# GenoGAM: Genome-wide generalized additive models

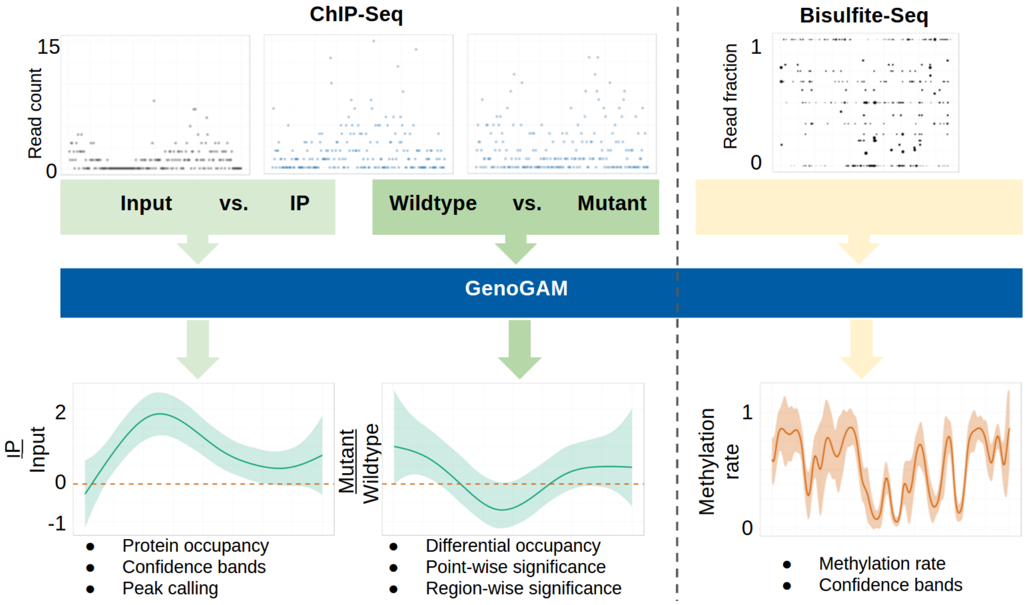
Georg Stricker[1], Julien Gagneur[1]

[1] Technische Universität München, Department of Informatics, Garching, Germany

August 11, 2016

**Abstract**

Many genomic assays lead to noisy observations of a biological quantity of interest varying along the genome. This is the case for ChIP-Seq, for which read counts reflect local protein occupancy of the ChIP-ed protein. The *GenoGAM* package allows statistical analysis of genome-wide data with smooth functions using generalized additive models. It provides methods for the statistical analysis of ChIP-Seq data including inference of protein occupancy, and pointwise and region-wise differential analysis. Estimation of dispersion and smoothing parameters is performed by cross-validation. Scaling of generalized additive model fitting to whole chromosomes is achieved by parallelization over overlapping genomic intervals. This vignette explains the use of the package for typical ChIP-Seq analysis tasks.



**GenoGAM version:** 1.0.3

If you use *GenoGAM* in published research, please cite:

Stricker, *et al.* **Genome-wide generalized additive models** *bioRxiv*

# Contents

# 1 Standard ChIP-Seq analysis

This version of *GenoGAM* only supports smoothing and differential analysis of ChIP-Seq data.

## 1.1 Goal of the analysis

A small dataset is provided to illustrate the ChIP-Seq functionalities. This is a subset of the data published by Thornton et al[1], who assayed histone H3 Lysine 4 trimethylation (H3K4me3) by ChIP-Seq comparing wild type yeast versus a mutant with a truncated form of Set1, the yeast H3 Lysine 4 methylase. The goal of this analysis is the identification of genomic positions that are significantly differentially methylated in the mutant compared to the wild type strain.

To this end, we will build a *GenoGAM* model that models the logarithm of the expected ChIP-seq fragment counts $y$ as sums of smooth functions of the genomic position $x$. Specifically, we write (with simplified notations) that:

$$\log(\mathrm{E}(y)) = f(x) + \text{genotype} \times f_{\text{mutant/wt}}(x) \tag{1}$$

where genotype is 1 for data from the mutant samples, and 0 for the wild type. Here the function $f(x)$ is the reference level, i.e. the log-rate in the wild type strain. The function $f_{\text{mutant/wt}}(x)$ is the log-ratio of the mutant over wild-type. We will then statistically test the null hypothesis $f_{\text{mutant/wt}}(x) = 0$ at each position $x$. In the following we show how to build the dataset, perform the fitting of the model and perform the testing.

