GenoGAM: Genome-wide generalized additive models

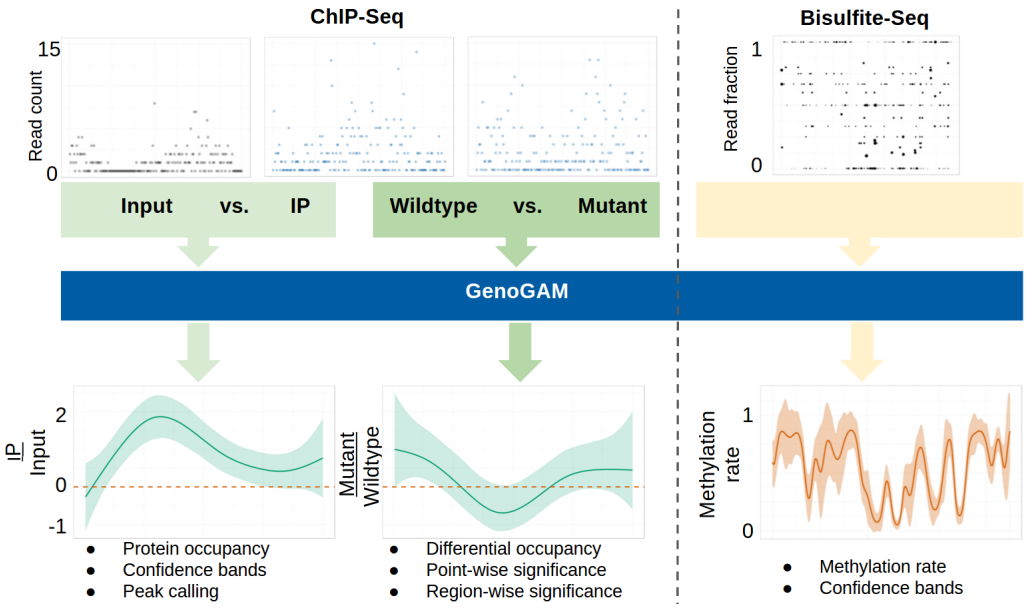
Georg Stricker[1], Julien Gagneur[1]

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

October 20, 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 as well as peak calling with position-wise confidence bands. 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: 1.2.1

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

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

Contents

1 Standard ChIP-Seq analysis

Additional to the basic smoothing and point-wise significance computation this version of *GenoGAM* now also supports differential analysis and peak calling with position-wise confidence intervals on 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[**?**], 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 can be registered at anytime as *GenoGAM* will just call the current one.

IMPORTANT: According to this and this posts on the Bioconductor support page and R-devel mailing list, the most important core feature of the *multicore* backend, shared memory, is compromised by Rs own garbage collecto,r resulting in a replication of the entire workspace across all cores. Given that the amount of data in memory is big it might crash the entire system. **We highly advice to register the SnowParam backend** to avoid this **if working on a multicore machine**. This way the overhead is a little bigger, but only necessary data is copied to the workers keeping memory consumption relatively low. We never experienced a higher load than 4GB per core, usually it was around 2GB on human genome.

```
library(GenoGAM)

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

```
## $MulticoreParam
## class: MulticoreParam
##   bpisup: FALSE; bpnworkers: 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.. Check *BiocParallel* for other possible backends and more options for `SnowParam`

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

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

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

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

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:

```
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 or call peaks 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()`). Additionally we filter the data for enriched regions only. A threshold is estimated from the data as the median + 6*MAD (Median Absolute Deviation) or can be supplied through the *threshold* argument. Especially on large genomes such as human this can boost the computation time significantly.

