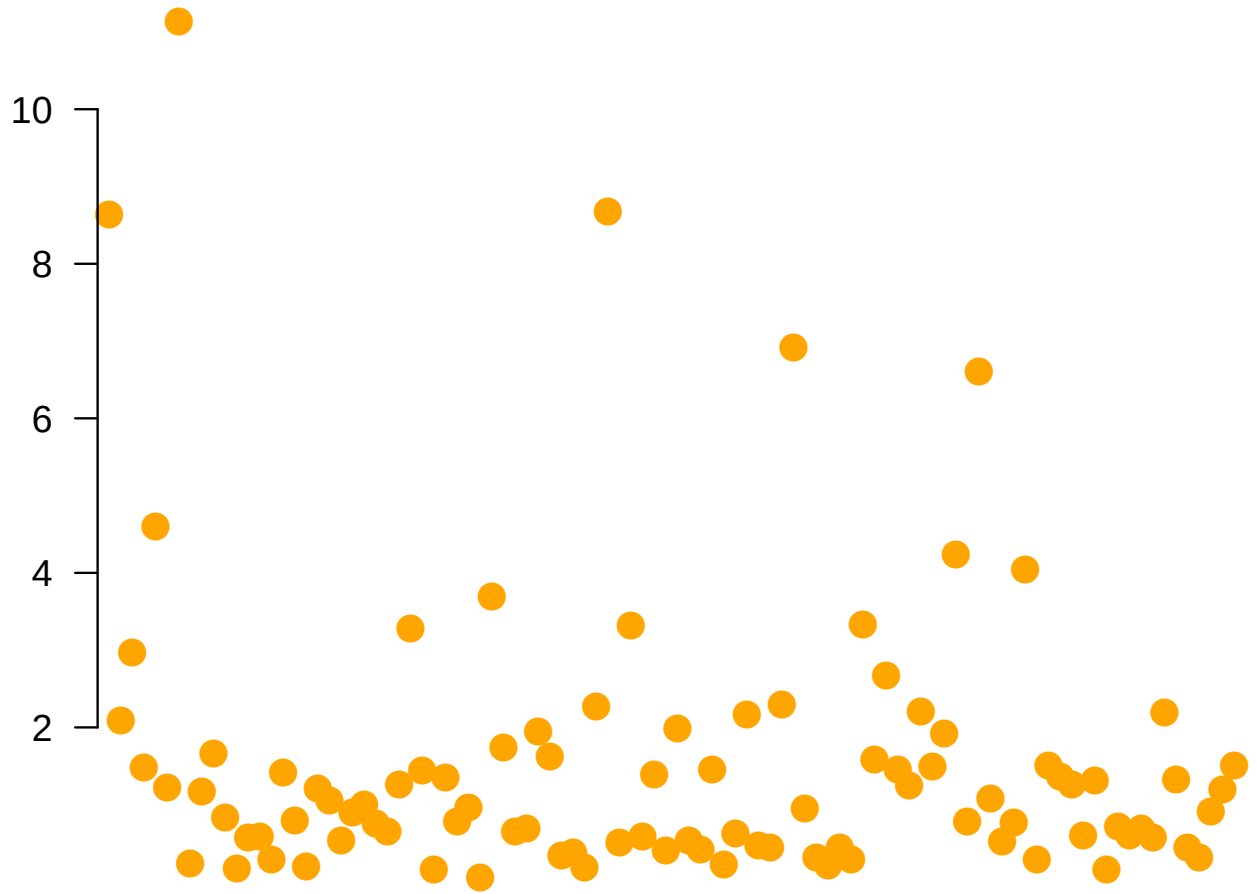
- The first lines print the `head` for `base` and `value`. The next ones inform us of the graphical parameters such as `lwd` (line width).
- Lets save our track into the object `a` assigning it to the positive strand.

```
> a <- makeBaseTrack(1:100, rlnorm(100),
+      strand = "+")
```

- Lets make the plot now :)