## 1.2 Registering a parallel backend

The parallel backend is registered using the *BiocParallel* package. See the documentation in *BiocParallel* for the correct use. Also note, that *BiocParallel* is just an interface to multiple parallel packages. For example in order to use *GenoGAM*on a cluster, the *BatchJobs* package might be required. The parallel backend should be registered before creating the *GenoGAM* class, as this setup will be used throughout the analysis.

```
library(GenoGAM)

## On multicore machines by default the number of available cores - 2 are registered
BiocParallel::registered()[1]

## $MulticoreParam
## class: MulticoreParam
##   bpisup: FALSE; bpworkers: 4; bptasks: 0; bpjobname: BPJOB
##   bplog: FALSE; bpthreshold: INFO; bpstopOnError: TRUE
##   bptimeout: 2592000; bpprogressbar: FALSE
##   bpRNGseed:
##   bplogdir: NA
##   bpresultdir: NA
##   cluster type: FORK
```

For this small example we would like to assign less workers and activate the progress bar. Check *BiocParallel* for other possible backends and more options for `MulticoreParam`

```r
BiocParallel::register(BiocParallel::MulticoreParam(workers = 4, progressbar = TRUE))
```

If we check the current registered backend, we see that the number of workers has changed.

```r
BiocParallel::registered()[1]

## $MulticoreParam
## class: MulticoreParam
##   bpisup: FALSE; bpworkers: 4; bptasks: 0; bpjobname: BPJOB
##   bplog: FALSE; bpthreshold: INFO; bpstopOnError: TRUE
##   bptimeout: 2592000; bpprogressbar: TRUE
##   bpRNGseed:
##   bplogdir: NA
##   bpresultdir: NA
##   cluster type: FORK
```

## 1.3 Building a GenoGAM dataset

BAM files restricted to a region of chromosome XIV around the gene *YNL176C* are provided in the `inst/extdata` folder of the *GenoGAM* package. This folder also contains a flat file describing the experimental design.

We start by loading the experimental design from the tab-separated text file `experimentDesign.txt` into a data frame:

```r
folder <- system.file("extdata/Set1", package='GenoGAM')

expDesign <- read.delim(
  file.path(folder, "experimentDesign.txt")
)

expDesign

##         ID                                         file paired genotype
## 1     wt_1 H3K4ME3_Full_length_Set1_Rep_1_YNL176C.bam  FALSE        0
## 2     wt_2 H3K4ME3_Full_length_Set1_Rep_2_YNL176C.bam  FALSE        0
## 3 mutant_1  H3K4ME3_aa762-1080_Set1_Rep_1_YNL176C.bam  FALSE        1
## 4 mutant_2  H3K4ME3_aa762-1080_Set1_Rep_2_YNL176C.bam  FALSE        1
```

Each row of the experiment design corresponds to the alignment files in BAM format of one ChIP sample. In case of multiplexed sequencing, the BAM files must have been demultiplexed. The experiment design have named columns. Three column names have a fixed meaning for *GenoGAM* and must be provided: `ID`, `file`, and `paired`. The field `ID` stores a unique identifier for each alignment file. It is recommended to use short and easy to understand identifiers because they are subsequently used for labelling data and plots. The field `file` stores the BAM file name. The field `paired` values `TRUE` for paired-end sequencing data, and `FALSE` for single-end sequencing data. Further named columns can be added at wish without naming and data type constraints. Here the important one is the `genotype` column. Note that it is an indicator variable (i.e. valuing 0 or 1). It will allow us modeling the differential occupancy later on.