```
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 [2016-10-20 22:20:03] Reading in data.
## INFO [2016-10-20 22:20:06] Check if tile settings match the data.
## INFO [2016-10-20 22:20:06] All checks passed.
## INFO [2016-10-20 22:20:06] DONE
```

```
ggd
```

```
## class: GenoGAMDataSet
## dimension: 784333 4
## assays(1):
## position variables(0):
## sample variables(1): genotype
## samples(4): wt_1 wt_2 mutant_1 mutant_2
## tiles size: 1.6kbp
## number of tiles: 785
## chromosomes: chrXIV
```

```
## data filtered for regions with zero counts
filtered_ggd <- filterData(ggd)
```

```
## INFO [2016-10-20 22:20:06] Filtering dataset for enriched regions
## INFO [2016-10-20 22:20:08] Threshold estimated at 0.859781495869741
## INFO [2016-10-20 22:20:08] DONE. A total of 781,030 positions were dropped, 3,303 are left.
```

```
filtered_ggd
```

```
## class: GenoGAMDataSet
## dimension: 3303 4
## assays(1):
## position variables(0):
## sample variables(1): genotype
## samples(4): wt_1 wt_2 mutant_1 mutant_2
## tiles size: 1.6kbp
## number of tiles: 4
## chromosomes: chrXIV
```

```
## alternatively we can restricts the GenoGAM dataset to the positions of
## interest as we know them by design of this example
ggd <- subset(ggd, seqnames == "chrXIV" & pos >= 305000 & pos <= 308000)

## They are almost the same as found by the filter
range(getIndex(filtered_ggd))
```

```
## GRanges object with 1 range and 0 metadata columns:
##       seqnames           ranges strand
##          <Rle>        <IRanges>  <Rle>
##   [1]   chrXIV [304849, 308151]      *
##   -------
##   seqinfo: 1 sequence from an unspecified genome
```

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 (Figure 1), 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*.

Note: genogam have a settings-class available but not open to the user, as it is not yet fully developed. The `GenoGAMDataSet` function however can take an argument *settings* that would take such an object as input. The argument is there to allow for some specific workarounds if necessary. The most common one would be to load only a subset of the data from a BAM file for example defined by *chromosome* names or supplying a `ScanBamParam` object from the *GenomicAlignments* package. The examples are not run.

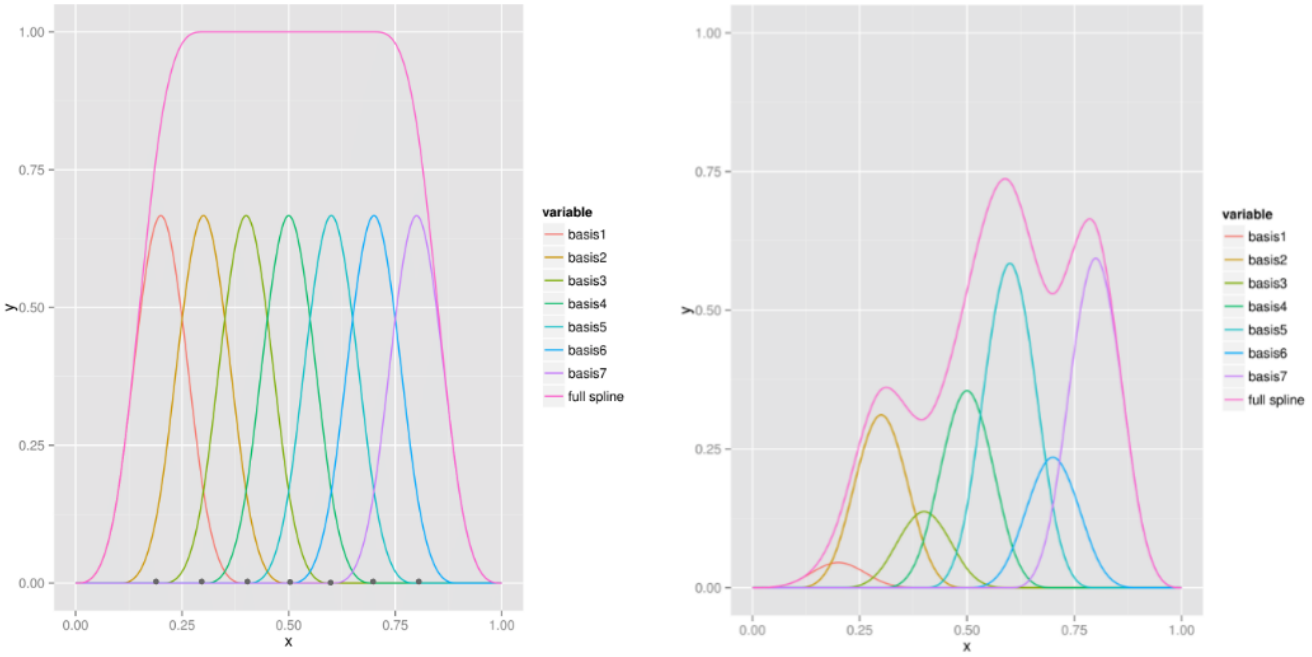


Figure 1: **Left:** An example spline without coefficients. Displayed are seven cubic B-spline basis functions which together make up the complete spline (pink). The knots are depicted as dark-grey dots at the bottom-center of each basis function. **Right:** The same spline as on the left side, but with basis functions multiplied by their respective coefficient.

