Wave correction for arrays

Eitan Halper-Stromberg and Robert B. Scharpf

October 13, 2015

Abstract

`ArrayTV` is a GC correction algorithm for microarray data designed to mitigate the appearance of waves. This work is inspired by a method for mitigating waves in sequencing data [1]. We find the genomic window for computing GC that best captures the dependence of signal intensity on GC content for each array using a score similar to the total variance (TV) statistic defined in [1]. The correction each probe receives is the mean signal intensity of all probes sharing the same local GC. We center windows at the starting position of each probe.

1 Adjusting copy number estimates for GC waves

Importing data. The wave correction methods in `ArrayTV` handle simple matrices with rows indexing markers and columns indexing samples, as well as more complex data structures commonly used to process Affymetrix and Illumina platforms made available in the `oligoClasses` package. In this vignette, we illustrate our approach with a single female lymphoblastoid cell line sample assayed on the Agilent 1M , Affymetrix 2.7M, and NimbleGen 2.1M platforms (available in full from http://rafalab.jhsph.edu/cnvcomp, spike-in tube 2)[2]. For each platform, we provide a matrix. The rows of the matrix correspond to markers on chromosome 15 and the columns correspond to the physical position of the first nucleotide of the marker using UCSC build hg18 (note: for genotyping arrays, the position should be adjusted to the start of the marker and not the location of the SNP) and preprocessed signal intensities that we expect to be proportional to the copy number as well as sequence artifacts such as GC content. Agilent and NimbleGen signal intensities were preprocessed by taking the log ratio of loess normalized raw signal. Affymetrix performed their own preprocessing to obtain log R ratios. Hereafter, we refer to the preprocessed signal as M values. To keep the size of this package small, we provide the M values as integers ($M \times 1000$). In the following code, we load the data for each platform, inspect the first few rows, and transform the M values for each platform to the original scale.

```
> library(ArrayTV)
> ### parallelization will be discussed later but for now we provide the following
> ### command to avoid warnings
> if(require(doParallel)){
+ cl <- makeCluster(1)
+ registerDoParallel(cl)
+ }
> path <- system.file("extdata", package="ArrayTV")
> load(file.path(path, "array_logratios.rda"))
> head(agilent)
```

```
     position    M
[1,] 18362555  233
[2,] 18427151  -29
[3,] 18432556   50
[4,] 18450562    7
[5,] 18452491   14
[6,] 18692865 -604
```

```
> head(affymetrix)

     position    M
[1,]      601  -47
[2,]     1189 -131
[3,]     3590 -290
[4,]     4668 -376
[5,]     5942  109
[6,]     6677 -405

> head(nimblegen)

     position    M
[1,] 18305309   55
[2,] 18308562 -173
[3,] 18314533 -115
[4,] 18317177  148
[5,] 18318410   -4
[6,] 18321680   -6

> agilent[, "M"] <- agilent[, "M"]/1000
> affymetrix[, "M"] <- affymetrix[, "M"]/1000
> nimblegen[, "M"] <- nimblegen[, "M"]/1000
```

To determine the optimal window(s) for GC correction, we generate TV scores for several window sizes. Again, we assume that the positions denote the start location of each probe. While our analysis in this vignette is limited to chromosome 15 for computational reasons, in practice we recommend implementing the wave correction on all autosomes simultaneously. We specify the windows to search by providing two vectors that indicate the maximum window size and the increment:

```
> max.window <- c(100,10e3,1e6)
> increms <- c(20, 2000, 200e3)
```

The above specification indicates that we will compute TV scores for three sequential window sizes as follows. The first series of windows has a maximum window extension of 100 from center, (`max.window[1]`) (effective maximum window size is therefore 200 since we extend in both directions) and increment extension value 20 (`increms[1]`) again, extending from the center of the window, indicating that windows extending 20, 40, ..., and 100 from the probe starts will be evaluated. For the second series, the maximum window extension is 10e3 (`max.window[2]`) and window extensions 2000, 4000, ..., 10e3 will be evaluated since the increment value is 2000 (`increms[2]`). For each of the three series, the number of windows to be evaluated is 5. The sequence for each series must have the same length.