Simple gdPlot

```
> gdPlot(a)
```



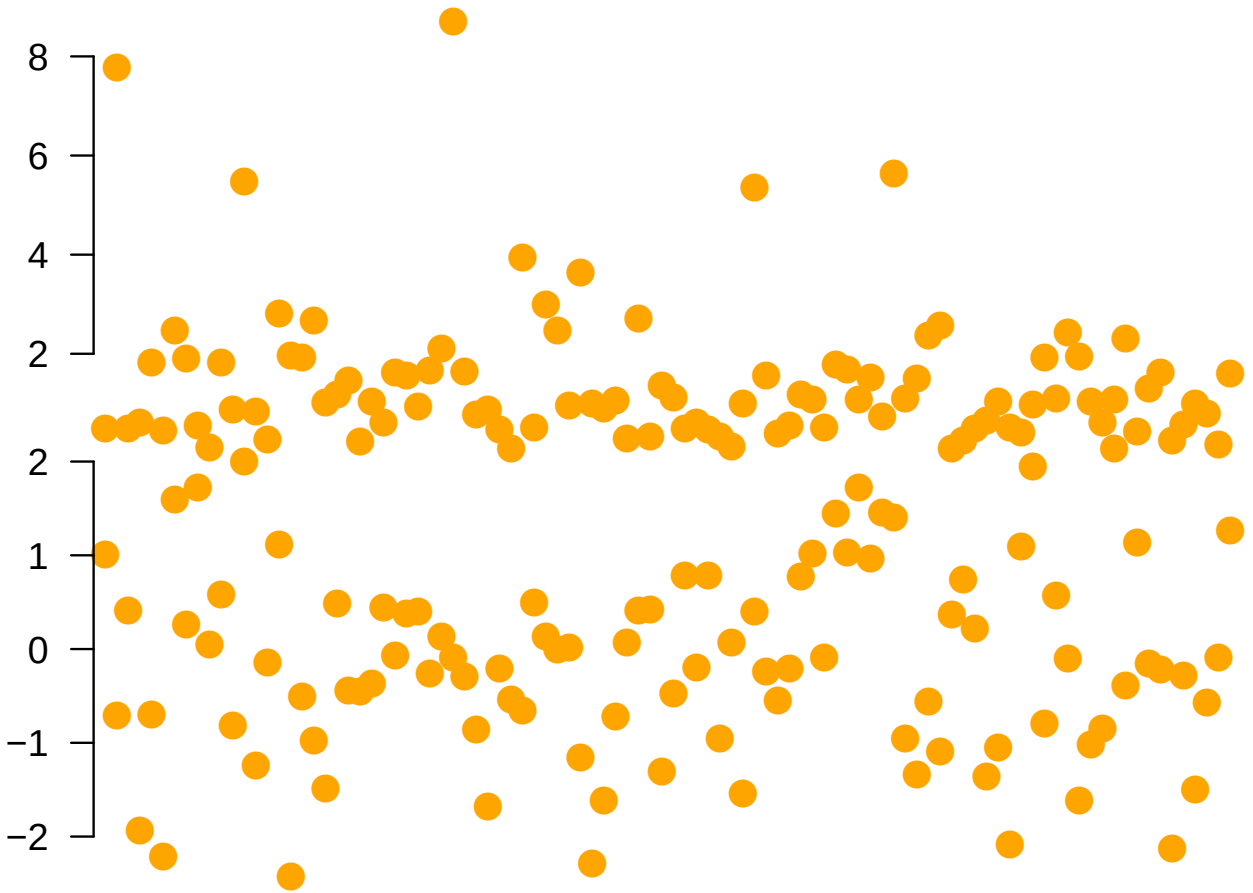
gdPlot exercise

- Now create and 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

```
> info <- list(makeBaseTrack(1:100,
+     rlnorm(100), strand = "+"),
+     makeBaseTrack(1:100, rnorm(100),
+         strand = "-"))
> gdPlot(info)
```

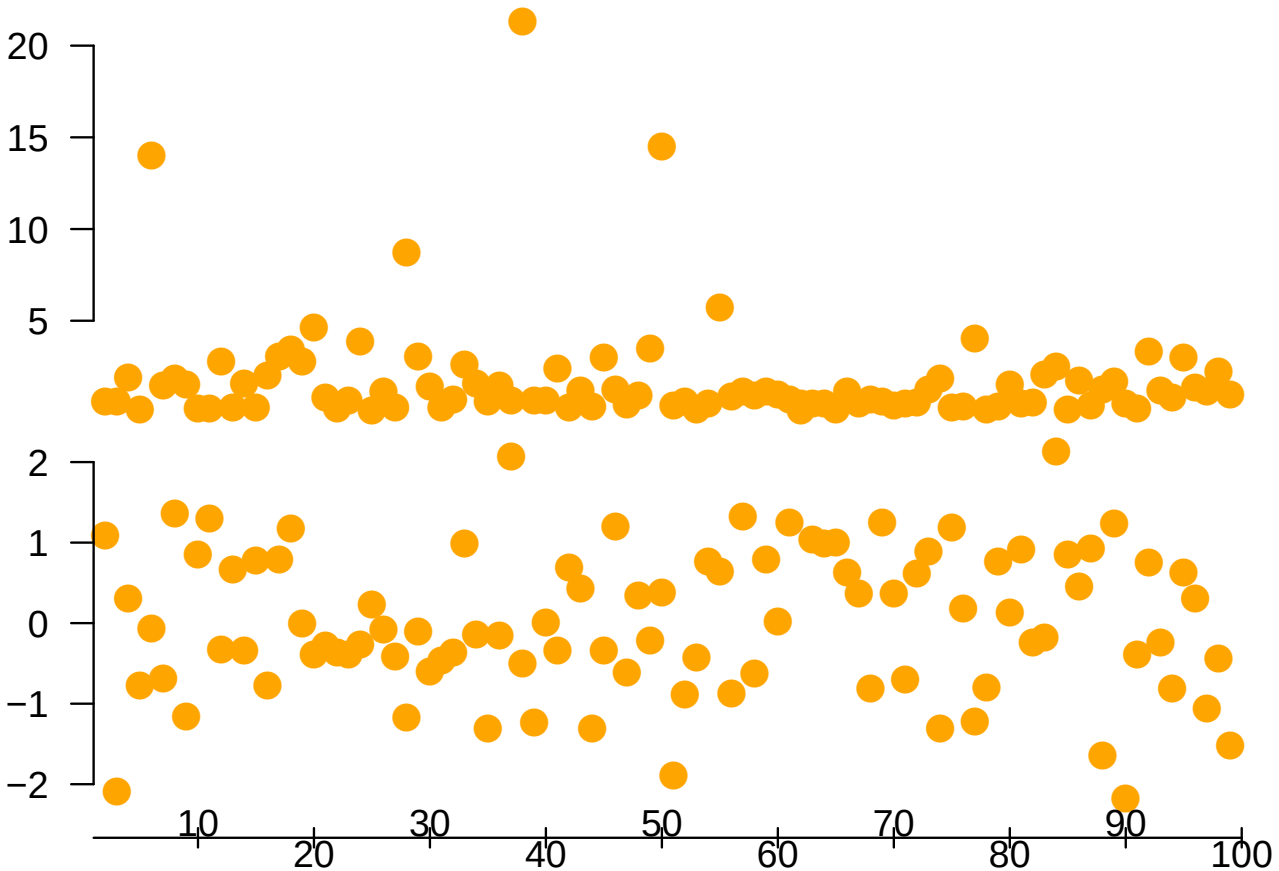
Short solution



What is the difference?

```
> info2 <- list(makeBaseTrack(1:100,
+     rlnorm(100), strand = "+"),
+     makeBaseTrack(1:100, rnorm(100),
+         strand = "-"), makeGenomeAxis())
> gdPlot(info2)
```

What is the difference?