```
## specify specific chromosomes
settings <- GenoGAM:::GenoGAMSettings(chromosomeList = c('chrI', 'chrII'))

## specify parameters through ScanBamParam
gr <- GRanges("chrI", IRanges(1,100000))
params <- ScanBamParam(which = gr)
settings <- GenoGAM:::GenoGAMSettings(bamParams = params)
```

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 be of value 0 or 1 (hence binary dummy encoding). Here by setting `'s(x, by=genotype)'` we encode the term "genotype $\times f_{\text{mutant/wt}}(x)$" in Equation **??**.

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 **??**. The estimation of these constants is performed by the function computeSizeFactor() as follows:

```
ggd <- computeSizeFactors(ggd)
```

```
## INFO [2016-10-20 22:20:09] Computing size factors
## INFO [2016-10-20 22:20:09] DONE
```

```
sizeFactors(ggd)
```

```
##     wt_1     wt_2 mutant_1 mutant_2
##   0.0429   0.2595  -0.5295   0.2901
```

Note: The size factors in *GenoGAM*are in the natural logarithm scale. Also factor groups are identified automatically from the design. Given more complex design, this might fail. Or, maybe different factor groups are desired. In this case the groups can be provided separatelly through the *factorGroups* argument.

The additional function `qualityCheck` will perform a simple quality check of the data and plot the results into the folder *qc*, which is created in the working directory if not present. So far it's only applicable to the *GenoGAMDataSet* object and creates two plots: One showing the distribution of counts over all tiles and the second showing multiple scatterplots of the counts from different samples against each other before and after size factor normalization

```
## the function is not run as it creates new plots each time,
## which are not very meaningful for such a small example
qualityCheck(ggd)
```

1.6 Model fitting

A *GenoGAM* model requires two further parameters to be fitted: the regularization parameter, λ, and the dispersion parameter θ. The regularization parameters λ controls the amount of smoothing. The larger λ is, the smoother the smooth functions are. The dispersion parameter θ controls how much the observed counts deviate from their expected value modeled by Equation **??**. 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 **??** and with variance $v = \mu + \mu^2/\theta$.

If not provided, the parameters λ and θ 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 [2016-10-20 22:20:09] Check if tile settings match the data.
## INFO [2016-10-20 22:20:09] All checks passed.
## INFO [2016-10-20 22:20:09] Process data
## INFO [2016-10-20 22:20:09] Fitting model
##
  |
  |                                                                      |   0%
  |
  |=======================                                               |  33%
  |
  |===============================================                       |  67%
  |
  |======================================================================| 100%
##
## INFO [2016-10-20 22:20:32] 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: 0x7fdf3089fb48>
##
## 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: 3
##   Interval size: 20
##
## Tile settings:
##   chunk size: 1000
##   tile size: 1600
##   overhang size: 300
```

```
##   number of tiles: 3
```