Evaluating TV scores as a function of window size for computing GC. The class of the first argument to the `gcCorrect` method determines the dispatch to the low-level function `gcCorrectMain`. Any of the arguments to `gcCorrectMain` can be passed through the `...` operator and we refer the user to the documentation for `gcCorrectMain` for additional arguments and `gcCorrect` for the classes of objects handled by the `gcCorrect` method. Computing the GC content of the sequence for each marker requires that the appropriate BSgenome package is loaded. For example, here the package `BSgenome.Hsapiens.UCSC.hg18` is loaded internally. The following codechunk performs GC correction on the M values for the NimbleGen, Agilent, and Affymetrix platforms, respectively, for the windows specified previously.

```
> nimcM1List <- gcCorrect(object=nimblegen[, "M", drop=FALSE],
+                         chr=rep("chr15", nrow(nimblegen)),
+                         starts=nimblegen[, "position"],
+                         increms=increms,
+                         maxwins=max.window,
```

```
+                          build='hg18')
> afcM1List <- gcCorrect(affymetrix[, "M", drop=FALSE],
+                        chr=rep("chr15", nrow(affymetrix)),
+                        starts=affymetrix[, "position"],
+                        increms=increms,
+                        maxwins=max.window,
+                        build="hg18")

> ## We load the corrected NimbleGen values computed during unit-testing and Affy
> ## values, pre-computed, to save time
> load(file=file.path(path,"nimcM1List.rda"))
> load(file=file.path(path,"afcM1List.rda"))
> agcM1List <- gcCorrect(agilent[, "M", drop=FALSE],
+                        chr=rep("chr15", nrow(agilent)),
+                        starts=agilent[, "position"],
+                        increms=increms,
+                        maxwins=max.window,
+                        build="hg18")
> if(require(doParallel))
+     stopCluster(cl)
```

To plot the TV score for each of the windows evaluated, we concatenate the results from each platform into a list and then convert the elements representing the TVscores into a data.frame. Plotting is based on the size of the genomic windows used to compute GC (small, medium, and large).

```
> results <- list(nimcM1List,agcM1List,afcM1List)
> tvscores <- do.call("rbind", lapply(results, "[[", "tvScore"))[, 1]
> tv.df <- data.frame(tv=tvscores, platform=rep(c("NimbleGen", "Agilent", "Affy 2.7M"), each=15),
+                     window=c(seq(20, 100, 20),
+                     seq(2000, 10e3, 2e3), seq(2e5, 1e6, 2e5))*2)
> op <- par(no.readonly = TRUE)
> par(las=1, mar=c(6, 4, 3, 2)+0.1)
> with(tv.df, plot(log10(window), tv, pch=20, col=tv.df$platform,
+                  xaxt="n", xlab="window (bp)", ylab="TV score", cex=1.2))
> invisible(lapply(split(tv.df, tv.df$platform), function(x)
+                   points(log10(x$window),x$tv, type="l",col=x$platform, lwd=1.5)))
> axis(1, at=log10(tv.df$window), labels=FALSE) #labels=FALSE, tick=TRUE)a
> text(x=log10(tv.df$window), par("usr")[3]-0.005,
+      labels=tv.df$window,
+      srt=45, xpd=TRUE, cex=0.7, adj=1)
> legend("bottomright", legend=unique(tv.df$platform),
+        pch=20, col=unique(tv.df$platform), lwd=1.2)
> par(op)
```

View corrected and uncorrected M values

Our GC-adjusted signal was computed when we called `gcCorrect` and is now stored in objects `nimcM1List`, `agcM1List`, and `afcM1List`. Here we create a *data.frame* to store corrected and uncorrected signal, ahead of plotting.

```
> wave.df <- data.frame(position=c(rep(nimblegen[, "position"]/1e6, 2),
+                       rep(agilent[, "position"]/1e6, 2),
+                       rep(affymetrix[, "position"]/1e6,2)),
+                       r=c(nimblegen[, "M"],
+                       nimcM1List[['correctedVals']],
```