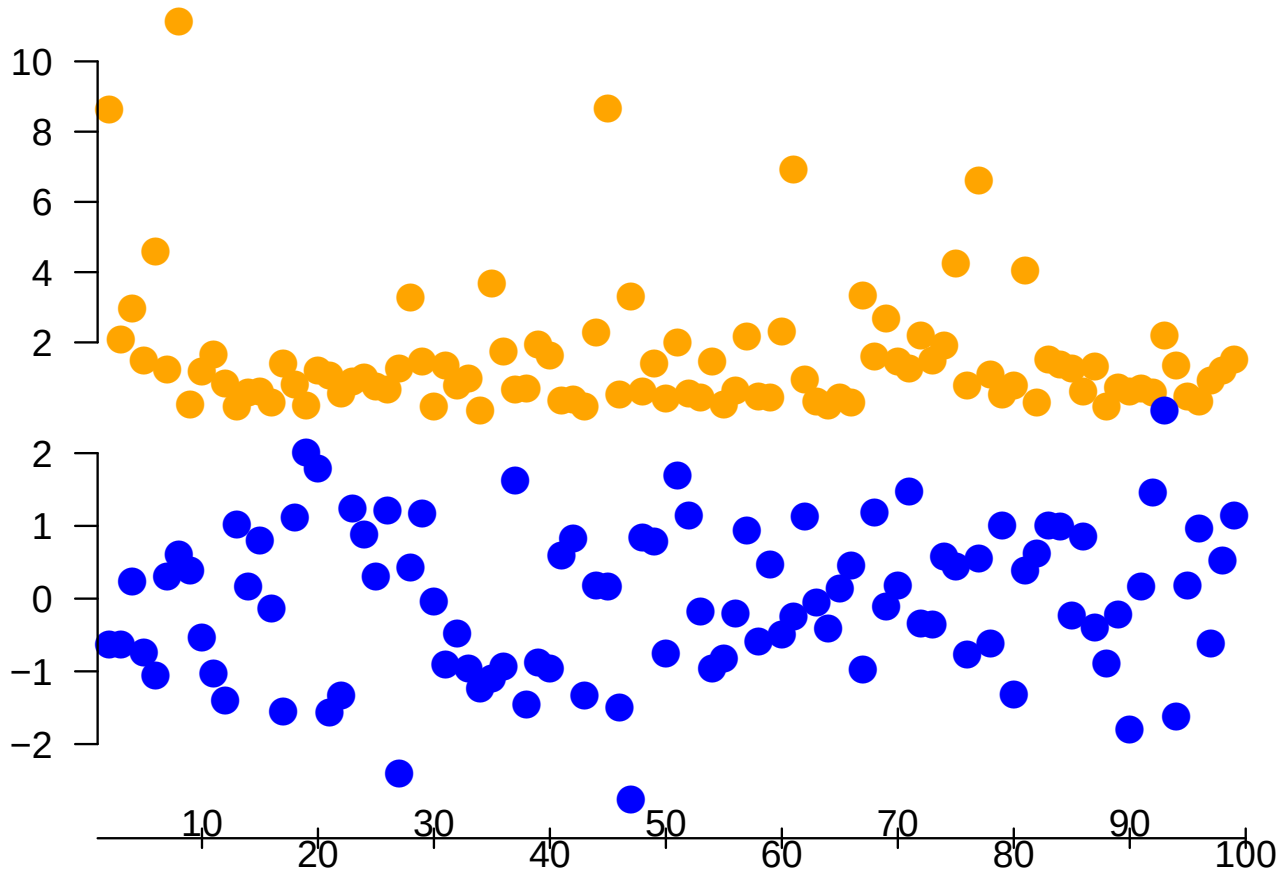
DisplayPars

- 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.
- Lets go back to `makeBaseTrack` and change the color of the negative strand values.

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

Changing colors

```
> gdPlot(list(a, b, makeGenomeAxis()))
```



Finding args

- In practice, its better to find the arguments using showDisplayOptions
- For example:

```
> showDisplayOptions("BaseTrack")

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

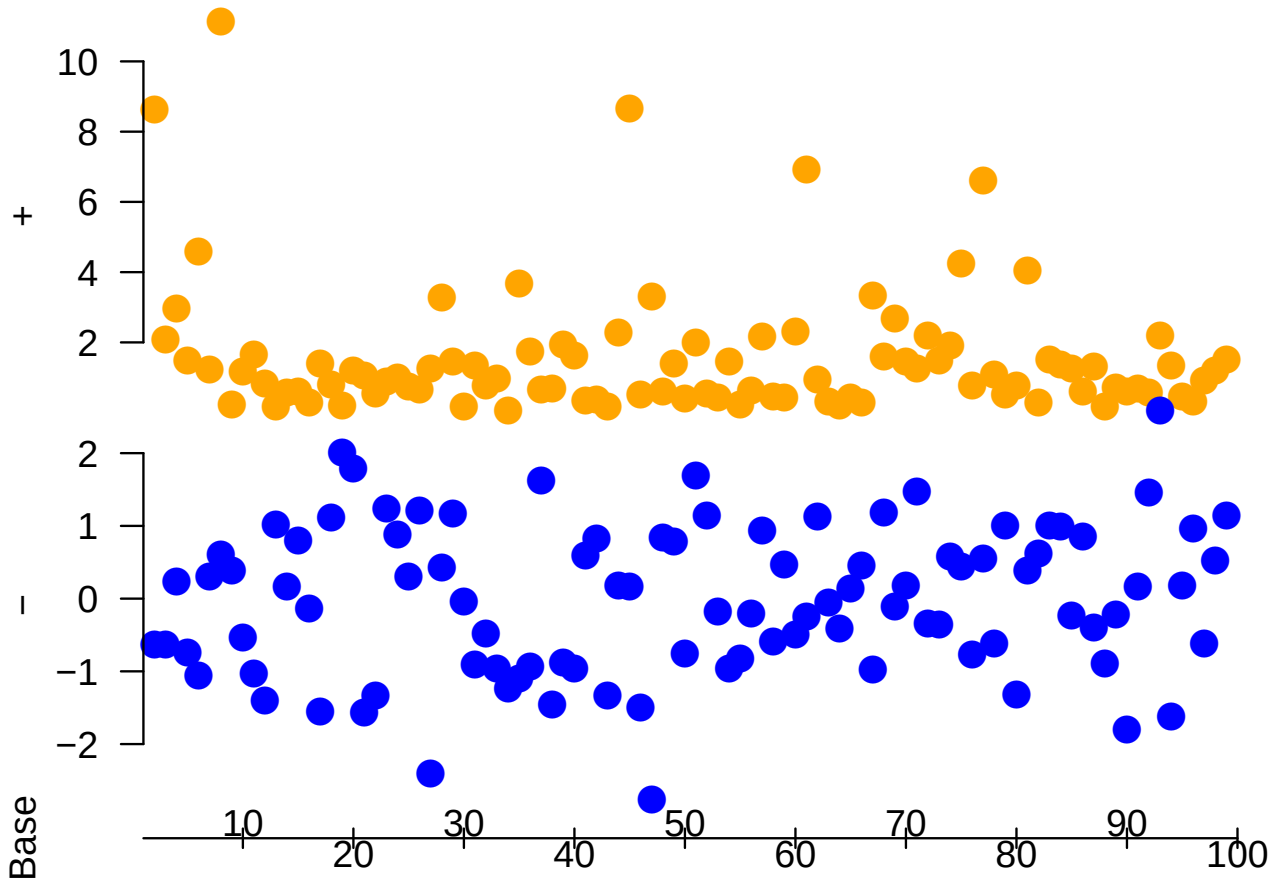
- How many graphical arguments does the genome axis object have?

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:

```
> args(makeGeneRegion)

function (start, end, chromosome, strand, biomart, dp = NULL)
NULL
```

- The biomart argument is a mart object. Lets create one:

```
> bsub <- useMart("bacterial_mart_8",
+     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 chromsome name, I simply checked the attributes list.