We will now count sequencing fragment centers per genomic position and sample and store these counts into a *GenoGAMDataSet*. *GenoGAM* reduces ChIP-Seq data to fragment center counts rather than full base coverage so that each fragment is counted only once. This reduces artificial correlation between adjacent nucleotides. For single-end libraries, the fragment center is estimated by shifting the read end position by a constant (Details in the help on the constructor function `GenoGAMDataSet()`).

```r
bpk <- 20
chunkSize <- 1000
overhangSize <- 15*bpk

## build the GenoGAMDataSet
ggd <- GenoGAMDataSet(
  expDesign, directory = folder,
  chunkSize = chunkSize, overhangSize = overhangSize,
  design = ~ s(x) + s(x, by = genotype)
)

## INFO [2016-08-11 22:50:47] Reading in data.
## INFO [2016-08-11 22:50:50] Check if tile settings match the data.
## INFO [2016-08-11 22:50:50] All checks passed.
## INFO [2016-08-11 22:50:50] DONE

## restricts the GenoGAM dataset to the positions of interest
## (this step is only required for running this small example)
ggd <- subset(ggd, seqnames == "chrXIV" & pos >= 305000 & pos <= 308000)
```

A *GenoGAMDataSet* stores this count data into a structure that index genomic positions over *tiles*, defined by `chunkSize` and `overhangSize`. A bit of background is required to understand these parameters. The smoothing in *GenoGAM* is based on splines, which are piecewise polynomials. The *knots* are the positions where the polynomials connect. In our experience, one knot every 20 to 50 bp is required for enough resolution of the smooth fits in typical applications. The fitting of generalized additive models involves steps demanding a number of operations proportional to the square of the number of knots, preventing fits along whole chromosomes. To make the fitting of GAMs genome-wide, *GenoGAM* performs fitting on small overlapping intervals (*tiles*), and join the fit at the midpoint of the overlap of consecutive tiles. The parameters `chunkSize` and `overhangSize` defines the tiles, where the chunk is the core part of a tile that does not overlap other tiles, and the overhangs are the two overlapping parts. Overhangs of about 10 times the knot spacing gives reasonable results.

The `design` parameter is explained in the next section.

Finally, the last line of code calls the function `subset()` to restrict the *GenoGAMDataset* to the positions of interest. This line is necessary for running this small example but would not be present in a standard genome-wide run of *GenoGAM*.

## 1.4 Modeling with smooth functions: the `design` parameters

*GenoGAM* models the logarithm of the rate of the count data as sums of smooth functions of the genomic position, denoted `x`. The `design` parameter is an R formula which allows encoding how the smooth functions depend on the experimental design. *GenoGAM*follows formula convention of the R package *mgcv*. A smooth function is denoted `s()`. For now, *GenoGAM* only supports smooth function that are cubic splines of the genomic position `x`. The `by` variable allows selecting to which samples the smooth contributes to (see also the documentation of `gam.models` in the *mgcv*). For now, *GenoGAM* only allows `by` variables to (value 0 or 1). Here by setting '`s(x, by=genotype)`' we encode the term "genotype × $f_{\text{mutant/wt}}(x)$" in Equation 1.

**Note:** As for other generalized additive models packages (*mgcv*, *gam*), *GenoGAM* use the natural logarithm as link function. This is different than other packages of the bioinformatics files such as *DESeq2* which works in base 2 logarithm.

## 1.5 Size factor estimation

Sequencing libraries typically vary in sequencing depth. Such variations is controlled for in *GenoGAM* by adding a sample-specific constant to the right term of Equation 1. The estimation of these constants is performed by the function computeSizeFactor() as follows:

```
ggd <- computeSizeFactors(ggd)

## INFO [2016-08-11 22:50:50] Computing size factors
## INFO [2016-08-11 22:50:50] DONE

sizeFactors(ggd)

##      wt_1      wt_2 mutant_1 mutant_2
##   -0.0198    0.2184  -0.5119    0.3237
```

**Note:** The size factors in *GenoGAM*are in the natural logarithm scale.

## 1.6 Model fitting

A *GenoGAM* model requires two further parameters to be fitted: the regularization parameter, $\lambda$, and the dispersion parameter $\theta$. The regularization parameters $\lambda$ controls the amount of smoothing. The larger $\lambda$ is, the smoother the smooth functions are. The dispersion parameter $\theta$ controls how much the observed counts deviate from their expected value modeled by Equation 1. The dispersion captures biological and technical variation which one typically sees across replicate samples, but also errors of the model. In *GenoGAM*, the dispersion is modeled by assuming the counts to follow a negative binomial distribution with mean $\mu = \mathrm{E}(y)$ whose logarithm is modeled by Equation 1 and with variance $v = \mu + \mu^2/\theta$.

