Segmentation demo

Wolfgang Huber, Joern Toedling

December 15, 2008

Contents

1 Introduction

This vignette demonstrates how to run *tilingArray*'s segmentation function on the *davidTiling* data.

First we load the package *tilingArray*, which contains the algorithms, and the package *davidTiling*, which contains the data and the array annotation.

```
> library("tilingArray")
> library("davidTiling")
> data("davidTiling")
> data("probeAnno")
> library("RColorBrewer")
> library("geneplotter")
> library("grid")
```

2 Normalization of the data

For an explanation of the following code, please see the vignette *Normalisation with the normalizeByReference function in the tilingArray package* (in the file `assessNorm.Rnw`).

```
> isRNA = davidTiling$nucleicAcid %in% c("poly(A) RNA","total RNA")
> isDNA = davidTiling$nucleicAcid %in% "genomic DNA"
> stopifnot(sum(isRNA)==5, sum(isDNA)==3)
```

```
> xn = normalizeByReference(
+   x = davidTiling[,isRNA],
+   reference = davidTiling[,isDNA],
+   pm = PMindex(probeAnno),
+   background=BGindex(probeAnno))
> pData(xn)[, 2, drop=FALSE]
```

```
                              nucleicAcid
05_04_27_2xpolyA_NAP3.cel     poly(A) RNA
05_04_26_2xpolyA_NAP2.cel     poly(A) RNA
05_04_20_2xpolyA_NAP_2to1.cel poly(A) RNA
050409_totcDNA_14ug_no52.cel    total RNA
030505_totcDNA_15ug_affy.cel    total RNA
```

3 Segmentation

3.1 Prerequisites: Avoid oversampling

The spacing between probe-matched positions is not completely regular, as Figure 1 exemplarily shows for the probes mapped to the Watson strand of chromosome 1. In particular, repetitive regions are highly oversampled. To have these repetitive, rather uninformative regions not dominate the segmentation algorithm, the probe positions are subsampled in the segmentation function to have a regular spacing. The result of this subsampling is shown in the comparison between Figures 1b and 1c.

```
> chr1p.probeStarts = sort(probeAnno$"1.+.start")
> sampled.probeStarts = chr1p.probeStarts[sampleStep(chr1p.probeStarts, 7)]
```

```
> par(mfrow=c(3,1))
> hist(chr1p.probeStarts, col="mistyrose", 100, main="(a)")
> barplot(table(diff(chr1p.probeStarts)), col="sienna", main="(b)")
> barplot(table(diff(sampled.probeStarts)), col="palegreen", main="(c)")
```

3.2 Call the segmentation algorithm

The segmentation algorithm needs two parameters. `maxk` is the maximum length of any individual segment. `nrBasesPerSegment` is used to calculate `maxseg`, the maximum number of segments that the algorithm is going to consider by dividing the length of the region to be segmented (in the current case, a chromosome) by `nrBasesPerSegment`. The `nrBasesPerSegment` does not enforce a minimum length restriction for individual segments.

We choose `nrBasesPerSegment` to be quite low, an *average* length per segment of 750 bases, such that it corresponds to a quite high number of segments, `maxseg` . The algorithm will calculate all optimal segmentations with $1, 2, \ldots,$ `maxseg` segments, and we can still later choose our preferred one. Note that `maxk` is measured in number of data points, not in genomic coordinates. Our choice of the parameter `maxk` corresponds to a maximum segment length of about $7.5 \times 3,000 = 22,500$ bases.