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

```
                          name
1               ensembl_gene_id
2         ensembl_transcript_id
3            ensembl_peptide_id
4 canonical_transcript_stable_id
5                   description
6               chromosome_name
                       description
1                 Ensembl Gene ID
2            Ensembl Transcript ID
3               Ensembl Protein ID
4 Canonical transcript stable ID(s)
5                     Description
6                   Chromosome Name
```

Step by step solution

- Of course, we can verify this with by using getBM[3]:

```
> 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:

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

Step by step solution

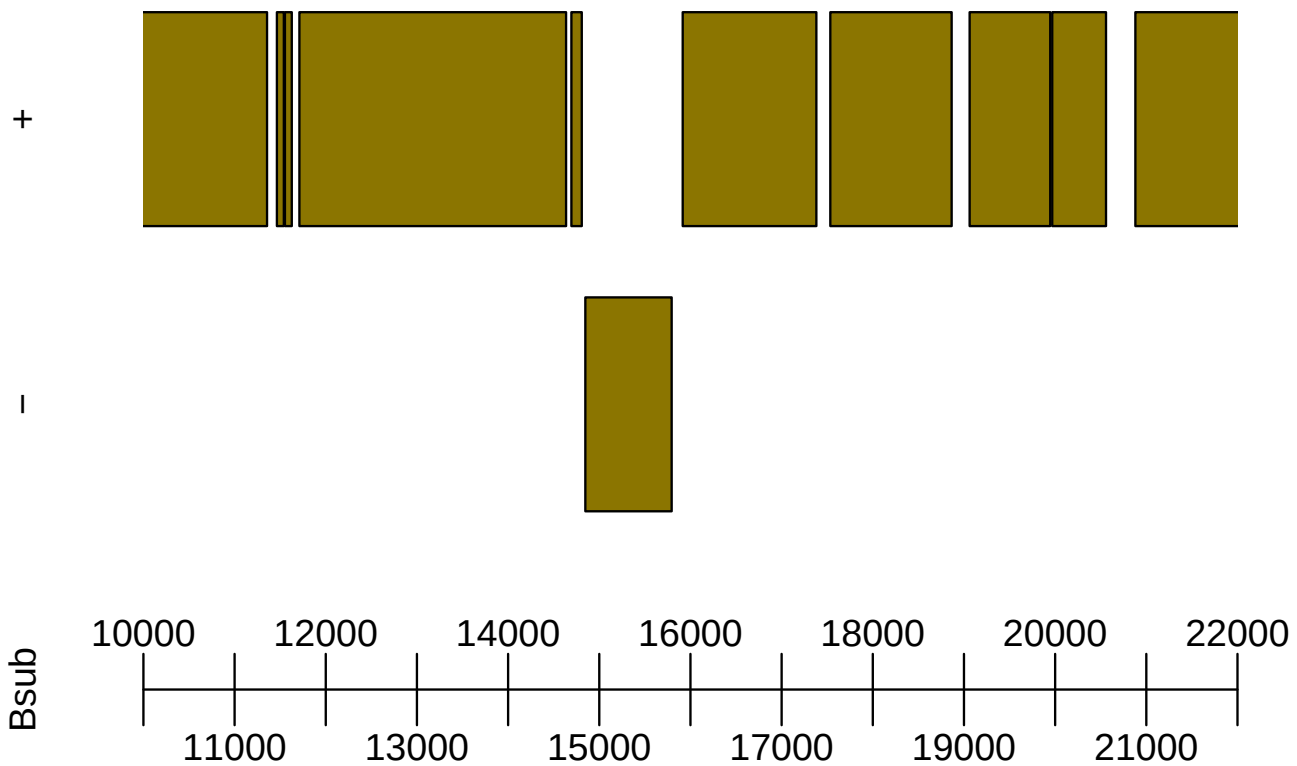
- For the d object, we use nearly the same code and we finish the job with gdPlot:

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

[3] I just modified one of the code lines we used on the biomaRt class.

Resulting plot

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



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

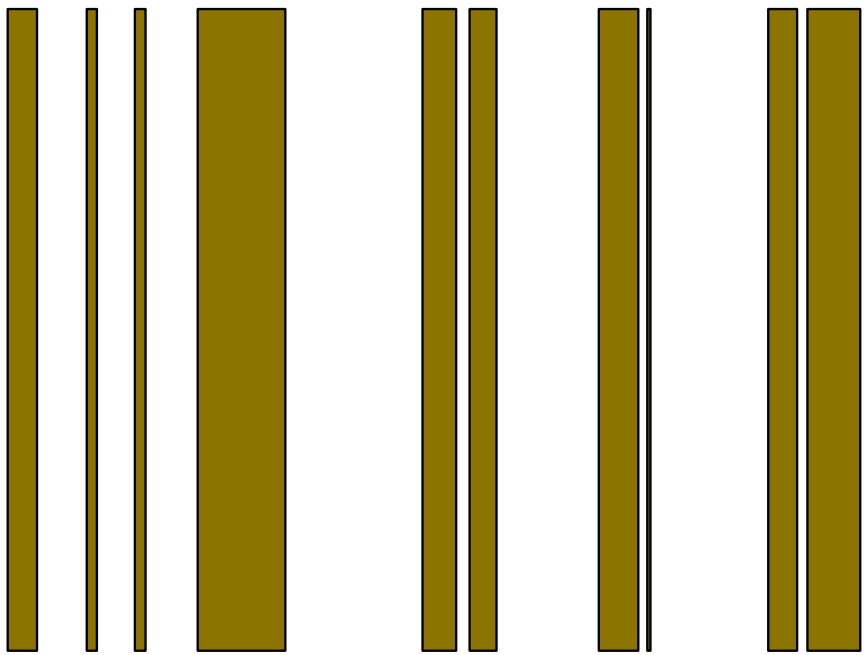
	chr	location	strand	expr
1	4	1300003	-1	-0.20
2	4	1300007	1	0.61
3	4	1300011	-1	0.07
4	4	1300015	1	1.25
5	4	1300019	-1	-0.29
6	4	1300023	1	0.61

- Lets take a peak at chromosome IV:

Basic plot

