Segmentation demo

Wolfgang Huber

September 25, 2006

Contents

1 Introduction **1**

2 Normalization of the data **1**

3 Segmentation **2**

3.1 Selecting the probes in along-chromosome order 2

3.1.1 Avoid oversampling 3

3.2 Call the segmentation algorithm 3

3.3 Calculate confidence intervals 5

3.4 Model selection. 5

3.5 Size of the confidence intervals as a function of S 6

3.6 Definition of segFun 7

3.7 Using the `plotAlongChrom` function for more elaborate displays. 8

1 Introduction

This script presents a demo of the 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")
```

2 Normalization of the data

Please see the vignette *Assessing signal/noise ratio before and after normalization* (`assessNorm.Rnw`) for an explanation of the following code.

```
> nc = as.integer(2560)
> PMind = rep(seq(as.integer(1), nc - as.integer(3), by = as.integer(2)),
+     each = nc) * nc + (1:nc)
> MMind = PMind + nc
> ispm = rep(FALSE, nc * nc)
> ispm[PMind] = TRUE
> isbg = (probeAnno$probeReverse$no_feature == "no" & probeAnno$probeDirect$no_feature ==
+     "no" & ispm)
> 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(davidTiling[, isRNA], davidTiling[,
+     isDNA], pm = PMind, background = isbg)
> 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 Selecting the probes in along-chromosome order

Extract for all probes that map to the "+" strand of chromosome 1 their start and end coordinate, and their index in the exprs(davidTiling) data matrix. Sort them by midpoint.

```
> chrstrd = "1.+"
> what = c("start", "end", "index", "unique")
> prbs = do.call("data.frame", mget(paste(chrstrd, what, sep = "."),
+     probeAnno))
> colnames(prbs) = what
> prbs$mid = (prbs$start + prbs$end)/2
> prbs = prbs[order(prbs$mid), ]
```

Throw out the missing (NA) values.

```
> numna = rowSums(is.na(exprs(xn)[prbs$ind, ]))
> stopifnot(all(numna %in% c(0, ncol(xn))))
> prbs = prbs[numna == 0, ]
```

3.1.1 Avoid oversampling

Figure 1 shows that the spacing between the probes is not completely regular, in particular, repetitive regions are highly oversampled. We subsample the probes, the result of this is shown in the comparison between Figures 1b and 1c.

```
> sprb = prbs[sampleStep(prbs$mid, step = 7), ]
```

```
> par(mfrow = c(3, 1))
> hist(prbs$mid, col = "mistyrose", 100, main = "(a)")
> barplot(table(diff(prbs$mid)), main = "(b)")
> barplot(table(diff(sprb$mid)), main = "(c)")
```

3.2 Call the segmentation algorithm

The segmentation algorithm needs two parameters, `maxseg`, the maximum number of segments that the algorithm is going to consider, and `maxk`, the maximum length of individual segments. We choose `maxseg` to be quite high, such that it corresponds to an *average* length per segment of 750 bases. The algorithm will calculate all optimal segmentations with $1, 2, \ldots,$ `maxseg` segments, and we can still later choose our prefered 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. Note that there is no minimum length restriction for the segments.

```
> maxseg = round(sprb$end[nrow(sprb)]/750)
```

```
[1] 307
```

```
> y = exprs(xn)[sprb$ind, xn$nucleicAcid == "poly(A) RNA", drop = FALSE]
> segw = segment(y, maxseg = maxseg, maxk = 3000)
```

We also add additional information to the object that was not used for the actual segmentation, but will be useful for the visualization: into the slot `x`, the x-coordinates of the probes, and into the slot `flag`, the uniqueness status of the probes (0 iff the probe has exactly one match in the genome):

```
> segw@x = sprb$mid
> segw@flag = sprb$unique
```

Having to access the `x` and `flag` slots directly, as in the code above, is a bit unelegant. I intend to provide accessor functions in subsequent versions of the package.