To demonstrate the algorithm, we run the segmentation on the Watson strand of chromosome 1.

```
> segEnv = segChrom(xn[,xn$nucleicAcid=="poly(A) RNA"],
+                   probeAnno=probeAnno, chr="1", strands="+",
+                   nrBasesPerSegment = 750)
```

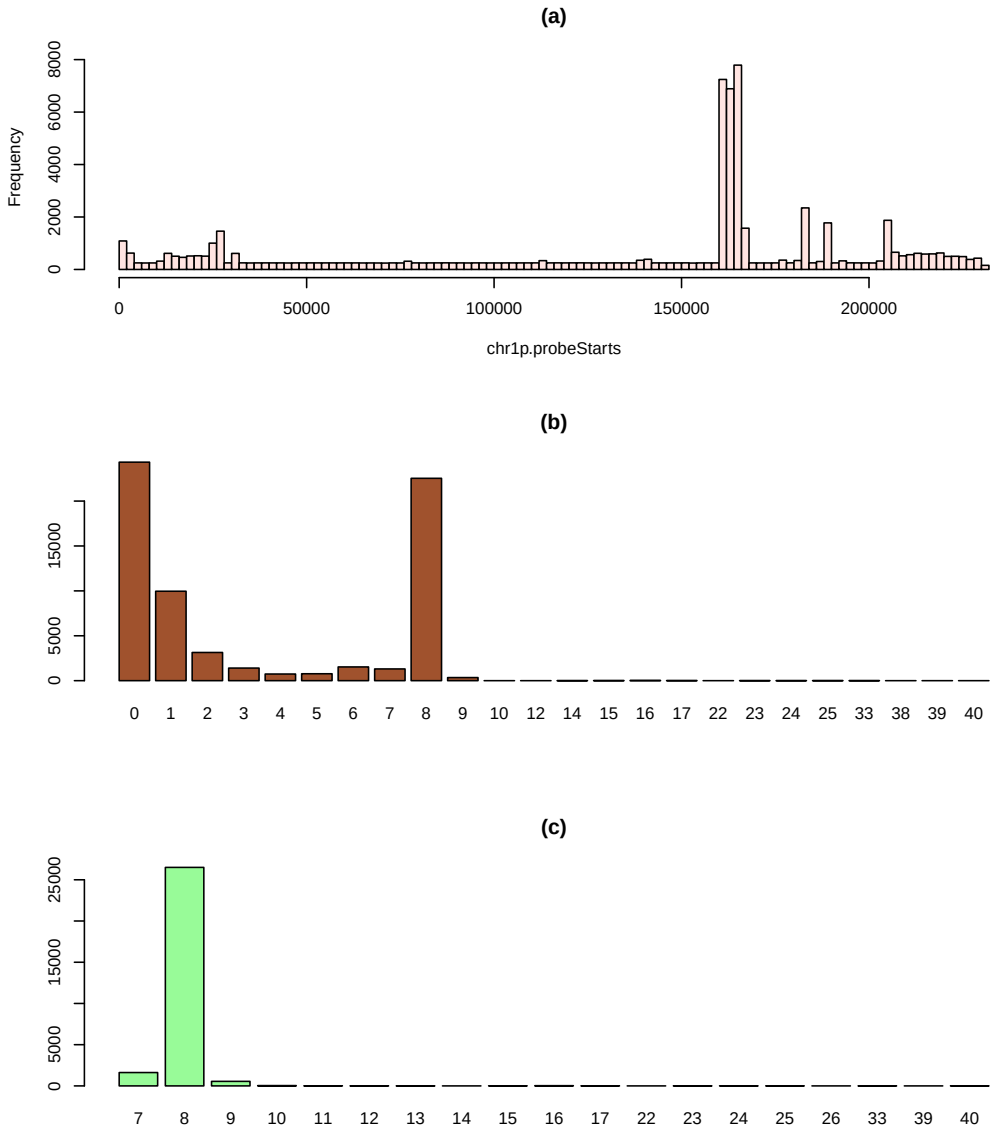


Figure 1: (a): Histogram of probe midpoints along the "+" strand of chromosome 1. There are some probe dense regions in particular around 160,000. The sequence of that region is repeated multiple times in the genome, and due to the way the chip was designed, there are also a lot of probes (more than necessary) for that region. (b): histogram of differences between probe start points (`chr1p.probeStarts`). The intention of the chip design was to have a regular spacing of 8 bases. In some cases, the spacing is wider, probably due to updates in the genome sequence between when the chip was designed and when probes were re-aligned. In many cases, it is tighter with multiple probes for the same target sequence, or only 1 or 2 bases offset. This occurs in the regions of duplicated sequence. (c): histogram of differences between probe midpoints after subsampling (`sampled.probeStarts`)