If not provided, the parameters $\lambda$ and $\theta$ are obtained by cross-validation. This step is a bit time-consuming. For sake of going through this example quickly, we provide the values manually:

```
## fit model without parameters estimation
fit <- genogam(ggd,
               lambda = 40954.1,
               family = mgcv::nb(theta = 6.927986),
               bpknots = bpk
)

## INFO [2016-08-11 22:50:51] Check if tile settings match the data.
## INFO [2016-08-11 22:50:51] All checks passed.
## INFO [2016-08-11 22:50:51] Process data
## INFO [2016-08-11 22:50:51] Fitting model
##
```

```
  |
  |                                                                      |   0%
  |
  |==================                                                    |  25%
  |
  |===================================                                   |  50%
  |
  |====================================================                  |  75%
  |
  |======================================================================| 100%
##
## INFO [2016-08-11 22:51:00] DONE
```

```
fit
```

```
##
## Family: negative binomial
## Link function: log
##
## Formula:
## value ~ offset(offset) + s(x, bs = "ps", k = 80, m = 2) + s(x,
##     by = genotype, bs = "ps", k = 80, m = 2)
## <environment: 0x198a51d8>
##
## Experiment Design:
##          genotype
## wt_1            0
## wt_2            0
## mutant_1        1
## mutant_2        1
##
## Global Estimates:
##   Lambda: 40954
##   Theta: 6.93
##   Coefficient of Variation: 0.38
##
## Cross Validation: Not performed
##   K-folds: 10
##   Number of tiles: 4
##   Interval size: 20
##
## Tile settings:
##   chunk size: 1000
##   tile size: 1600
##   overhang size: 300
##   number of tiles: 4
```

**Remark on parameter estimation:** To estimate the parameters $\lambda$ and $\theta$ by cross-validation, call `genogam()` without setting their values. This will perform 10 fold cross-validation on each tile with initial parameter values and iterate until convergence, often for about 50 iterations. We recommend to do it for 20 to 40 different regions representative of your data (of 1.5kb each). This means that estimation of the parameters will require the equivalent of a *GenoGAM* fit with fixed $\lambda$ and $\theta$ on 30 Mb (1.5kb x10x40x50). For a genome like yeast (12Mb) the cross-validation thus can take more time than a genome-wide fit.

```
fit_CV <- genogam(ggd,bpknots = bpk)
```

**Remark on parallel computing:** *GenoGAM* run parallel computations on multicore architecture (using the *BiocParallel* package). Computing time reduces almost linearly with the number of cores of the machine.

## 1.7 Plotting results

Count data and fits for a region of interest can be extracted using the function `view()`. Following the *mgcv* and *gam* convention the names of the fit for the smooth function defined by the `by` variable follow the pattern `s(x):{by-variable}`. Here, the smooth function of interest $f_{\text{mutant/wt}}(x)$ is thus named `s(x):genotype`.

```
# extract count data into a data frame
df_data <- view(ggd)
head(df_data)

##   seqnames    pos strand wt_1 wt_2 mutant_1 mutant_2
## 1   chrXIV 305000      *    0    0        0        0
## 2   chrXIV 305001      *    0    0        0        0
## 3   chrXIV 305002      *    0    1        0        0
## 4   chrXIV 305003      *    0    0        0        0
## 5   chrXIV 305004      *    0    0        0        0
## 6   chrXIV 305005      *    0    1        0        0

# extract fit into a data frame
df_fit <- view(fit)
head(df_fit)

##   seqnames    pos strand  s(x) s(x):genotype se.s(x) se.s(x):genotype
## 1   chrXIV 305000      * -3.05        0.0833   0.309            0.402
## 2   chrXIV 305001      * -3.05        0.0858   0.307            0.400
## 3   chrXIV 305002      * -3.04        0.0883   0.305            0.398
## 4   chrXIV 305003      * -3.04        0.0907   0.304            0.396
## 5   chrXIV 305004      * -3.04        0.0932   0.302            0.394
## 6   chrXIV 305005      * -3.04        0.0956   0.300            0.392
```