Remark on parameter estimation: To estimate the parameters λ and θ 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 λ and θ 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      * -3.10         0.163   0.309            0.402
## 2   chrXIV 305001      * -3.10         0.165   0.307            0.400
## 3   chrXIV 305002      * -3.10         0.168   0.305            0.398
## 4   chrXIV 305003      * -3.09         0.170   0.304            0.396
## 5   chrXIV 305004      * -3.09         0.173   0.302            0.394
## 6   chrXIV 305005      * -3.09         0.175   0.300            0.392
```

If only a subset of the data supposed to be viewed it is best to use a `GRanges` object and supply it to the emphranges argument.

```
gr <- GRanges("chrXIV", IRanges(306000, 308000))
head(view(ggd, ranges = gr))
```

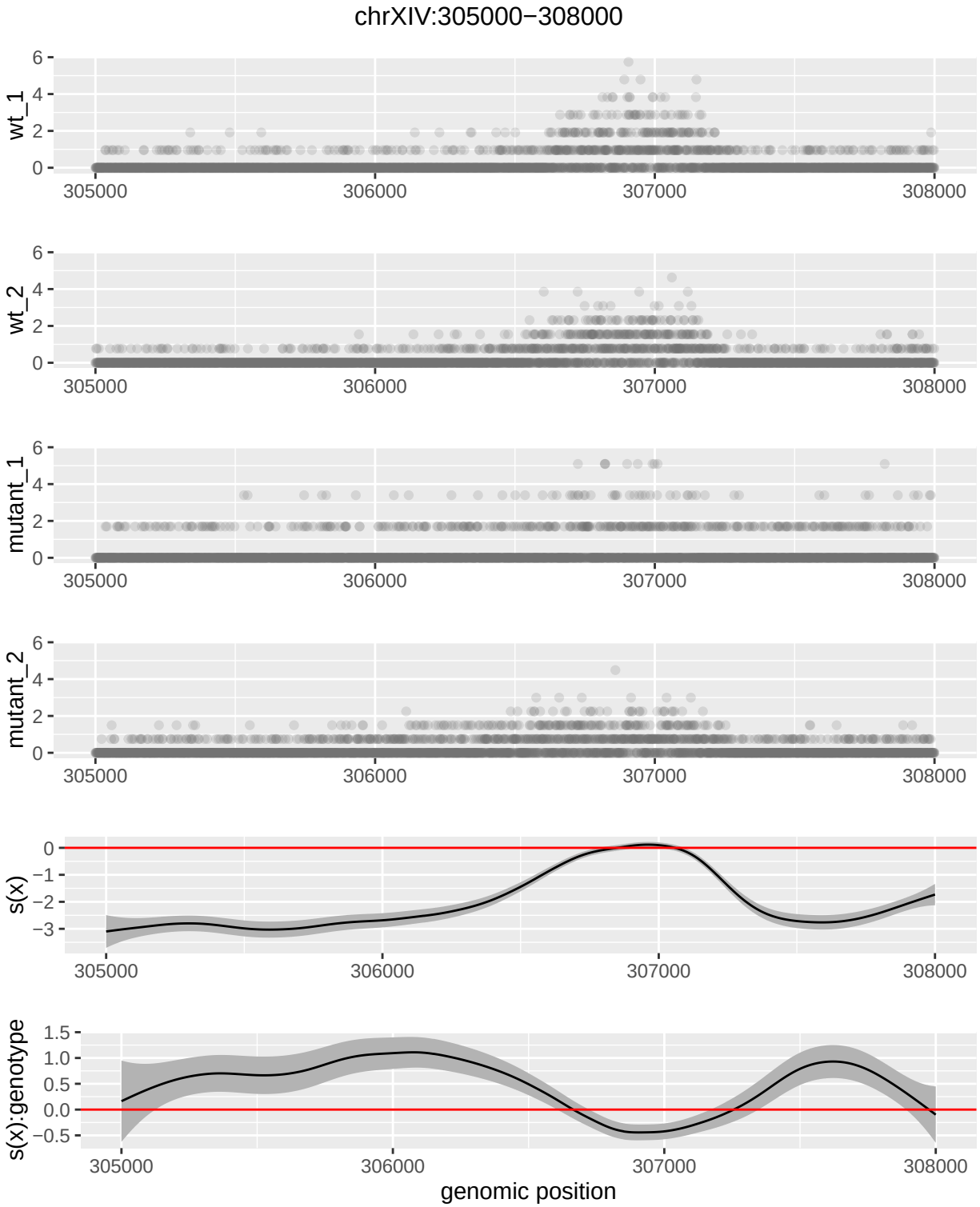
```
##   seqnames    pos strand wt_1 wt_2 mutant_1 mutant_2 assay
```