The function `segChrom` already contains code similar to that shown in Section 3.1 to prevent oversampling, see the function argument `step`. The resulting object `segEnv` is an environment holding the segmentation results for each chromosome and strand as individual objects of class *segmentation*.

```
> ls(segEnv)
```

```
[1] "1.+"
```

```
> segChr1p <- get("1.+", env=segEnv)
> segChr1p
```

```
Object of class 'segmentation':
Data matrix: 27585 x 3
Change point estimates for number of segments S = 1:307
Selected S = 307
```

3.3 Calculate confidence intervals

We can compute confidence intervals for each segment boundary. This is simply a call to the `confint` method for the *segmentation* class.

```
> nseg = round(max(get("1.+.end",env=probeAnno))/1500)
> confintLevel = 0.95
> segChr1p = confint(segChr1p, parm=nseg, level=confintLevel)
> segChr1p
```

```
Object of class 'segmentation':
Data matrix: 27585 x 3
Change point estimates for number of segments S = 1:307
Confidence intervals for 1 fits from S = 153 to 153
Selected S = 153
```

Now we are ready to have a look at the result via the `plot` method of the *segmentation* class. The plot in Figure 2 shows a small section of the probe levels mapped to the Watson strand of chromosome 1, the fitted segment borders. The confidence interval for each border is indicated by the parentheses around it on the bottom side.

```
> plot(segChr1p, nseg, pch=16, cex=0.2, xlim=c(30000, 40000))
```

3.4 Model selection.

The log-likelihood is

$$\log L = -\frac{n}{2}\left(\log 2\pi + 1 + \log\frac{\sum_i r_i}{n}\right), \tag{1}$$

where r_i the i-th residual and n the number of data points. AIC and BIC are defined as

$$\begin{aligned} \mathrm{AIC} &= -2\log L + 2p & (2)\\ \mathrm{BIC} &= -2\log L + p\log n & (3) \end{aligned}$$

where p is the number of parameters of the model. In our case, $p = 2S$, since for a segmentation with S segments, we estimate $S-1$ changepoints, S mean values, and 1 standard deviation. We can also consider the penalized likelihoods

$$\begin{aligned} \log L_{\mathrm{AIC}} &= \log L - p & (4)\\ \log L_{\mathrm{BIC}} &= \log L - \frac{p}{2}\log n & (5) \end{aligned}$$

We plot them as functions of S, see Figure 3(a).