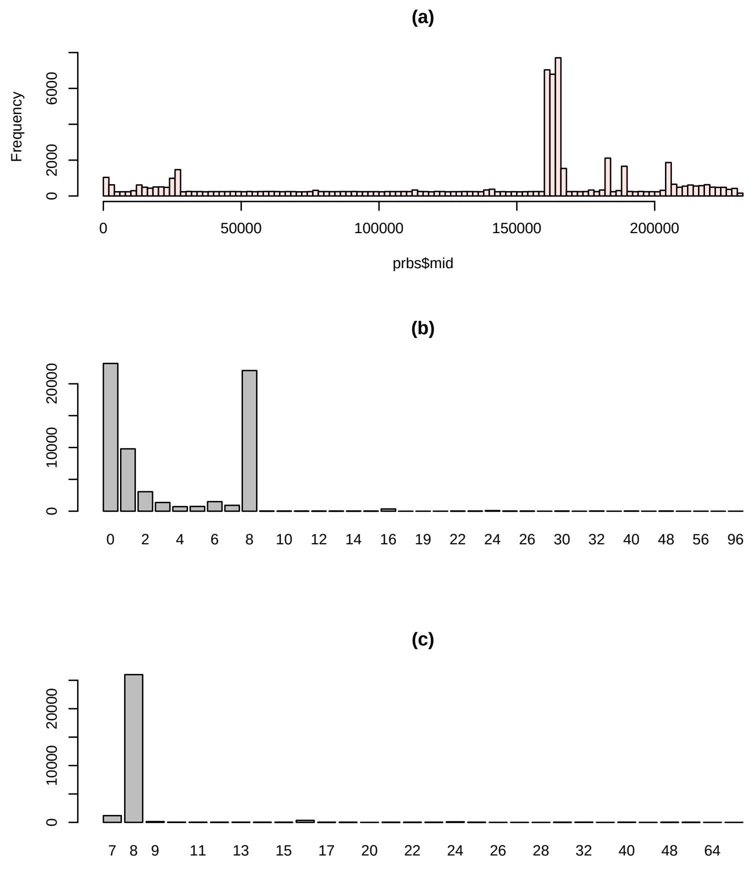

Figure 1: (a): Histogram of probe midpoints along the "+" strand of chromosome 1. There are some probe dense regions in particular around 160,000. The sequence of that region is repeated multiple times in the genome, and due to the way the chip was designed, there are also a lot of probes (more than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

3.3 Calculate confidence intervals

This is simply a call to the `confint` method of the *segmentation* class.

```
> nseg = round(sprb$end[nrow(sprb)]/1500)
> confintLevel = 0.95
> segwi = confint(segw, parm = nseg, level = confintLevel)
```

Now we are ready to have a look at the result via the `plot` method of the *segmentation* class. The plot is shown in Figure 2.

```
> plot(segwi, nseg, pch = ".", xlim = c(0, 40000))
```

```
Note: slot 'y' has more than one column, calculating 'rowMeans'
```

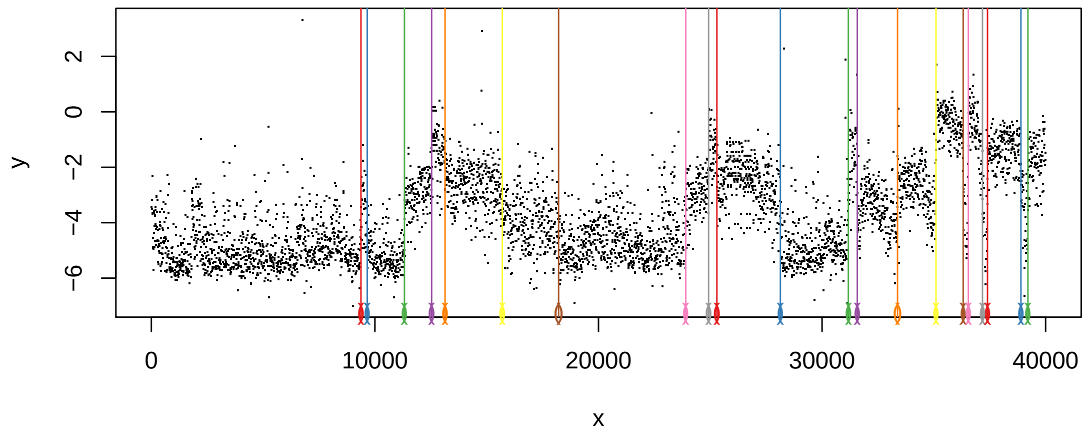

Figure 2: Segmentation with confidence intervals.