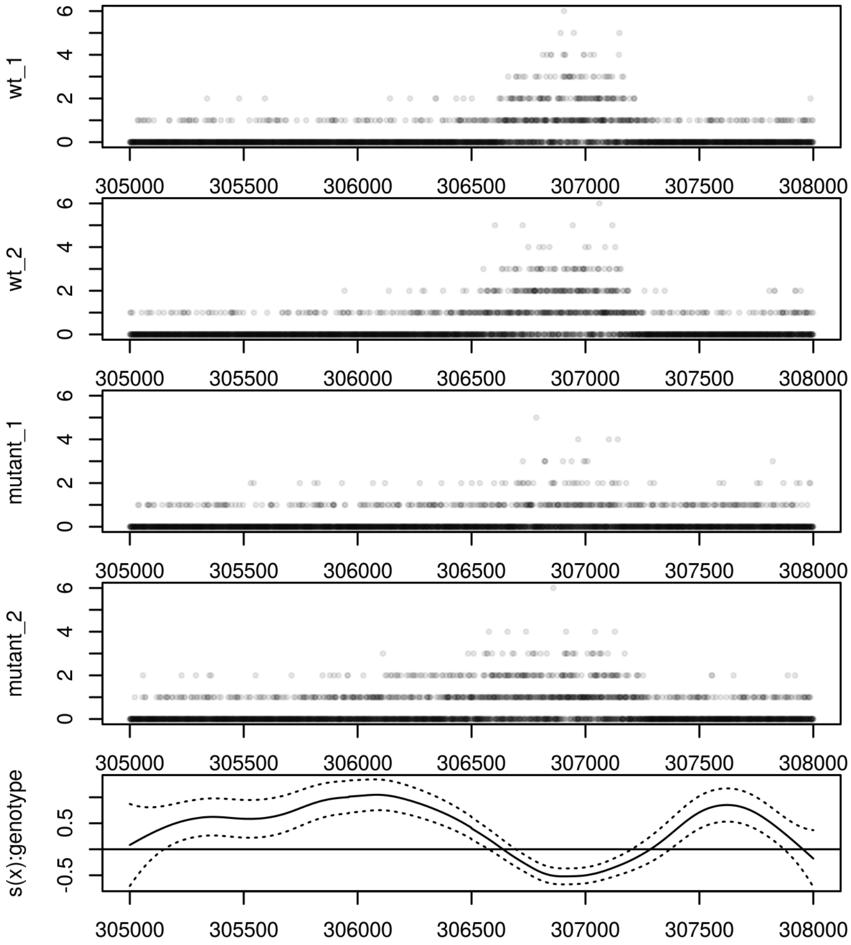
The code below then plots the counts and the fitted smooth log-ratio of mutant over wild type together with its 95% confidence band. In the count data, the peak of methylation in the two replicates of the wild type (first two panels) in the region 306,500-307,000 seems attenuated in the two replicates of the mutant (3rd and 4th panel). There are relatively more counts for the mutant in the region 305,500-306,500. The *GenoGAM* fit of the log-ratio (last panel, confidence band dotted) indicates that that these differences are significant. This redistribution of the methylation mark from the promoter (wild type) into the gene body (mutant) was reported by the authors of the study [1].

```
## plot function for the count data
dataplot <- function(df, col, ...){
  x <- df[["pos"]]
```

```r
  y <- df[[col]]
  plot(0, type='n',
       xlim=range(x),
       ylab=col,
       ...
  )
  points(x, y, pch=19, col="#00000015", cex=0.5)
}

# plot function for a fit with confidence band
fitplot <- function(df, smooth, ...){
  x <- df[["pos"]]
  y <- df[[smooth]]
  se.y <- df[[paste0("se.",smooth)]]
  plot(0, type='n',
       xlim=range(x),
       ylim=range(c(y  - 1.96*se.y, y  + 1.96*se.y)),
       ylab=smooth,
       ...
  )
  lines(x, y)
  lines(x, y+1.96*se.y, lty='dotted')
  lines(x, y-1.96*se.y, lty='dotted')
  abline(h=0)
}

## plot
par(mfrow=c(5,1))
par(mar=c(1,4,1,1))
for(id in expDesign$ID)
  dataplot(df_data, id, xlab="", main="", ylim=c(0,6))
par(mar=c(2,4,1,1))
fitplot(df_fit, 's(x):genotype', xlab="", main="")
```