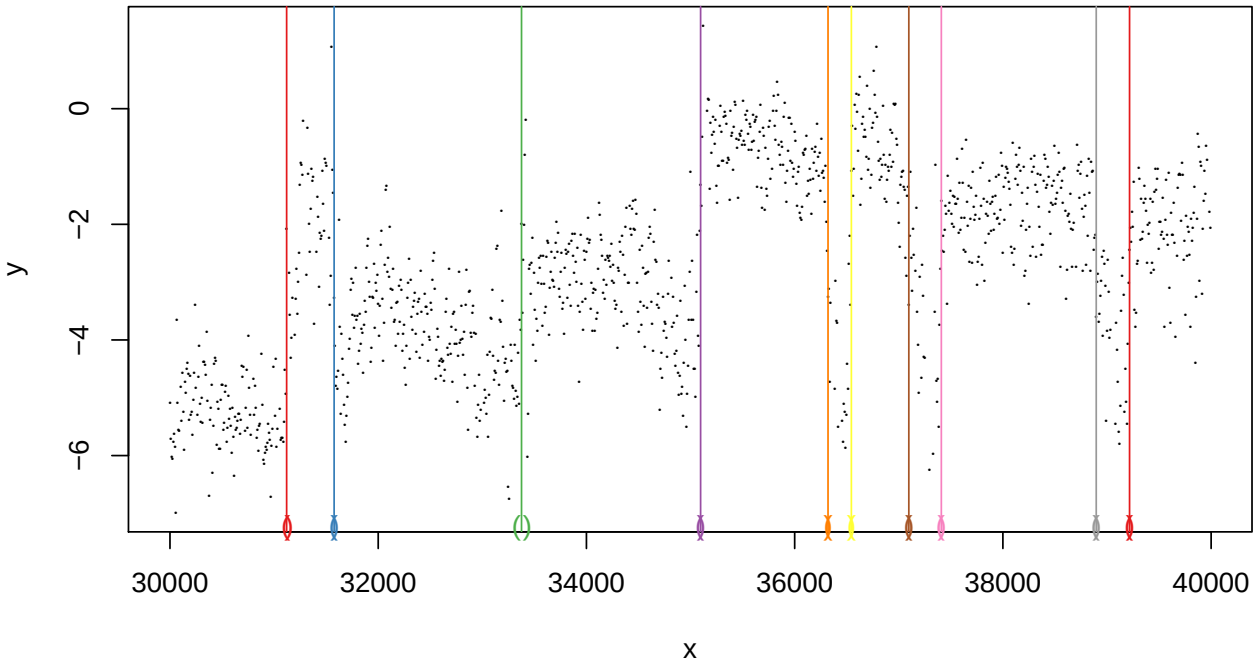
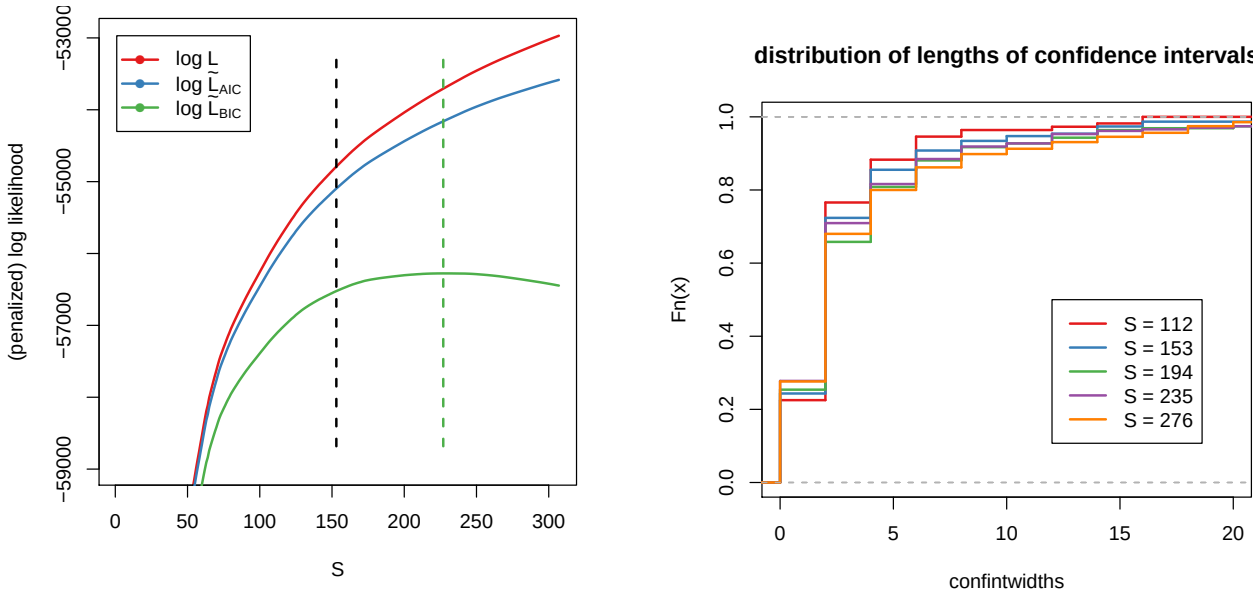