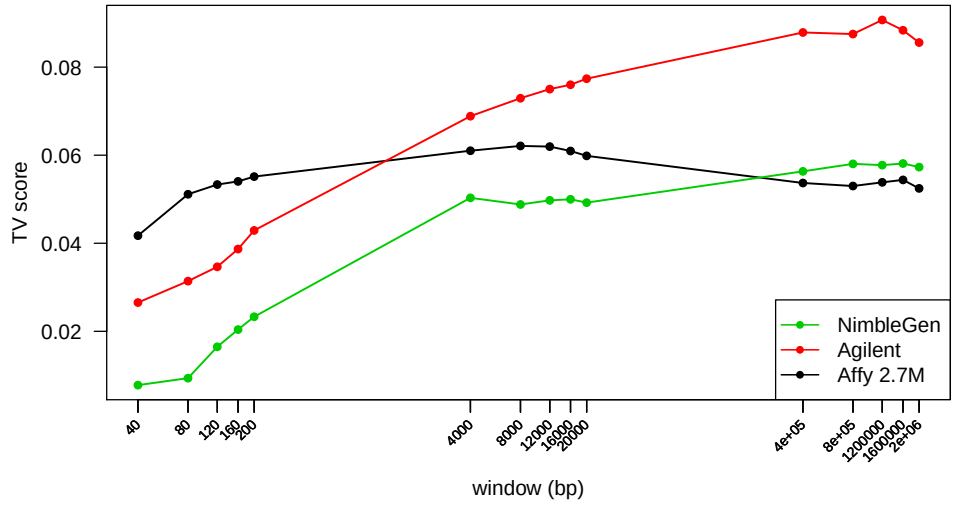

Figure 1: TV scores for three array platforms indicated by color. For the Agilent and NimbleGen platforms, the window selected for wave correction is relatively large ($\approx$ 1.2Mb). For the Affymetrix platform, the optimal window is $\approx$ 8kb. Often, the optimal window is closer to the size of the probe or the PCR fragment. A correction for both large and small windows could be acheived by a second iteration of the `gcCorrect` function.

```
+                        agilent[,"M"],
+                        agcM1List[['correctedVals']],
+                        affymetrix[,"M"], afcM1List[['correctedVals']]),
+                        platform=rep(c(rep("Nimblegen", 2),
+                        rep("Agilent", 2),
+                        rep("Affymetrix", 2)), c(rep(length(nimblegen[, "M"]),2),
+                                                 rep(length(agilent[,"M"]),2),
+                                                 rep(length(affymetrix[,"M"]),2))),
+                        wave.corr=c(rep("raw", length(nimblegen[, "M"])),
+                        rep("ArrayTV", length(nimblegen[, "M"])),
+                        rep("raw", length(agilent[,"M"])),
+                        rep("ArrayTV", length(agilent[,"M"])),
+                        rep("raw", length(affymetrix[,"M"])),
+                        rep("ArrayTV",length(affymetrix[,"M"])))
> wave.df$platform <- factor(wave.df$platform, levels=c("Nimblegen",
+                                                       "Agilent", "Affymetrix"))
> wave.df$wave.corr <- factor(wave.df$wave.corr, levels=c("raw", "ArrayTV"))
> ## remove some points to make subsequent figure smaller
> wave.df <- wave.df[seq(1,nrow(wave.df),4),]
```

We use the R package `lattice` to visualize the M values before and after correction for the three platforms (Figure 2)

```
>     ## For each of the 3 example platforms, the uncorrected signal appears immediately above the
>     ## corrected signal, across chromosome 15. A smoothed loess line may be drawn through each
>     ## by uncommenting the commented line in the xyplot function
> require("lattice") && require("latticeExtra")
```

```
[1] TRUE
```