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

$$\text{AIC} = -2\log L + 2p \tag{2}$$

$$\text{BIC} = -2\log L + p\log n \tag{3}$$

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

$$\log L_{\mathrm{AIC}} = \log L - p \tag{4}$$
$$\log L_{\mathrm{BIC}} = \log L - \frac{p}{2}\log n \tag{5}$$

We plot them as functions of S, see Figure 3

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

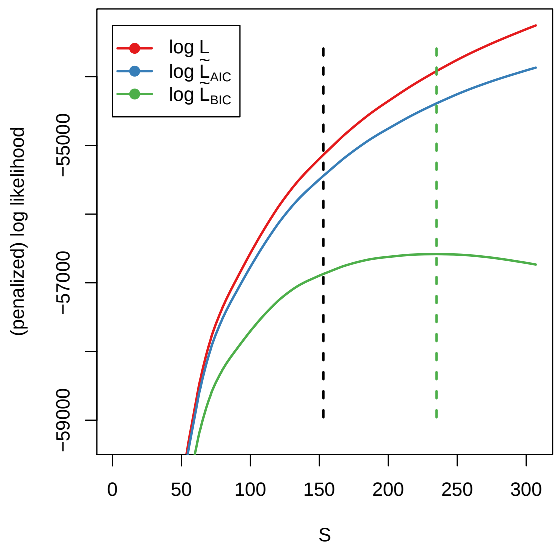

Figure 3: Model selection: log-likelihood and two versions of penalized log-likelihood (AIC and BIC) as a function of the number of segments S. Vertical dashed green bar corresponds to optimal $\log L_{\mathrm{BIC}}$, vertical dashed grey bar to our "subjective" choice of average segment length 1,500 bases.

3.5 Size of the confidence intervals as a function of S

```
> segwj = confint(segw, parm = as.integer(c(112, 153, 194, 235,
+     276)), level = confintLevel)
> nBp = which(segwj@hasConfint)
> confintwidths = lapply(segwj@breakpoints[nBp], function(m) (m[,
+     3] - m[, 1]))

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

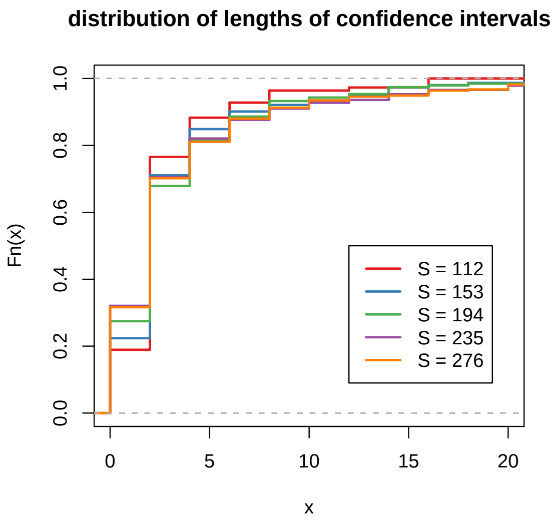

Figure 4: Size of the confidence intervals as a function of S. Cumulative distribution functions (CDFs) for the distributions of confidence interval widths for S =112, 153, 194, 235, 276. For larger S, the confidence intervals are wider.

3.6 Definition of segFun

For the subsequent considerations, it will be useful to define the function `segFun`. It encapsulates the complete set of segmentation computations, as shown above, for one chromosome strand. Its result is a *segmentation* object with confidence intervals.