Figure 2: Segmentation with confidence intervals, shown by little brackets at the bottom.

```
> par(mai=c(1,1,0.1,0.01))
> plotPenLL(segChr1p,
+   extrabar =c("black"=round(segChr1p@x[length(segChr1p@x)]/1500)),
+   type="l", lwd=2)
```

3.5 Size of the confidence intervals as a function of S

```
> segMultiCI = confint(segChr1p,
+   parm = c(112L, 153L, 194L, 235L, 276L),
+   level = confintLevel)
> nBp = which(segMultiCI@hasConfint)
> confintwidths = lapply(segMultiCI@breakpoints[nBp], function(m) (m[,3]-m[,1]))
```



(a) (b)

Figure 3: **(a)** Model selection: log-likelihood and two versions of penalized log-likelihood (AIC and BIC) as a function of the number of segments S. Vertical dashed green bar corresponds to optimal $\log L_{\mathrm{BIC}}$, vertical dashed grey bar to our "subjective" choice of average segment length 1,500 bases. **(b)** Size of the confidence intervals as a function of S. Cumulative distribution functions (CDFs) for the distributions of confidence interval widths for S =112, 153, 194, 235, 276. For larger S, the confidence intervals are wider.

```
> par(mai=c(0.9,0.9,0.8,0.01))
> maxx = 20
> colors = brewer.pal(length(nBp), "Set1")
> multiecdf(confintwidths, xlim=c(0, maxx),
+   main='distribution of lengths of confidence intervals',
+   verticals=TRUE, lwd=2, col=colors)
> legend(x=0.6*maxx, y=0.5, legend=paste("S =", nBp),
+   col=colors, lty=1, lwd=2)
```

The result is shown in Figure 3(b).

3.6 Computing the segmentation for both strands on 7 chromosomes

Since the data in the *davidTiling* package are strand-specific, we can do the segmentation for the "-" strand of chromosome 1 as well and produce the along-chromosome plot shown in Figure 4. For Figures 5 and 6, we also call it on six other chromosomes.

This computation will take a couple of hours (about 18h on mine). Note that it could easily be parallelized if needed, since the computations for different chromosome strands are independent of each other.

```
> segEnv = segChrom(xn[, xn$nucleicAcid=="poly(A) RNA"],
+             probeAnno = probeAnno,
+             chr = c(1, 2, 5, 9, 13, 14, 15),
+             strands = c("+","-"),
+             nrBasesPerSegment = 1500)
```

`segEnv` is an environment holding the 14 individual segmentation results.

4 Visualizing segmentations with `plotAlongChrom`

```
> data("gff")
> myGff = gff[ gff$Name!="tR(UCU)E", ]
> ylim  = quantile(exprs(xn)[,1:3], probs=c(0.001, 0.999), na.rm=TRUE)
```

The function `plotAlongChrom` accepts an environment as its first argument, which is expected to contain objects of class *segmentation* with names given by `paste(chr, c("+", "-"), sep=".")`, where `chr` is the chromosome identifier. The output of `segChrom`, the function we called above, is such an environment.