```
> gdPlot(makeGeneRegion(10000, 50000,
+      chr = "IV", strand = "+", biomart = mart),
+      10000, 50000)
```

Basic plot



Gene names

- Lets add the gene names (plot ids).

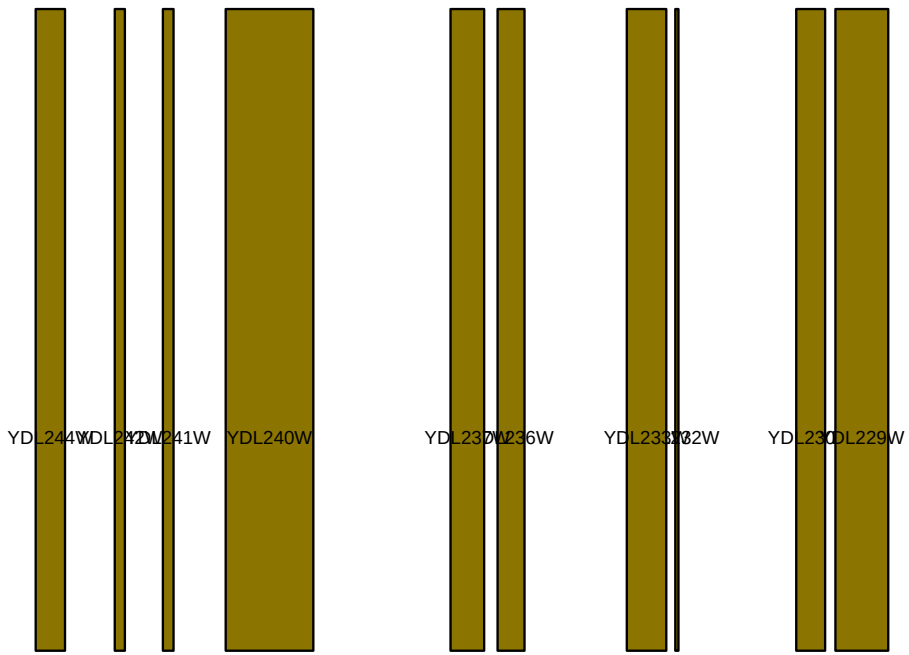
  ```
  > showDisplayOptions("GeneRegion")
  ```

- 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

```
> 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



GenericArray

- Not much, right?
- Lets make one with the microarray data using makeGenericArray:

```
> args(makeGenericArray)

function (intensity, probeStart, probeEnd, trackOverlay, dp = NULL)
NULL
```

- For less typing, lets save the data into a shorter object:

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

- We did a `head` earlier on the data, so lets use `location` for `probeStart` and `expr` for `intensity` arguments respectively.
- As a short parenthesis, look at this neat trick:

```
> 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])
```

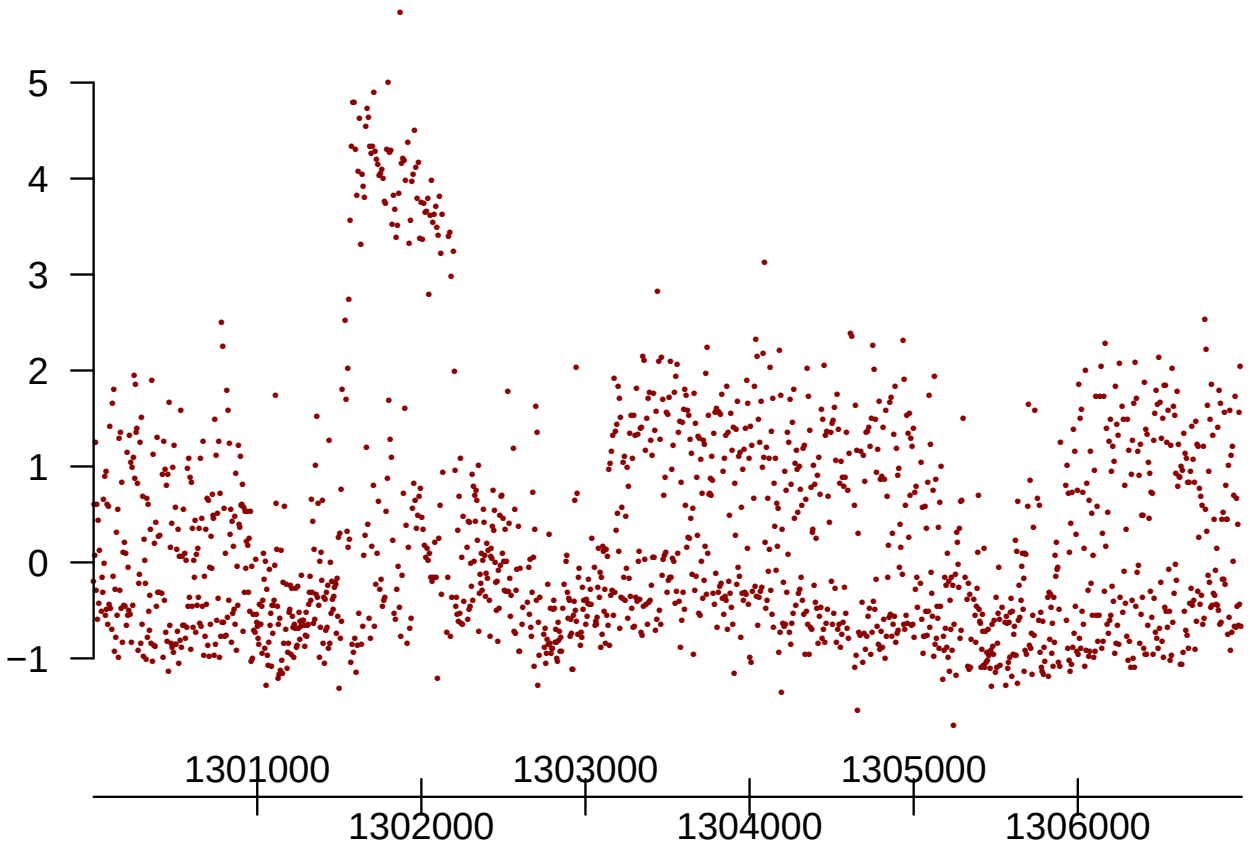
```
    1     2     3     4     5     6
-0.20  0.61  0.07  1.25 -0.29  0.61
```

- Neat eh? :) Lets use makeGenericArray now:

```
> e <- makeGenericArray(david[, "expr",
+     drop = FALSE], david[, "location"])
```

GenericArray plot

```
> gdPlot(list(e, makeGenomeAxis()))
```



Something... complicated :)

Now, lets make a complicated plot

- One GeneRegion for each strand
- One GenericArray for each strand

Code:

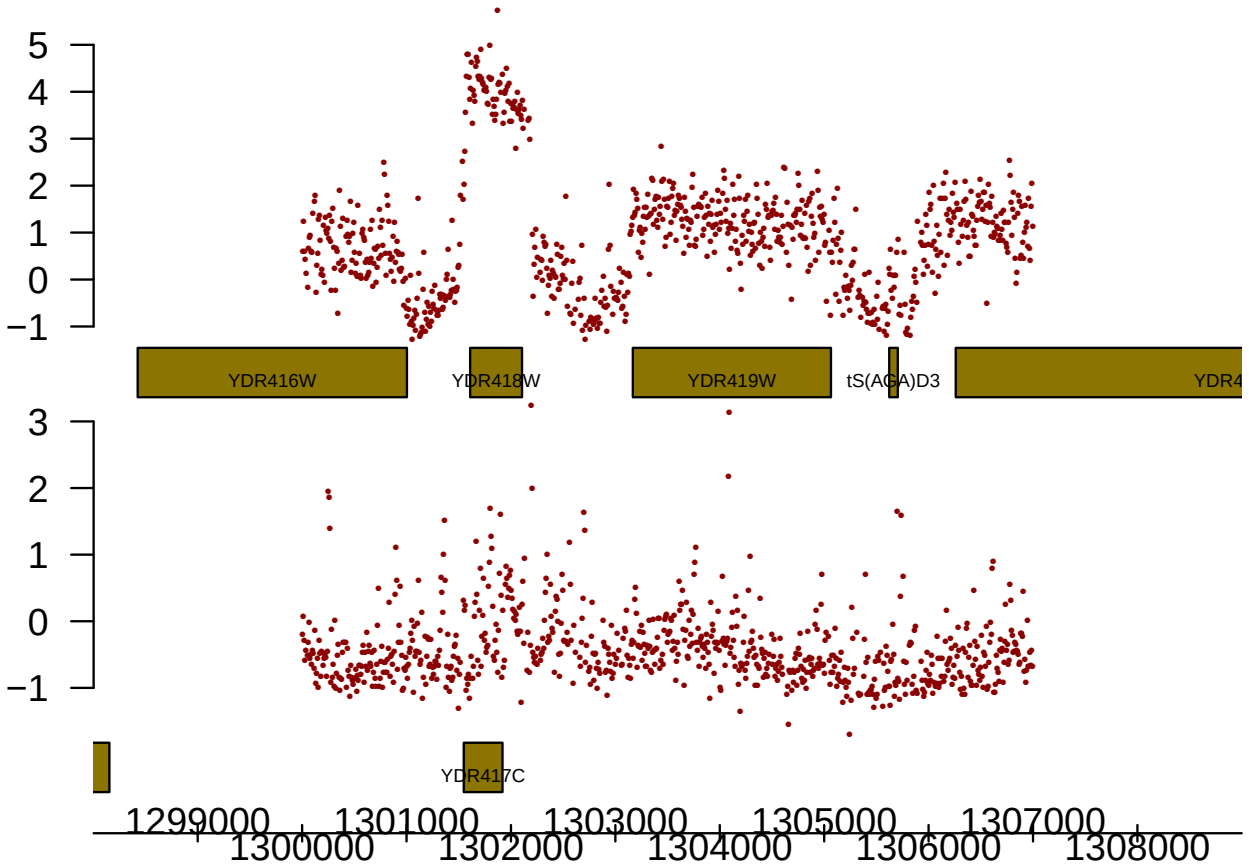
```
> df <- as.data.frame(seqDataEx$david)
> lst <- lapply(c("+", "-"), function(s) {
+     a <- as.matrix(subset(df, strand ==
+         ifelse(s == "+", 1, -1)))
+     c(makeGenericArray(a[, "expr",
+         drop = FALSE], a[, "location"]),
+         makeGeneRegion(start = min(df[,
+             "location"]),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

Code:

```
+ })
> 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:

```
> 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")
```

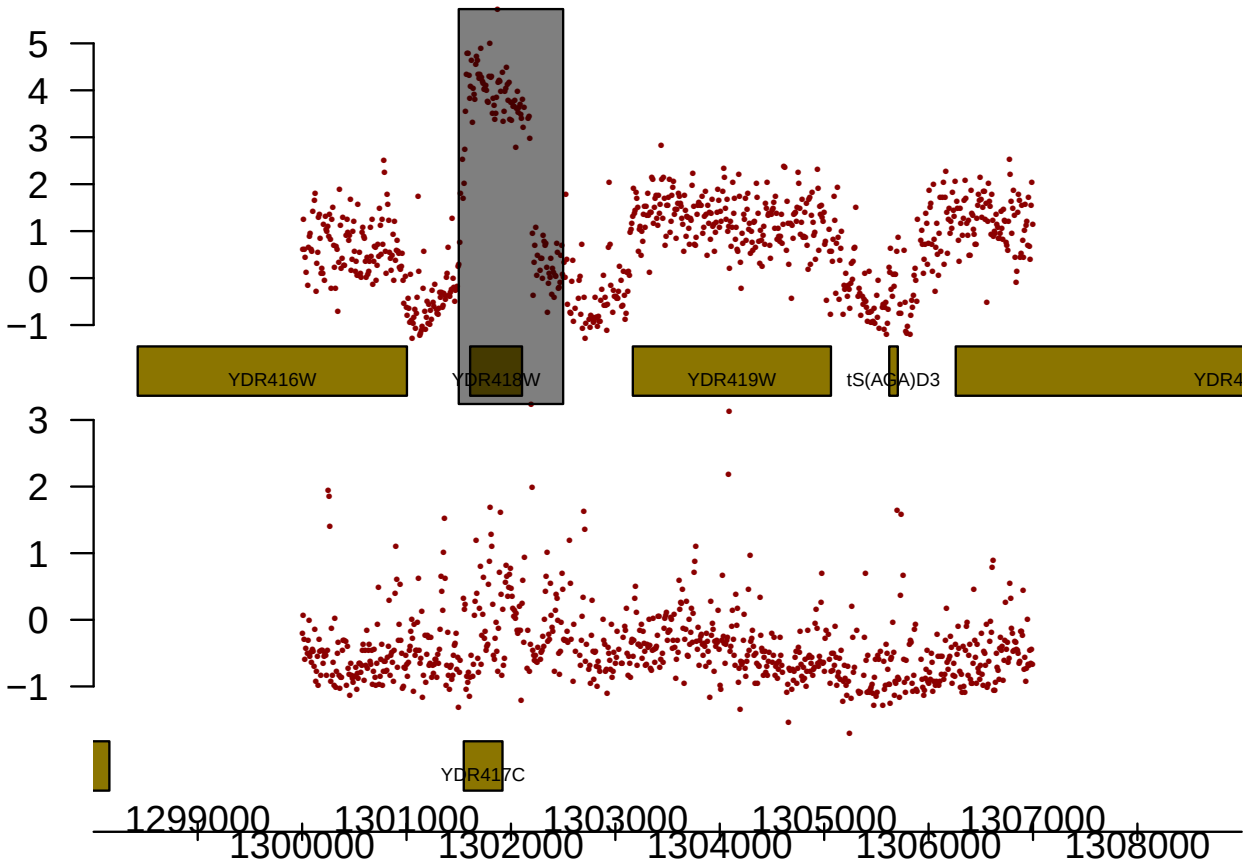
```
alpha  =  0
color  =  black
fill  =  black
lty  =  solid
lwd  =  1
```