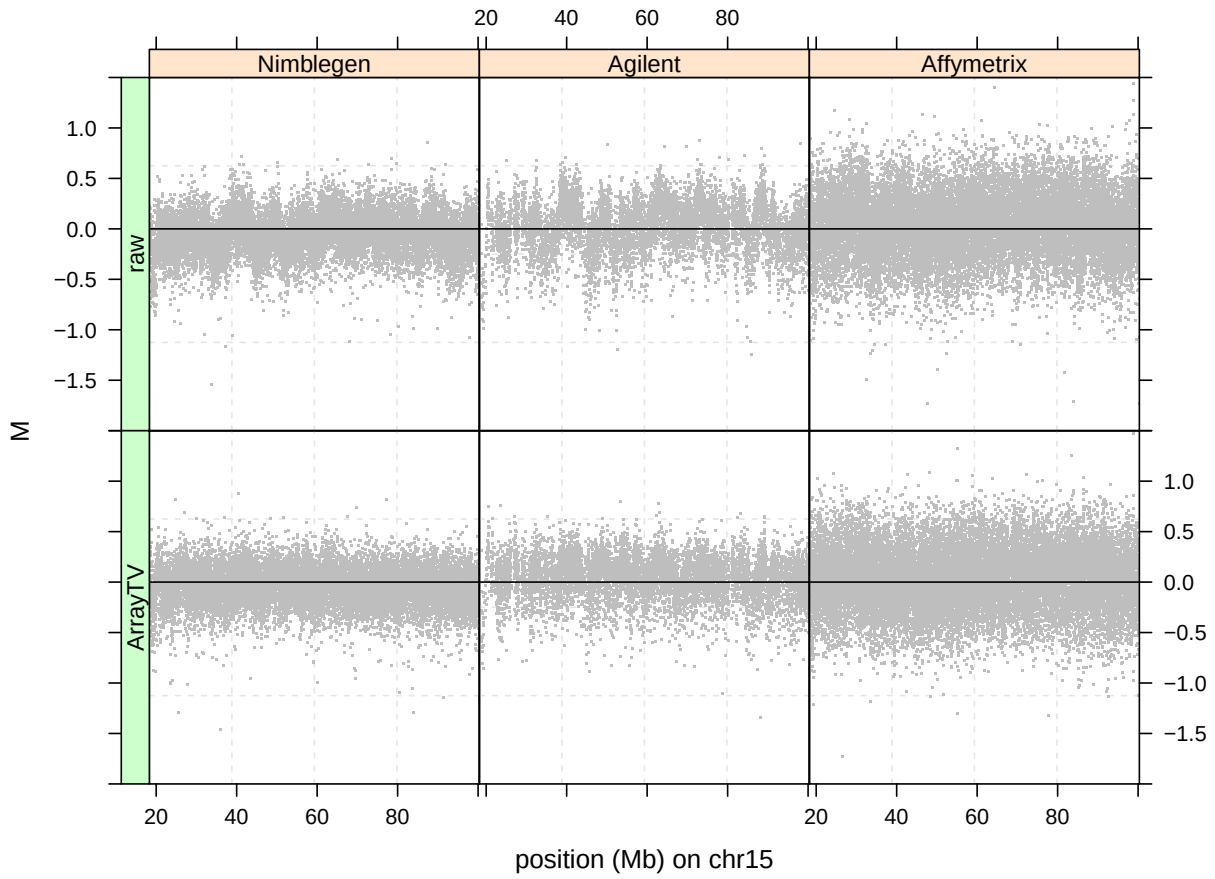

Figure 2: Comparison of M values before and after `ArrayTV` wave correction (rows) for three array platforms (columns).

```
> p <- xyplot(r~position|platform+wave.corr, wave.df, pch=".", col="gray",
+             ylim=c(-2,1.5), xlim=range(nimblegen[, "position"]/1e6),
+             ylab="M", xlab="position (Mb) on chr15",
+             scales=list(y=list(tick.number=8)),
+             panel=function(x,y,subscripts,...){
+                     panel.grid(lty=2)
+                     panel.xyplot(x, y, ...)
+                     panel.abline(h=0)
+                     ## panel.loess(x, y, span=1/20, lwd=2, col="black")
+             }, par.strip.text=list(cex=0.9),
+             layout=c(2,3),
+             as.table=TRUE)

> p2 <- useOuterStrips(p)
> print(p2)
```