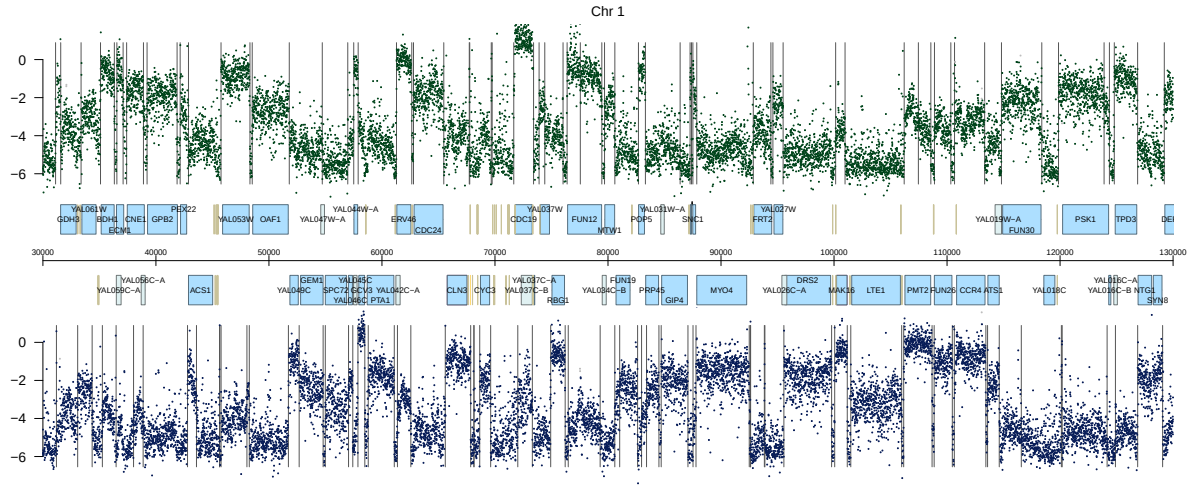


Figure 4: Along-chromosome plot similar to Figure 1 of [1].

In the following, the code to generate Figure 1 of [1].

```
> grid.newpage()
> plotAlongChrom(segObj=segEnv, chr=1, coord = c(30, 130)*1e3, ylim=ylim,
+                gff=myGff, showConfidenceIntervals=FALSE,
+                featureNoLabel = c("uORF", "binding_site", "TF_binding_site"),
+                doLegend=FALSE, main="")
```

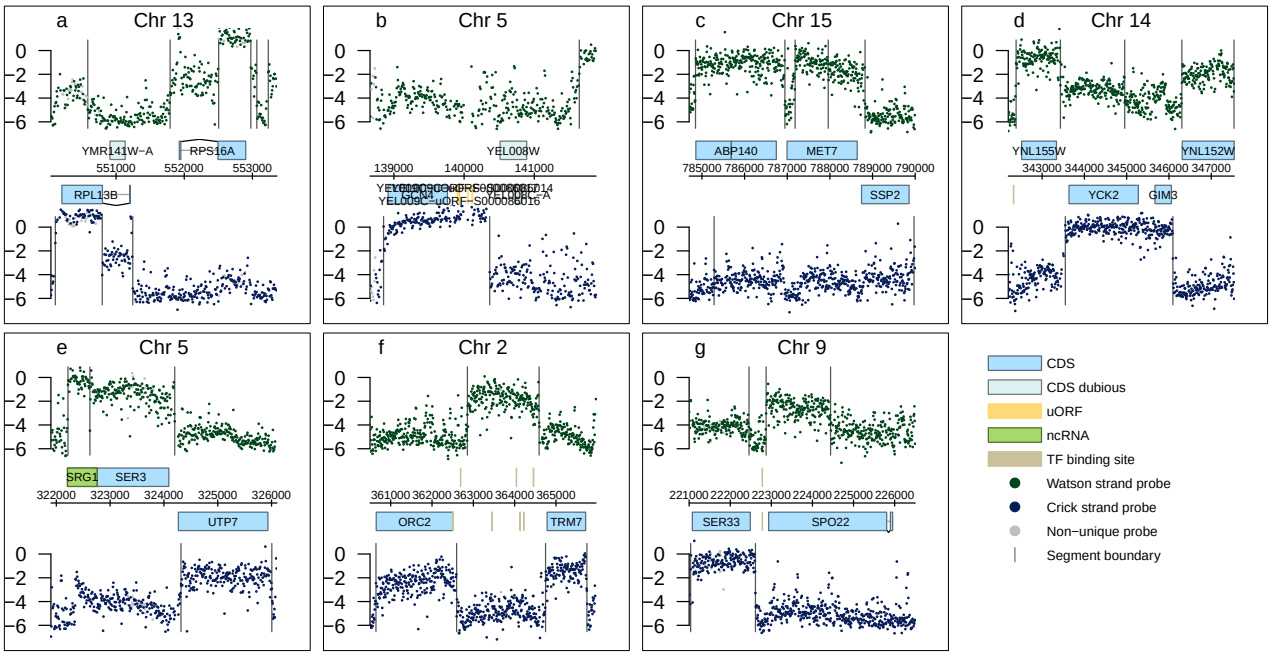


Figure 5: Along-chromosome plots similar to Figure 2 of [1].