```
> segFun = function(chrstrd, nrBasesPerSegment = 1500) {
+     writeLines(sprintf("Working on %s", chrstrd), con = "segmentation.log")
+     what = c("start", "end", "index", "unique")
+     prbs = do.call("data.frame", mget(paste(chrstrd, what, sep = "."),
+         probeAnno))
+     colnames(prbs) = what
+     prbs$mid = (prbs$start + prbs$end)/2
+     prbs = prbs[order(prbs$mid), ]
+     numna = rowSums(is.na(exprs(xn)[prbs$ind, ]))
+     stopifnot(all(numna %in% c(0, ncol(xn))))
+     prbs = prbs[numna == 0, ]
+     sprb = prbs[sampleStep(prbs$mid, step = 7), ]
+     nseg = round(sprb$end[nrow(sprb)]/nrBasesPerSegment)
+     y = exprs(xn)[sprb$ind, xn$nucleicAcid == "poly(A) RNA",
+         drop = FALSE]
+     s = segment(y, maxseg = nseg, maxk = 3000)
+     s@x = sprb$mid
+     s@flag = sprb$unique
+     confint(s, parm = nseg, level = confintLevel)
+ }
```

3.7 Using the `plotAlongChrom` function for more elaborate displays.

Since the data in the *davidTiling* package are strand-specific, we can do the segmentation for the "-" strand as well and produce the along-chromosome plot shown in Figure 5.

For Figure 5, we call `segFun` on the "-" strand of chromosome 1. For Figures 6 and 7, we also call it on a number of other chromosomes.

This computation will take a couple of hours (about 18h on mine). Note that the `for`–loop below can be trivially parallelized since the computations for different chromosome strands are independent of each other. A simple synchronization mechanism through creation of a *lock file* is already provided in the code example below.

```
> toDo = c("1.-", "2.+", "2.-", "5.+", "5.-", "9.+", "9.-", "13.+",
+     "13.-", "14.+", "14.-", "15.+", "15.-")
> for (w in toDo) {
+     fn = paste(w, "rda", sep = ".")
+     if (!file.exists(fn)) {
+         writeLines(date(), con = fn)
+         assign(w, segFun(w))
+         save(list = w, file = fn, compress = TRUE)
+     }
+ }
```

Finally, we collect all results in the environment `segObj`.

```
> segObj = new.env(parent = baseenv())
> assign("1.+", segwi, segObj)
> for (w in toDo) {
+     load(paste(w, "rda", sep = "."))
+     assign(w, get(w), segObj)
+ }
> 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. In the following, the code to

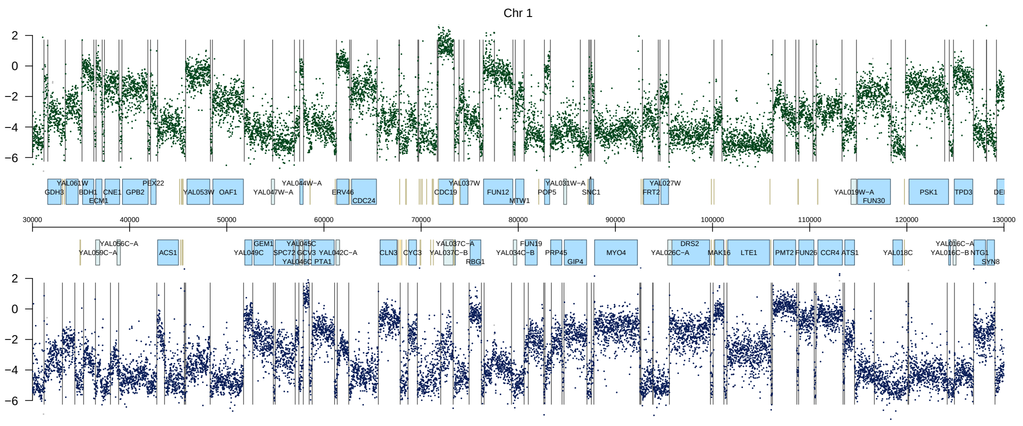

Figure 5: Along-chromosome plot similar to Figure 1 in [1].

generate Figure 1 in [1].

```
> grid.newpage()
> plotAlongChrom(segObj = segObj, chr = 1, coord = c(30, 130) *
+     1000, ylim = ylim, gff = myGff, showConfidenceIntervals = FALSE,
+     featureNoLabel = c("uORF", "binding_site", "TF_binding_site"),
+     doLegend = FALSE, main = "")
```