```
##   chrXIV 306000      *    0    0        0        0    V1
##   chrXIV 306001      *    0    0        0        0    V1
##   chrXIV 306002      *    1    0        0        0    V1
##   chrXIV 306003      *    0    0        0        0    V1
##   chrXIV 306004      *    0    0        0        0    V1
##   chrXIV 306005      *    0    1        0        0    V1
```

The GenoGAM specific plot function will make use of the ggplot2 framework if available or fall back to base R otherwise. Note, that without any restrictions on positions or chromosomes it will attempt to plot all the data and throw an error if it is too big.

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 [**?**]. Additionally a smooth for the input is shown (second to last). Both smooths added together give the log-fit of the mutant data (not shown).

```
plot(fit, ggd, scale = FALSE)
```


chrXIV:305000−308000
wt_1
wt_2
mutant_1
mutant_2
s(x)
s(x):genotype
305000
306000
307000
308000
genomic position

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()`. This gives us pointwise p-values.

```
fit <- computeSignificance(fit)
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      * -3.10         0.163   0.309            0.402
## 2   chrXIV 305001      * -3.10         0.165   0.307            0.400
## 3   chrXIV 305002      * -3.10         0.168   0.305            0.398
## 4   chrXIV 305003      * -3.09         0.170   0.304            0.396
## 5   chrXIV 305004      * -3.09         0.173   0.302            0.394
## 6   chrXIV 305005      * -3.09         0.175   0.300            0.392
##   pvalue.s(x) pvalue.s(x):genotype
## 1      0.0617                0.686
## 2      0.0608                0.680
## 3      0.0599                0.674
## 4      0.0590                0.668
## 5      0.0582                0.662
## 6      0.0573                0.655
```

1.9 Differential binding

If region-wise significance is needed, as in the case of differential binding, then we call `computeRegionSignificance()`. This returns the provided `GRanges` regions object updated by a p-value and FRD column.

```
gr
```

```
## GRanges object with 1 range and 0 metadata columns:
##       seqnames           ranges strand
##          <Rle>        <IRanges>  <Rle>
##   [1]   chrXIV [306000, 308000]      *
##   -------
##   seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

```
gr <- computeRegionSignificance(fit, regions = gr, what = 'genotype')
```

```
## INFO [2016-10-20 22:20:35] Estimating region p-values and FDR
## INFO [2016-10-20 22:20:35] Done
```

```
gr
```

```
## GRanges object with 1 range and 2 metadata columns:
##       seqnames           ranges strand |             pvalue               FDR
##          <Rle>        <IRanges>  <Rle> |          <numeric>         <numeric>
##   [1]   chrXIV [306000, 308000]      * | 3.325468426486e-10 3.325468426486e-10
##   -------
##   seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

1.10 Peak calling

The computed fit allows for easy peak calling of narrow and broad peaks via the `callPeaks()`. The resulting `data.table` provides a similar structure as the ENCODE narrowPeak and broadPeak format. Note, the *score* is given as the negative natural logarithm of the p-values. Also peak borders are different from the usually computed peak borders of other methods. The borders are in fact a 95-% confidence interval around the peak position. As this is not peak calling data, we use a high threshold do demonstrate the functionality.