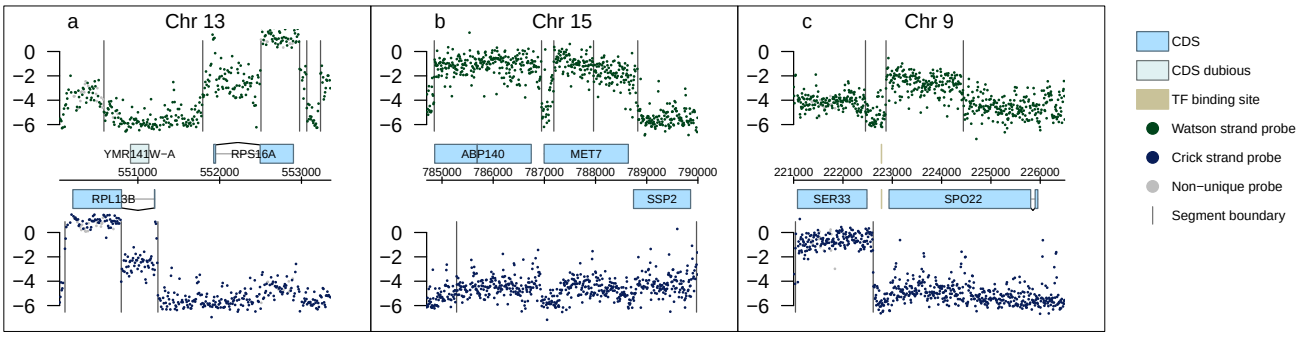


Figure 6: Along-chromosome plots similar to Figure 2 of [2].

The folowing code was used to generate Figure 2 of [1].

```
> myPlot = function(row, col, ...) {
+   pushViewport(viewport(layout.pos.row=row, layout.pos.col=col))
+   grid.rect(x=-0.1, width=1.15, y=0.0, height=1.02, just=c("left", "bottom"),
+             default.units="npc", gp=gpar(lwd=0.2))
+   plotAlongChrom(..., segObj=segEnv, ylim=ylim,
+                  gff=myGff,
+                  featureNoLabel = c("binding_site", "TF_binding_site"),
+                  doLegend=FALSE)
+   popViewport()
+ }
> myLegend = function(row, col, what) {
+   fc = tilingArray:::featureColors(1)
+   fc = switch(what,
+     fc[c("CDS", "CDS_dubious", "uORF", "ncRNA", "TF_binding_site"), ],
+     fc[c("CDS", "CDS_dubious", "TF_binding_site"), ])
+
+   pc = c("Watson strand probe"="#00441b", "Crick strand probe"="#081d58",
+          "Non-unique probe"="grey")
+   sc = c("Segment boundary"="#777777")
+
+   pushViewport(dataViewport(xscale=c(0,1), yscale=c(-7,nrow(fc)+1),
+                layout.pos.col=col, layout.pos.row=row))
+   h1  = nrow(fc):1
+   h2  = 0:(1-length(pc))
+   h3  = -length(pc)
+
+   w   = 0.2
+   grid.rect(x=0, width=w, y=h1, height = unit(1, "native")- unit(2, "mm"),
+             just  = c("left", "center"), default.units="native",
+             gp    = do.call("gpar", fc))
+   grid.circle(x = w/2, y=h2, r=0.2, default.units="native",
+               gp = gpar(col=pc, fill=pc))
+   grid.lines(x=w/2, y=h3+c(-.3,+.3), default.units="native", gp=gpar(col=sc))
+   grid.text(label = c(gsub("_", " ", rownames(fc)), names(pc), names(sc)),
+             x = w*1.1, y = c(h1,h2,h3),
+             just  = c("left", "center"), default.units="native",
+             gp=gpar(cex=.7))
+   popViewport()
+ }
> dx = 0.20
> dy = 0.05
> grid.newpage()
> pushViewport(viewport(x=0.02, width=0.96, height=0.96, just=c("left", "center"),
+                       layout=grid.layout(3, 8,
+                         height=c(1, dy, 1),
+                         width =c(dx, 1, dx, 1, dx, 1, dx, 1))))
> ## A) 13:550k splicing RPS16A, RPL13B
> myPlot(1, 2, chr=13, coord = c(550044, 553360), main="a")
```

```
> ## B) GCN4
> myPlot(1, 4, chr=5, coord = c(138660, 141880), main="b")
> ## C) MET7, novel architecture
> myPlot(1, 6, chr=15, coord = c(784700, 790000), main="c")
> ## D) overlapping transcripts
> myPlot(1, 8, chr=14, coord = c(342200, 347545), main="d")
> ## E) SER3
> myPlot(3, 2, chr=5, coord = c(321900, 326100), main="e")
> ## F) 2:360.5-366.5: novel isolated
> myPlot(3, 4, chr=2, coord = c(360500, 365970), main="f")
> ## G) 9:221-227: novel antisense SPO22
> myPlot(3, 6, chr=9, coord = c(221000, 226500), main="g")
> myLegend(3, 8, 1)
> popViewport()
```