2 Spike-In Example

To demonstrate that the GC-correction does not remove true CNV signals, we illustrate our approach on a spike-in datasets containing a small amplification. While the signal for the spike-in is subtle, the main point here is that the small inflection is not effected by the GC correction. The amplification was spiked in to each of the three array platforms as described in Halper-Stromberg *et al.* [2]. As the segmentation of the data using circular binary segmentation (CBS; [3]) is only for visualization and is not critical to the functionality of this package, the following code chunk is static (not evaluated) to reduce computation:

```
> if(require("DNAcopy")){
+         cbs.segs <- list()
+         platforms <- c("Nimblegen", "Agilent", "Affymetrix")
+         correction <- c("raw", "ArrayTV")
+         Ms <- list(nimblegen[, "M"], nimcM1List[['correctedVals']],
+                    agilent[, "M"], agcM1List[['correctedVals']],
+                    affymetrix[, "M"], afcM1List[['correctedVals']])
+         starts <- list(nimblegen[, "position"],
+                        agilent[, "position"],
+                        affymetrix[, "position"])
+         k <- 0
+         for(i in 1:3){
+                 for(j in 1:2){
+                         k <- k+1
+                         MsUse <- Ms[[k]]
+                         chrUse <- rep("chr15", length(MsUse))
+                         startsUse <- starts[[i]]
+                         toUse <- !(duplicated(startsUse))
+                         cna1 <- CNA(as.matrix(MsUse[toUse]),
+                                     chrUse[toUse],
+                                     startsUse[toUse],
+                                     data.type="logratio",
+                                     paste(platforms[i], correction[j], sep="_"))
+                         smooth.cna1 <- smooth.CNA(cna1)
+                         cbs.segs[[k]] <- segment(smooth.cna1, verbose=1,min.width = 5)
+                 }
+             }
+         cbs.out <- do.call("rbind", lapply(cbs.segs, "[[", "output"))
+     }
```

We load the results from the above computation into our workspace as follows:

```
> load(file.path(path, "cbs_out.rda"))
```

To facilitate downstream analyses such as visualization of the results, we coerce the results from the CBS algorithm to an object of class `GRanges`.

```
> ids <- cbs.out$ID
> ## create vectors indicating corrected/uncorrected and platform for each cbs segment
> platforms <- gsub("_raw", "", ids)
> platforms <- gsub("_ArrayTV", "", platforms)
> correction <- gsub("Nimblegen_", "", ids)
> correction <- gsub("Affymetrix_", "", correction)
> correction <- gsub("Agilent_", "", correction)
> ## store cbs segments in a GRanges object with designations for corrected/uncorrected
> ## and platform
```

```
> cbs.segs <- GRanges(rep("chr14", nrow(cbs.out)),
+                     IRanges(cbs.out$loc.start, cbs.out$loc.end),
+                     numMarkers=cbs.out$num.mark,
+                     seg.mean = cbs.out$seg.mean,
+                     platform = platforms,
+                     correction=correction)
> ## separate corrected and uncorrected cbs segments int separate objects
> cbs.segs.raw <- cbs.segs[cbs.segs$correction=="raw", ]
> cbs.segs.arrayTV <- cbs.segs[cbs.segs$correction=="ArrayTV", ]
> ## create GRanges objects with M values, platform, and
> ## corrected/uncorrected designation
> marker.data <- GRanges(rep("chr14", nrow(wave.df)),
+                     IRanges(wave.df$position*1e6, width=1),
+                     numMarkers=1,
+                     seg.mean = wave.df$r,
+                     platform=wave.df$platform,
+                     correction=wave.df$wave.corr)
> marker.raw <- marker.data[marker.data$correction=="raw",]
> marker.arrayTV <- marker.data[marker.data$correction=="ArrayTV",]
> ## populate cbs.seg metadata column by joining M values with CBS segments by location
> olaps1 <- findOverlaps(marker.raw, cbs.segs.raw, select="first")
> marker.raw$cbs.seg <- cbs.segs.raw$seg.mean[olaps1]
> olaps2 <- findOverlaps(marker.arrayTV, cbs.segs.arrayTV, select="first")
> marker.arrayTV$cbs.seg <- cbs.segs.arrayTV$seg.mean[olaps2]
> ## populate SpikeIn metadata column if a marker falls within the Spiked-in region
> spikeIn <- IRanges(start=45396506,end=45537976)
> spikeinStart <- 45396506; spikeinEnd <- 45537976
> marker.raw$spikeIn <- marker.raw$cbs.seg
> marker.arrayTV$spikeIn <- marker.arrayTV$cbs.seg
> index1 <- which(is.na(findOverlaps(unlist(ranges(marker.raw)),
+                                    spikeIn,select="first")))
> index2 <- which(is.na(findOverlaps(unlist(ranges(marker.arrayTV)),
+                                    spikeIn,select="first")))
> marker.raw$spikeIn[index1] <- NA
> marker.arrayTV$spikeIn[index2] <- NA
> ## put GRanges objects into a dataframe ahead of plotting
> rawd <- as.data.frame(marker.raw)
> atvd <- as.data.frame(marker.arrayTV)
> rawd <- rbind(rawd, atvd)
> rawd$platform <- factor(rawd$platform, levels=c("Nimblegen",
+                                          "Agilent", "Affymetrix"))
> rawd$correction <- factor(rawd$correction, levels=c("raw", "ArrayTV"))
> ## remove some points to make subsequent figure smaller
> rawd <- rawd[seq(1,nrow(rawd),2),]
```