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

Solution :)

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

Solution :)


5
4
3
2
1
0
−1
YDR416W
YDR418W
YDR419W
tS(AGA)D3
YDR4
3
2
1
0
−1
YDR417C
1299000
1300000
1301000
1302000
1303000
1304000
1305000
1306000
1307000
1308000

With text

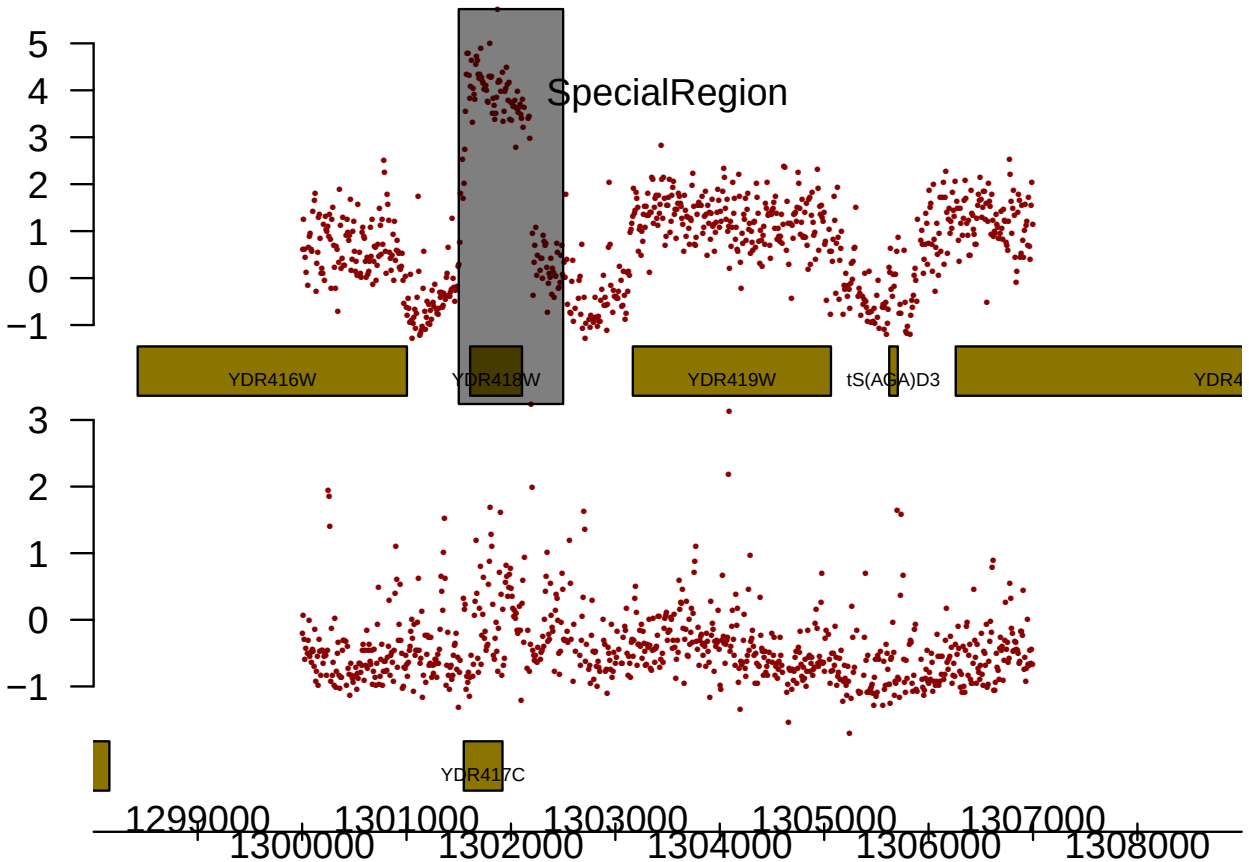
- Now, lets add some text using makeTextOverlay

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

End result

```
> gdPlot(yeastLst, overlays = c(ovlay,
+     tovlay))
```

End result


SpecialRegion
YDR416W
YDR418W
YDR419W
tS(AGA)D3
YDR4
YDR417C
1299000
1300000
1301000
1302000
1303000
1304000
1305000
1306000
1307000
1308000

Transcripts

- For those of you who love splicing, makeTranscript will be most useful :)
- Lets take a look at gene ENSG00000168309:

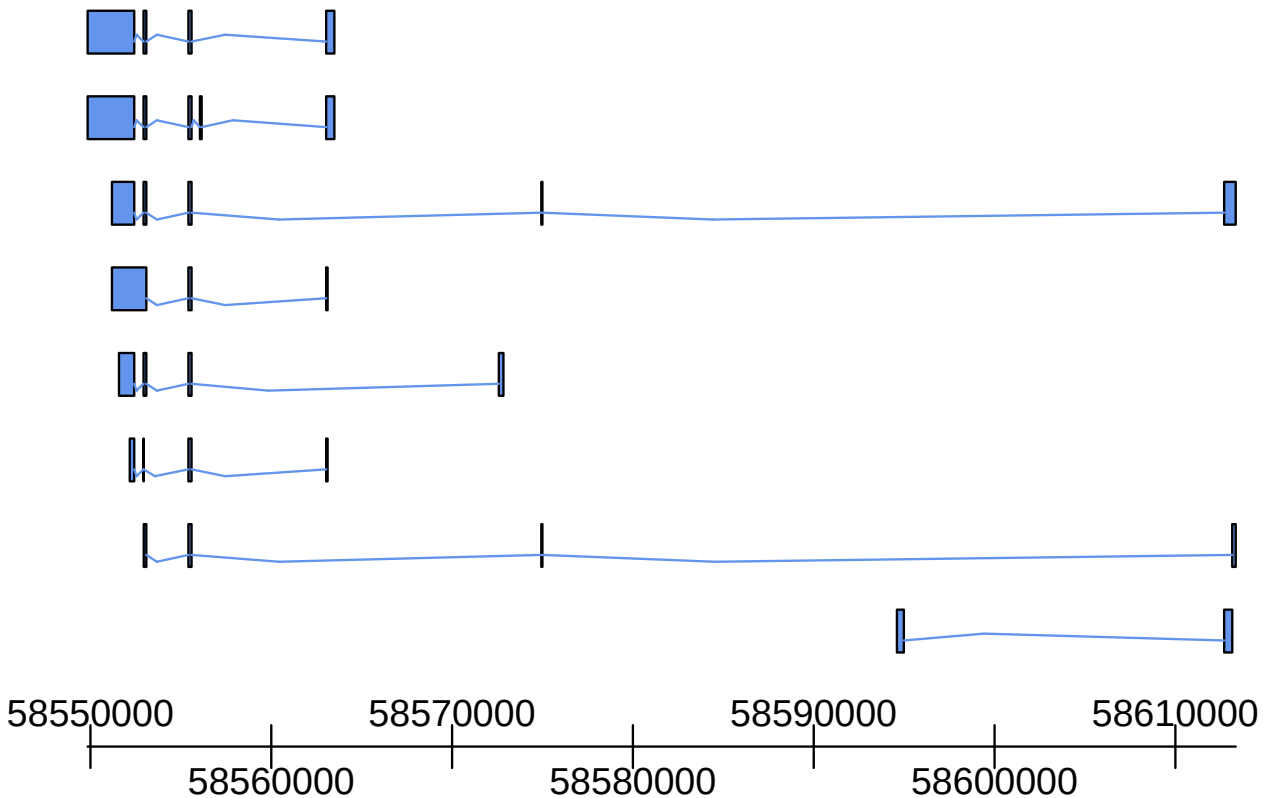
```
> args(makeTranscript)

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

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

Alternative splicing