In the following, the code to generate Figure 2 of [2].

```
> grid.newpage()
> pushViewport(viewport(x=0.02, width=0.96, height=0.96, just=c("left", "center"),
+                       layout=grid.layout(1, 8,
+                         height=c(1),
+                         width =c(0.05, 1, 0.15, 1, 0.15, 1, 0.15, 0.5))))
>          ## 13:550k splicing RPS16A, RPL13B
> myPlot(1, 2, chr=13, coord = c(550044, 553360), main="a")
> ## MET7, novel architecture
> myPlot(1, 4, chr=15, coord = c(784700, 790000), main="b")
> ## 9:221-227: novel antisense SPO22
> myPlot(1, 6, chr=9, coord = c(221000, 226500), main="c")
> myLegend(1, 8, 2)
> popViewport()
```

5 Version information

This vignette was generated using the following package versions:

```
> toLatex(sessionInfo())
```

- R version 2.9.0 Under development (unstable) (2008-12-12 r47169), `x86_64-unknown-linux-gnu`
- Locale: `LC_CTYPE=it_IT;LC_NUMERIC=C;LC_TIME=it_IT;LC_COLLATE=it_IT;LC_MONETARY=C;LC_MESSAGES=i`
- Base packages: base, datasets, graphics, grDevices, grid, methods, stats, tools, utils
- Other packages: annotate 1.21.2, AnnotationDbi 1.5.6, Biobase 2.3.5, codetools 0.2-1, davidTiling 1.2.8, DBI 0.2-4, digest 0.3.1, fortunes 1.3-6, geneplotter 1.21.0, GO.db 2.2.5, lattice 0.17-17, pixmap 0.4-9, RColorBrewer 1.0-2, RSQLite 0.7-1, tilingArray 1.21.4, weaver 1.9.0
- Loaded via a namespace (and not attached): affy 1.21.0, affyio 1.11.2, genefilter 1.23.0, KernSmooth 2.22-22, limma 2.17.3, preprocessCore 1.5.2, splines 2.9.0, strucchange 1.3-5, survival 2.34-1, vsn 3.10.2, xtable 1.5-4

References

[1] Lior David, Wolfgang Huber, Marina Granovskaia, Joern Toedling, Curtis J. Palm, Lee Bofkin, Ted Jones, Ronald W. Davis, and Lars M. Steinmetz A high-resolution map of transcription in the yeast genome. *PNAS*, 2006. 7, 8, 9

[2] Wolfgang Huber, Joern Toedling and Lars M. Steinmetz Transcript mapping with oligonucleotide high-density tiling arrays. *Bioinformatics*, 2006. 8, 10