Black points in Figure 3 represent signal intensities for each probe, and red line segments correspond to the segmentation of the intensities from CBS. The red line segments demarcate the spike-in. As in the previous figure, raw signal intensities appear immediately above the corrected intensities for each platform

```
> p2 <- xyplot(seg.mean+cbs.seg+spikeIn~start/1e6|platform+correction, rawd,
+              ylim=c(-1.5, 1.5),
+              pch=".",
+              xlim=c(spikeinStart/1e6-1.5,spikeinEnd/1e6+1.5),
+              ylab="M",
+              xlab="position (Mb)",##pch=c(".", NA,NA),
```

```
+               col=c('gray','black','red'),
+               distribute.type=TRUE,
+               type=c('p','l','l'),lwd=c(NA,2,3),
+               scales=list(y=list(tick.number=8)),
+               par.strip.text=list(cex=0.9), layout=c(2,3), as.table=TRUE)
> p3 <- useOuterStrips(p2)

> print(p3)
```

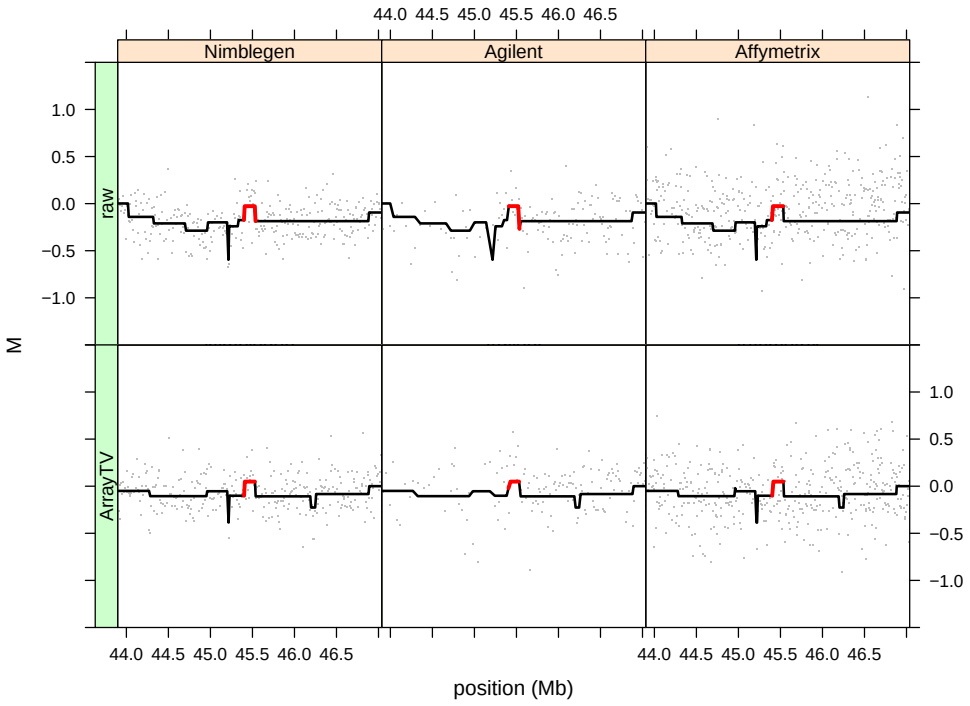

Figure 3: Comparison of CBS segments (lines) and M values (points) before and after `ArrayTV` wave correction (rows) for three array platforms (columns) in the vicinity of a known amplification of copy number 2.5 (red).