## 1.8 Statistical testing

We test for each smooth and at each position $x$ the null hypothesis that it values 0 by a call to `computeSignificance()`. False discovery rate can be computed using the Benjamini-Hochberg procedure with the R function `p.adjust()`:

```
fit <- computeSignificance(fit)
df_fit <- view(fit)
df_fit[["fdr.s(x):genotype"]] <- p.adjust(df_fit[["pvalue.s(x):genotype"]], method="BH")
head(df_fit)
```

```
##   seqnames    pos strand  s(x) s(x):genotype se.s(x) se.s(x):genotype
## 1   chrXIV 305000      * -3.05        0.0833   0.309            0.402
## 2   chrXIV 305001      * -3.05        0.0858   0.307            0.400
## 3   chrXIV 305002      * -3.04        0.0883   0.305            0.398
## 4   chrXIV 305003      * -3.04        0.0907   0.304            0.396
## 5   chrXIV 305004      * -3.04        0.0932   0.302            0.394
## 6   chrXIV 305005      * -3.04        0.0956   0.300            0.392
##   pvalue.s(x) pvalue.s(x):genotype fdr.s(x):genotype
## 1      0.0665                0.836             0.851
## 2      0.0656                0.830             0.846
## 3      0.0647                 0.825              0.841
## 4      0.0637                 0.819              0.836
## 5      0.0628                 0.813              0.831
## 6      0.0620                 0.807              0.826
```

## 2 Other functionalities

Other functionalities demonstrated in the manuscript (peak calling and testing, methylation data, see BioRxiv: http://dx.doi.org/10.1101/047464) will be gradually integrated into the package. Interested users should check the developer version of *GenoGAM* for updates.

## 3 Acknowledgments

We thank Alexander Engelhardt, Hervé Pagès, and Martin Morgan for input in the development of *GenoGAM*.

## 4 Session Info

- R version 3.3.1 (2016-06-21), `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: Biobase 2.32.0, BiocGenerics 0.18.0, Biostrings 2.40.2, GenoGAM 1.0.3, GenomeInfoDb 1.8.3, GenomicRanges 1.24.2, IRanges 2.6.1, Rsamtools 1.24.0, S4Vectors 0.10.2, SummarizedExperiment 1.2.3, XVector 0.12.1, knitr 1.13
- Loaded via a namespace (and not attached): AnnotationDbi 1.34.4, BiocParallel 1.6.5, BiocStyle 2.0.3, DBI 0.4-1, DESeq2 1.12.4, Formula 1.2-1, GenomicAlignments 1.8.4, Hmisc 3.17-4, Matrix 1.2-6, RColorBrewer 1.1-2, RSQLite 1.0.0, Rcpp 0.12.6, ShortRead 1.30.0, XML 3.98-1.4, acepack 1.3-3.3, annotate 1.50.0, bitops 1.0-6, chipseq 1.22.0, chron 2.3-47, cluster 2.0.4, codetools 0.2-14, colorspace 1.2-6, data.table 1.9.6, digest 0.6.10, evaluate 0.9, foreign 0.8-66, formatR 1.4, futile.logger 1.4.3, futile.options 1.0.0, genefilter 1.54.2, geneplotter 1.50.0, ggplot2 2.1.0, grid 3.3.1, gridExtra 2.2.1, gtable 0.2.0, highr 0.6, hwriter 1.3.2, lambda.r 1.1.9, lattice 0.20-33, latticeExtra 0.6-28, locfit 1.5-9.1, magrittr 1.5, mgcv 1.8-13, munsell 0.4.3, nlme 3.1-128, nnet 7.3-12, plyr 1.8.4, reshape2 1.4.1, rpart 4.1-10, scales 0.4.0, splines 3.3.1, stringi 1.1.1, stringr 1.0.0, survival 2.39-5, tools 3.3.1, xtable 1.8-2, zlibbioc 1.18.0

## References

[1] J. Thornton, G.H.. Westfield, Y. Takahashi, M. Cook, X. Gao, Woodfin A.R., Lee J., A.M. Morgan, J. Jackson, E.R. Smith, J. Couture, G. Skiniotis, and A. Shilatifard. Context dependency of Set1/COMPASS-mediated histone H3 Lys4 trimethylation. *Genes & Development*, 28(2):115–120, 2014. `doi:10.1101/gad.232215.113`.