```
peaks <- callPeaks(fit, smooth = "genotype", threshold = 1)
```

```
## INFO [2016-10-20 22:20:35] Calling narrow peaks
## INFO [2016-10-20 22:20:35] DONE
```

```
peaks
```

```
##    seqnames position summit  zscore score   fdr  start    end
## 1:   chrXIV   305364   2.02 -0.2449 0.516 0.667 305080 305648
## 2:   chrXIV   306077   3.03  0.3716 1.035 1.000 305835 306319
## 3:   chrXIV   307622   2.54  0.0999 0.776 1.000 307492 307752
```

```
peaks <- callPeaks(fit, smooth = "genotype", threshold = 1,
                   peakType = "broad", cutoff = 0.75)
```

```
## INFO [2016-10-20 22:20:35] Calling broad peaks
## INFO [2016-10-20 22:20:36] DONE
```

```
peaks
```

```
##    seqnames  start    end width strand score meanSignal  fdr
## 1:   chrXIV 305102 306523  1422      * 0.288       2.30 0.75
## 2:   chrXIV 307388 307863   476      * 0.289       2.15 0.75
```

The function `writeToBEDFile` provides an easy way to write the peaks data.table to a *narrowPeaks* or *broadPeaks* file. The suffix will be determined automatically

```
writeToBEDFile(peaks, file = 'myPeaks')
```

2 Computation statistics

At the moment the computation time for genome-wide analysis ranges from a couple of hours on yeast to a 2-3 days on human. The memory usage never exceeded 4GByte per core. Usually it was around 2-3 GByte per core. We are also currently implementing an optimized model fitting procedure, that reduces computation time significantly. Unfortunately we were not able to include it into the package for this Bioconductor release, but are confident to be able to do it into the next release. Interested users should therefore 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-apple-darwin13.4.0`
- Locale: `C/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8`
- Base packages: base, datasets, grDevices, graphics, methods, parallel, stats, stats4, utils
- Other packages: Biobase 2.34.0, BiocGenerics 0.20.0, Biostrings 2.42.0, GenoGAM 1.2.1, GenomeInfoDb 1.10.0, GenomicRanges 1.26.1, IRanges 2.8.0, Rsamtools 1.26.0, S4Vectors 0.12.0, SummarizedExperiment 1.4.0, XVector 0.14.0, knitr 1.14
- Loaded via a namespace (and not attached): AnnotationDbi 1.36.0, BiocParallel 1.8.0, BiocStyle 2.2.0, DBI 0.5-1, DESeq2 1.14.0, Formula 1.2-1, GenomicAlignments 1.10.0, Hmisc 3.17-4, Matrix 1.2-7.1, RColorBrewer 1.1-2, RCurl 1.95-4.8, RSQLite 1.0.0, Rcpp 0.12.7, ShortRead 1.32.0, XML 3.98-1.4, acepack 1.3-3.3, annotate 1.52.0, bitops 1.0-6, chipseq 1.24.0, chron 2.3-47, cluster 2.0.5, codetools 0.2-15, colorspace 1.2-7, data.table 1.9.6, digest 0.6.10, evaluate 0.10, foreign 0.8-67, formatR 1.4, futile.logger 1.4.3, futile.options 1.0.0, genefilter 1.56.0, geneplotter 1.52.0, ggplot2 2.1.0, grid 3.3.1, gridExtra 2.2.1, gtable 0.2.0, highr 0.6, hwriter 1.3.2, labeling 0.3, lambda.r 1.1.9, lattice 0.20-34, latticeExtra 0.6-28, locfit 1.5-9.1, magrittr 1.5, mgcv 1.8-15, 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, snow 0.4-2, splines 3.3.1, stringi 1.1.2, stringr 1.1.0, survival 2.39-5, tools 3.3.1, xtable 1.8-2, zlibbioc 1.20.0