3 Parallelization

When multiple cores are available, computation can proceed in parallel for computing the GC content for the various window sizes specified. To enable parallelization across 8 cores using the `doParallel` package, one could execute the following (unevaluated) code chunk prior to calling the `gcCorrect` function:

```
> if(require(doParallel)){
+ cl <- makeCluster(8)
+ registerDoParallel(cl)
+ }
> ## ... execute gcCorrect
> if(require(doParallel))
+     stopCluster(cl)
```

4 Session Information

- R version 3.2.2 (2015-08-14), `x86_64-pc-linux-gnu`
- Locale: `LC_CTYPE=en_US.UTF-8`, `LC_NUMERIC=C`, `LC_TIME=en_US.UTF-8`, `LC_COLLATE=C`, `LC_MONETARY=en_US.UTF-8`, `LC_MESSAGES=en_US.UTF-8`, `LC_PAPER=en_US.UTF-8`, `LC_NAME=C`, `LC_ADDRESS=C`, `LC_TELEPHONE=C`, `LC_MEASUREMENT=en_US.UTF-8`, `LC_IDENTIFICATION=C`
- Base packages: base, datasets, grDevices, graphics, methods, parallel, stats, stats4, utils
- Other packages: ArrayTV 1.8.0, BSgenome 1.38.0, BSgenome.Hsapiens.UCSC.hg18 1.3.1000, BiocGenerics 0.16.0, Biostrings 2.38.0, GenomeInfoDb 1.6.0, GenomicRanges 1.22.0, IRanges 2.4.0, RColorBrewer 1.1-2, S4Vectors 0.8.0, XVector 0.10.0, doParallel 1.0.8, foreach 1.4.3, iterators 1.0.8, lattice 0.20-33, latticeExtra 0.6-26, rtracklayer 1.30.0
- Loaded via a namespace (and not attached): Biobase 2.30.0, BiocInstaller 1.20.0, BiocParallel 1.4.0, DBI 0.3.1, DNAcopy 1.44.0, GenomicAlignments 1.6.0, RCurl 1.95-4.7, RSQLite 1.0.0, Rsamtools 1.22.0, SummarizedExperiment 1.0.0, XML 3.98-1.3, affyio 1.40.0, bit 1.1-12, bitops 1.0-6, codetools 0.2-14, compiler 3.2.2, ff 2.2-13, futile.logger 1.4.1, futile.options 1.0.0, grid 3.2.2, lambda.r 1.1.7, oligoClasses 1.32.0, tools 3.2.2, zlibbioc 1.16.0

References

[1] Yuval Benjamini and Terence P. Speed. Summarizing and correcting the GC content bias in high-throughput sequencing. *Nucleic Acids Res*, 40(10):e72, May 2012.

[2] Eitan Halper-Stromberg, Laurence Frelin, Ingo Ruczinski, Robert Scharpf, Chunfa Jie, Benilton Carvalho, Haiping Hao, Kurt Hetrick, Anne Jedlicka, Amanda Dziedzic, Kim Doheny, Alan F. Scott, Steve Baylin, Jonathan Pevsner, Forrest Spencer, and Rafael A. Irizarry. Performance assessment of copy number microarray platforms using a spike-in experiment. *Bioinformatics*, 27(8):1052–1060, 2011.

[3] Adam B Olshen, E. S. Venkatraman, Robert Lucito, and Michael Wigler. Circular binary segmentation for the analysis of array-based DNA copy number data. *Biostatistics*, 5(4):557–72, Oct 2004.