In the following, the code to generate Figure 2 in [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, height = 1.02, just = c("left",
```

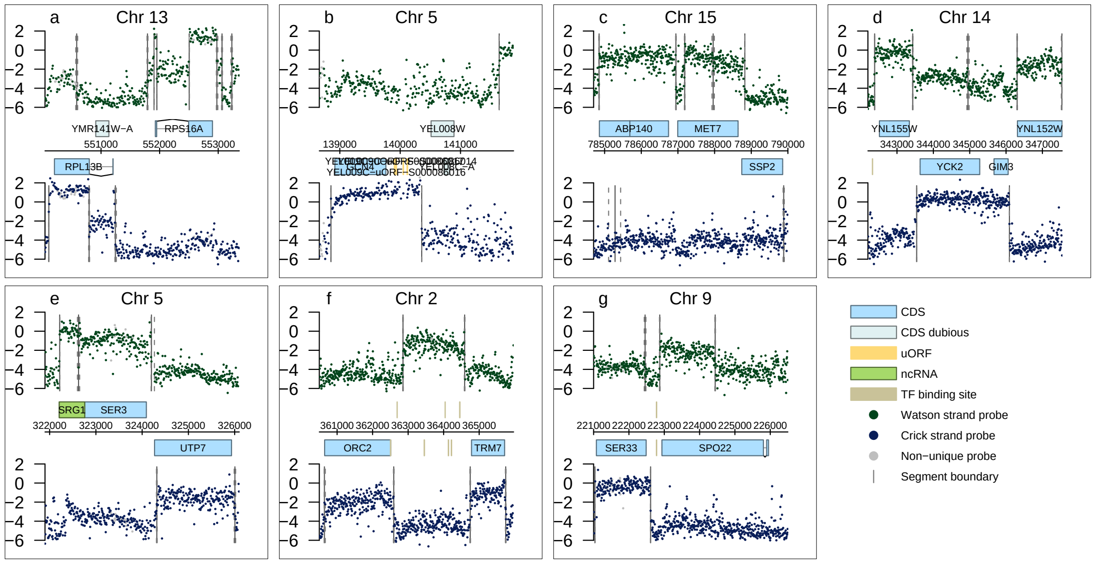

Figure 6: Along-chromosome plots similar to Figure 2 in [1].

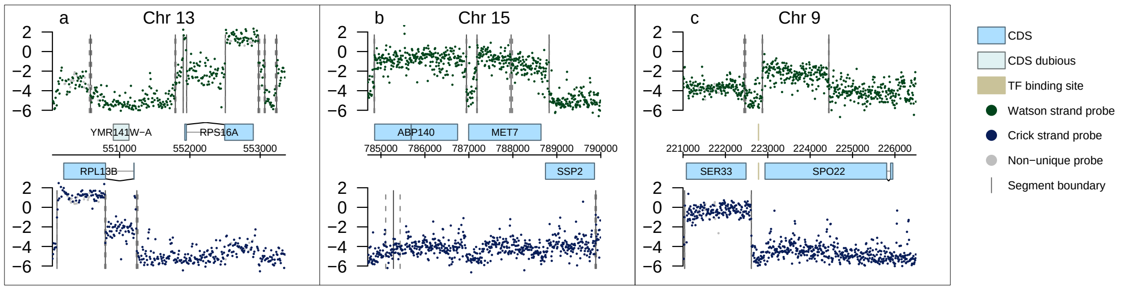

Figure 7: Along-chromosome plots similar to Figure 2 in [2].

```
+         "bottom"), default.units = "npc", gp = gpar(lwd = 0.2))
+     plotAlongChrom(..., segObj = segObj, 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(-0.3, +0.3), 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 = 0.7))
+     popViewport()
+ }
> dx = 0.2
> 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))))
> myPlot(1, 2, chr = 13, coord = c(550044, 553360), main = "a")
> myPlot(1, 4, chr = 5, coord = c(138660, 141880), main = "b")
> myPlot(1, 6, chr = 15, coord = c(784700, 790000), main = "c")
> myPlot(1, 8, chr = 14, coord = c(342200, 347545), main = "d")
> myPlot(3, 2, chr = 5, coord = c(321900, 326100), main = "e")
> myPlot(3, 4, chr = 2, coord = c(360500, 365970), main = "f")
> 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 in [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))))
> myPlot(1, 2, chr = 13, coord = c(550044, 553360), main = "a")
> myPlot(1, 4, chr = 15, coord = c(784700, 790000), main = "b")
> myPlot(1, 6, chr = 9, coord = c(221000, 226500), main = "c")
> myLegend(1, 8, 2)
> popViewport()
```

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. 9, 10

[2] Wolfgang Huber, Joern Toedling and Lars M. Steinmetz Transcript mapping with oligonucleotide high-density tiling arrays. *Bioinformatics*, 2006. 10, 12