```
> 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)

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

> args(makeGeneModel)

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

- Here is an example using the unrData dataset:

Exons and gene models

```
> data("unrData", package = "GenomeGraphs")
> class(unrData)

[1] "matrix"

> dim(unrData)

[1] 117  33

> head(unrPositions)
```

Exons and gene models

```
  probesetId chromosome     start
1    2429278          1 115061081
2    2429279          1 115061152
3    2429280          1 115061275
4    2429281          1 115061486
5    2429282          1 115061888
6    2429283          1 115062185
       stop
1 115061119
2 115061198
3 115061409
4 115061528
5 115062089
6 115062218
```

Exons and gene models

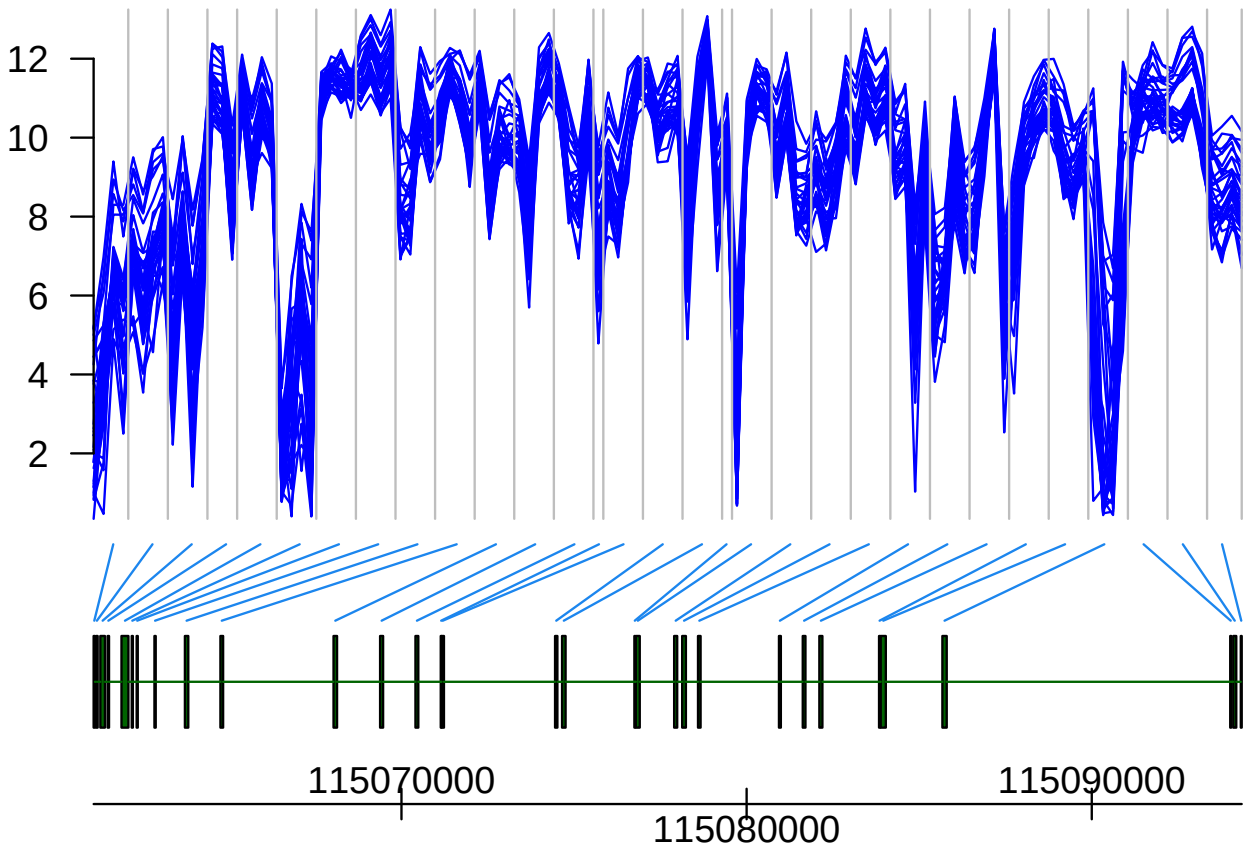
- First we create the exon track which zooms into every exon, and then a gene model so we don't lose the forest :)

```
> 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])
```

[4] For the curious ones, you can make custom annotation tracks using makeAnnotationTrack.

Example plot

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



Conclusions

- Fast! Which is great for a quick exploration of your data by regions.
- Once you get the basics, its 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 :)

SessionInfo

```
> sessionInfo()

R version 2.12.0 (2010-10-15)
Platform: i386-pc-mingw32/i386 (32-bit)

locale:
[1] LC_COLLATE=English_United States.1252
[2] LC_CTYPE=English_United States.1252
[3] LC_MONETARY=English_United States.1252
[4] LC_NUMERIC=C
[5] LC_TIME=English_United States.1252

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

other attached packages:
[1] GenomeGraphs_1.10.0
[2] biomaRt_2.6.0
```

SessionInfo

```
loaded via a namespace (and not attached):
[1] RCurl_1.4-4.1 tools_2.12.0
[3] XML_3.2-0.1
```